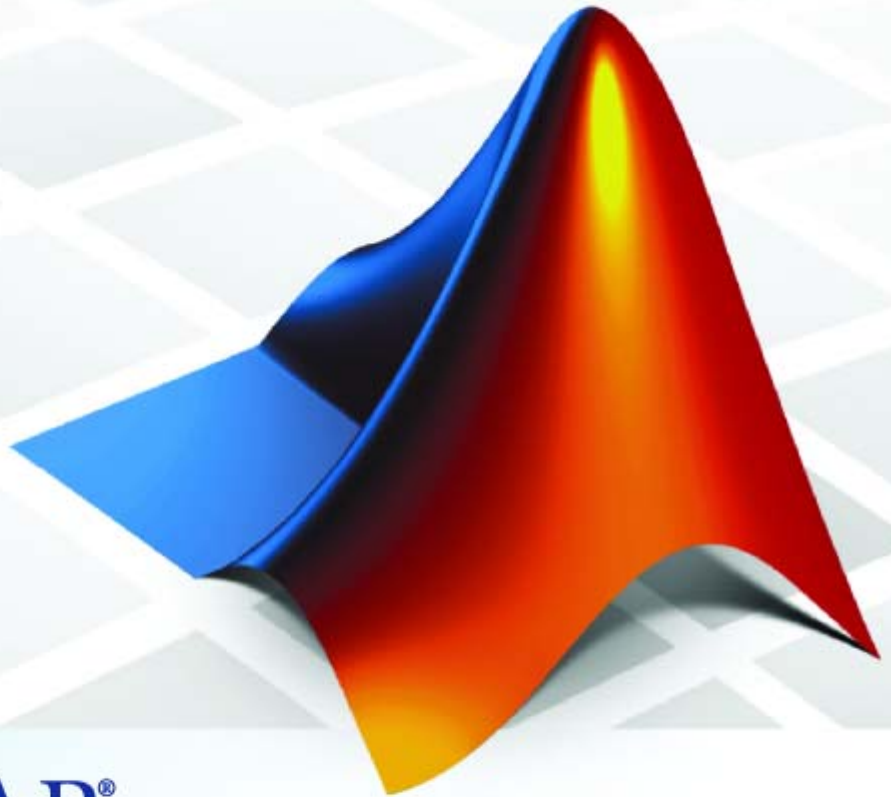


Link for Cadence® Incisive® 2 User's Guide



MATLAB®
& **SIMULINK®**

How to Contact The MathWorks



www.mathworks.com
comp.soft-sys.matlab
www.mathworks.com/contact_TS.html

Web
Newsgroup
Technical Support



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

Product enhancement suggestions
Bug reports
Documentation error reports
Order status, license renewals, passcodes
Sales, pricing, and general information



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

Link for Cadence Incisive User's Guide

© COPYRIGHT 2006–2007 by The MathWorks, Inc.

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

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

Trademarks

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

Cadence, Incisive, and the Cadence logo are either trademarks or registered trademarks of Cadence Design Systems, Inc. in the United States and/or other jurisdictions. Cadence's trademarks are used by The MathWorks, Inc. under license.

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

Patents

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

Revision History

September 2006 Online only
March 2007 Online only

New for Version 1 (Release 2006b+)
Updated for Version 2.0 (Release 2007a)

Getting Started

1

What Is the Link for Cadence Incisive Software?	1-2
Typical Applications	1-3
Expected Users	1-4
Key Features	1-5
The Cosimulation Environment	1-7
Modes of Communication	1-9
Working with MATLAB and the HDL Simulator	1-10
Working with Simulink and the HDL Simulator	1-11
Installation and Setup	1-13
What Are Your Environment Requirements?	1-13
Deciding on a Configuration	1-15
Identifying a Server in a Network Configuration	1-17
Choosing TCP/IP Socket Ports	1-18
Checking Product Requirements	1-20
Installing Related Application Software	1-22
Installing the Link for Cadence Incisive Software	1-23
Setting Up the HDL Simulator for Use with the Link for Cadence Incisive Software	1-23
Getting Help with the Link for Cadence Incisive Software	1-27
Documentation Overview	1-27
Online Help	1-28
Demos and Tutorials	1-28

Coding a Link for Cadence Incisive MATLAB Application

2

Overview	2-2
---------------------------	------------

Coding Entities or Modules for MATLAB Verification ..	2-3
Overview of Steps for Coding Entities or Modules	2-3
Choosing an Entity or Module Name	2-4
Specifying Port/Signal and Module Paths	2-4
Specifying Ports for the Entity or Module	2-6
Specifying Port Direction Modes	2-6
Specifying Port Data Types	2-6
Sample VHDL Entity Definition	2-8
Compiling the HDL Model	2-9
Coding a MATLAB Test Bench Function	2-10
Overview of the Steps for Coding a MATLAB Test Bench Function	2-10
Data Type Conversions	2-11
Naming a MATLAB Test Bench Function	2-15
Passing Parameters to and from the MATLAB Function ..	2-16
Gaining Access to and Applying Port Information	2-17
Converting Data for Manipulation	2-20
Converting Data for Return to the HDL Simulator	2-21
Sample MATLAB Test Bench Function	2-26
Coding a MATLAB Component Function	2-33
Function Definition and Parameters	2-33
Placing a MATLAB Test Bench or Component Function on the MATLAB Search Path	2-35

Starting and Controlling MATLAB Link Sessions

3

Overview	3-3
Checking the MATLAB Server's Link Status	3-5
Starting the MATLAB Server	3-7

Starting the HDL Simulator for Use with MATLAB ...	3-10
Deciding on MATLAB Link Session Scheduling Options	3-11
Controlling Callback Timing from a MATLAB Test Bench or Component Function	3-12
Initializing the HDL Simulator for a MATLAB Link Session	3-13
Applying Stimuli with the HDL Simulator force Command	3-18
Running and Monitoring a MATLAB Link Session	3-20
Stopping a MATLAB Link Session	3-22

Modeling and Verifying an HDL Design with Simulink

4

Overview	4-3
Creating a Hardware Model Design in Simulink	4-5
Handling Signal Values Across Simulators	4-7
How Simulink Drives Cosimulation Signals	4-7
Representation of Simulation Time	4-8
Handling Multirate Signals	4-15
Clock Signal Latency	4-16
Block Simulation Latency	4-16
Configuring Simulink for HDL Models	4-18

Adding the HDL Representation of a Model Component into a Simulink Model	4-19
Configuring an HDL Cosimulation Block	4-20
What Are Your HDL Cosimulation Block Requirements? ..	4-20
Opening the Block Parameters Dialog Box	4-23
Mapping HDL Signals to Block Ports	4-23
Specifying Data Types for Output Ports	4-28
Configuring the Simulink and Incisive Simulator Timing Relationship	4-30
Configuring the Communication Link	4-32
Creating Optional Clocks	4-34
Executing Tcl Commands Before and After Cosimulation	4-37
Applying Your Block Parameters Configuration Settings ..	4-40
Running and Testing a Cosimulation Model in Simulink	4-41
Using Frame-Based Processing in Cosimulation	4-42
Overview	4-42
Using Frame-Based Processing	4-42
Using a Value Change Dump File for Design Verification	4-44
Generating a VCD File	4-44
VCD File Format	4-47

MATLAB Functions — Alphabetical List

5

HDL Simulator Tcl Commands — Alphabetical List

6

Simulink Blocks — Alphabetical List

7

Index

Getting Started

What Is the Link for Cadence Incisive Software? (p. 1-2)

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

Installation and Setup (p. 1-13)

Explains how to install and set up Link for Cadence Incisive software.

Getting Help with the Link for Cadence Incisive Software (p. 1-27)

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

What Is the Link for Cadence Incisive Software?

The Link for Cadence® Incisive® cosimulation interface is software that integrates MathWorks tools into the Electronic Design Automation (EDA) workflow for application-specific integrated circuit (ASIC) and field programmable gate array (FPGA) development. The interface provides a fast bidirectional link between the Cadence Design System’s hardware description language (HDL) simulators (Incisive® simulators) 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:

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

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

The following sections discuss

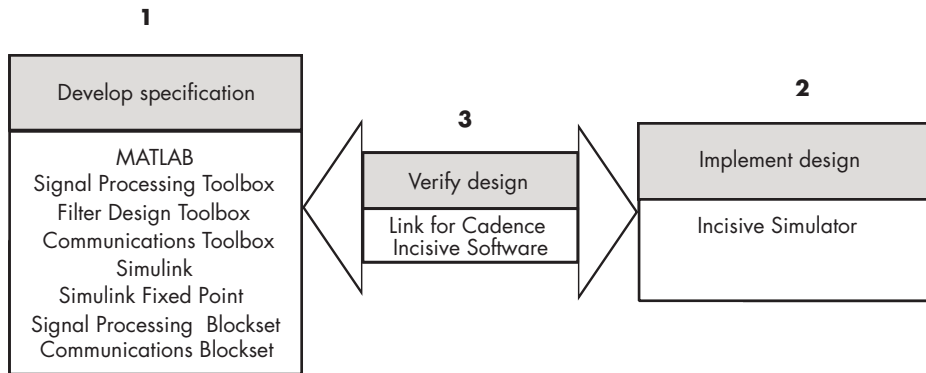
- “Typical Applications” on page 1-3
- “Expected Users” on page 1-4
- “Key Features” on page 1-5
- “The Cosimulation Environment” on page 1-7
- “Modes of Communication” on page 1-9
- “Working with MATLAB and the HDL Simulator” on page 1-10
- “Working with Simulink and the HDL Simulator” on page 1-11

Typical Applications

Link for Cadence Incisive software streamlines FPGA and ASIC development by integrating tools available for

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

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



As the figure shows, Link for Cadence Incisive software connects tools that are traditionally used discretely to accomplish specific steps in the design process. By connecting the tools, Link for Cadence Incisive software 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 Cadence Incisive software enables you to use

- MATLAB or Simulink to create test signals and software test benches for HDL code
- MATLAB or Simulink to provide a behavioral model for an HDL simulation
- MATLAB analysis and visualization capabilities for real-time insight into an HDL implementation
- Simulink to translate legacy HDL descriptions into system-level views

Expected Users

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

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

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

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

- Hardware design and system integration
- VHDL and/or Verilog
- Incisive simulators from Cadence Design Systems, Inc.
- 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 is also useful:

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

Key Features

Key features of Link for Cadence Incisive software include

- Ability to link the HDL simulator to MATLAB and Simulink for bidirectional cosimulation, verification, and visualization
- Support for Window and Unix platforms (see the MathWorks Link for Cadence Incisive product requirements page for specific platforms supported)
- Full Verilog and Native VHDL support
- MATLAB testbench capability, giving the ability to use MATLAB code to stimulate and check HDL code
- MATLAB component capability, enabling simulation of MATLAB code in place of HDL
- Frame-based simulation, providing accelerated verification (with the Signal Processing Blockset, available separately)
- User-selectable communication modes between MATLAB and Simulink and the Incisive simulator, providing shared memory (for faster performance) and TCP/IP sockets (for versatility)
- A Simulink block for cosimulating HDL models (VHDL or Verilog) in Simulink

- A Simulink block for exporting test vectors and results as value change dump (VCD) files
- Multiple simulation options from one Simulink model, including connection of multiple Simulink HDL cosimulation blocks to one or more Incisive simulators
- Interactive or batch mode cosimulation, debugging, testing, and verification of HDL code from within MATLAB
- Multiple simulation options from MATLAB, including connection of multiple MATLAB components or test benches to one or more MATLAB servers

VHDL and Verilog Language Support

All Link for Cadence Incisive MATLAB functions and the HDL Cosimulation block offer the same language-transparent feature set for both Verilog and VHDL models.

Link for Cadence Incisive software also supports mixed-language HDL models (models with both Verilog and VHDL components), allowing you to cosimulate VHDL and Verilog signals simultaneously. However, only Simulink can access components in different languages at any level; MATLAB can access signals only with the language of the top-level module instance or component.

Mixed-Language Model Limitation. The Cadence VHPI reports the incorrect simulator precision when simulating mixed Verilog/VHDL design. (It is correct when in a pure VHDL design.) In a mixed-HDL model, the VHPI *always* returns a precision of 1 fs. The actual simulator precision is properly modified by `-vhd1_time_precision`, but the returned value does not reflect that value in a mixed-HDL model.

You will get incorrect or non-running simulations if *both* the following conditions exist:

- You have a mixture of VHDL and Verilog in your design AND you have set `-vhd1_time_precision` to TP and `TP != 1fs`
- You also are cosimulating *either* of the following:
 - Only VHDL signals and there is a Simulink sample time finer than TP (after accounting for the cosimulation block timescale calculations)

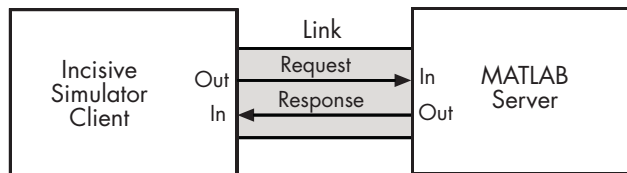
- Both VHDL and Verilog signals and TP is coarser than the Verilog time precision set by timescale or the -timescale command line and there is a Simulink sample time finer than TP (after accounting for the cosimulation block timescale calculations)

The Cosimulation Environment

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

MATLAB and HDL Simulator Links

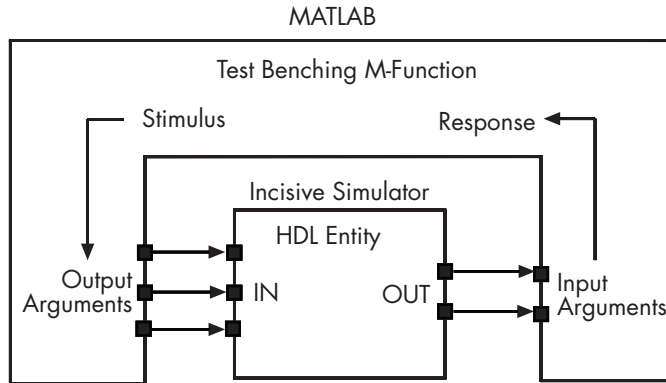
When linked with MATLAB, the HDL simulator functions as the client, as the following figure shows.



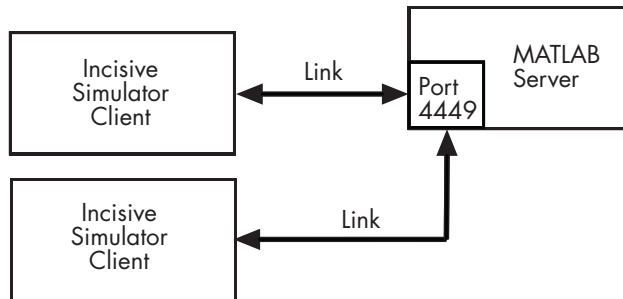
In this scenario, a MATLAB server function waits for service requests that it receives from an Incisive simulation session. After receiving a request, the server establishes a communication link and invokes a specified MATLAB function wrapper that computes data for, verifies, or visualizes the HDL model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

Note You cannot initiate Link for Cadence Incisive communication between MATLAB and the HDL simulator from MATLAB. The MATLAB server simply responds to function call requests that it receives from the HDL simulator.

