Link for ModelSim®

For Use with MATLAB® and Simulink®

■ Computation

■ Visualization

■ Programming

■ Simulation

User's Guide

Version 1

The MathWorks

How to Contact The MathWorks:

	`www.mathworks.com`	Web
	`comp.soft-sys.matlab`	Newsgroup
@	`support@mathworks.com`	Technical Support
	`suggest@mathworks.com`	Product enhancement suggestions
	`bugs@mathworks.com`	Bug reports
	`doc@mathworks.com`	Documentation error reports
	`service@mathworks.com`	Order status, license renewals, passcodes
	`info@mathworks.com`	Sales, pricing, and general information
☎	508-647-7000	Phone
	508-647-7001	Fax
✉	The MathWorks, Inc. 3 Apple Hill Drive Natick, MA 01760-2098	Mail

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

Trademarks

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

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

Patents

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

Revision History

Contents

Getting Started

1

MATLAB and ModelSim Tutorial

2

Simulink and ModelSim Tutorial

3

MATLAB and ModelSim Manchester Receiver Tutorial

4

Coding a Link for ModelSim MATLAB Application

5

Starting and Controlling MATLAB Test Bench Sessions

6

Modeling and Verifying a VHDL Design with Simulink

7

MATLAB Functions — Alphabetical List

8

ModelSim Commands — Alphabetical List

9

Simulink Blocks — Alphabetical List

10

Examples

A

Index

1

Getting Started

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 the Mentor Graphics hardware description 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.
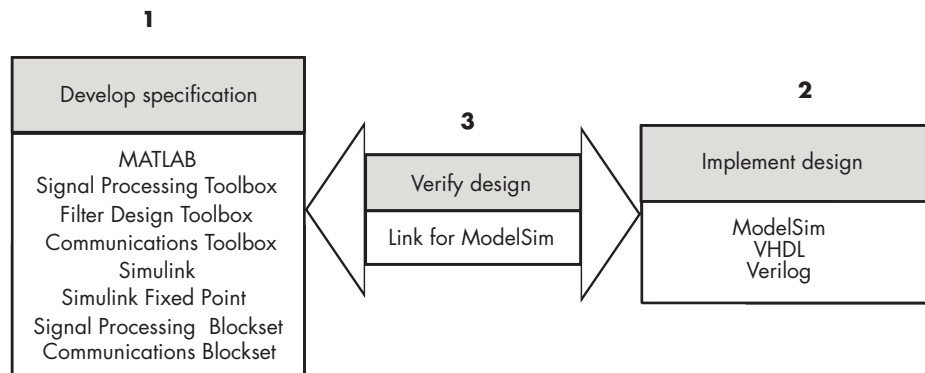
As the figure shows, the Link for ModelSim connects tools that traditionally have been used discretely to accomplish specific steps in the design process. By connecting the tools, the Link for ModelSim simplifies verification by allowing you to cosimulate the implementation and original specification 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. The 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 Simulink Fixed Point 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   Signal Processing Blockset

Filter Design Toolbox   Communications Blockset

Communications Toolbox

Key Features

Key features of the 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

- MATLAB test bench functions that support verification of the performance of the VHDL model, or of components within the model

- MATLAB component functions that simulate the behavior of entities in the VHDL model

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.
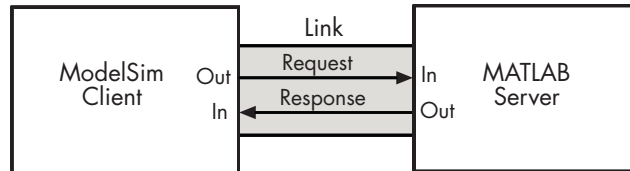
In this scenario, a MATLAB server function waits for service requests that it receives from a ModelSim simulator session. After receiving a request, the server establishes a communication link, and invokes a specified MATLAB function wrapper that computes data for, verifies, or visualizes the HDL model 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.
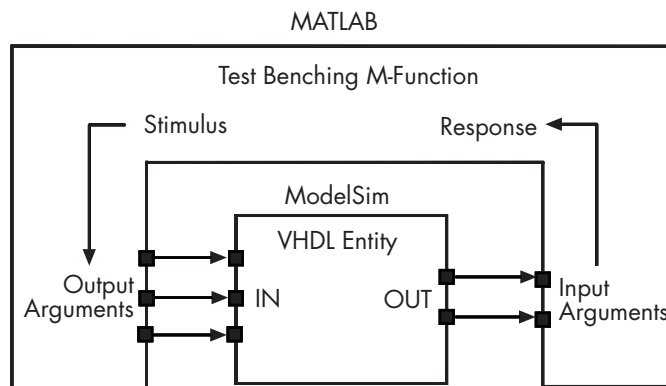
The MATLAB server can service multiple simultaneous ModelSim sessions and HDL entities. However, you should adhere to recommended guidelines to ensure the server can track the I/O associated with each entity and session. The following figure shows a multiple-client scenario connecting to the server at TCP/IP socket port 4449.
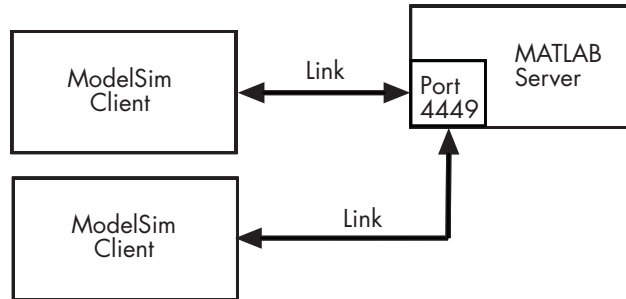
Simulink and ModelSim Links

When linked with Simulink, ModelSim functions as the server, as shown in the following figure.
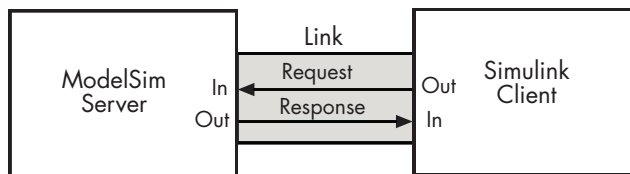
In this case, ModelSim responds to simulation requests it receives from cosimulation blocks in a Simulink model. You initiate a cosimulation session from Simulink. Once a session is started, you can use Simulink and ModelSim to monitor simulation progress and results. For example, you might add signals 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.
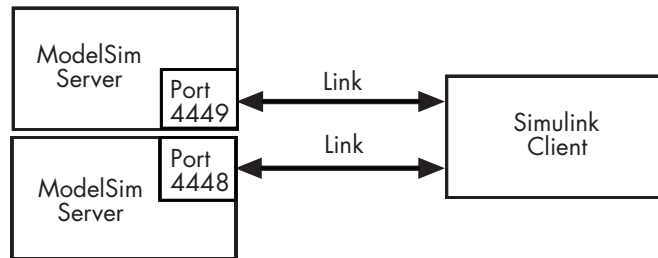
Modes of Communication

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, the 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

- Block input and output ports that correspond to signals (including internal signals) of a VHDL model. You can specify sample times and fixed-point data types for individual block output ports if desired.
- Type of communication and communication settings used for exchanging data between the simulation tools.
- Rising-edge or falling-edge clocks to apply to your model. The period of each clock is individually specifiable.
- Tcl commands 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, the 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 can 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 1-13
- "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 1-18
- "Checking Product Requirements" on page 1-20
- "Installing Related Application Software" on page 1-21
- "Installing the Link for ModelSim" on page 1-21
- "Setting Up ModelSim for Use with the Link for ModelSim" on page 1-22

The following figure summarizes the installation and setup process in a flow diagram. Topics that follow explain the steps in more detail.
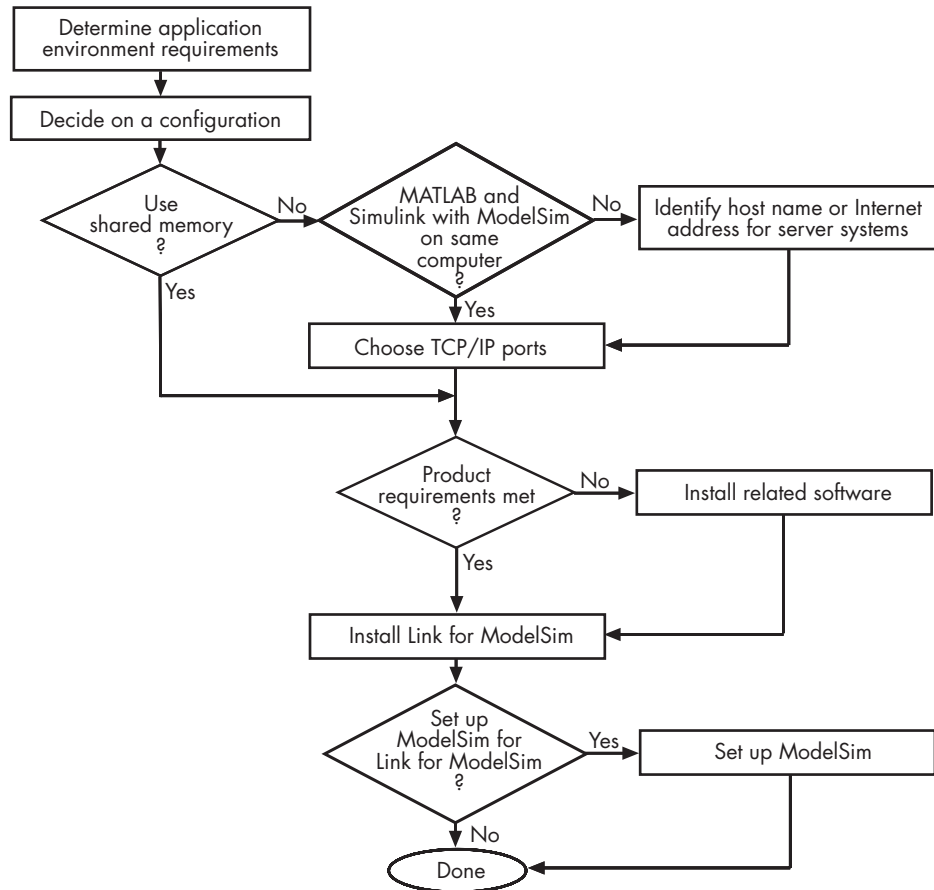
What Are Your Environment Requirements?

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...
Configurations	
☐ Will your application use ModelSim with the MATLAB, Simulink, or both MATLAB and Simulink?	"Deciding on a Configuration" on page 1-15
☐ Will your application use multiple communication links?	"Deciding on a Configuration" on page 1-15
☐ How many instances of the MATLAB server are required?	"Deciding on a Configuration" on page 1-15
☐ 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
☐ How many MATLAB functions do you need to write to model your VHDL implementation?	"Deciding on a Configuration" on page 1-15
☐ 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
☐ To how many ModelSim sessions will your Simulink model connect?	"Deciding on a Configuration" on page 1-15
Mode of Communication	
☐ 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
☐ Does your application require only one communication link (channel) on a single computing system?	"Modes of Communication" on page 1-8
☐ Is configuration flexibility a high priority for your application? Does the application have growth potential?	"Modes of Communication" on page 1-8
☐ 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 1-18

Environment Requirements Checklist (Continued)

Requirement	For More Information, See...
Network Configurations	
☐ 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
☐ 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
☐ 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 1-18
Related Software	
☐ Is ModelSim installed on all systems as needed for your application?	"Installing Related Application Software" on page 1-21
☐ Is MATLAB installed on all systems as needed for your application?	"Installing Related Application Software" on page 1-21
☐ 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 1-21
☐ Will you be using the Simulink component of the Link for ModelSim? If so, are Simulink and Simulink Fixed Point installed on all systems as needed for your application? Are the required blocksets installed?	"Installing Related Application Software" on page 1-21
ModelSim Setup	
☐ 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 1-22

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 1-18.

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.

- On the Windows platform, do not choose a filtered TCP/IP port. The Windows TCP/IP port filtering mechanism allows disabling access to selected ports for security purposes. TCP/IP port filtering on either the client or server side can cause the Link for ModelSim interface to fail to make a connection.

  In such cases the error messages displayed by the Link for ModelSim indicate the lack of a connection, but do not explicitly indicate the cause. A typical scenario caused by port filtering would be a failure to start a simulation in ModelSim, with the following warning displayed in ModelSim if the simulation is restarted:

  ```
  #MLWarn - MATLAB server not available (yet),
  The entity 'entityname' will not be active
  ```

  In MATLAB, checking the server status at this point indicates that the server is running with no connections:

  ```
  x=hdldaemon('status')
  HDLDaemon server is running with 0 connections
  x=
      4449
  ```

  If you suspect that your chosen socket port is filtered, you can check it as follows:

  a From the Windows **Start** menu, select **Settings—>Network Connections**.

  b Select **Local Area Connection** from the **Network and Dialup Connections** window.

    c From the **Local Area Connection** dialog, select **Properties—> Internet Protocol (TCP/IP)—>Properties—>Advanced—> Options—>TCP/IP filtering—>Properties**.

    d If your port is listed in the **TCP/IP filtering>Properties** dialog, you should select an unfiltered port. The easiest way to do this is to specify 0 for the socket port number to let the Link for ModelSim choose an available port number for you.

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

The Link for ModelSim requires the following:

Platform	Linux
	Solaris
	Windows 2000
	Windows XP
Application software	ModelSim SE/PE, a Mentor Graphics product. The current release of the Link for ModelSim has been tested with ModelSim versions 5.8b, 5.8c, 5.8d, and 6.0. The Link for ModelSim requires the ModelSim VHDL interface, and therefore requires that your ModelSim installation be licensed for VHDL. If your ModelSim installation is licensed for Verilog only, you should contact Mentor Graphics to obtain an additional VHDL license.
	MATLAB

Application software required for cosimulation	Simulink
	Simulink Fixed Point
Optional application software	Communications Blockset
	Signal Processing Blockset
	Filter Design Toolbox
	Signal Processing Toolbox
Platform-specific software	On the Linux platform, the `gcc` c++ libraries (3.2 or later) are required by the Link for ModelSim. You should install a recent version of the `gcc` c++ library on your computer. To determine which libraries are installed on your computer, type the command:

```
gcc -v
```

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 the 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 configure ModelSim for use with the Link for ModelSim as part of the installation process, use the MATLAB function `configuremodelsim`. The `configuremodelsim` function registers new MATLAB and Simulink related Tcl commands for the ModelSim simulator by creating the file `...\tcl\ModelSimTclFunctionsForMATLAB.tcl` within in the ModelSim installation directory. Note that `configuremodelsim` does *not* select the configured ModelSim executable as the default simulator to be used by the `vsim` command.

You can use the `configuremodelsim` function either in interactive mode or command-line mode. The interactive mode displays messages and prompts you for input. The command-line mode is available for use in scripts. The following sections describe how to use `configuremodelsim` in each of these modes.

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:

```
configuremodelsim
```

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

```
configuremodelsim('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 configuremodelsim 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 configuremodelsim locate and display a list of installed versions, enter y or press the **Enter** key. 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, 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, configuremodelsim overwrites it and then displays the following message:

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

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

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

Running the Setup Program in Command-Line Mode
To run the ModelSim setup program in command-line mode, call the configuremodelsim 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 configuremodelsim for more information.

Removing Link for ModelSim Configuration Information from ModelSim
To remove Link for ModelSim configuration information from ModelSim, issue the configuremodelsim function with the property name/property value pair 'action','uninstall'.

```
configuremodelsim ('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 1-25
- "Online Help" on page 1-26
- "Demos and Tutorials" on page 1-26

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 configuremodelsim`.

- 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 1-28

- "Running the Simulink and ModelSim Manchester Receiver Demo" on page 1-33

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

The 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 the 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 Start MATLAB and make it your active window.

2 Set up and change to a writable working directory that is outside the context of your MATLAB installation directory.

3 Enter the function name modsimrand at the prompt in the MATLAB Command Window.

```
modsimrand
```

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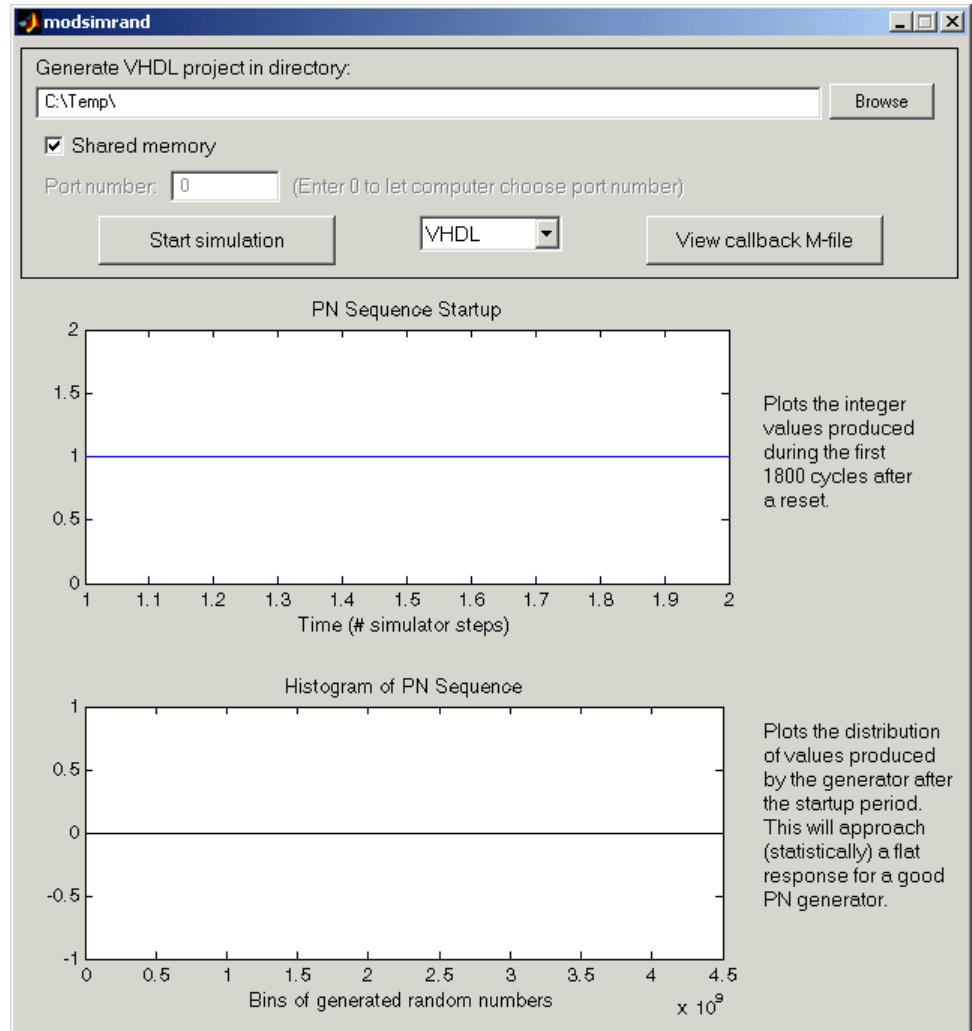
4 Specify the location into which the demo is to place the ModelSim project files that it generates. This location must be writable. You can type a path in the **Generate VHDL project in directory** text field or you can click **Browse** to find an appropriate directory.

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 1-18.

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 communication 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.
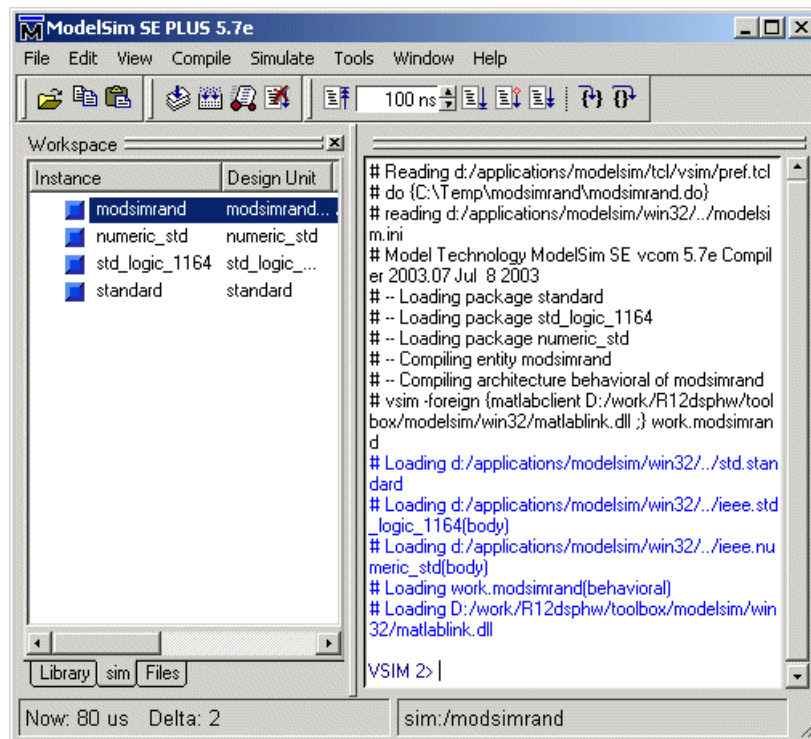
Also note the changes that occur in the **modsimrand** window plots.
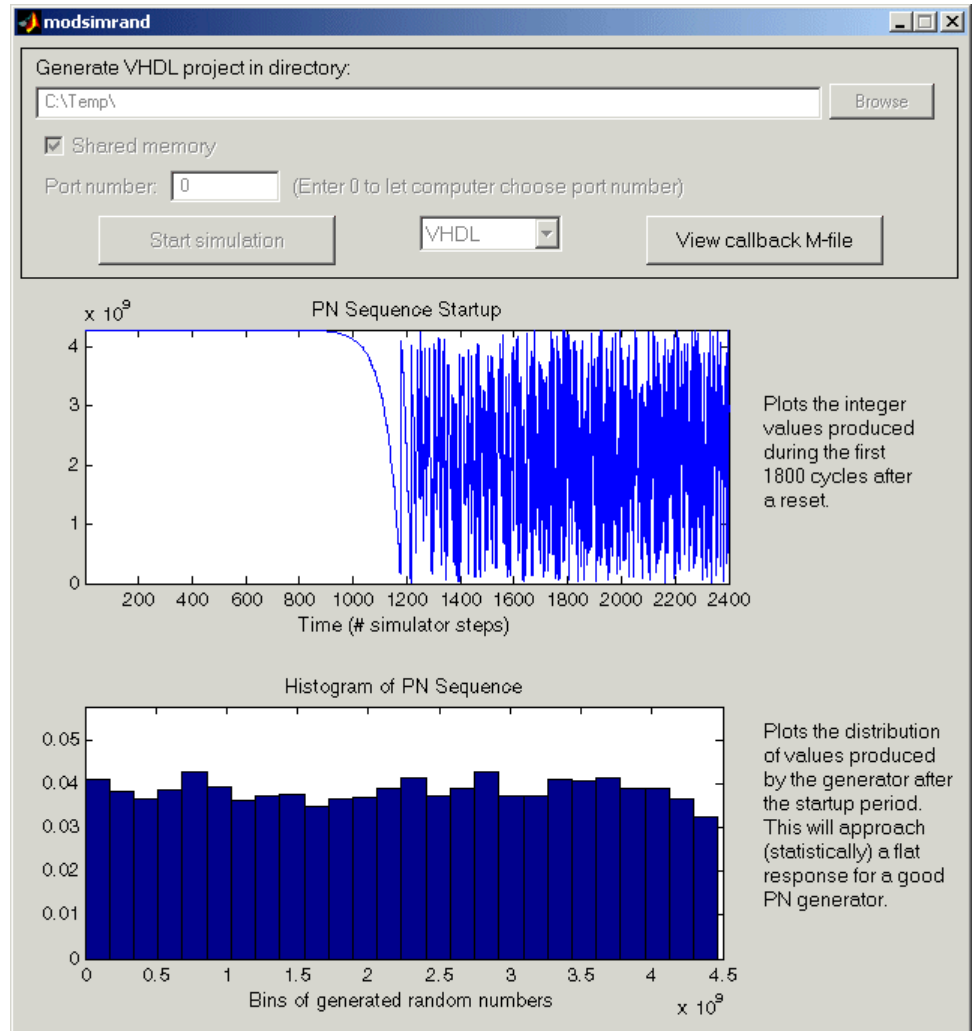
9 End the simulation in ModelSim by entering the quit command at the VSIM n> prompt.

10 Shut down 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

- Simulink Fixed Point

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

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

1 Start MATLAB and make it your active window.

2 Open the Simulink model `manchestermodel`.

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 dialog appears.
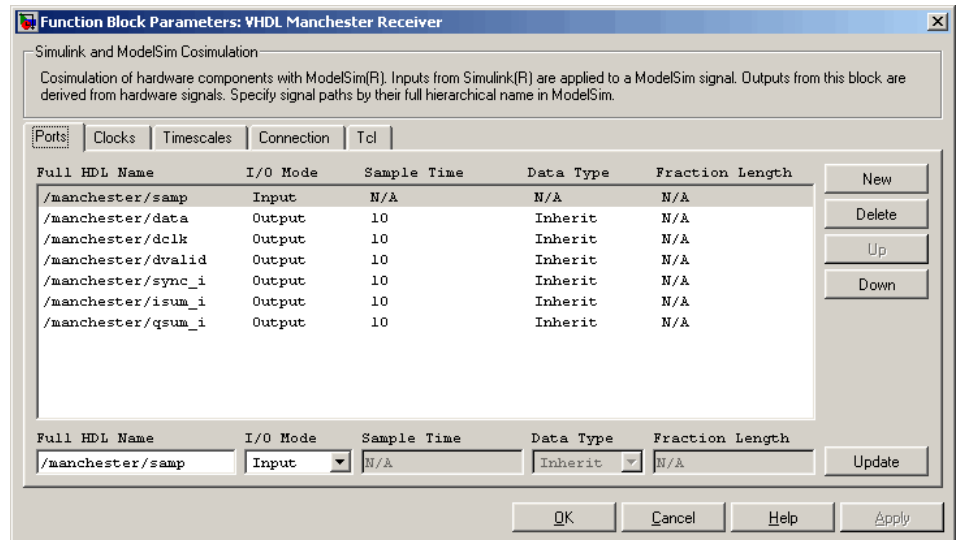
b Click the **Connection** tab. The dialog displays communication configuration information.
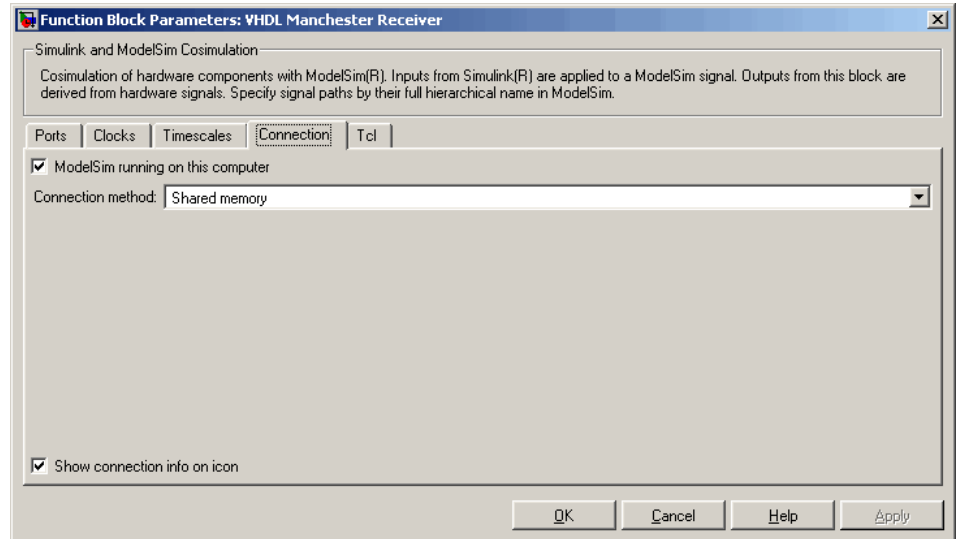
c Select Socket from the **Connection method** list.

