Link for ModelSim® 2
User's Guide

MATLAB®
&SIMULINK®

The MathWorks™
Accelerating the pace of engineering and science
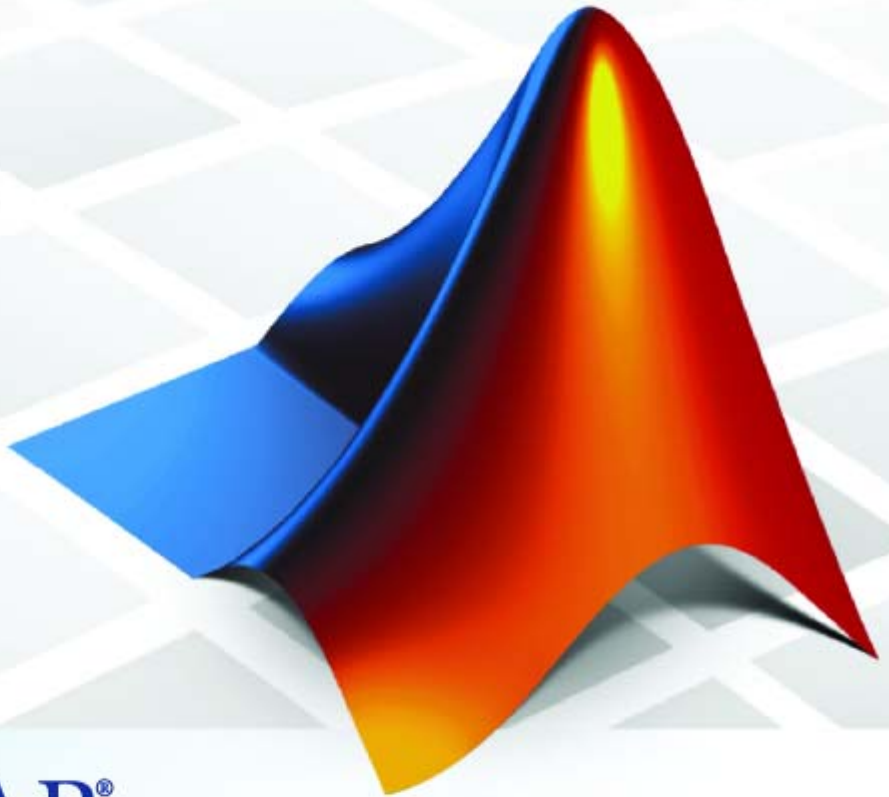
How to Contact The MathWorks

	`www.mathworks.com`	Web
	`comp.soft-sys.matlab`	Newsgroup
	`www.mathworks.com/contact_TS.html`	Technical Support
	`suggest@mathworks.com`	Product enhancement suggestions
	`bugs@mathworks.com`	Bug reports
	`doc@mathworks.com`	Documentation error reports
	`service@mathworks.com`	Order status, license renewals, passcodes
	`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Link for ModelSim User's Guide

© COPYRIGHT 2003–2007 by The MathWorks, Inc.

Trademarks

MATLAB, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, SimBiology, SimHydraulics, SimEvents, and xPC TargetBox are registered trademarks and The MathWorks, the L-shaped membrane logo, Embedded MATLAB, and PolySpace are trademarks of The MathWorks, Inc.

ModelSim® is a registered trademark of Mentor Graphics Corporation.

Screenshots of ModelSim in the Link for ModelSim documentation are copyright protected by Mentor Graphics Corporation.

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.

Contents

Getting Started

1

Linking MATLAB to ModelSim

2

Linking Simulink to ModelSim

3

4 MATLAB Functions — Alphabetical List

5 ModelSim Commands — Alphabetical List

6 Simulink Blocks — Alphabetical List

A Examples

1

Getting Started

What Is Link for ModelSim?

Overview

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.

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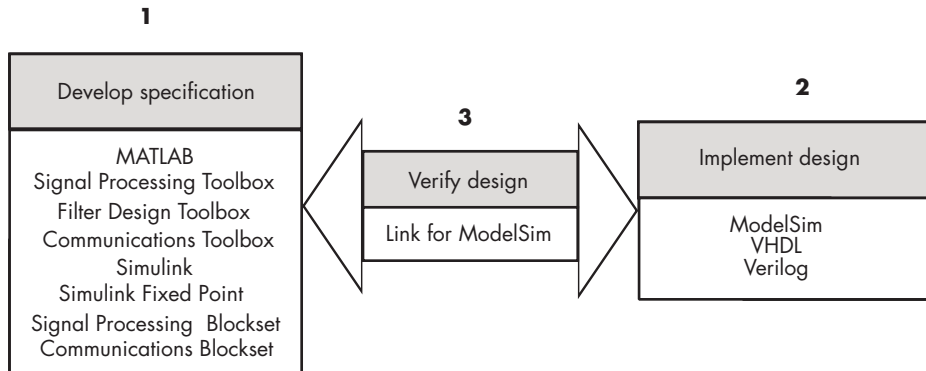
As the figure shows, Link for ModelSim connects tools that traditionally have been used discretely to accomplish specific steps in the design process. By connecting the tools, Link for ModelSim simplifies verification by allowing you to cosimulate the implementation and original specification directly.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

In addition to the preceding design scenario, 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

---

Note You can cosimulate a model using SystemVerilog and/or SystemC with MATLAB or Simulink using Link for ModelSim. Write simple wrappers around the SystemC and make sure that the SystemVerilog cosimulation connections are to ports or signals of data types supported by Link for ModelSim.

How Link for ModelSim Works with MATLAB and ModelSim

When linked with MATLAB, ModelSim functions as the client, as the following figure shows.
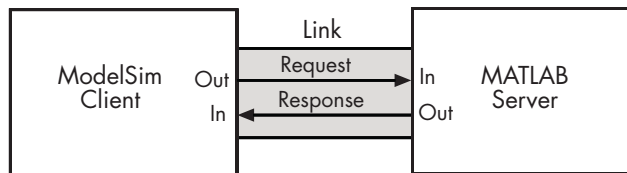
In this scenario, a MATLAB server function waits for service requests that it receives from a ModelSim simulator session. After receiving a request, the server establishes a communication link, and invokes a specified MATLAB function that computes data for, verifies, or visualizes the HDL model (coded in VHDL or Verilog) that is under simulation in ModelSim.

After the server is running, you can start and configure ModelSim for use with MATLAB with the supplied Link for ModelSim function `vsim`. 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 following figure shows how a MATLAB test bench function wraps around and communicates with ModelSim during a test bench simulation session.
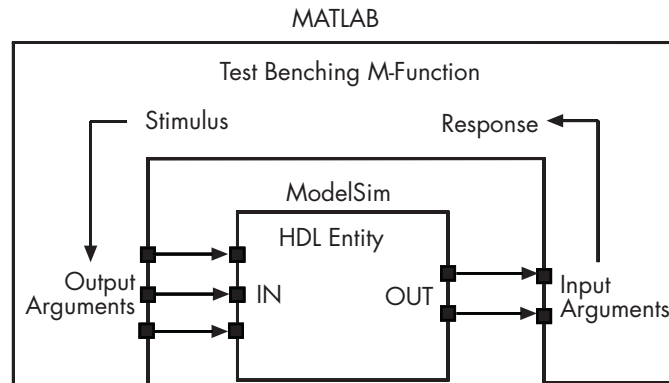
The following figure shows how a MATLAB component function is wrapped around by and communicates with ModelSim during a component simulation session.
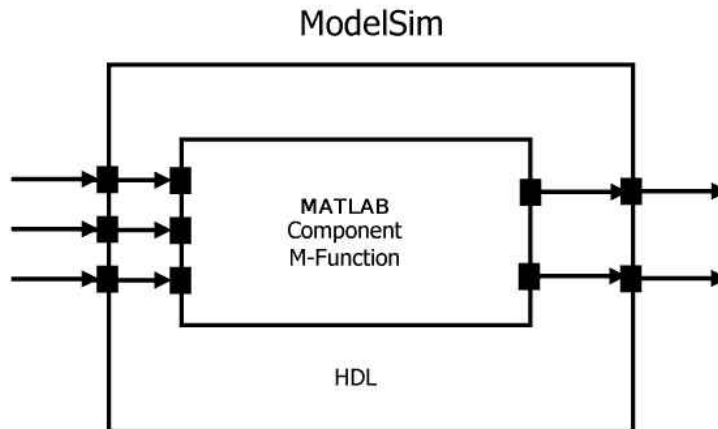
During the configuration process, Link for ModelSim equips ModelSim with a set of Link for ModelSim command extensions you use to

- Load the ModelSim simulator, `vsim`, with an instance of a HDL entity to be tested with MATLAB

- Initiate a MATLAB test bench or component 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 MATLAB function that attaches to and executes on behalf of the HDL entity

- Timing specifications and other control data that specifies when the entity's MATLAB function is to be called

How Link for ModelSim Works with Simulink and ModelSim

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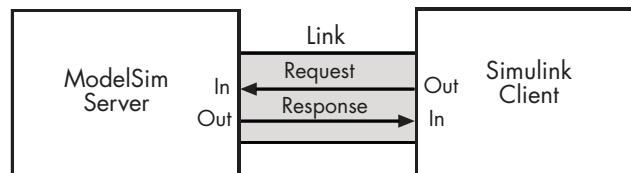
In this case, ModelSim responds to simulation requests it receives from cosimulation blocks in a Simulink model. You initiate a cosimulation session from Simulink. After a session is started, you can use Simulink and ModelSim to monitor simulation progress and results. For example, you might add signals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

Using the Block Parameters dialog for a HDL Cosimulation block, you can configure

- Block input and output ports that correspond to signals (including internal signals) of an HDL 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.

Link for ModelSim equips ModelSim with a set of Link for ModelSim command extensions. Using the supplied command `vsimulink`, you execute the ModelSim simulator with an HDL entity for cosimulation with Simulink. After the entity is loaded, you can start the cosimulation session from Simulink.

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

Modes of Communication

The mode of communication that 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. This option offers the greatest scalability. For more on TCP/IP socket communication, see "Choosing TCP/IP Socket Ports" on page D-2.

Prerequisites for Using Link for ModelSim

In this section...
"What You Need to Know" on page 1-8
"Required Products" on page 1-8

What You Need to Know

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

- Hardware design and system integration
- VHDL and/or Verilog
- 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
- Filter Design Toolbox
- Communications Toolbox
- Signal Processing Blockset
- Communications Blockset

Required Products

Link for ModelSim requires the following:

Platform	Visit The MathWorks Link for ModelSim requirements page for specific platforms supported with the current release of Link for ModelSim.
Application software	ModelSim SE/PE, a Mentor Graphics product. Visit The MathWorks Link for ModelSim requirements page for specific versions supported with the current release of Link for ModelSim.
	MATLAB
Application software required for cosimulation	Simulink
	Simulink Fixed Point
Optional application software	Communications Blockset
	Signal Processing Blockset
	Filter Design Toolbox
	Signal Processing Toolbox
	Note that many Link for ModelSim demos require one or more of the above.
Platform-specific software	On the Linux platform, the gcc c++ libraries (3.2 or later) are required by 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
```

Setting Up Your Environment for Link for ModelSim

In this section...
"Installing Link for ModelSim" on page 1-10
"Installing Related Application Software" on page 1-10
"Setting Up ModelSim for Use with Link for ModelSim" on page 1-10
"Using the Link for ModelSim Libraries" on page 1-16

Installing Link for ModelSim

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

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 on different machines, you need to install MATLAB and any applicable toolbox software on multiple systems. Each instance of MATLAB can run only one instance of the server.

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

Setting Up ModelSim for Use with Link for ModelSim

Link for ModelSim provides a guided set-up script (syschecklfm) for configuring your simulator. This script works whether you have installed Link for ModelSim and MATLAB on the same machine as ModelSim or installed them on different machines.

The set-up script creates a configuration file containing the location of the appropriate Link for ModelSim MATLAB and Simulink libraries. You can then include this configuration with any other calls you make using Mentor Graphics vsim from ModelSim. You only need to run this script once.

> **Note** The Link for ModelSim configuration and diagnostic script works only on UNIX and Linux. Windows users: please see instructions below.

If you plan to use the MATLAB `vsim.m` function instead, no setup is required. Refer to "Using the Link for ModelSim Libraries" on page 1-16 for the correct Link for ModelSim library for your platform. Then see "Starting ModelSim from MATLAB" on page 1-21.

After you've created your configuration files, see "Starting ModelSim from a Shell" on page 1-23.

Using the Configuration and Diagnostic Script for UNIX/Linux

`syschecklfm` provides an easy way to configure your simulator setup to work with Link for ModelSim.

The following is an example of running `syschecklfm` on a Linux 64 machine with Link for ModelSim libraries in a different location than where they were initially installed and specifying a TCP/IP connection.

Start `syschecklfm`:

```
% syschecklfm
********************************************************************************

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

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

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

```
*********************************************************************************

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

Next, the script needs to know if the Link for ModelSim libraries are in the
default directory (where they were initially installed) or if you have moved
them to another directory. If you have ModelSim and MATLAB on separate
machines, move the Link for ModelSim libraries to the ModelSim machine.

```
Select method to search for Link for ModelSim libraries:
1. Use libraries installed with Link for ModelSim.
2. Prompt me to specify the direct path to the libraries.
2
Enter the path to liblfmhdlc_gcc41.so and liblfmhdls_gcc41.so:
/tmp/lfmconfig/linux64
Found /tmp/lfmconfig/linux64/liblfmhdlc_gcc41.so
and /tmp/lfmconfig/linux64/liblfmhdls_gcc41.so.
```

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

```
*********************************************************************************

Running dependency checker "ldd /tmp/lfmconfig/linux64/liblfmhdlc_gcc41.so".
Dependency checker passed.
Dependency status:
        librt.so.1 => /lib64/tls/librt.so.1 (0x00002aaaaac0a000)
        libpthread.so.0 => /lib64/tls/libpthread.so.0 (0x00002aaaaad12000)
        libstdc++.so.6 => /devel/Ahdl/nightly/matlab/sys/os/glnxa64/libstdc++.so
.6 (0x00002aaaaae27000)
        libm.so.6 => /lib64/tls/libm.so.6 (0x00002aaaab029000)
        libgcc_s.so.1 => /devel/Ahdl/nightly/matlab/sys/os/glnxa64/libgcc_s.so.1
 (0x00002aaaab180000)
        libc.so.6 => /lib64/tls/libc.so.6 (0x00002aaaab28d000)
```

```
         /lib64/ld-linux-x86-64.so.2 (0x0000555555555000)
********************************************************************************
```

This next step loads the Link for ModelSim libraries and compiles a test module to verify the libraries loaded correctly.

```
Press Enter to load Link for ModelSim or enter 'n' to skip this test:

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

Top level modules:
        d24154

********************************************************************************

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

# 6.0e

# vsim -do exit -foreign {matlabclient /tmp/lfmconfig/linux64/liblfmhdlc_gcc41.so}
      -c worklfx24154.d24154
# //  ModelSim SE-64 6.0e Jun 17 2005 Linux 2.6.11.4-20a-smp
# //
# //  Copyright Mentor Graphics Corporation 2005
# //             All Rights Reserved.
# //
# //  THIS WORK CONTAINS TRADE SECRET AND
# //  PROPRIETARY INFORMATION WHICH IS THE PROPERTY
# //  OF MENTOR GRAPHICS CORPORATION OR ITS LICENSORS
# //  AND IS SUBJECT TO LICENSE TERMS.
# //
# Loading worklfx24154.d24154
# Loading /tmp/lfmconfig/linux64/liblfmhdlc_gcc41.so
# exit

********************************************************************************
```

```
Link for ModelSim libraries loaded successfully.
*************************************************************************
```

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

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

Enter an available port [5001]

Enter remote host [localhost]

Press Enter to continue

ttcp_glnx -t -p5001 localhost
Connection successful
```

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

```
*************************************************************************

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

*************************************************************************

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

Diagnosis Completed
```

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

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

Using the Configuration and Diagnostic Script with Windows

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

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

```
//Command file for MATLAB Link for ModelSim.
//Loading of foreign Library, usage example: vsim -f matlab14455.arg entity.
//You can manually change the following line to point to the correct library.
//The default location of the 32-bit Windows library is at
//MATLABROOT/toolbox/modelsim/windows32/liblfmhdlc_vs05.dll.

-foreign "matlabclient c:/path/liblfmhdlc_vs05.dll"
```

where *path* is the path to the particular Link for ModelSim shared library you want to invoke (in this example, liblfmhdlc_vs05.so. See "Using the Link for ModelSim Libraries" on page 1-16).

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

The comments in the above text are optional.

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

```
//Command file for Simulink Link for ModelSim.
//Loading of foreign Library, usage example: vsim -f simulink14455.arg entity.
//You can manually change the following line to point to the correct library.
//For example the default location of the 32-bit Windows library is at
//MATLABROOT/toolbox/modelsim/windows32/liblfmhdls_vs05.so.

//For socket connection uncomment and modify the following line:
-foreign "simlinkserver c:/path/liblfmhdls_vs05.so  ; -socket 5001"

//For shared connection uncomment and modify the following line:
//-foreign "simlinkserver c:/path/liblfmhdls_vs05.so"
```

Where *path* is the path to the particular Link for ModelSim shared library you want to invoke (in this example, liblfmhdls_vs05.so. See "Using the Link for ModelSim Libraries" on page 1-16.

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

The comments in the above text are optional.

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

Using the Link for ModelSim Libraries

In general, you want to use the same compiler for all libraries linked into the same executable. Link for ModelSim provides many versions of the same library by using the same compilers that MATLAB is compiled with (they vary by platform) as well as the compilers that are available with the HDL simulators (usually some version of gcc). Using the same libraries ensures compatibility with other C++ libraries that may get linked into the HDL simulator, including SystemC libraries.

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

- Link other 3rd party applications into your HDL simulator

- Compile and link in SystemC code as part of your design or testbench

- Write custom C/C++ applications and link them into your HDL simulator

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

> **Note** Link for ModelSim supports running in 32-bit mode on a 64-bit Solaris machine, but it does not support running on a 32-bit Solaris platform.

Library Names

The Link for ModelSim libraries are named according to the following format:

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

where

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

Not all combinations are supported. See "Default Libraries" on page 1-17 for valid combinations.

Default Libraries

Link for ModelSim scripts fully support the use of designated default libraries. The default libraries are different for each of the products, mostly due to backward-compatibility issues.

With Link for ModelSim, the default library for each platform is the version compiled using the same compiler that The MathWorks uses to compile MATLAB and Simulink. The following table lists all the libraries shipped with Link for ModelSim. The default libraries for each platform are in bold text.

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

Using an Alternative Library

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

For example, if you want to use a nondefault library:

1 Copy the system libraries from the MATLAB installation (found in *matlabroot*/sys/os/*platform*) to the machine with ModelSim (where *matlabroot* is your MATLAB installation and *platform* is one of the above architecture, e.g., linux32).

2 Modify the LD_LIBRARY_PATH environment variable to add the path to the system libraries that were copied in step 1.

Example: Link for ModelSim Alternative Library Using vsim. In this example, you run the 32-bit Solaris version of ModelSim 6.1f on the same 64-bit Solaris machine which is running MATLAB. Because you want to incorporate some SystemC designs, you are using the Link for ModelSim version compiled with GCC 3.3. You can download the appropriate version of GCC with its associated system libraries from Mentor Graphics, instead of using the default library version compiled with SunPro 11.

In MATLAB:

```
>> currPath = getenv('PATH');
>> currLdPath = getenv('LD_LIBRARY_PATH');
>> setenv('PATH', ['/tools/modelsim-6.1e/bin:' currPath]);
>> setenv('LD_LIBRARY_PATH', ['/tools/mtigcc/gcc-3.3-sunos510/lib:' currLdPath]);
>> setenv('MTI_VCO_MODE', '32');
>> vsim('tclstart', { 'vlib work', 'vcom inverter.vhd', 'vsimulink inverter' }, ...
        'libfile', 'liblfmhdls_gcc33');
```

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

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

```
vsim> exec ldd /path/to/liblfmhdls_gcc33.so
#       librt.so.1 =>       /lib/librt.so.1
#       libpthread.so.1 => /lib/libpthread.so.1
#       libstdc++.so.5 =>  /tools/mtigcc/gcc-3.3-sunos510/lib/libstdc++.so.5
#       libm.so.2 =>       /lib/libm.so.2
#       libgcc_s.so.1 =>   /tools/mtigcc/gcc-3.3-sunos510/lib/libgcc_s.so.1
#       libaio.so.1 =>     /lib/libaio.so.1
#       libmd5.so.1 =>     /lib/libmd5.so.1
#       libc.so.1 =>       /lib/libc.so.1
#       /platform/SUNW,Sun-Blade-1000/lib/libmd5_psr.so.1
#       /platform/SUNW,Sun-Blade-1000/lib/libc_psr.so.1
```

Example: Link for ModelSim Alternative Library Using System Shell.
This example shows how to load a ModelSim session by explicitly specifying
the Link for ModelSim library (either the default or one of the alternatives).
By explicitly using a system shell, you can execute this example on the same
machine as MATLAB, on a different machine, and even on a machine with a
different operating system.

In this example, you are running the 64-bit Linux version of QuestaSim 6.2c.
It does not matter which machine is running MATLAB. Instead of using the

Link for ModelSim default library version compiled with GCC 4.1.1, you are using the version compiled with GCC 4.0.2. You can download the appropriate version of GCC with its associated system libraries from Mentor Graphics.

In a csh-compatible system shell:

```
csh> setenv PATH /tools/modelsim-6.2c/bin:${PATH}
csh> setenv LD_LIBRARY_PATH /tools/mtigcc/gcc-4.0.2-linux_x86_64/lib64:${LD_LIBRARY_PATH}
csh> setenv MTI_VCO_MODE 64
csh> vlib work
csh> vcom +acc+inverter inverter.vhd
csh> vsim +acc+inverter -foreign
 "matlabclient /tools/matlab-7b/toolbox/modelsim/linux64/liblfmhdlc_gcc40.so" inverter.vhd
```

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

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

Starting ModelSim for Use with Link for ModelSim

In this section...

Starting ModelSim from MATLAB

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

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

```
>> vsim
```

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

See "vsim Examples" on page 1-21 for examples of using `vsim` with various property/name value pairs and other parameters.

When you specify a communication mode using `vsim`, the `vsim` function applies the specified communication mode to all invocations of MATLAB or Simulink from ModelSim.

vsim Examples

The following example changes the directory location to `VHDLproj` and then calls the function `vsim`. Because the command line omits the `'vsimdir'` and `'startupfile'` properties, `vsim` creates a temporary DO file. The `'tclstart'` property specifies Tcl commands that load and initialize the ModelSim simulator for test bench instance `modsimrand`.

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

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

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

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

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

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

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

Or

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

Starting ModelSim from a Shell

To start ModelSim from a shell and include the Link for ModelSim libraries, you need to first run the configuration script. See "Using the Configuration and Diagnostic Script for UNIX/Linux" on page 1-11.

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

```
% vsim -f matlabconfigfile entity
```

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

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

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

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

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

where path is the path to this particular Link for ModelSim library. See "Using the Link for ModelSim Libraries" on page 1-16 to find the correct library name for your machine.

Note You can also issue this exact same command from inside of ModelSim.

Workflow for Using Link for ModelSim with MATLAB

The following diagram illustrates the steps necessary to create and run a MATLAB test bench or component session.
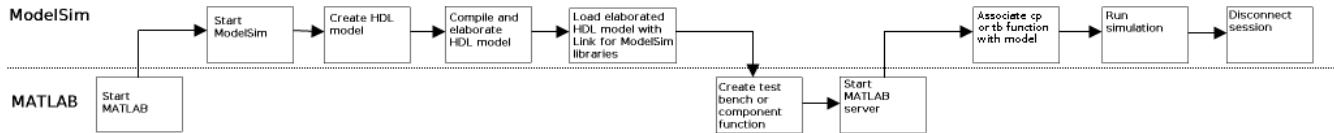
This diagram assumes you are using the configuration and diagnostic script (either UNIX/Linux or Windows) and starting ModelSim outside of MATLAB.

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

Workflow for Using Link for ModelSim with Simulink
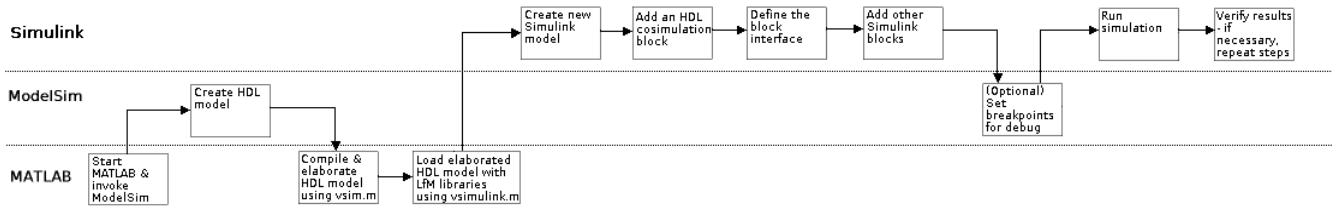
This diagram assumes you are using the configuration and diagnostic script (either UNIX/Linux or Windows) and starting ModelSim outside of MATLAB.

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

Learning More About Link for ModelSim

In this section...
"Documentation Overview" on page 1-26
"Online Help" on page 1-27
"Demos and Tutorials" on page 1-27

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, "Linking MATLAB to ModelSim"	Explains how to code HDL models and MATLAB functions for Link for ModelSim MATLAB applications. Provides details on how the Link for ModelSim interface maps HDL data types to MATLAB data types and vice versa. Explains how to start and control ModelSim and MATLAB test bench sessions.
Chapter 3, "Linking Simulink to ModelSim"	Explains how to use ModelSim and Simulink for cosimulation modeling.
Chapter 4, "MATLAB Functions — Alphabetical List"	Describes Link for ModelSim functions for use with MATLAB.
Chapter 5, "ModelSim Commands — Alphabetical List"	Describes Link for ModelSim commands for use with ModelSim.
Chapter 6, "Simulink Blocks — Alphabetical List"	Describes Link for ModelSim blocks for use with Simulink.

Appendix B, "Link for ModelSim Machine Configuration Requirements"	Explains the machine configurations permissible when linking the HDL simulator to MATLAB or Simulink
Appendix C, "ADMS Support"	Describes cosimulation support for digital blocks within the ADvanceMS (ADMS) environment.
Appendix D, "TCP/IP Socket Communication"	Provides information for choosing TCP/IP socket ports.

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

  ```
  doc modelsim
  ```

  or

  ```
  help modelsim
  ```

  at the MATLAB command prompt.

- Block reference pages accessible through the Simulink interface.

Demos and Tutorials

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. You can find the Link for ModelSim demos with the online documentation. To access demos, type at the MATLAB command prompt:

```
>> demos
```

Select **Links and Targets > Link for ModelSim** from the navigational pane.

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

- "MATLAB and ModelSim Tutorial" on page 2-58
- "Simulink and ModelSim Tutorial" on page 3-74
- "toVCD Block Tutorial" on page 3-92

2

Linking MATLAB to ModelSim

MATLAB — ModelSim Workflow

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

In MATLAB...	In ModelSim...
1 Start MATLAB and invoke ModelSim (see "Starting ModelSim for Use with Link for ModelSim" on page 1-21)	
	2 Create the HDL model.
	3 Compile and elaborate the HDL model.
	4 Load elaborated HDL model with Link for ModelSim libraries. See "Loading an HDL Design for Verification" on page 2-11.

In MATLAB...	In ModelSim...
5 Create test bench or component function (see "Coding a Link for ModelSim MATLAB Application" on page 2-4). **6** Start the server. See "Starting the MATLAB Server" on page 2-51.	
	7 Use `matlabcp`, `matlabtb`, or `matlabtbeval` to associate the function you wrote in step 5 with a component or entity of the loaded model currently in ModelSim (see "Associating a MATLAB Link Function with an HDL Model" on page 2-37). For additional scheduling and communication options, see "Scheduling Options for a Link Session" on page 2-40. See also the reference pages for `matlabcp`, `matlabtb`, and `matlabtbeval`. **8** Run the simulation. **9** Disconnect the session by using `nomatlabtb`.

Coding a Link for ModelSim MATLAB Application

Overview

Link for ModelSim provides an interface for verifying and visualizing ModelSim HDL modules within the MATLAB environment. To apply the interface, you need to code an HDL model and a MATLAB function that can share data with the HDL model. This chapter discusses the programming, interfacing, and scheduling conventions for MATLAB functions that communicate with ModelSim.

Link for ModelSim supports two types of MATLAB functions that interface to HDL modules:

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

  The syntax of a MATLAB test bench function is

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

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

  The syntax of a MATLAB component function is

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

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

As you can see, the programming, interfacing, and scheduling conventions for test bench functions and MATLAB component functions are almost identical. Most of this chapter focuses on test bench functions, but in general all operations can be performed on and with both link functions.

Refer to Appendix B, "Link for ModelSim Machine Configuration Requirements" for valid machine configurations.

Process for Coding a Link for ModelSim MATLAB Application

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

1 Code the HDL model for MATLAB verification (see "Coding HDL Modules for MATLAB Verification" on page 2-7).

2 Compile and debug the HDL model (see "Compiling and Debugging the HDL Model" on page 2-11).

3 Code the required MATLAB test bench or MATLAB component functions (see "Compiling and Debugging the HDL Model" on page 2-11).