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

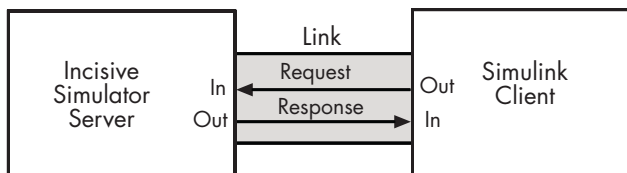


The MATLAB server can service multiple simultaneous HDL simulator sessions and HDL models. However, you should follow recommended guidelines to ensure the server can track the I/O associated with each model and session. The following figure shows a multiple-client scenario connecting to the server at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



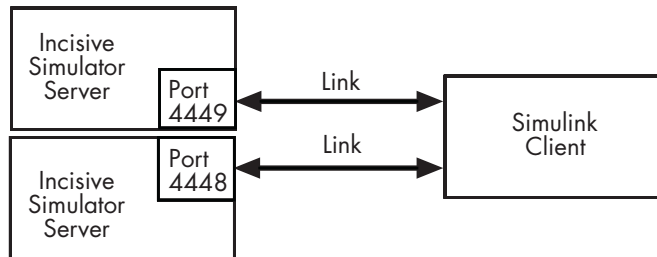
Simulink and HDL Simulator Links

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



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

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



Modes of Communication

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

TCP/IP socket mode is more versatile. You can use it for single-system and network configurations. This option offers the greatest scalability.

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

Working with MATLAB and the HDL Simulator

When linked with MATLAB, the HDL simulator functions as the client, initiating requests of MATLAB that focus on numerical computing, algorithm development, and visualization. The MATLAB server, which you start with a supplied MATLAB function, waits for connection requests from instances of the HDL simulator running on the same or different computers. When the server receives a request, it executes a specified wrapper MATLAB function you have coded to perform tasks on behalf of a component 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.

After the server is running, you can start and configure the HDL simulator for use with MATLAB with a supplied Link for Cadence Incisive function. Optional parameters allow you to specify

- Tool Command Language (Tcl) commands that execute as part of startup
- A specific HDL simulator executable to start
- The name of an HDL simulator Tcl script file to store the complete startup script for future use or reference

For more on configuring the HDL simulator for use with the Link for Cadence Incisive software, see “Setting Up the HDL Simulator for Use with the Link for Cadence Incisive Software” on page 1-23.

When you initiate a specific MATLAB link session, you specify parameters that identify

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

The MATLAB server can service multiple simultaneous HDL simulator designs and clients. For more about initiating MATLAB link sessions, see Chapter 3, “Starting and Controlling MATLAB Link Sessions”.

Working with Simulink and the HDL Simulator

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

Using the Block Parameters dialog box for an 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.

Using the Link for Cadence Incisive MATLAB function `nclaunch`, you can start and configure the HDL simulator with optional parameters that allow you to specify the same behavior as when you configure the simulator for MATLAB (see “Working with MATLAB and the HDL Simulator” on page 1-10). In addition, you can specify the default mode of communication to be used for the link and, if appropriate, a TCP/IP socket port. For more on configuring the HDL simulator for use with the Link for Cadence Incisive software, see “Setting Up the HDL Simulator for Use with the Link for Cadence Incisive Software” on page 1-23.

Link for Cadence Incisive software equips the HDL simulator with a set of Link for Cadence Incisive command extensions. Using one of those commands, you execute the HDL simulator with an instance of an HDL model for cosimulation with Simulink. After the model is loaded, you can start the cosimulation session from Simulink.

Link for Cadence Incisive software also includes a block for generating value change dump (VCD) files. You can use VCD files generated with this block

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

Installation and Setup

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

- “What Are Your Environment Requirements?” on page 1-13
- “Deciding on a Configuration” on page 1-15
- “Identifying a Server in a Network Configuration” on page 1-17
- “Choosing TCP/IP Socket Ports” on page 1-18
- “Checking Product Requirements” on page 1-20
- “Installing Related Application Software” on page 1-22
- “Installing the Link for Cadence Incisive Software” on page 1-23
- “Setting Up the HDL Simulator for Use with the Link for Cadence Incisive Software” on page 1-23

What Are Your Environment Requirements?

As part of the installation and setup process, review the following checklist to identify environment requirements that pertain to your Link for Cadence Incisive application. Questions to ask yourself about configuration requirements are in the first column of the table; go to the topic listed in the second column for information on how to address the requirement.

Environment Requirements Checklist

Requirement	For More Information, See...
Configurations	
<input type="checkbox"/> Will your application use multiple communication links?	“Deciding on a Configuration” on page 1-15
<input type="checkbox"/> How many instances of the MATLAB server are required?	“Deciding on a Configuration” on page 1-15

Environment Requirements Checklist (Continued)

Requirement	For More Information, See...
<input type="checkbox"/> Will a MATLAB server be handling multiple HDL simulator client connections? If so, how many? Will they be from the same or different HDL simulator sessions?	“Deciding on a Configuration” on page 1-15
<input type="checkbox"/> How many MATLAB functions do you need to write to model your HDL implementation?	“Deciding on a Configuration” on page 1-15
<input type="checkbox"/> If your application will be using Simulink, how many cosimulation blocks are needed? Will the blocks be connecting to the same or different HDL simulator sessions?	“Deciding on a Configuration” on page 1-15
<input type="checkbox"/> To how many HDL simulator sessions will your Simulink model connect?	“Deciding on a Configuration” on page 1-15
<p>Mode of Communication</p>	
<input type="checkbox"/> Is performance the highest priority for your application? If so, can you run MATLAB and Simulink and the HDL simulator on the same computer system?	“Modes of Communication” on page 1-9
<input type="checkbox"/> Does your application require only one communication link (channel) on a single computing system?	“Modes of Communication” on page 1-9
<input type="checkbox"/> Is configuration flexibility a high priority for your application? Does the application have growth potential?	“Modes of Communication” on page 1-9
<input type="checkbox"/> Do you prefer to use the TCP/IP socket mode of communication for a single-computer configuration? If so, do you want the Link for Cadence Incisive software to identify an available socket port on the system or do you want to choose a socket port yourself?	“Choosing TCP/IP Socket Ports” on page 1-18
<p>Network Configurations</p>	
<input type="checkbox"/> Have you identified the computer systems that will function as Link for Cadence Incisive servers?	“Identifying a Server in a Network Configuration” on page 1-17
<input type="checkbox"/> What is the Internet address or host name of each computer system that will function as a server?	“Identifying a Server in a Network Configuration” on page 1-17

Environment Requirements Checklist (Continued)

Requirement	For More Information, See...
<input type="checkbox"/> Do you want the Link for Cadence Incisive software to identify an available TCP/IP socket port on server systems for establishing communication links? Instead, do you want to choose or identify TCP/IP socket ports yourself?	“Choosing TCP/IP Socket Ports” on page 1-18
Related Software	
<input type="checkbox"/> Is the HDL simulator installed on all systems as needed for your application?	“Installing Related Application Software” on page 1-22
<input type="checkbox"/> Is MATLAB installed on all systems as needed for your application? (See also HDL Simulator Setup, later in this table.)	“Installing Related Application Software” on page 1-22
<input type="checkbox"/> Does your application require the use of any toolboxes? If so, are the toolboxes installed on all systems as needed for your application?	“Installing Related Application Software” on page 1-22
<input type="checkbox"/> Will you be using the Simulink component of the Link for Cadence Incisive software? If so, are Simulink and Simulink Fixed Point installed on all systems as needed for your application? Are the required blocksets installed?	“Installing Related Application Software” on page 1-22
HDL Simulator Setup	
<input type="checkbox"/> Will you be running the HDL simulator on a machine that does not have MATLAB installed?	“Setting Up the HDL Simulator for Use with the Link for Cadence Incisive Software” on page 1-23

Deciding on a Configuration

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

- Shared memory communication is an option for configurations that require only one communication link on a single computing system.

- TCP/IP socket communication is required for configurations that use multiple communication links on one or more computing systems. Unique TCP/IP socket ports distinguish the communication links.
- In any configuration, an instance of MATLAB can run only one instance of the Link for Cadence Incisive MATLAB server (`hdldaemon`) at a time.
- In a TCP/IP configuration, the MATLAB server can handle multiple client connections to one or more HDL simulator sessions.
- HDL Cosimulation blocks in a Simulink model can connect to the same or different HDL simulator sessions.

The following lists provide samples of valid configurations for using Incisive simulators with MATLAB and Simulink, respectively. The scenarios apply whether the HDL simulator is running on the same or different computing system as MATLAB or Simulink. In a network configuration, you use an Internet address in addition to a TCP/IP socket port to identify the servers in an application environment.

MATLAB

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

- An HDL simulator session linked to a MATLAB function `foo` through a single instance of the MATLAB server
- An HDL simulator session linked to multiple MATLAB functions (for example, `foo` and `bar`) through a single instance of the MATLAB server
- An HDL simulator session linked to a MATLAB function `foo` through multiple instances of the MATLAB server (each running within the scope of a unique MATLAB session)
- Multiple HDL simulator sessions each linked to a MATLAB function `foo` through multiple instances of the MATLAB server (each running within the scope of a unique MATLAB session)
- Multiple HDL simulator sessions each linked to a different MATLAB function (for example, `foo` and `bar`) through the same instance of the MATLAB server

- Multiple HDL simulator sessions each linked to MATLAB function foo through a single instance of the MATLAB server

Note Although multiple HDL simulator sessions can link to the same MATLAB function in the same instance of the MATLAB server, as the last configuration scenario suggests, such links are not recommended. If the MATLAB function maintains state (for example, maintains global or persistent variables), you may experience unexpected results because the MATLAB function does not distinguish between callers when handling input and output data. If you must apply this configuration scenario, consider deriving unique instances of the MATLAB function to handle requests for each HDL model.

Simulink

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

- An HDL Cosimulation block in a Simulink model linked to a single HDL simulator session
- Multiple HDL Cosimulation blocks in a Simulink model linked to the same HDL simulator session
- An HDL Cosimulation block in a Simulink model linked to multiple HDL simulator sessions
- Multiple HDL Cosimulation blocks in a Simulink model linked to different HDL simulator sessions

Identifying a Server in a Network Configuration

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

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

As part of your application setup, you must identify

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

For guidelines on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page 1-18.

Choosing TCP/IP Socket Ports

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

A TCP/IP socket port number (or service name or alias, see “TCP/IP Services” on page 1-20) 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 Cadence Incisive 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.
- Choose a port number that does not contain patterns or have a known meaning. That is, avoid port numbers that more likely to be used by others because they are easier to remember.

- Do not use ports 1 to 1023. These ports are reserved for use by the Internet Assigned Numbers Authority (IANA).
- Avoid using ports 49152 through 65535. These are dynamic ports that operating systems use randomly. If you choose one of these ports, you risk a potential port conflict.
- On the Windows platform, do not choose a filtered TCP/IP port. The Windows TCP/IP port filtering mechanism allows disabling access to selected ports for security purposes. TCP/IP port filtering on either the client or server side can cause the Link for Cadence Incisive interface to fail to make a connection.

In such cases the error messages displayed by the Link for Cadence Incisive software indicate the lack of a connection, but do not explicitly indicate the cause.

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

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

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

- a** From the Windows **Start** menu, select **Settings > Network Connections**.
- b** Select **Local Area Connection** from the **Network and Dialup Connections** window.
- c** From the **Local Area Connection** dialog, select **Properties > Internet Protocol (TCP/IP)**. From there, select **Properties > Advanced > Options**. Finally, select **TCP/IP filtering > Properties**.
- d** If your port is listed in the **TCP/IP filtering>Properties** dialog, you should select an unfiltered port. The easiest way to do this is to specify 0 for the socket port number to let the Link for Cadence Incisive software choose an available port number for you.

Note The socket port resource is associated with the server component of a Link for Cadence Incisive 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 the HDL simulator.

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.

Checking Product Requirements

Link for Cadence Incisive software requires the following:

Platform

For the specific platforms supported with the current release of the Link for Cadence Incisive software, visit the MathWorks Link for Cadence Incisive product requirements page.

Application software

Incisive HDL Simulator, Incisive Design Team Simulator, or Incisive Enterprise Specman Simulator. Visit the MathWorks Link for Cadence Incisive product requirements page for specific versions supported with the current release of the Link for Cadence Incisive software.

MATLAB

Additional application software required for cosimulation with Simulink

Simulink

Simulink Fixed Point

Fixed Point Toolbox

Optional application software

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

Note Many Link for Cadence Incisive demos require one or more of the above.

Platform-specific software

The Link for Cadence Incisive shared libraries (`liblfihdls*.so`, `liblfihdlc*.so`) are built using the gcc included in the Incisive simulator platform distribution. If you are linking your own applications into the HDL simulator, the recommendation is that you also build against this gcc. See the HDL simulator documentation for more details about how to build and link your own applications.

Installing Related Application Software

Based on your configuration decisions and the software required for your Link for Cadence Incisive 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 Cadence Incisive MATLAB server, you need to install MATLAB and any applicable toolbox software on multiple systems. Each instance of MATLAB can run only one instance of the server.

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

Installing the Link for Cadence Incisive Software

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

For details on how to install Link for Cadence Incisive software, see the MATLAB installation instructions.

Setting Up the HDL Simulator for Use with the Link for Cadence Incisive Software

You can choose to have the HDL simulator run on the same machine as MATLAB or on a separate machine.

- If you choose the same machine, then you must run `nclaunch` from the MATLAB prompt at least once. This command creates a Tcl script that sets up Link for Cadence Incisive commands for use with Incisive simulators. See “Setting Up Link for Cadence Incisive Software for Use with the Incisive Simulator on the Same Machine as MATLAB” on page 1-23.
- If you choose to use a different machine, follow the instructions in “Setting Up Link for Cadence Incisive Software for Use with the Incisive Simulator on a Separate Machine from MATLAB” on page 1-24.

Setting Up Link for Cadence Incisive Software for Use with the Incisive Simulator on the Same Machine as MATLAB

After all the required software is installed, set up the Incisive simulator so that it is always ready for use with MATLAB and Simulink and so that you can invoke the HDL simulator outside of MATLAB by creating a specialized Tcl startup script. The first time you want to connect MATLAB or Simulink and an Incisive simulator through the Link for Cadence Incisive software, use the `nclaunch` command with the following arguments.

```
nclaunch ('tclstart', 'puts "Initializing Link for Cadence Incisive software",  
          'startupfile','lfiinit','starthdlsim','no')
```

Where `lfiinit` is the name you choose for the Tcl startup script. The property name/value pair `'starthldsimsim'` and `'no'` indicate to the `nclaunch` function *not* to start the HDL simulator when this line is executed.

After the Tcl script has been created, you can launch the Incisive simulator from outside of MATLAB and still have access to Link for Cadence Incisive commands by typing:

```
%tclsh
source tclscript
hdlsimmatlab arguments
```

Where `tclscript` is the name of the script created with `nclaunch` (`lfiinit` in this example). `hdlsimulink` can also be used in place of `hdlsimmatlab`.