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.

    If you configured the model for a shared memory link, use the command

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

    Alternatively, you can just double-click the button labeled **ModelSim Startup Command**.

    If you configured the model for a TCP/IP socket link, use the command

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

Alternatively, you can just double-click the button labeled **ModelSim Startup Command 1**.

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.
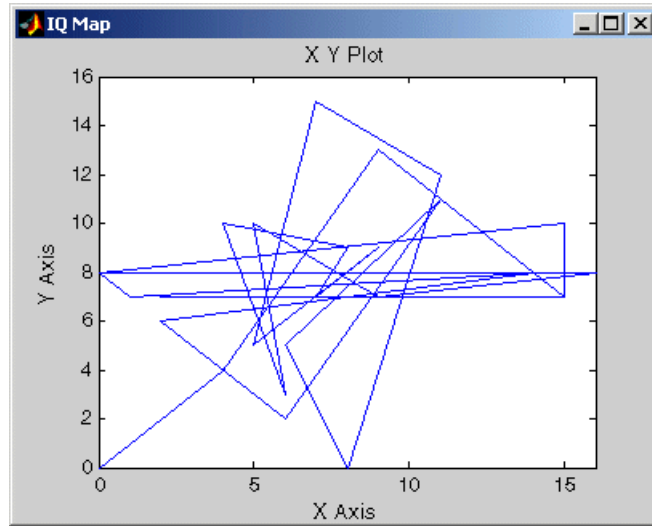
2

MATLAB and ModelSim Tutorial

This chapter guides you through the basic steps for setting up an application of the Link for ModelSim that uses MATLAB to verify a simple VHDL model of a pseudorandom 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

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

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

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

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

Starting the MATLAB Server

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

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 in TCP/IP socket mode, the message reads

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

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

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

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

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

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

```
HDLDaemon server was shut down
```

5 The next step is to start the server in TCP/IP socket mode. To do this, call hdldaemon with the property name/property value pair 'socket' 0. The

value 0 specifies that the operating system assign the server a TCP/IP socket port that is available on your system. For example

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

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

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

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

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

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

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

Setting Up ModelSim

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

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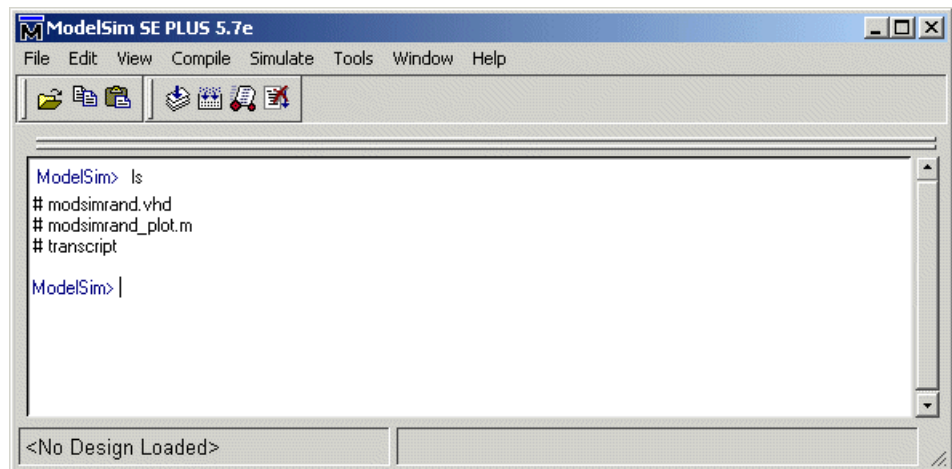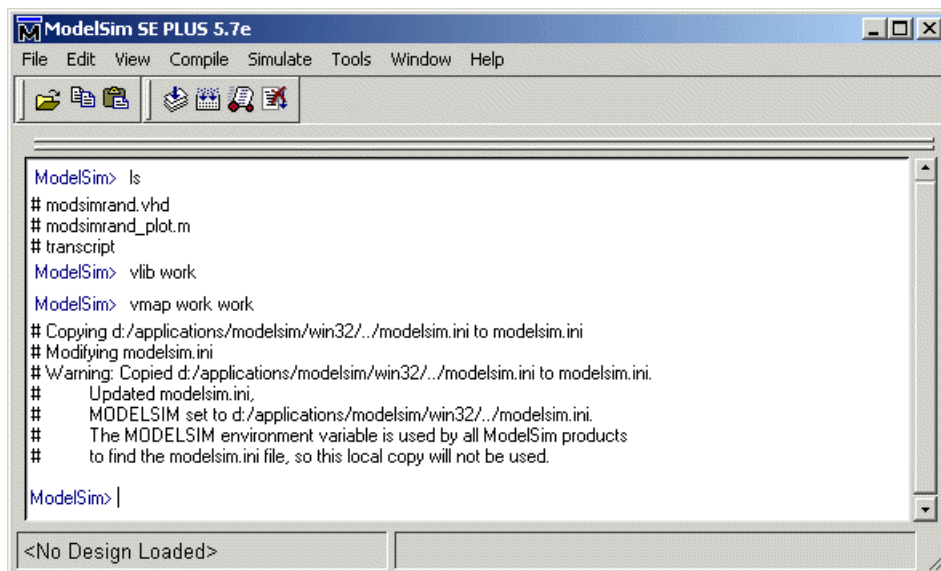
The command should list the files `modsimrand.vhd`, `modsimrand_plot.m`, and `transcript`.

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
```


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
```
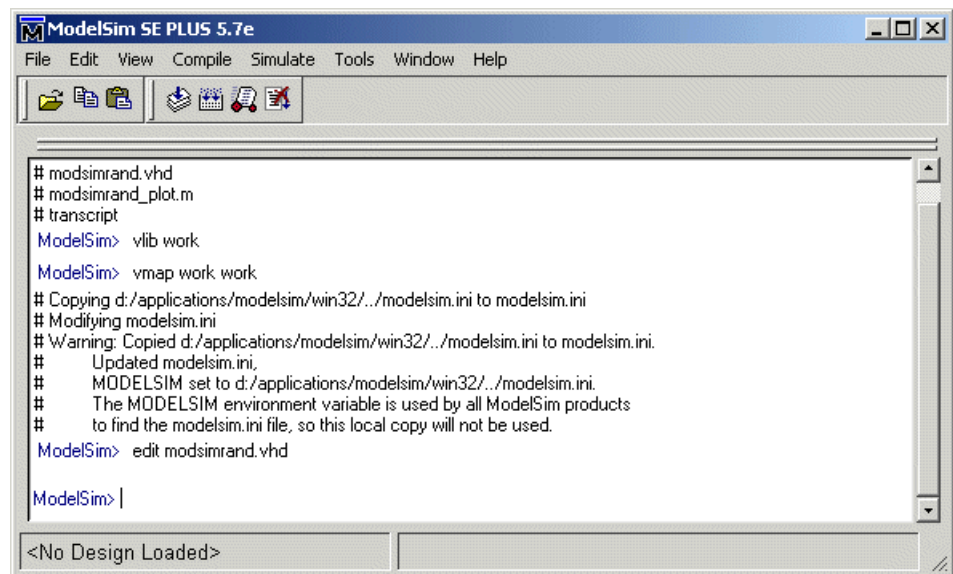

ModelSim opens its **edit** window and displays the VHDL code for modsimrand.vhd.

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

```
ENTITY modsimrand IS
PORT (
  clk     : IN std_logic ;
  clk_en  : IN std_logic ;
  reset   : IN std_logic ;
  dout    : OUT std_logic_vector (31 DOWNTO O);
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.
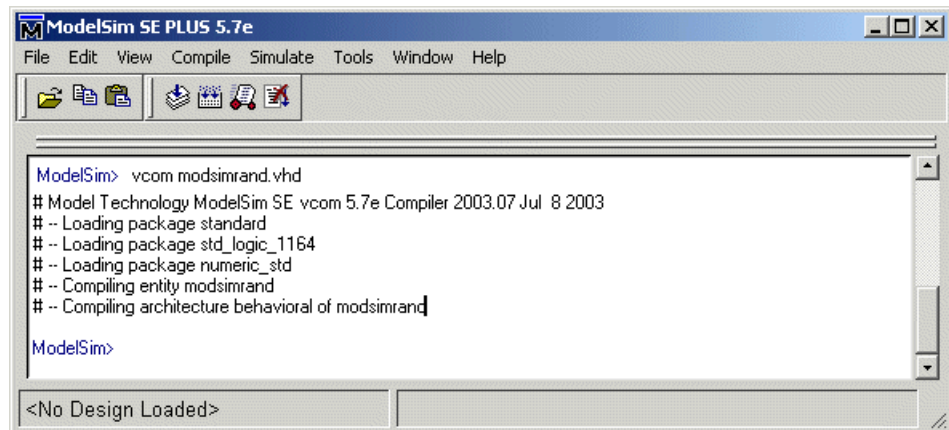
3 Browse through the rest of modsimrand.vhd. The remaining code defines a behavioral architecture for modsimrand that writes a randomly generated Fibonacci sequence to an output register when the clock experiences a rising edge.

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.

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.
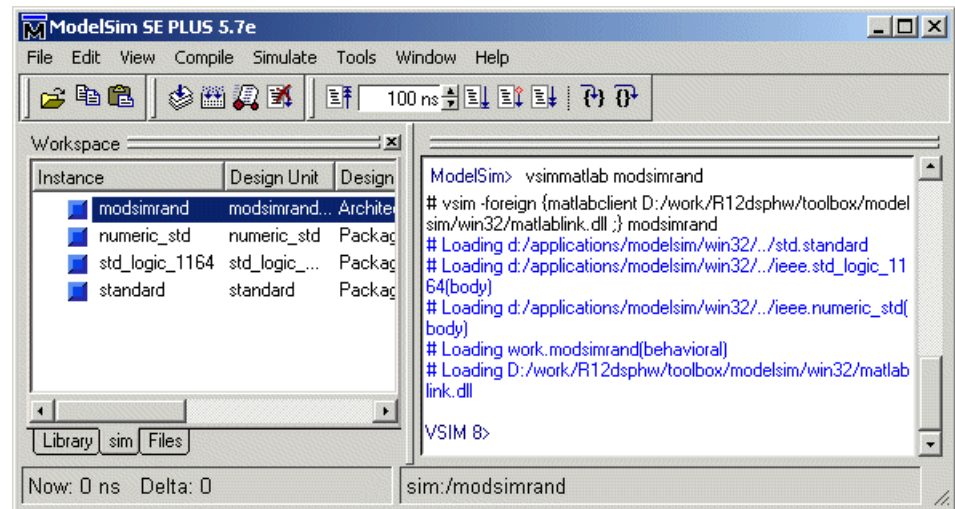
2 Initialize the simulator for verifying `modsimrand` with MATLAB. You initialize ModelSim by using the `matlabtb` or `matlabtbeval` ModelSim command. These commands define the communication link and a callback to a MATLAB function that executes in MATLAB on behalf of ModelSim.

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/clk -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 O 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:

`modsimrand`	The instance of the VHDL entity that is to be attached to a MATLAB function.
`-mfunc modsimrand_plot`	The MATLAB function to be called on behalf of entity `modsimrand`.
`-rising /modsimrand/clk`	The function `modsimrand_plot.m` be called when the signal `/modsimrand/clk` 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.
`-socket portnum`	The TCP/IP socket port *portnum* 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/clk 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/clk 0 0 ns, 1 5 ns -repeat 10 ns
VSIM n> force /modsimrand/clk_en 1
VSIM n> force /modsimrand/reset 1 0, 0 50 ns
```

The first command forces the clk 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 clk_en to 1 and reset to 1 at 0 nanoseconds and to 0 at 50 nanoseconds.

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

Developing the MATLAB Function

The 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.

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

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, the 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:

iport	Structure that specifies IN ports to be forced.
tnext	Specifies an optional future time at which the MATLAB function is called back.

oport	Structure that receives signal values from the OUT ports defined for the corresponding VHDL entity at the time specified by tnow.
tnow	Receives the simulation time at which the MATLAB function is called.
portinfo	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 Note that the function outputs iport and tnext must be initialized to empty values, as in the following code excerpt:

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

4 Make note of the data types of ports defined for the entity under simulation. The 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.

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

```
ud.buffer(cyc) = mvl2dec(oport.dout)
```

In this case, the function receives `STD_LOGIC_VECTOR` data on `oport`. The function `mvl2dec` 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.

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

4 Search for `oport.dout` and set a breakpoint at that line by clicking next to the line number. A red breakpoint marker will appear.

5 Return to ModelSim and enter the following command in the command window:

```
Vsim n> run 80000
```

This command instructs ModelSim to advance the simulation 80,000 time steps (80,000 nanoseconds using the default time step period). Since you previously set a breakpoint in `modsimrand_plot.m`, however, the simulation runs in MATLAB until it reaches the breakpoint. ModelSim is now blocked and remains blocked until you explicitly unblock it. While the

simulation is blocked, note that MATLAB displays the data that ModelSim passed to the MATLAB function in the **Workspace** window.
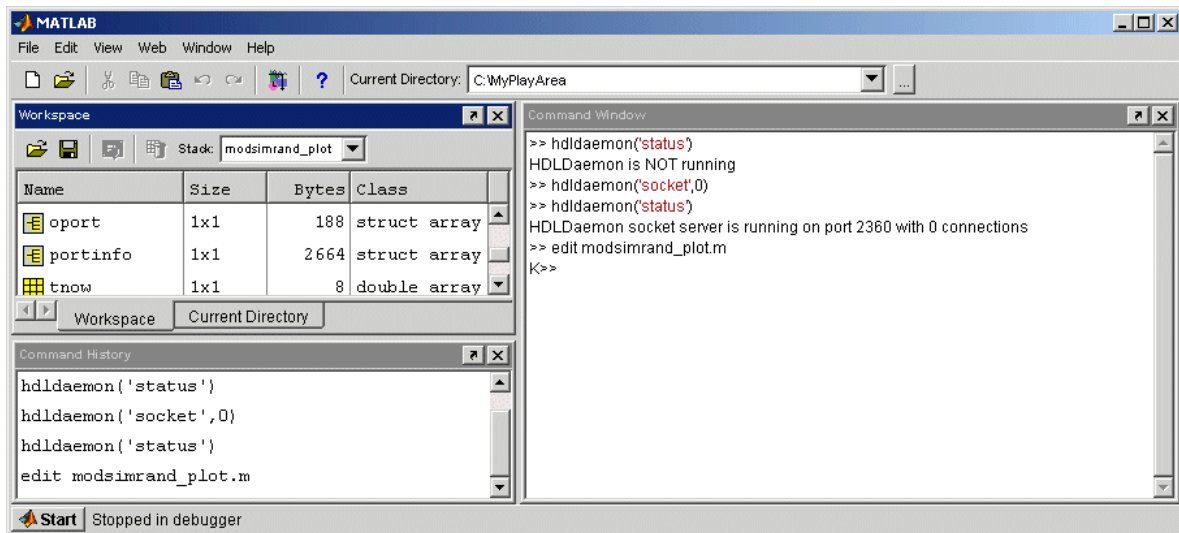
Note also that a figure window is displayed. This window is used to plot data generated by the simulation; initially it is empty.

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

7 Click **Debug–>Continue** in the MATLAB Edit/Debug window. The next time the breakpoint is reached, notice that portinfo is no longer visible in the MATLAB workspace. This is because portinfo is passed in only on the first function invocation. Also note that the value of tnow advances from 0 to 5e-009.

8 Clear the breakpoint by clicking the red breakpoint marker.

9 Unblock ModelSim and continue the simulation by clicking **Debug–>Continue** in the MATLAB Edit/Debug window.

The simulation runs to completion. As the simulation progresses, it plots generated data in a figure window. When the simulation completes, the figure window appears as shown below.
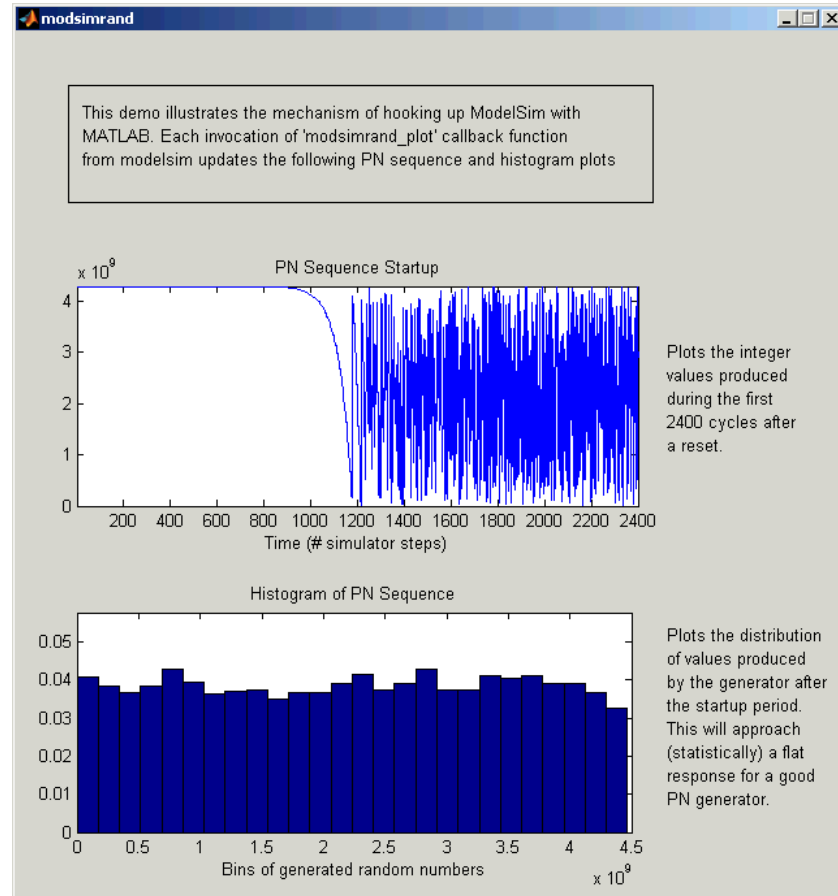
If you want to run the simulation again, you must restart the simulation in ModelSim, reinitialize the clock, and reset input signals. To do this:

1 Close the figure window.

2 Restart the simulation with the following command:

```
VSIM n> restart
```

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

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

---

3 Reissue the `matlabtb` command.

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

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

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

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

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

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

```
Vsim n> Vsim n> run 80000
```

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

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

2-21

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 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.

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:
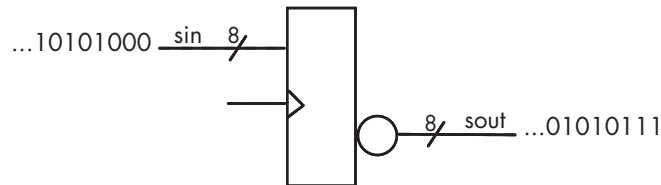
The VHDL entity for this model will represent 8-bit streams of input and output signal values with an IN port and OUT port of type STD_LOGIC_VECTOR. An input clock signal of type STD_LOGIC will trigger the bit inversion process when set:

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 O);
  sout: OUT std_logic_vector(7 DOWNTO O);
  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:
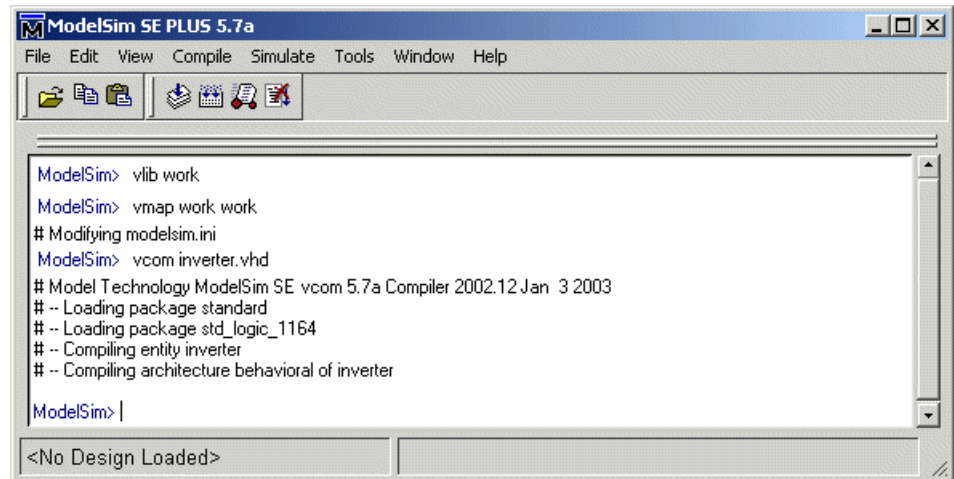
```
# ** Warning: (vlib-34) Library already exists at "work".
```

> **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.

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.

Start by creating a model, as follows:

1 Start MATLAB, if it is not already running. Open a new model window. Then, open the Simulink Library Browser.

2 Drag the following blocks from the Simulink Library Browser to your model window.

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

Arrange the three blocks as shown below.
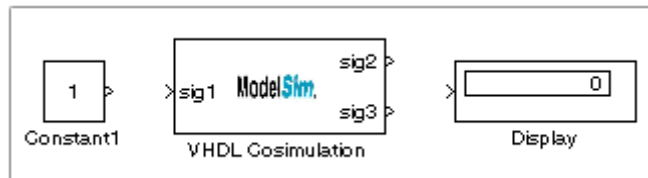
Next, configure the Constant block, which is the model's input source:

1 Double-click the Constant block icon to open the Constant block parameters dialog. Enter the following parameter values in the **Main** pane:

- **Constant value**: 0
- **Sample time**: 10

Later you can change these initial values to see the effect various sample times have on different simulation runs.

The dialog box should now appear as follows.
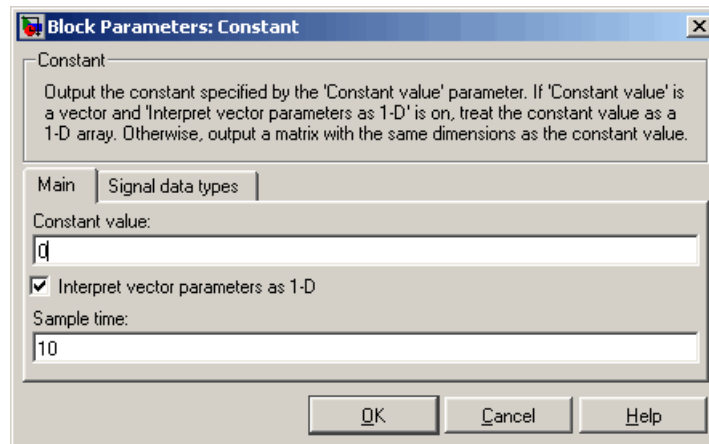
2 Click the **Signal data types** tab. The dialog box now displays the **Output data type mode** menu.

Select uint8 from the **Output data type mode** menu. This data type specification is supported by 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).

The dialog box should now appear as follows.
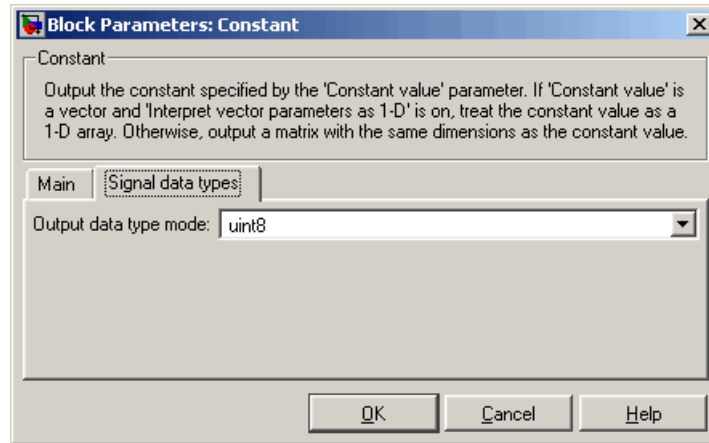
3 Click **OK**. The Constant block parameters dialog closes and the value in the Constant block icon changes to 0.

Next, configure the VHDL Cosimulation block, which represents the inverter model written in VHDL. Start with the **Ports** pane:

1 Double-click the VHDL Cosimulation block icon. The Block Parameters dialog for the VHDL Cosimulation block appears. Click on the **Ports** tab.

2 In the **Ports** pane, select the sample signal /top/sig1 from the signal list in the center of the pane.

3 In the **Full HDL Name** edit field, replace the sample signal pathname /top/sig1 with /inverter/sin. Then click the **Update** button. The signal name in the selected list entry changes.

4 Similarly, select the sample signal /top/sig2. Change the **Full HDL Name** to /inverter/sout. Change the **Sample Time** parameter to 10. Then select Output from the **I/O Mode** list. Then click the **Update** button to update the list.

5 Select the sample signal /top/sig3. Click the **Delete** button. Again, click the **Update** button. The signal is now removed from the list.

The **Ports** pane should appear as follows.
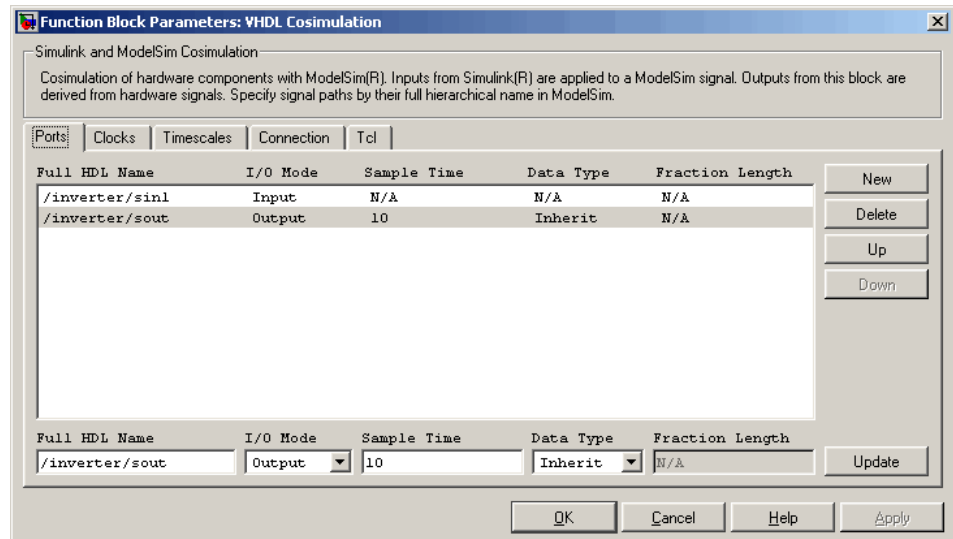
6 Note that the signal list has been changed, but the edits you have applied are not communicated to the Simulink model until you apply them. To do so, click **Apply**.