The following figure shows how a MATLAB function wraps around and communicates with ModelSim during a test bench simulation session.
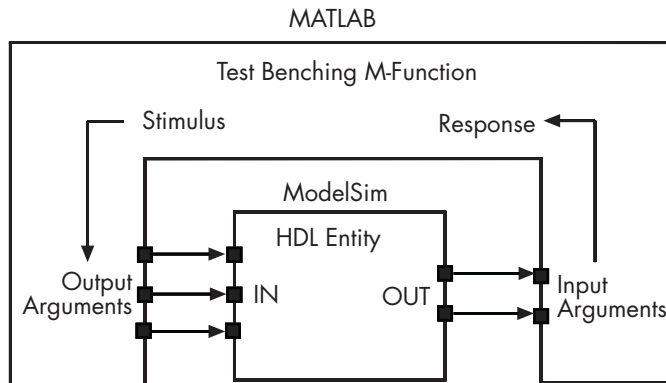
The following figure shows how a MATLAB component function is wrapped around by and communicates with ModelSim during a component simulation session.
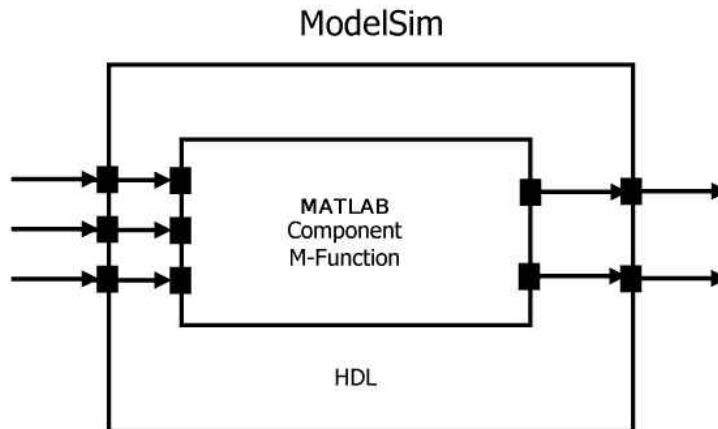
When linked with MATLAB, ModelSim functions as the client, MATLAB as the server. The following figure shows a multiple-client scenario connecting to the server at TCP/IP socket port 4449.
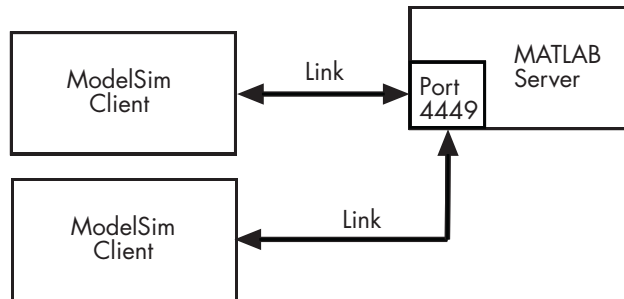
The MATLAB server can service multiple simultaneous ModelSim sessions and HDL entities. However, you should adhere to recommended guidelines to ensure the server can track the I/O associated with each entity and session. The MATLAB server, which you start with the supplied MATLAB function `hdldaemon`, waits for connection requests from instances of ModelSim running on the same or different computers. When the server receives a request, it executes the specified MATLAB function you have coded to perform tasks on behalf of an entity in your HDL design. Parameters that you specify when you start the server indicate whether the server establishes shared memory or TCP/IP socket communication links.

Coding HDL Modules for MATLAB Verification

The most basic element of communication in the Link for ModelSim interface is the HDL module. The interface passes all data between ModelSim and MATLAB as port data. Link for ModelSim works with any existing VHDL entity or Verilog module. However, when coding an HDL module that is targeted for MATLAB verification, you should consider its name, the types of data to be shared between the two environments, and the direction modes. The following sections cover these topics:

- "Choosing an HDL Module Name" on page 2-8

- "Specifying Port Direction Modes" on page 2-8

- "Specifying Port Data Types" on page 2-8

- "Sample VHDL Entity Definition" on page 2-10

- "Compiling and Debugging the HDL Model" on page 2-11

- "Loading an HDL Design for Verification" on page 2-11

Choosing an HDL Module Name

Although not required, when naming the HDL module, consider choosing a name that also can be used as a MATLAB function name. (Generally, naming rules for VHDL or Verilog and MATLAB are compatible.) By default, Link for ModelSim assumes that an HDL module and its simulation function share the same name. See .

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

Specifying Port Direction Modes

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

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

Specifying Port Data Types

This section describes how to specify data types compatible with MATLAB for ports in your VHDL or Verilog modules. For details on how Link for ModelSim converts data types for the MATLAB environment, see "Performing Data Type Conversions" on page 2-19.

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

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

- `STD_LOGIC`, `STD_ULOGIC`, `BIT`, `STD_LOGIC_VECTOR`, `STD_ULOGIC_VECTOR`, and `BIT_VECTOR`

- `INTEGER` and `NATURAL`

- `REAL`

- `TIME`

- Enumerated types, including user-defined enumerated types and `CHARACTER`

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

Note Link for ModelSim does not support VHDL extended identifiers for

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

Basic identifiers for VHDL are supported.

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

- `reg`

- `integer`

- `wire`

Note Link for ModelSim does not support Verilog escaped identifiers for port and signal names used in cosimulation. Simple identifiers for Verilog are supported.

Sample VHDL Entity Definition

The sample VHDL code fragment below defines the entity `decoder`. By default, the entity is 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 Link for ModelSim converts data of standard logic scalar and array types for use in the MATLAB environment, see "Performing Data Type Conversions" on page 2-19.

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

Compiling and Debugging the HDL Model

After you create or edit your VHDL or Verilog 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
```

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

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

For more examples, see the Link for ModelSim tutorials. For details on using the ModelSim compiler, see the ModelSim documentation.

Loading an HDL Design for Verification

After you start ModelSim from MATLAB with a call to `vsim`, load an instance of a VHDL entity or Verilog module for verification with the ModelSim command `vsimmatlab`. At this point, it is assumed that you have coded and compiled your HDL model as explained in "Coding a Link for ModelSim MATLAB Application" on page 2-4. Issue the ModelSim command `vsimmatlab` for each instance of an entity or module in your model that you want to cosimulate. For example:

```
vsimmatlab work.modsimrand
```

This command loads the Link for ModelSim library, opens a simulation workspace for `modsimrand`, and displays a series of messages in the ModelSim command window as the simulator loads the entity.

Coding MATLAB Link Functions

When coding a MATLAB function that is to verify or visualize an HDL 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:

- "Process for Coding MATLAB Link Functions" on page 2-12
- "Defining Link Functions and Link Function Parameters" on page 2-13
- "Performing Data Type Conversions" on page 2-19
- "Sample MATLAB Test Bench Function" on page 2-28

Process for Coding MATLAB Link Functions

To code a MATLAB link function that is to verify or visualize an HDL model,

1 Learn the syntax for a MATLAB link function (see "Defining Link Functions and Link Function Parameters" on page 2-13).

2 Understand how Link for ModelSim converts VHDL entity or Verilog module data for use in the MATLAB environment (see "Performing Data Type Conversions" on page 2-19).

3 Consider naming the MATLAB function with the name of the VHDL entity or Verilog module the function is to test (see "Choosing an HDL Module Name" on page 2-8).

4 Define expected parameters in the function definition line (see "Defining Link Functions and Link Function Parameters" on page 2-13).

5 Determine the types of port data being passed into the function (see "Defining Link Functions and Link Function Parameters" on page 2-13).

6 Extract and, if appropriate for the simulation, apply information received in the portinfo structure (see "Gaining Access to and Applying Port Information" on page 2-17).

7 Convert data for manipulation in the MATLAB environment, as necessary (see "Converting Data to Send to MATLAB" on page 2-19).

8 Convert data that needs to be returned to ModelSim (see "Converting Data for Return to ModelSim" on page 2-25).

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

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

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

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

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

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. The Tcl commands direct compilation and simulation of the model

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

Defining Link Functions and Link Function Parameters

The syntax of a MATLAB component function is

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

The syntax of a MATLAB test bench function is

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

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

Parameter	Test Bench	Component
iport	*Output* Structure that forces (by deposit) values onto signals connected to output ports of the associated VHDL entity or Verilog module.	*Input* Structure that receives signal values from the input ports defined for the associated VHDL entity or Verilog module at the time specified by tnow
tnext	*Output, 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 matlabtb or matlabtbeval command.	*Output, 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.
oport	*Input* Structure that receives signal values from the input ports defined for the associated VHDL entity or Verilog module at the time specified by tnow	*Output* Structure that forces (by deposit) values onto signals connected to output ports of the associated VHDL entity or Verilog module.

Parameter	Test Bench	Component
tnow	*Input* Receives the simulation time at which the MATLAB function is called. By default, time is represented in seconds. For more information see "Scheduling Options for a Link Session" on page 2-40.	Same as test bench
portinfo	*Input* 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 or Verilog module. For each port, the portinfo structure passes information such as the port's type, direction, and size. You can use the port information to create a generic MATLAB function that operates differently depending on the port information supplied at startup. For more information on port data, see "Gaining Access to and Applying Port Information" on page 2-17.	Same as test bench

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

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();
```

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

Oscfilter Function Example. In the Oscillator demo, 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 component function.

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

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

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

`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 ──────────→ ┌──────────────┐ ──→ oport.filter1x_out
                          │ osc_filter.vhd│ ──→ oport.filter4x_out
                          └──────────────┘ ──→ oport.filter8x_out

Gaining Access to and Applying Port Information. Link for ModelSim passes information about the entity or module under test in the `portinfo` structure. The `portinfo` structure is passed as the third argument to the function. It is passed only in the first call to your MATLAB function. The information passed in the `portinfo` structure is useful for validating the entity or module under simulation. 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 or Verilog module.

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

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

HDL Port Information

Field...	Can Contain...	Which...	And Applies to...
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 a bidirectional 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

HDL Port Information (Continued)

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

To use portinfo in your MATLAB function to verify port data, perform the following tasks:

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
```

Example

In the Oscillator demo, all I/O ports for the osc_filter entity are defined as

```
STD_LOGIC_VECTOR(21 DOWNTO 0)
```

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.

Performing Data Type Conversions

To successfully use Link for ModelSim with ModelSim and MATLAB or Simulink, you need to understand the data type conversions that Link for ModelSim performs in order to transmit and receive data between HDL modules and the MATLAB environment.

Converting Data to Send to MATLAB. If your ModelSim application needs to send data to a MATLAB function, it may be necessary for you to first convert the data to a type supported by MATLAB and Link for ModelSim. This section explains the following types of conversions:

• VHDL Data Type Conversions

• Verilog Data Type Conversions

VHDL Data Type Conversions

Link for ModelSim 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

2-19

may also need to handle differences between the array indexing conventions employed by VHDL and MATLAB (see below).

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 Link for ModelSim converts supported VHDL data types to MATLAB types based on whether the type is scalar or array.

VHDL-to-MATLAB Data Type Conversions

VHDL Types...	As Scalar Converts to...	As Array Converts to...
STD_LOGIC, STD_ULOGIC, and BIT	A character that matches the character literal for the desired logic state.	
STD_LOGIC_VECTOR, STD_ULOGIC_VECTOR, BIT_VECTOR, SIGNED, and UNSIGNED		A column vector of characters (as defined in VHDL Conversions for ModelSim on page 2-25) 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 Array Converts to...
TIME	Type `double` for time values in seconds and type `int64` for values representing simulator time increments (see the description of the `'time'` option in "Starting the MATLAB Server" on page 2-59).	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

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

In HDL, a multidimensional array declared as:

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

has a memory layout as follows:

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

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

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

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

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

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

Verilog Data Type Conversions

Link for ModelSim converts Verilog module data to types that apply in the MATLAB environment. To program a MATLAB function for a Verilog model, you must understand the type conversions required by your application.

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 Link for ModelSim converts supported Verilog data types to MATLAB types. Only scalar data types are supported for Verilog.

Verilog-to-MATLAB Data Type Conversions

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

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 `mvl2dec` function to convert the data to an unsigned decimal value. For example: `uval = mvl2dec(oport.val')` This example assumes the standard logic or bit vector is composed of the character literals `'1'` and `'0'` only. These are the only two values that can be converted to an integer equivalent.
Convert a standard logic or bit vector to a signed integer	Use the following application of the `mvl2dec` function to convert the data to a signed decimal value. For example: `suval = mvl2dec(oport.val')-2^length(oport.val);` This example assumes the standard logic or bit vector is composed of the character literals `'1'` and `'0'` only. These are the only two values that can be converted to an integer equivalent.
Test port values of VHDL 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'`.

Example

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

```
InDelayLine(1) = InputScale * mvl2dec(iport.osc_in')/2^(Nbits-1);
```

The `mvl2dec` 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 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 Link for ModelSim. The following tables list circumstances under which such conversions are required for VHDL and Verilog.

VHDL 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', '0', '1', 'Z', 'W', 'L', 'H', or '-'. For BIT, the character can be '0' or '1'. For example:
	
	`iport.s1 = 'X'; %STD_LOGIC`
	`iport.bit = '1'; %BIT`

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:
	
	`iport.s1v = 'X10ZZ'; %STD_LOGIC_VECTOR`
	`iport.bitv = '10100'; %BIT_VECTOR`
	`iport.uns = dec2mvl(10,8); %UNSIGNED, 8 bits`

VHDL Conversions for ModelSim (Continued)

To Return Data to an IN Port of Type...	Then...

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:
	
	```PORT (sta : OUT ARRAY(1 TO 2) OF BIT_VECTOR(1 TO 8););...sta(2)(7) <= '1'```
	
	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);```

VHDL Conversions for ModelSim (Continued)

To Return Data to an IN Port of Type...	Then...

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 2-51), 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`
	`                         %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 2-17). 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 dec2mvl function to convert the integer. For example: `oport.slva =dec2mvl([23 99],8)';` This example converts two integers to a 2-element array of standard logic vectors consisting of 8 bits.

Verilog Conversions for ModelSim

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

Sample MATLAB Test Bench Function

This section uses a sample MATLAB function to identify sections of a MATLAB test bench function required by Link for ModelSim. The example uses a VHDL entity and MATLAB function code drawn from the decoder portion of the Manchester Receiver demo. For the complete VHDL and M-code listings, see the following files:

- *matlabroot*\toolbox\modelsim\modelsimdemos\vhdl\manchester\decoder.vhd

- *matlabroot*\toolbox\modelsim\modelsimdemos\manchester_decoder.m

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

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

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

1 Specify the MATLAB function name and required parameters.

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

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

See "Defining Link Functions and Link Function Parameters" on page 2-13.

The function declaration performs the following actions:

- Names the function. This declaration names the function manchester_decoder, which differs from the entity name decoder. Because the names differ, the function name must be specified explicitly later when the entity is initialized for verification with the matlabtb or matlabtbeval ModelSim command. See .

- 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. See "Defining Link Functions and Link Function Parameters" on page 2-13.

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.

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

Input Signals

Output Signals

iport.isum (5) →
iport.qsum (5) →

decoder.vhd

→ oport.adj (2)
→ oport.dvalid(1)
→ oport.odata(1)

For more information on the required MATLAB link function parameters, see "Defining Link Functions and Link Function Parameters" on page 2-13.

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

Link for ModelSim converts HDL data types to comparable MATLAB data types and vice versa. As you develop your MATLAB function, you must know the types of the data that it receives from ModelSim and needs to return to ModelSim.

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

VHDL Example Port Definitions

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

VHDL Example Port Definitions (Continued)

Port	Direction	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 "Performing Data Type Conversions" on page 2-19.

3 Set up any required timing parameters.

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

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

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

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

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

The two calls to `mvl2dec` convert the binary data that the MATLAB function receives from the entity's output ports, `adj`, `dvalid`, and `odata` to unsigned decimal values that MATLAB can compute. The function converts the 2-bit transposed vector `oport.adj` to a decimal value in the range 0 to 4 and `oport.dvalid` and `oport.odata` to the decimal value 0 or 1.

"Performing Data Type Conversions" on page 2-19 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 = dec2mvl(isum,5);
   iport.qsum = dec2mvl(qsum,5);
else
   iport.isum = dec2mvl(isum,5);
end
```

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

"Converting Data for Return to ModelSim" on page 2-25 provides a summary of the types of data conversions to consider when returning data to ModelSim.

M-Function Example: manchester_decoder.m

```
function [iport,tnext] = manchester_decoder(oport,tnow,portinfo)
% MANCHESTER_DECODER  Test bench for VHDL 'decoder'
%  [IPORT,TNEXT]=MANCHESTER_DECODER(OPORT,TNOW,PORTINFO) -
%    Implements a test of the VHDL decoder entity which is part
%    of the Manchester receiver demo.  This test bench plots
%    the IQ mapping produced by the decoder.
%
%      iport              oport
%            +-----------+
% isum -(5)->|           |-(2)-> adj
% qsum -(5)->| decoder   |-(1)-> dvalid
%            |           |-(1)-> odata
%            +-----------+
%
%   isum - Inphase Convolution value
%   qsum - Quadrature Convolution value
%   adj  - Clock adjustment ('01','00','10')
%   dvalid - Data validity ('1' = data is valid)
%   odata - Recovered data stream
%
% Adjust = 0 (00b), generate full 16 cycle waveform

%   Copyright 2003-2004 The MathWorks, Inc.
%   $Revision: 1.1.4.1 $  $Date: 2007/06/05 11:13:58 $

persistent isum;
persistent qsum;
%persistent ga;
persistent x;
persistent y;
persistent adj;
persistent data;
global testisdone;
% This useful feature allows you to manually
```

2-33

```
% reset the plot by simply typing: >manchester_decoder
tnext = [];
iport = struct();

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

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

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

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

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

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

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

```
firout1 = sum(Hresp{1}.*InDelayLine);
outputvalue = dec2mvl(floor(firout1*2^(Nbits-1)), Nbits);
outputvalue(find(outputvalue=='/')) = '1'; % fix negative numbers
oport.filter1x_out = outputvalue';
```

The `dec2mvl` 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.

Associating a MATLAB Link Function with an HDL Model

In this section...

Naming a MATLAB Link Function

You can name and specify a MATLAB link function however you like, so long as you adhere to MATLAB function and file naming guidelines. By default, Link for ModelSim assumes the name for a MATLAB function matches the name of the VHDL entity or Verilog module 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`.

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

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

Overview

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

Associating the HDL Module Component with the MATLAB Link Function

By default, Link for ModelSim assumes the name for a MATLAB function matches the name of the VHDL entity or Verilog module that the function verifies or visualizes. See .

In the Oscillator demo, the VHDL model instantiates a VHDL 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 modsimosc.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.

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

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

In the following example,

```
matlabcp u_osc_filter -mfunc oscfilter
```

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

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

Path Specifications for MATLAB Link Sessions with Verilog Top Level

- Path specification must start with a top-level module name.

- Path specification can include "." or "/" path delimiters, but cannot include a mixture.

- The leaf module or signal must match the HDL language of the top-level module.

The following are valid signal and module path specification examples:

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

The following are invalid signal and module path specification examples:

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

Path Specifications for MATLAB Link Sessions with VHDL Top Level

- Path specification may include the top-level module name but it is not required.

- Path specification can include "." or "/" path delimiters, but cannot include a mixture.

- The leaf module or signal must match the HDL language of the top-level module.

The following are valid signal and module path specification examples:

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

2-39

```
top.sub1.sub2
```

The following are invalid signal and module path specification examples:

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

Specifying TCP/IP Values

When providing TCP/IP information for a MATLAB link function, you can choose a TCP/IP port number, TCP/IP port alias or service name, or the value zero, indicating that the port is to be assigned by the operating system.

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.

Some valid TCP/IP specification examples follow:

Format	Example
Port number	4449
Port alias	matlabservice
Port number and remote hostname	4449@compa
Remote hostname and port number	compa:4449
Port alias and remote host Internet address	matlabservice@123.34.55.23

For example,

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

Scheduling Options for a Link Session

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

- Using the arguments to the `matlabcp` or `matlabtb` functions ("Scheduling Link Functions Using Link Function Parameters" on page 2-41)

- Inside the MATLAB m-function using the `tnext` parameter ("Scheduling Link Functions Using the tnext Parameter of an M-Function" on page 2-47)

You can schedule a MATLAB simulation function to execute

- At a time that the MATLAB function passes to ModelSim with the `tnext` parameter

- Based on a time specification that can include discrete time values, repeat intervals, and a stop time

- When a specified signal experiences a rising edge — changes from `'0'` to `'1'`

- When a specified signal experiences a falling edge — changes from `'1'` to `'0'`

- Based on a sensitivity list — when a specified signal changes state

Decide on a combination of options that best meet your test bench application requirements. For details on using the `tnext` parameter and information on setting other scheduling parameters, see "Scheduling Options for a Link Session" on page 2-40.

Scheduling Link Functions Using Link Function Parameters

By default, Link for ModelSim 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 Link for ModelSim 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` function.

After you decide on the controls you need to apply for a link session, you are ready to register callbacks for a specific MATLAB link session with one of the link functions. These functions

- Identify the instance of an entity or module in the HDL model being simulated and test benched

2-41

- 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, the matlabtb function can include parameters that control when
the MATLAB function executes.

You must specify at least one instance of a VHDL entity or Verilog module
in your HDL model. By default, the command establishes a shared memory
communication link and 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. See "Associating the HDL Module
Component with the MATLAB Link Function" on page 2-38.

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

Simulation Scheduling Options

To Specify MATLAB Function Execution...	Include...	Where...
At explicit times	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.

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

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 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.

> **Note** The port number or service name that you specify with `-socket` must match the port value returned by or specified with the call to `hdldaemon` that started the MATLAB server. If you need to verify the port number, issue a call to the `hdldaemon` function with `'status'` as follows:
>
> ```
> hdldaemon('status')
> HDLDaemon socket server is running on port 4795 with 0
>  connections
> ```
>
> This function call indicates that the server is using TCP/IP socket communication with socket port 4795 and is running with no connections. If a shared memory link is in use, the message will reflect that mode of communication.

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
```

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

You can control the callback timing of a MATLAB test bench function by using that function's tnext parameter. This parameter passes a time value to 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 s (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 ns'`
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.

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

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

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

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

Running MATLAB Link Sessions

Overview

Link for ModelSim offers flexibility in how you start and control an HDL model test bench or component session with MATLAB. A MATLAB link session is the application of a `matlabtb`, `matlabtbeval`, or `matlabcp` function. 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 the HDL simulator.

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 or component control.

Process for Running MATLAB Link Sessions

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

1 Place MATLAB link function on the MATLAB search path.

2 Check the MATLAB server's link status.

3 Start the MATLAB server.

4 Launch ModelSim for use with MATLAB.

5 Load an HDL model in ModelSim for simulation and verification with MATLAB.

6 Decide on how you want to schedule invocations of the MATLAB test bench function.

7 Register callbacks for the MATLAB link session.

8 Apply test bench stimuli.

9 Run and monitor the test bench session.

10 Restart simulator during a test bench session.

11 Stop a test bench session.

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

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

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

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

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

Starting the MATLAB Server

Start the MATLAB server as follows:

1 Start MATLAB.

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

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

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

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

Note The communication mode that you specify (shared memory or TCP/IP sockets) must match what you specify for the communication mode when you initialize the 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 D-2. For more information on establishing the ModelSim end of the communication link, see "Associating a MATLAB Link Function with an HDL Model" on page 2-37.

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.

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

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
```

See 'Link Status' "Description" on page 4-9 in the hdldaemon reference documentation for information on determining the mode of communication and the TCP/IP socket in use.

Starting ModelSim for Use with MATLAB

Start ModelSim directly from MATLAB by calling the MATLAB function vsim. See "Starting ModelSim for Use with Link for ModelSim" on page 1-21 for instructions on using vsim.

Applying Stimuli with the ModelSim force Command

After 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.
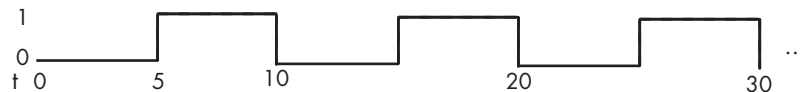
- Force the clk_en signal to 1 at 0 nanoseconds after the current simulation time.

- Forces the reset signal to 0 at 0 nanoseconds after the current simulation time.

Running a Link Session

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 HDL 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 Link 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 HDL 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 Link Session

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

1 Make 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.

MATLAB and ModelSim Tutorial

Tutorial Overview

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

Note This tutorial requires MATLAB, ModelSim, and Link for ModelSim.

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 2-58.

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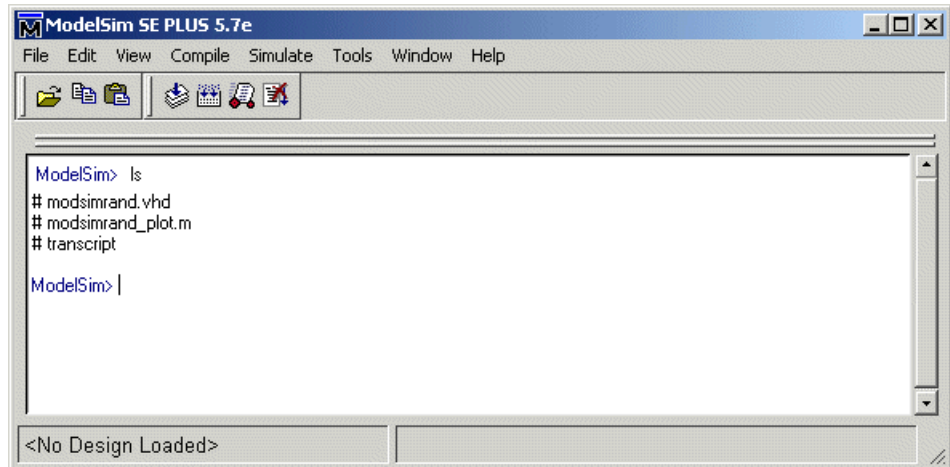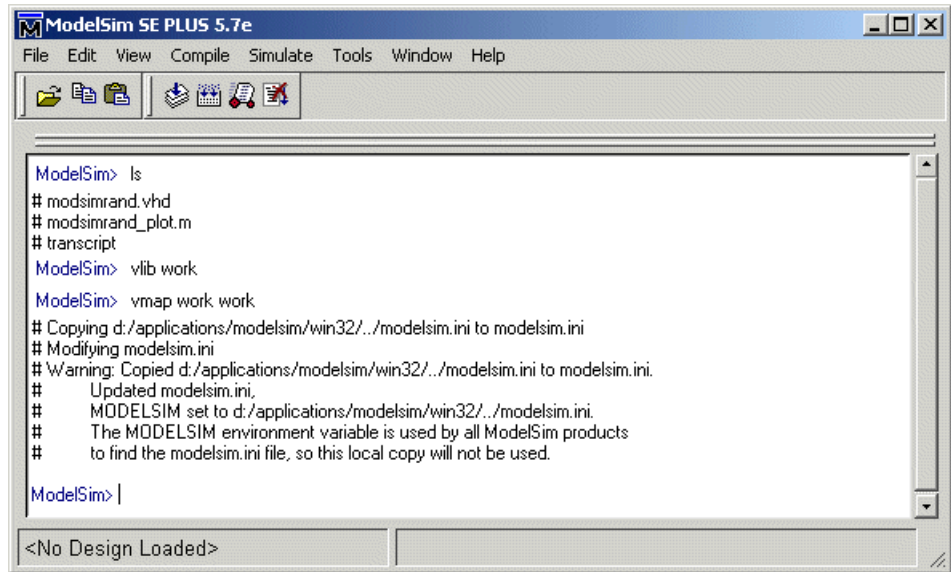
The command should list the files modsimrand.vhd, modsimrand_plot.m, and transcript.

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 HDL code. For this tutorial, open and examine the existing file modsimrand.vhd. This section highlights areas of code in modsimrand.vhd that are of interest for a ModelSim and MATLAB test bench:

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

```
ModelSim> edit modsimrand.vhd
```
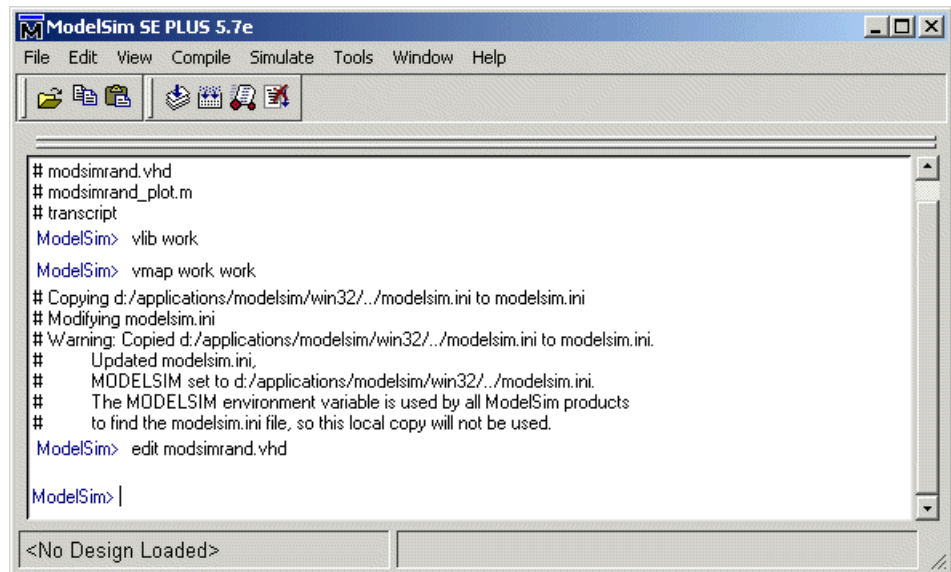
ModelSim opens its **edit** window and displays the VHDL code for
`modsimrand.vhd`.
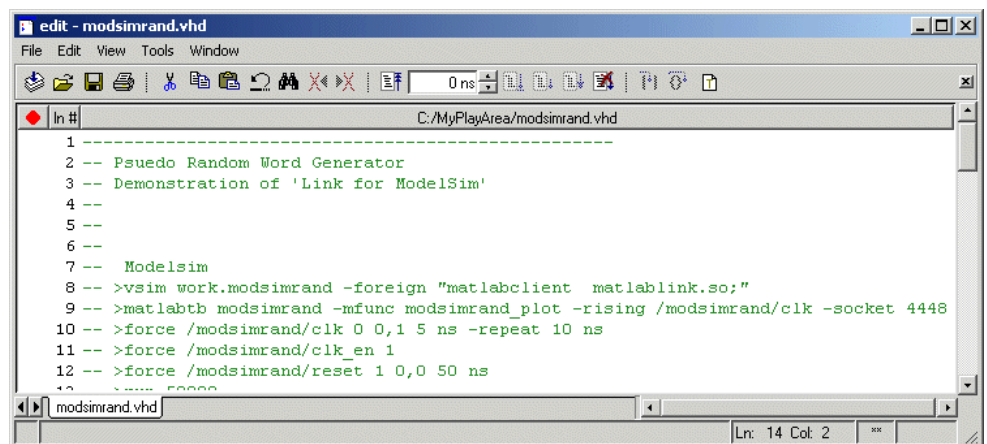
2 Search for ENTITY modsimrand. This line defines the VHDL entity
modsimrand:

```
ENTITY modsimrand IS
```

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

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

• By default, the MATLAB server assumes that the name of the MATLAB function that verifies the entity in the MATLAB environment is the same as the entity name. You have the option of naming the MATLAB function explicitly. However, if you do not specify a name, the server expects the function name to match the entity name. In this example, the MATLAB function name is modsimrand_plot and does not match.