Setting Up Link for Cadence Incisive Software for Use with the Incisive Simulator on a Separate Machine from MATLAB

If you are running the Incisive simulator on a machine that does not have MATLAB or if you are interested in setting up your own scripting for the building and running of the Incisive simulator, you must provide the Incisive simulator with the libraries and configuration information it needs to communicate with MATLAB.

Every time you start the Incisive simulator, and want it to communicate with MATLAB, you must run `ncsim` with the appropriate arguments, as shown in the following procedure.

Note This setup is supported for the platform configurations as described in the MathWorks Link for Cadence Incisive product requirements page.

Copying Libraries and Creating Simulation Requirements.

- 1 On the machine with MATLAB, go to the root directory for the Link for Cadence Incisive software:

```
MATLABROOT/toolbox/incisive/arch/
```

Where `arch` is the system type of the platform running the HDL simulator: `linux32`, `linux64`, or `solaris64`.

Note If you are running `ncsim` in 32-bit mode on a 64-bit Linux platform, copy the libraries from `linux86`.

- 2 Copy all the shared libraries from this directory into the desired destination directory on the machine running the Incisive simulator.
- 3 Create a text file that includes the following lines:

```
proc nomatlabtb {args} {call nomatlabtb $args}
proc matlabtb {args} {call matlabtb $args}
proc matlabcp {args} {call matlabcp $args}
proc matlabtbeval {args} {call matlabtbeval $args}
```

You may give the text file any valid file name.

- 4 Update your scripts, makefiles, or other means of invoking the simulator to include the following arguments to `ncsim`, where `IUS_VERSION` is the release number of your Incisive simulator installation (e.g., 05.70), *yourpath* is the link for Cadence Incisive root directory in the first step, and *filename* is the name of the text file you created in step 3:
 - a For the link to MATLAB (`matlabcp`, `matlabtb`):

```
-loadcfc /yourpath/liblfihd1c_IUS_VERSION:matlabclient -input filename
```

- b For the link to Simulink:

```
-loadvpi /yourpath/liblfihd1s_IUS_VERSION:simlinkserver
+socket=socketNumber
```

Note If *yourpath* is `pwd`, reference it as `./liblfihds`.

Note The Link for Cadence Incisive shared libraries were built against the GCC libraries included with the Incisive simulator platform distribution. It is required that your LD_LIBRARY_PATH specify the location of these libraries as explained in the Cadence documentation.

Here is an example for properly setting up the linux64 architecture in a csh:

```
% setenv LD_LIBRARY_PATH install_dir/tools/lib/64bit:\
install_dir/tools/systemc/gcc/64bit/install/lib64
```

Getting Help with the Link for Cadence Incisive Software

The following sections explain how to get help with using the Link for Cadence Incisive software:

- “Documentation Overview” on page 1-27
- “Online Help” on page 1-28
- “Demos and Tutorials” on page 1-28

Documentation Overview

The following documentation is available with this product.

Title	Description
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.
Coding a Link for Cadence Incisive MATLAB Application	Explains how to code HDL models and MATLAB functions for Link for Cadence Incisive MATLAB applications. Provides details on how the Link for Cadence Incisive interface maps HDL data types to MATLAB data types and vice versa.
Starting and Controlling MATLAB Link Sessions	Explains how to start and control the HDL simulator and MATLAB test bench and component sessions.
Modeling and Verifying an HDL Design with Simulink	Explains how to use the HDL simulator and Simulink for cosimulation modeling.
MATLAB Functions — Alphabetical List	Describes Link for Cadence Incisive functions for use with MATLAB.

Title	Description
HDL Simulator Tcl Commands — Alphabetical List	Describes Link for Cadence Incisive Tcl commands for use with the HDL simulator.
Simulink Blocks — Alphabetical List	Describes Link for Cadence Incisive blocks for use with Simulink.

Online Help

The following online help is available:

- Online help in the MATLAB Help browser. Click the Link for Cadence Incisive product link in the browser's Contents.
- M-help for Link for Cadence Incisive MATLAB functions. This help is accessible with the MATLAB `help` command. For example, enter the command line `help nclaunch`.
- Block reference pages accessible through the Simulink interface.

Demos and Tutorials

The Link for Cadence Incisive product 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. Tutorials provide procedural instruction on how to apply the product.

To see a list of Link for Cadence Incisive demos and tutorials that you can run, type the following at a MATLAB command prompt:

```
>> demos
```

Select **Toolboxes > Link for Cadence Incisive** from the navigational pane.

Coding a Link for Cadence Incisive MATLAB Application

Overview (p. 2-2)

Provides an overview of MATLAB test bench and component functions, and of the steps involved in coding these functions for use with Link for Cadence Incisive software.

Coding Entities or Modules for MATLAB Verification (p. 2-3)

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

Compiling the HDL Model (p. 2-9)

Explains how to compile an HDL design.

Coding a MATLAB Test Bench Function (p. 2-10)

Explains how to code a MATLAB function to verify or visualize an HDL design.

Coding a MATLAB Component Function (p. 2-33)

Explains how to code a MATLAB component function.

Placing a MATLAB Test Bench or Component Function on the MATLAB Search Path (p. 2-35)

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

Overview

Link for Cadence Incisive software supports two types of MATLAB functions that interface to HDL models:

- *Test bench functions* are functions that let you verify the performance of the HDL model, or of components within the model. A test bench function drives values onto signals connected to input ports of an HDL design under test, and receives signal values from the output ports of the module.
- *MATLAB component functions* are functions that simulate the behavior of components in the HDL model. A stub module (providing port definitions only) in the HDL model passes its input signals to the MATLAB component function. The MATLAB component processes this data and returns the results to the outputs of the stub module. A MATLAB component typically provides some functionality (such as a filter) that is not yet implemented in the HDL code.

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

This section provides an overview of the steps required to develop an HDL model for use with MATLAB and Link for Cadence Incisive software. To program the HDL component of a Link for Cadence Incisive application, you must perform the following tasks:

- 1 Code the HDL model for MATLAB verification “Coding Entities or Modules for MATLAB Verification” on page 2-3.
- 2 Compile the HDL model.
- 3 Code the required MATLAB test bench or MATLAB component functions.
- 4 Place the MATLAB functions on the MATLAB search path.

Coding Entities or Modules for MATLAB Verification

The most basic element of communication in the Link for Cadence Incisive interface is the VHDL entity or Verilog module. The interface passes all data between the HDL simulator and MATLAB as port data. Link for Cadence Incisive software works with any existing VHDL entity or Verilog module. However, when coding a VHDL entity or Verilog 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:

- “Overview of Steps for Coding Entities or Modules” on page 2-3
- “Choosing an Entity or Module Name” on page 2-4
- “Specifying Port/Signal and Module Paths” on page 2-4
- “Specifying Ports for the Entity or Module” on page 2-6
- “Specifying Port Direction Modes” on page 2-6
- “Specifying Port Data Types” on page 2-6
- “Sample VHDL Entity Definition” on page 2-8

Note When using a mixed-language HDL model (one that contains both VHDL and Verilog components), MATLAB can access signals only with the language of the top-level module instance or component.

Overview of Steps for Coding Entities or Modules

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

- 1** Consider choosing an entity or module name that can be used as a valid MATLAB function name.
- 2** Determine the number of ports required and name them.
- 3** Specify a direction mode for each port.

4 For each port, specify a VHDL or Verilog data type that is supported by Link for Cadence Incisive software.

5 Compile the model.

The following sections provide more detail on the preceding steps.

Choosing an Entity or Module Name

Although not required, when naming the VHDL entity or Verilog 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 Cadence Incisive software assumes that a VHDL entity or Verilog module and its simulation function share the same name.

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

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

Specifying Port/Signal and Module Paths

These rules are for signal/port and module path specifications for MATLAB. Other specifications may work but are not guaranteed to work in this or future releases. For Simulink path specifications, see “Full HDL Name” in the “Ports Pane” on page 7-6 section of the HDL Cosimulation block reference.

Path specifications in MATLAB:

- If the top level is Verilog:
 - 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.sub1.sub2
```

The following are invalid signal and module path specification examples:

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

- If the top level is VHDL:
 - Path specification can include the top-level module or begin with ":" to represent the top-level module.
 - 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
:sub:port_or_sig
top
top.sub1.sub2
:
:sub
```

The following are invalid signal and module path specification examples:

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

Specifying Ports for the Entity or Module

Determine the number of ports required for the entity or module to be simulated and tested. Name them within the port list for the entity or module.

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 models. For details on how Link for Cadence Incisive software converts data types for the MATLAB environment, see “Data Type Conversions” on page 2-11.

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

Port Data Types for VHDL Entities

In your entity statement, you must define each port, which you plan to test with MATLAB, with a VHDL data type that is supported by the Link for

Cadence Incisive interface. The interface can convert scalar and composite data of the following VHDL types to comparable MATLAB types:

- STD_LOGIC, STD_ULOGIC, BIT, STD_LOGIC_VECTOR, STD_ULOGIC_VECTOR, BIT_VECTOR, and BOOLEAN
- 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 Cadence Incisive software 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 the Link for Cadence Incisive software. The interface can convert data of the following Verilog port types to comparable MATLAB types:

- reg
- integer
- wire

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

Sample VHDL Entity Definition

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

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

Both input ports are defined as vectors consisting of five standard logic values. The output port `adj` is also defined as a standard logic vector, but consists of only two values. The output ports `dvalid` and `odata` are defined as scalar standard logic ports. For information on how the Link for Cadence Incisive software converts data of standard logic scalar and composite types for use in the MATLAB environment, see “Data Type Conversions” on page 2-11.

```
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 the HDL Model

After you create or edit your HDL design source files, use the HDL simulator tools to compile and elaborate the code. The Incisive simulator allows for 1-step and 3-step processes for HDL compilation, elaboration, and simulation.

The following Incisive simulator command compiles the VHDL file `modsimrand.vhd`:

```
sh> ncvhdl modsimrand.vhd
```

The following Incisive simulator command compiles and elaborates the Verilog design `test.v`, and then loads it for simulation, in a single step:

```
sh> ncverilog +gui +access+rwc +linedebug test.v
```

The following sequence of Incisive simulator commands performs all the same processes in multiple steps:

```
sh> ncvlog linedebug test.v
sh> ncelab access +rwc test
sh> ncsim test
```

Note You should provide read/write access to the signals that are connecting to the MATLAB session for cosimulation. The previous example demonstrates how to provide read/write access to all signals in your design. For higher performance, you want to provide access only to those signals used in cosimulation. See the description of the `+access` flag to `ncverilog` and the `-access` argument to `ncelab` for details.

See the Incisive simulator documentation for complete details on compiling and elaborating your HDL designs. For more examples, see [Link for Cadence Incisive demos and tutorials](#).

Coding a MATLAB Test Bench Function

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 the HDL simulator. The following sections cover these topics:

- “Overview of the Steps for Coding a MATLAB Test Bench Function” on page 2-10
- “Data Type Conversions” on page 2-11
- “Naming a MATLAB Test Bench Function” on page 2-15
- “Passing Parameters to and from the MATLAB Function” on page 2-16
- “Gaining Access to and Applying Port Information” on page 2-17
- “Converting Data for Manipulation” on page 2-20
- “Converting Data for Return to the HDL Simulator” on page 2-21
- “Sample MATLAB Test Bench Function” on page 2-26

Overview of the Steps for Coding a MATLAB Test Bench Function

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

- 1 Understand how Link for Cadence Incisive software converts HDL model data for use in the MATLAB environment.
- 2 Name the MATLAB test function. Consider naming it with the name of the HDL model the function is to test.
- 3 Define expected parameters in the function definition line.
- 4 Determine the types of port data being passed into the function.
- 5 Extract and, if appropriate for the simulation, apply information received in the `portinfo` structure.
- 6 Convert data for manipulation in the MATLAB environment, as necessary.

7 Convert data that needs to be returned to the HDL simulator.

Data Type Conversions

This section describes data type conversions that Link for Cadence Incisive software performs in order to transmit and receive data between HDL models and the MATLAB environment.

VHDL Data Type Conversions

Link for Cadence Incisive software converts VHDL entity data to types that apply in the MATLAB environment. To program a MATLAB function for a VHDL model, you must understand the type conversions required by your application. You may also need to handle differences between the array indexing conventions employed by VHDL and MATLAB.

The data types of arguments passed in to the function determine

- The types of conversions required before and after data is manipulated
- The types of conversions required to return data to the Incisive simulator

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

VHDL-to-MATLAB Data Type Conversions

VHDL Types...	As Scalar Converts to...	As Composite Converts to...
STD_LOGIC, STD_ULOGIC, and BIT	A character that matches the character literal for the desired logic state.	
STD_LOGIC_VECTOR, STD_ULOGIC_VECTOR, BIT_VECTOR, SIGNED, and UNSIGNED		A column vector of characters (as defined in VHDL Conversions for the Incisive Simulator on page 2-22) with one bit per character.

VHDL-to-MATLAB Data Type Conversions (Continued)

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

VHDL-to-MATLAB Data Type Conversions (Continued)

VHDL Types...	As Scalar Converts to...	As Composite Converts to...
TIME	Type double for time values in seconds and type int64 for values representing simulator time increments (see the description of the 'time' option in “Starting the MATLAB Server” on page 3-7).	Arrays of type double or int64 with a size that is equivalent to the VHDL port size.
Enumerated types	Character array (string) that contains the MATLAB representation of a VHDL label or character literal. For example, the label high converts to 'high' and the character literal 'c' converts to ''c''.	Cell array of strings with each element equal to a label for the defined enumerated type. Each element is the MATLAB representation of a VHDL label or character literal. For example, the vector (one, '2', three) converts to the column vector ['one'; ''2''; 'three']. A user-defined enumerated type that contains only character literals, converts to a vector or array of characters as indicated for the types STD_LOGIC_VECTOR, STD_ULONGIC_VECTOR, BIT_VECTOR, SIGNED, and UNSIGNED.

Array Indexing Differences Between MATLAB and VHDL. MATLAB indexes array elements by using a column-major numbering scheme, starting with column 1. That is, MATLAB internally stores data elements from the first column first, the second column second, and so on through the last column. This reverses the order of indexes between MATLAB and VHDL. For example, the following VHDL program declares the port `sta` as an array of two 8-bit bytes.

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

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

USE WORK.myporttype.all;

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

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

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

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

Note also that VHDL arrays indices are commonly zero-based. That is, they are defined as (0 to n) or (n DOWNT0 0). In such cases, an offset of 1 is applied because MATLAB array indexing always begins at 1.

Verilog Data Type Conversions

Link for Cadence Incisive software 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 the Incisive simulator

The following table summarizes how Link for Cadence Incisive software converts supported Verilog data types to MATLAB types. Only scalar data types are supported for Verilog.

Verilog-to-MATLAB Data Type Conversions

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

Array Indexing Differences Between MATLAB and Verilog. MATLAB indexes array elements by using a column-major numbering scheme, starting with column 1. Thus, MATLAB internally stores data elements from the first column first, the second column second, and so on through the last column. This storage alignment reverses the order of indexes between MATLAB and Verilog.

Naming a MATLAB Test Bench Function

You can name and specify a MATLAB test bench function however you like, so long as you adhere to MATLAB function and file naming guidelines. By default, the Link for Cadence Incisive interface assumes the name for a MATLAB function matches the name of the HDL model that the function verifies or visualizes.

For example, if you name the VHDL entity `mystdlogic`, Link for Cadence Incisive software assumes the corresponding MATLAB function is `mystdlogic` and resides in the file `mystdlogic.m`.

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

Passing Parameters to and from the MATLAB Function

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

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

The data passed into the function through the output parameters is defined by the structure of the corresponding HDL model. The function parameters are

- `iport` — Structure that drives (by deposit) values onto signals connected to ports of the associated HDL model.
- `tnext` (optional) — Specifies time at which the MATLAB callback function is executed. This parameter should be initialized to an empty value (`[]`). If it is not subsequently updated, no new entries are added to the simulation schedule. By default, time is represented in seconds. The interface accepts 64-bit integers, which are interpreted as multiples of the HDL simulator resolution limit.
- `oport` — Structure that receives signal values from the output ports defined for the associated HDL model at the time specified by `tnow`.
- `tnow` — Receives the simulation time at which the MATLAB function is called. By default, time is represented in seconds. The interface also supports full 64-bit time resolution. For more information see “Starting the MATLAB Server” on page 3-7.
- `portinfo` — For the first call to the function (at the start of the simulation) only, receives a structure whose fields describe the ports defined for the associated HDL model. For each port, the `portinfo` structure passes information such as the port’s type, direction, and size. The information passed to this parameter is useful for validating the module under test. You can use the port information to create a generic MATLAB function that operates differently depending on the port information supplied at startup.

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

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

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

For more information on using `tnext` and `tnow` for simulation scheduling, see “Deciding on MATLAB Link Session Scheduling Options” on page 3-11 and “Controlling Callback Timing from a MATLAB Test Bench or Component Function” on page 3-12. For more information on port data, see “Gaining Access to and Applying Port Information” on page 2-17.

Gaining Access to and Applying Port Information

The Link for Cadence Incisive interface passes information about the HDL design 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 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 HDL model.