Now configure the parameters of the **Connection** pane:

1 Click the **Connection** tab.

2 Select socket from the **Connection method** list. This option specifies that Simulink and ModelSim will communicate via a designated TCP/IP socket port. Observe that two additional fields, **Port number or service** and **Host name**, are now visible.

Note that, because the **ModelSim running on this computer option** is selected by default, the **Host name** field is disabled. In this configuration, both Simulink and ModelSim execute on the same computer, so you do not need to enter a remote host system name.

3 In the **Port number or service** text box, enter socket port number 4449 or, if this port is not available on your system, another valid port number or service name. The model will use TCP/IP socket communication to link with ModelSim. Note what you enter for this parameter. You will specify

the same socket port information when you set up ModelSim for linking with Simulink.

The **Connection** pane should appear as follows.
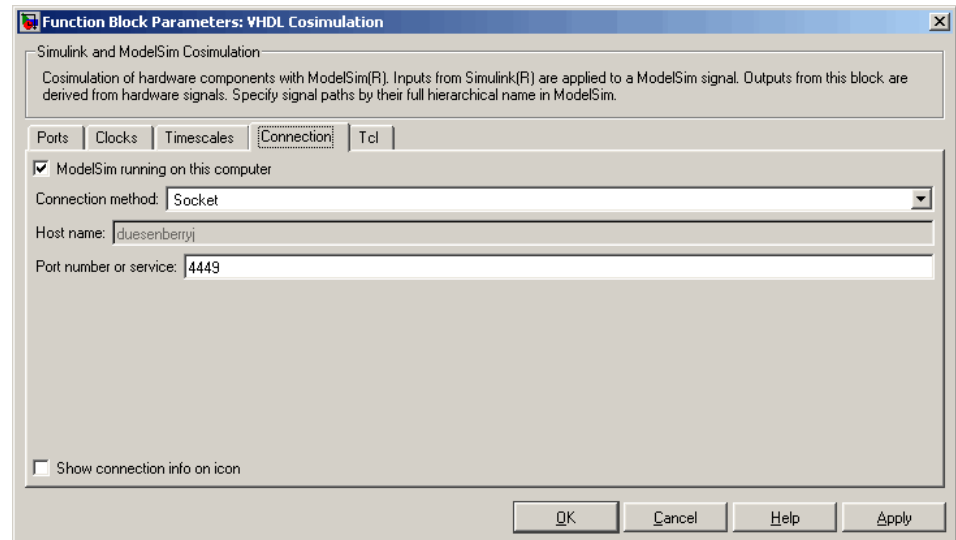
4 Click **Apply**.

Now configure the **Clocks** pane:

1 Click the **Clocks** tab.

2 Click the **New** button. A new clock signal with an empty signal name is added to the signal list; the new signal is selected for editing.

3 In the **Full HDL Name** text box, enter the signal path /inverter/clk. Then select Rising from the **Edge** list. Set the **Period** parameter to 10.

The **Clocks** pane should appear as follows.
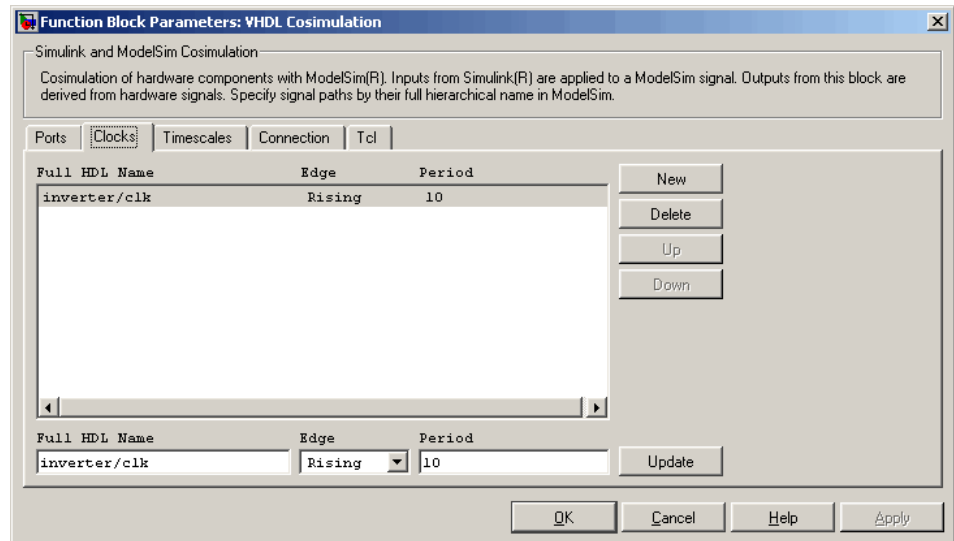
4 Click **Apply**.

Next, enter some simple Tcl commands to be executed before and after simulation:

1 Click the **Tcl** tab.

2 In the **Pre-simulation commands** text box, enter the following Tcl command:

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

3 In the **Post-simulation commands** text box, enter

```
echo "Done"
```

The **Tcl** pane should appear as follows.
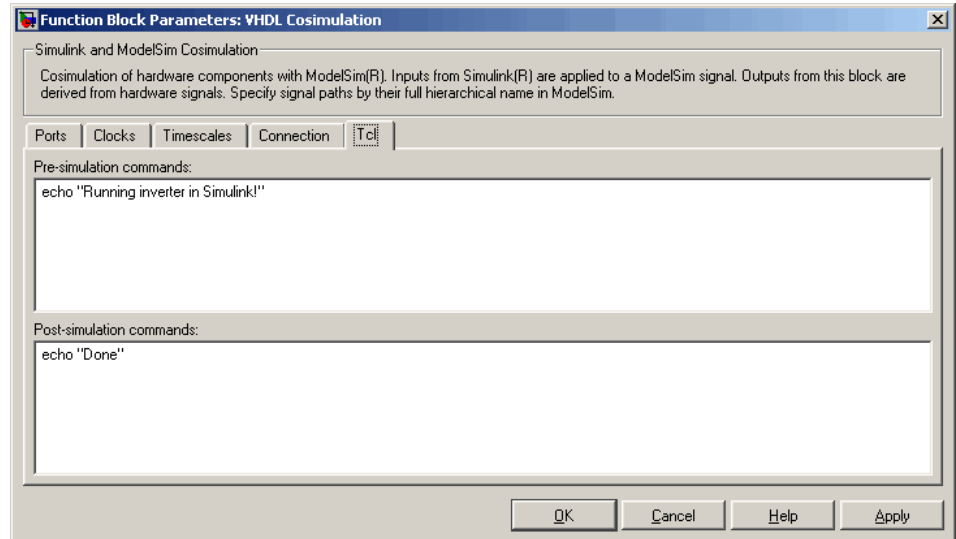
4 Click **Apply** and then **OK**.

Next, view the **Timescales** pane to make sure it is set to its default parameters.

1 Click the **Timescales** tab.

2 The default settings of the **Timescales** pane are shown below. These settings are required for correct operation of this example. See "Representation of Simulation Time" on page 7-9 for further information.
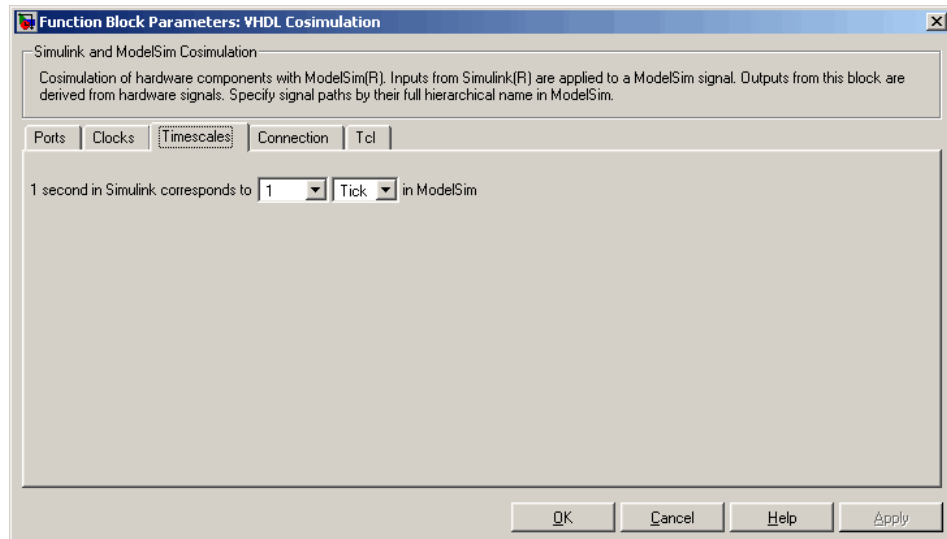
The final step is to connect the blocks, configure model-wide parameters, and save the model:

1 Connect the blocks as shown below.
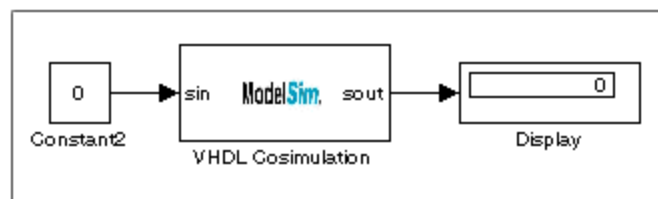
At this point, you might also want to consider adjusting block annotations.

2 Configure the Simulink solver options for a fixed-step, discrete simulation; this is required for correct cosimulation operation.

    a Select **Configuration Parameters** from the **Simulation** menu in the model window. The Configuration Parameters dialog box opens, displaying the **Solver options** pane.

    b Select Fixed-step from the **Type** menu.

c Select discrete (no continuous states) from the **Solver** menu.

d Click **Apply**. The **Solver options** pane should appear as shown below.
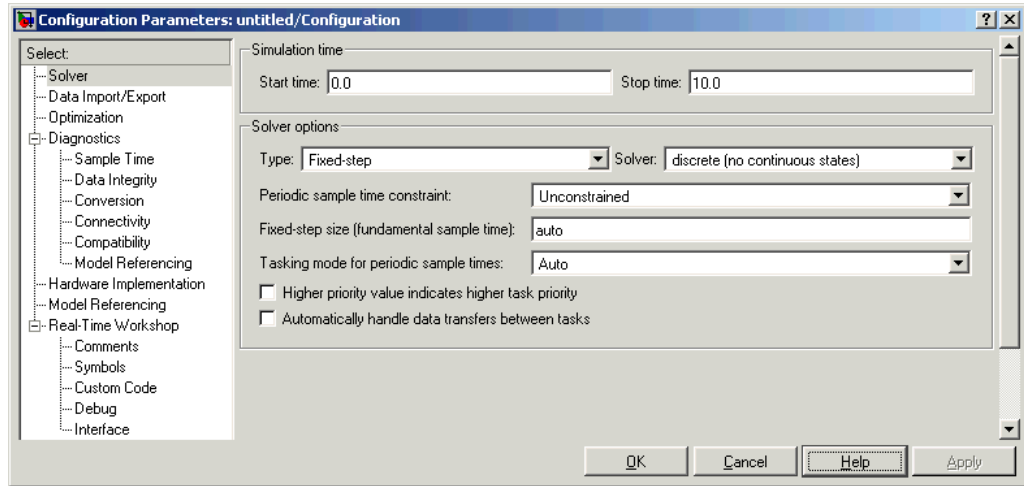
e Click **OK** to close the Configuration Parameters dialog box.

See "Configuring Simulink for VHDL Models" on page 7-26 for further information on Simulink settings that are optimal for use with the Link for ModelSim,

3 Save the model.

Setting Up ModelSim for Use with Simulink

You now have a VHDL representation of an inverter and a Simulink model that applies the inverter. To start ModelSim such that it is ready for use with Simulink, enter the following command line in the MATLAB Command Window:

```
vsim('socketsimulink', 4449)
```

Note If you entered a different socket port specification when you configured the 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.
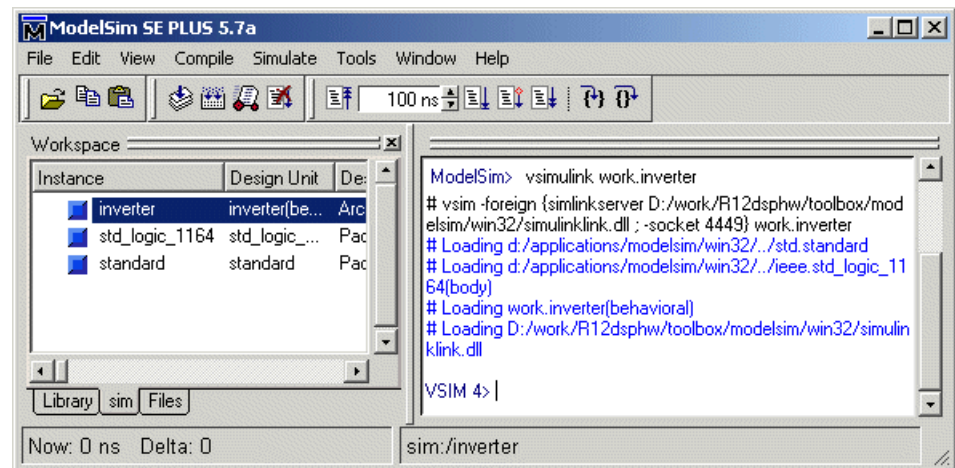
Running the Simulation

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 nn> add wave /inverter/*
```

The following **wave** window appears.
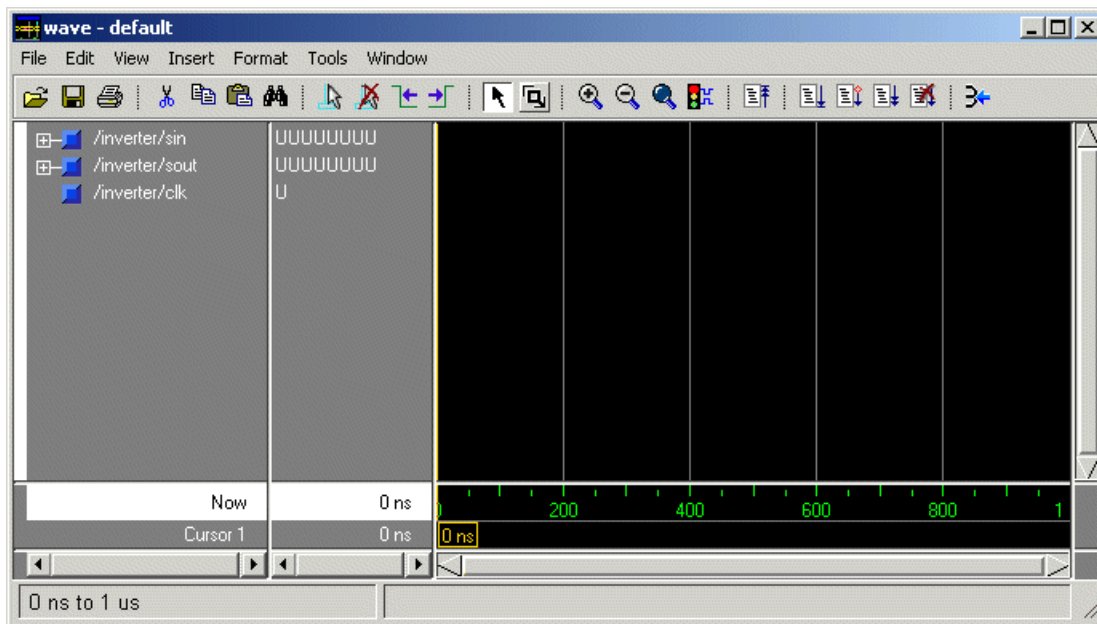
2 Change your input focus to your Simulink model window.

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.

3-17
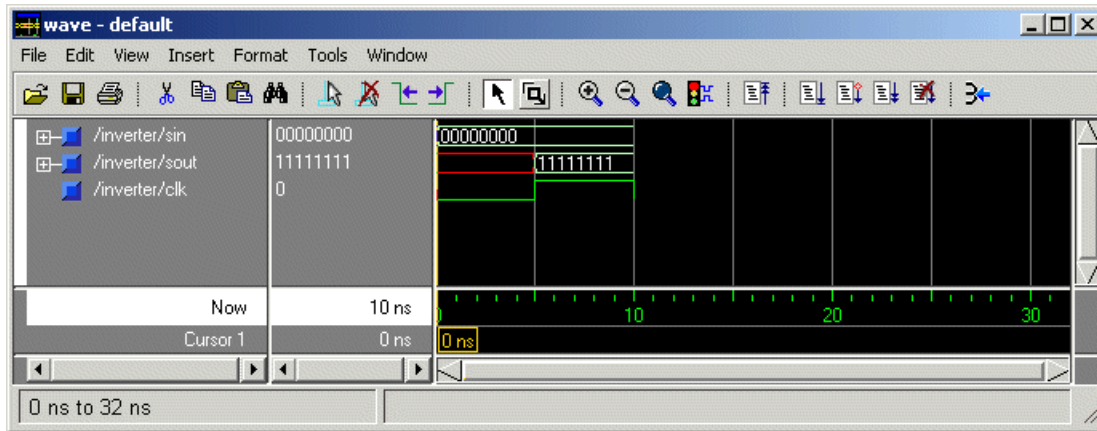
4 In the Simulink model, change **Constant value** to 255, save the model, and start another simulation. The value in the Display block changes to 0 and the ModelSim **wave** window is updated as follows.
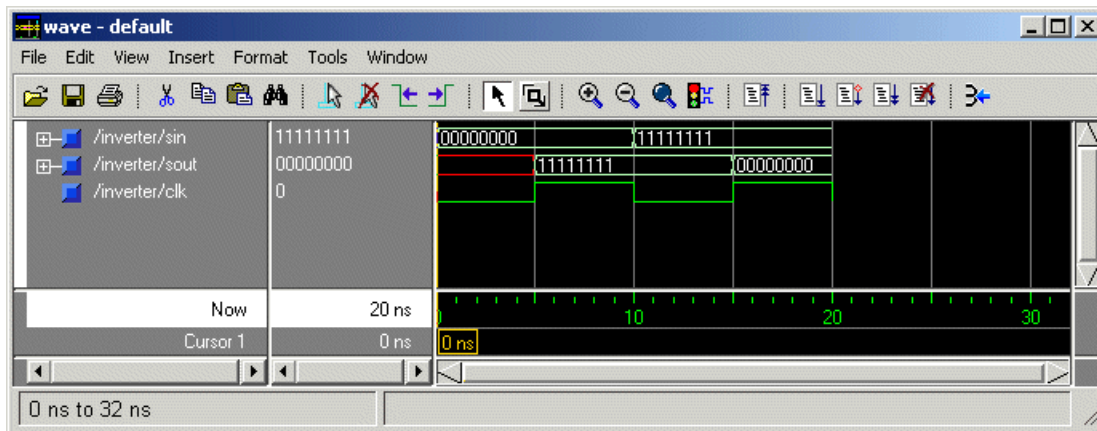
5 In the Simulink Model, change **Constant value** to 2 and **Simulation time** to 20 and start another simulation. This time, the value in the Display block changes to 253 and the ModelSim **wave** window appears as shown below.
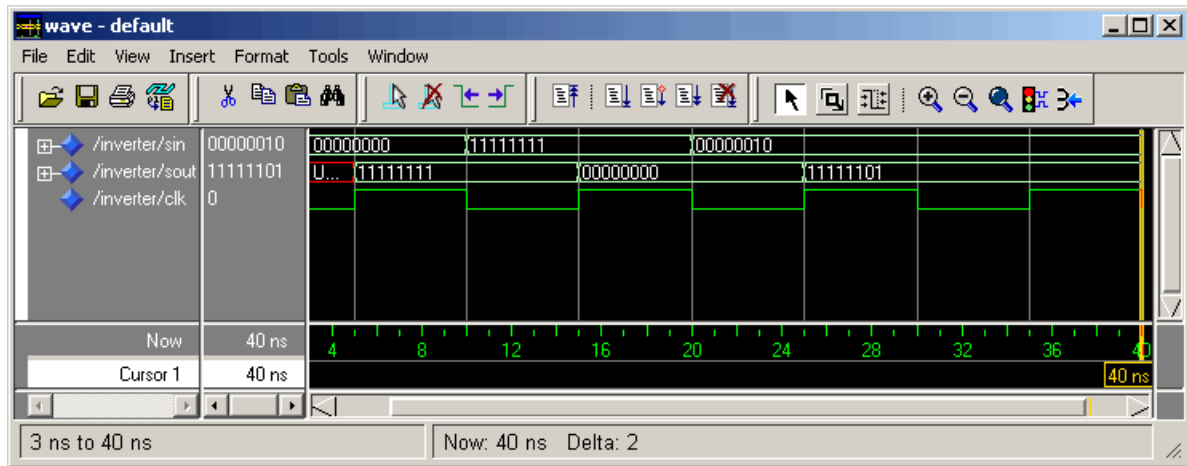
Note the change in the sample time in the **wave** window.

Shutting Down the Simulation

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

1 In ModelSim, stop the simulation by selecting **Simulate–>End Simulation**.

2 Quit ModelSim.

3 Close the Simulink model window.

4

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 the Link for ModelSim must be installed.

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

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.

Bit Setting	Transition	Encoded Waveform
1	1 to 0	
0	0 to 1	
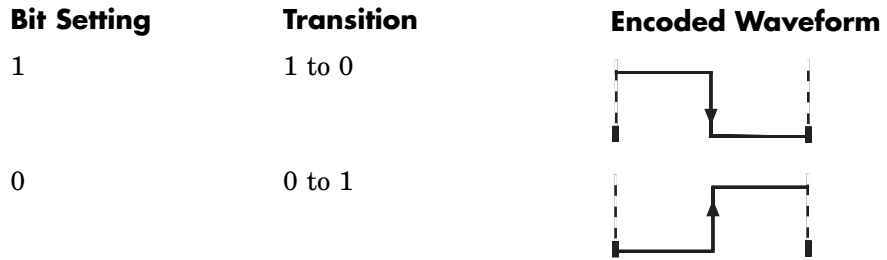
Transitions in the Manchester encoding always occur at the center of each clock cycle. The transition at the center is defined by the bit value. Transitions at the edges of data periods are possible, depending on the values of the previous and next bits. Consider the following diagram.
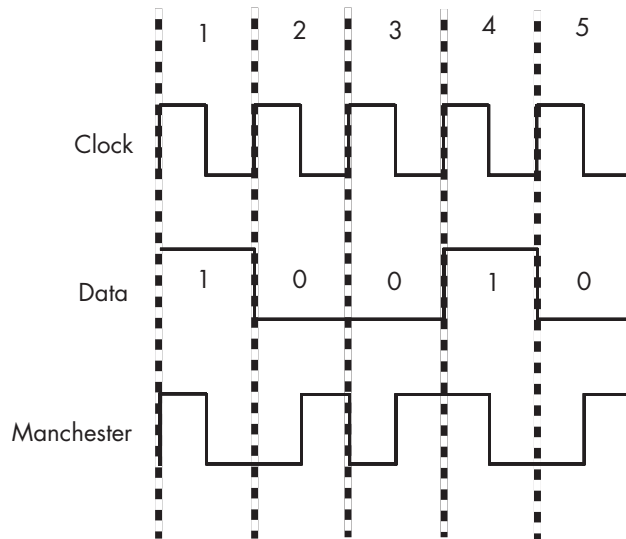
As the Manchester encoded signal in the diagram shows:

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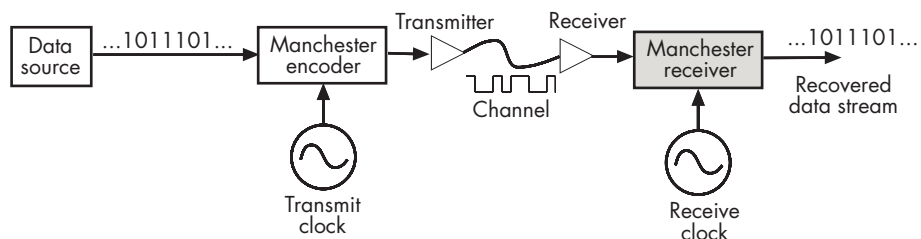
The Manchester receiver component validates the computations performed by a Manchester receiver device that is modeled in VHDL and simulated in ModelSim. Numerous approaches are available for implementing a

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 `isum` and `qsum` 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.
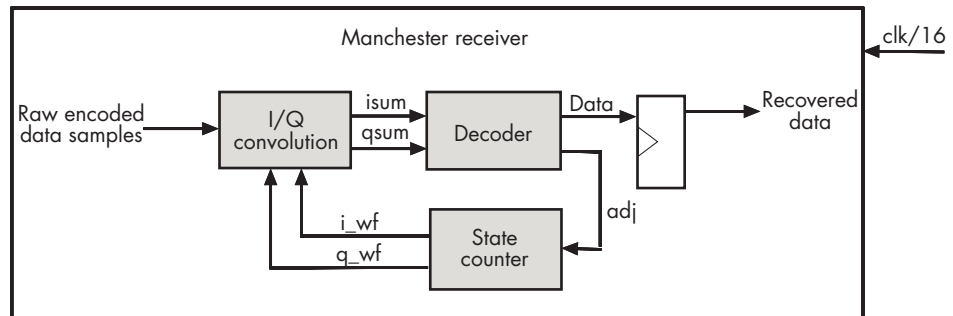
The following table describes the three sections of code.

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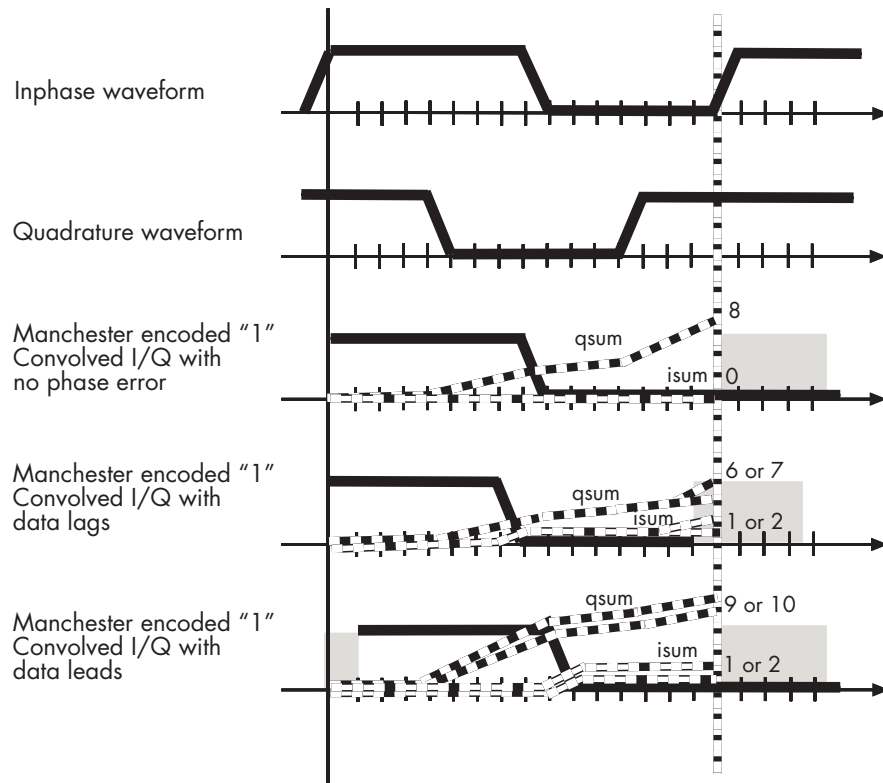
The following sections highlight areas of code in each of the three VHDL files that are of interest for a ModelSim and MATLAB test bench. The files are located in the modelsimdemos/vhdl/manchester directory:

- "VHDL Code for the I/Q Convolver" on page 4-11
- "VHDL Code for the Decoder " on page 4-13

• "VHDL Code for the State Counter" on page 4-14

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" on page 5-3.

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 `/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 `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 /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 `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" on page 5-3.

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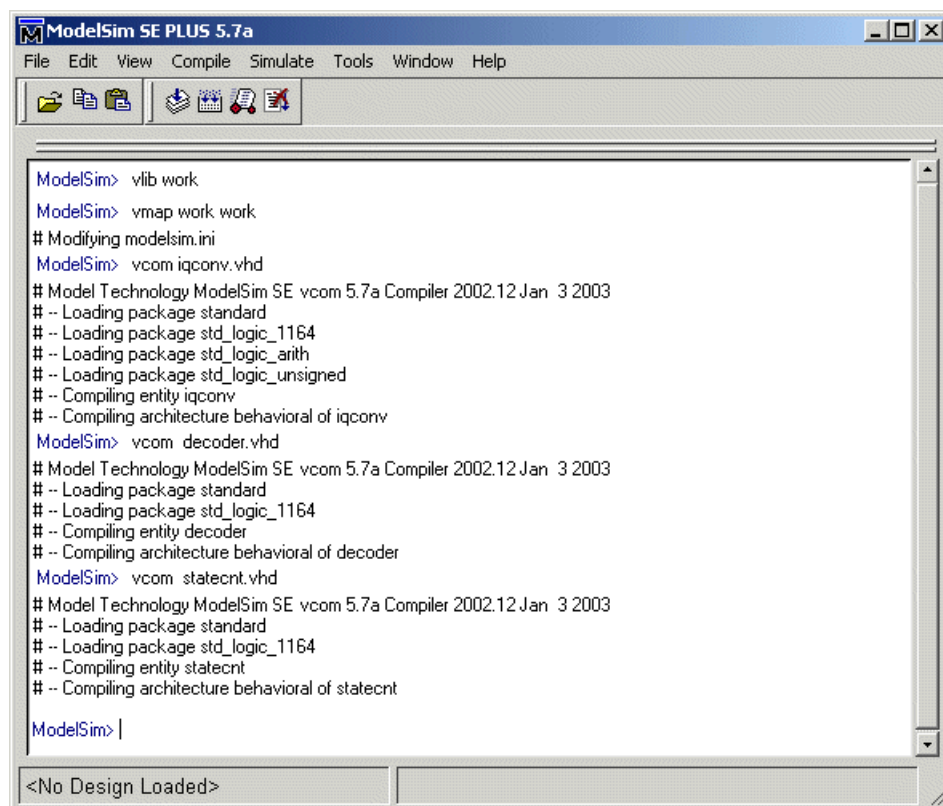
5 Compile the three VHDL files. One way of compiling a file is to click the filename in the project workspace and select **Compile–>Compile All**. Another alternative is to specify the name of the VHDL file with the `vcom` command, as follows:

```
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.