• The entity must be defined with a PORT clause that includes at least one port definition. Each port definition must specify a port mode (IN, OUT, or INOUT) and a VHDL data type that is supported by Link for ModelSim. For a list of the supported types, see "Coding HDL Modules for MATLAB Verification" on page 2-7.

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

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

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

4 Close the ModelSim **edit** window.

Compiling the VHDL File

After you create or edit your VHDL source files, compile them. As part of this tutorial, compile modsimrand.vhd. One way of compiling the file is to click the 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.
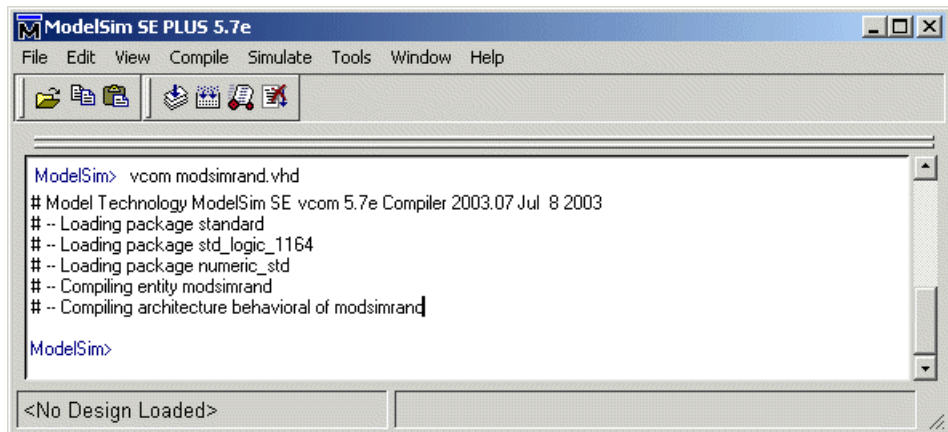
Loading the Simulation

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

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

```
ModelSim> vsimmatlab modsimrand
```

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

ModelSim displays a series of messages in the command window as it loads the entity's packages and architecture.
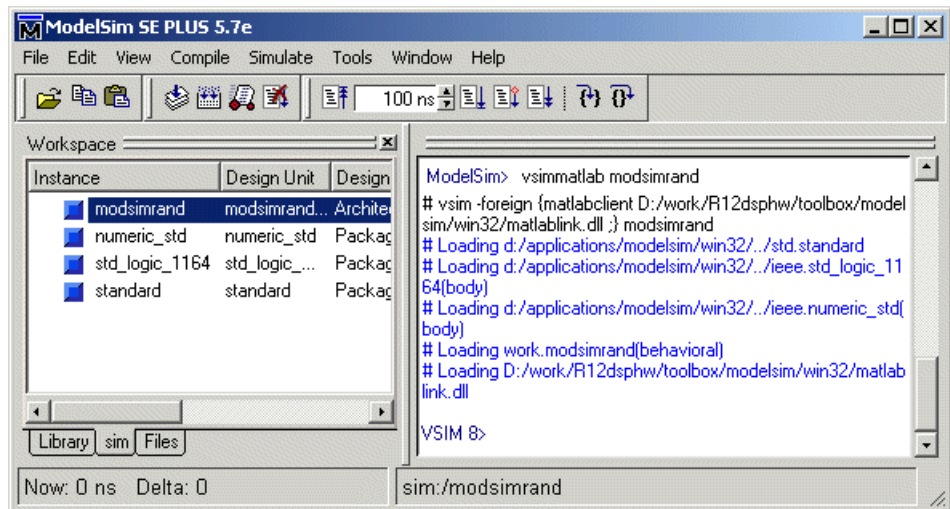
2 Initialize the simulator for verifying `modsimrand` with MATLAB. You initialize ModelSim by using the `matlabtb` or `matlabtbeval` ModelSim command. These commands define the communication link and a callback to a MATLAB function that executes in MATLAB on behalf of ModelSim. In addition, the `matlabtb` 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
```

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

Arguments in the command line specify the following:

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 HDL instance `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, Link for ModelSim invokes a MATLAB function that has the same name as the specified HDL instance. Thus, if the names are the same, you can omit the `-mfunc` option.

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

```
VSIM n> force /modsimrand/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

Link for ModelSim verifies HDL hardware in MATLAB as a function. 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 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, 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.

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

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

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

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

4 Make note of the data types of ports defined for the entity under simulation. Link for ModelSim 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`.

- Data of type `STD_LOGIC_VECTOR` consists of a column vector of characters with one bit per character.

- The interface converts scalar data of type `STD_LOGIC` to a character that matches the character literal for the corresponding enumerated type.

For more information on interface data type conversions, see "Performing Data Type Conversions" on page 2-19.

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. "Performing Data Type Conversions" on page 2-19 provides a summary of the types of data conversions to consider when coding your own MATLAB functions.

6 Browse through the rest of modsimrand_plot.m.

Running the Simulation

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

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

Running the Simulation for the First Time

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

1 Open ModelSim and MATLAB windows.

2 In MATLAB, verify the client connection by calling hdldaemon with the 'status' option:

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

This function returns a message indicating a connection exists:

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

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

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

---

3 Open modsimrand_plot.m in the MATLAB Edit/Debug window.

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

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

```
Vsim n> run 80000
```

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

ModelSim is now blocked and remains blocked until you explicitly unblock
it. While the simulation is blocked, note that MATLAB displays the data
that ModelSim passed to the MATLAB function in the **Workspace** window.
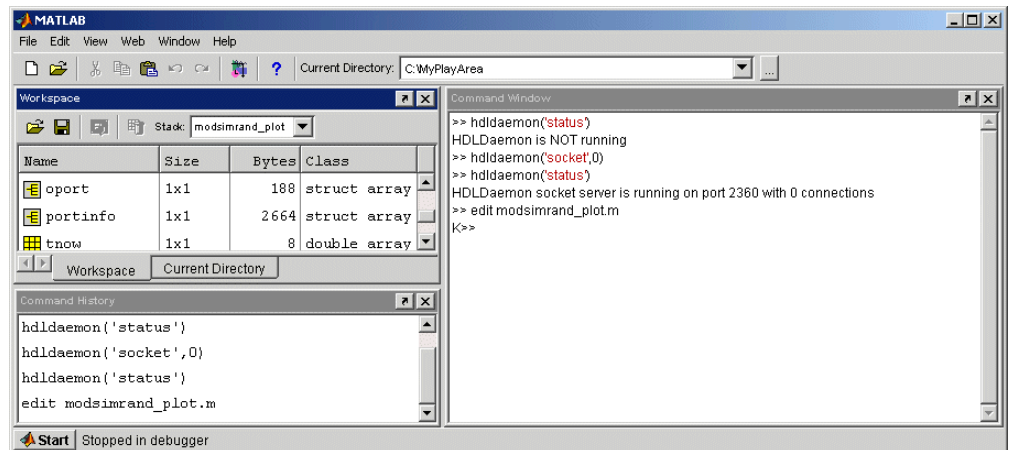
Note also that a ModelSim figure window opens. This window is used to
plot data generated by the simulation. Initially, it is empty.

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

7 Click **Debug > Continue** in the MATLAB Edit/Debug window. The next
time the breakpoint is reached, notice that portinfo is no longer visible
in the MATLAB workspace. This is because portinfo is passed in only on

2-71

the first function invocation. Also note that the value of `tnow` advances
from 0 to 5e-009.

8 Clear the breakpoint by clicking the red breakpoint marker.

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

The simulation runs to completion. As the simulation progresses, it plots
generated data in a figure window. When the simulation completes, the
figure window appears as shown below.
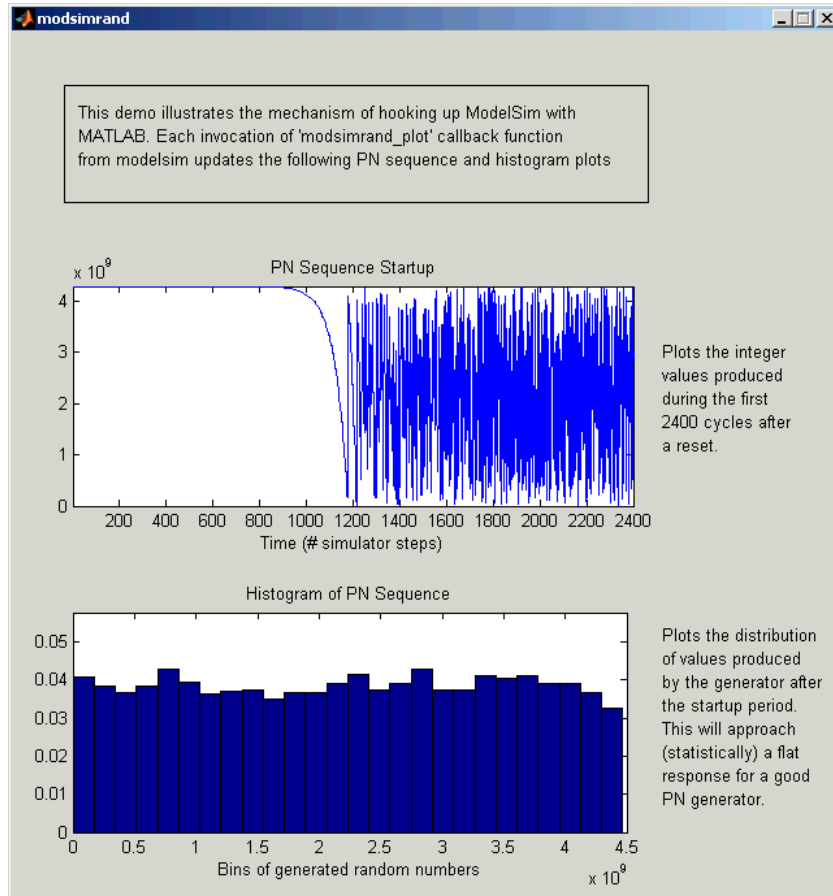
The simulation runs in MATLAB until it reaches the breakpoint that you just set. Continue the simulation/debugging session as desired.

Rerunning the Simulation

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

1 Close the figure window.

2 Restart the simulation with the following command:

```
VSIM n> restart
```

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

Note The **Restart** button clears the simulation context established by a 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> run 80000
```

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
```

3

Linking Simulink to ModelSim

Simulink — ModelSim Workflow

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

In MATLAB...	In ModelSim...	In Simulink...
1 Start MATLAB and invoke ModelSim (see "Starting ModelSim for Use with Link for ModelSim" on page 1-21)		
	2 Create the HDL model.	
3 Compile and elaborate the HDL model using `vsim`. Load elaborated HDL model with Link for ModelSim libraries. See "Loading an HDL Design for Verification" on page 2-11.		

In MATLAB...	In ModelSim...	In Simulink...
		4 Create a new Simulink model.
		5 Add an HDL Cosimulation block (see "Incorporating Hardware Designs Into a Simulink Model" on page 3-42).
		6 Define the block interface (see "Defining the Block Interface" on page 3-47).
		7 Add other Simulink blocks to complete the Simulink model.

In MATLAB...	In ModelSim...	In Simulink...
	8 (Optional) Set breakpoints for interactive HDL debug.	
		9 Run the simulation.
		10 Verify that the revised model runs as expected. If it does not, then:
		a Modify the VHDL or Verilog code and simulate it in ModelSim.
		b Determine whether you need to reconfigure the HDL Cosimulation block. If you do, repeat steps 7 and 10.
		11 Consider using a To VCD File block to verify cosimulation results.

Introduction to Cosimulation

In this section...

Creating a Hardware Model Design for Use in Simulink

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

- Will you start by developing an HDL 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, after the model runs as expected, replace the black boxes with HDL Cosimulation blocks?

- What other Simulink blocksets might apply to your application? Blocksets of particular interest for EDA applications include the Communications Blockset, Signal Processing Blockset, and Simulink Fixed Point.

- Will you set up HDL Cosimulation blocks as a subsystem in your model?

- What sample times will be used in the model? Will any sample times need to be scaled?

- Will you generate a Value Change Dump (VCD) file?

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

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

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

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

The HDL Cosimulation block (labeled VHDL Manchester Receiver) 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.

The Link for ModelSim HDL Cosimulation Block

The Link for ModelSim HDL Cosimuation Block links hardware components that are concurrently simulating in ModelSim to the rest of a Simulink model.

Two potential use cases follow:

- A single HDL Cosimulation block fits into the framework of a larger system-oriented Simulink model.

- The Simulink model is a collection of HDL Cosimulation blocks, each representing a specific hardware component.

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

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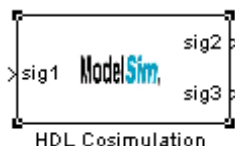
HDL Cosimulation

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

- **Ports Pane**: Block input and output ports that correspond to signals, including internal signals, of your HDL design, and an output sample time. See "Ports Pane" on page 6-3.
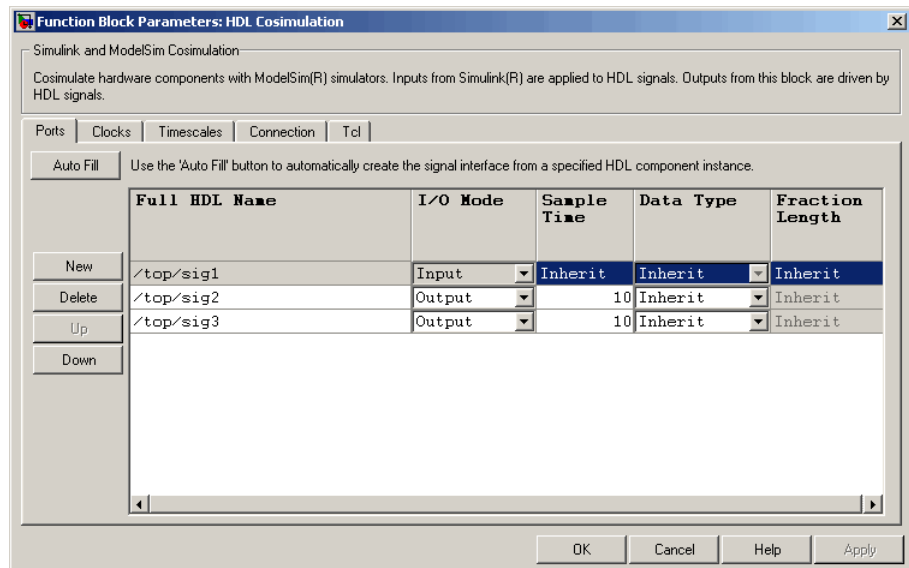
- **Connection Pane**: Type of communication and communication settings to be used for exchanging data between simulators. See "Connection Pane" on page 6-14.
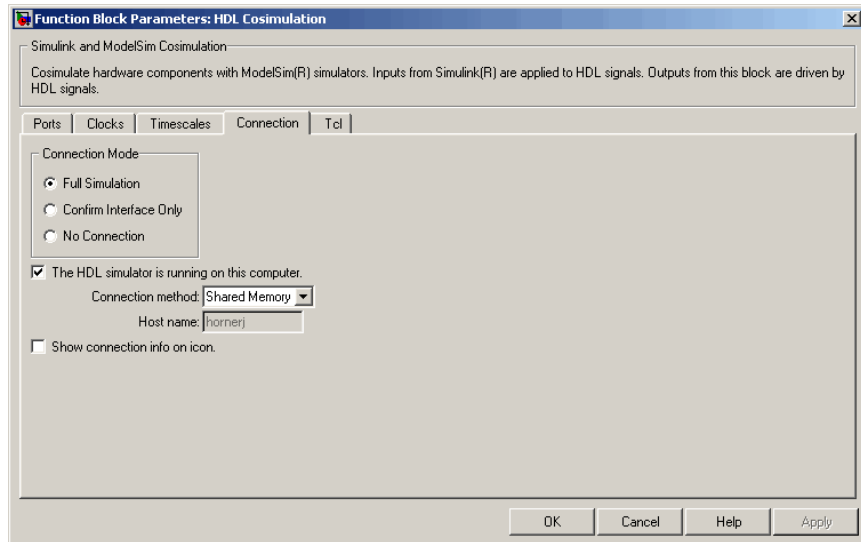
- **Timescales Pane**: Timing relationship between Simulink and ModelSim. See "Timescales Pane" on page 6-12.
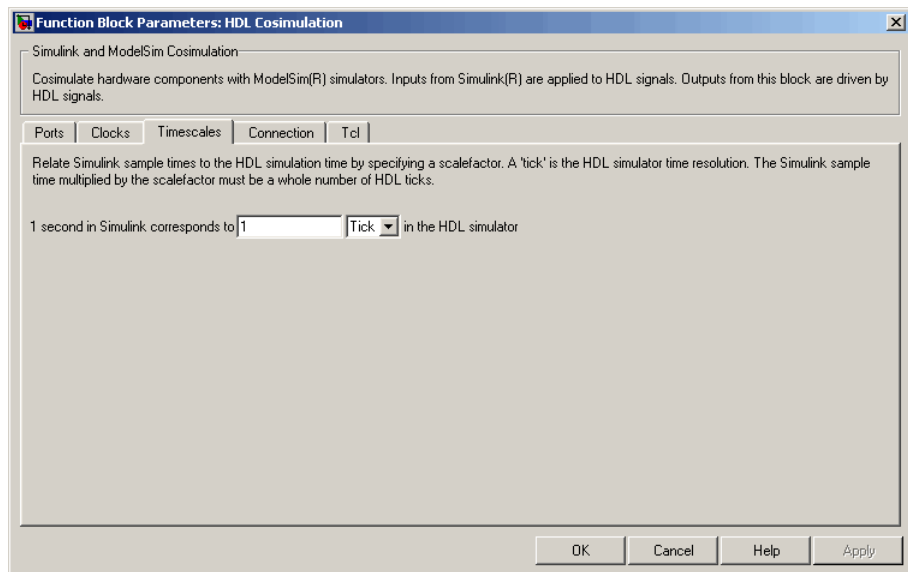
- **Clocks Pane**: Optional rising-edge and falling-edge clocks to apply to your model. See "Clocks Pane" on page 6-9.
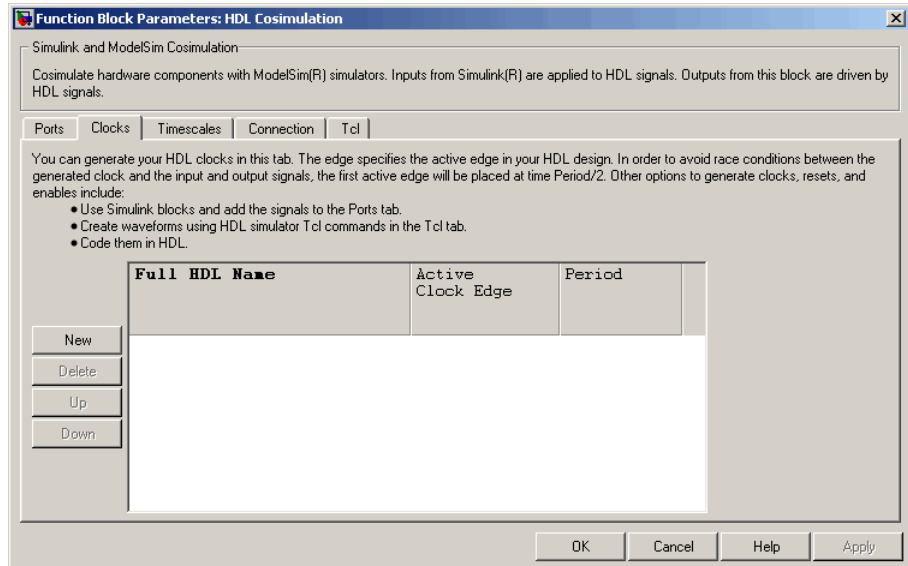
- **Tcl Pane**: Tcl commands to run before and after a simulation. See "Tcl Pane" on page 6-19.
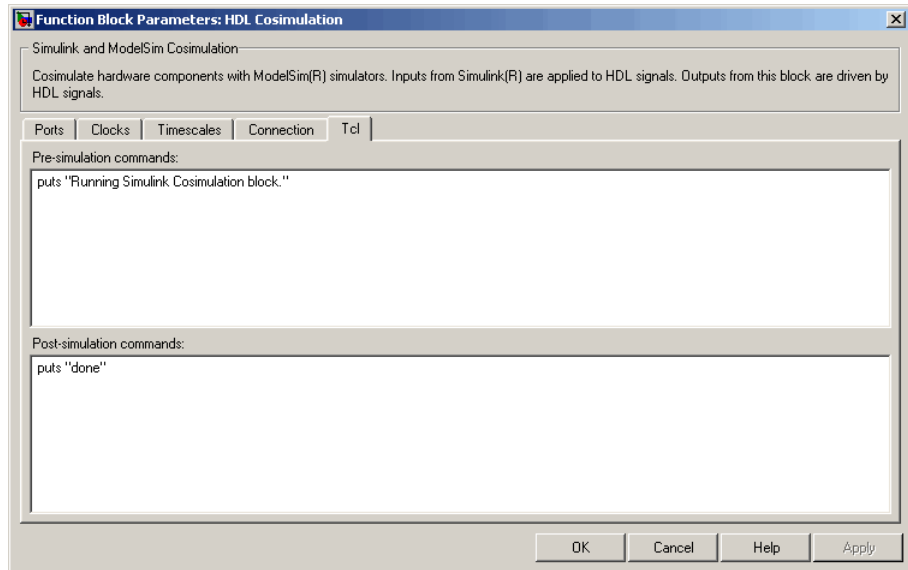
Note You must make sure that signals being used in cosimulation have read/write access (this is done through the HDL simulator – see product documentation for details). This rule applies to all signals on the **Ports**, **Clocks**, and **Tcl** panes.

Communicating Between ModelSim and Simulink

When linked with Simulink, ModelSim functions as the server, as shown in the following figure.
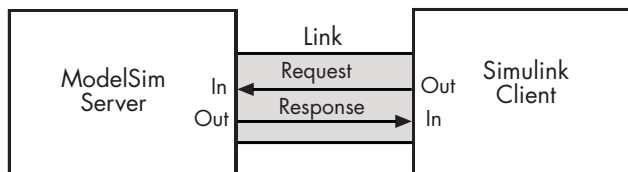
In this case, ModelSim responds to simulation requests it receives from cosimulation blocks in a Simulink model. You initiate a cosimulation session from Simulink. After a session is started, you can use Simulink and ModelSim

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.
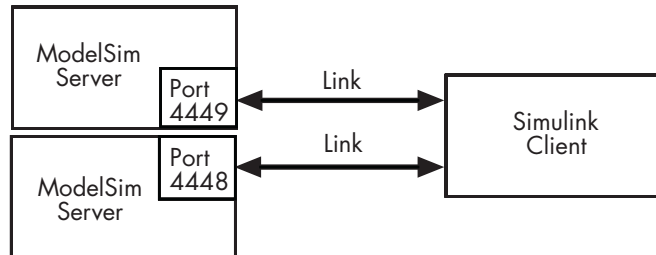
Preparing for Cosimulation

Overview

The Link for ModelSim HDL Cosimulation block serves as a bridge between the Simulink and ModelSim simulation domains. The block represents an HDL component model within Simulink. Using the block, Simulink writes (drives) signals to and reads signals from the HDL 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
- Representation of simulation time
- Handling multirate signals
- Handling Frame-based signals

- Clock signal latency

- Block simulation latency

- Interfacing with continuous time signals

- Setting Simulink configuration parameters

- Setting the communication link

- Starting the HDL simulator

How Simulink Drives Cosimulation Signals

Although you can connect the output ports of an HDL Cosimulation block to any signal in an HDL model hierarchy, you must use some caution when connecting signals to input ports. 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 an HDL 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 VHDL type STD_LOGIC) , Simulink applies the resolution function at each time step defined by the signal's Simulink sample rate. Depending on the other drivers, the Simulink value may or may not get applied. Furthermore, Simulink has no control over signal changes that occur between its sample times.

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

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

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, sampled Simulink time must be converted to ModelSim time (ticks) and ModelSim must run for the computed number of ticks.

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

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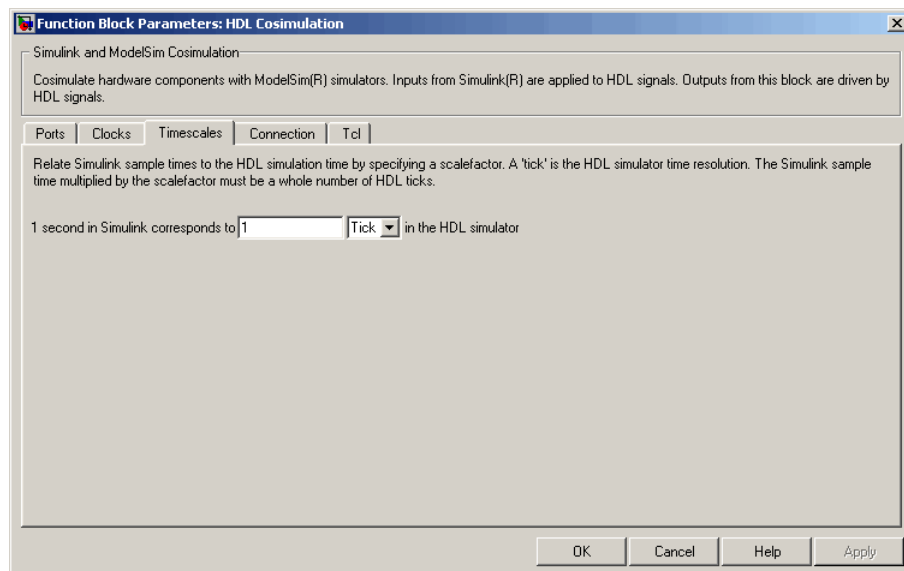
The **Timescales** pane defines a correspondence between one second of Simulink time and some quantity of ModelSim time. This quantity of ModelSim time can be expressed in one of the following ways:

- 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:

```
InTicks = N * tInSecs
```

where InTicks 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 HDL Cosimulation block parameters dialog.

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

3 Enter a scale factor in the text box on the left. The default scale factor is 1.

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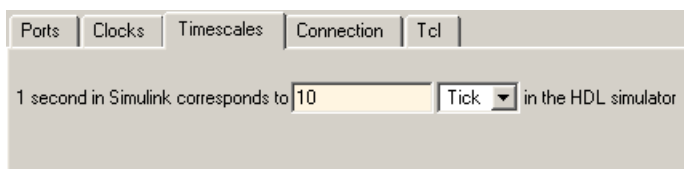
4 Click **Apply** to commit your changes.

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

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

- Similarly, if Simulink computes the sample time of an HDL Cosimulation block input port as *Tsi* seconds, new values will be deposited on the HDL 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.
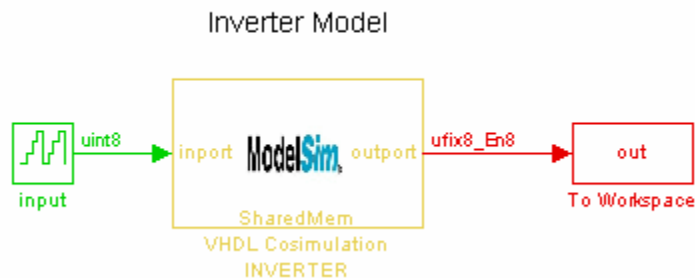
Inverter Model

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

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

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

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ARCHITECTURE behavioral OF inverter IS
BEGIN
  PROCESS(clk)
  BEGIN
    IF (clk'EVENT AND clk = '1') THEN
        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.
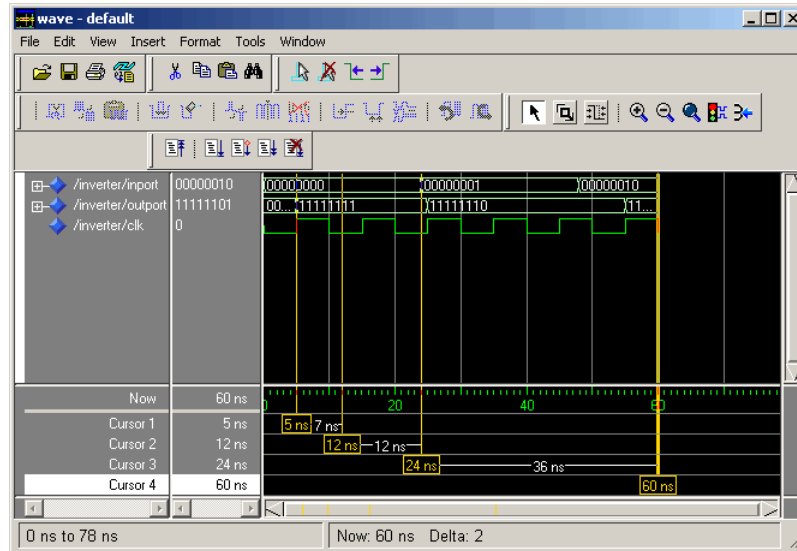
Now consider a cosimulation of the same model, this time configured with a scale factor of 100 in the **Timescales** pane.

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.
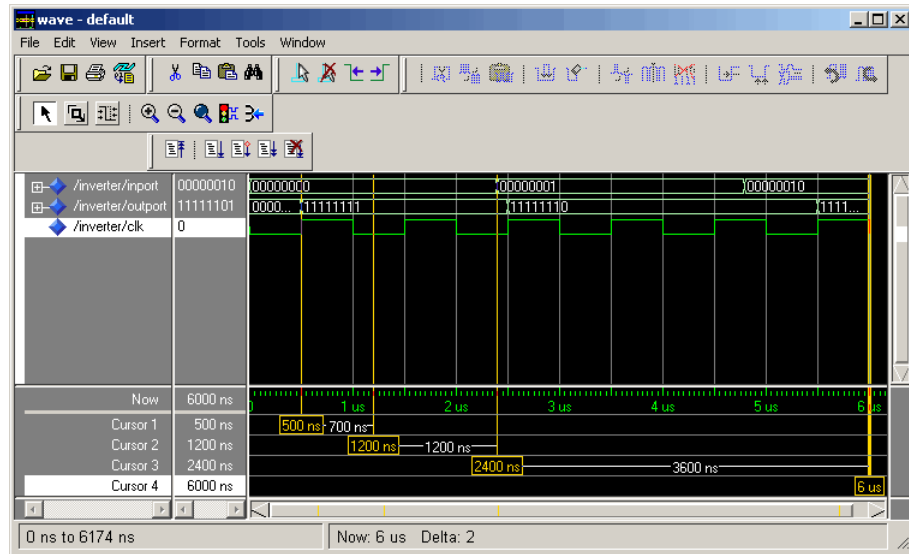
Absolute Timing Mode

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 HDL Cosimulation block parameters dialog.