```
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 Indicates...	And Applies to...
<i>field1</i>	in	The port is an input port	All port types
	out	The port is an output port	All port types
	inout	The port is a bidirectional port	All port types
	tscale	The simulator resolution limit in seconds as specified in the HDL simulator	All types
<i>field2</i>	<i>portname</i>	The name of the port	All port types

HDL Port Information (Continued)

Field...	Can Contain...	Which Indicates...	And Applies to...
<i>field3</i>	type	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, do the following:

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

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

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

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

Converting Data for Manipulation

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

Required Data Conversions

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

Required Data Conversions (Continued)

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

Converting Data for Return to the HDL Simulator

If your simulation MATLAB function needs to return data to the HDL simulator, it may be necessary for you to first convert the data to a type

supported by Link for Cadence Incisive software. The following tables list circumstances requiring such conversions for VHDL and Verilog.

VHDL Conversions for the Incisive Simulator

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: <pre>iport.s1 = 'X'; %STD_LOGIC iport.bit = '1'; %BIT</pre>
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: <pre>iport.s1v = 'X10ZZ'; %STD_LOGIC_VECTOR iport.bitv = '10100'; %BIT_VECTOR iport.uns = dec2bin(10,8); %UNSIGNED, 8 bits</pre>

VHDL Conversions for the Incisive Simulator (Continued)

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

VHDL Conversions for the Incisive Simulator (Continued)

To Return Data to an IN Port of Type...	Then...
REAL	<p>Declare the data as an array of type double with a size that is equivalent to the VHDL port size. For example:</p> <pre data-bbox="733 526 1055 552">iport.dbl = ones(2,2);</pre>
TIME	<p>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 hd1daemon 'time' option (see "Starting the MATLAB Server" on page 3-7), which applies to IN ports only. Declare an array of TIME values by using a MATLAB array of identical size and shape. All elements of a given port are restricted to time in seconds (type double) or simulator increments (type int64), but otherwise you can mix the formats. For example:</p> <pre data-bbox="733 1017 1322 1107">iport.t1 = int64(1:10)'; %Simulator time %increments iport.t2 = 1e-9; %1 nsec</pre>

VHDL Conversions for the Incisive Simulator (Continued)

To Return Data to an IN Port of Type...	Then...
Enumerated types	<p>Declare the data as a string for scalar ports or a cell array of strings for array ports with each element equal to a label for the defined enumerated type. The 'label' field of the portinfo structure lists all valid labels (see “Gaining Access to and Applying Port Information” on page 2-17). 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.</p> <pre data-bbox="731 753 1228 840"> iport.char = {'A', 'B'}; %Character %literal iport.undef = 'mylabel'; %User-defined label </pre>
Character array for standard logic or bit representation	<p>Use the dec2bin function to convert the integer. For example:</p> <pre data-bbox="731 986 1199 1012"> oport.slva =dec2bin([23 99],8)'; </pre> <p>This example converts two integers to a 2-element array of standard logic vectors consisting of 8 bits.</p>

Verilog Conversions for the Incisive Simulator

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: <code>iport.bit = '1';</code>
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 Cadence Incisive software. 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/incisive/incisivedemos/manchester/decoder.vhd`
- `matlabroot/toolbox/incisive/incisivedemos/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 definition on the first line represents the communication channel between MATLAB and the Incisive simulator. The following code is the function definition of the `manchester_decoder` MATLAB function.

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

The function definition

- Names the function. This definition names the function `manchester_decoder`, which differs from the entity name `decoder`. Because the names differ, the function name must be specified explicitly later when the entity is initialized for verification with the `matlabtb` or `matlabtbeval` HDL simulator Tcl command.
- Defines required input and output parameters. A MATLAB test bench function *must* include two input parameters, `iport` and `tnext`, and three output parameters, `oport`, `tnow`, and `portinfo`, and *must* appear in the order shown.

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

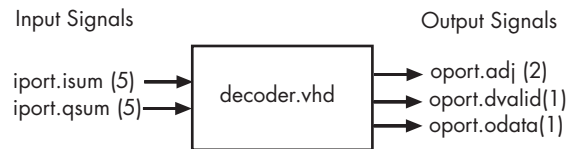
```
tnext = [];
iport = struct();
```

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

<code>iport</code>	Forces (by deposit) a value onto the signal connected to the entity's input ports, <code>isum</code> and <code>qsum</code> .
<code>tnext</code>	Specifies a time value that indicates when the Incisive simulator is to call back the MATLAB function.
<code>oport</code>	Receives VHDL signal values from the entity's output ports, <code>adj</code> , <code>dvalid</code> , and <code>odata</code> .

- `tnow` Receives the simulation time at which the Incisive simulator calls the MATLAB function.
- `portinfo` For the first call to the function, receives a structure that describes the ports defined for the entity.

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



For more information on the required MATLAB function parameters, see "Passing Parameters to and from the MATLAB Function" on page 2-16.

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

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

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

VHDL Example Port Definitions

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

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 “Data Type Conversions” on page 2-11.

3 Set up any required timing parameters.

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

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

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

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

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

The two calls to `bin2dec` convert the binary data that the MATLAB function receives from the entity's output ports, `adj`, `dvalid`, and `odata` to unsigned decimal values that MATLAB can compute. The function converts the 2-bit transposed vector `oport.adj` to a decimal value in the range 0 to 4 and `oport.dvalid` and `oport.odata` to the decimal value 0 or 1.

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

5 Convert data to be returned to the Incisive simulator.

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

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

The three calls to `dec2bin` convert the decimal values computed by MATLAB to binary data that the MATLAB function can deposit to the entity's input ports, `isum` and `qsum`. In each case, the function converts a decimal value to 5-element bit vector with each bit representing a character that maps to a character literal representing a logic state.

“Converting Data for Return to the HDL Simulator” on page 2-21 provides a summary of the types of data conversions to consider when returning data to the Incisive simulator.

Coding a MATLAB Component Function

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

Function Definition and Parameters

The syntax of a MATLAB component function is

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

The function returns the following outputs:

- **oport** — Structure that drives (by deposit) values onto signals connected to output ports of the associated HDL design.
- **tnext** (optional) — Specifies the time at which the HDL simulator schedules the next callback to MATLAB. **tnext** should be initialized to an empty value (`[]`). If **tnext** is not subsequently updated, no new entries are added to the simulation schedule. In that case, callback scheduling is controlled by the `matlabcp` command.

For more information see “Controlling Callback Timing from a MATLAB Test Bench or Component Function” on page 3-12.

It is strongly recommended that you initialize the function outputs to empty values at the beginning of the function as in the following example:

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

The following parameters are passed to the function:

- **iport** — Structure that receives signal values from the input ports defined for the associated HDL design at the time specified by **tnow**.
- **tnow** — Receives the simulation time at which the MATLAB function is called. By default, time is represented in seconds. For more information see “Controlling Callback Timing from a MATLAB Test Bench or Component Function” on page 3-12.

- `portinfo` — For the first call to the function only (at the start of the simulation), `portinfo` receives a structure whose fields describe the ports defined for the associated HDL design. 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.

For more information on using `tnext` and `tnow` for simulation scheduling, see “Deciding on MATLAB Link Session Scheduling Options” on page 3-11.

Note 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. Thus, the MATLAB component function returns signal data to the *outputs*, and receives data from the *inputs*, of the associated HDL design.

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

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

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

```
>> which MyVerilogFunction  
/work/incisive/MySym/MyVerilogFunction.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 and Controlling MATLAB Link Sessions

Overview (p. 3-3)	Provides an overview of the steps for starting and controlling a MATLAB link session.
Checking the MATLAB Server's Link Status (p. 3-5)	Explains how to check the status of the MATLAB server.
Starting the MATLAB Server (p. 3-7)	Explains how to start the MATLAB server.
Starting the HDL Simulator for Use with MATLAB (p. 3-10)	Explains how to start the HDL simulator for use with MATLAB.
Deciding on MATLAB Link Session Scheduling Options (p. 3-11)	Describes different ways of scheduling the invocations of a MATLAB test bench or component function.
Controlling Callback Timing from a MATLAB Test Bench or Component Function (p. 3-12)	Explains how to control callback timing from a MATLAB test bench or component function.
Initializing the HDL Simulator for a MATLAB Link Session (p. 3-13)	Explains how to initialize the HDL simulator for use with MATLAB as a link session tool.
Applying Stimuli with the HDL Simulator force Command (p. 3-18)	Explains how to apply MATLAB link session stimuli with HDL simulator force commands.

Running and Monitoring a MATLAB Link Session (p. 3-20)	Explains how to run and monitor a MATLAB link session.
Stopping a MATLAB Link Session (p. 3-22)	Explains how to stop a MATLAB link session.

Overview

Link for Cadence Incisive software 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.

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

- 1** Check the MATLAB server's link status. (See "Checking the MATLAB Server's Link Status" on page 3-5.)
- 2** Start the MATLAB server. (See "Starting the MATLAB Server" on page 3-7.)
- 3** Launch the HDL simulator with the compiled and elaborated model for use with MATLAB. (See "Starting the HDL Simulator for Use with MATLAB" on page 3-10.)
- 4** Schedule invocations of the MATLAB test bench or component function. (See "Deciding on MATLAB Link Session Scheduling Options" on page 3-11.)
- 5** Control callback timing from the MATLAB test bench or component function. (See "Controlling Callback Timing from a MATLAB Test Bench or Component Function" on page 3-12.)
- 6** Initialize the HDL simulator for use with MATLAB as a link session tool. (See "Initializing the HDL Simulator for a MATLAB Link Session" on page 3-13.)
- 7** Apply MATLAB link session stimuli. (See "Applying Stimuli with the HDL Simulator force Command" on page 3-18.)

- 8 Run and monitor the MATLAB link session. (See “Running and Monitoring a MATLAB Link Session” on page 3-20.)
- 9 Stop a MATLAB link session. (See “Stopping a MATLAB Link Session” on page 3-22.)

Each of these steps is described in more detail in this chapter.

A complete example of starting and controlling a MATLAB component session appears in the oscillator filter demo.

Checking the MATLAB Server's Link Status

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

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

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

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

If the server is not running, the message reads

```
HLDaemon is NOT running
```

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

```
x=hlddaemon('status')
HLDaemon socket server is running on port 4449 with 0 connections
x =
    comm: 'sockets'
  connections: 0
    ipc_id: '4449'
```

This function call indicates that the server is using TCP/IP socket communication with socket port 4449 and is running with no connections. If a shared memory link is in use, the value of `comm` is `'shared memory'` and the value of `ipc_id` is a file system name for the shared memory communication channel. For example:

```
x=hlddaemon('status')
HLDaemon shared memory server is running with 0 connections
x =
    comm: 'shared memory'
```

```
connections: 0  
ipc_id: [1x45 char]
```


Starting the MATLAB Server

Start the MATLAB server as follows:

- 1 Start MATLAB.
- 2 In the MATLAB Command Window, call the `hdldaemon` function with property name/property value pairs that specify whether the Link for Cadence Incisive interface is to
 - Use shared memory or TCP/IP socket communication
 - Return time values in seconds or as 64-bit integers

Use the following syntax:

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

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

Note The communication mode that you specify (shared memory or TCP/IP sockets) must match what you specify for the communication mode when you initialize the HDL simulator for use with a MATLAB with the `matlabtb` or `matlabtbeval` HDL simulator Tcl command. In addition, if you specify TCP/IP socket mode, the socket port that you specify with this function and the HDL simulator Tcl command must match. For more information on modes of communication, see “Choosing TCP/IP Socket Ports” on page 1-18. For more information on establishing the HDL simulator end of the communication link, see “Initializing the HDL Simulator for a MATLAB Link Session” on page 3-13.

If Your Application Is to...**Do the Following...**

Operate in shared memory mode

Omit the 'socket', *tcp_spec* property name/property value pair. The interface operates in shared memory mode by default. You should use shared memory mode if your application configuration consists of a single system and uses a single communication channel.

Operate in TCP/IP socket mode, using a specific TCP/IP socket port

Specify the 'socket', *tcp_spec* property name and value pair. The *tcp_spec* can be a socket port number or service name. Examples of valid port specifications include '4449', 4449, and MATLAB Service. For information on choosing a TCP/IP socket port, see “Choosing TCP/IP Socket Ports” on page 1-18.

Operate in TCP/IP socket mode, using a TCP/IP socket that the operating system identifies as available

Specify 'socket', 0 or 'socket', '0'.

Return time values in seconds (type double)

Specify 'time', 'sec' or omit the parameter. This is the default time value resolution.

Return 64-bit time values (type int64)

Specify 'time', 'int64'.

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

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

You also can start the server from a script. Consider the following function call sequence:

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

```
portnum = dstat.ipc_id
```

The first call to `hdldaemon` specifies that the server use TCP/IP communication with a port number that the operating system identifies and returns connection status information, including the assigned port number, to `dstat`. The statement on the second line assigns the socket port number to `portnum` for future reference.

For more information on modes of communication, see “Choosing TCP/IP Socket Ports” on page 1-18. For more information on establishing the HDL simulator end of the communication link, see “Initializing the HDL Simulator for a MATLAB Link Session” on page 3-13.

Starting the HDL Simulator for Use with MATLAB

After you compile and elaborate your model, start the HDL simulator from outside of MATLAB by calling the HDL simulator Tcl command `hdlsimmatlab` from inside the HDL simulator.

First, in the OS shell type:

```
% simvision -input tclscript
```

where *tclscript* is the name of the Tcl startup script you created when setting up Link for Cadence Incisive software. See “Setting Up Link for Cadence Incisive Software for Use with the Incisive Simulator on the Same Machine as MATLAB” on page 1-23.

Next, at the SimVision prompt type:

```
SimVision> hdlsimmatlab -gui component_instance
```

where *component_instance* is the instance of the component you created for this particular link session.

Deciding on MATLAB Link Session Scheduling Options

A MATLAB link session is the application of a `matlabtb`, `matlabtbeval`, or `matlabcp` function. By default, Link for Cadence Incisive software invokes a MATLAB test bench or component function once (when time equals 0). If you want to apply more control and execute the MATLAB function more than once, decide on scheduling options that specify when and how often the Link for Cadence Incisive software is to invoke the relevant MATLAB function. Depending on your choices, you may need to modify the function or specify specific arguments when you initiate a MATLAB link session with the `matlabtb`, `matlabtbeval`, or `matlabcp` command.

You can schedule a MATLAB simulation function to execute

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

Decide on a combination of options that best meet your test bench or component application requirements. For details on using the `tnext` parameter, see “Controlling Callback Timing from a MATLAB Test Bench or Component Function” on page 3-12. For information on setting other scheduling parameters, see “Initializing the HDL Simulator for a MATLAB Link Session” on page 3-13.

Controlling Callback Timing from a MATLAB Test Bench or Component Function

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

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

Time Representations for `tnext` Parameter

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

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

Initializing the HDL Simulator for a MATLAB Link Session

After you decide on scheduling options, you are ready to initialize the HDL simulator for a specific MATLAB link session. You initialize the HDL simulator for a cosimulation session with the `matlabtb`, `matlabtbeval`, or `matlabcp` command, which do the following:

- Identify the instance of a module in the HDL model being simulated and identified with a test bench or component
- Define the communication link between the HDL simulator and MATLAB
- Specify a callback to a MATLAB function that executes in the context of MATLAB on behalf of the instance under simulation in the HDL simulator

In addition, `matlabtb` commands 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 `hdlsimrand`, the command links the instance with the MATLAB function `hdlsimrand` in file `hdlsimrand.m`. Alternatively, you can specify a different function name with the option `-mfunc`.

To apply TCP/IP socket communication, specify the command with the `-socket` option and a TCP/IP specification. For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page 1-18.

Note The communication mode and, if appropriate, the TCP/IP specification that you specify with the `matlabtb` or `matlabtbeval` command must match what you specify for the communication mode when you call the `hdldaemon` function in MATLAB.

For more information on modes of communication, see “Modes of Communication” on page 1-9. For information on choosing socket ports, see “Choosing TCP/IP Socket Ports” on page 1-18. For more information

on starting the MATLAB end of the communication link, see “Starting the MATLAB Server” on page 3-7.

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

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

For more about ticks and HDL time resolution, see “Representation of Simulation Time” on page 4-8.

Simulation Scheduling Options

To Specify MATLAB Function Execution...	Include...	Where...
At explicit times	<code>time[, ...]</code>	<p><code>time</code> represents one of n time values, past time 0, at which the MATLAB function executes.</p> <p>For example:</p> <pre>matlabtb entity 10 ns, 10 ms, 10 s -mfunc function</pre> <p>The MATLAB function executes when time equals 0 and then 10 nanoseconds, 10 milliseconds, and 10 seconds from time zero.</p>

Simulation Scheduling Options (Continued)

To Specify MATLAB Function Execution...	Include...	Where...
<p>At a combination of explicit times and repeatedly at an interval</p> <p>Stop the executions after <i>x</i> amount of time</p>	<p>time[, ...] -repeat <i>n</i></p> <p>time[, ...] <i>n</i> -repeat <i>x</i> -cancel</p>	<p>time represents a time value at which the MATLAB function executes and the <i>n</i> specified with -repeat represents an interval between MATLAB function executions.</p> <p>For example:</p> <pre>matlabtb entity 5 ns -repeat 10 ns -mfunc function</pre> <p>The MATLAB function executes at time equals 0 ns, 5 ns, 15 ns, 25 ns, and so on. This repetition continues indefinitely, unless cancel with <i>x</i> time value is used.</p> <p>For example:</p> <pre>matlabtb entity 5 ns -repeat 10 ns -cancel 1 us -mfunc function</pre> <p>This cancellation stops the execution after 100 microseconds.</p>

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 the pathname of a signal defined as a logic type.
On change of signal values (sensitivity list)	-sensitivity signal[, ...]	<p>signal represents the pathname of a signal defined as any type. If the value of one or more signals in the specified list changes, the interface invokes the MATLAB function.</p> <hr/> <p>Note Use of this option for INOUT ports can result in double calls.</p> <hr/>

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 the HDL simulator rules to resolve signal specifications.

The following `matlabtb` command:

```
ncsim> matlabtb hdlsimrand -rising hdlsimrand.clk,
        -socket 4449
```

links an instance of the module `hdlsimrand` to function `hdlsimrand.m`, which executes within the context of MATLAB based on specified timing parameters. In this case, the MATLAB function is called when the signal `hdlsimrand.clk` experiences a rising edge.

Arguments in the command line specify the following:

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

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

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

If a connection exists, the function returns the message

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

Applying Stimuli with the HDL Simulator force Command

After you establish a link between the HDL simulator and MATLAB, you are ready to apply stimuli to the MATLAB link session environment. One way of applying stimuli is through the `iport` return parameter of the linked MATLAB function. This parameter drives signal values by deposit. Another option is to issue force commands in the HDL simulator main window.

For example, the following sequence of force commands:

```
force osc_top.clk_enable 1 -after 0ns
force osc_top.reset 0 -after 0ns 1 -after 40ns 0 -after 120ns
force osc_top.clk 1 -after 0ns 0 -after 40ns -repeat 80ns
```

can be entered at the `ncsim` prompt or in the Tcl pane of the HDL cosim block (in the presimulation entry box).

These commands drive

- The `clk` signal to 0 at 0 nanoseconds after the current simulation time and to 1 at 5 nanoseconds after the current HDL simulator 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.



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

Note You should consider using HDL to code clock signals as force is a lower performance solution in the current version of Cadence Incisive simulators.

The following are ways that a periodic force might be introduced:

- Via the Clock pane in the HDL Cosim block
- Via pre/post Tcl commands in the HDL Cosim block
- Via a user-input Tcl script to ncsim

All three approaches may lead to performance degradation.

Running and Monitoring a MATLAB Link Session

Start a MATLAB link session from the HDL simulator. The HDL simulator offers a number of options for running a simulation to debug, analyze, or verify an HDL model. The following sequence is typical for running a simulation interactively from the main HDL simulator window:

- 1** Start the simulation by entering the HDL simulator run command or selecting the **Simulation > Run** option in the SimVision console of the Incisive simulator.

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 the HDL simulator to run the loaded simulation for 50000 time steps:

```
run 50000
```

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

The following HDL simulator command sets a breakpoint at line 50 in the Verilog file `hdlsimrand.v`:

```
bp hdlsimrand.v 50
```

- 3** Step through the simulation and examine values.
- 4** When you block execution of the MATLAB function, the HDL simulator also blocks and remains blocked until you clear all breakpoints in the function's M-code.
- 5** Resume the simulation, as needed.

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

Note When you restart a simulation in Simulink, Simulink starts at time 0, but ModelSim continues from where the last simulation was stopped. To synchronize the time axis, issue the following Tcl command in ModelSim before restarting the cosimulation:

```
restart -f
```

Stopping a MATLAB Link Session

When you are ready to stop a MATLAB link 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 in the following sequence:

- 1 Make the HDL simulator your active window, if your input focus was not already set to that application.
- 2 Halt the simulation by selecting the **Simulation > Stop** option on the main window.
- 3 Exit the HDL simulator, if you are finished with the application.
- 4 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 Incisive simulator sessions, see the Incisive simulator documentation.

Modeling and Verifying an HDL Design with Simulink

Overview (p. 4-3)	Provides an overview of the process for integrating Link for Cadence Incisive blocks into a Simulink design.
Creating a Hardware Model Design in Simulink (p. 4-5)	Lists questions to think about as you decide to include Simulink in an EDA solution.
Handling Signal Values Across Simulators (p. 4-7)	Explains how Link for Cadence Incisive software addresses the differences in treatment of simulation time in the HDL simulator and Simulink.
Configuring Simulink for HDL Models (p. 4-18)	Gives suggestions for configuring Simulink more optimally for use with Link for Cadence Incisive blocks.
Adding the HDL Representation of a Model Component into a Simulink Model (p. 4-19)	Explains how to integrate the HDL representation of a model component into a Simulink model with Link for Cadence Incisive blocks.
Configuring an HDL Cosimulation Block (p. 4-20)	Explains how to use a Simulink block parameters dialog to configure Link for Cadence Incisive blocks.

Running and Testing a Cosimulation Model in Simulink (p. 4-41)	Explains how to start a cosimulation model in Simulink. This section also explains how to reset clocks and restart the HDL simulator during testing.
Using Frame-Based Processing in Cosimulation (p. 4-42)	Explains how to improve the performance of your cosimulation by using frame-based signals.
Using a Value Change Dump File for Design Verification (p. 4-44)	Explains how to use the To VCD File block to generate Value Change Dump files.

Overview

HDL simulators, Simulink, and Simulink blocksets provide a powerful modeling and cosimulation environment for Electronic Design Automation (EDA). This chapter explains how to set up a cosimulation environment in Simulink that includes HDL models designed and simulated with Incisive simulators.

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

Two potential use cases follow:

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

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

- 1** Design your application model in Simulink. One or more components of the model can represent hardware that you intend to describe with HDL.
- 2** Run and test the model design in Simulink.
- 3** Verify that the model runs as expected. If it does not, repeat steps 1 and 2 to rework and fine tune the design.
- 4** Use the HDL simulator to simulate a discrete model component of the design coded in HDL.
- 5** Integrate the HDL representation of the model component into the Simulink model as an HDL Cosimulation block.
- 6** Configure the HDL Cosimulation block. The block parameters dialog box includes tabs for configuring port, communication, clock, and Tool Command Language (Tcl) commands.

- 7** Run and test the revised model design in Simulink.
- 8** Verify that the revised model runs as expected. If it does not,
 - a** Modify the HDL code and simulate it in the HDL simulator.
 - b** Determine whether you need to re-configure the HDL Cosimulation block. If you do, repeat steps 6 to 8. If you do not, repeat steps 7 and 8.
- 9** Determine whether you need to replace another component of the Simulink model with an HDL Cosimulation block. If you do, go to step 4.
- 10** Consider using a To VCD File block to verify cosimulation results.

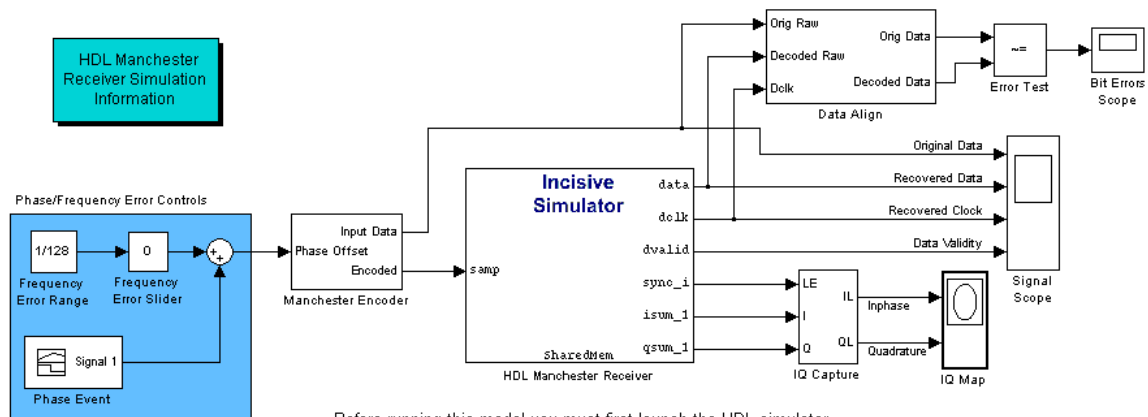
Creating a Hardware Model Design in Simulink

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 an HDL simulator, 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.

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



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

The HDL Cosimulation block (labeled HDL Manchester Receiver) models a Manchester receiver that is coded in VHDL and Verilog. Other blocks and subsystems in the model include the following:

- Frequency Error Range block, Frequency Error Slider block, and Phase Event block
- Manchester encoder subsystem
- Data alignment subsystem
- Inphase/Quadrature (I/Q) capture subsystem
- Error Rate Calculation block from the Communications Blockset
- Bit Errors block
- Data Scope block
- Discrete-Time Scatter Plot Scope block from the Communications Blockset

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

Handling Signal Values Across Simulators

The Link for Cadence Incisive HDL Cosimulation block serves as a bridge between the Simulink and HDL simulators. The block represents an HDL component model within Simulink. Using the block, Simulink writes signals to and reads signals from the HDL model under simulation in the HDL simulator. Signal exchange between the two simulators occurs at regularly scheduled time steps defined by the Simulink sample time.

As you develop a Link for Cadence Incisive cosimulation application, you should be familiar with how signal values are handled across simulators. See the following topics:

- “How Simulink Drives Cosimulation Signals” on page 4-7
- “Representation of Simulation Time” on page 4-8
- “Handling Multirate Signals” on page 4-15
- “Clock Signal Latency” on page 4-16
- “Block Simulation Latency” on page 4-16

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 HDL model.

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 the HDL simulator and Simulink.

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

To determine the current HDL simulator resolution limit, enter `echo $timescale` at the HDL simulator prompt. See the HDL simulator documentation for the application you are using for further information.

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

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

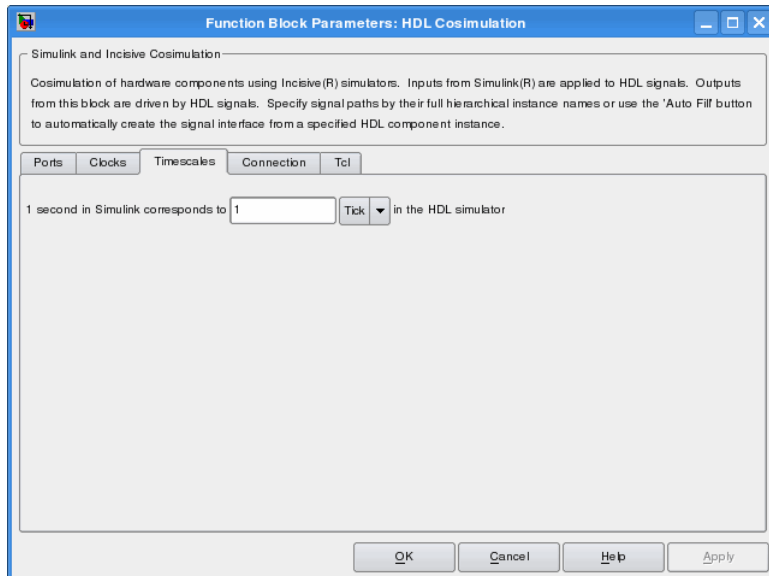
- Total simulation time
- Input port sample times
- Output port sample times
- Clock periods

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

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

Defining the Simulink and HDL Simulator Timing Relationship

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



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

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

Note In both timing modes, all sample times and clock periods in Simulink must be an integer multiple of the resolution units. An error occurs if they are not.

The following sections discuss these two timing modes.

Relative Timing Mode

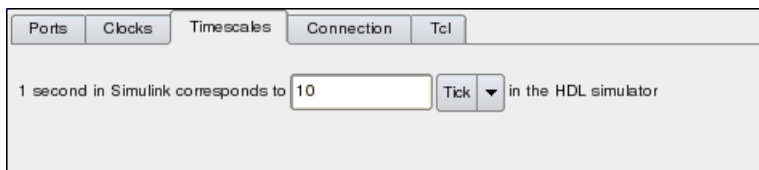
Relative timing mode lets you define the timing relationship between Simulink and the HDL simulator in terms of relative time units and a scale factor, e.g., *One second* in Simulink corresponds to *N ticks* in the HDL simulator, where N is a scale factor.

This correspondence holds regardless of the HDL simulator timing resolution.

To configure relative timing mode for a cosimulation:

- 1 Click the **Timescales** tab of the HDL Cosimulation block parameters dialog.
- 2 Select **Tick** (default value) from the list on the right.
- 3 Enter a scale factor in the edit box on the left. The default scale factor is 1.

For example, in the following figure, the **Timescales** pane is configured for a relative timing correspondence of 10 HDL simulator ticks to 1 Simulink second.



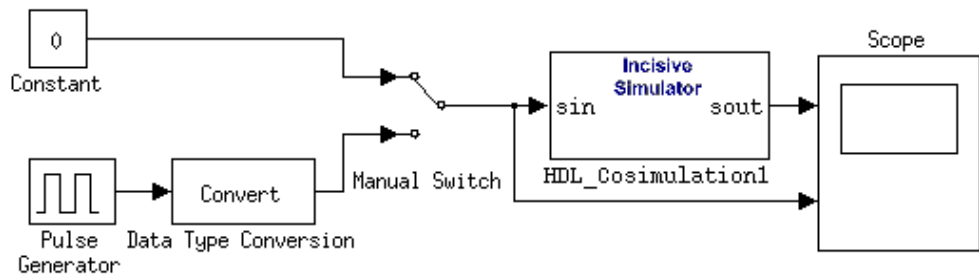
- 4 Click **Apply** to commit your changes.

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

- If the total simulation time in Simulink is specified as N seconds, then the HDL simulation runs for exactly N ticks (i.e., N ns at the default resolution limit).
- Similarly, if Simulink computes the sample time of an HDL Cosimulation block input port as T_{si} seconds, new values are deposited on the HDL input port at exact multiples of T_{si} ticks. If an output port has an explicitly specified sample time of T_{so} seconds, values are read from the HDL simulator at multiples of T_{so} 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 the HDL simulator 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 the HDL simulator as an input signal that stays low for $T/2$ ticks and stays high for $(T/2) + 1$ ticks.

Note Simulink requires such clocks to have a period of at least 2 resolution units (ticks). Simulink throws an error if specified value of T is less than 2 ticks.

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



The model contains an HDL Cosimulation block (labeled HDL_Cosimulation1) simulating an 8-bit inverter that is enabled by an explicit clock. The inverter has a single input and a single output. The following lists the Verilog code for the inverter:

```
module inverter_clock_v1(sin, sout,clk);

    input [7:0] sin;
    output [7:0] sout;
    input clk;
    reg [7:0] sout;

    always @(posedge clk)
        sout <= ! (sin);
endmodule
```

A cosimulation of this model might have the following settings:

- Simulation parameters in Simulink
 - **Timescales** parameters: 1 Simulink second = 10 HDL simulator ticks
 - Total simulation time: 30 s
 - Input port (inverter_clock_v1.sin) sample time: N/A
 - Output port (inverter_clock_v1.sout) sample time: 1 s
 - Clock (inverter_clock_v1.clk) period: 5 s
- HDL simulator resolution limit: 1 ns

The previous example was excerpted from the [Link for Cadence Incisive Inverter tutorial](#). For more information, see [Link for Cadence Incisive demos](#).

Absolute Timing Mode

Absolute timing mode lets you define the timing relationship between Simulink and the HDL simulator in terms of absolute time units and a scale factor, e.g., *One second* in Simulink corresponds to $(N * Tu)$ seconds in the HDL simulator, where Tu is an absolute time unit (e.g., ms, ns, etc.) and N is a scale factor.

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

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 edit box on the left. The default scale factor is 1.

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



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

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

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

where

- $qtInTicks$ is the integer multiple of HDL simulator time in ticks (minimum 2).
- $tInSecs$ is the Simulink time in seconds.
- $tScale$ is the time scale setting (unit and scale factor) chosen in the **Timescales** pane of the HDL Cosimulation block.
- tRL is the HDL simulator resolution limit.

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

$$qtInTicks = (12\text{ns} * (1\text{s} / 1\text{ns})) = 12$$

Operation of Absolute Timing Mode. To understand the operation of absolute timing mode, review the example model discussed in “Representation of Simulation Time” on page 4-8. Suppose that the model is re-configured as follows:

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

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

Timing Mode Usage Restrictions

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

- All HDL Cosimulation blocks in the model that communicate with the same single instance of the HDL simulator must all be configured either in relative timing mode or in absolute timing mode.
- When multiple HDL Cosimulation blocks in a model are communicating with a single instance of the HDL simulator, all HDL Cosimulation blocks must have the equivalent **Timescales** pane settings.

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

Setting HDL Cosimulation Block Port Sample Times

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

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

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

- Specify the output sample time for an HDL Cosimulation block as an integer multiple of the resolution limit defined in the HDL simulator. Use the HDL simulator command `echo $timescale` to check the resolution limit of the loaded model.
- Specify the Simulink model's start and stop time values (see the **Solver** pane of the Simulink Configuration Parameters dialog box) as integers. Start time equals a multiple of all sample/frame rates.
- 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.

Handling Multirate Signals

Link for Cadence Incisive software supports the use of multirate signals, signals that are sampled or updated at different rates, in a single HDL Cosimulation block. An HDL Cosimulation block exchanges data for each signal at the Simulink sample rate for that signal. For input signals, 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 explicit setting lets you control the rate at which Simulink updates an output port by reading the corresponding signal from the HDL simulator.

Clock Signal Latency

In an HDL simulator, 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.

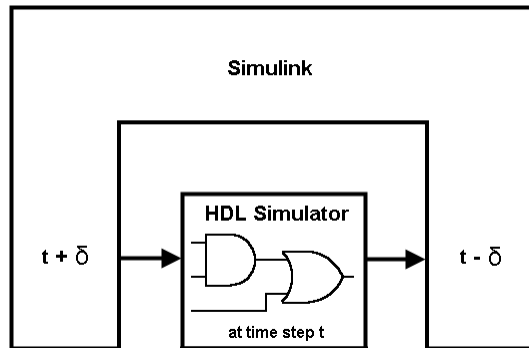
To avoid such race conditions, Link for Cadence Incisive software delays all such clocks by $\frac{1}{2}$ clock period, in effect inverting the sense of the rising or falling edge. The delay provides a setup and hold time for input data, ensuring that data inputs are always applied before the driving clock edge is applied. For example, in the case of a rising-edge clock, inputs are applied first, and $\frac{1}{2}$ clock period later, the rising edge of the clock is applied.

Where the Simulink sample time is even, the clock delay is exactly $\frac{1}{2}$ period. For odd Simulink sample times, the $\frac{1}{2}$ period delay is approximated as closely as possible. While this apparent inversion or delay by $\frac{1}{2}$ 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 Link for Cadence Incisive cosimulation blocks supplement the hardware simulator environment, rather than operate as part of it. During cosimulation, Simulink does not participate in HDL simulator delta-time iteration. From the Simulink perspective, all signal drives (reads) occur during a single delta-time cycle. For this reason, and due to fundamental differences between HDL simulators and Simulink with regard to use and treatment of simulation time, some degree of latency is introduced when you use Link for Cadence Incisive 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 HDL simulator time step ($t+\delta$) and block output reflects signal values just before the current HDL simulator step time ($t-\delta$).



Regardless of whether your HDL code is specified with latency, the cosimulation block has a minimum latency that is equivalent to the cosimulation block's output sample time. For large sample times, the delay can appear to be quite long, but this apparent length is because of the cosimulation block, which exchanges data with the HDL simulator at the block's output sample time only. This condition 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, you may need to adjust the sample time to achieve simulation performance required for circuit analysis.

For cosimulation blocks that model combinatorial circuits, you may want to experiment with a faster sample frequency for output ports. Although this type of parameter tuning can increase simulation performance, it can also make a model more difficult to debug. For example, you may need to adjust the output sample time for each cosimulation block.

Configuring Simulink for HDL Models

When you create a Simulink model that includes one or more Link for Cadence Incisive 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 via the GUI. These are some of the default settings you might expect to use in cosimulation:

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.

Adding the HDL Representation of a Model Component into a Simulink Model

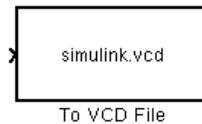
After you code one of your model's components in Verilog or VHDL and simulate it in the Incisive simulator environment, integrate the HDL representation into your Simulink model as an HDL Cosimulation block:

- 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 Cadence Incisive library. The browser displays the following block icons.



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 "Using a Value Change Dump File for Design Verification" on page 4-44.

- 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.

Configuring an HDL Cosimulation Block

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

- **Ports** — Block input and output ports that correspond to signals, including internal signals, of your HDL design, and an output sample time
- **Connection** — Type of communication and communication settings to be used for exchanging data between simulators
- **Timescales** — Timing relationship between Simulink and the Link for Cadence Incisive interface
- **Clocks** — Rising-edge and falling-edge clocks to apply to your model
- **Tcl** — Tcl commands to run before and after a simulation

The following sections help you identify what you need to configure, how to open the Block Parameters dialog box, and how to configure each pane:

- “What Are Your HDL Cosimulation Block Requirements?” on page 4-20
- “Opening the Block Parameters Dialog Box” on page 4-23
- “Mapping HDL Signals to Block Ports” on page 4-23
- “Specifying Data Types for Output Ports” on page 4-28
- “Configuring the Simulink and Incisive Simulator Timing Relationship” on page 4-30
- “Configuring the Communication Link” on page 4-32
- “Creating Optional Clocks” on page 4-34
- “Executing Tcl Commands Before and After Cosimulation” on page 4-37
- “Applying Your Block Parameters Configuration Settings” on page 4-40

What Are Your HDL Cosimulation Block Requirements?

Before you start to configure an HDL Cosimulation block, review the following checklist. The checklist helps you identify the parameters you need to set.

If your answer to a question is something other than “no,” go to the topic listed in the second column of the table for information on how to adjust the parameter setting to meet your block requirements.

HDL Cosimulation Block Requirements Checklist

Requirement	For More Information, See...
Ports	
<input type="checkbox"/> Does the HDL model you are mapping to Simulink receive signals on input ports? If so, what are the input ports?	“Mapping HDL Signals to Block Ports” on page 4-23
<input type="checkbox"/> Does the HDL model you are mapping to Simulink transmit signals to output ports? If so, what are the output ports?	“Mapping HDL Signals to Block Ports” on page 4-23
<input type="checkbox"/> If the block is modeling an input and output device, do you want to specify explicit sample times for output ports?	“Mapping HDL Signals to Block Ports” on page 4-23
<input type="checkbox"/> If the block is modeling an input and output device, do you want to specify explicit fixed point data types for output ports? By default the data types are either inherited from the signals connected to the HDL Cosimulation block output ports or derived from the HDL model.	“Specifying Data Types for Output Ports” on page 4-28
<input type="checkbox"/> If the block is modeling a source device, do you want to specify an output sample time other than two clock ticks? If you do not specify an input port, the block uses a default sample time of two clock ticks.	“Mapping HDL Signals to Block Ports” on page 4-23
Timing	
<input type="checkbox"/> What is the optimal timing relationship between Simulink and the Incisive simulator for your cosimulation?	“Representation of Simulation Time” on page 4-8
<input type="checkbox"/> Do you need to specify a relative (Simulink seconds corresponding to Incisive simulator ticks) timing relationship between Simulink and the Incisive simulator?	“Configuring the Simulink and Incisive Simulator Timing Relationship” on page 4-30

HDL Cosimulation Block Requirements Checklist (Continued)

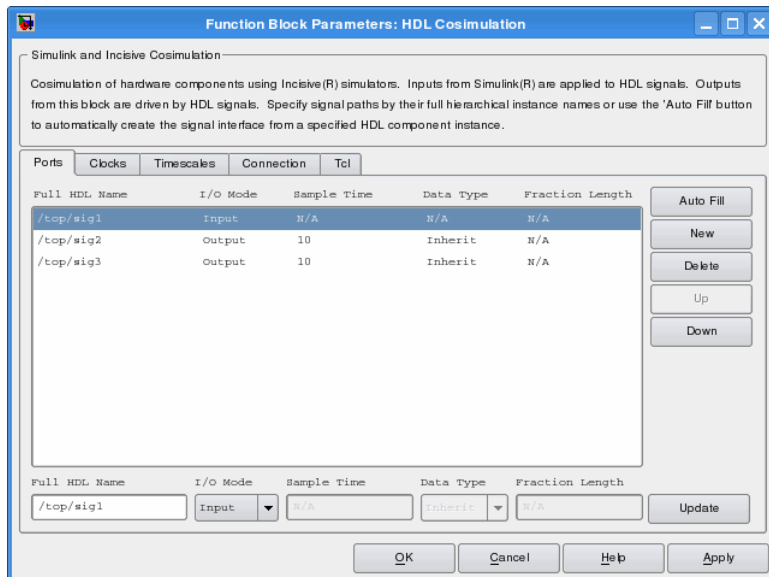
Requirement	For More Information, See...
<input type="checkbox"/> Do you need to specify an absolute (Simulink seconds corresponding to Incisive simulator absolute time units) timing relationship between Simulink and the Incisive simulator?	“Configuring the Simulink and Incisive Simulator Timing Relationship” on page 4-30
Communication	
<input type="checkbox"/> Is it critical that communication performance be as optimal as possible?	“Configuring the Communication Link” on page 4-32
<input type="checkbox"/> Are you running the Incisive simulator and Simulink on the same computer?	“Configuring the Communication Link” on page 4-32
<input type="checkbox"/> If the Incisive simulator and Simulink are running on the same computer, do you want to use shared memory communication?	“Configuring the Communication Link” on page 4-32
<input type="checkbox"/> Do you want to choose a TCP/IP socket port? If so, what port number or service will you use to establish a link?	“Configuring the Communication Link” on page 4-32
<input type="checkbox"/> If you are running the Incisive simulator and Simulink different computers, what is the host name of the computer running the Incisive simulator?	“Configuring the Communication Link” on page 4-32