Developing the Manchester Receiver MATLAB Functions

The 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 (`i_wf` and `q_wf`). 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 4-24
- "MATLAB Function for the State Counter" on page 4-27

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 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`.

`iport`	Forces (by deposit) values onto signals connected to input ports of the VHDL entity — `reset`, `i_wf`, `q_wf`, and `samp`.
`tnext`	Specifies an optional time at which the MATLAB function is to be called back.

oport	Receives signal values from the output ports of the VHDL entity — isum and qsum — at the time specified by tnow.
tnow	Receives the simulation time at which the MATLAB function is called. By default, time is represented in seconds.
portinfo	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)
```

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

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

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

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 services the 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

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

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

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

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

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

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

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

    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 4-31
- "Starting the MATLAB Server from the Test Script" on page 4-32
- "Writing Script Code for the Decoder Test " on page 4-32
- "Writing Script Code for the I/Q Convolver Test" on page 4-35
- "Writing Script Code for the State Counter Test" on page 4-38

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 = hdldaemon('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 = strrep(projectdir,'\','/');
unixsrcfile = strrep(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.');
disp('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/clk 1 0, 0 5 ns -repeat 10 ns ',...
           'force /iqconv/enable 1',...
           'force /iqconv/reset 1',...
           'run 100',...
           ['matlabtb iqconv -rising /iqconv/clk -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 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 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 runs the ModelSim simulation for 1000 iterations of the current resolution limit. By default, the simulation runs for 1000 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','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 = 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

```
tclcmd = { ['cd '  unixprojectdir ],...
           'catch {wm geometry . 500x200+0+0}',...
           'vlib work',...
           ['vcom -performdefaultbinding ' unixsrcfile],...
           'vsimmatlab -t 1ns work.statecnt ',...
           'force /statecnt/clk 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 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 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 4-31. 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 O 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.
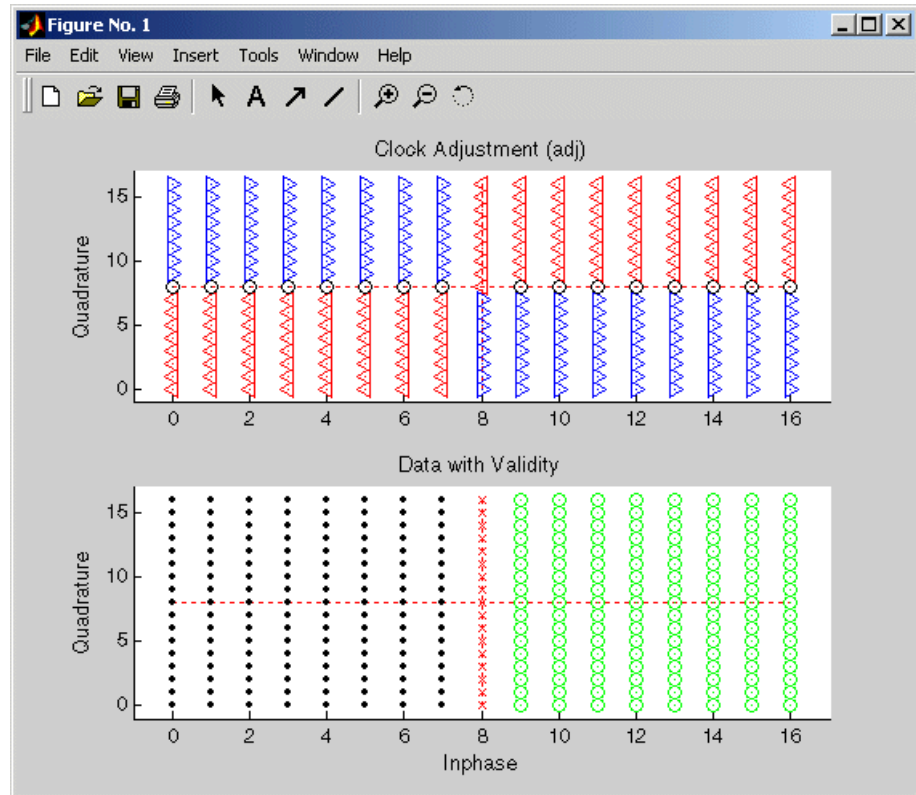
4 The decoder test then displays the following message in the MATLAB Command Window:

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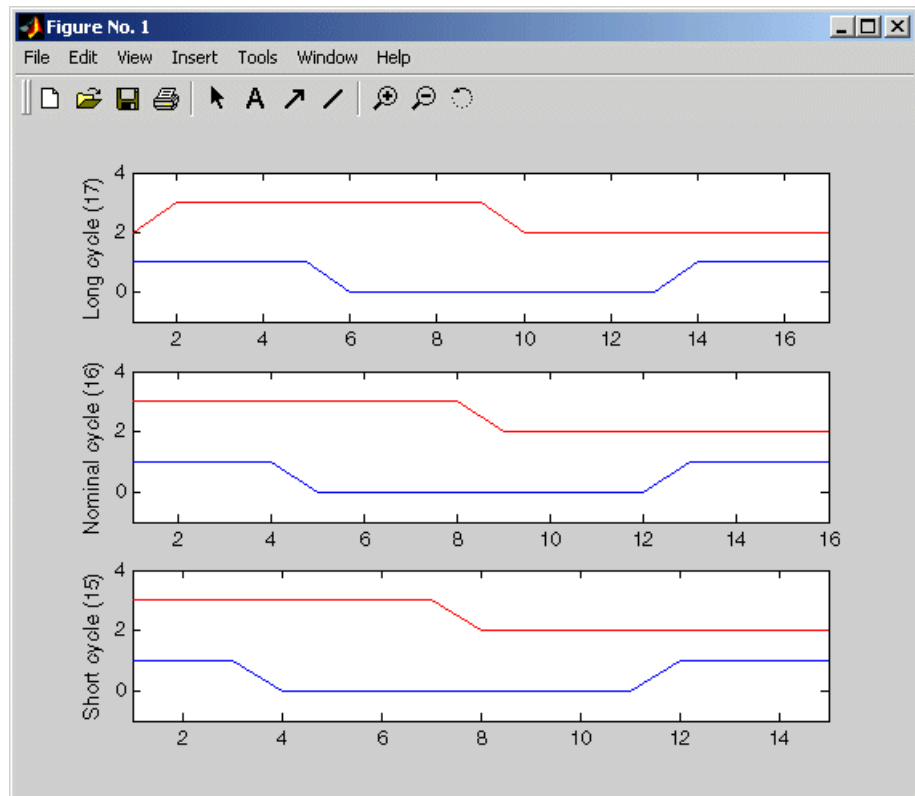
8 The state counter test then displays the following message in the MATLAB Command Window:

```
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 that can share data with the VHDL model. This chapter discusses the programming, interfacing, and scheduling conventions for MATLAB functions that communicate with ModelSim. The following topics are covered:

Overview

The Link for ModelSim supports two types of MATLAB functions that interface to VHDL models:

- *Test bench* functions let you verify the performance of the VHDL model, or of components within the model. A test bench function drives values onto signals connected to input ports of a VHDL entity under test, and receives signal values from the output ports of the VHDL entity.

- *MATLAB component* functions simulate the behavior of entities in the VHDL model. A stub entity (providing port definitions only) in the VHDL model passes its input signals to the MATLAB component function. The MATLAB component processes this data and returns the results to the outputs of the stub entity. A MATLAB component typically provides some functionality (such as a filter) that is not yet implemented in the VHDL code.

The programming, interfacing, and scheduling conventions for test bench functions and MATLAB component functions are almost identical. Most of this chapter focuses on test bench functions. The test bench section is followed by a discussion of MATLAB component functions and how to use them.

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

1 Code the VHDL model for MATLAB verification.

2 Compile and debug the VHDL model.

3 Code the required MATLAB test bench or MATLAB component 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 5-4
- "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 5-6

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...
IN	Represent signals that can be driven by a MATLAB function

Use Mode...	For Ports that...
OUT	Represent signal values that are passed to a MATLAB function
INOUT	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 creates and maps the design library work and compiles the VHDL file modsimrand.vhd:

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

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 5-14
- "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 5-21

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.
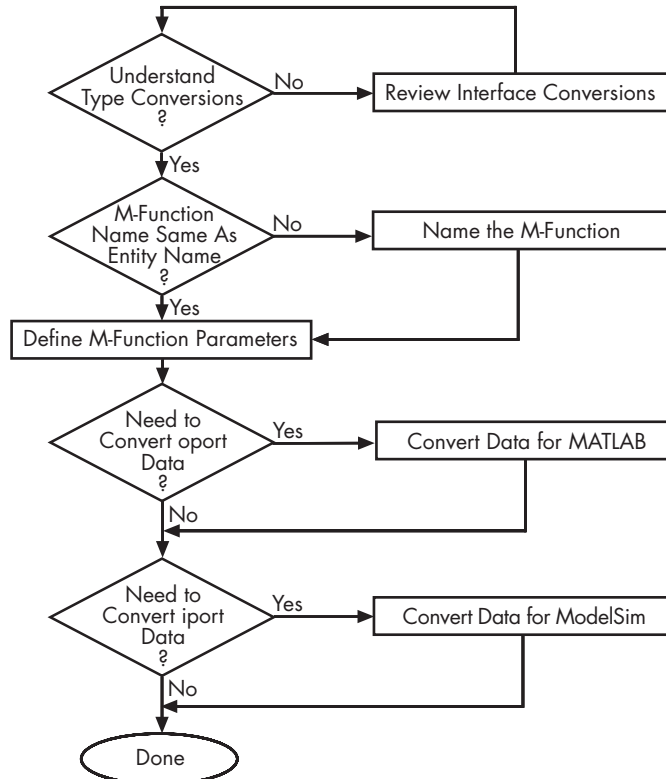
Coding a MATLAB Test Bench Function

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 required by your application. You may also need to handle differences between the array indexing conventions employed by VHDL and MATLAB.

The data types of arguments passed in to the function determine

- The types of conversions required before and after data is manipulated

- The types of conversions 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.

VHDL-to-MATLAB Data Type Conversions

VHDL Types...	As Scalar Converts to...	As Composite Converts to...
STD_LOGIC, STD_ULOGIC, and BIT	A character that matches the character literal for the desired logic state.	
STD_LOGIC_VECTOR, STD_ULOGIC_VECTOR, BIT_VECTOR, SIGNED, and UNSIGNED		A column vector of characters (as defined 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-to-MATLAB Data Type Conversions (Continued)

VHDL Types...	As Scalar Converts to...	As Composite Converts to...
TIME	Type double for time values in seconds and type int64 for values representing simulator time increments (see the description of the 'time' option in "Starting the MATLAB Server" on page 6-7).	Arrays of type double or int64 with a size that is equivalent to the VHDL port size.
Enumerated types	Character array (string) that contains the MATLAB representation of a VHDL label or character literal. For example, the label high converts to 'high' and the character literal 'c' converts to '''c'''.	Cell array of strings with each element equal to a label for the defined enumerated type. Each element is the MATLAB representation of a VHDL label or character literal. For example, the vector (one, '2', three) converts to the column vector ['one'; '''2'''; 'three']. A user-defined enumerated type that contains only character literals, converts to a vector or array of characters as indicated for the types STD_LOGIC_VECTOR, STD_ULOGIC_VECTOR, BIT_VECTOR, SIGNED, and UNSIGNED.

Array Indexing Differences Between MATLAB and VHDL

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 reverses the order of indexes between MATLAB and VHDL. For example the following VHDL program declares the port sta as an array of two 8-bit bytes.

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.numeric_std.all;

PACKAGE myporttype IS
    TYPE twobytes IS ARRAY(1 TO 2) OF BIT_VECTOR(1 TO 8);
END myporttype;

USE WORK.myporttype.all;

ENTITY index_port IS
PORT (
    sta : OUT twobytes );
END index_port ;

ARCHITECTURE rtl OF index_port IS
  CONSTANT myvalue : twobytes := ("00001011", "10101101");  -- 0x0bad
BEGIN
  sta <= myvalue;
END rtl;
```

In MATLAB, you could address a single element of this array as in the following example:

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

Note also that VHDL arrays indices are commonly zero-based. That is, they are 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.

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 lowercase 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 passed into the function through the output parameters is defined by the structure of the corresponding VHDL entity. The function parameters are

- `iport`: Structure that forces (by deposit) values onto signals connected to ports of the associated VHDL entity.

- `tnext` (optional): Specifies time at which the MATLAB callback function is executed. This parameter should be initialized to an empty value (`[]`). If it is not subsequently updated, 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.

- `oport`: Structure that receives VHDL signal values from the output ports defined for the associated VHDL entity at the time specified by `tnow`.

- `tnow`: 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.

- `portinfo`: For the first call to the function (at the start of the simulation) only, receives a structure whose fields describe the ports defined for the associated VHDL entity. For each port, the `portinfo` structure passes 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 can use the port information to create a generic MATLAB function that operates differently depending on the port information supplied at startup.

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

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

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

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 or Component Function" on page 6-14. For an example of how to use these parameters, see "Sample MATLAB Test Bench Function" on page 5-21. For more information on port data, see "Gaining Access to and Applying Port Information" on page 5-14.

Gaining Access to and Applying Port Information

The Link for ModelSim interface passes information about the entity under test in the `portinfo` structure. The `portinfo` structure is passed as the third argument to the function. It is passed only in the first call to your MATLAB function. The information passed in the `portinfo` structure is useful for validating the entity under simulation. You could use 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. The content of these 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...
field1	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.	All types
field2	*portname*	Is the name of the port	All port types

VHDL Port Information (Continued)

Field...	Can Contain...	Which...	And Applies to...
field3	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. 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 double	Use the double function to convert the data to type double before performing the computation. For example: `datas(inc+1) = double(idata);`
Convert a standard logic or bit vector to an unsigned integer	Use the bin2dec function to convert the data to an unsigned decimal value. For example: `uval = bin2dec(oport.val')` 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.

Required Data Conversions (Continued)

If the Function Needs to...	Then...

Convert a standard logic or bit vector to a signed integer	Use the following application of the bin2dec function to convert the data to a signed decimal value. For example:
	
	``
	suval =
	bin2dec(oport.val')-2^length(oport.val);
	``
	
	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.

Test port values of type STD_LOGIC and STD_LOGIC_VECTOR	Use the all function as follows: ``` all(oport.val == '1' | oport.val == '0') ``` This example returns True if all elements are '1' or '0'.

Converting Data for Return to 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

To Return Data to an IN Port of Type...	Then...
STD_LOGIC, STD_ULOGIC, or BIT	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', 'O', '1', 'Z', 'W', 'L', 'H', or '-'. For BIT, the character can be 'O' or '1'. For example: ```\niport.s1 = 'X'; %STD_LOGIC\niport.bit = '1'; %BIT\n```
STD_LOGIC_VECTOR, STD_ULOGIC_VECTOR, BIT_VECTOR, SIGNED, or UNSIGNED	Declare the data as a column vector or row vector of characters (as defined above) with one bit per character. For example: ```\niport.s1v = 'X1OZZ'; %STD_LOGIC_VECTOR\niport.bitv = '10100'; %BIT_VECTOR\niport.uns = dec2bin(10,8); %UNSIGNED, 8 bits\n```
Array of STD_LOGIC_VECTOR, STD_ULOGIC_VECTOR, BIT_VECTOR, SIGNED, or UNSIGNED	Declare the data as an array of type character with a size that is equivalent to the VHDL port size. 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. VHDL array indexing: ```\nPORT (\nsta : OUT ARRAY(1 TO 2) OF\nBIT_VECTOR(1 TO 8););\n.\n.\n.\nsta(2)(7) <= '1'\n```

Conversions for ModelSim (Continued)

To Return Data to an IN Port of Type...	Then...
	MATLAB equivalent array indexing: `iport.sta(7,2) = '1';`
`INTEGER or NATURAL`	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: `iport.int = int32(1:10)';`
`REAL`	Declare the data as an array of type `double` with a size that is equivalent to the VHDL port size. For example: `iport.dbl = ones(2,2);`
`TIME`	Declare a VHDL `TIME` value as time in seconds, using type `double`, or as an integer of simulator time increments, using type `int64`. You can use the two formats interchangeably and what you specify does not depend on the `hdldaemon 'time'` option (see "Starting the MATLAB Server" on page 6-7), which applies to IN ports only. Declare an array of `TIME` values by using a MATLAB array of identical size and shape. All elements of a given port are restricted to time in seconds (type `double`) or simulator increments (type `int64`), but otherwise you can mix the formats. For example: `iport.t1 = int64(1:10)'; %Simulator time`

Conversions for ModelSim (Continued)

To Return Data to an IN Port of Type...	Then...
	``` %increments iport.t2 = 1e-9; %1 nsec ```
Enumerated types	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 5-14). 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. ``` iport.char = {'''A''', '''B'''}; %Character %literal iport.udef = 'mylabel'; %User-defined label ```
Character array for standard logic or bit representation	Use the dec2bin function to convert the integer. For example: ``` oport.slva =dec2bin([23 99],8)'; ``` This example converts two integers to a 2-element array of standard logic vectors consisting of 8 bits.

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 files:

- MATLABROOT\toolbox\modelsim\modelsimdemos\vhdl\manchester\decoder.vhd

- MATLABROOT\toolbox\modelsim\modelsimdemos\decoder.m

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

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

The following discussion highlights key lines of code in the definition of the manchester_decoder MATLAB function.

1 Specify the MATLAB function name and required parameters.

The function definition on the first line represents the communication channel between MATLAB and ModelSim. The following code is the function definition of the manchester_decoder MATLAB function.

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

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.

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

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

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

iport   Forces (by deposit) a value onto the signal connected to the entity's input ports, `isum` and `qsum`.

tnext   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 call to the function, receives a structure that describes the ports defined for the entity.

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

Port...	Of Direction...	And Type...	Converts to/Requires Conversion to...
isum	IN	STD_LOGIC_VECTOR(4 DOWNTO 0)	A 5-bit column or row vector of characters where each bit maps to standard logic character 0 or 1.
qsum	IN	STD_LOGIC_VECTOR(4 DOWNTO 0)	A 5-bit column or row vector of characters where each bit maps to standard logic character 0 or 1.

Example Port Definitions (Continued)

Port...	Of Direction...	And Type...	Converts to/Requires Conversion to...
adj	OUT	STD_LOGIC_VECTOR(1 DOWNTO 0)	A 2-element column vector of characters. Each character matches a corresponding character literal that represents a logic state and maps to a single bit.
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.

5-25

3 Set up any required timing parameters.

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

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

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

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

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

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 following code excerpt illustrates data type conversion of data to be returned to ModelSim.

```
if isum == 17
   iport.isum = dec2bin(isum,5);
   iport.qsum = dec2bin(qsum,5);
else
   iport.isum = dec2bin(isum,5);
end
```

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.

Coding a MATLAB Component Function

This section discusses the syntax of a MATLAB component function and the relationship of the function to its associated VHDL entity.

Function Definition and Parameters

The syntax of a MATLAB component function is

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

The function returns the following outputs:

- `oport`: Structure that forces (by deposit) values onto signals connected to output ports of the associated VHDL entity.

- `tnext` (optional): Specifies the time at which ModelSim schedules the next callback to MATLAB. `tnext` should be initialized to an empty value (`[]`). If `tnext` is not subsequently updated, no new entries are added to the simulation schedule. In that case, callback scheduling is controlled by the `matlabcp` command.

  For more information see "Controlling Callback Timing from a MATLAB Test Bench or Component Function" on page 6-14.

We strongly recommend that you initialize the function outputs to empty values at the beginning of the function as in the following example:

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

The parameters passed in to the function are as follows:

- `iport`: Structure that receives VHDL signal values from the input ports defined for the associated VHDL entity at the time specified by `tnow`.

- `tnow`: Receives the simulation time at which the MATLAB function is called. By default, time is represented in seconds. For more information see "Controlling Callback Timing from a MATLAB Test Bench or Component Function" on page 6-14.

- `portinfo`: For the first call to the function only (at the start of the simulation), `portinfo` receives a structure whose fields describe the ports defined for the associated VHDL entity. For each port, the `portinfo` structure passes information such as the port's type, direction, and size. You can use the port information to create a generic MATLAB function that operates differently depending on the port information supplied at startup. For more information on port data, see "Gaining Access to and Applying Port Information" on page 5-14.

For more information on using `tnext` and `tnow` for simulation scheduling, see "Deciding on Test Bench Scheduling Options" on page 6-13.

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

The next section provides an example of how to use the parameters of a MATLAB component function.

Sample MATLAB Component Function

This section illustrates the programming conventions required for a MATLAB component function, using code examples from the Link for ModelSim Oscillator demo.

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

To run the demo:

1 Type the following command at the MATLAB prompt:

```
demos
```

2 The Help browser opens with the **Demos** pane selected. In the **Demos** pane, select **Toolboxes– >Link for ModelSim–>Link for ModelSim MATLAB Component demos–>Implementing the filter component of an oscillator in MATLAB**.

3 In the right pane of the Help browser, click **Run in the Command window**.

4 The demo then displays instructions in the MATLAB Command Window.

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

- simple_osc.vhd: Contains entity and architecture definitions for oscillator component.

- osc_filter.vhd: Contains entity and architecture (stub) definitions for filter component.

- osc_top.vhd: Top-level VHDL behavioral model; instantiates and connects oscillator and filter components.

The directory MATLABROOT\toolbox\modelsim\modelsimdemos contains the demo M-files:

- modsimosc.m: Top-level demo script; starts up hdldaemon and ModelSim, passing in a cell array of Tcl commands to vsim..

- osccmds.m: M-function builds and returns a cell array of Tcl commands directing compilation and simulation of the model.

- oscfilter.m: MATLAB component function, called from ModelSim; performs filter computations.

The next section examines how the oscfilter function receives, processes, and returns data to the VHDL model.

The VHDL Entity

The VHDL entity osc_filter is defined in osc_filter.vhd as follows:

```
ENTITY osc_filter IS
   PORT( clk              : IN    std_logic;
         clk_enable       : IN    std_logic;
         reset            : IN    std_logic;
         osc_in           : IN    std_logic_vector(21 DOWNTO 0);
         filter1x_out     :  OUT   std_logic_vector(21 DOWNTO 0);
         filter4x_out     :  OUT   std_logic_vector(21 DOWNTO 0);
         filter8x_out     :  OUT   std_logic_vector(21 DOWNTO 0)
         );
END osc_filter;
```

Since all processing for this entity is done by the MATLAB component function, an empty architecture is defined:

```
ARCHITECTURE matlab OF osc_filter IS

BEGIN
END matlab;
```

Associating the VHDL Component with the MATLAB Component Function

The VHDL model instantiates this entity as the component u_osc_filter (see osc_top.vhd). After ModelSim compiles and loads the VHDL model, an association must be formed between the u_osc_filter component and the MATLAB component function oscfilter. To do this, the ModelSim command matlabcp is invoked when the simulation is set up (see osccmds.m).