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

3 Enter a scale factor in the text box on the left. The default scale factor is 1.

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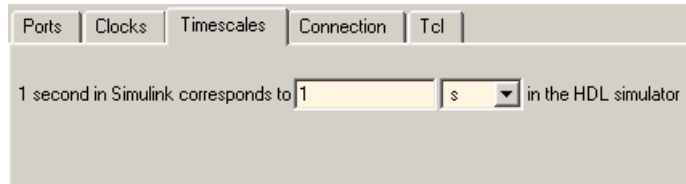
4 Click **Apply** to commit your changes.

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

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

where

- `tInTicks` 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 HDL 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:

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

Operation of Absolute Timing Mode. To understand the operation of absolute timing mode, we will again consider the example model discussed in "Operation of Relative Timing Mode" on page 3-19. 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.
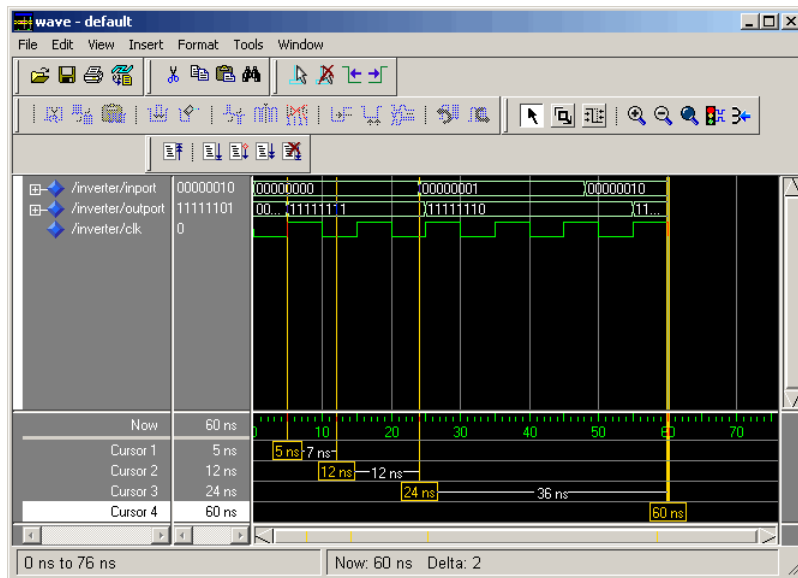
Timing Mode Usage Restrictions
The following restrictions apply to the use of absolute and relative timing modes:

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

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

Setting HDL Cosimulation Port Sample Times

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

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

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

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

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

- Specify the output sample time for an HDL Cosimulation block as an integer multiple of the resolution limit defined in 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 20 ns.

Handling Multirate Signals

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

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

Handling Frame-Based Signals

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

- "Overview" on page 3-26
- "Using Frame-Based Processing" on page 3-27
- "Frame-Based Cosimulation Example" on page 3-28

Overview

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

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

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

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

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

Using Frame-Based Processing

You do not need to configure the HDL Cosimulation block in any special way for frame-based processing. To use frame-based processing in a cosimulation, connect one or more single-channel frame-based signals to the input port(s) of the HDL Cosimulation block. All such signals must meet the requirements described in "Requirements and Restrictions" on page 3-27. The HDL Cosimulation block automatically configures its output(s) for frame-based operation at the appropriate frame size.

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

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

- Connection of mixed frame-based and sample-based signals to the same HDL Cosimulation block is not supported.

- Only single-channel frame-based signals can be connected to the HDL Cosimulation block. Use of multichannel (matrix) frame-based signals is not supported in this release.

- All frame-based signals connected to the HDL Cosimulation block must have the same frame size.

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

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

Frame-Based Cosimulation Example

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

The example files are:

- *matlabroot*\toolbox\modelsim\modelsimdemos\frame_filter_cosim.mdl: the example model.

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

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

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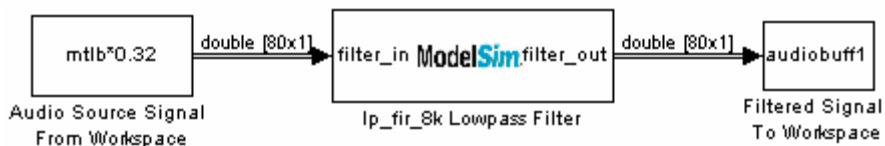
The Audio Source Signal From Workspace block provides an input signal from the workspace variable mtlb. The block is configured for an 8 kHz sample rate, with a frame size of 80, as shown in this figure.
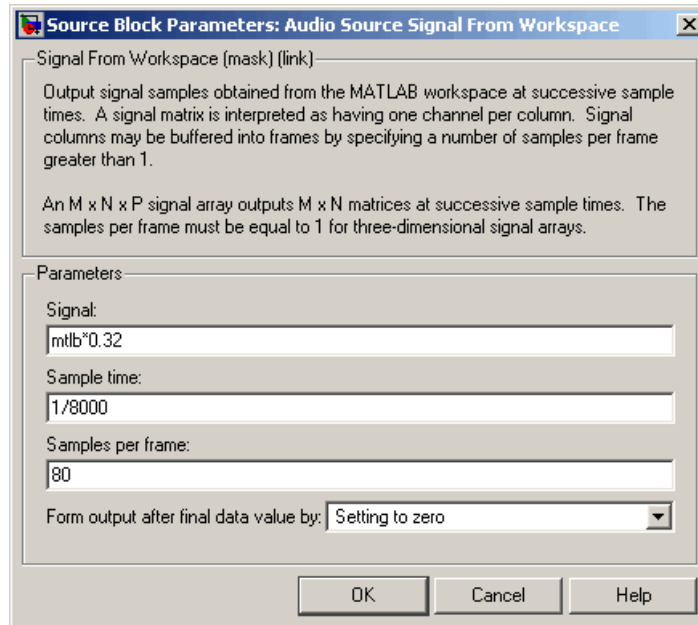
The sample rate and frame size of the input signal propagate throughout the model.

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

Note that for the ModelSim simulation to execute correctly, the clk_enable signal of the lp_fir_8k entity must be forced high. The signal is forced by a pre-simulation command transmitted by the HDL Cosimulation block. The command has been entered into the **Tcl** pane of the HDL Cosimulation block, as shown in the figure below.
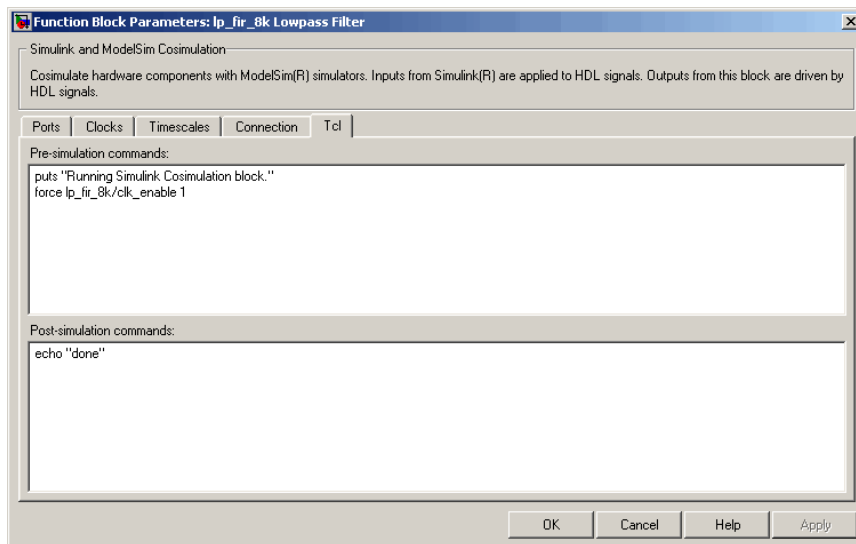
Output data from the HDL Cosimulation block is returned the workspace variable `audiobuff1` via the `Filtered Signal To Workspace` block.

To run the cosimulation:

1 Start MATLAB and make it your active window.

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

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

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

5 Open the example model.

```
open frame_filter_cosim.mdl
```

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

```
load mtlb
```

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

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

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

```
vsim
```

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

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

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

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

```
vsimulink lp_fir_8k
```

ModelSim is now set up for cosimulation.

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

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

ans =

    2.7190
```

The timing above is typical for a run of this model given a simulation **Stop time** of 1 second and a frame size of 80 samples. Timings are

system-dependent and will vary slightly from one simulation run to the next.

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

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

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

12 Open the block parameters dialog of the `Audio Source Signal From Workspace` block and set the **Samples per frame** property to 1, as shown in this figure.
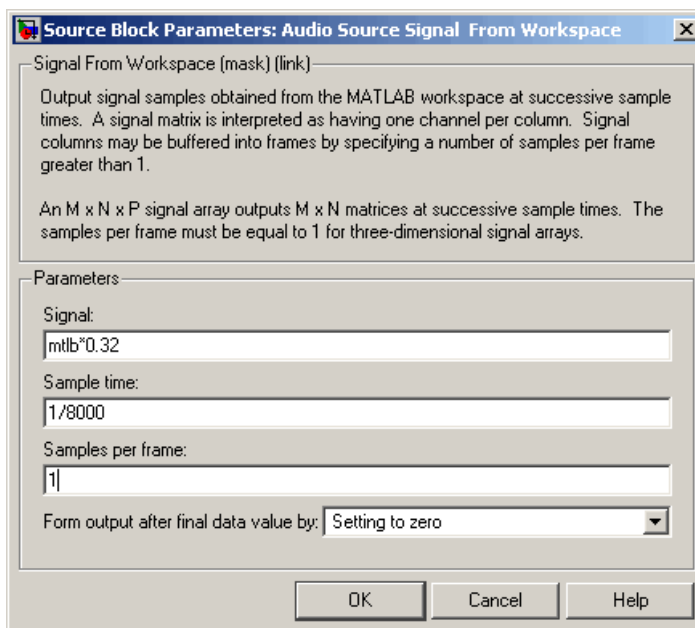
13 Close the dialog and activate the Simulink window. Select **Update
diagram** from the **Edit** menu.

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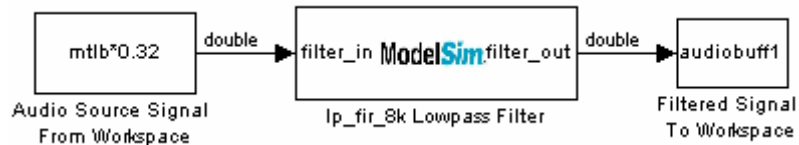
14 Activate ModelSim. At the ModelSim prompt, type

```
restart
```

15 Activate MATLAB. Run a simulation and measure elapsed time as follows:

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

ans =

    3.8440
```

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

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

Clock Signal Latency

In ModelSim, it is not possible to guarantee the order in which clock signals
(rising-edge or falling-edge) defined in the HDL Cosimulation block are
applied, relative to the data inputs driven by these clocks. Therefore, it is
possible that during a cosimulation, race conditions could develop between a
clock and the data inputs associated with the clock.

To avoid such race conditions, Link for ModelSim delays all such clocks by ½ clock period, in effect inverting the sense of the rising or falling edge. The delay provides a setup and hold time for input data, ensuring that data inputs are always applied before the driving clock edge is applied. For example, in the case of a rising-edge clock, inputs are applied first, and ½ clock period later, the rising edge of the clock is applied.

Where the Simulink sample time is even, the clock delay will be exactly ½ period. For odd Simulink sample times, the ½ period delay is approximated as closely as possible. While this apparent "inversion" or delay by ½ period of the active edge of the clock can be confusing, it enables cosimulation to work correctly without race conditions and without requiring separately specified setup and hold times for the data.

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.

As the following figure shows, Simulink cosimulation block input affects signal values just after the current ModelSim time step $(t+\delta)$ and block output reflects signal values just before the current ModelSim step time $(t-\delta)$.
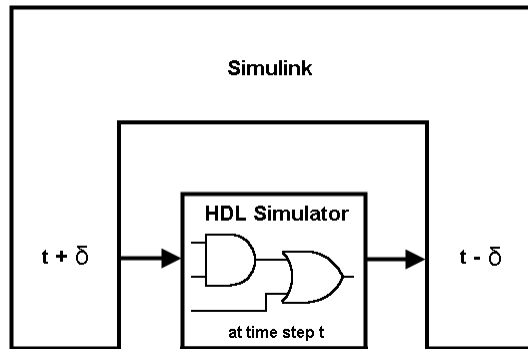
Regardless of whether your HDL code is specified with latency, the cosimulation block has a minimum latency that is equivalent to the cosimulation block's output sample time. For large sample times, the delay can appear to be quite long, but this is an artifact of the cosimulation block, which exchanges data with the HDL simulator at the block's output sample time only. This may be reasonable for a cosimulation block that models a device that operates on a clock edge only, such as a register-based device. For cosimulation blocks that 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;
```
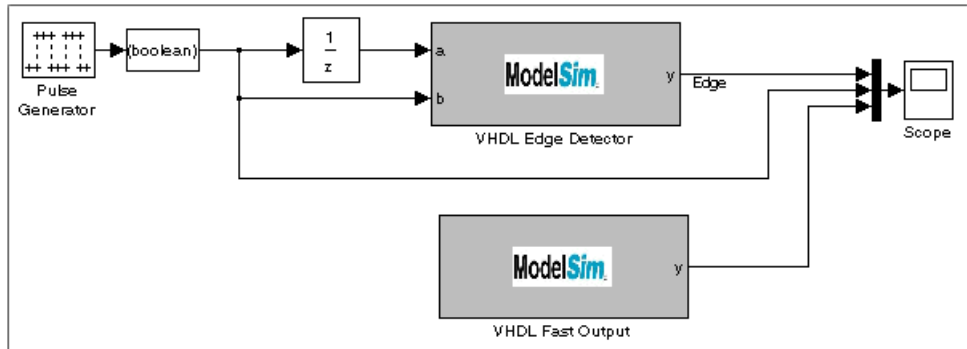
In the Simulink model, the cosimulation block VHDL Edge Detector contains an XOR circuit. The second cosimulation block, VHDL Fast Output, simply reads the same XOR output. The first block is driven by a signal generated by the Pulse Generator block. The Data Type Conversion block converts the signal 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
HDL Cosimulation block — Edge Detector	Input sample time	Inherited from Unit Delay block
	Output sample time	100
HDL Cosimulation block — Fast Output (source)	Output sample time	100

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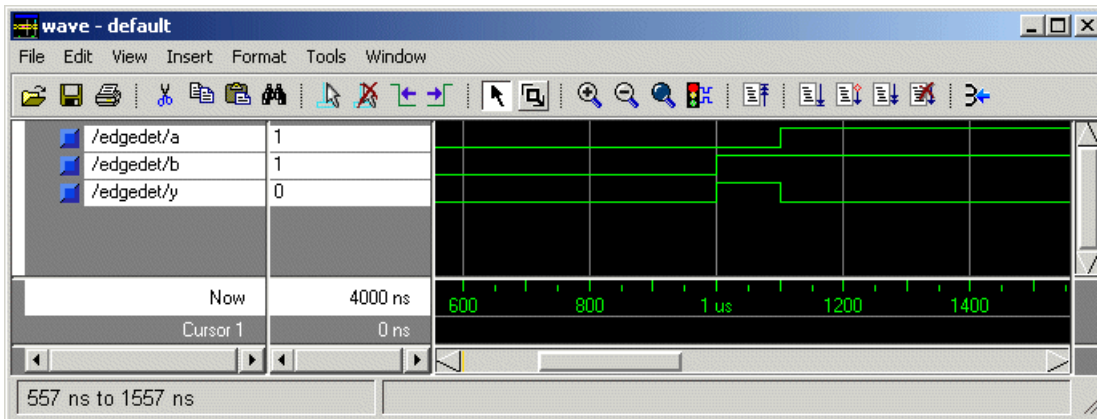
Note the following:

- Signal a gets asserted high after a 100 ns delay. This is due to the unit delay applied by the Simulink model.

- Signal b gets asserted high immediately.

- Signal y experiences a falling edge as a result of the XOR computation.

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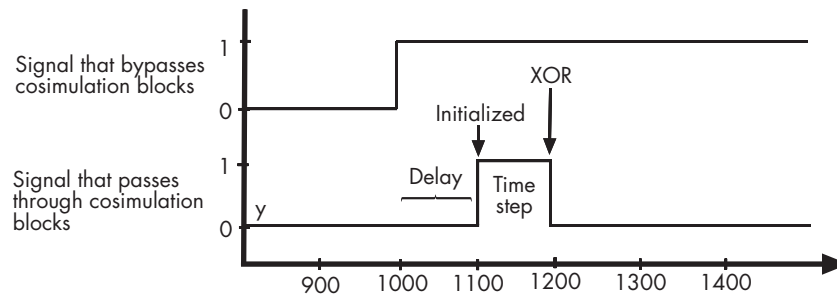
The signal that bypasses the cosimulation blocks rises at t=1000. That signal stays high for the duration of the sample period. However, the signals that are read from output port y of the two cosimulation blocks, display in the Scope window as follows:

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

- After the next time step, the signal value falls due to the VHDL XOR operation.

For cosimulation blocks that model combinatorial circuits, such as the one in the preceding example, you may want to experiment with a faster sample frequency for output ports. For example, suppose you change the **Output sample time** for the VHDL Fast Output cosimulation block from 100 to 20. The following figure highlights the individual signal paths that appear in the Scope window for this scenario.
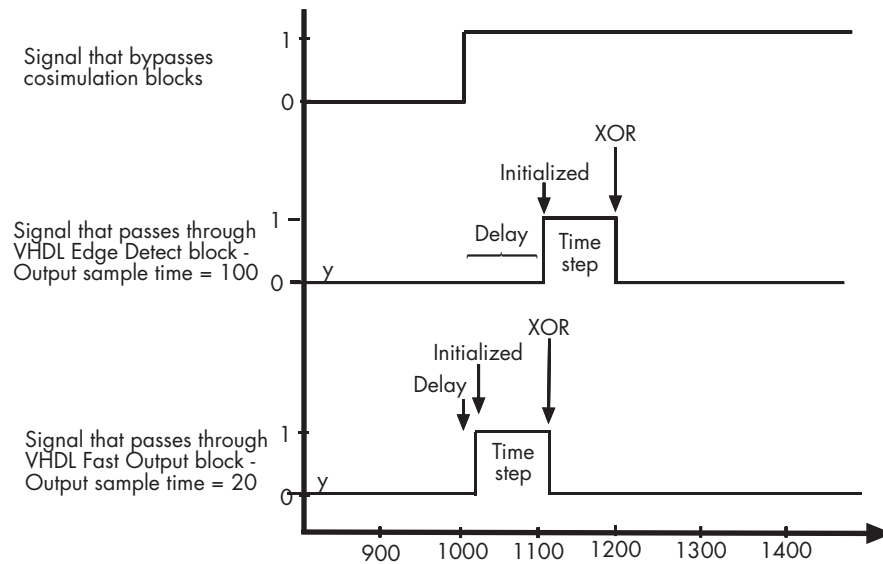
In this case, the signal that bypasses the cosimulation blocks and the output signal read from the VHDL Edge Detect block remain the same. However, the delay for the signal read from the VHDL Fast Output block is 20 ticks instead of 100. Although the size of the time step is still tied to the ModelSim resolution limit, the delay that occurs before the VHDL code is processed is significantly reduced and the time of execution more closely reflects simulation time in ModelSim.

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

Interfacing with Continuous Time Signals

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

Setting Simulink Configuration Parameters

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 HDL modeling. For example, you might want to adjust the value of the **Stop time** parameter in the **Solver** pane of the Configuration Parameters dialog box.

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

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

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

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

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

Running and Testing a Hardware Model in Simulink

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

Simulink and HDL Simulator Communication Options

Select shared memory or socket communication. See "Modes of Communication" on page 1-7.

Starting the HDL Simulator

See "Starting ModelSim for Use with Link for ModelSim" on page 1-21.

Incorporating Hardware Designs Into a Simulink Model

In this section...

Overview

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

1 Open your Simulink model, if it is not already open.

2 Delete the model component that the HDL Cosimulation block is to replace.

3 In the Simulink Library Browser, click the Link for ModelSim library. The browser displays the block icons shown below.

	HDL Cosimulation	Block that has at least one input port and one output port.
	To VCD File	Generates a Value Change Dump (VCD) file. For information on using this block, see "Adding a Value Change Dump (VCD) File" on page 3-68.
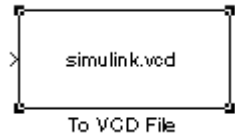
4 Copy the HDL Cosimulation block icon from the Library Browser to your model. Simulink creates a link to the block at the point where you drop the block icon.

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

Specifying HDL Signal/Port and Module Paths for Cosimulation

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

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

Path Specifications for Simulink Cosimulation Sessions with Verilog Top Level

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

The following are valid signal and module path specification examples:

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

The following are invalid signal and module path specification examples:

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

Path Specifications for Simulink Cosimulation Sessions with VHDL Top Level

- Path specification may include the top-level module name but it is not required.

- Path specification can include "." or "/" path delimiters, but cannot include a mixture.

- The leaf module or signal must match the HDL language of the top-level module.

The following are valid signal and module path specification examples:

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

The following are invalid signal and module path specification examples:

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

Driving Clocks, Resets, and Enables

Creating Optional Clocks

You can create rising-edge or falling-edge clocks that apply internal stimuli to your cosimulation model. When you specify a clock in your block definition, Simulink creates a rising-edge or falling-edge clock that drives the specified HDL signals by depositing them.

Simulink attempts to create a clock that has a 50% duty cycle and a predefined phase that is inverted for the falling edge case. If necessary, Simulink degrades the duty cycle to accommodate odd Simulink sample times, with a worst case duty cycle of 66% for a sample time of T=3.

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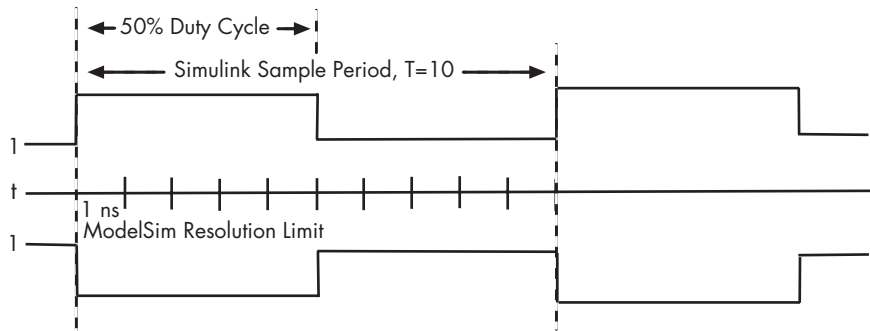
Rising Edge Clock

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 HDL Signals to Block Ports" on page 3-48.

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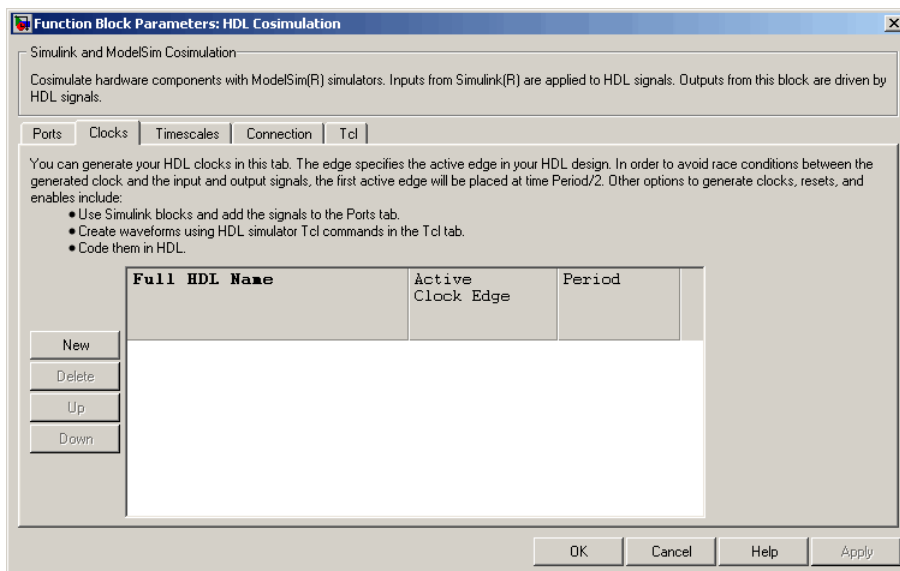
3 Click the **New** button to add a new clock signal.

4 Edit the clock signal pathname directly in the table under the **Full HDL Name** column by double-clicking on the default clock signal name (`/top/clk`). Specify your new clock using ModelSim pathname syntax. See "Specifying HDL Signal/Port and Module Paths for Cosimulation" on page 3-43.

Note that vectored signals in the **Clocks** pane are not supported. Signals must be logic types with '1' and '0' values.

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

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

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

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

The following dialog defines the rising-edge clock clk for the HDL Cosimulation block, with a default period of 2.
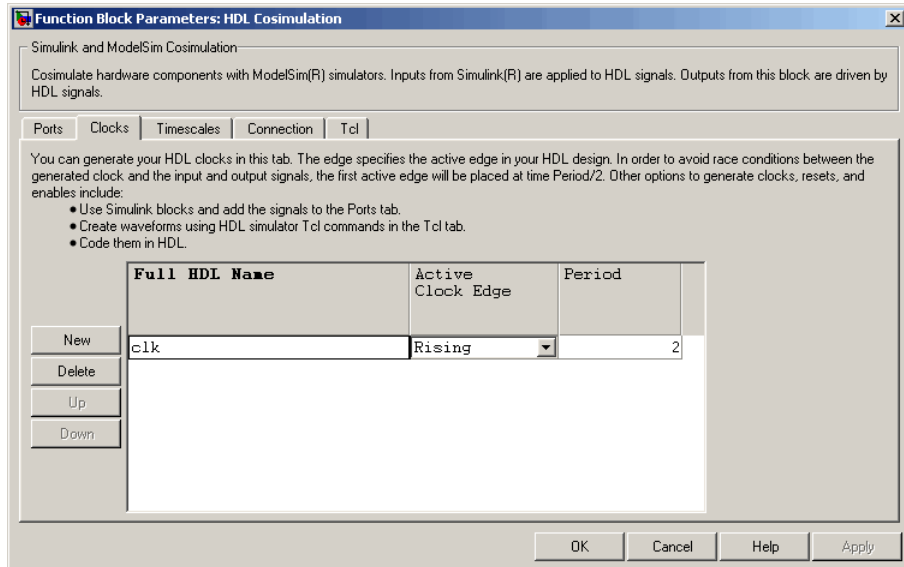
Defining the Block Interface

To open the block parameters dialog for the HDL Cosimulation block, double-click the block icon.
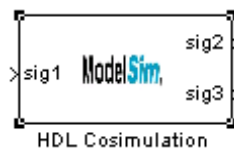
Simulink displays the following Block Parameters dialog.
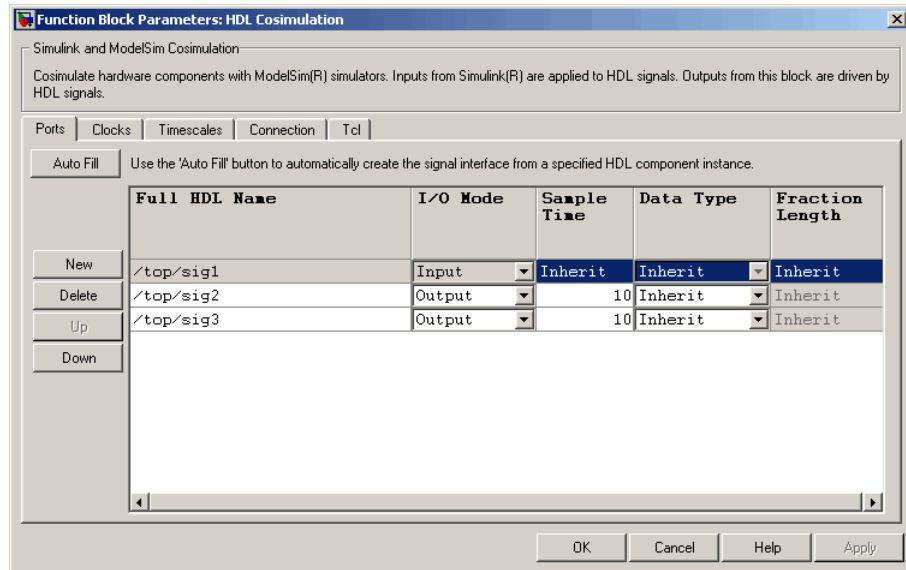
Mapping HDL Signals to Block Ports

The first step to configuring your Link for ModelSim block is to map signals and signal instances of your HDL design to port definitions in your HDL Cosimulation block. In addition to identifying input and output ports, you can specify a sample time for each output port. You can also specify a fixed-point data type for each output port.

The signals that you map can be at any level of the HDL design hierarchy.

To map the signals, you can

- Enter signal information manually into the **Ports** pane of the HDL Cosimulation Block Parameters dialog (see "Entering Signal Information Manually" on page 3-54). This approach can be more efficient when you want to connect a small number of signals from your HDL model to Simulink.

- Use the **Auto Fill** button to obtain signal information automatically by transmitting a query to ModelSim. This approach can save significant effort when you want to cosimulate an HDL model that has a large number of signals that you want to connect to your Simulink model. Note, however,

that in many cases you will need to edit the signal data returned by the query. See "Obtaining Signal Information Automatically from ModelSim" on page 3-49 for details.

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

Obtaining Signal Information Automatically from ModelSim. The **Auto Fill** button lets you initiate a ModelSim query and supply a path to a component or module in an HDL model under simulation in ModelSim. Usually, some modification of the port information is required after the query completes.

The required steps are outlined in the example below. The example is based on a modified copy of the Manchester Receiver model (see "Creating a Hardware Model Design for Use in Simulink" on page 3-5), in which all signals were initially deleted from the **Ports** and **Clocks** panes.

1 Open the block parameters dialog for the HDL Cosimulation block. Click the **Ports** tab. The **Ports** pane opens.
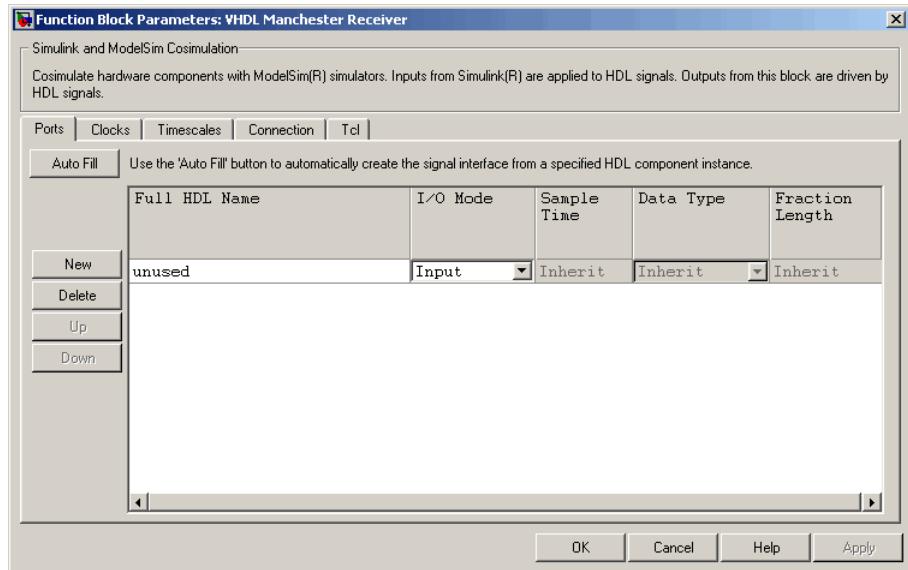
2 Click the **Auto Fill** button. The **Auto Fill** dialog opens.
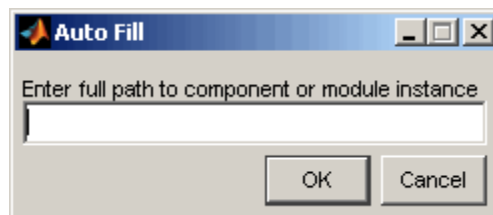
This modal dialog requests a path to a component or module in your HDL model; here you enter an explicit HDL path into the edit field.

3 In this example, we will obtain port data for a VHDL component called manchester. The HDL path is specified as /top/manchester.
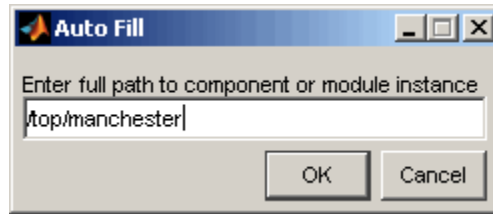
4 Click **OK**. The dialog is dismissed and the query is transmitted.

5 Port data is returned and entered into the **Ports** pane almost instantaneously, as shown in the figure below.
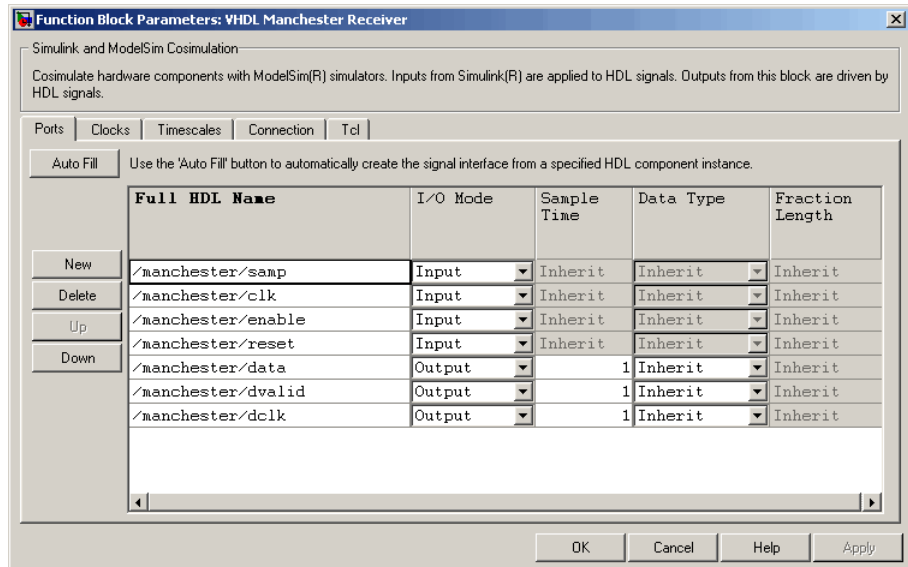
6 Click **Apply** to commit the port additions.

7 Observe that **Auto Fill** has returned information about *all* inputs and outputs for the targeted component. In many cases, this will include signals that function in ModelSim but cannot be connected in the Simulink model. You should delete any such entries from the list in the **Ports** pane unless you are adding blocks to the Simulink model to represent these signals.

The figure above shows that the query entered clock, clock enable, and reset ports (labelled `clk`, `enable`, and `reset` respectively) into the ports list. In this example, the `clk` signal is entered in the **Clocks** pane, and the `enable` and `reset` signals are deleted from the **Ports** pane, as shown in the figures below.

Note Enter force commands in the **Tcl** pane to drive the reset and enable signals; for example:

```
force design/reset value time
```

where *value* is '1' or '0' and *time* is in nanoseconds.

---
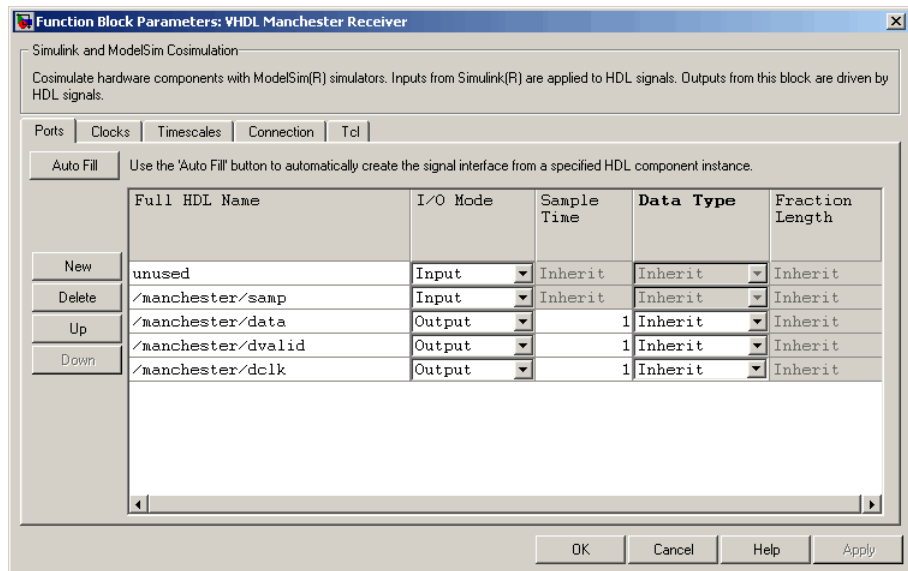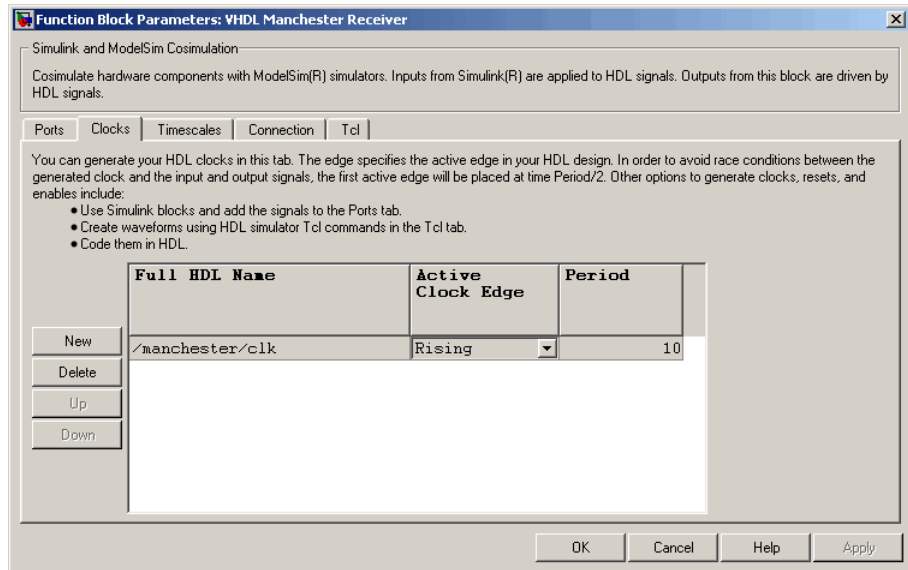
8 **Auto Fill** returns default values for output ports:

- **Sample time**: 1
- **Data type**: Inherit
- **Fraction length**: N/A

You may need to change these values as required by your model. In this example, the **Sample time** should be set to 10 for all outputs. See also "Specifying the Signal Datatypes" on page 3-58.

9 Note that **Auto Fill** does not return information for internal signals. If your Simulink model needs to access such signals, you must enter them into the **Ports** pane manually. For example, in the case of the Manchester Receiver model, you would need to add output port entries for /manchester/sync_i, /manchester/isum_i, and /manchester/qsum_i, as shown below.

10 Before closing the HDL Cosimulation block parameters dialog, click **Apply** to commit any edits you have made.
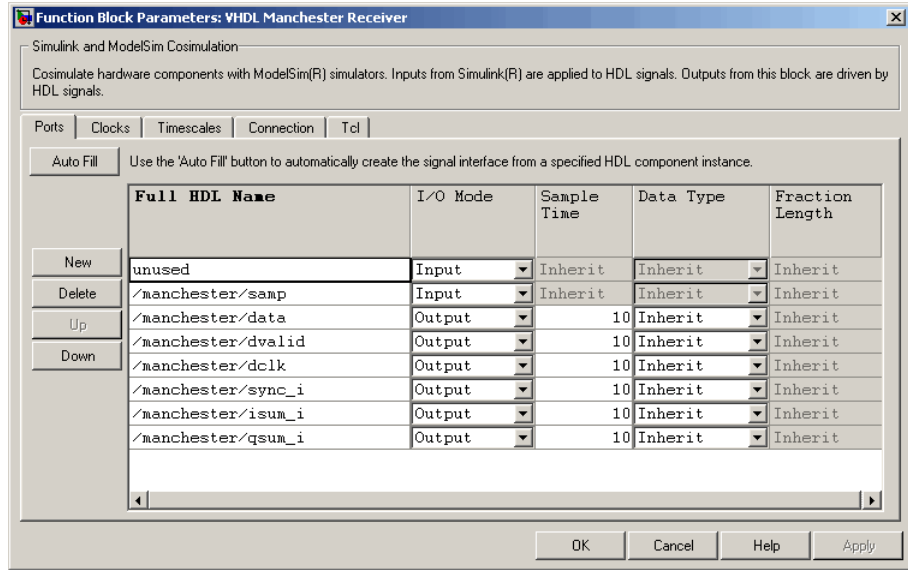
Entering Signal Information Manually. To enter signal information
directly in the **Ports** pane:

1 In ModelSim, determine the test signal pathnames for the HDL 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 HDL
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 shows all signals are subordinate to
the top-level entity manchester.
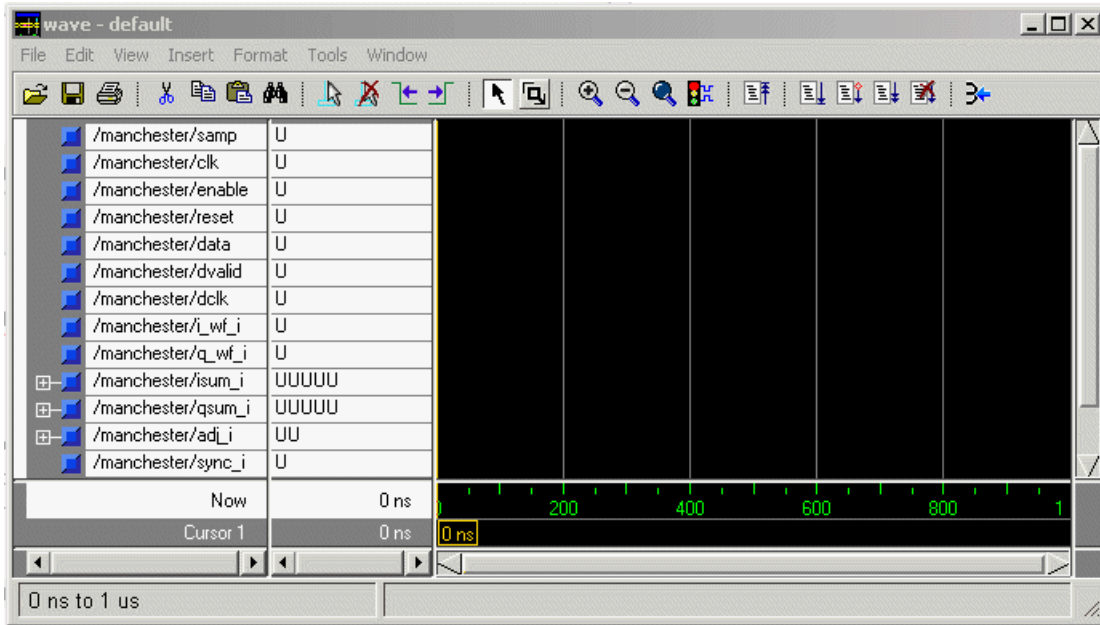
2 In Simulink, open the block parameters dialog for your HDL Cosimulation block, if it is not already open.

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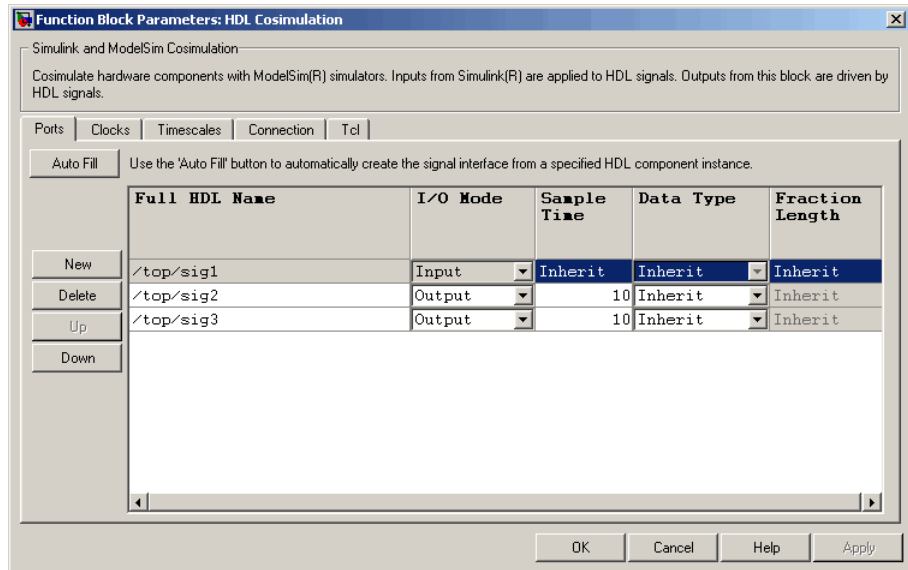
In this pane, you define the HDL signals of your design that you want to include in your Simulink block and set a sample time and data type for output ports. The parameters that you should specify on the **Ports** pane depend on the type of device the block is modeling as follows:

- For a device having both inputs and outputs: specify block input ports, block output ports, output sample times and output data types. For output ports, accept the default or enter an explicit sample time. Data types can be specified explicitly, or set to Inherit (the default). In the default case, the output port data type is inherited either from the signal connected to the port, or derived from the HDL model.

- For a sink device: specify block output ports

- For a source device: specify block input ports

4 Enter test signal pathnames in the **Full HDL name** column by double-clicking on the existing default signal. Use ModelSim pathname syntax (see "Specifying HDL Signal/Port and Module Paths for Cosimulation" on page 3-43). If you are adding signals, click **New** and then edit the default values. Select either Input or Output from the **I/O Mode** column. If desired, set the **Sample Time**, **Data Type**, and **Fraction Length** parameters for signals explicitly, as discussed below.

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

The following dialog shows port definitions for an HDL Cosimulation block. Note the signal pathnames match pathnames that appear in the ModelSim **wave** window shown in step 1.
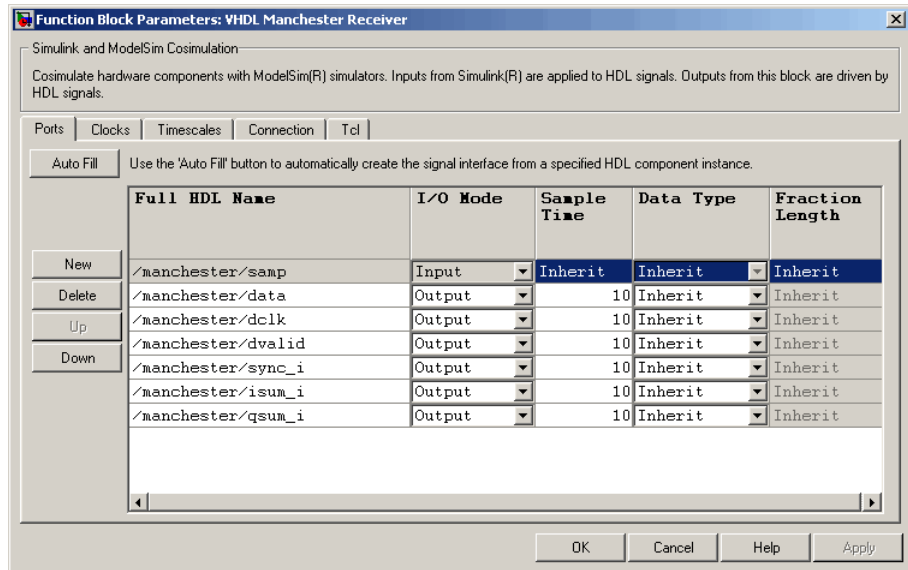
Note When you define an input port, make sure that only one source is set up to force input to that port. For example, you should avoid defining an input port that has multiple instances. If multiple sources drive a signal, your Simulink model may produce unpredictable results.

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 3-15.

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

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

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

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

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

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

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

Specifying the Signal Datatypes

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

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

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

The **Data Type** and **Fraction Length** properties will apply only to

- VHDL signals of `STD_LOGIC` or `STD_LOGIC_VECTOR` type

- Verilog signals of `wire` or `reg` type

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

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

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

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

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

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

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

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

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

Relating Simulink's and ModelSim's Sense of Time

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 3-15 to understand the supported timing modes and the issues that will determine your choice of timing mode.

You can specify either a relative or an absolute timing relationship between Simulink and 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 HDL Cosimulation block parameters dialog.

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

3 Enter a scale factor in the text box on the left. The default scale factor is 1.

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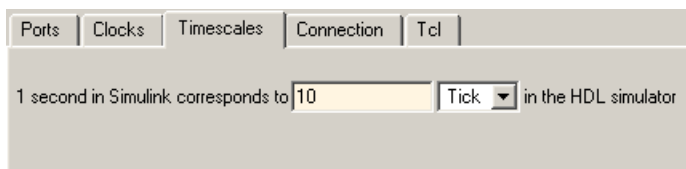
4 Click **Apply** to commit your changes.

Specifying an Absolute Timing Relationship

To configure absolute timing mode for a cosimulation:

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

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

3 Enter a scale factor in the text box on the left. The default scale factor is `1`.

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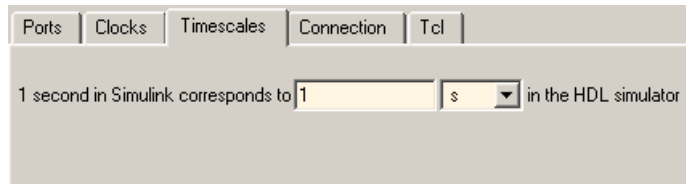
4 Click **Apply** to commit your changes.

Configuring the Communication Link in the HDL Cosimulation Block

Configure a block's communication link with the **Connection** pane of the block parameters dialog.

The following steps guide you through the communication configuration. 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 **The HDL simulator is 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 HDL 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 D-2. 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 1-7.

If you choose TCP/IP socket communication, specify a valid port number or service for your computer system in the **Port number or service** text field. For information on choosing TCP/IP socket ports, see "Choosing TCP/IP Socket Ports" on page D-2.

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

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

- **Full Simulation**: Confirm interface and run HDL simulation (default).

- **Confirm Interface Only**: Check HDL simulator for proper signal names, dimensions, and data types, but do not run HDL simulation.

- **No Connection**: Do not communicate with the HDL simulator. The HDL simulator does not need to be started.

With the 2nd and 3rd options, Link for ModelSim does not communicate with the HDL simulator during Simulink simulation.

6 Click **Apply**.
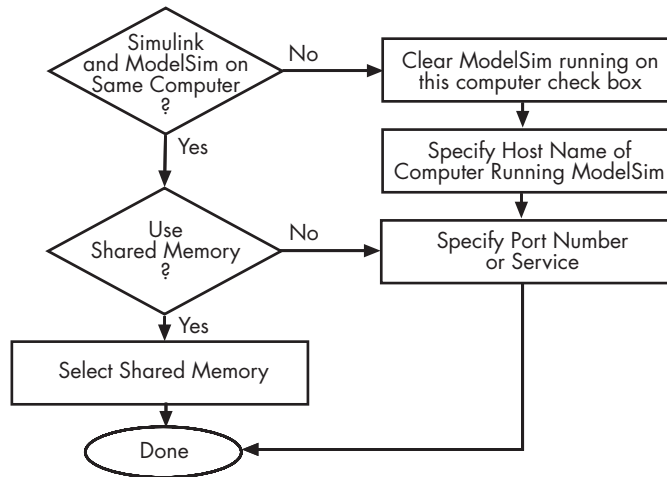
The following example dialog shows communication definitions for an HDL Cosimulation block. The block is configured for Simulink and ModelSim running on the same computer, communicating in TCP/IP socket mode over TCP/IP port 4449.
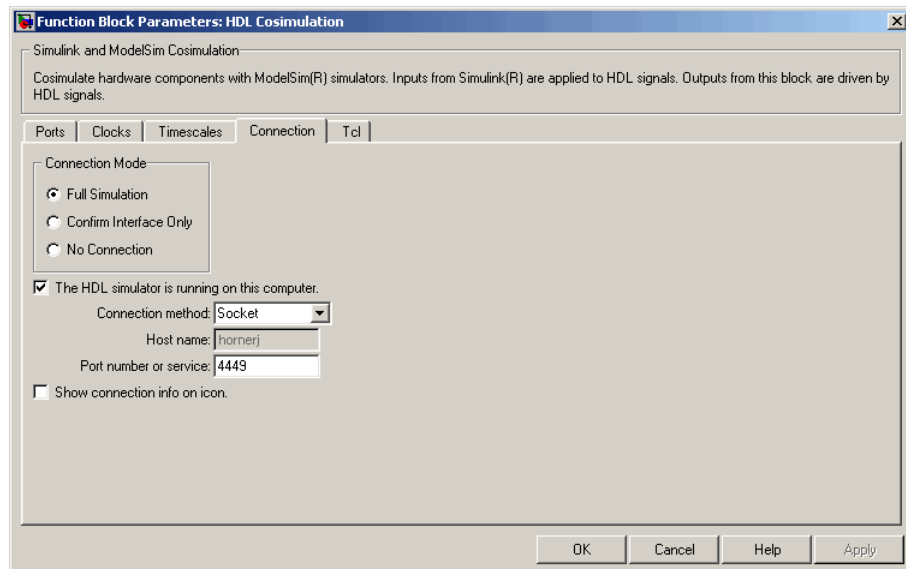
Specifying Pre- and Post-Simulation Tcl Commands with HDL Cosimulation Block Parameters Dialog Box

You have the option of specifying Tcl commands to execute before and after ModelSim simulates the HDL 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. For example, the **Post- simulation command** field on the Tcl Pane is particularly useful for instructing ModelSim to restart at the end of a simulation run.

You can specify the pre- and post-simulation Tcl commands by entering Tcl commands in the Pre-simulation commands or Post-simulation commands text fields of the HDL Cosimulation block.

To specify Tcl commands,

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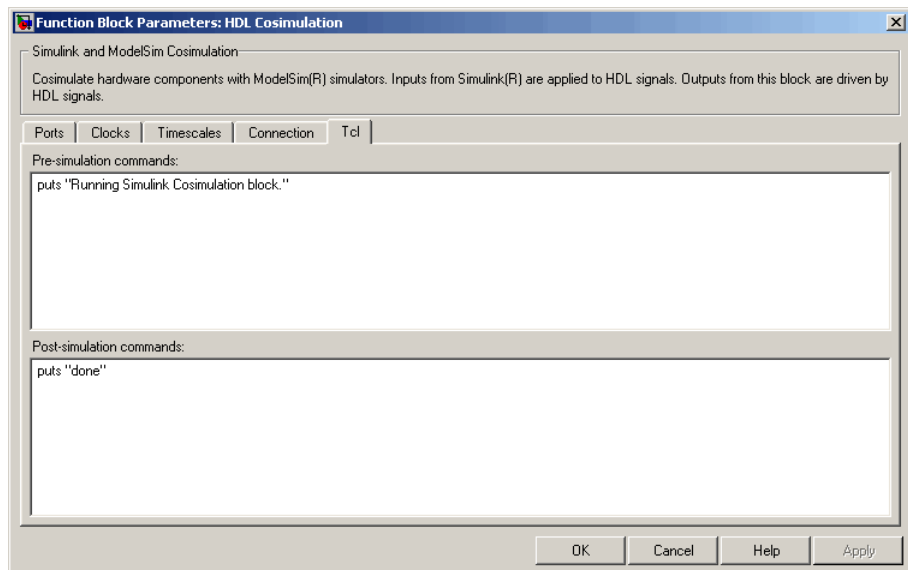
The **Pre–simulation commands** text box includes an echo command for reference purposes.

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 (;), which is the standard Tcl concatenation operator.

Alternatively, you can create a ModelSim DO file that lists Tcl commands and then specify that file with the ModelSim do command as shown in the following figure.
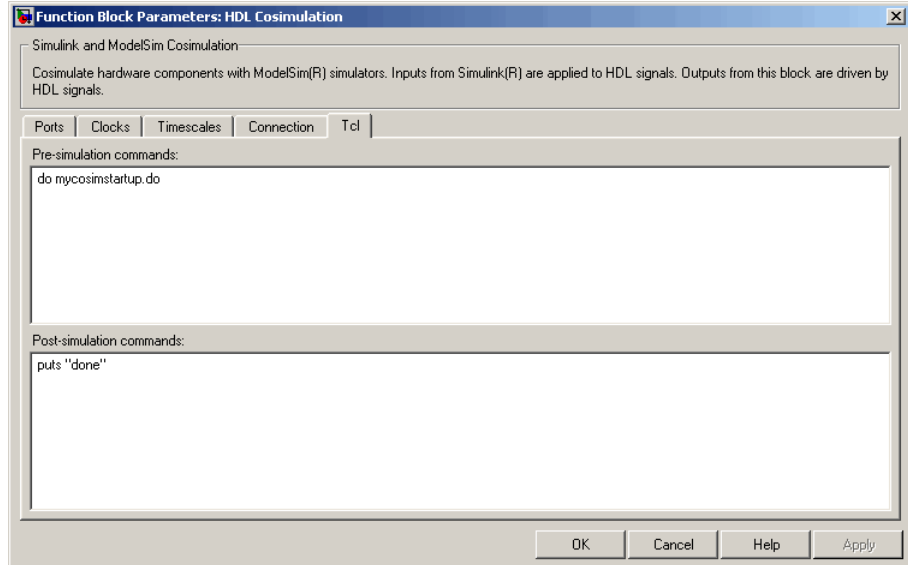
3 Click **Apply**.

Programmatically Controlling the Block Parameters

One way to control block parameters is through theHDL Cosimulation block graphical dialog box. However, you can also control blocks by programmatically controlling the mask parameter values and the running of simulations. Parameter values can be read using the Simulink get_param

function and written using the Simulink set_param function. All block parameters have attributes that indicate whether they are:

- Tunable—the attributes can change during the simulation run

- Evaluated— the parameter string value is put through an evaluation to determine its actual value used by the S-Function

The HDL Cosimulation block does not have any tunable parameters; thus, you get an error if you try to change a value while the simulation is running, but it does have a few evaluated parameters.

You can see the list of parameters and their attributes by performing a right-mouse click on the block, selecting **View Mask**, and then the **Parameters** tab. The **Variable** column shows the programmatic parameter names. Alternatively, you can get the names programmatically by selecting the HDL Cosimulation block and then typing at the MATLAB prompt:

```
>> get_param(gcb, 'DialogParameters')
```

Some examples of using MATLAB to control simulations and mask parameter values follow. Usually, the commands are put into an M-script or M-function file and automatically called by several callback hooks available to the model developer. You can place the code in any of these suggested locations, or anywhere you choose:

- In the model workspace, e.g., **View > Model Explorer > Simulink Root > *model_name* > Model Workspace > Data Source is M-Code**.

- In a model callback, e.g., **File > Model Properties > Callbacks**.

- A subsystem callback (right-mouse click on an empty subsystem and then select **Block Properties > Callbacks**). Many of the Link for ModelSim demos use this technique to start the HDL simulator by placing M-code in the OpenFcn callback.

- The HDL Cosimulation block callback (right-mouse click on HDL Cosimulation block, and then select **Block Properties > Callbacks**)

Examples
The following examples demonstrate the use of programmatically controlling the HDL Cosimulation block parameters.

- "Scripting the Value of the Socket Number for HDL Simulator Communication" on page 3-67
- "Programmatically Adding Debug Commands to Pre/Post Tcl Commands" on page 3-67

Scripting the Value of the Socket Number for HDL Simulator Communication. In a regression environment, you may need to determine the socket number for the Simulink/HDL simulator connection during the simulation to avoid collisions with other simulation runs. This example demonstrates code that could handle that task. The script is for a 32-bit Linux platform.