```
matlabcp u_osc_filter -mfunc oscfilter
```

The matlabcp command instructs ModelSim to call back the oscfilter function when u_osc_filter executes in the simulation. matlabcp also defines a mapping between the data modeled in the VHDL entity and data in the MATLAB environment. See the matlabcp reference documentation for further information.

MATLAB Function Definition and Required Parameters

The function definition for the oscfilter function represents the communication channel between MATLAB and ModelSim. The following code is the function definition of the oscfilter MATLAB function.

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

Note that the function name oscfilter, differs from the entity name u_osc_filter. Therefore, the function name must be passed in explicitly to the matlabcp command (as shown above) that connects the function to the associated VHDL entity.

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

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

The following figure shows the relationship between the VHDL entity's ports and the MATLAB function's iport and oport parameters.
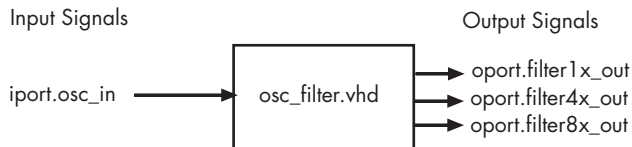
Input Signals   Output Signals

iport.osc_in → osc_filter.vhd → oport.filter1x_out
               → oport.filter4x_out
               → oport.filter8x_out

For more information on the required MATLAB function parameters, see "Setting Up Expected Parameters" on page 5-13.

Callback Scheduling

In this example, the matlabcp command invoked at the start of simulation (see above) did not specify any callback timing period. Therefore, the MATLAB component function is responsible for scheduling each subsequent callback, starting from the initial callback.

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

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

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

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

Port Data Types

All I/O ports for the osc_filter entity are defined as

```
STD_LOGIC_VECTOR(21 DOWNTO O)
```

The MATLAB component must convert input signal data from this representation to a 22-bit column or row vector of characters where each bit maps to standard logic character 0 or 1. The inverse conversion is required when outputs are returned to ModelSim.

For more information on interface data type conversions, see "Data Type Conversions" on page 5-9.

Conversion of Port Data Received from ModelSim

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

```
InDelayLine(1) = InputScale * bin2dec(iport.osc_in')/2^(Nbits-1);
```

The bin2dec function converts the binary data that the MATLAB function receives from the entity's osc_in port to unsigned decimal values that MATLAB can compute.

"Converting Data for Manipulation" on page 5-17 provides a summary of the types of data conversions to consider when coding MATLAB component or test bench functions.

Conversion of Data for Return to ModelSim

The following code excerpt illustrates data type conversion of data to be returned to ModelSim:

```
firout1 = sum(Hresp{1}.*InDelayLine);
outputvalue = dec2bin(floor(firout1*2^(Nbits-1)), Nbits);
outputvalue(find(outputvalue=='/')) = '1'; % fix negative numbers
oport.filter1x_out = outputvalue';
```

The dec2bin call converts the decimal filter output value computed by MATLAB to binary data that the MATLAB function can deposit to the entity's output port, filter1x_out. Note the special handling for negative numbers.

"Converting Data for Return to ModelSim" on page 5-18 provides a summary of the types of data conversions to consider when coding MATLAB component or test bench functions.

Placing a MATLAB Test Bench or Component 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.

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

6

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

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.
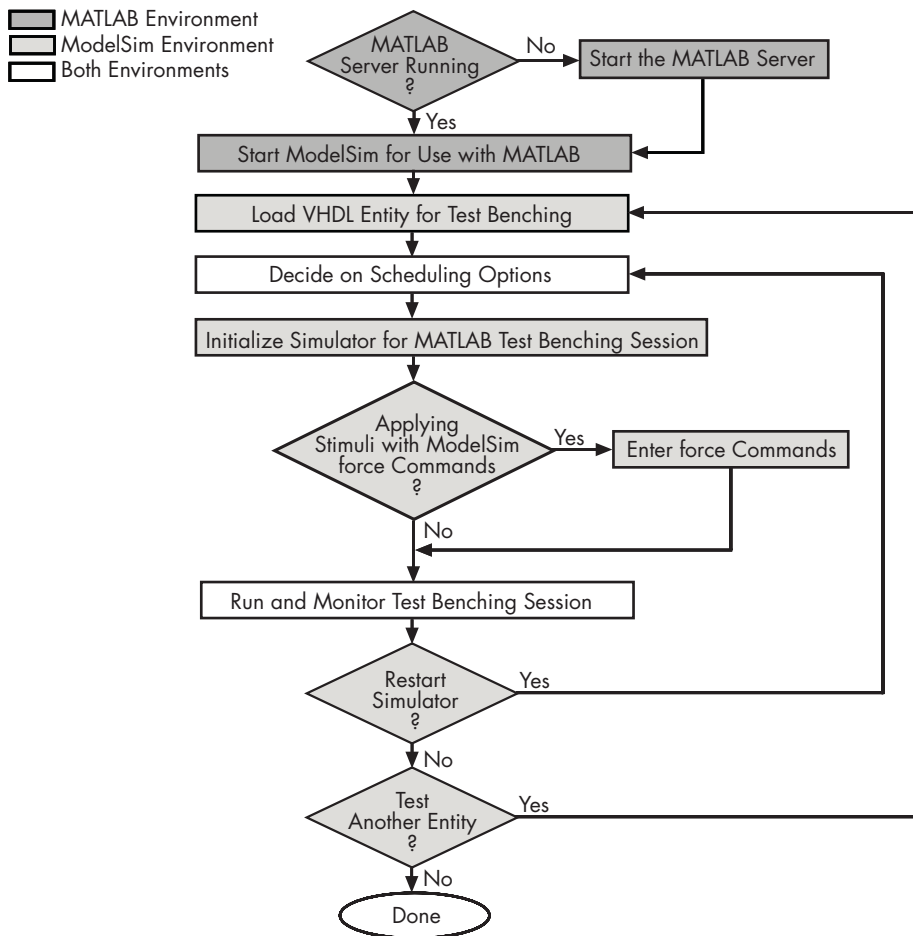
Checking the MATLAB Server's Link Status

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: O
    ipc_id: [1x45 char]
```

Starting the MATLAB Server

Start the MATLAB server as follows:

1 Start MATLAB.

2 In the MATLAB Command Window, call the `hdldaemon` function with property name/property value pairs that specify whether the 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 1-18. 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 1-18.
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 `configuremodelsim` function.

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

To...	Specify...
Include one or more Tcl commands in a DO file that executes after ModelSim launches	`'tclstart'`, `'tcl_commands'`, where `tcl_commands` is a command string or cell array of command strings, which can include the `matlabtb` and `matlabtbevalModelSim` 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...	Specify...
Start a specific version of the simulator or a 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 launch
Create a DO file for future reference or use	`'startupfile'`, `'pathname'`, where `pathname` specifies a path and filename 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 or Component 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 or Component Function

You can control the callback timing of a MATLAB test bench function by using that function's `tnext` output 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	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. `tnext = '12.2 nsec'`
String that does not specify units	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. `tnext = '1e2'`

Time Representations for tnext Parameter (Continued)

If You Specify a...	The Interface...
double value	Converts the value to seconds. For example, the following value converts to the simulation time nearest to 1 nanosecond as a multiple of the current ModelSim time resolution. tnext = 1e-9
int64 value	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. tnext=int64(100)

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 hostname or Internet address in addition to the socket port. The following table lists different ways of specifying a TCP/IP socket address.

Format	Example
Port number	`4449`
Port alias	`matlabservice`
Port number and remote hostname	`4449@compa`

Format	**Example**
Remote hostname 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 1-18.

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 1-18. 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`, `us`, 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

To Specify MATLAB Function Execution...	Include...	Where...
At explicit times	time[, ...]	time represents one of n time values, past time 0, at which the MATLAB function executes. For example: ```10 ns, 10 ms, 10 sec``` The MATLAB function executes when time equals 0 and then 10 nanoseconds, 10 milliseconds, and 10 seconds from time zero.
At a combination of explicit times and repeatedly at an interval	time[, ...] -repeat n	time represents one of n time values at which the MATLAB function executes and the n specified with -repeat represents an interval between MATLAB function executions. The interface applies the union of the two options. For example: ```5 ns -repeat 10 ns``` The MATLAB function executes at time equals 0 ns, 5 ns, 15 ns, 25 ns, and so on.

Simulation Scheduling Options (Continued)

To Specify MATLAB Function Execution...	Include...	Where...
When a specific signal experiences a rising or falling edge	-rising signal[, ...] -falling signal[, ...]	signal represents a pathname of a signal defined as a logic type — STD_LOGIC, BIT, X01, and so on.
On change of signal values (sensitivity list)	-sensitivity signal[, ...]	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. **Note** Use of this option for INOUT ports can result in double calls. If you specify the option with no signals, the interface is sensitive to value changes for all signals. For example: -sensitivity /randnumgen/dout The MATLAB function executes if the value of dout changes.

> **Note** When specifying signals with the `-rising`, `-falling`, and `-sensitivity` options, specify them in full 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/clk,
-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/clk` experiences a rising edge.

Arguments in the command line specify the following:

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

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

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

If a connection exists, the function returns the message

```
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 selecting 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	**MATLAB Environment**
Enter the `bp` command	Click next to an executable statement in the breakpoint alley of the Editor/Debugger
Select **Simulate>Break** in the **Main** window	Click the Set/Clear Breakpoint button on the toolbar
Click the Break button on the **Main** or **wave** window toolbar	Select **Set/Clear Breakpoint** on the **Breakpoints** menu
	Select **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	MATLAB Environment
Click the **Step** or **Step Over** button on the **Main** or **wave** window toolbar	Click the Step, Step In, or Step Out toolbar button
Click the **Step** or **Step-Over** options on the **Simulate>Run** menu	Select the **Step**, **Step In**, or **Step Out** option on the **Debug** menu
Enter the step command	Select 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	MATLAB Environment
Click the **Run Continue** button on the **Main** or **wave** window toolbar	Click the Continue toolbar button
Select the **Continue** option on the **Simulate>Run** menu	Select the **Continue**, **Run**, or **Save and Run** option on the **Debug** menu
Enter the run command with the -continue option	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 selecting the **Simulate>End Simulation** option on the main window.

3 Close your project by selecting 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.

7

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

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.
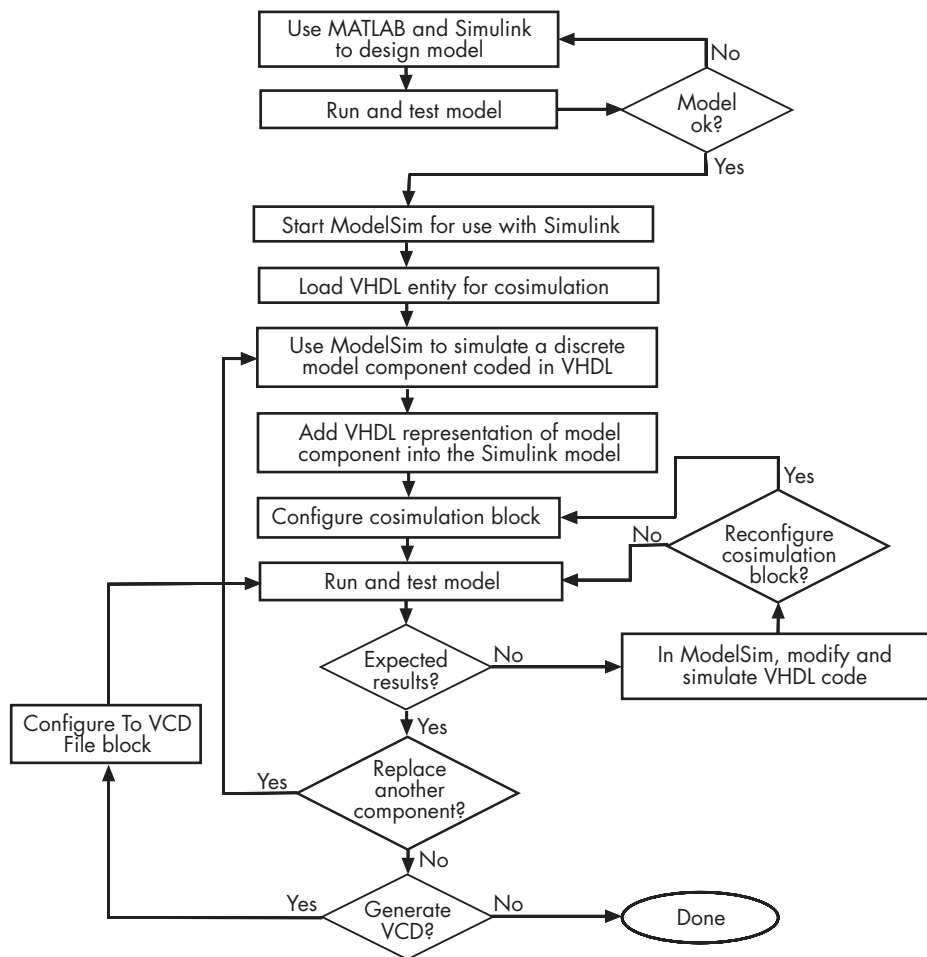
Creating a Hardware Model Design in Simulink

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 Blockset, Signal Processing Blockset, and Simulink Fixed Point.

- 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 Value Change Dump (VCD) file?

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

This figure 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 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 Multirate Signals" on page 7-19
- "Block Simulation Latency" on page 7-20

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

The representation of simulation time differs significantly between ModelSim and Simulink.

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

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

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

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

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

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

The Link for ModelSim provides controls that let you configure the timing relationship between ModelSim and Simulink and avoid timing errors caused by differences in timing representation.

Defining the Simulink and ModelSim Timing Relationship

The **Timescales** pane of the VHDL Cosimulation block parameters dialog lets you choose an optimal timing relationship between Simulink and ModelSim. The figure below shows the default settings of the **Timescales** pane.
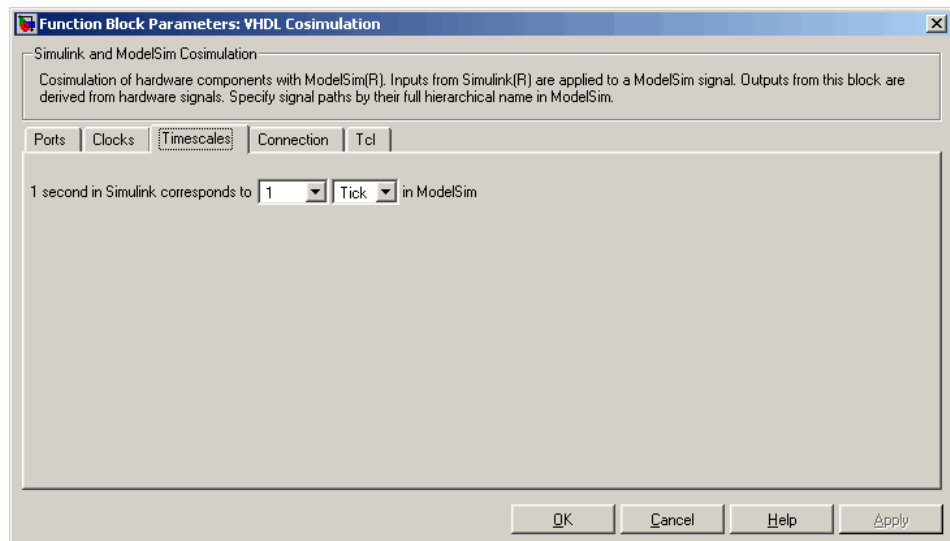
The **Timescales** pane defines a correspondence between one second of Simulink time and some quantity of ModelSim time. This quantity of ModelSim time can be expressed in one of the following ways:

- In *relative* terms (i.e., as some number of ModelSim ticks). In this case, the cosimulation is said to operate in *relative timing mode*. Relative timing mode is the default.

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

The following sections discuss these two timing modes.

Relative Timing Mode

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

• *One second* in Simulink corresponds to *N ticks* in ModelSim, where `N` is a scale factor.

This correspondence holds regardless of the ModelSim timing resolution.

In relative timing mode, all sample times and clock periods in Simulink are rounded to the nearest integer number of seconds so that they can be directly translated into ticks in ModelSim. The following pseudocode shows how Simulink time units are quantized to ModelSim ticks:

```
qtInTicks = round(N * tInSecs)
```

where `qtInTicks` is the ModelSim time in ticks, `tInSecs` is the Simulink time in seconds, and `N` is a scale factor.

To configure relative timing mode for a cosimulation:

1 Click the **Timescales** tab of the VHDL Cosimulation block parameters dialog.

2 Select `Tick` from the list on the right. (This is the default.)

3 Select a scale factor from the list on the left. Available scale factors are 1, 10, 100, and 1000. The default scale factor is 1.

For example, in the figure below, the **Timescales** pane is configured for a relative timing correspondence of 10 ModelSim ticks to 1 Simulink second.
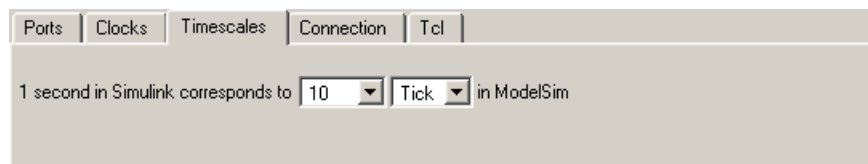
4 Click **Apply** to commit your changes.

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

- If the total simulation time in Simulink is specified as N seconds, then the ModelSim VHDL simulation will run for exactly N ticks (i.e., N ns at the default resolution limit).

- Similarly, if Simulink computes the sample time of a VHDL Cosimulation block input port as *Tsi* seconds, new values will be deposited on the VHDL input port at exact multiples of *Tsi* ticks. If an output port has an explicitly specified sample time of *Tso* seconds, values will be read from ModelSim at multiples of *Tso* ticks.

- Clocks operate in a similar fashion. Where a clock has a period of *T* seconds:
  - If *T* is even, the clock signal is forced in ModelSim as an input signal that stays low for *T/2* ticks and stays high for *T/2* ticks.
  - If *T* is odd, the clock signal is forced in ModelSim as an input signal that stays low for *T/2* ticks and stays high for *(T/2) + 1* ticks.

  Note that Simulink requires such clocks to have a period of at least 2 seconds. Simulink throws an error if specified value of *T* is less than 2 seconds.

To understand how relative timing mode operates, we will look at cosimulation results from the example model below.
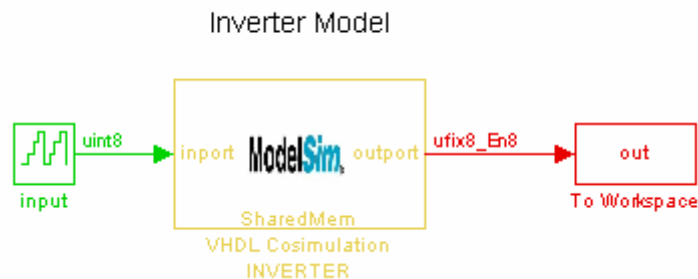
The model contains a VHDL Cosimulation block simulating an 8-bit inverter that is enabled by an explicit clock. The inverter has a single input and a single output. The VHDL code for the inverter is listed below:

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

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

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

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

Consider a cosimulation of this model with the following settings:

- Simulation parameters in Simulink

  - **Timescales** parameters: default (relative timing with a scale factor of 1)

  - Total simulation time: 60 s

  - Input port (/inverter/inport) sample time: 24 s

  - Output port (/inverter/outport) sample time: 12 s

  - Clock (inverter/clk) period: 10 s

- ModelSim resolution limit: 1 ns

The figure below shows the ModelSim **wave** window after a cosimulation run of the example Simulink model for 60 ns. The **wave** window shows that ModelSim simulated for 60 ticks (60 ns). The inputs change at multiples of 24 ns and the outputs are read from ModelSim at multiples of 12 ns. The clock is driven low and high at intervals of 5 ns.
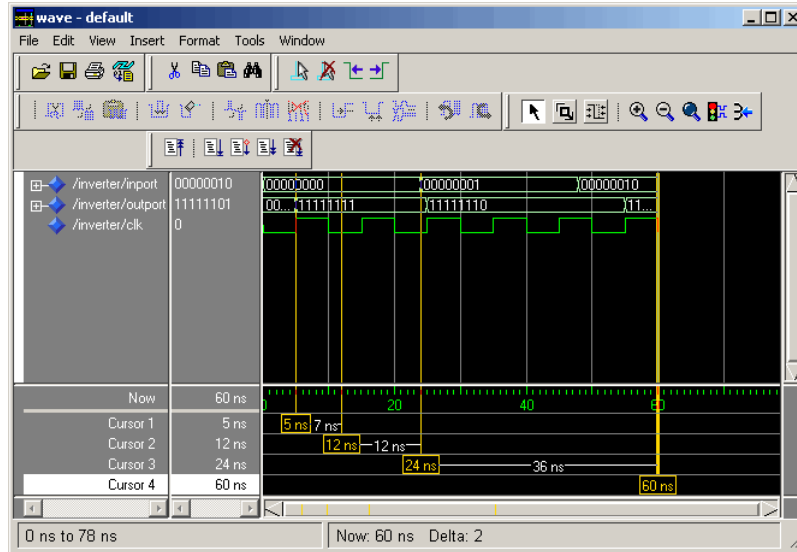
Now consider a cosimulation of the same model, this time configured with a scale factor of 100 in the **Timescales** pane.

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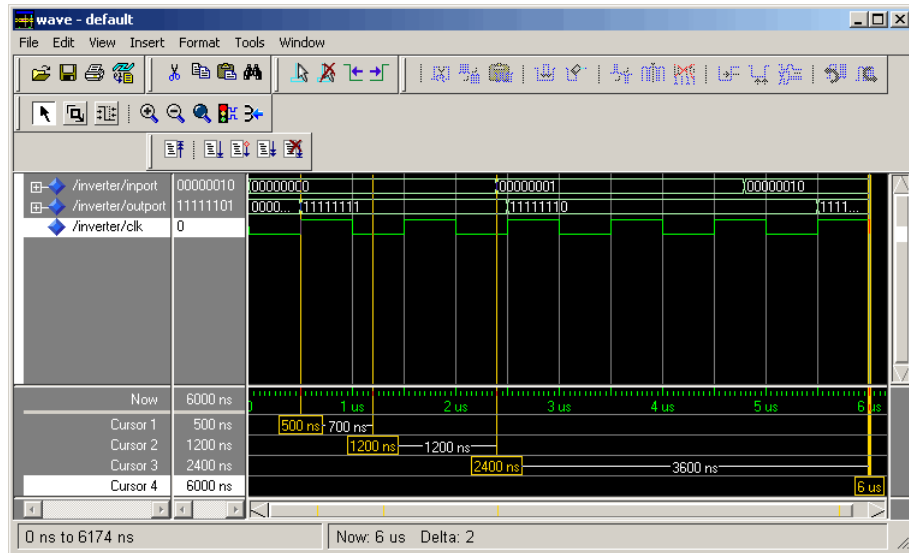
Absolute Timing Mode

In the example model discussed in the previous section, all sample times were specified as integers. In real-world applications, this may not be practical.

In relative timing mode, all Simulink times are quantized to ModelSim ticks by rounding the Simulink times to the nearest integer. This can be a serious limitation for models that use physical time constants (small real number values) as sample times. To cosimulate such models with ModelSim in relative timing mode, you must rescale sample times throughout the model to integer values. Furthermore, use of nonuniform time units between Simulink and ModelSim (e.g., seconds in Simulink vs. ticks in ModelSim) can be an impediment to advanced users.

To overcome these limitations, the Link for ModelSim provides an absolute timing mode. Absolute timing mode lets you define the timing relationship between Simulink and ModelSim in terms of absolute time units and a scale factor:

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

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

To configure absolute timing mode for a cosimulation:

1 Select the **Timescales** tab of the VHDL Cosimulation block parameters dialog.

2 Select a unit of absolute time from the list on the right. Available units are fs, ps, ns, us, ms, and s.

3 Select a scale factor from the list on the left. Available scale factors are 1, 10, 100, and 1000. The default scale factor is 1.

For example, in the figure below, the **Timescales** pane is configured for an absolute timing correspondence of 1 ModelSim second to 1 Simulink second.

| Ports | Clocks | Timescales | Connection | Tcl |

1 second in Simulink corresponds to 1 ▼ s ▼ in ModelSim
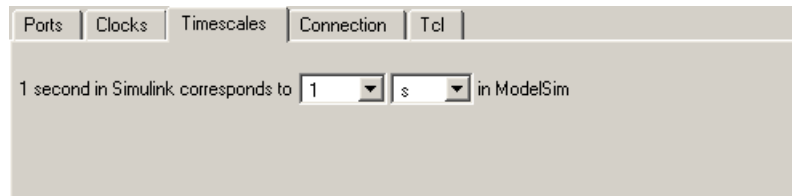
4 Click **Apply** to commit your changes.

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

```
qtInTicks = round( tInSecs * (tScale / tRL))
```

where

- qtInTicks is the ModelSim time in ticks.

- tInSecs is the Simulink time in seconds.

- tScale is the timescale setting (unit and scale factor) chosen in the **Timescales** pane of the VHDL Cosimulation block.

- tRL is the ModelSim resolution limit.

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

```
qtInTicks = round(12ns * (1s / 1ns)) = 12
```

Operation of Absolute Timing Mode. To understand the operation of absolute timing mode, we will again consider the example model discussed in "Relative Timing Mode" on page 7-11. Suppose that the model is reconfigured as follows:

- Simulation parameters in Simulink

  - **Timescale** parameters: `1 s` of Simulink time corresponds to `1 s` of ModelSim time.

  - Total simulation time: `60e-9 s (60ns)`

  - Input port (`/inverter/inport`) sample time: `24e-9 s (24 ns)`

  - Output port (`/inverter/outport`) sample time: `12e-9 s (12 ns)`

  - Clock (`inverter/clk`) period: `10e-9 s (10 ns)`

- ModelSim resolution limit: `1 ns`

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

The figure below shows the ModelSim **wave** window after a cosimulation run.
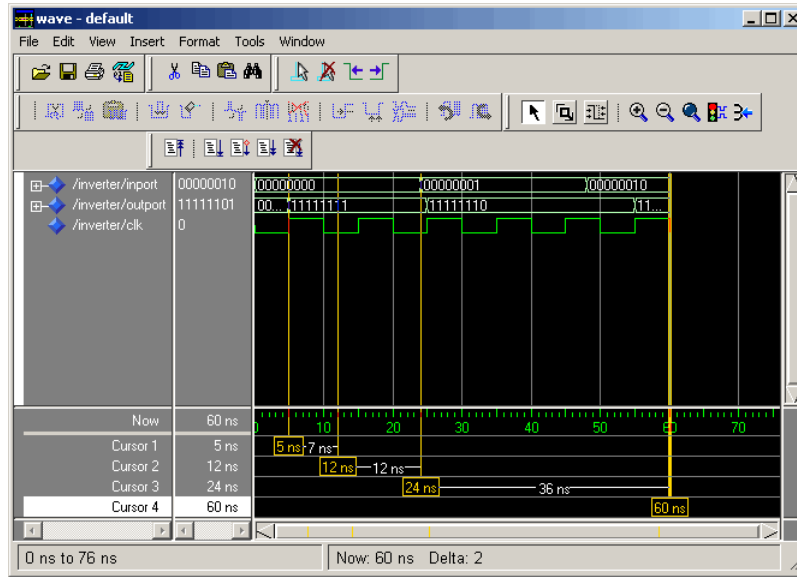
Timing Mode Usage Restrictions

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

- Mixing of timing modes in the same Simulink model is disallowed. All VHDL Cosimulation blocks in the model must be configured either in relative timing mode or in absolute timing mode.

- When multiple VHDL Cosimulation blocks in a model are communicating with a single instance of ModelSim, all VHDL Cosimulation blocks must have the same **Timescales** pane settings.

- If you change the **Timescales** pane settings in a VHDL Cosimulation block between consecutive cosimulation runs, you must restart simulation in ModelSim.

Setting VHDL Cosimulation Port Sample Times

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

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

- If an input port is connected to a signal that *does not have* an explicit sample time, Simulink assigns a sample time that is equal to the least common multiple (LCM) of all identified input port sample times for the model.

- After Simulink sets the input port sample periods, it applies user-specified output sample times to all output ports. Sample times must be explicitly defined for all output ports.

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

- Specify the output sample time for a VHDL 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 1 ns and you specify a block's output sample time as 20, Simulink interacts with ModelSim every 2 ns.

- Specify the Simulink model's start and stop time values (see the **Solver** pane of the Simulink Configuration Parameters dialog) as integers. To calculate the actual simulation start and stop times, the Link for ModelSim multiplies the Simulink total simulation time by the number of ticks of the ModelSim resolution limit.

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

Handling Multirate Signals

The 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 VHDL Cosimulation block exchanges data for each signal at the Simulink sample rate for that signal. For input signals, a VHDL Cosimulation block accepts and honors all signal rates.

The VHDL Cosimulation block also lets you specify an independent sample time for each output port. You must explicitly set the sample time for each output port, or accept the default. This lets you control the rate at which

Simulink updates an output port by reading the corresponding signal from ModelSim.

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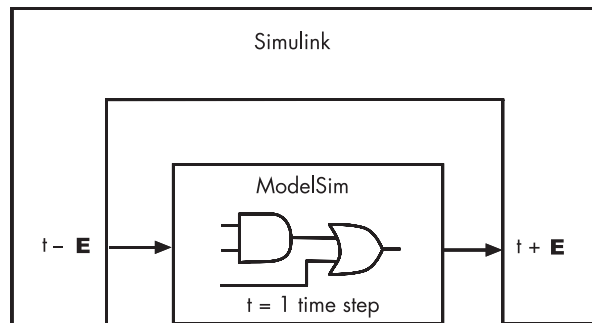
As the figure shows, Simulink cosimulation block input affects signal values just after the current ModelSim time step (t+E) and block output reflects signal values just before the current ModelSim step time (t-E).

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 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;
```
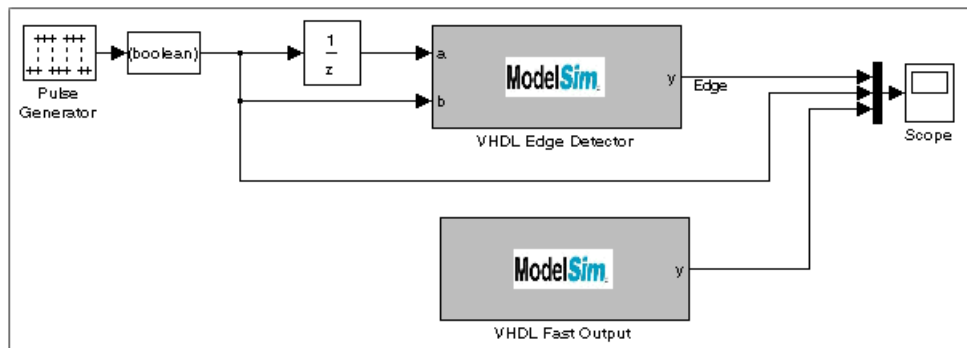

In the Simulink model, the cosimulation block VHDL Edge Detector contains an XOR circuit. The second cosimulation block, VHDL Fast Output, simply reads the same XOR output. The first block is driven by a signal generated by the Pulse Generator block. The Data Type Conversion block converts the signal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

- A Unit Delay block applies a sample and hold to the signal and drives block input port a. The delay is equal to one period of the signal's Simulink sample time. When the delay is applied to the XOR, the pulse equals the period specified by the delay block after any edges.

- The signal without a delay drives block input port b.

- The third signal bypasses the cosimulation block and goes directly to the Scope block for display.

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

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

Block	Sample Time	Value
Pulse Generator	Sample time	100
Data Type Conversion block	Sample time	Inherited from Pulse Generator block
Unit Delay block	Sample time	Inherited from Data Type Conversion block
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.
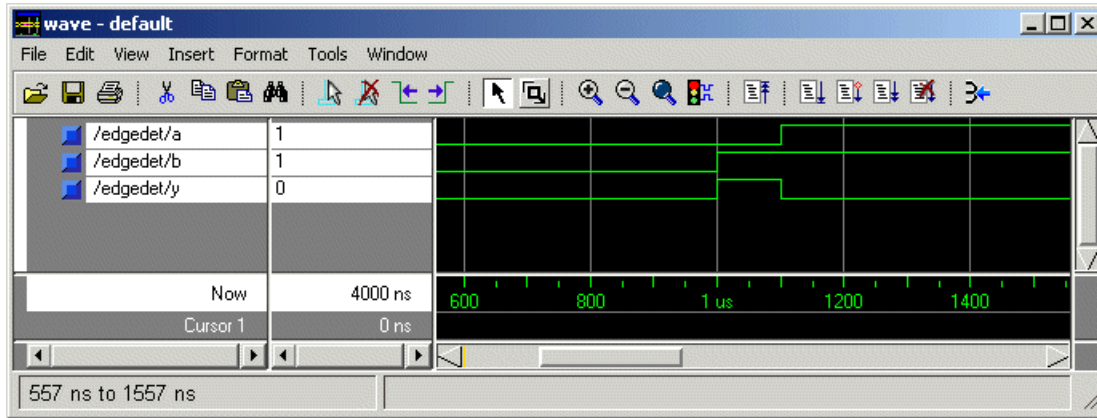
Note the following:

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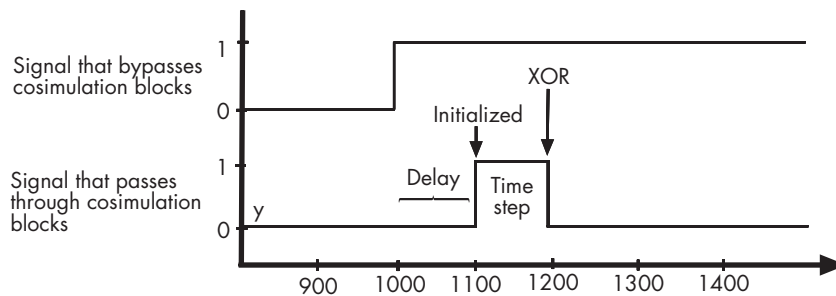
The signal that bypasses the cosimulation blocks rises at t=1000. That signal stays high for the duration of the sample period. However, the signals that are read from output port y of the two cosimulation blocks, display in the Scope window as follows:

7-23

- 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.
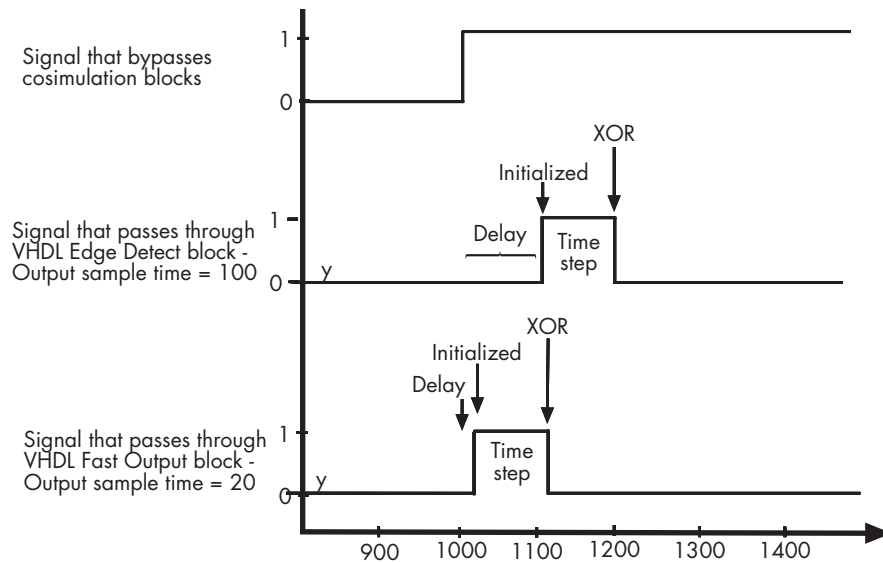
In this case, the signal that bypasses the cosimulation blocks and the output signal read from the VHDL Edge Detect block remain the same. However, the delay for the signal read from the VHDL Fast Output block is 20 ticks instead of 100. Although the size of the time step is still tied to the ModelSim 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 **Solver** pane of the Configuration Parameters dialog box.

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

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

Running and Testing a Hardware Model in Simulink

If you take the approach of designing a Simulink model first, run and test your model thoroughly before replacing or adding hardware model components as 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 `configuremodelsim`. To check on current settings, search for and browse through the contents of the file `\tcl\ModelSimTclFunctionsForMATLAB.tcl` in your ModelSim installation path. Any options that you explicitly specify with the MATLAB `vsim` function override these default settings.

To...	Specify...
Include one or more Tcl commands in the DO file that are to execute during ModelSim startup	`'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...	Specify...
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 1-18.
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 `configuremodelsim` function.

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

The 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 the block icons shown below.
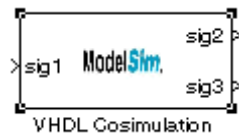
	VHDL Cosimulation	Block that has at least one input port and one output port.
	To VCD File	Generates a Value Change Dump (VCD) file. For information on using this block, see "Using a Value Change Dump File for Design Verification" on page 7-54.
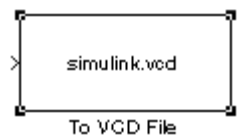
4 Copy the VHDL Cosimulation block icon from the Library Browser to your model. Simulink creates a link to the block at the point where you drop the block icon.

5 Connect any VHDL block ports to appropriate blocks in your Simulink model. To model a sink device, configure the block with inputs only. To model a source device, configure the block with outputs only.

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

- **Connection**: Type of communication and communication settings to be used for exchanging data between simulators

- **Timescales**: Timing relationship between Simulink and ModelSim

- **Clocks**: Rising-edge and falling-edge clocks to apply to your model

- **Tcl**: Tcl commands to run before and after a simulation

The following sections help you identify what you need to configure, how to open the Block Parameters 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...
Ports	
☐ 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 7-37
☐ 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 7-37

VHDL Cosimulation Block Requirements Checklist (Continued)

Requirement	For More Information, See...
☐ If the block is modeling an input and output device, do you want to specify explicit sample times for output ports?	"Mapping VHDL Signals to Block Ports" on page 7-37
☐ If the block is modeling an input and output device, do you want to specify explicit fixed point data types for output ports? By default the data types are either inherited from the signals connected to the VHDL Cosimulation block output ports or derived from the VHDL model.	"Specifying Data Types for Output Ports" on page 7-41
☐ If the block is 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 7-37
Timing	
☐ What is the optimal timing relationship between Simulink and ModelSim for your cosimulation?	"Representation of Simulation Time" on page 7-9
☐ Do you need to specify a relative (Simulink seconds corresponding to ModelSim ticks) timing relationship between Simulink and ModelSim?	"Configuring the Simulink and ModelSim Timing Relationship" on page 7-43
☐ Do you need to specify an absolute (Simulink seconds corresponding to ModelSim absolute time units) timing relationship between Simulink and ModelSim?	"Configuring the Simulink and ModelSim Timing Relationship" on page 7-43
Communication	
☐ Is it critical that communication performance be as optimal as possible?	"Configuring the Communication Link" on page 7-44
☐ Are you running ModelSim and Simulink on the same computer?	"Configuring the Communication Link" on page 7-44
☐ If ModelSim and Simulink are running on the same computer, do you want to use shared memory communication?	"Configuring the Communication Link" on page 7-44

VHDL Cosimulation Block Requirements Checklist (Continued)

Requirement	For More Information, See...
☐ 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 7-44
☐ 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 7-44
Clocks	
☐ Do you want to create a rising-edge clock to apply stimuli to your cosimulation model?	"Creating Optional Clocks" on page 7-46
☐ Do you want to create a falling-edge clock to apply stimuli to your cosimulation model?	"Creating Optional Clocks" on page 7-46
☐ Do you want to specify the period for rising/falling edge clocks specified in the model?	"Creating Optional Clocks" on page 7-46
Tcl	
☐ Are there any Tcl commands that you want ModelSim to execute before running a simulation, but after loading the project in ModelSim?	"Executing Tcl Commands Before and After Cosimulation" on page 7-49
☐ Are there any Tcl commands that you want ModelSim to execute after running a simulation?	"Executing Tcl Commands Before and After Cosimulation" on page 7-49

Opening the Block Parameters Dialog

To open the block parameters dialog for the VHDL Cosimulation block, double-click the block icon.

VHDL Cosimulation

Simulink displays the following Block Parameters dialog.
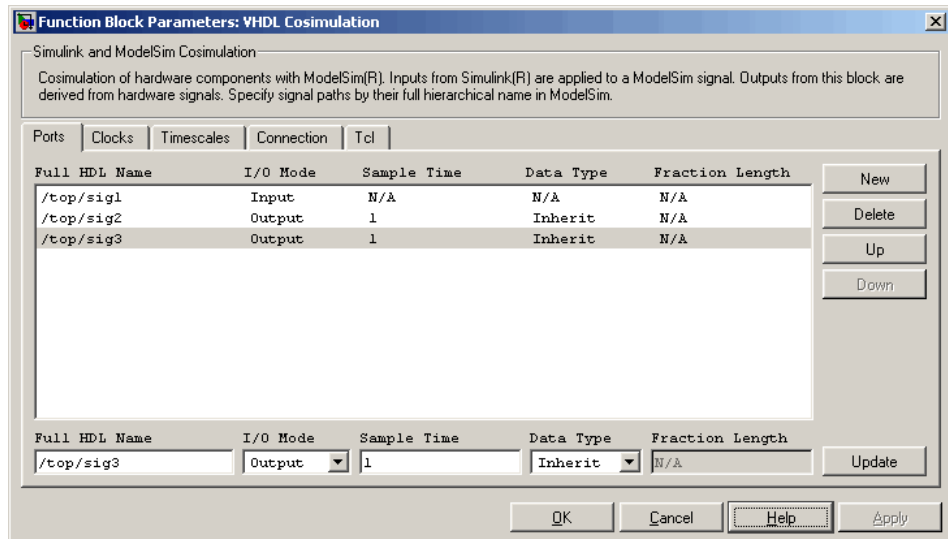
Mapping VHDL Signals to Block Ports

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 VHDL Cosimulation block. In addition to identifying input and output ports, you can specify a sample time for each output port. You can also specify a fixed-point data type for each output port.

The signals that you map can be at any level of the 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.
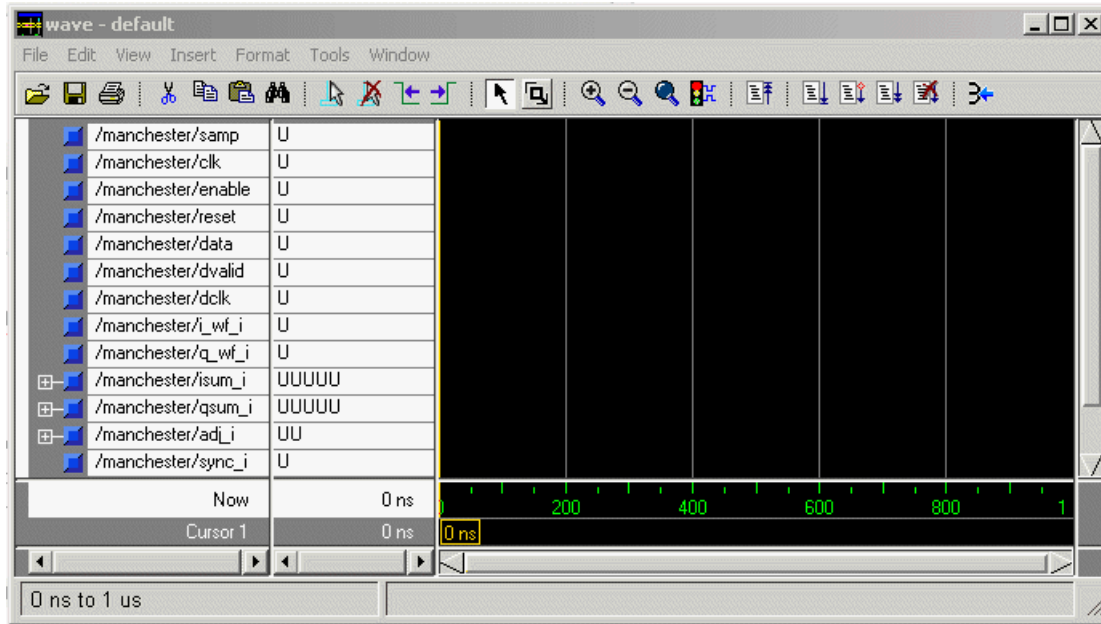
2 In Simulink, open the block parameters dialog for your VHDL Cosimulation block, if it is not already open.

3 Select the **Ports** tab of the Block Parameters dialog. Simulink displays the dialog as shown below.
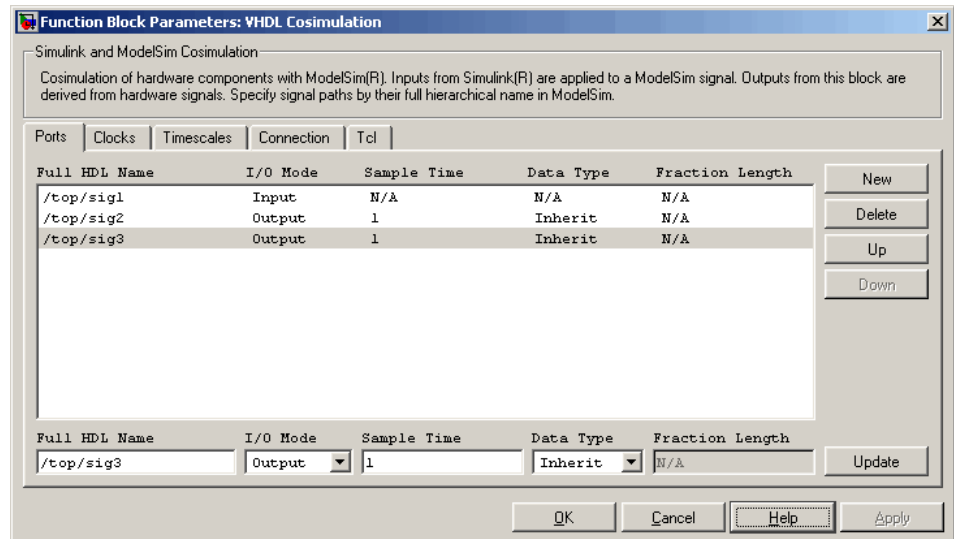
In this pane, you define the VHDL signals of your design that you want to include in your Simulink block and set a sample time and data type for output ports. The parameters that you should specify on the **Ports** pane depend on the type of device the block is modeling.

For...	Specify...
An input and output device	Block input ports, block output ports, output sample times and output data types. For output ports, accept the default or enter an explicit sample time. Data types can be specified explicitly, or set to Inherit (the default). In the default case, the output port data type is inherited either from the signal connected to the port, or derived from the VHDL model.
A sink device	Block input ports
A source device	Block output ports, output sample times, and output data types. For output ports, accept the default or enter an explicit sample time. You can specify explicit fixed-point data types per port. For example, specifying an output port data type as Unsigned with a fraction length F forces the output port data type to be ufixN_EnF, where N is the size (in bits) of the output port signal in the VHDL model, and F is the size (in bits) of the fractional part of the signal.

4 Enter test signal pathnames in the **Full HDL name** text field, using ModelSim pathname syntax. Select either Input or Output from the **I/O Mode** menu. If desired, set the **Data Type** and **Fraction Length** parameters for signals explicitly, as discussed below.

Note After entering signal parameters, click **Update** to enter your changes into the signal list.

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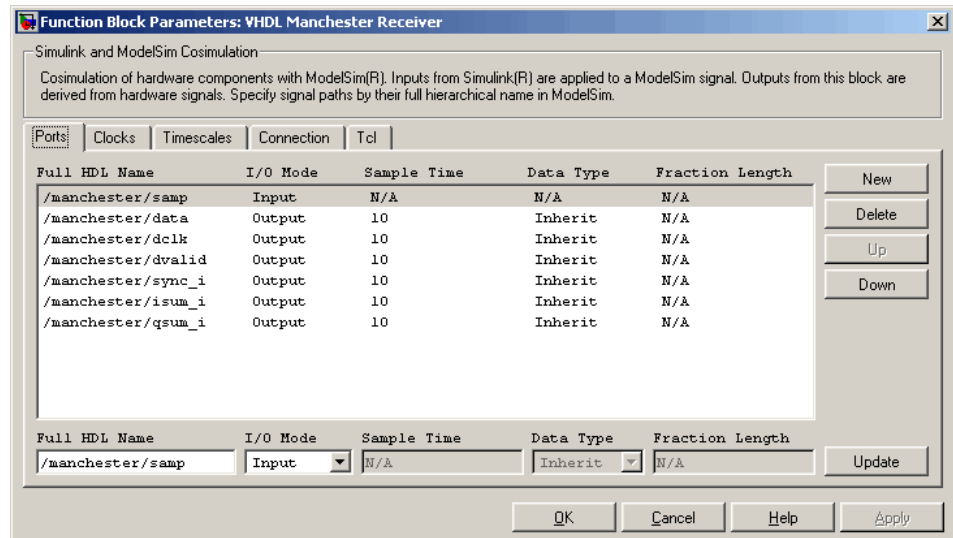
Note When you define an input port, make sure that only one source is set up to force input to that port. For example, you should avoid defining an input port that has multiple instances. If multiple sources drive a signal, your Simulink model may produce unpredictable results.

---

5 You must specify a sample time for the output ports. Output sample times are specified as integers. Simulink uses the value that you specify, and the current settings of the **Timescales** pane, to calculate an actual simulation sample time.

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

6 You can configure the fixed-point data type of each output port explicitly if desired, or use a default (`Inherited`). In the default case, Simulink determines the data type for an output port as follows:

If Simulink can determine the data type of the signal connected to the output port, it applies that data type to the output port. For example, the data type of a connected Signal Specification block is known by back-propagation. Otherwise, Simulink queries ModelSim to determine the data type of the signal from the VHDL model.

To assign an explicit fixed-point data type to a signal:

a Select either `Signed` or `Unsigned` from the **Data Type** menu.

b If the signal has a fractional part, enter the **Fraction Length**.

For example, an 8–bit signal with `Signed` data type and a **Fraction Length** of 5 is assigned the data type `sfix8_En5`. An `Unsigned` 16–bit signal with no fractional part (a **Fraction Length** of 0) is assigned the data type `ufix16`.

7 Before closing the dialog, be sure to click **Apply** to register your edits.

Specifying Data Types for Output Ports

The **Data Type** and **Fraction Length** parameters apply only to output signals.

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

The **Fraction Length** property specifies the size, in bits, of the fractional part of the signal in fixed-point representation. The **Fraction Length** property is enabled when the signal **Data Type** property is not set to Inherit.

Output port data types are determined by the signal width and by the **Data Type** and **Fraction Length** properties of the signal. To assign a port data type, set the **Data Type** and **Fraction Length** properties as follows:

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

  Inherit is the default setting. When Inherit is selected, the **Fraction Length** edit field is disabled.

  Simulink attempts to compute the data type of the signal connected to the output port by backward propagation. For example, if a Signal Specification block is connected to an output, Simulink will force the data type specified by Signal Specification block on the output port.

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

  Note that **Data Type** and **Fraction Length** properties apply only to VHDL output signals of type STD_LOGIC or STD_LOGIC_VECTOR.

- Select Signed from the **Data Type** list if you want to explicitly assign a signed fixed-point data type. When Signed is selected, the **Fraction Length** edit field is enabled. The port is assigned a fixed point type sfixN_EnF, where N is the signal width and F is the **Fraction Length**.

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

- Select Unsigned from the **Data Type** list if you want to explicitly assign an unsigned fixed point data type. When Unsigned is selected, the **Fraction Length** edit field is enabled. The port is assigned a fixed point type ufixN_EnF, where N is the signal width and F is the **Fraction Length** value.

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

Configuring the Simulink and ModelSim Timing Relationship

You configure the timing relationship between Simulink and ModelSim by using the **Timescales** pane of the block parameters dialog. Before setting the **Timescales** parameters, you should read "Representation of Simulation Time" on page 7-9 to understand the supported timing modes and the issues that will determine your choice of timing mode.

You can specify either a relative or an absolute timing relationship between Simulink and ModelSim, as described in the sections below.

Specifying a Relative Timing Relationship

To configure relative timing mode for a cosimulation:

1 Select the **Timescales** tab of the VHDL Cosimulation block parameters dialog.

2 Select Tick from the list on the right. (This is the default.)

3 Select a scale factor from the list on the left. Available scale factors are 1, 10, 100, and 1000. The default scale factor is 1.

For example, in the figure below, the **Timescales** pane is configured for a relative timing correspondence of 10 ModelSim ticks to 1 Simulink second.
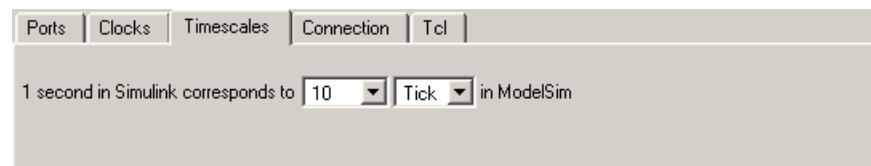
4 Click **Apply** to commit your changes.

Specifying an Absolute Timing Relationship

To configure absolute timing mode for a cosimulation:

1 Select the **Timescales** tab of the VHDL Cosimulation block parameters dialog.

2 Select a unit of absolute time from the list on the right. Available units are fs, ps, ns, us, ms, and s.

3 Select a scale factor from the list on the left. Available scale factors are 1, 10, 100, and 1000. The default scale factor is 1.

For example, in the figure below, the **Timescales** pane is configured for an absolute timing correspondence of 1 ModelSim second to 1 Simulink second.
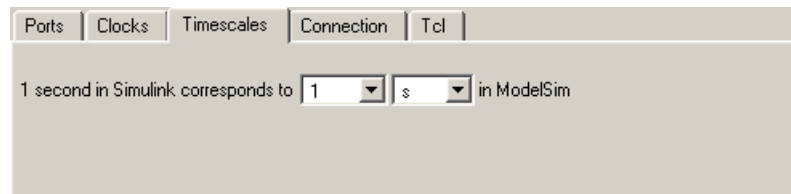
4 Click **Apply** to commit your changes.

Configuring the Communication Link

Configure a block's communication link with the **Connection** pane 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.) Note that since Simulink and ModelSim are running on different computer, **Connection method** is automatically set to Socket.