```
ttcp_exec = [matlabroot '/toolbox/shared/hdllink/scripts/ttcp_glnx'];
[status, results] = system([ttcp_exec ' -a']);
if ~s
  parsed_result = strread(results,'%s');
  avail_port = parsed_result{2};
else
  error(results);
end

set_param('MyModel/HDL Cosimulation', 'CommPortNumber', avail_port);
```

Programmatically Adding Debug Commands to Pre/Post Tcl Commands. This example code demonstrates how to dynamically control the use of HDL waveform generation when you move from interactive debug testing to regression testing.

```
% note: the 'hdlDebugMode' variable is assumed to be defined in the workspace.
%       there are many ways to define the variable and control its value, including
%       some of the callback methods described above.  Another idea is to use a
%       Simulink Switch or Manual Switch block.
if (hdlDebugMode)
  tclPreCmds = 'puts { Starting cosimulation.}; add wave /top';
else
  tclPreCmds = 'puts { Starting cosimulation.};'
end

set_param('MyModel/HDL Cosimulation', 'TclPreSimCommand', tclPreCmds);
```

Adding a Value Change Dump (VCD) File

A value change dump (VCD) file logs changes to variable values, such as the values of signals, in a file during a simulation session. VCD files can be useful during design verification. Some examples of how you might apply VCD files include

- 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'`.

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.
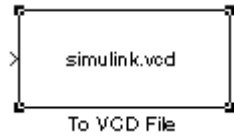
4 Copy the To VCD File block from the Library Browser to your model by clicking the block and dragging it from the browser to your model window.

5 Connect the block ports to appropriate blocks in your Simulink model.

Note The To VCD File block does not support floating point signal types.

Note Because multi-dimensional signals are not part of the VCD specification, they are flattened to a 1D vector in the file.

6 Configure the To VCD File block by specifying values for parameters in the Block Parameters dialog.

  a Double-click the block icon. Simulink displays the following dialog.
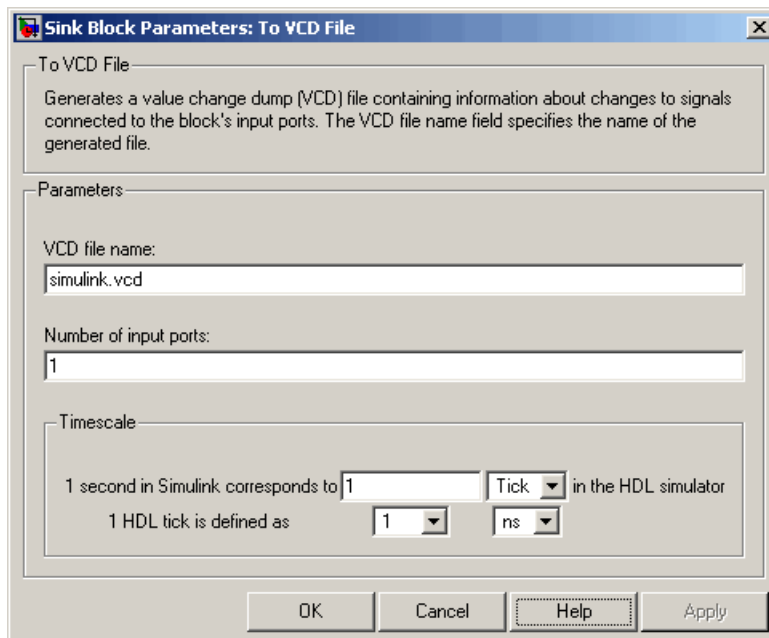
b Specify a filename for the generated VCD file in the **VCD file name** text box. If you specify a filename only, Simulink places the file in your current MATLAB directory. Specify a complete pathname to place the generated file in a different location.

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

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

---

c Specify an integer in the **Number of input ports** text box that indicates the number of block input ports on which signal data is to be collected. The block can handle up to 94^3 (830,584) bits, each of which maps to a unique symbol in the VCD file.

d Click **OK**.

7 Choose an optimal timing relationship between Simulink and the HDL simulator. The time scale options specify a correspondence between one second of Simulink time and some quantity of HDL simulator time. Choose relative time or absolute time. For more on the To VCD File time scale, see To VCD File.

8 Run the simulation. Simulink captures the simulation data in the VCD file as the simulation runs.

For a description of the VCD file format see "VCD File Format" on page 6-25. For a sample application of a VCD file, see "toVCD Block Tutorial" on page 3-92.

Running Cosimulations

In this section...
"Starting ModelSim for Use with Simulink" on page 3-72
"Determining an Available Socket Port Number" on page 3-72
"Checking the Connection Status" on page 3-73
"Managing a Simulink Cosimulation Session" on page 3-73

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`. See "Starting ModelSim for Use with Link for ModelSim" on page 1-21.

Loading an HDL Module for Cosimulation

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

```
vsimulink work.manchester
```

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

Determining an Available Socket Port Number

To determine an available socket port number, use `hldaemon` with the property name 'socket' set to value 0 at the MATLAB command prompt:

```
>> hdldaemon('socket',0)
HDLDaemon socket server is running on port 2274 with 0 connections
```

Checking the Connection Status

You can check the connection status by clicking the Update diagram button

[icon] or by selecting **Edit > Update Diagram**. If there is a connection error, Simulink will notify you.

The MATLAB command `pingHdlSim` can also be used to check the connection status. If a -1 is returned, then there is no connection with the HDL simulator.

Managing a Simulink Cosimulation Session

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

or the Start Simulation button [icon] 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.

If you change any part of the Simulink model, including the HDL Cosimulation block parameters, re-run the simulation or click the Update

diagram button [icon] or select **Edit > Update Diagram** so that the diagram reflects those changes.

Simulink and ModelSim Tutorial

In this section...

Tutorial Overview

This chapter guides you through the basic steps for setting up a Link for ModelSim session that uses Simulink and the HDL Cosimulation block to verify an HDL model. The HDL Cosimulation block cosimulates a hardware component by applying input signals to and reading output signals from an HDL model under simulation in ModelSim. The HDL Cosimulation block supports simulation of either VHDL or Verilog models. In the tutorial in this section, we will cosimulate a simple VHDL model.

Developing the VHDL Code

A typical Simulink and ModelSim scenario is to create a model for a specific hardware component in ModelSim that you later need to integrate into a larger Simulink model. The first step is to design and develop a VHDL model in ModelSim. In this tutorial, you use ModelSim and VHDL to develop a model that represents the following inverter:
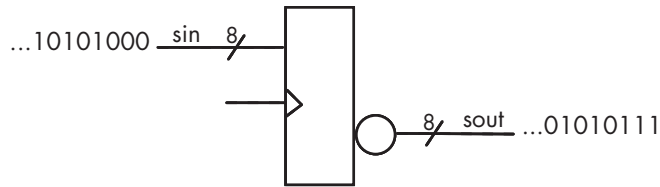
The VHDL entity for this model will represent 8-bit streams of input and output signal values with an IN port and OUT port of type STD_LOGIC_VECTOR. An input clock signal of type STD_LOGIC will trigger the bit inversion process when set:

1 Start ModelSim

2 Change to the writable directory MyPlayArea, which you may have created for another tutorial. If you have not created the directory, create it now. The directory must be writable.

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

3 Open a new VHDL source edit window.

4 Add the following VHDL code:

```
--------------------------------------------------
--  Simulink and ModelSim Inverter Tutorial
--
--  Copyright 2003 The MathWorks, Inc.
--  $Date: 2003/11/13 22:18:11 $
--------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY inverter IS PORT (
  sin : IN  std_logic_vector(7 DOWNTO 0);
  sout: OUT std_logic_vector(7 DOWNTO 0);
  clk : IN  std_logic
);
END inverter;

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

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

5 Save the file to inverter.vhd.

Compiling the VHDL File

This section explains how to set up a design library and compile inverter.vhd:

1 Verify that the file inverter.vhd is in the current directory by entering the ls command at the ModelSim command prompt.

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

```
ModelSim> vlib work

ModelSim> vmap work work
```

If the design library work already exists, ModelSim *does not* overwrite the current library, but displays the following warning:

```
# ** Warning: (vlib-34) Library already exists at "work".
```

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

3 Compile the VHDL file. One way of compiling the file is to click the filename in the project workspace and select **Compile > Compile All**.

Another alternative is to specify the name of the VHDL file with the vcom command, as follows:

```
ModelSim> vcom inverter.vhd
```

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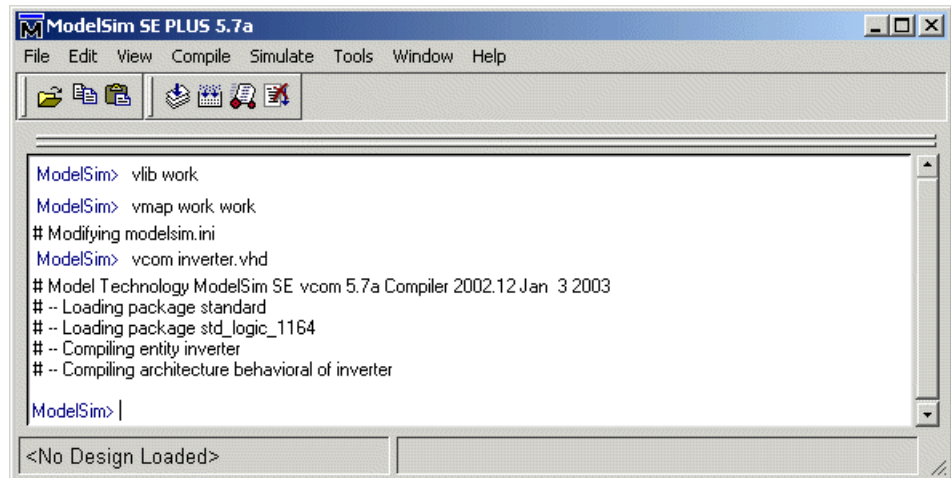
Creating the Simulink Model

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 3-74 and displays the inverted output.

Start by creating a model, as follows:

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

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

- Constant block from the Simulink Source library

- HDL Cosimulation block from the Link for ModelSim library

- Display block from the Simulink Sink library

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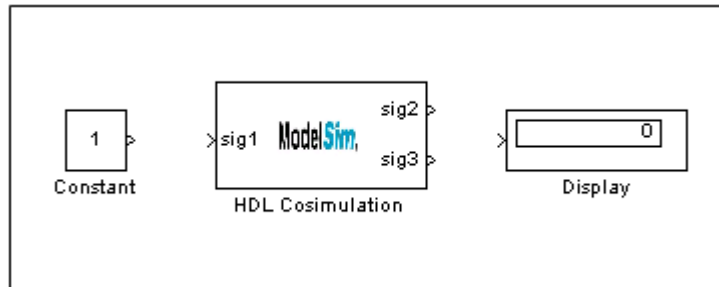
Next, configure the Constant block, which is the model's input source:

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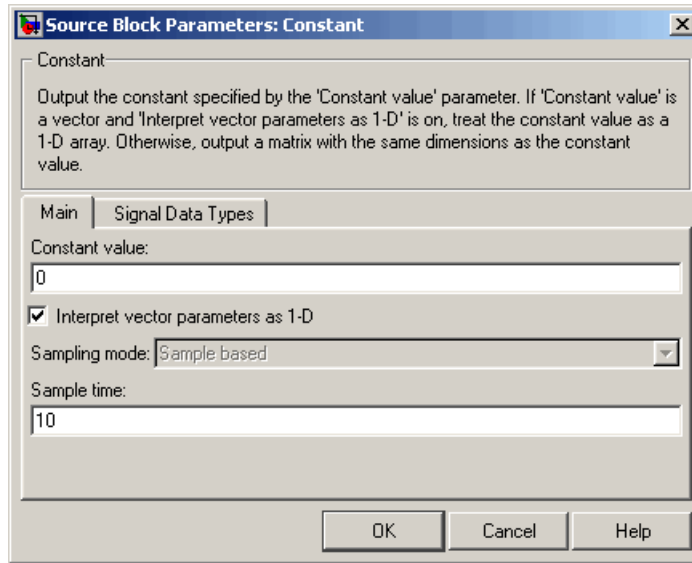
2 Click the **Signal data types** tab. The dialog box now displays the **Output data type mode** menu.

Select uint8 from the **Output data type mode** menu. This data type specification is supported by 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.
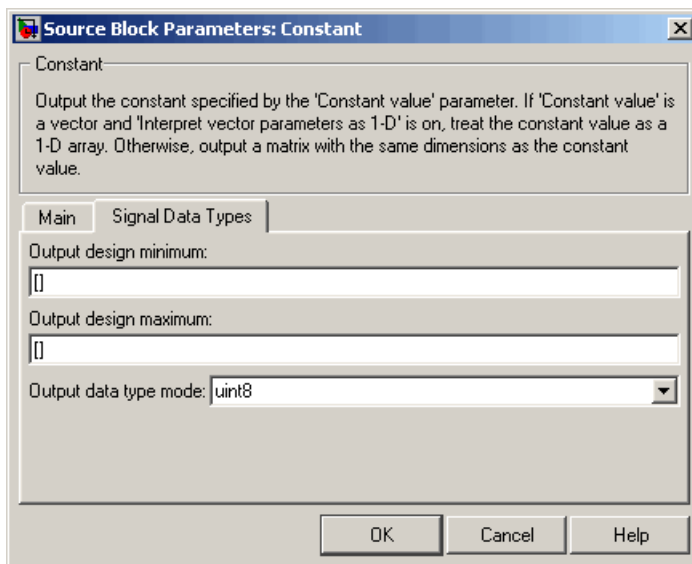
3 Click **OK**. The Constant block parameters dialog closes and the value in the Constant block icon changes to 0.

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

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

2 In the **Ports** pane, select the sample signal /top/sig1 from the signal list in the center of the pane by double-clicking on it.

3 Replace the sample signal pathname /top/sig1 with /inverter/sin. Then click **Apply**. The signal name on the HDL Cosimulation block changes.

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

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

The **Ports** pane should appear as follows.
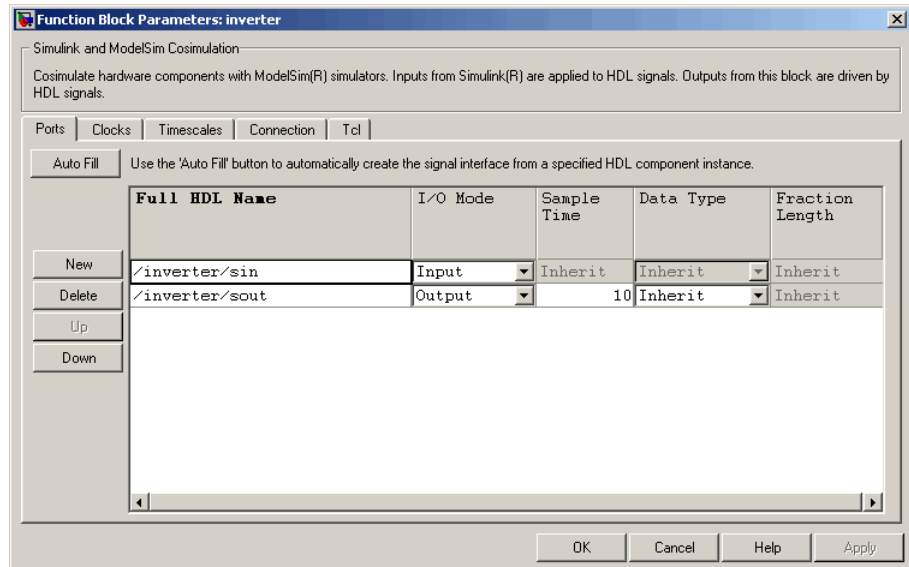
Now configure the parameters of the **Connection** pane:

1 Click the **Connection** tab.

2 Leave **Connection Mode** as **Full Simulation**.

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

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

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

3-81

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

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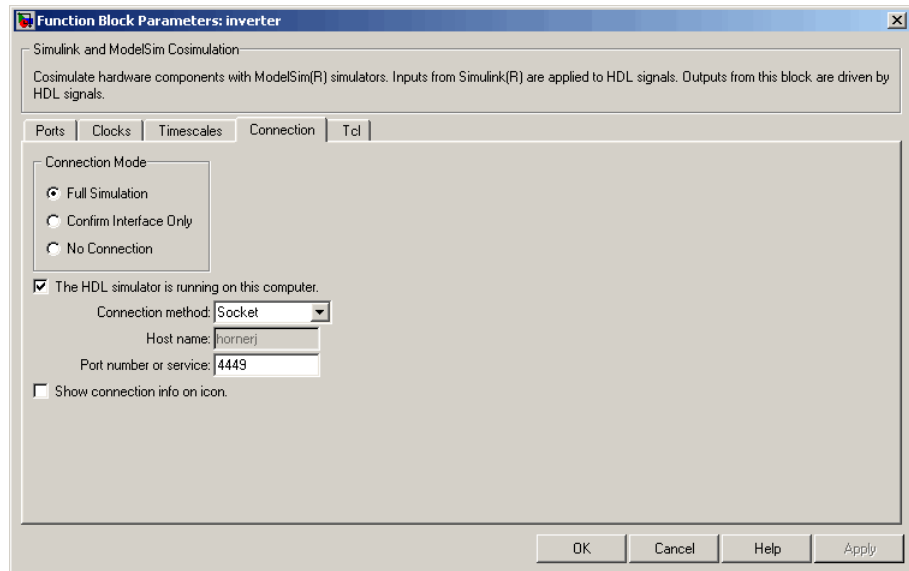
5 Click **Apply**.

Now configure the **Clocks** pane:

1 Click the **Clocks** tab.

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

3 Double-click on the new signal name to edit. Enter the signal path `/inverter/clk`. Then select `Rising` from the **Edge** list. Set the **Period** parameter to `10`.

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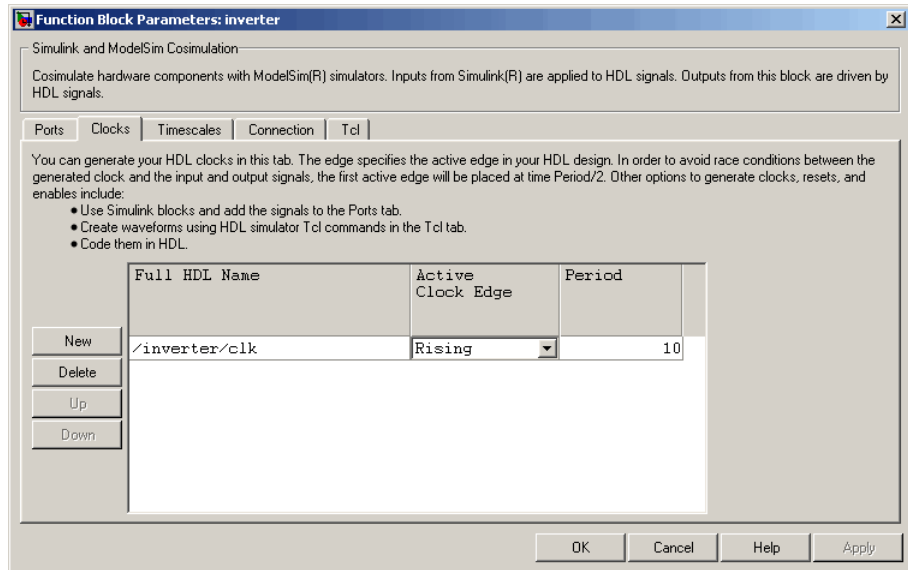
5 Click **Apply**.

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

1 Click the **Tcl** tab.

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

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

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

```
echo "Done"
```

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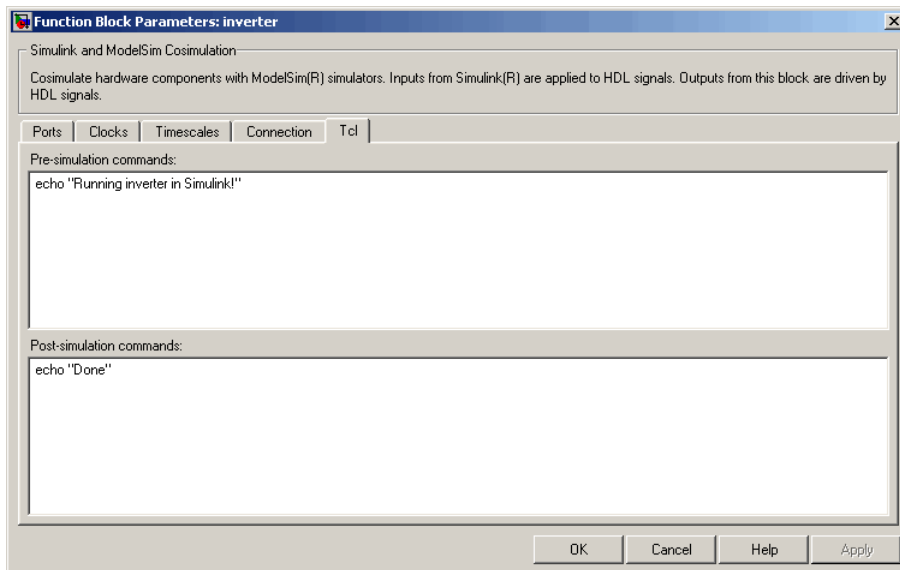
4 Click **Apply**.

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

1 Click the **Timescales** tab.

2 The default settings of the **Timescales** pane are shown below. These settings are required for correct operation of this example. See "Representation of Simulation Time" on page 3-15 for further information.
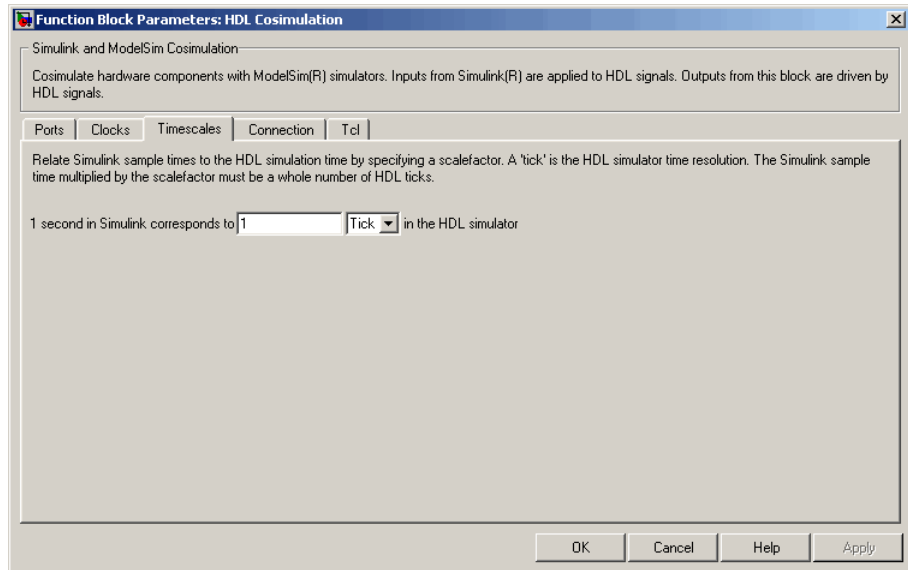
3 Click **OK** to close the Function Block Parameters dialog box.

The final step is to connect the blocks, configure model-wide parameters, and save the model:

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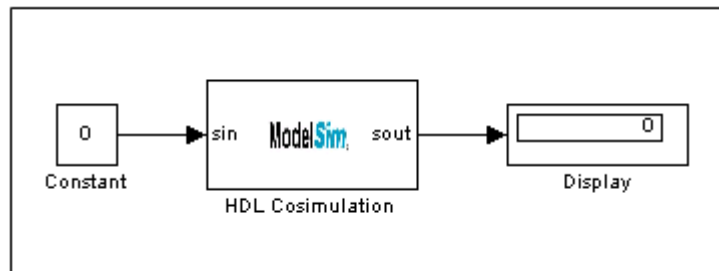
At this point, you might also want to consider adjusting block annotations.

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.
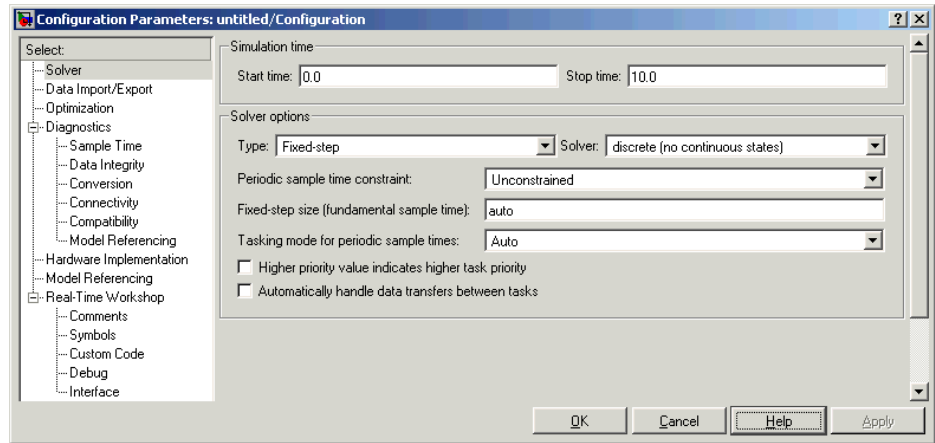
e Click **OK** to close the Configuration Parameters dialog box.

See "Setting Simulink Configuration Parameters" on page 3-40 for further information on Simulink settings that are optimal for use with 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 HDL Cosimulation block in Simulink, replace the port number 4449 in the preceding command line with the correct socket port information for your model. The `vsim` function informs ModelSim of the TCP/IP socket to use for establishing a communication link with your Simulink model.

Loading Instances of the VHDL Entity for Cosimulation with Simulink

This section explains how to use the `vsimulink` command to load an instance of your VHDL entity for cosimulation with Simulink. The `vsimulink` command is 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.
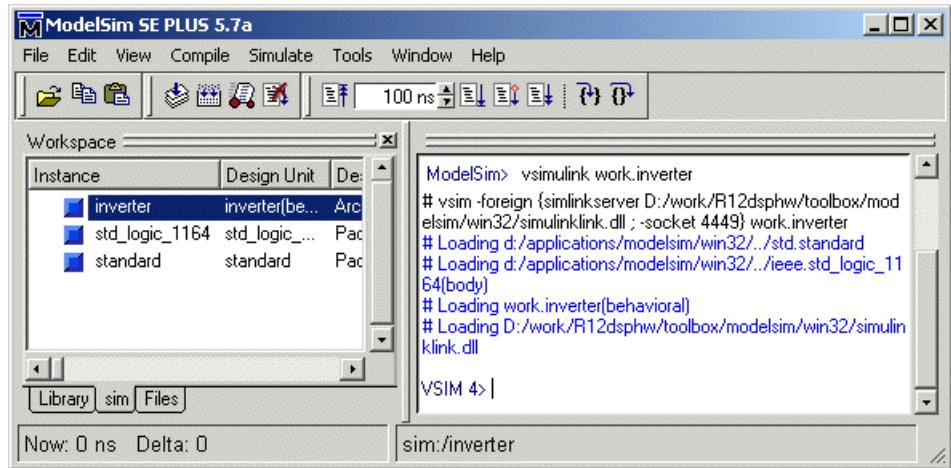
Running the Simulation

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 n> add wave /inverter/*
```

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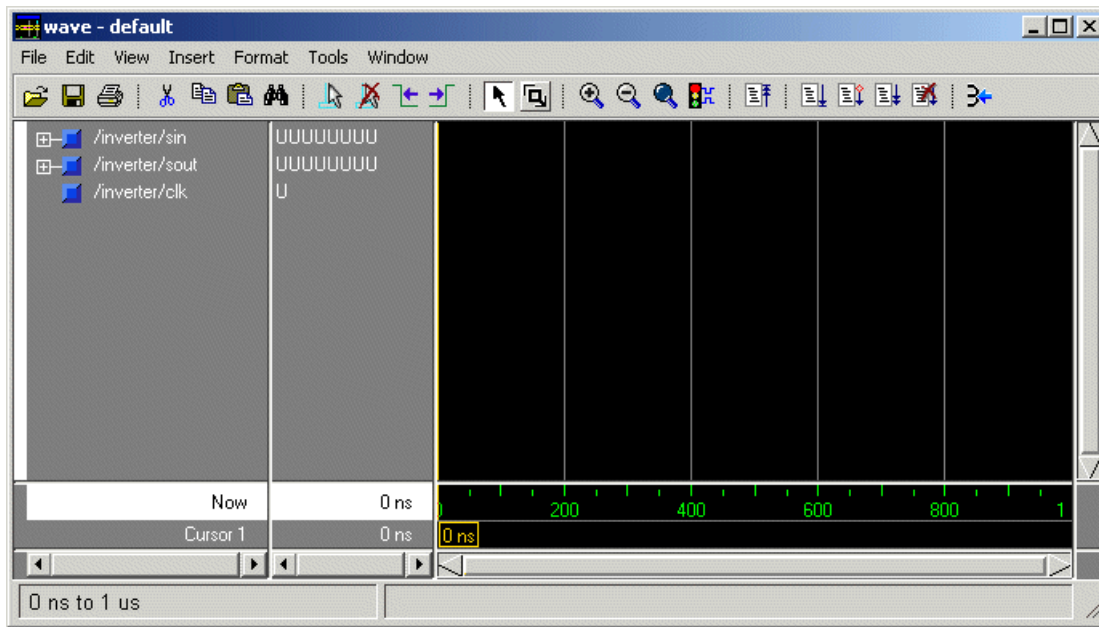
2 Change your input focus to your Simulink model window.

3 Start a Simulink simulation. The value in the Display block changes to 255. Also note the changes that occur in the ModelSim **wave** window. You might need to zoom in to get a better view of the signal data.
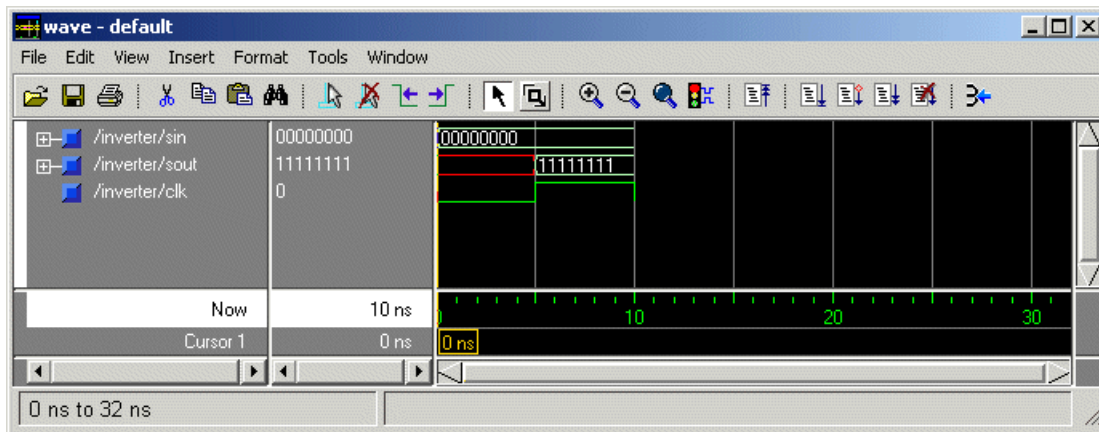
4 In the Simulink model, change **Constant value** to 255, save the model, and start another simulation. The value in the Display block changes to 0 and the ModelSim **wave** window is updated as follows.
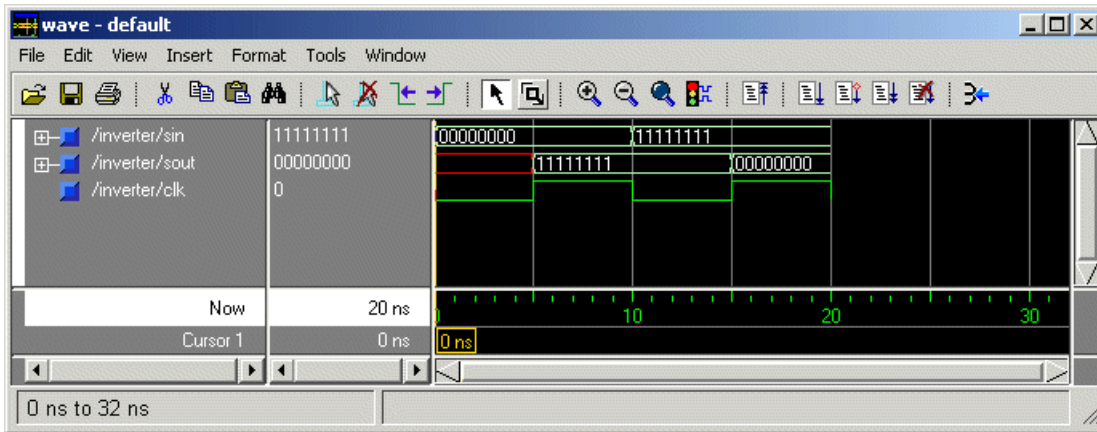
5 In the Simulink Model, change **Constant value** to 2 and **Simulation time** to 20 and start another simulation. This time, the value in the Display block changes to 253 and the ModelSim **wave** window appears as shown below.
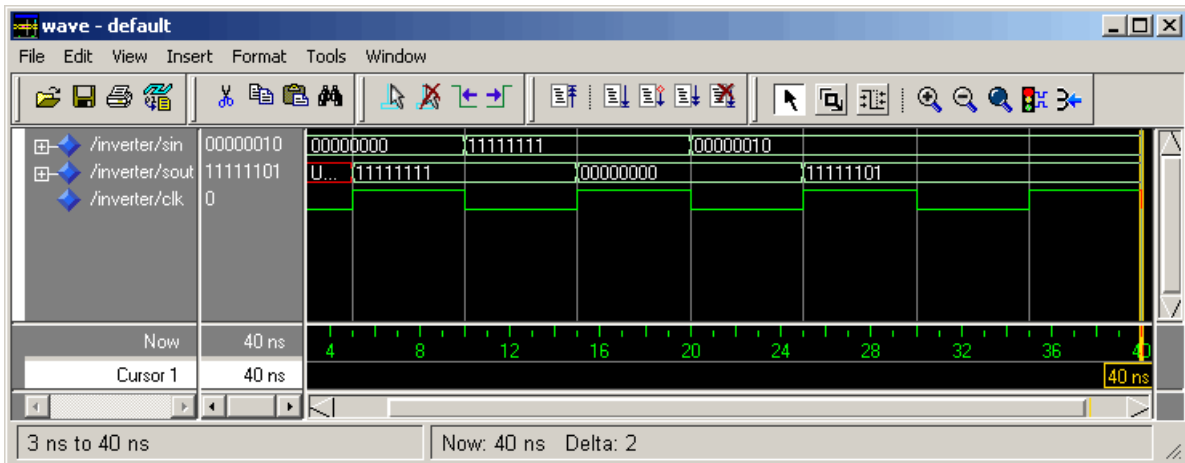
Note the change in the sample time in the **wave** window.

Shutting Down the Simulation

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

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

2 Quit ModelSim.

3 Close the Simulink model window.

toVCD Block Tutorial

VCD files include data that can be graphically displayed or analyzed with postprocessing tools. An example of such a tool is the ModelSim `vcd2wlf` tool, which converts a VCD file to a WLF file that you can then view in a ModelSim **wave** window. This section shows how you might apply the `vcd2wlf` tool:

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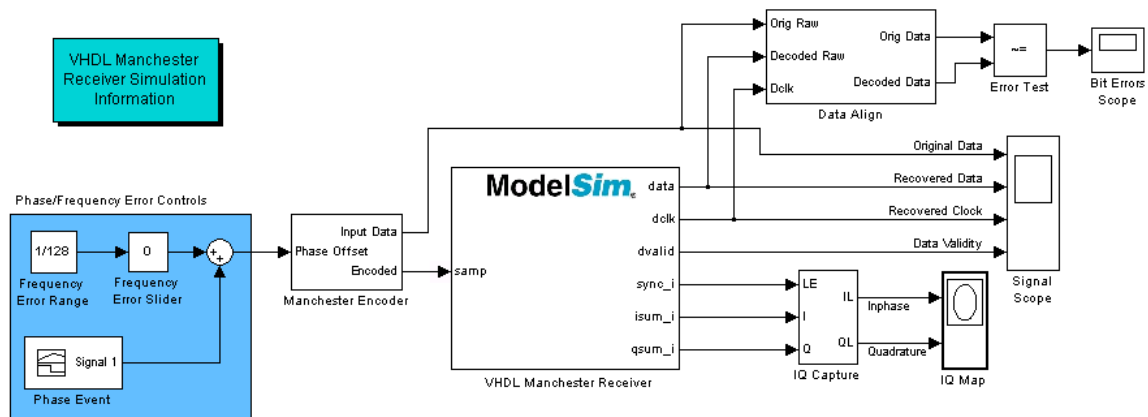
Before running this model you must first launch ModelSim.
You can launch ModelSim on this computer using either a
shared memory link or a TCP/IP socket link.

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

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

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

ModelSim Startup Command(Shared Memory)

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

ModelSim Startup Command(TCP/IP Socket)

3 Open the Library Browser.

4 Replace the Signal Scope block with a To VCD File block.

   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.

3-93

c Copy the To VCD File block from the Library Browser to the model by clicking the block and dragging it from the browser to the location in your model window previously occupied by the Signal Scope block.

d Double-click the To VCD File block icon. The Block Parameters dialog appears.
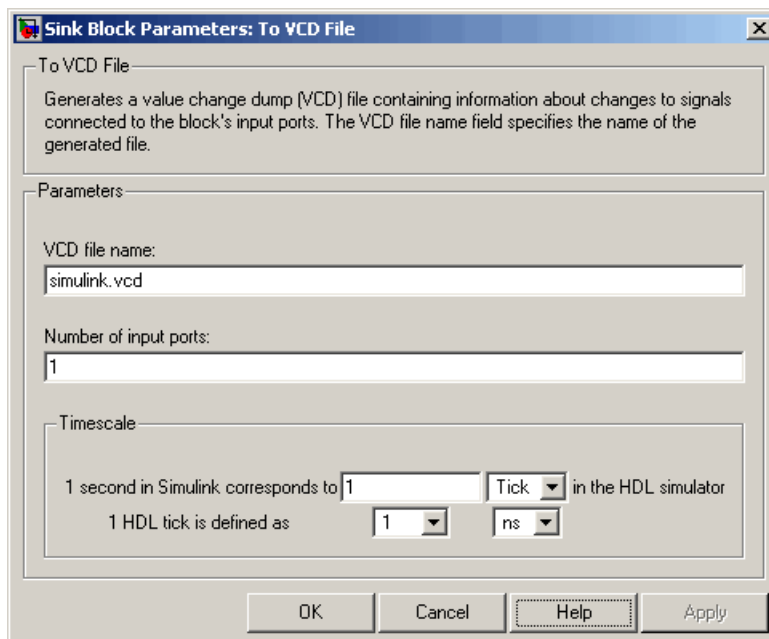
e Type MyVCDfile.vcd in the **VCD file name** text box.

f Type 4 in the **Number of input ports** text box.

g Click **OK**. Simulink applies the new parameters to the block.

5 Connect the signals Original Data, Recovered Data, Recovered Clock, and Data Validity to the block ports. The following display highlights the modified area of the model.
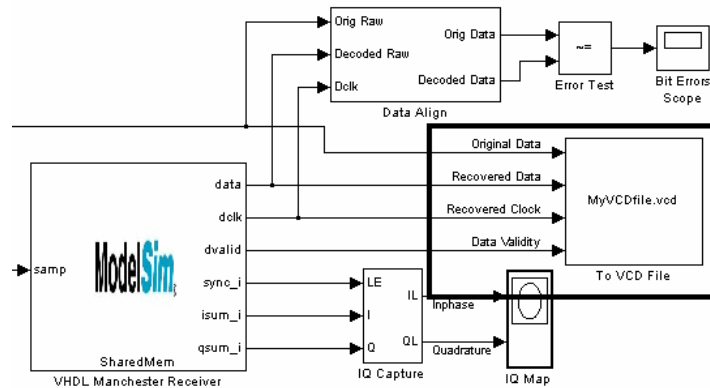
6 Save the model.

7 Select the following command line from the instructional text that appears in the demonstration model:

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

8 Paste the command in the MATLAB Command Window and execute the command line. This command starts ModelSim and configures it for a Simulink cosimulation session.

Note You might need to adjust the TCP/IP socket port. The port you specify in the vsim command must match the value specified for the HDL Cosimulation block. To check the port setting for that block, double-click the block icon and then select the **Connection** tab in the Block Parameters dialog.

---

9 Start the simulation from the Simulink model window.

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

11 Close the VCD file.

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

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

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

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

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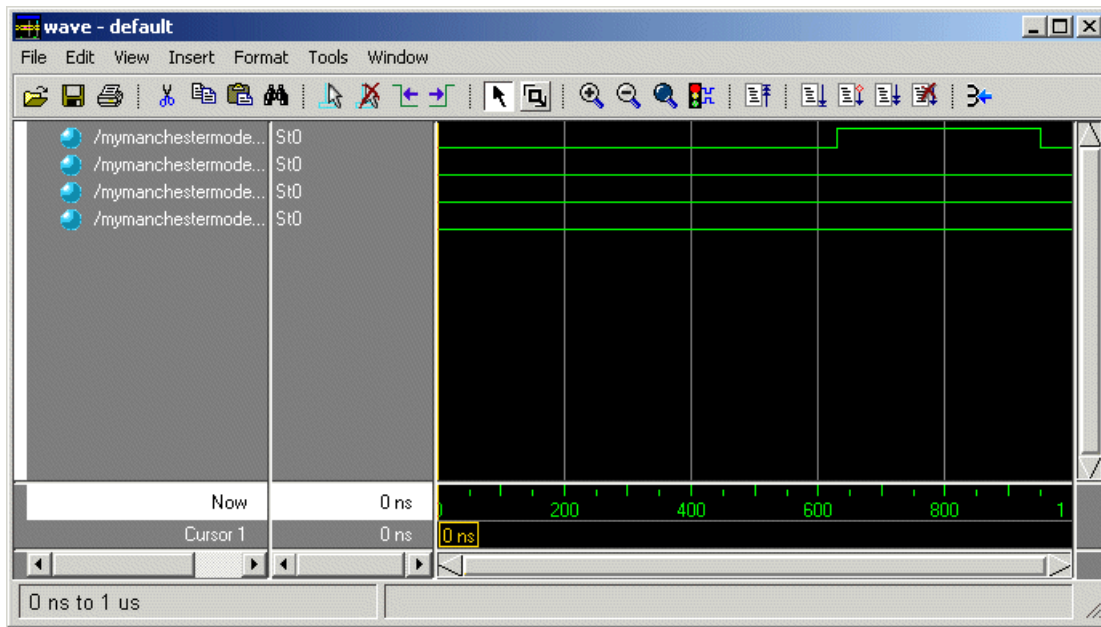
17 Exit the simulation. One way of exiting is to enter the following command:

```
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
Configure ModelSim for use with Link for ModelSim

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

Description

Note `configuremodelsim` has been replaced by the guided setup script (`syschecklfm`) for configuring your simulator setup. Although `configuremodelsim` supported for backward compatibility, it is recommended that you use the setup script instead. See "Using the Configuration and Diagnostic Script for UNIX/Linux" on page 1-11.

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

- To configure ModelSim so that it may access Link for ModelSim when invoked from outside of MATLAB

- To add Tcl commands to the Tcl startup script that runs every time ModelSim is invoked

When it is used without any arguments, `configuremodelsim` prompts you to either let `configuremodelsim` find the installed ModelSim executable or provide the path to the ModelSim installation you want to use. If no previous configuration was performed (no Tcl DO file), `configuremodelsim` creates a new `ModelSimTclFunctionsForMATLAB.tcl` script in the `tcl` directory under the ModelSim installation. If a previous configuration exists, `configuremodelsim` prompts you to decide if you want to replace the existing configuration.

`configuremodelsim('PropertyName', 'PropertyValue'...)` starts an interactive or programmatic script (depending on which property name/value pairs you select) that allows you to customize the ModelSim configuration. See "Property Name/Property Value Pairs" on page 4-4.

After you call this function, ModelSim is ready to use Link for ModelSim when ModelSim is invoked from outside of MATLAB. You can use Link for ModelSim commands from the ModelSim environment to

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

- Initiate MATLAB test bench or component sessions for loaded instances

- Terminate MATLAB test bench or component sessions

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

Usage Notes

`configuremodelsim` is intended to be used for setting up ModelSim and MATLAB when you plan to start ModelSim from outside of MATLAB. If you intend to invoke `vsim` from the MATLAB prompt then you do not need to use `configuremodelsim`. (MATLAB will find `vsim` if it is already on the system path, and, if it is not, you can set the `vsimdir` property value of `vsim` in MATLAB to provide the path information.) In addition, if you are starting ModelSim from outside of MATLAB, you should define your environment with the path to the ModelSim executable before running `configuremodelsim`.

The `vsimdir` property value of `configuremodelsim` only instructs `configuremodelsim` where to put the Tcl DO file. It does not set up MATLAB workspace for MATLAB invocation of ModelSim (this setup is done instead with the `vsimdir` property value of `vsim`).

If you are using `configuremodelsim` to add Tcl commands to the Tcl startup DO file, to change the location of the Tcl startup DO file, or to remove the Tcl startup DO file, you can run `configuremodelsim` as many times as you desire.

configuremodelsim

> **Note** You need to run `configuremodelsim` only once to set the location of the Tcl DO file (you may run `configuremodelsim` multiple times to add additional Tcl commands to the Tcl startupDO file).

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

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

Property Name/Property Value Pairs

`'action', 'install'`
   Instructs `configuremodelsim` to create a new `ModelSimTclFunctionsForMATLAB.tcl` script.

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

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

`'action', 'uninstall'`
   Removes the Link for ModelSim configuration from the ModelSim startup DO file. The contents of `ModelSimTclFunctionsForMATLAB.tcl` are replaced with this single line of text: "# MATLAB and Simulink option was deconfigured."

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

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

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

`'vsimdir'`, `'pathname'`
Specifies where to put the Tcl script containing Link for ModelSim Tcl commands. This script is programmatic only; if no directory is specified with this property, `configuremodelsim` uses the first `vsim` it encounters on the system path and installs the Tcl DO file (`ModelSimTclFunctionsForMATLAB.tcl`) in the `\tcl` directory under this ModelSim installation.

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

```
configuremodelsim
```

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

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

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

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

configuremodelsim

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

When `configuremodelsim` is run on an existing configuration, the dialog looks like this:

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

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

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

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

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

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

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

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

vsimoptions =

echo Starting ModelSim and Link for ModelSim

Modelsim successfully configured to be used with MATLAB and Simulink
```
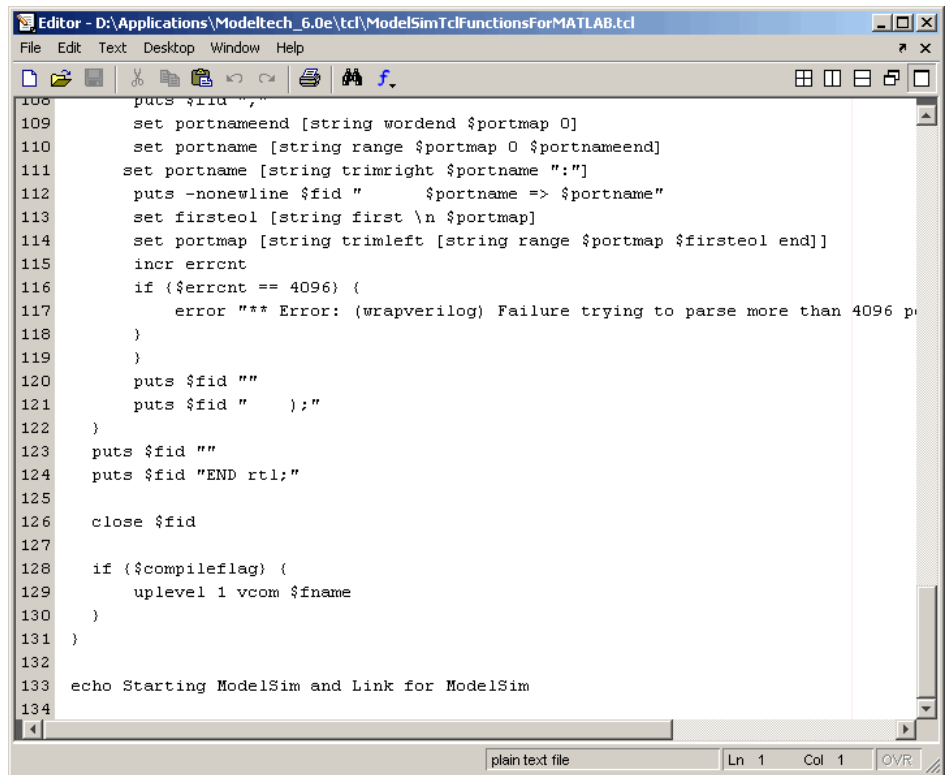
If you now inspect `ModelSimTclFunctionsForMATLAB.tcl` you will find this last Tcl command appended to the file, as shown in the following graphic.

The following example demonstrates removing the Link for ModelSim configuration from ModelSim:

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

Identify the Modelsim installtion to be deconfigured for MATLAB and Simulink

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

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

configuremodelsim

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

If you now inspect ModelSimTclFunctionsForMATLAB.tcl you will find
that the contents of the file have been removed.
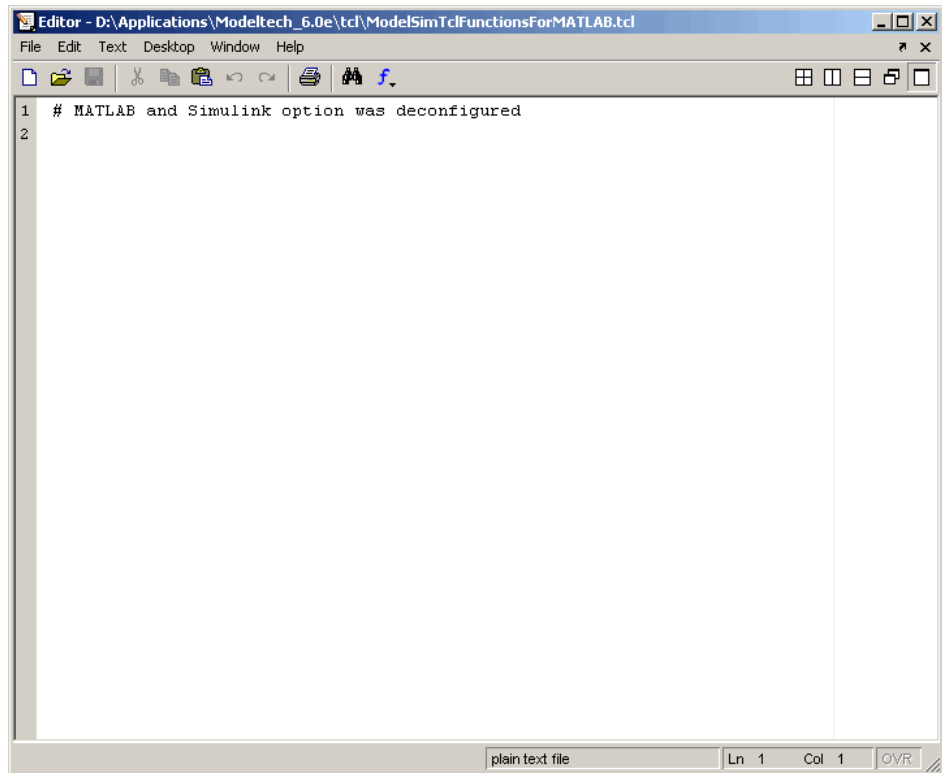
Purpose   Start MATLAB server component of Link for ModelSim interface

Syntax
```
hdldaemon
hdldaemon('PropertyName', 'PropertyValue'...)
hdldaemon('status')
hdldaemon('kill')
```

Description   **Server Activation**

hdldaemon starts the MATLAB server component of 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 1-7. For more information on establishing the ModelSim end of the communication link, see "Scheduling Options for a Link Session" on page 2-40.

---

hdldaemon('PropertyName', 'PropertyValue'...) starts
the MATLAB server component of 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 4-11
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.
For example:

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

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

Server Shutdown

hdldaemon('kill') shuts down the MATLAB server without shutting down MATLAB.

Property Name/Property Value Pairs

'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. See "Specifying TCP/IP Values" on page 2-40 for some valid tcp_spec examples.

For more information on choosing TCP/IP socket ports, see "Choosing TCP/IP Socket Ports" on page D-2.

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

`'tclcmd', 'command'`
Passes a TCL command string, to be executed in ModelSim, from MATLAB to ModelSim.

Note The TCL command string 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`.

`'quiet', 'true'`
Suppresses printing messages to the standard queue. Errors are still shown.

The following table provides guidelines on when and how to specify property name/property value pairs.

If Your Application Is to...	Do the Following...
Operate in shared memory mode	Omit the `'socket'`, *tcp_spec* property name/property value pair. The interface operates in shared memory mode by default. You should use shared memory mode if your application configuration consists of a single system and uses a single communication channel.
Operate in TCP/IP socket mode, using a specific TCP/IP socket port	Specify the `'socket'`, *tcp_spec* property name and value pair. The *tcp_spec* can be a socket port number or service name. Examples of valid port specifications include `'4449'`, `4449`, and `MATLAB Service`. For information on choosing a TCP/IP socket port, see "Choosing TCP/IP Socket Ports" on page D-2.
Operate in TCP/IP socket mode, using a TCP/IP socket that the operating system identifies as available	Specify `'socket'`, `0` or `'socket'`, `'0'`.
Return time values in seconds (type `double`)	Specify `'time'`, `'sec'` or omit the parameter. This is the default time value resolution.
Return 64-bit time values (type `int64`)	Specify `'time'`, `'int64'`.
Suppress server shutdown message when using `hdldaemon` to get an unused socket number (message can appear confusing)	Specify `'quiet'`, `'true'`.

Examples The following function call starts the MATLAB server with shared memory communication enabled and a 64–bit time resolution format for the MATLAB function's 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')
```

The following function call causes the string This is a test to be displayed at the ModelSim prompt:

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

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.

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')

pingHdlSim

Purpose   Blocks cosimulation until HDL simulator is ready

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

Description pingHdlSim(timeout) blocks cosimulation by not returning until
                 the Simulink server is loaded or until the specified timeout occurs.
                 pingHdlSim returns the process ID of the HDL simulator or -1 if a
                 timeout occurs. You must enter a timeout value.

                 This function is useful if you are trying to automate a cosimulation
                 and you need to know that the Simulink server has loaded before your
                 script continues the simulation.

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

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

Examples   The following function call blocks further cosimulation until the
                 Simulink server is loaded or until 30 seconds have passed:

                     pingHdlSim(30)

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

                 The following function call blocks further cosimulation on port 5678
                 until the Simulink server is loaded or until 20 seconds have passed:

                     pingHdlSim(20, '5678')

                 The following function call blocks further cosimulation on port 5678
                 on hostname msuser until the Simulink server is loaded or until 20
                 seconds have passed:

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

tclHdlSim

Purpose Executes Tcl command in HDL simulator

Syntax

```
tclHdlSim(tclCmd)
tclHdlSim(tclCmd,'portNumber')
tclHdlSim(tclCmd, 'portnumber', 'hostname')
```

Description `tclHdlSim(tclCmd)` executes a Tcl command on the HDL simulator using a shared connection.

`tclHdlSim(tclCmd,'portNumber')` executes a Tcl command on the HDL simulator by connecting to the local host on port *portNumber*.

`tclHdlSim(tclCmd, 'portnumber', 'hostname')` executes a Tcl command on the HDL simulator by connecting to the host *hostname* on port *portname*.

ModelSim must be connected to MATLAB using Link for ModelSim for this function to work (see "Starting ModelSim for Use with Link for ModelSim" on page 1-21. If you start from within MATLAB, you must use `vsimmatlab`.).

Examples The following function call displays a message in the ModelSim command window using port 5678 on hostname msuser:

```
tclHdlSim('puts "Done"', '5678', 'msuser')
```

Purpose   Start and configure ModelSim for use with 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 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 or Verilog modules for simulations that use MATLAB for verification

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

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

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

Property Name/Property Value Pairs

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

`'vsimdir', 'pathname'`
    Specifies the 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
```

`'rundir'`, `'dirname'`

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

- If `dirname` is specified and the directory exists, ModelSim is run in the specified directory.

- If no `rundir` property/value pair is specified or if `dirname` is empty, ModelSim is run in the current working directory.

- If the value of `dirname` is "TEMPDIR", the function creates a temporary directory in which it runs ModelSim.

- If `dirname` is specified and the directory does *not* exist, you will get an error.

`'socketsimulink'`, `'tcp_spec'`

Specifies TCP/IP socket communication for links between ModelSim and Simulink. For TCP/IP socket communication on a single computing system, the `tcp_spec` can consist of just a TCP/IP port number or service name. If you are setting up communication between computing systems, you must also specify

the name or Internet address of the remote host. See "Specifying TCP/IP Values" on page 2-40 for some valid `tcp_spec` examples.

For more information on choosing TCP/IP socket ports, see "Choosing TCP/IP Socket Ports" on page D-2

If ModelSim and Simulink are running on the same computing system, you have the option of using shared memory for communication. Shared memory is the default mode of communication and takes effect if you omit `socketsimulink` *tcp_spec* from the function call.

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

`'startms', ['yes' | 'no']`
Determines whether ModelSim is launched from `vsim`. The default is `yes`, which launches ModelSim and creates a startup Tcl file. If `startms` is set to `no`, ModelSim is not launched, but a startup Tcl file is still created.

This startup Tcl file contains pointers to MATLAB libraries. To run ModelSim on a machine without MATLAB, copy the startup Tcl file and MATLAB library files to the remote machine and start ModelSim manually. See "Using the Link for ModelSim Libraries" on page 1-16.

`'libdir', 'directory'`
Specifies the directory containing MATLAB library files. This property creates an entry in the startup Tcl file that points to the directory with the libraries needed for ModelSim to communicate with MATLAB when ModelSim is running on a machine that does not have MATLAB.

vsim

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 component function with instantiated VHDL entity or Verilog module

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 or Verilog module 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 or Verilog module 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`). See "Specifying TCP/IP Values" on page 2-40 for some valid `tcp_spec` examples.

For more information on choosing TCP/IP socket ports, see "Choosing TCP/IP Socket Ports" on page D-2.

If ModelSim and 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 1-7. For more information on establishing the MATLAB end of the communication link, see "Starting the MATLAB Server" on page 2-51.

`-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.

`-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.

`-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 in the hierarchy of the HDL model.

`-mfunc <name>`
> The name of the MATLAB function that is attached to the HDL instance you specify for `instance`. If you omit this argument, `matlabcp` attaches the HDL instance to a MATLAB function that has the same name as the HDL instance. For example, if the HDL instance is `myfirfilter`, `matlabcp` associates the HDL

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

Description The `matlabcp` command

- Starts the ModelSim client component of Link for ModelSim.
- Associates a specified instance of a VHDL entity or Verilog module created in ModelSim with a MATLAB function.
- Creates a process that schedules invocations of the specified MATLAB function.

MATLAB component functions simulate the behavior of entities in the HDL model. A stub entity or module (providing port definitions only) in the HDL model passes its input signals to the MATLAB component function. The MATLAB component processes this data and returns the results to the outputs of the stub entity or module. A MATLAB component typically provides some functionality (such as a filter) that is not yet implemented in the HDL code. See "Coding MATLAB Link Functions" on page 2-12.

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 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\modsimosc.m

Purpose   Schedule MATLAB test bench session for instantiated VHDL entity or Verilog module

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

Arguments   `<instance>`
> Specifies the instance of a VHDL entity or Verilog module 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 or Verilog module 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`). See "Specifying TCP/IP Values" on page 2-40 for some valid `tcp_spec` examples.

For more information on choosing TCP/IP socket ports, see "Choosing TCP/IP Socket Ports" on page D-2.

If ModelSim and 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 1-7. For more information on establishing the MATLAB end of the communication link, see "Starting the MATLAB Server" on page 2-51.

-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 .

-mfunc <name>
    The name of the MATLAB function that is attached to the HDL instance you specify for instance. If you omit this argument, matlabtb attaches the HDL instance to a MATLAB function that has the same name as the HDL instance . For example, if the HDL instance is myfirfilter, matlabtb associates the HDL instance with the MATLAB function myfirfilter. If you omit

matlabtb

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 Link for ModelSim.

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

- Creates a process that schedules invocations of the specified MATLAB function.

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

Note   For 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 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 Link for ModelSim, and initiates a remote TCP/IP socket-based session using remote MATLAB host `compb` and TCP/IP port 4449. Based on the specified test bench stimuli, `myfirfilter.m` executes 10 nanoseconds from the current time, each time signal `\work\fclk` experiences a rising edge, and each time signal `\work\din` changes state.

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

matlabtbeval

Purpose   Call specified MATLAB function once and immediately on behalf of instantiated VHDL entity or Verilog module

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

Arguments   `<instance>`
                 Specifies the instance of a VHDL entity or Verilog module 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. See "Specifying TCP/IP Values" on page 2-40 for some valid `tcp_spec` examples.

                 For more information on choosing TCP/IP socket ports, see "Choosing TCP/IP Socket Ports" on page D-2.

                 If ModelSim and 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
communication modes, see "Modes of Communication" on page
1-7. For more information on establishing the MATLAB end of
the communication link, see "Starting the MATLAB Server" on
page 2-51.

`-mfunc <name>`
 The name of the MATLAB function that is attached to the HDL
 instance you specify for `instance`. If you omit this argument,
 `matlabtbeval` attaches the HDL instance to a MATLAB function
 that has the same name as the HDL instance. For example, if
 the HDL instance is `myfirfilter`, `matlabtbeval` associates the
 HDL instance with the MATLAB function `myfirfilter`. If you
 omit this argument and `matlabtbeval` does not find a MATLAB
 function with the same name, the command displays an error
 message.

Description The `matlabtbeval` command

- Starts the ModelSim client component of Link for ModelSim.

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

- Executes the specified MATLAB function once and immediately on
  behalf of the specified entity instance.

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

> **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 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 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
```

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 or Verilog module for verification with MATLAB
Syntax	vsimmatlab <instance> [<vsim_args>]

Arguments	<instance>
	    Specifies the instance of a VHDL entity or Verilog module 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 or Verilog module for cosimulation
with Simulink

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

Argument   <instance>
                     Specifies the instance of a VHDL entity or Verilog module to load
                     for cosimulation.