3 Enter the hostname of the computer that is running your VHDL simulation in ModelSim in the **Host name** text field. In the **Port number or service** text field, specify a valid port number or service for your computer system. For information on choosing TCP/IP socket ports, see "Choosing TCP/IP Socket Ports" on page 1-18. Skip to step 5.

4 If ModelSim and Simulink are running on the same computer, decide whether you are going to use shared memory or TCP/IP sockets for the communication channel. For information on the different modes of communication, see "Modes of Communication" on page 1-8.

If you choose TCP/IP socket communication, specify a valid port number or service for your computer system in the **Port number or service** text field. For information on choosing TCP/IP socket ports, see "Choosing TCP/IP Socket Ports" on page 1-18.

If you choose shared memory communication, select the **Shared memory** check box.

5 Click **Apply**.
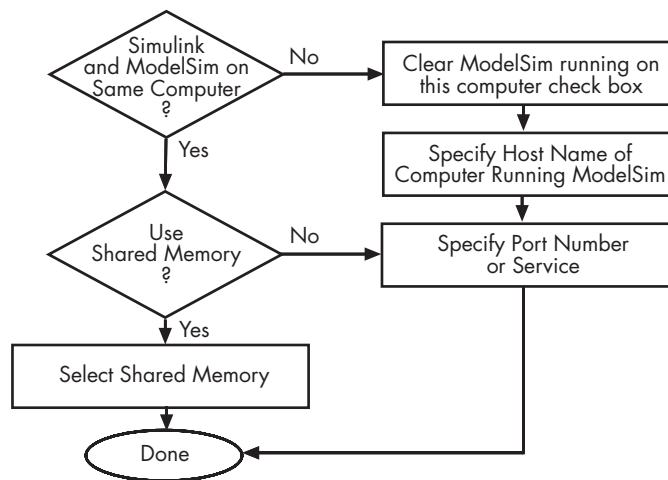
The following example dialog display shows communication definitions for a VHDL Cosimulation block. The block is configured for Simulink and ModelSim running on the same computer, communicating in TCP/IP socket mode over TCP/IP port 4449.
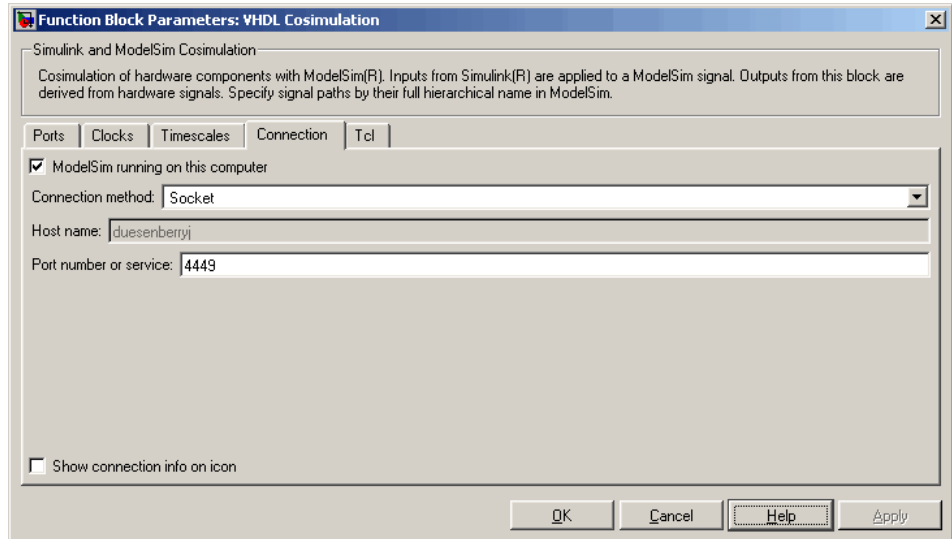
Creating Optional Clocks

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 odd Simulink sample times, with a worst case duty cycle of 66% for a sample time of T=3.

The following figure shows a timing diagram that includes rising and falling edge clocks with a Simulink sample time of T=10 and a ModelSim resolution limit of 1 ns. The figure also shows that given those timing parameters, the clock duty cycle is 50%.
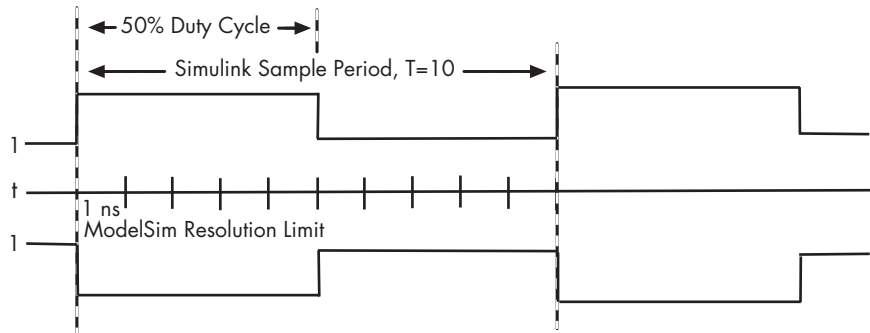
Rising Edge Clock

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 7-37.

2 Select the **Clocks** tab of the Block Parameters dialog. Simulink displays the dialog as shown below.
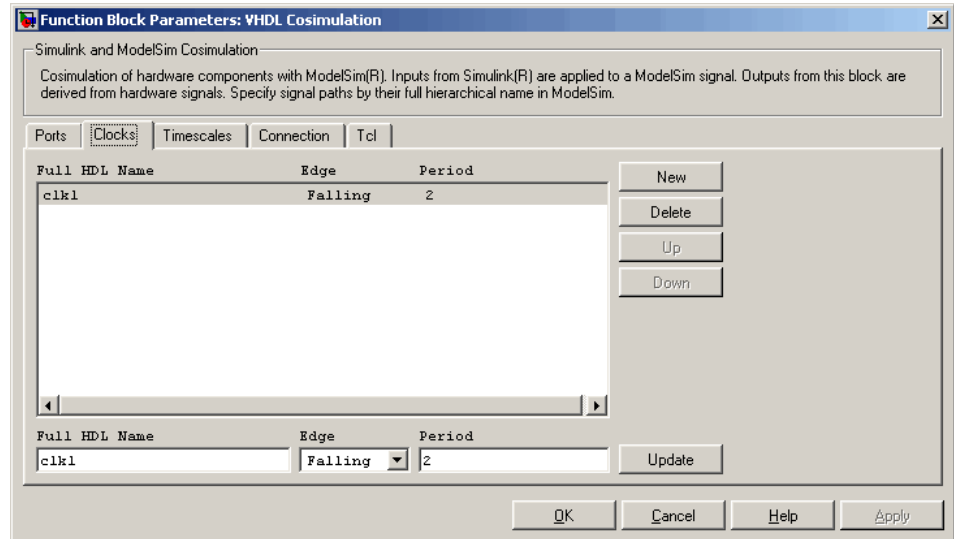
3 Click the **New** button to add a new clock signal.

4 Enter the clock signal pathname in the **Full HDL Name** text field, using ModelSim pathname syntax.

5 To specify whether the clock generates a rising-edge or falling edge signal, select Rising or Falling from the **Edge** list.

6 The **Period** field specifies the clock period. Accept the default (2), or override it by entering the desired clock period explicitly in the **Period** field.

   Specify the **Period** field as an even integer, with a minimum value of 2.

7 After entering the desired property values, click **Update**. This enters the signal values into the signal list in the center of the **Clocks** pane.

8 When you have finished editing clock signals, click **Apply** to register your changes with Simulink.

The following dialog display defines the rising-edge clock clk for the VHDL Cosimulation block. The clock period is set to the default value, directing Simulink to determine the period.
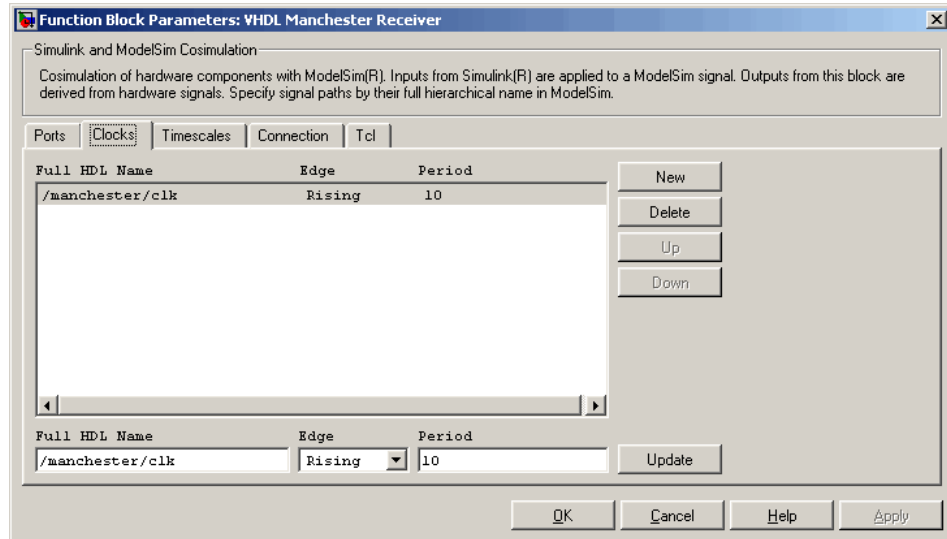
Executing Tcl Commands Before and After Cosimulation

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 Select the **Tcl** tab of the Block Parameters dialog. The dialog display appears as follows.
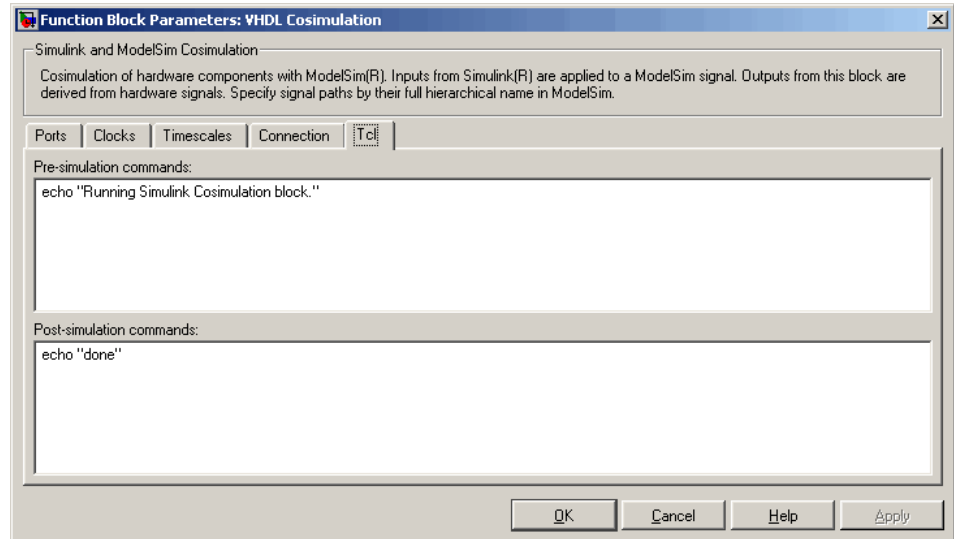
The **Pre–simulation commands** text box includes an echo command for reference purposes.

2 Enter one or more commands in the **Pre–simulation command** and **Post–simulation command** text boxes. You can specify one Tcl command per line in the text box, or enter multiple commands per line by appending each command with a semicolon (;), the standard Tcl concatenation operator.

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.
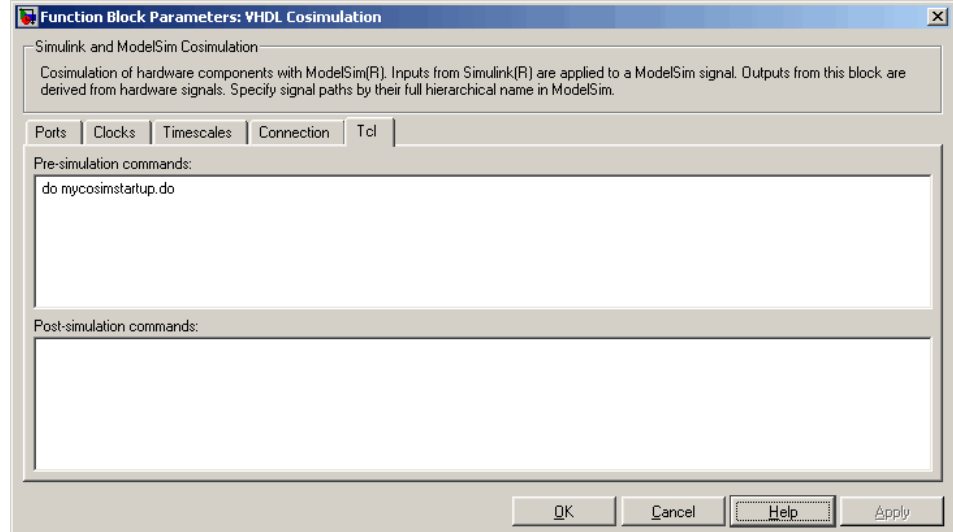
3 Click **Apply**.

Applying Your Block Parameters Configuration Settings

After you enter your block parameters settings,

1 Review the content of each dialog pane.

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:
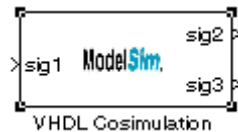
After Configuration:
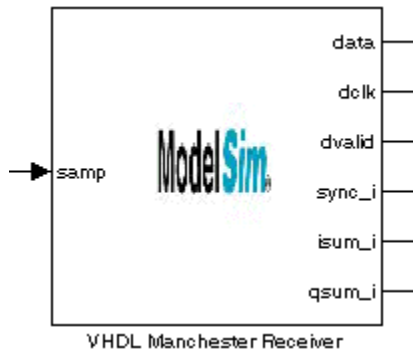
To verify the connection with ModelSim and the signal names, select **Edit—>Update diagram** or press **Ctrl+D**.

Running and Testing a Cosimulation Model in Simulink

To run and test a cosimulation model in Simulink, click **Simulation–>Start**

or the Start Simulation button ▶ in your Simulink model window. Simulink
runs the model and displays any errors that it detects.

If you need to reset a clock during a cosimulation, you can do so by entering
ModelSim `force` commands at the ModelSim command prompt or by
specifying ModelSim `force` commands in the **After simulation command**
text field on the **Tcl** pane 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 Wave Log Format (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 7-55

- "VCD File Format" on page 7-57

- "A Sample VCD File Application " on page 7-59

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 Double-click the block icon. Simulink displays the following dialog.
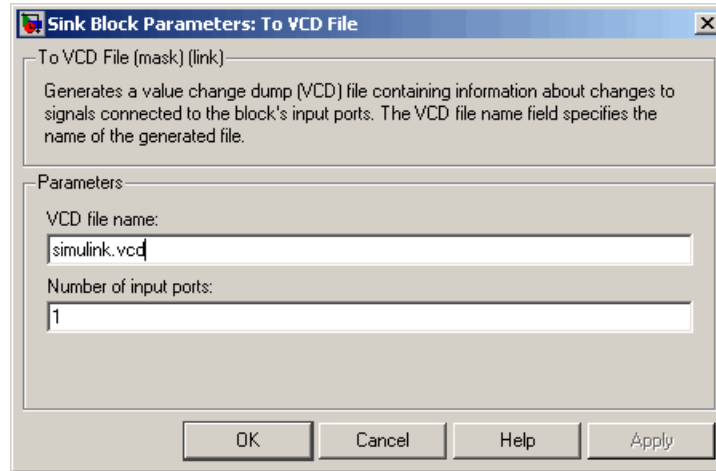
b Specify a filename for the generated VCD file in the **VCD file name** text box. If you specify a filename only, Simulink places the file in your current MATLAB directory. Specify a complete pathname to place the generated file in a different location. If you specify the same name for multiple To VCD 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 7-57. For a sample application of a VCD file, see "A Sample VCD File Application " on page 7-59.

VCD File Format

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

File Content	Description
``` $date 23-Sep-2003 14:38:11 $end ```	Data and time the file was generated.
``` $version Link for ModelSim version 1.0 $ end ```	Version of the VCD block that generated the file.
``` $timescale 1 ns $ end ```	The time scale that was used during the simulation.
``` $scope module manchestermodel $end ```	The scope of the module being dumped.

File Content	Description
``` $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 ```	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.
``` $upscope $end ```	Marks a change to the next higher level in the HDL design hierarchy.
``` $enddefinitions $end ```	Marks the end of the header and definitions section.
``` #0 ```	Simulation start time.
``` $dumpvars 0! 0" 0# 0$ $end ```	Lists the values of all defined variables at time equals 0.

File Content	Description
``` #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 vcd2wlf tool, which converts a VCD file to a WLF file that you can then view in a ModelSim **wave** window. This section shows how you might apply the vcd2wlf tool:

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, select **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.
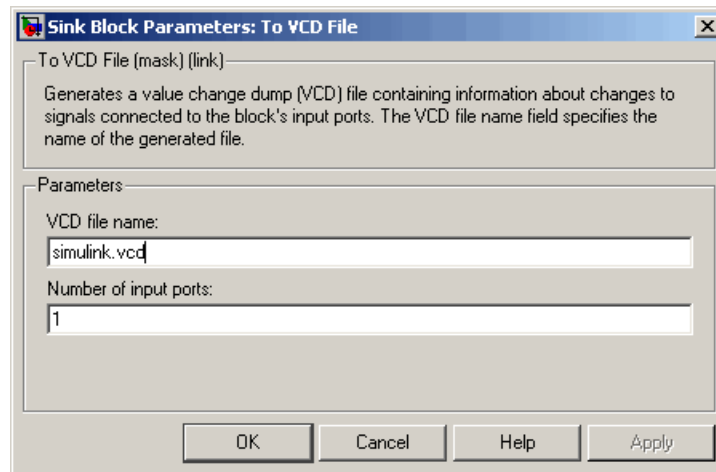
e Type `MyVCDfile.vcd` in the **VCD file name** text box.

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.
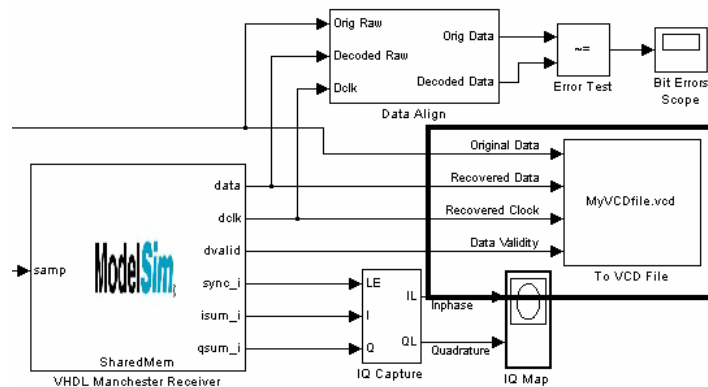
6 Save the model.

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 select the **Connection** tab in the Block Parameters dialog.

---

9 Start the simulation from the Simulink model window.

10 When the simulation is complete, locate, open, and browse through the generated VCD file, MyVCDfile.vcd.

11 Close the VCD file.

12 Change your input focus to ModelSim and end the simulation.

13 Change the current directory to the directory containing the VCD file and enter the following command at the ModelSim command prompt:

```
vcd2wlf MyVCDfile.vcd MyVCDfile.wlf
```

The `vcd2wlf` utility converts the VCD file to a WLF file that you display with the command `vsim -view`.

14 In ModelSim, open the wave file `MyVCDfile.wlf` as 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 button to view the signal data. The **wave** window should appear as follows.
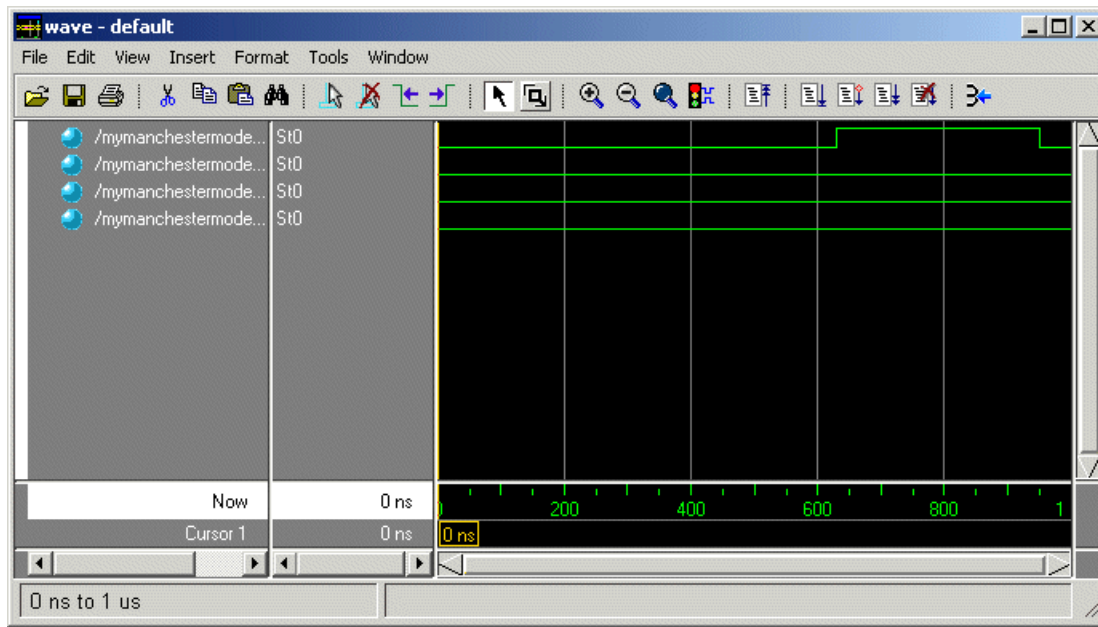
17 Exit the simulation. One way of exiting is to enter the following command:

```
dataset close MyVCDfile
```

ModelSim closes the data set, clears the **wave** window, and exits the simulation.

For more information on the vcd2wlf utility and working with data sets, see the ModelSim documentation.

MATLAB Functions — Alphabetical List

configuremodelsim

Purpose   Install or uninstall ModelSim for use with MATLAB and Simulink

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

Description configuremodelsim starts an interactive installation script that installs and configures ModelSim for use with MATLAB and Simulink features of the Link for ModelSim.

configuremodelsim('PropertyName', 'PropertyValue'...) starts an interactive or programmatic script that installs or uninstalls ModelSim for use with MATLAB and Simulink. If you specify only the 'action' property, the script runs in interactive mode.

The function modifies an installed version of ModelSim. After modification, ModelSim is ready to use the Link for ModelSim based on settings of property name/property value pairs that specify

- Whether the function installs or uninstalls 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 the Link for ModelSim (see the wrapverilog command)

Property Name/Property Value Pairs

`'action', 'install'`
　　Installs and configures an installed version of ModelSim for use
　　with the MATLAB and Simulink. After modification, ModelSim
　　starts up ready to use the Link for ModelSim.

`'action', 'uninstall'`
　　Uninstalls ModelSim for use with the MATLAB and Simulink.

`'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.

`'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.

Examples
The following function call starts the interactive installation script that
installs ModelSim for use with the MATLAB and Simulink:

```
configuremodelsim
```

The following function call installs and 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 ns.

configuremodelsim

```
configuremodelsim('action','install','tclstart','vsimmatlab
work.parse; matlabtb parse 10 ns -socket 4449')
```

Purpose   Start 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" on page 8-7 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.

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 1-18.

Note If you specify the operating system option (`'0'` or `0`), use `hdldaemon('status')` to acquire the assigned socket port number. You must specify this port number when you issue a link request with the `matlabtb` or `matlabtbeval` command in 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')
```

Purpose   Convert multivalued logic to decimal

Syntax   `mvl2dec('L')`

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

Examples   The following function call returns the decimal value 23:

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

The following function call returns `NaN`:

```
mvl2dec('UUUUUU')
```

vsim

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 the Link for ModelSim (see the `wrapverilog` command)

The property name/propert 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`.

Format	**Example**
`<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 1-18

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 the 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 ns.

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

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

communication on the same computer is to be used for links between
Simulink and ModelSim, using socket port 4449:

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

ModelSim Commands — Alphabetical List

matlabcp

Purpose   Associate MATLAB function with instantiated VHDL entity

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

**Arguments**   `<instance>`

Specifies an instance of a VHDL entity that is associated with
a MATLAB function. By default, `matlabcp` associates the
instance to a MATLAB function that has the same name as the
instance. For example, if the instance is `myfirfilter`, `matlabcp`
associates the instance with the MATLAB function `myfirfilter`.
Alternatively, you can specify a different MATLAB function with
`-mfunc`.

Do not specify an instance of a VHDL entity that has already been
associated with a MATLAB test bench function (via `matlabtb`).

`<time-specs>`

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

| | |
|---|---|
| `<timen>,...` | Specifies one or more discrete time values at which the specified MATLAB function is called. Each time value is relative to the current simulation time. Note that the MATLAB function is always called once at the start of the simulation, even if you do not specify a time. |
| `-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 computer, the <tcp_spec> can consist of just a TCP/IP port number or service name (alias). If you are setting up communication between computers, you must also specify the name or Internet address of the remote host that is running the MATLAB server (hdldaemon). The following table lists different ways of specifying <tcp_spec>.

| Format | Example |
|---|---|
| <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 1-18.

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
> `matlabcp` command must match what you specify for the
> communication mode when you issue the `hdldaemon` command in
> MATLAB. For more information on modes of communication, see
> "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 on the
rising edge (transition from `'0'` to `'1'`) of any of the specified
signals. 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, `matlabcp`
attaches the entity to a MATLAB function that has the same name
as the entity. For example, if the entity is `myfirfilter`, `matlabcp`
associates the entity with the MATLAB function `myfirfilter`. If
you omit this argument and `matlabcp` does not find a MATLAB
function with the same name, the command generates an error
message.

**Description**    The `matlabcp` command

- Starts the ModelSim client component of the Link for ModelSim.
- Associates a specified instance of a VHDL entity created in ModelSim with a MATLAB function.
- 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 `matlabcp` command.

This command cancels any pending events scheduled by a previous `matlabcp` command that specified the same instance. For example, if you issue the command `matlabcp` for instance `foo`, all previously scheduled events initiated by `matlabcp` on `foo` are canceled.

**Examples**    The following command starts the ModelSim client component of the Link for ModelSim, associates an instance of the entity `u_osc_filter` with the MATLAB function `oscfilter`, and initiates a MATLAB and ModelSim simulation session using (by default) shared memory communication.

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

For an example of the use of `matlabcp` in the context of an automated simulation session, see the following file from the Link for ModelSim Oscillator demo:

```
MATLABROOT\toolbox\modelsim\modelsimdemos\osccmnds.m
```

matlabtb

Purpose   Initiate MATLAB test bench session for instantiated VHDL entity

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

Arguments   `<instance>`

Specifies the instance of a VHDL entity that attaches to a
MATLAB test bench 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`.

Do not specify an instance of a VHDL entity that has already been
associated with a MATLAB component function (via `matlabcp`).

`<time-specs>`

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

`<timen>,...`	Specifies one or more discrete time values at which the 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 computer, the <tcp_spec> can consist of just a TCP/IP
    port number or service name (alias). If you are setting up
    communication between computers, you must also specify the
    name or Internet address of the remote host that is running the
    MATLAB server (hdldaemon). The following table lists different
    ways of specifying <tcp_spec>.

Format	Example
<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 1-18.

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
> 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 on the
rising edge (transition from '0' to '1') of any of the specified
signals. 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

- Starts the ModelSim client component of the Link for ModelSim.

- Associates a specified instance of a VHDL entity created in ModelSim with a MATLAB function.

- 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:

```
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.

matlabtb

```
vsim> matlabtb myfirfilter 10 ns -rising \work\fclk
-sensitivity \work\din -socket 4449@compa
```

Purpose   Call a specified MATLAB function on behalf of instantiated 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 computer, the <tcp_spec> can consist of just a TCP/IP
                   port number or service name (alias). If you are setting up
                   communication between computers, you must also specify the
                   name or Internet address of the remote host. The following table
                   lists different ways of specifying <tcp_spec>.