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

                     For more information on choosing TCP/IP socket ports, see
                     "Choosing TCP/IP Socket Ports" on page D-2.

                     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 1-7.
                     For more information on establishing the Simulink end of the
                     communication link, see "Configuring the Communication Link in
                     the HDL Cosimulation Block" on page 3-61.

                     ---

vsimulink

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

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
```

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 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
```

Note Link for ModelSim now supports Verilog models directly, without requiring a VHDL wrapper. All Link for ModelSim MATLAB functions, and the HDL Cosimulation block, offer the same language-transparent feature set for both Verilog and VHDL models. The wrapverilog function is supported for backward compatibility, and is still in use by many Link for ModelSim demos.

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

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

Simulink Blocks — Alphabetical List

HDL Cosimulation

Purpose   Cosimulate hardware component by communicating with HDL model executing in ModelSim

Library   Link for ModelSim

Description   The HDL Cosimulation block cosimulates a hardware component by applying input signals to and reading output signals from an HDL model under simulation in 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 HDL Cosimulation block parameters dialog box let you configure:

- Block input and output ports that correspond to signals (including internal signals) of an HDL model. You must specify a sample time for each output port; you can also specify a data type for each output port. See "Ports Pane" on page 6-3.

- Rising-edge or falling-edge clocks to apply to your model. You can specify the period for each clock signal. See "Clocks Pane" on page 6-9.

- The timing relationship between units of simulation time in Simulink and ModelSim. See "Timescales Pane" on page 6-12.

- Type of communication and communication settings used to exchange data between simulators. See "Connection Pane" on page 6-14.

- Tcl commands to run before and after the simulation. See "Tcl Pane" on page 6-19.

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

Ports Pane

The **Ports** pane provides fields for mapping signals of your HDL design to input and output ports in your block. To specify INOUT ports of your HDL model, specify one entry for the signal in the Ports Pane as an input and another entry as an output. The signals can be at any level of the HDL 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.

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

- Verilog signals of `wire` or `reg` type
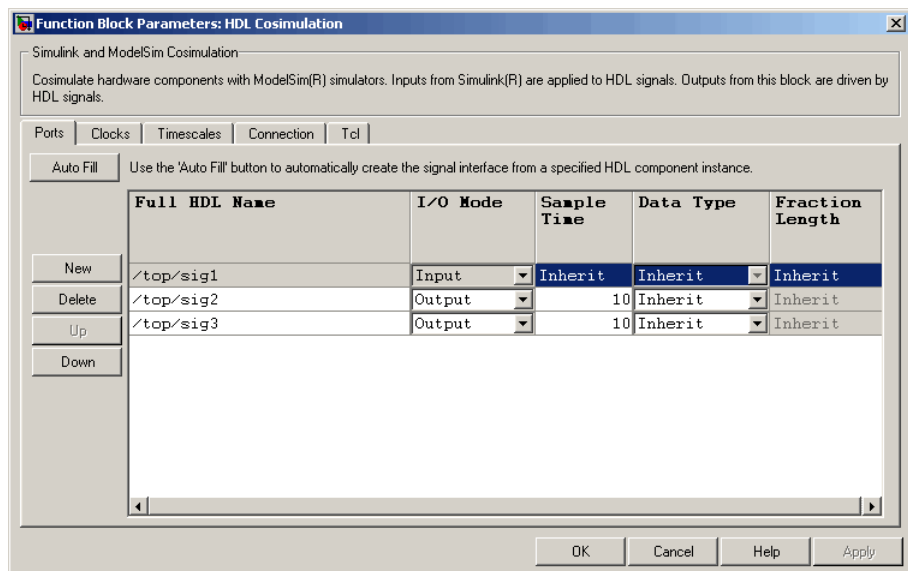
The list at the center of the pane displays HDL signals corresponding to
ports on the Cosimulation block.

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

- **Auto Fill**: Transmit a port information request to ModelSim. The
  port information request returns port names and information from
  an HDL model under simulation in ModelSim, and automatically
  enters this information into the ports list. See "Obtaining Signal

Information Automatically from ModelSim" on page 3-49 for a detailed description of this feature.

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

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

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

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

To commit edits to the Simulink model, you must also click **Apply**, located at the bottom of the pane.

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

Full HDL Name

Specifies the signal 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 HDL design hierarchy. The Cosimulation block port corresponding to the signal is labeled with the **Full HDL Name**.

For rules on specifying signal/port and module path specifications in Simulink, see "Specifying HDL Signal/Port and Module Paths for Cosimulation" on page 3-43. These rules are for signal/port and module path specifications in Simulink. Other specifications may work but are not guaranteed to work in this or future releases.

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, click **Apply**.

I/O Mode

Select either `Input` or `Output`.

- `Input` designates signals of your HDL model that are to be driven by Simulink. Simulink deposits values on the specified ModelSim signal at the signal's sample rate.

  When you define a block input port, make sure that only one source is set up to force input to that signal. If multiple sources (in Simulink or in HDL) drive values on a single signal, your simulation model may product unexpected results.

- `Output` designates signals of your HDL 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).

To specify `INOUT` ports of your HDL model, specify one entry for the signal in the Ports Pane as an input and another entry as an output. When you define a block to interface to an `INOUT` port in the HDL, the same care must be taken as in a pure HDL design to ensure that the signal can be resolved to a valid logic value (0 or 1).

Sample Time

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

Sample Time represents the time interval between consecutive samples applied to the output port. The default sample time is `1`. The exact interpretation of the output port sample time depends on the settings of the **Timescales** pane of the HDL Cosimulation block (see "Timescales Pane" on page 6-12). See also "Representation of Simulation Time" on page 3-15.

Data Type
Fraction Length

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

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

The **Data Type** and **Fraction Length** properties will apply only to

- VHDL signals of STD_LOGIC or STD_LOGIC_VECTOR type

- Verilog signals of wire or reg type

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

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

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

  Simulink 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 VHDL data type STD_LOGIC_VECTOR for a signal of size N bits, the data type ufixN is forced on the output port. (The implicit fraction length is 0.)

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

  For example, if you specify **Data Type** as Signed and a **Fraction Length** of 5 for a 16-bit VHDL 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 VHDL 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.

Clocks Pane

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.

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

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

- If the clock period (whether explicitly specified or defaulted) is not an even integer, Simulink cannot create a 50% duty cycle, and therefore Link for ModelSim creates the falling edge at

$$clockperiod/2$$

(rounded down to the nearest integer).

The following figure shows a timing diagram that includes rising-edge and falling-edge clocks with a Simulink sample period of T=10 and 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 scrolling list at the center of the pane displays HDL clocks that drive values to the HDL signals that you are modeling, using the deposit method.

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

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

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

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

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

To commit edits to the Simulink model, you must also click **Apply**, located at the bottom of the pane.

To edit the name of a clock signal, double-click it and enter the correct name. To edit the properties of a clock signal, select the appropriate property in that signal row. The properties of a clock signal are

Full HDL Name

> Specify each clock as a signal pathname, using ModelSim pathname syntax. A sample pathname for a clock might be `/manchester/clk`.
>
> For information about and requirements for path specifications in Simulink, see "Full HDL Name" under "Ports Pane" on page 6-3.
>
> ---
>
> **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, click **Apply**.
>
> ---

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.

Note Vectored signals in the **Clocks** pane are not supported. Signals must be logic types with '1' and '0' values.

Timescales Pane

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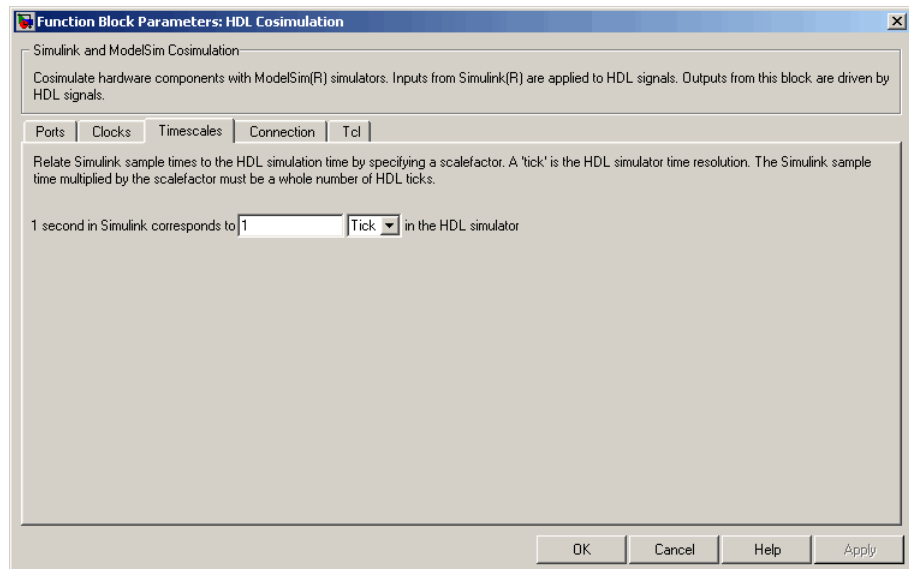
The **Timescales** pane specifies a correspondence between one second of Simulink time and some quantity of ModelSim time. This quantity of ModelSim time can be expressed in one of the following ways:

- 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 list on the right, and enter the desired number of ticks in the edit box. For example, in the figure below the **Timescales** pane is configured for a relative timing correspondence of 10 ModelSim ticks to 1 Simulink second.
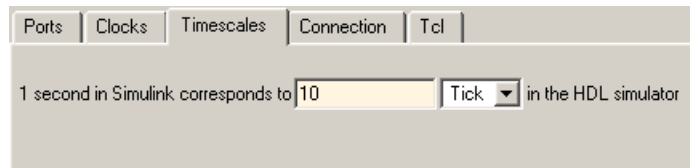
- In *absolute* units (such as milliseconds or nanoseconds). In this case, the cosimulation is said to operate in *absolute timing mode*.

  To use absolute mode, select a unit of absolute time (available units are `fs`, `ps`, `ns`, `us`, `ms`, `s`) from the list on the right. Then enter a scale factor in the edit box. For example, in the figure below the **Timescales** pane is configured for an absolute timing correspondence of 1 ModelSim second to 1 Simulink second.
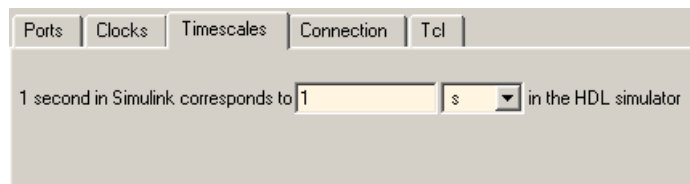
To set the absolute time, you must know the value of the HDL simulator tick (resolution unit) to understand how Link for ModelSim handles the timing of the falling edge when the duty cycle does not fall at 50%. The following restrictions apply to clock periods:

- You must enter a sample time equal to or greater than 2 resolution units (ticks) (no falling edge can occur in < 2 ticks).

- If the clock period (whether explicitly specified or defaulted) is not an even integer multiple, Simulink cannot create a 50% duty cycle, and therefore Link for ModelSim creates the falling edge at

  clockperiod/2

  (rounded down to the nearest integer).

  You must know how many ticks your selected time represents so that you know how the falling edge will occur. This next example demonstrates how to calculate the number of HDL simulator ticks for an absolute clock period of 1 Simulink second = 3 HDL simulator seconds.

  ```
  1 HDL simulator second = 10⁹ HDL simulator ns
  1 HDL simulator tick = 10 HDL simulator ns
  1 HDL simulator second = (10⁹/10) or 10⁸ HDL simulator ticks

  1 Simulink seconds = 3 HDL simulator seconds
  1 Simulink second = 3x10⁸ HDL simulator ticks
  ```

  In this example, the number of ticks is greater than 2 and an even integer multiple, therefore the duty cycle will fall at 50%. If 1 HDL simulator tick was instead equal to 13 ns, the end result would have the falling edge occur at 1153846153 ticks, or a just under 50% duty cycle.

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 3-15

Connection Pane

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 **Connection** pane also

provides the option for bypassing the HDL Cosimulation block during Simulink simulation.

The **Clocks** pane lets you create optional rising-edge and falling-edge clocks that apply stimuli to your cosimulation model. You can either specify an explicit period for each clock, or accept a default period of 2. Simulink attempts to create a clock that has a 50% duty cycle and a predefined phase that is inverted for the falling edge case.

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

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

- If the clock period (whether explicitly specified or defaulted) is not an even integer, Simulink cannot create a 50% duty cycle, and therefore Link for ModelSim creates the falling edge at

  clockperiod / 2

  (rounded down to the nearest integer).

The following figure shows a timing diagram that includes rising-edge and falling-edge clocks with a Simulink sample period of T=10 and 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

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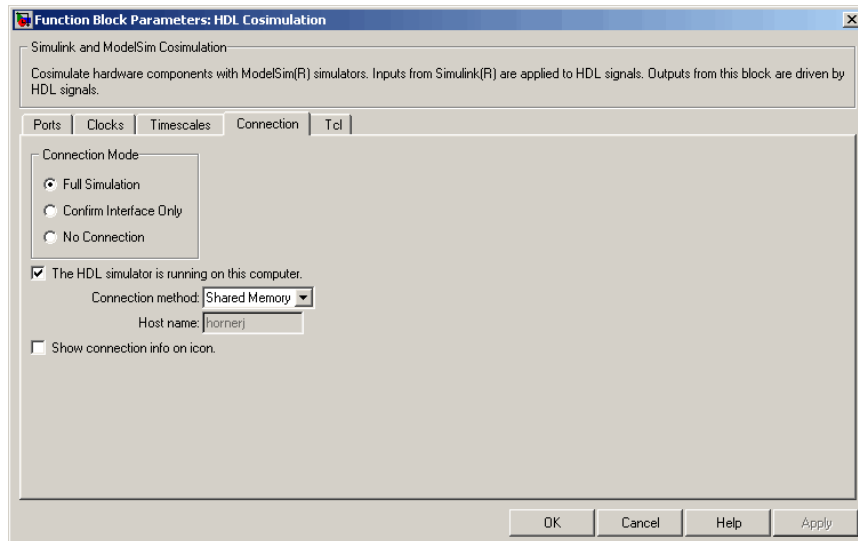
If you select TCP/IP socket mode communication, the pane displays additional properties, as shown in the figure below.
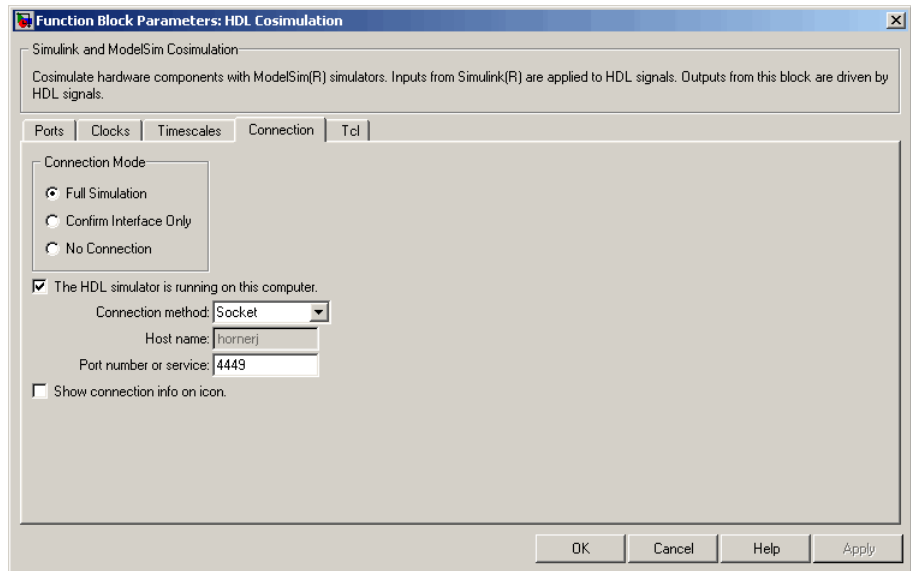
Connection Mode

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

- **Full Simulation**: Confirm interface and run HDL simulation (default).

- **Confirm Interface Only**: Check HDL simulator for proper signal names, dimensions, and data types, but do not run HDL simulation.

- **No Connection**: Do not communicate with the HDL simulator. The HDL simulator does not need to be started.

With the 2nd and 3rd options, Link for ModelSim does not communicate with the HDL simulator during Simulink simulation.

HDL Cosimulation

The HDL simulator is 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 **The HDL simulator is 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 in the HDL Cosimulation Block" on page 3-61.

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 HDL 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 D-2.

Show connection info on icon

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

In a model that has multiple HDL Cosimulation blocks, with each communicating to different instances of ModelSim in different

modes, this information helps to distinguish between different cosimulations.

Tcl Pane

The **Tcl** pane provides a way of specifying tools command language (Tcl) commands to be executed before and after ModelSim simulates the HDL component of your Simulink model. The **Pre–simulation commands** field on this pane is particularly useful for simulation initialization and startup operations, but cannot be used to change simulation state.
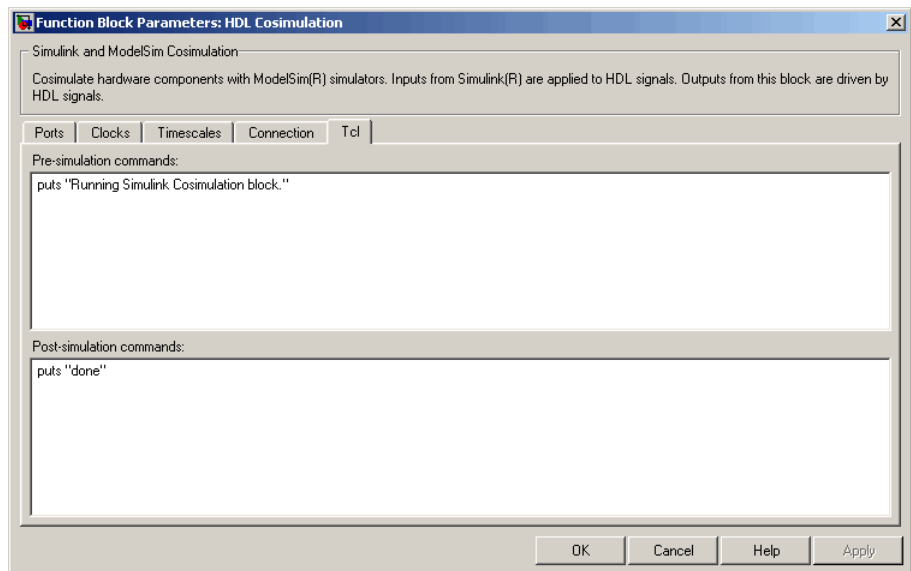
Pre–simulation commands

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

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

```
do mycosimstartup.do
```

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

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

Post–simulation commands

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

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

```
do mycosimcleanup.do
```

Notes

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

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

To VCD File

Purpose   Generate 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 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 6-25.

Note The toVCD block is integrated into the Simulink Signal & Scope Manager. See the Simulink User's Guide, "Signal & Scope Manager" for more information on using the Signal & Scope Manager.

Dialog Box
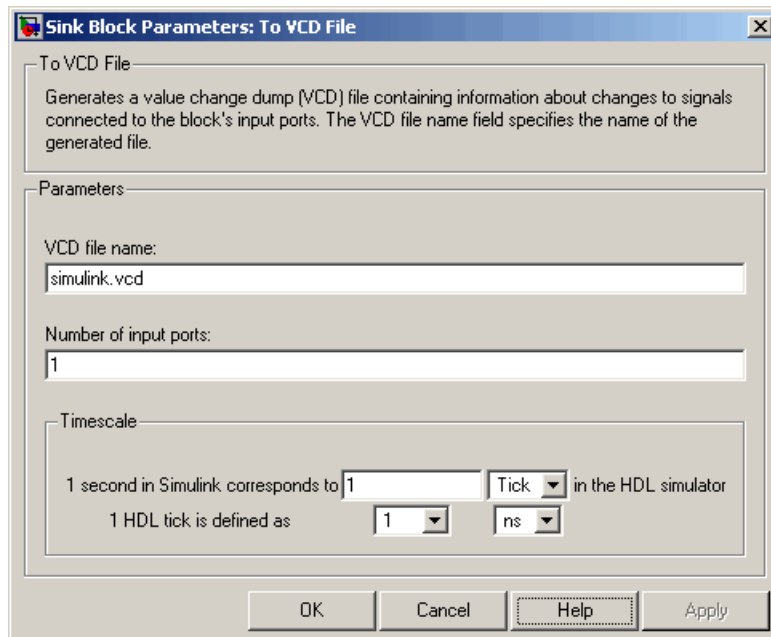
VCD file name

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.

To VCD File

> **Note** If you want the generated file to have a `.vcd` file type extension, you must specify it explicitly.
>
> Do not give the same file name to different VCD blocks. Doing so results in invalid VCD files.

Number of input ports

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

In some cases, a single input port maps to multiple signals (and symbols). This occurs when the input port receives one of the following:

- Vector of real numbers
- Fixed-point real number

> **Note** The To VCD File block does not support floating point signal types.

> **Note** Because multi-dimensional signals are not part of the VCD specification, they are flattened to a 1D vector in the file.

Timescale

Choose an optimal timing relationship between Simulink and ModelSim.

The timescale options specify a correspondence between one second of Simulink time and some quantity of HDL simulator

time. This quantity of HDL simulator time can be expressed in one of the following ways:

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

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

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

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

VCD File Format

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

Generated VCD File Format

File Content	Description
``` $date 23-Sep-2003 14:38:11 $end ```	Data and time the file was generated.

Generated VCD File Format (Continued)

File Content	Description
`$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.

`$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.

Generated VCD File Format (Continued)

File Content	Description
#0	Simulation start time.
$dumpvars 0! 0" 0# 0$ $end	Lists the values of all defined variables at time equals 0.
#630 1!	The starting point of logged value changes. Variable values are checked at each simulation time increment and are logged if a change occurs. This entry indicates that at 63 nanoseconds, the value of signal Original Data changed from 0 to 1.

6-27

Generated VCD File Format (Continued)

File Content	Description

.
.
.
#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).

Examples

Use this list to find examples in the documentation.

Invoking ModelSim Using the MATLAB Function vsim

MATLAB and ModelSim Random Number Generator Tutorial

Frame-Based Processing

Simulink and ModelSim Inverter Tutorial

Generating a VCD File

Link for ModelSim Machine Configuration Requirements

Configuration Restrictions when Using Link for ModelSim with MATLAB

The following list provides samples of valid configurations for using ModelSim with MATLAB . The scenarios apply whether ModelSim is running on the same or different computing system as MATLAB. 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.

- 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

  Although multiple ModelSim sessions can link to the same MATLAB function in the same instance of the MATLAB server, as this configuration scenario suggests, such links are not recommended. If the MATLAB function maintains state (for example, maintains global or persistent variables), you may experience unexpected results because the MATLAB function does not distinguish between callers when handling input and output data. If you must apply this configuration scenario, consider deriving unique instances of the MATLAB function to handle requests for each HDL entity.

Notes

- Shared memory communication is an option for configurations that require only one communication link on a single computing system.

- TCP/IP socket communication is required for configurations that use multiple communication links on one or more computing systems. Unique TCP/IP socket ports distinguish the communication links.

- In any configuration, an instance of MATLAB can run only one instance of the 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.

Configuration Restrictions when Using Link for ModelSim with Simulink

The following list provides samples of valid configurations for using ModelSim with Simulink. 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.

- An HDL Cosimulation block in a Simulink model linked to a single ModelSim session

- Multiple HDL Cosimulation blocks in a Simulink model linked to the same ModelSim session

- An HDL Cosimulation block in a Simulink model linked to multiple ModelSim sessions

- Multiple HDL Cosimulation blocks in a Simulink model linked to different ModelSim sessions

Notes

- HDL Cosimulation blocks in a Simulink model can connect to the same or different ModelSim sessions.

- TCP/IP socket communication is required for configurations that use multiple communication links on one or more computing systems. Unique TCP/IP socket ports distinguish the communication links.

- Shared memory communication is an option for configurations that require only one communication link on a single computing system.

ADMS Support

Adding Libraries for ADMS Support

No special library installation is needed for ADMS support.

If you need to add system libraries to the LD_LIBRARY_PATH you can add them in a .vams_setup file. Doing it this way (rather than specifying the path before calling vasim) will prevent vasim from overwriting the path addition each time it starts.

This example appends the system shared libraries to LD_LIBRARY_PATH:

```
proc fixldpath {args} {
   set pvpair [split [join $args]]
   set pval   [lindex $pvpair 1]
   append newpval /directory/of/system/dlls ":" $pval
   append setcmd { array set env [list LD_LIBRARY_PATH } " " $newpval " " ]
   uplevel 1 $setcmd
}

fixldpath [array get env LD_LIBRARY_PATH]
```

Linking MATLAB or Simulink to ModelSim in ADMS

In this section...

Starting ADMS for Use with Link for ModelSim

You must call `vasim` with all parameters manually; the configuration script available for ModelSim is not available for ADMS.

When you call `vasim`, you must provide the -ms and -foreign parameters. For example,

```
vasim -lib ADC12_ELDO_MS -cmd
     /devel/user/work/ams/adc12test.cmd TEST -ms -foreign matlabclient path/matlablibrary
```

where:

`-lib ADC12_ELDO_MS`	is the model library
`/devel/user/work/ams/adc12test.cmd`	is the command file
`TEST`	is the design
`path/matlablibrary`	is the path to and the name of the MATLAB shared library (see "Using the Link for ModelSim Libraries" on page 1-16)

A similar example for the Simulink link looks like this:

```
vasim -lib ADC12_ELDO_MS -cmd
     /devel/user/work/ams/adc12test.cmd TEST -ms -foreign simlinkserver path/simulinklibrary
```

All line arguments after 'ms' are sent to the ModelSim process.

See your ModelSim documentation for more about the -foreign option.

Using Tcl Test Bench Commands with ADMS

When you use any of the Link for ModelSim ModelSim link commands (e.g., `matlabcp`, `matlabtb`, `matlabtbeval`, `nomatlabtb`), you must precede each command with `ms` in the ADMS Tcl interpreter. For example:

```
ms matlabtb myfirfilter 5 ns -repeat 10 ns -socket 4449
```

All line arguments after 'ms' are sent to the ModelSim process.

Constraints

Setting Simulation Running Time

When cosimulating in Simulink, make sure that the runtime of the ADMS simulation is greater than or equal to the Simulink runtime.

Solaris Support

- ADMS is not supported on Solaris 64-bit, but is supported on Solaris 8, 9, 10 32-bits

- glibc errors on glnxa64 when cosimulating with ADMS

  If you get the following:

  ```
  *** glibc detected *** free(): invalid pointer: 0x0000000001eeb490 ***
  ```

  The you must do this:

  ```
  setenv MALLOC_CHECK_  O
  ```

  Which says, effectively, "Do not generate an error message, and do not kill the program".

TCP/IP Socket Communication

Choosing TCP/IP Socket Ports

Depending on your particular configuration (for example, when MATLAB and ModelSim reside on separate machines), when creating a Link for ModelSim MATLAB application or defining the block parameters of an HDL Cosimulation block, you may need to identify the TCP/IP socket port number or service name (alias) to be used for Link for ModelSim connections.

To use the TCP/IP socket communication, you must choose a TCP/IP socket port number for the server component to listen on that is available in your computing environment. Client components can connect to a specific server by specifying the port number on which the server is listening. For remote network configurations, the Internet address helps distinguish multiple connections.

The socket port resource is associated with the server component of 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.

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.

- TCP/IP port filtering on either the client or server side can cause Link for ModelSim to fail to make a connection.

  In such cases the error messages displayed by 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
  ```

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

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

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

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

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

TCP/IP Services

By setting up the MATLAB server as a service, you can run the service in the background, allowing it to handle different HDL simulator client requests over time without you having to start and stop the service manually each time. Although it makes less sense to set up a service for Simulink as you cannot really automate the starting of an HDL simulator service, you might want to use a service with Simulink to reserve a TCP/IP socket port.

Services are defined in the `etc/services` file located on each computer; consult the user's guide for your particular operating system for instructions and more information on setting up TCP/IP services.

For remote connections, the service name must be set up on both the client and server side. For example, if the service name is "matlabservice" and you are performing a Windows-Linux cross-platform simulation, the service name must appear in the service file on both the Windows machine and the Linux machine.

Index