Format	Example
<port-num>	4449 on this computer
<port-alias>	matlabservice on this computer
<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 1-18.

matlabtbeval

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

- Starts the ModelSim client component of the Link for ModelSim.

- Associates a specified instance of a VHDL entity created in ModelSim with a MATLAB function.

- 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

Purpose   Terminate active MATLAB test bench and MATLAB component sessions

Syntax   `nomatlabtb`

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

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

```
vsim> nomatlabtb
```

See Also   `matlabcp`,`matlabtb`

vsimmatlab

Purpose   Load instantiated VHDL entity for verification with MATLAB

Syntax   vsimmatlab <instance> [<vsim_args>]

Arguments   <instance>
                    Specifies the instance of a VHDL entity to load for verification.

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

Description The vsimmatlab 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.

Examples   The following command loads the entity instance parse from library
                 work for verification and sets up ModelSim so it can establish a
                 communication link with MATLAB:

                    ModelSim> vsimmatlab work.parse

vsimulink

Purpose   Load instantiated 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 matlabservice.

                       For more information on choosing TCP/IP socket ports, see
                       "Choosing TCP/IP Socket Ports" on page 1-18.

                       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 7-44.

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

Purpose   Apply VHDL wrapper to Verilog module

Syntax   `wrapverilog [-nocompile] <verilog_module>`

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

                 `-nocompile`
                   Suppresses automatic compilation of the resulting VHDL file,
                   *verilog_module_*`wrap.vhd`.

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

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

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

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

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

Simulink Blocks — Alphabetical List

To VCD File

Purpose
Generate a value change dump (VCD) file

Library
Link for ModelSim

Description

simulink.vcd
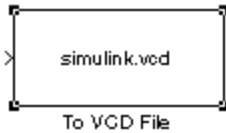
To VCD File

The To VCD File block generates a VCD file that contains information about changes to signals connected to the block's input ports and names the file with the specified 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 7-57.

**Dialog
Box**
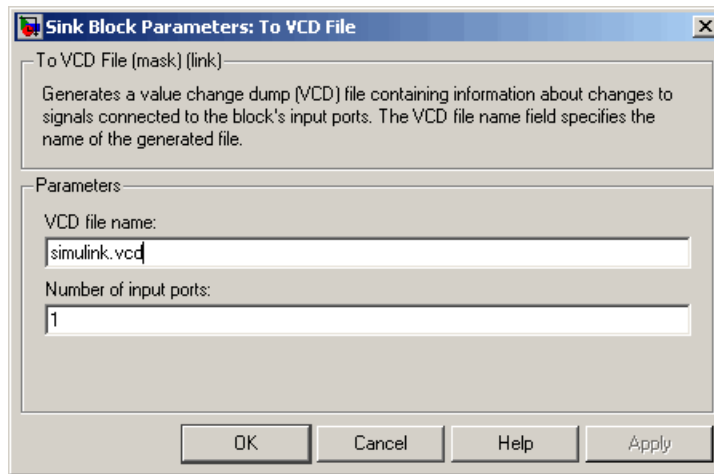
VCD file name

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 identify each instance uniquely.

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

Purpose   Cosimulate a hardware component by communicating with a VHDL model executing 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.
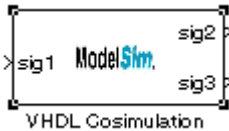
The tabbed panes on the block's dialog box let you configure:

- Block input and output ports that correspond to signals (including internal signals) of a VHDL model. You must specify a sample time for each output port; you can also specify a data type for each output port.

- Type of communication and communication settings used to exchange data between simulators

- The timing relationship between units of simulation time in Simulink and ModelSim

- Rising-edge or falling-edge clocks to apply to your model. You can specify the period for each clock signal.

- Tcl commands to run before and after the simulation

The **Ports** pane provides fields for mapping signals of your VHDL design to input and output ports in your block. The signals 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 specified sample rate.

VHDL Cosimulation

In general, Simulink handles port sample periods as follows:

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

- If an input port is connected to a signal that does not have an explicit sample period, Simulink assigns a sample period that is equal to the least common multiple (LCM) of all identified input port sample periods for the model.

- After Simulink sets the input port sample periods, it applies user-specified output sample times to all output ports. An explicit sample time must be specified for each output port.

In addition to specifying output port sample times, you can force the fixed point data types on output ports. For example, setting the **Data Type** property of an 8–bit output port to `Signed` and setting its **Fraction Length** property to `5` would force the data type to `sfix8_En5`.

Note The **Data Type** and **Fraction Length** properties will apply only to VHDL signals of `STD_LOGIC` or `STD_LOGIC_VECTOR` type.

The **Timescales** pane lets you choose an optimal timing relationship between Simulink and ModelSim. You can configure either a *relative* timing relationship (Simulink seconds correspond to a ModelSim defined tick interval) or an *absolute* timing relationship (Simulink seconds correspond to an absolute unit of ModelSim time).

The **Connection** pane specifies the communications mode used between Simulink and ModelSim. If you use TCP socket communication, this pane provides fields for specifying a socket port and for the hostname of a remote computer running ModelSim.

The **Clocks** pane lets you create optional rising-edge and falling-edge clocks that apply stimuli to your cosimulation model. You can either specify an explicit period for each clock, or accept a default period of 2).

Simulink attempts to create a clock that has a 50% duty cycle and a predefined phase that is inverted for the falling edge case.

If you have configured the **Timescales** pane for relative timing mode, the following restrictions apply to clock periods:

- If you specify an explicit clock period, enter an even integer with a minimum value of 2.

- If the clock period (whether explicitly specified or defaulted) is not an even integer, Simulink displays a warning and truncates the period to the nearest even integer.

The following figure shows a timing diagram that includes rising-edge and falling-edge clocks with a Simulink sample period of T=10 and a ModelSim resolution limit of 1 ns. The figure also shows that given those timing parameters, the clock duty cycle is 50%.
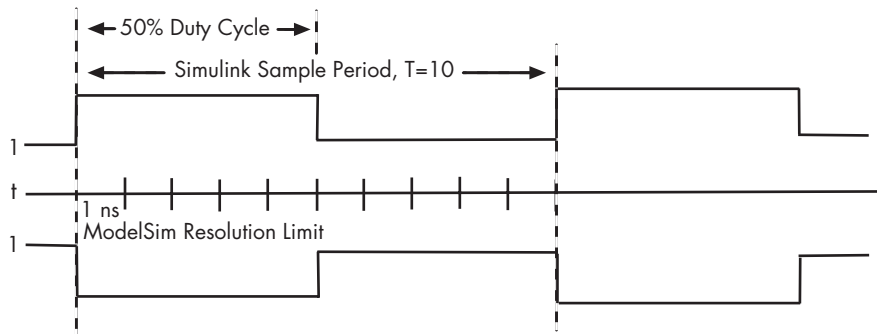
Rising Edge Clock

Falling Edge Clock

The **Tcl** pane 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 **Pre–simulation commands** field on this pane is particularly useful for simulation initialization and startup operations, but cannot be used to change simulation state.

VHDL Cosimulation

Dialog Box

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

- "Ports Pane" on page 10-8
- "Connection Pane" on page 10-12
- "Timescales Pane" on page 10-14
- "Clocks Pane" on page 10-17
- "Tcl Pane" on page 10-19

Ports Pane
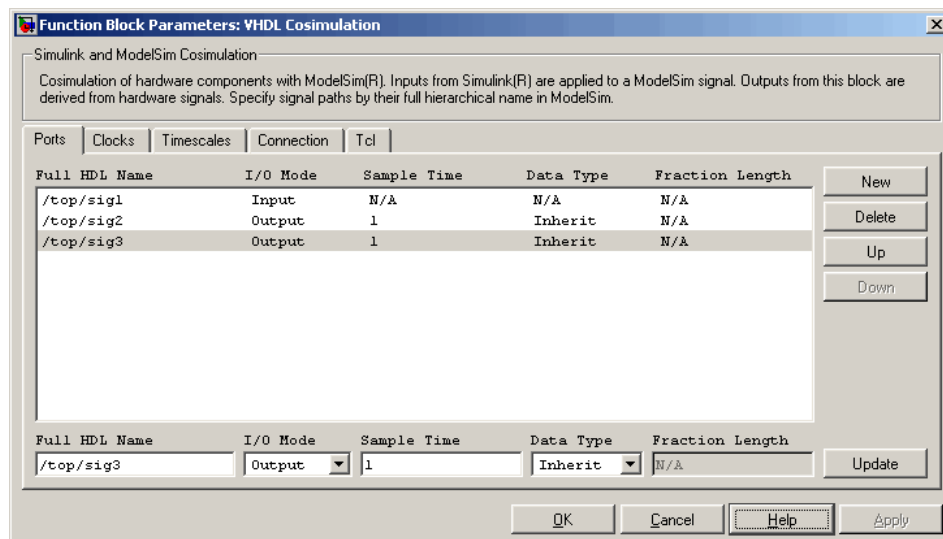
The list at the center of the pane displays VHDL signals corresponding to ports on the Cosimulation block.

The list is maintained by the buttons on the right of the pane:

- **New**: Add a new signal to the list and select it for editing.

- **Delete**: Remove a signal from the list.

- **Up**: Move the selected signal up one position in the list.

- **Down**: Move the selected signal down one position in the list.

- **Update**: Update the displayed values in the list for the selected signal. Note that this affects only the signal list. To commit edits to the Simulink model, you must also click **Apply**.

To edit the properties of a signal, select the signal from the list and set the desired values in the fields at the bottom of the pane. Then click **Update** to enter the new values into the list. The properties of a signal are as follows.

Full HDL Name

Specifies the signal pathname, using ModelSim pathname syntax. For example, a pathname for an input port might be /manchester/samp. The signal can be at any level of the VHDL design hierarchy. The Cosimulation block port corresponding to the signal is labeled with the **Full HDL Name**.

Note You can copy signal pathnames directly from the ModelSim **wave** window and paste them into the **Full HDL Name** field, using the standard copy and paste commands in ModelSim and Simulink. After pasting a signal pathname into the **Full HDL Name** field, you must click the **Update** button to complete the paste operation and update the signal list.

I/O Mode

Select either Input or Output.

Input designates 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.

VHDL Cosimulation

> **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.

`Output` designates signals of your VHDL model that are to be read by Simulink. For output signals, you must specify an explicit sample time. You can also specify a data type, if desired (see below).

Sample Time

This property is enabled only for output signals. You must specify an explicit sample time.

Sample Time represents the time interval between consecutive samples applied to the output port. The default sample time is `1`. The exact interpretation of the output port sample time depends on the settings of the **Timescale** pane of the VHDL Cosimulation block (see "Timescales Pane" on page 10-14). See also "Representation of Simulation Time" on page 7-9.

Data Type
Fraction Length

These two related parameters apply only to output signals.

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

The **Fraction Length** property specifies the size, in bits, of the fractional part of the signal in fixed-point representation. **Fraction Length** is enabled when the **Data Type** property is not set to `Inherit`.

Output port data types are determined by the signal width and by the **Data Type** and **Fraction Length** properties of the signal.

Note The **Data Type** and **Fraction Length** properties apply only to VHDL signals of STD_LOGIC or STD_LOGIC_VECTOR type.

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

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

  Inherit is the default setting. When Inherit is selected, the **Fraction Length** edit field is disabled.

  Simulink attempts to compute the data type of the signal connected to the output port by backward propagation. For example, if a Signal Specification block is connected to an output, Simulink will force the data type specified by Signal Specification block on the output port.

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

- Select Signed from the **Data Type** list if you want to explicitly assign a signed fixed point data type. When Signed is selected, the **Fraction Length** edit field is enabled. The port is assigned a fixed point type sfixN_EnF, where N is the signal width and F is the **Fraction Length**.

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

a **Data Type** set to Signed and **Fraction Length** of -5, Simulink forces the data type to sfix16_E5.

- Select Unsigned from the **Data Type** list if you want to explicitly assign an unsigned fixed point data type When Unsigned is selected, the **Fraction Length** edit field is enabled. The port is assigned a fixed point type ufixN_EnF, where N is the signal width and F is the **Fraction Length**.

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

Connection Pane

This figure shows the default configuration of the **Connection** pane. By default, the block is configured for shared memory communication between Simulink and ModelSim, running on a single computer.
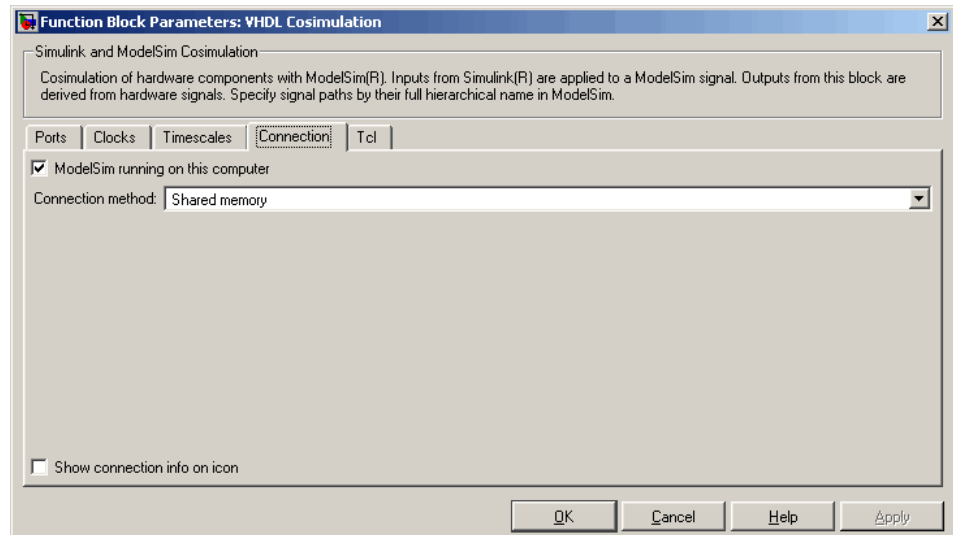
If you select TCP/IP socket mode communication, the pane displays additional properties, as shown in the figure below.
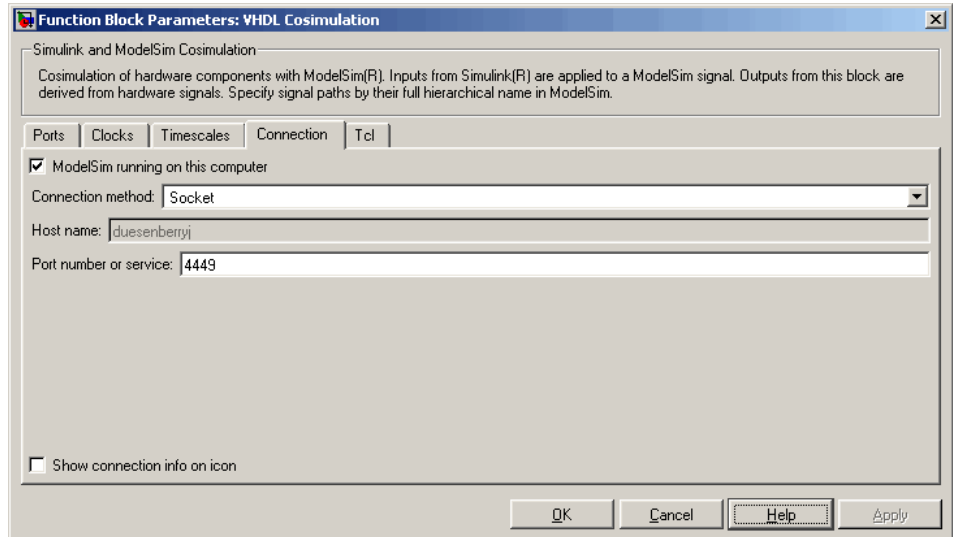
ModelSim running on this computer

> Select this option if you want to run Simulink and ModelSim on the same computer. When both applications run on the same computer, you have the choice of using shared memory or TCP sockets for the communication channel between the two applications. If this option is deselected, only TCP/IP socket mode is available, and the **Connection method** menu is disabled.

Connection method

> This list is enabled when **ModelSim running on this computer** is selected. Select Socket if you want Simulink and ModelSim to communicate via a designated TCP/IP socket. Select Shared memory if you want Simulink and ModelSim to communicate via shared memory. For more information on these connection methods, see "Configuring the Communication Link" on page 7-44.

VHDL Cosimulation

Host name

> If Simulink and ModelSim are running on different computers, this text field is enabled. The field specifies the host name of the computer that is running your VHDL simulation in ModelSim.

Port number or service

> A valid TCP socket port number or service for your computer system. For information on choosing TCP socket ports, see "Choosing TCP/IP Socket Ports" on page 1-18.

Show connection info on icon

> When this option is selected, Simulink indicates information about the selected communication method and (if applicable) communication options information on the VHDL Cosimulation block icon. If shared memory is selected, the icon displays the string SharedMem. If TCP socket communication is selected, the icon displays the host name and port number in the format hostname:port.
>
> In a model that has multiple VHDL Cosimulation blocks, with each communicating to different instances of ModelSim in different modes, this information helps to distinguish between different cosimulations.

Timescales Pane

The **Timescales** pane of the VHDL Cosimulation block parameters dialog lets you choose an optimal timing relationship between Simulink and ModelSim. The figure below shows the default settings of the **Timescales** pane.
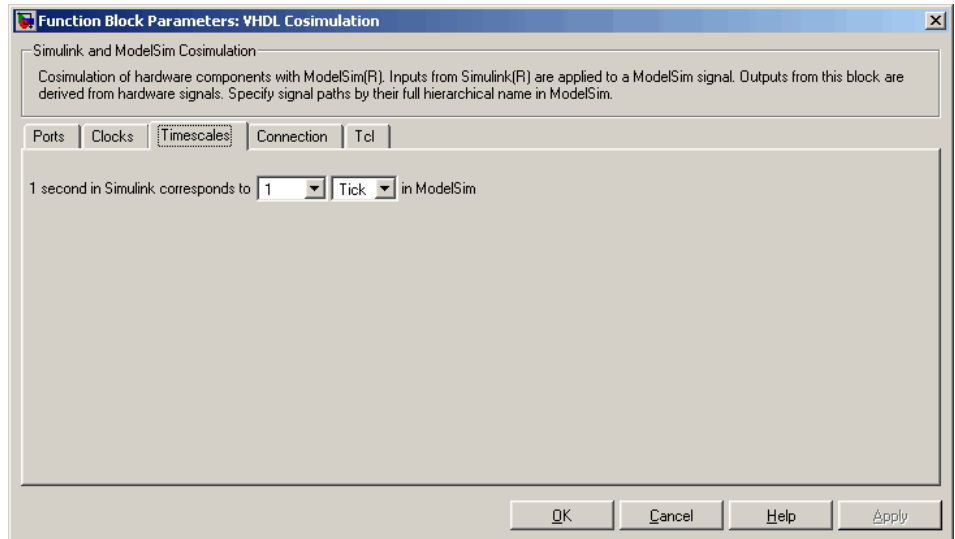
The **Timescales** pane specifies a correspondence between one second of Simulink time and some quantity of ModelSim time. This quantity of ModelSim time can be expressed in one of the following ways:

- In *relative* terms (i.e., as some number of ModelSim ticks). In this case, the cosimulation is said to operate in *relative timing mode*. Relative timing mode is the default.

  To use relative mode, select Tick from the right-hand list, and select the desired number of ticks (1, 10, 100, or 1000) from the left-hand list. For example, in the figure below the **Timescales** pane is configured for a relative timing correspondence of 10 ModelSim ticks to 1 Simulink second.
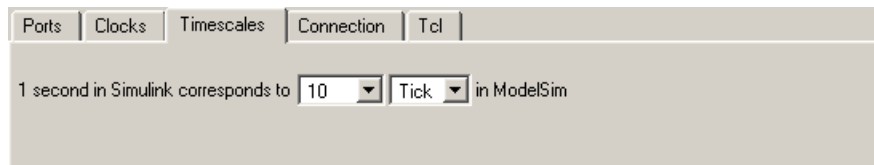
VHDL Cosimulation

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

  To use absolute mode, select a unit of absolute time (available units are fs, ps, ns, us, ms, s) from the right-hand list. Then select a scale factor (available values are 1, 10, 100, and 1000) from the left-hand list. For example, in the figure below the **Timescales** pane is configured for an absolute timing correspondence of 1 ModelSim second to 1 Simulink second.
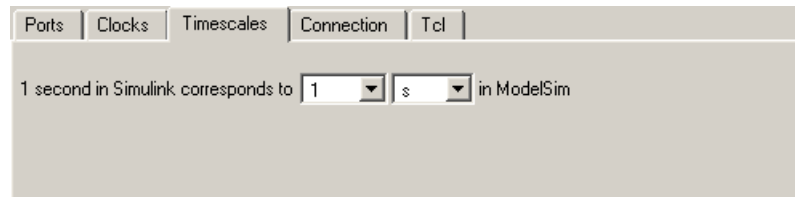
For detailed information on the relationship between Simulink and ModelSim during cosimulation, and on the operation of relative and absolute timing modes, see "Representation of Simulation Time" on page 7-9

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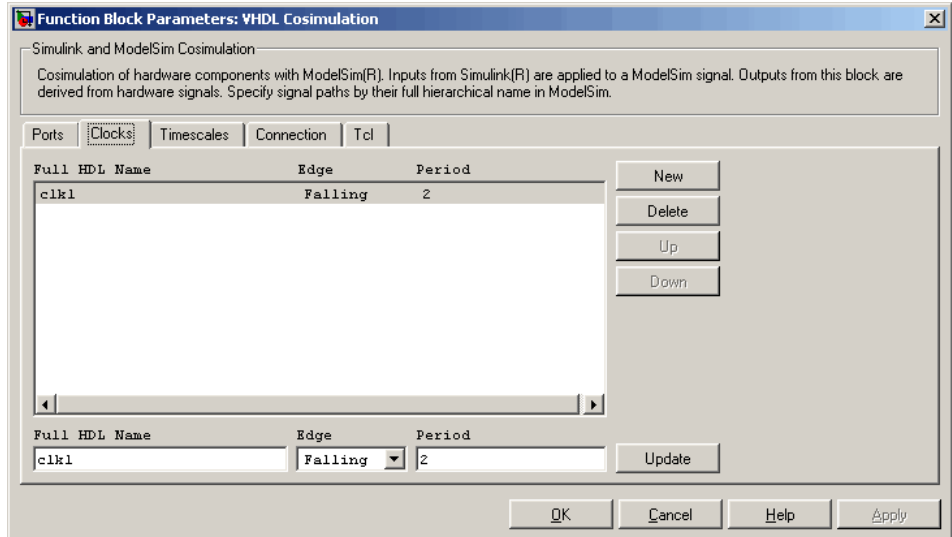
The scrolling list at the center of the pane displays VHDL clocks that drive values to the VHDL signals that you are modeling, using the deposit method.

The list of clock signals is maintained by the buttons on the right of the pane:

- **New**: Add a new clock signal to the list and select it for editing.

- **Delete**: Remove a clock signal from the list.

- **Up**: Move the selected clock signal up one position in the list.

- **Down**: Move the selected clock signal down one position in the list.

- **Update**: Update the displayed values in the list for the selected clock signal. Note that this affects only the signal list. To commit edits to the Simulink model, you must also click **Apply**.

VHDL Cosimulation

To edit the properties of a clock signal, select it from the list and enter (or select) desired values in the fields at the bottom of the pane. Then click **Update** to enter the new values into the list. The properties of a clock signal are

Full HDL Name

Specify each clock as a signal pathname, using ModelSim pathname syntax. A sample pathname for a clock might be `/manchester/clk`.

Note You can copy signal pathnames directly from the ModelSim **wave** window and paste them into the **Full HDL Name** field, using the standard copy and paste commands in ModelSim and Simulink. After pasting a signal pathname into the **Full HDL Name** field, you must click the **Update** button to complete the paste operation and update the signal list.

Edge

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

Period

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

The clock period must be an even integer with a minimum value of 2.

Tcl Pane
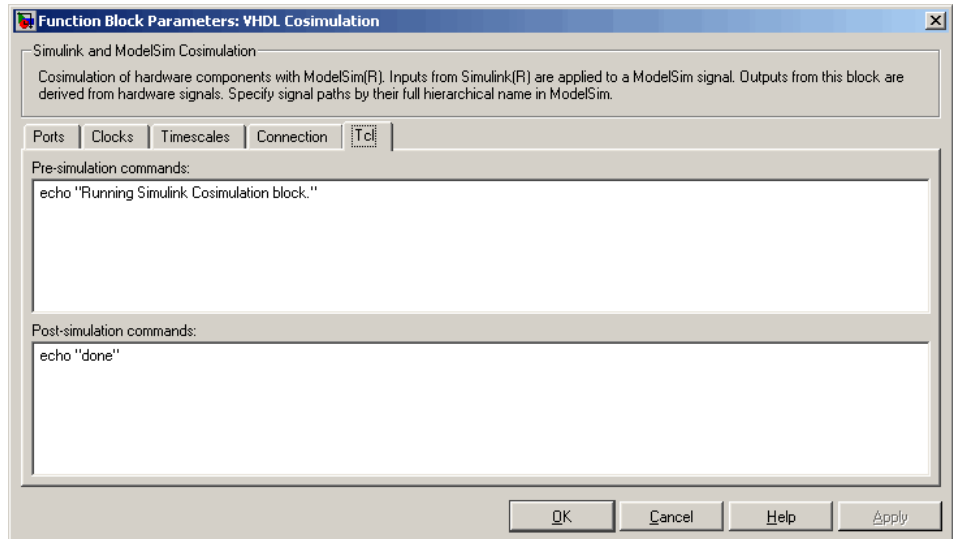
Pre–simulation commands

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

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

```
do mycosimstartup.do
```

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

Note The command string or DO file that you specify for this parameter cannot include commands that load a ModelSim project or modify simulator state. For example, they cannot include commands such as start, stop, or restart.

Post–simulation commands

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

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

```
do mycosimcleanup.do
```

Notes

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

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

Examples

Use this list to find examples in the documentation.

Demos

MATLAB and ModelSim Random Number Generator Tutorial

Simulink and ModelSim Inverter Tutorial

MATLAB and ModelSim Manchester Receiver Tutorial

Coding MATLAB and ModelSim Applications

Generating a VCD File

Index