Clocks	
<input type="checkbox"/> Do you want to create a rising-edge clock to apply stimuli to your cosimulation model?	“Creating Optional Clocks” on page 4-34
<input type="checkbox"/> Do you want to create a falling-edge clock to apply	“Creating Optional Clocks” on page
<input type="checkbox"/> Do you want to specify the period for rising/falling edge clocks specified in the model?	“Creating Optional Clocks” on page 4-34
Tcl	
<input type="checkbox"/> Are there any Tcl commands that you want the Incisive simulator to execute before running a simulation, but after loading the project in the Incisive simulator?	“Executing Tcl Commands Before and After Cosimulation” on page 4-37
<input type="checkbox"/> Are there any Tcl commands that you want the Incisive simulator to execute after running a simulation?	“Executing Tcl Commands Before and After Cosimulation” on page 4-37

Opening the Block Parameters Dialog Box

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



Simulink displays the following Block Parameters dialog box.



Mapping HDL Signals to Block Ports

The first step to configuring your Link for Cadence Incisive 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 use either of the following methods:

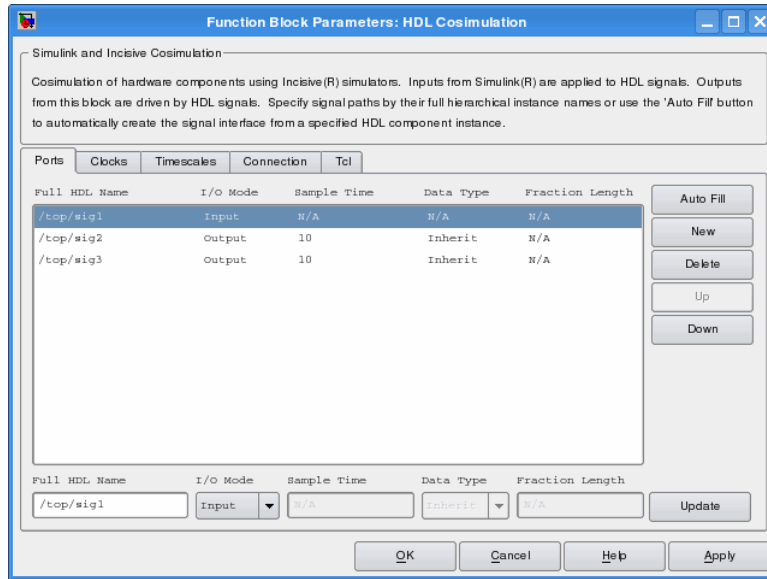
- Enter signal information manually into the **Ports** pane of the HDL Cosimulation Block Parameters dialog (see “Entering Signal Information Manually” on page 4-24). This approach can be more efficient when you want to connect a small number of signals from your HDL model to Simulink.
- Use the **Auto Fill** button to obtain signal information automatically by transmitting a query to the Incisive simulator. 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. In many cases, however, you will need to edit the signal data returned by the query. See “Obtaining Signal Information Automatically from the Incisive Simulator” on page 4-27 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.

Entering Signal Information Manually

To enter signal information directly in the **Ports** pane:

- 1** In the Incisive simulator, determine the test signal pathnames for the HDL signals you plan to define in your block. The Incisive simulator 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.
- 2** In Simulink, open the block parameters dialog box for your HDL Cosimulation block, if it is not already open.
- 3** Select the **Ports** tab of the Block Parameters dialog box. Simulink displays the dialog box as shown in the following figure.



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** text field, using the Incisive simulator pathname syntax. Select either **Input** or **Output** from the **I/O Mode** menu. If desired, set the **Data Type** and **Fraction Length** parameters for signals explicitly, as discussed in step 6.

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

Note When you define an input port, make sure that only one source is set up to drive 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 Cadence Incisive environment, see “Representation of Simulation Time” on page 4-8.

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

If Simulink can determine the data type of the signal connected to the output port, it applies that data type to the output port. For example, the data type of a connected Signal Specification block is known by back-propagation. Otherwise, Simulink queries the Incisive simulator 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** menu.
- b** If the signal has a fractional part, enter the **Fraction Length**.

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

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

Obtaining Signal Information Automatically from the Incisive Simulator

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

The required steps are outlined in the following example procedure.

- 1 Open the block parameters dialog box for the HDL Cosimulation block. Click the **Ports** tab. The **Ports** pane opens.

- 2 Click the **Auto Fill** button. The **Auto Fill** dialog box opens.

This modal dialog box requests a path to a component or module in your HDL model; here you enter an explicit HDL path into the edit field.

- 3 Click **OK** to dismiss the dialog and transmit the query.

- 4 Port data is returned and entered into the **Ports** pane almost instantaneously.

- 5 Click **Apply** to commit the port additions.

- 6 Observe that **Auto Fill** has returned information about *all* inputs and outputs for the targeted component. In many cases, this information includes signals that function in the Incisive simulator 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.

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.

- 7 **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. See also “Specifying Data Types for Output Ports” on page 4-28.

- 8 Before closing the HDL Cosimulation block parameters dialog box, click **Apply** to commit any edits you have made.

Note 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.

Specifying Data Types for Output Ports

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

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

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

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

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

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

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

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

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

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

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

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

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

Configuring the Simulink and Incisive Simulator Timing Relationship

You configure the timing relationship between Simulink and the Incisive simulator by using the **Timescales** pane of the block parameters dialog box. Before setting the **Timescales** parameters, you should read “Representation of Simulation Time” on page 4-8 to understand the supported timing modes and the issues that will determine your choice of timing mode.

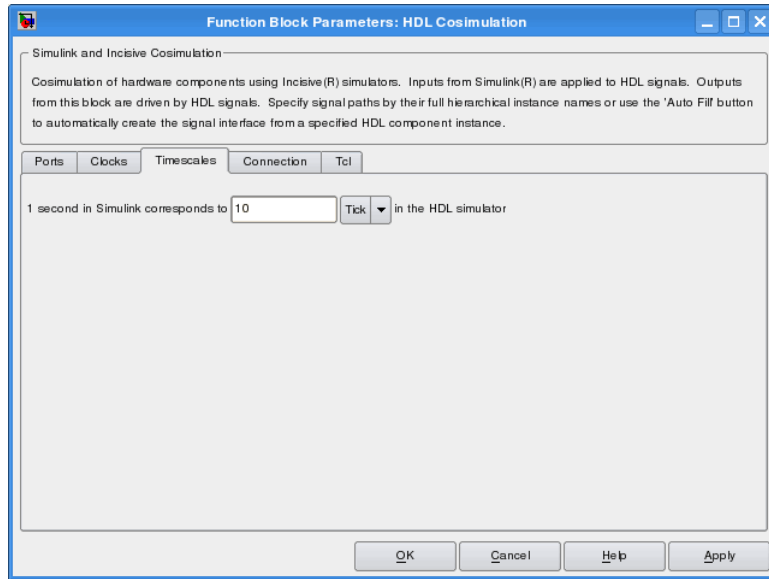
You can specify either a relative or an absolute timing relationship between Simulink and the Incisive simulator, as described in the following sections .

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 box.
- 2 Select Tick (default value) from the list on the right.
- 3 Enter a scale factor in the edit box on the left. The default scale factor is 1.

For example, in the following figure, the **Timescales** pane is configured for a relative timing correspondence of 10 Incisive simulator ticks to 1 Simulink second.



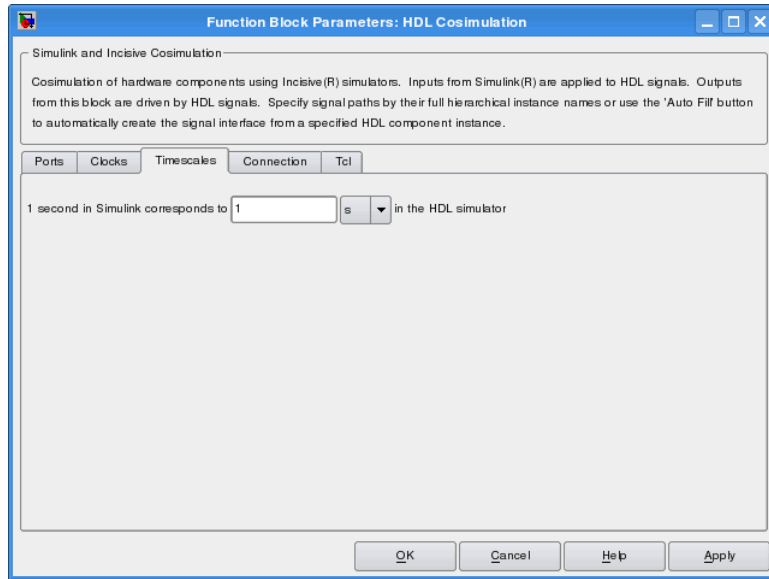
4 Click **Apply** to commit your changes.

Specifying an Absolute Timing Relationship

To configure absolute timing mode for a cosimulation:

- 1** Select the **Timescales** tab of the HDL Cosimulation block parameters dialog box.
- 2** Select a unit of absolute time from the list on the right. Available units are fs, ps, ns, us, ms, and s.
- 3** Enter a scale factor in the edit box on the left. The default scale factor is 1.

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



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

Configuring the Communication Link

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

The following steps guide you through the communication configuration.

- 1 Determine whether Simulink and the Incisive simulator are running on the same computer. If they are, skip to step 4.
- 2 Clear the **HDL simulator running on this computer** check box. (This check box is selected by default.) Because Simulink and the Incisive simulator 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 the **Host name** text field. In the **Port number or service** text field, specify a valid port number or service for your computer system. For information on choosing TCP/IP socket ports, see "Choosing TCP/IP Socket Ports" on page 1-18. Skip to step 5.

- 4 If the HDL simulator and Simulink are running on the same computer, decide whether you are going to use shared memory or TCP/IP sockets for the communication channel. For information on the different modes of communication, see “Modes of Communication” on page 1-9.

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

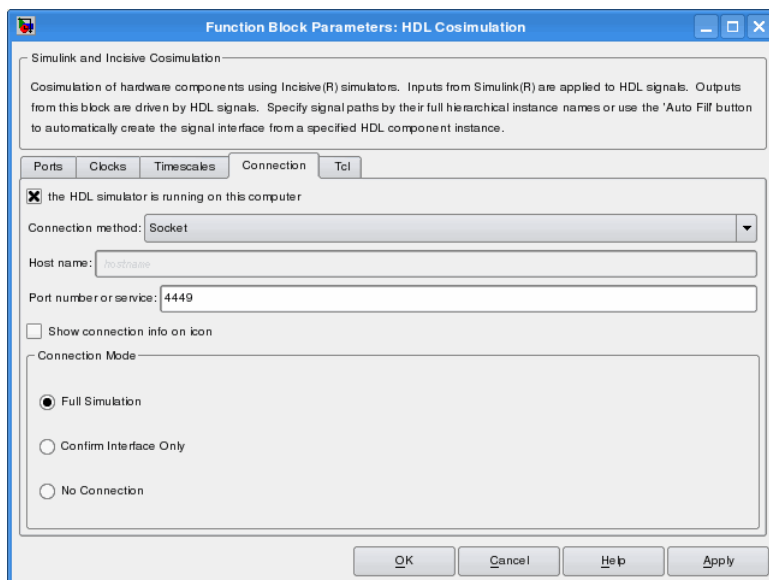
If you choose shared memory communication, select the **Shared memory** check box.

- 5 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, the Link for Cadence Incisive interface does not communicate with the HDL simulator during Simulink simulation.

- 6 Click **Apply**.

The following example dialog shows communication definitions for an HDL Cosimulation block. The block is configured for Simulink and the Incisive simulator running on the same computer, communicating in TCP/IP socket mode over TCP/IP port 4449.

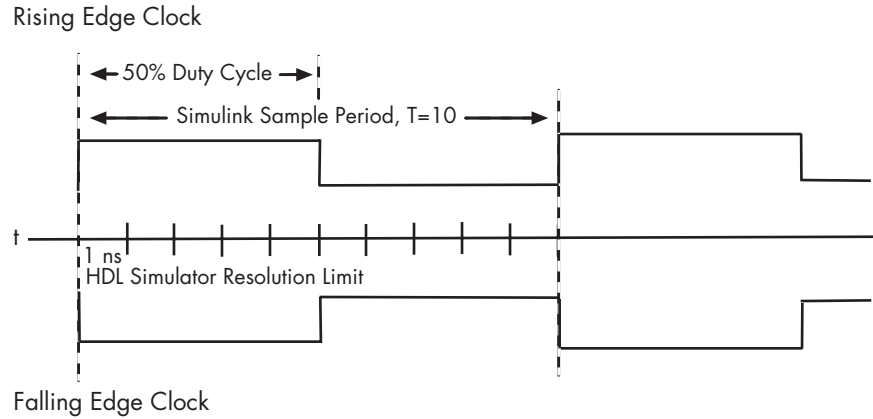


Creating Optional Clocks

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

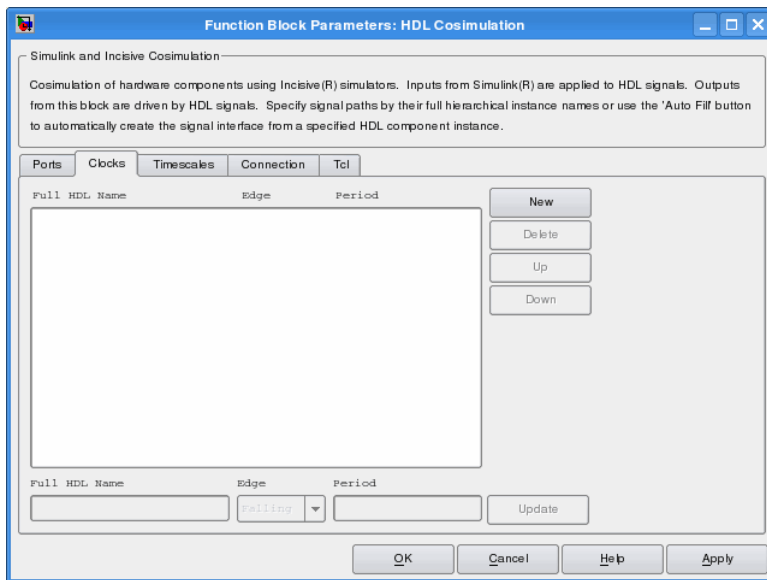
Simulink attempts to create a clock that has a 50% duty cycle and a predefined phase that is inverted for the falling edge case. If necessary, Simulink degrades the duty cycle to accommodate odd Simulink sample times, with a worst case duty cycle of 66% for a sample time of $T=3$.

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



To create clocks:

- 1** In the Incisive simulator, 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 4-23.
- 2** Select the **Clocks** tab of the Block Parameters dialog box. Simulink displays the dialog box as shown in the following figure.

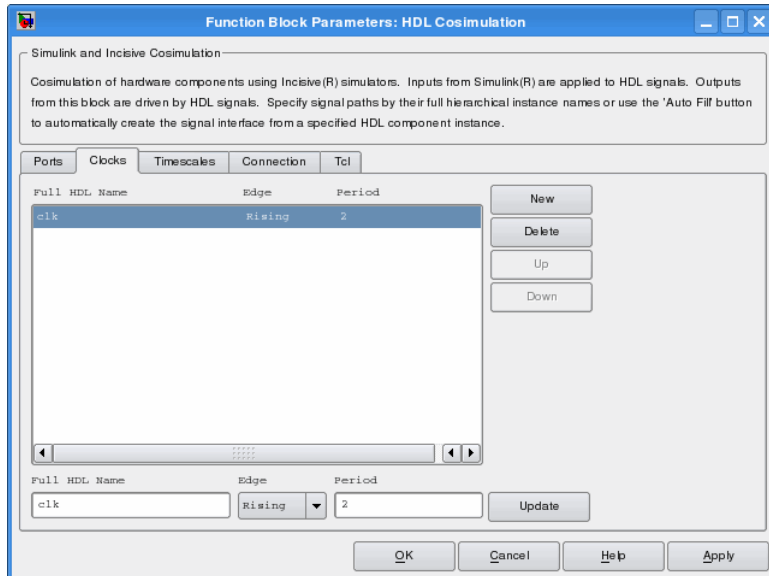


- 3 Click the **New** button to add a new clock signal.
- 4 Enter the clock signal pathname in the **Full HDL Name** text field, using Incisive simulator pathname syntax.

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

Specify the **Period** field as an even integer, with a minimum value of 2.
- 7 After entering the desired property values, click **Update**. This enters the signal values into the signal list in the center of the **Clocks** pane.
- 8 When you have finished editing clock signals, click **Apply** to register your changes with Simulink.

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



Executing Tcl Commands Before and After Cosimulation

You have the option of specifying Tcl commands to execute before and after the Incisive simulator simulates the HDL component of your Simulink model. You can use Tcl for 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 the Incisive simulator to restart at the end of a simulation run.

You can specify the pre- and post-simulation Tcl commands using one of the following methods:

- By entering Tcl commands in the Pre-simulation commands or Post-simulation commands text fields of the HDL Cosimulation block
- By using the Simulink model construction command `set_param`

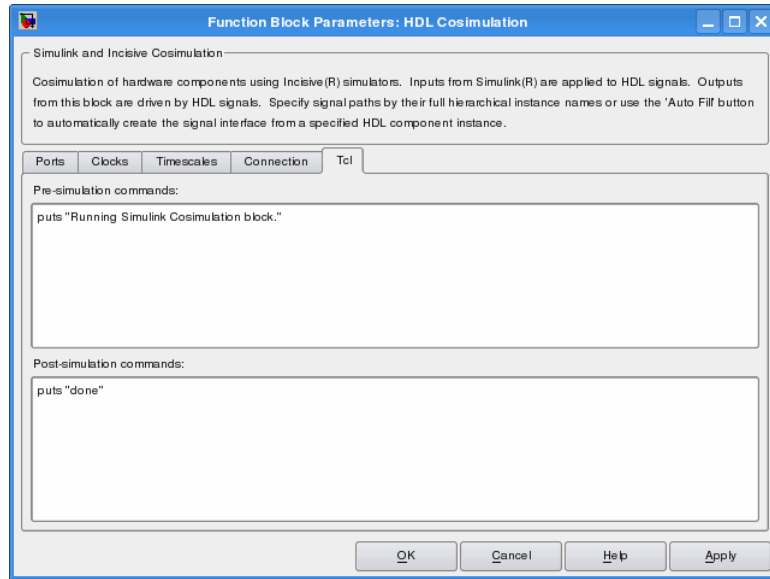
Notes

- You can include the `quit -f` command in a post-simulation Tcl command string to force the Incisive simulator 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.
 - With the exception of `quit` used in a post-simulation Tcl command, the Tcl script that you specify for either pre- simulation or post-simulation cannot include commands that load an Incisive simulator project or modify simulator state. For example, they cannot include commands such as `run`, `stop`, or `reset`.
-

Specifying Pre- and Post-Simulation Tcl Commands with HDL Cosimulation Block Parameters Dialog Box

To specify Tcl commands,

- 1 Select the **Tcl** tab of the Block Parameters dialog box. The dialog box appears as shown in the following figure.



The **Pre-simulation commands** text box includes a puts command for reference purposes.

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

- 3 Click **Apply**.

Specifying Pre- and Post-Simulation Tcl Commands with Simulink Command `set_param`

Use this command to specify pre-simulation and post-simulation Tcl commands. Set the Tcl commands with `set_param` at the MATLAB command prompt.

This example shows setting several pre-simulation Tcl commands:

```
set_param('cosim_blk', 'TclPreSimCommand',...
    ['force sim:/filter2d_v/clock_enable 1;'],...
```

```
'force sim:/filter2d_v/reset 1 0 ns, 0 {1 ns};',...  
'echo "Running Simulink Cosimulation block.";',...  
'echo [clock format [clock seconds]]')])
```

This example shows setting a post-simulation Tcl command:

```
set_param('cosim_blk', 'TclPostSimCommand', 'quit -force');
```

The Tcl pane of the HDL Cosimulation block is automatically updated with the new Tcl commands.

For more about `set_param`, refer to the Simulink documentation.

Applying Your Block Parameters Configuration Settings

After you enter your block parameters settings,

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

To verify the connection with the Incisive simulator and the signal names, select **Edit > Update diagram**, or press **Ctrl+D**.

Running and Testing a Cosimulation Model in Simulink

To run and test a cosimulation model in Simulink, click **Simulation > Start** in your Simulink model window. Simulink runs the model and displays any errors that it detects.

You can use **Edit > Update diagram** to check that the cosimulation interface is correct before running. This menu option connects to the HDL simulator and ensures that data types are correct.

Using Frame-Based Processing in Cosimulation

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

- “Overview” on page 4-42
- “Using Frame-Based Processing” on page 4-42

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 with frame-based processing Simulink interacts with the HDL simulator only once per frame, rather than once per sample. 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 or ports of the HDL Cosimulation block. All such signals must meet the requirements described in “Requirements and Restrictions

for Using Frame-Based Signals” on page 4-43. The HDL Cosimulation block automatically configures its output for frame-based operation at the appropriate frame size.

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

Requirements and Restrictions for Using Frame-Based Signals

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

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

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

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

Using a Value Change Dump File for Design Verification

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

- For comparing results of multiple simulation runs, using the same or different simulator environments
- As input to post-simulation analysis tools

VCD files include data that can be graphically displayed or analyzed with postprocessing tools. For example, VCD files can be displayed in HDL waveform viewers. 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 Cadence Incisive block library serves as a VCD file generator during an HDL simulator 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 file name.

Note The To VCD File block logs the logic states '1' and '0' only. The block does *not* log the logic states 'X' and 'Z'.

The following sections discuss:

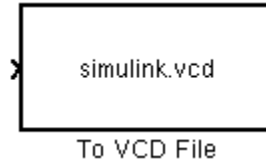
- “Generating a VCD File” on page 4-44
- “VCD File Format” on page 4-47

Generating a VCD File

To generate a VCD file,

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

- 2 Identify the location 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 Cadence Incisive library. The browser displays four types of blocks, one of which is the To VCD File block.

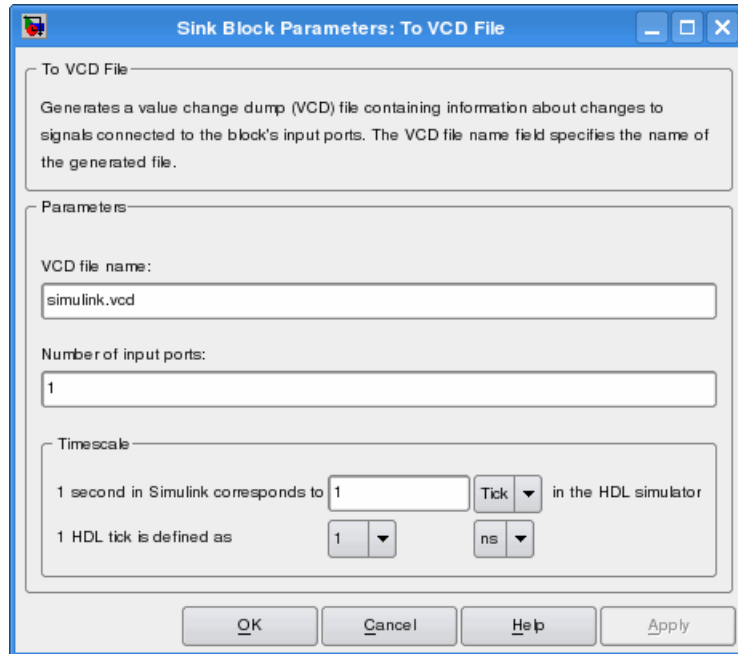


- 4 Copy the To VCD File block from the Library Browser to your model by clicking the block and dragging it from the browser to your model window.
- 5 Connect the block ports to appropriate blocks in your Simulink model.

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 box.
 - a Double-click the block icon. Simulink displays the following dialog box.



- b** Specify a file name for the generated VCD file in the **VCD file name** text box:
- If you specify a file name only, Simulink places the file in your current MATLAB directory.
 - Specify a complete pathname to place the generated file in a different location.
 - If you want the generated file to have a .vcd file type extension, you must specify it explicitly.

Caution 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.

In some cases, a single input port maps to multiple signals (and symbols). This mapping is necessary when the input port receives a vector of real numbers or a fixed-point real number. For example, a signal of type `sfix16_En15` requires 16 symbols.

d Click **OK**.

- 7** 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 4-47.

VCD File Format

The format of generated VCD files adheres to IEEE Std 1364-2001. The following table describes selected contents from a generated VCD file.

Examples of Generated VCD File Format

File Content	Description
<code>\$timescale 1 ns \$ end</code>	All timestamps for VCD variable value changes are related to this single timescale.
<code>\$scope module manchestermodel \$end</code>	The scope module name is a prefix for the signal name in the waveform viewer. The module matches the Simulink mdl file name. The VCD file name is the database prefix for the signal in the waveform viewer.

Examples of Generated VCD File Format (Continued)

File Content	Description
<pre>\$comment SL scale=1.000000 Tick; HDL tick=1 ns; SL2HDL Scaling Factor=1.000000 \$end</pre>	<p>This comment provides feedback about the cosimulation time-scaling specified in the ToVCD block dialog box parameters. In this example, the Simulink timescale is “1 s in Simulink corresponds to 1 tick in the HDL simulator” and the specified HDL timescale is “1 HDL Tick is defined as 1 ns”. These settings mean that the signal sampling times in Simulink are multiplied by 1.0 to determine the VCD timestamps for the signal value-changes.</p>
<pre>\$var wire 1 ! Original Data [0] \$end \$var wire 1 " Recovered Clock [0] \$end \$var wire 1 # Recovered Data [0] \$end \$var wire 1 \$ Data Validity [0] \$end</pre>	<p>Variable definitions. Each definition associates a signal with character identification code (symbol). 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.</p>

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).

MATLAB Functions — Alphabetical List

dec2mvl

Purpose Convert decimal integer to binary string

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

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

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

```
dec2mvl(23)
```

The following function call returns the string '01001':

```
dec2mvl(-23)
```

The following function call returns the string '11101001':

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

See Also `mvl2dec`

Purpose Start MATLAB server component of Link for Cadence Incisive software

Syntax

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

Description **Server Activation**

hdldaemon starts the Link for Cadence Incisive MATLAB server component with the following default settings:

- Shared memory communication enabled
- Time resolution for the MATLAB simulation function output ports set to scaled (type double)

Although you can use TCP/IP on a single system (one that is running both MATLAB and the Incisive simulator), using shared memory communication when your application configuration consists of a single system can result in increased performance.

Only one hlddaemon can be running at any given time.

Matching Communication Modes and Socket Ports

The communication mode that you specify (shared memory or TCP/IP sockets) must match what you specify for the communication mode when you issue the `matlabtb`, `matlabtbeval`, or `matlabcp` command in the Incisive simulator.

In addition, if you specify TCP/IP socket mode, you must also identify a socket port to be used for establishing links. You can choose and specify a socket port yourself, or you can use an option that instructs the operating system to identify an available socket port for you. Regardless of how the socket port is identified, the socket you specify with the Incisive simulator must match the socket being used by the server.

hdldaemon

For more information on modes of communication, see “Modes of Communication” on page 1-9. For more information on establishing the Incisive simulator end of the communication link, see “Initializing the HDL Simulator for a MATLAB Link Session” on page 3-13.

`hdldaemon('PropertyName', 'PropertyValue'...)` starts the Link for Cadence Incisive MATLAB server component with property-value pair settings that specify the communication mode for the link between MATLAB and the Incisive simulator, the time resolution for the MATLAB simulation function output ports, and, optionally, a Tcl command to be executed immediately in the HDL simulator. See “Property Name/Property Value Pairs” on page 5-6 for details.

Link Status

`hdldaemon('status')` returns the following message indicating that a link (connection) exists between MATLAB and the Incisive simulator:

```
HLDaemon 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 the interprocess communication identifier (`ipc_id`) being used for a link by assigning the return value of `hdldaemon` to a variable. The `ipc_id` identifies a port number for TCP/IP socket links or the file system name for a shared memory communication channel. For example:

```
x=hlddaemon('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 Incisive simulator clients. If a shared memory link is in use, the value of `comm` is 'shared memory' and the value of `ipc_id` is a file system name for the shared memory communication channel.

Server Shutdown

`hdldaemon('kill')` shuts down the MATLAB server without shutting down MATLAB.

hdldaemon

Property Name/Property Value Pairs

The following property name/property value pairs are valid for hdldaemon:

'socket', tcp_spec

Specifies the TCP/IP socket mode of communication for the link between MATLAB and the Incisive simulator. If you omit this argument, the server uses the shared memory mode of communication.

Note You *must* use TCP/IP socket communication when your application configuration consists of multiple computing systems.

The tcp_spec can be a TCP/IP port number, TCP/IP port alias or service name, or the value zero, indicating that the port is to be assigned by the operating system. Some valid tcp_spec examples follow:

Option	Examples
Port number	'4449' or 4449
Alias or service name	'MATLAB Service'
Operating system assigned	'0' or 0

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

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

`'time', 'sec' | 'time', 'int64'`

Specifies the time resolution for MATLAB function output ports and simulation times (`tnow`).

Specify...

`'time' 'sec'`
(default)

`'time' 'int64'`

For...

A double value that is scaled to seconds based on the current Incisive simulation resolution

64-bit integer representing the number of simulation steps

If you omit this argument, the server uses scaled resolution time.

`'tclcmd', 'command'`

Passes a Tcl command string, to be executed immediately in the Incisive simulator, from MATLAB to the Incisive simulator. You may use a compound command and separate the commands with semicolons.

Note The Tcl command string you specify cannot include commands that load an Incisive simulator project or modify simulator state. For example, the string cannot include commands such as `run`, `stop`, or `reset`.

Examples

If Your Application Is to...

Operate in shared memory mode

Operate in TCP/IP socket mode, using a specific TCP/IP socket port

Operate in TCP/IP socket mode, using a TCP/IP socket that the operating system identifies as available

Return time values in seconds (type double)

Return 64-bit time values (type int64)

Execute Tcl command immediately upon simulator connection

Do the Following...

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.

Specify the 'socket', *tcp_spec* property name and value pair. The *tcp_spec* can be a socket port number or service name. Examples of valid port specifications include '4449', 4449, and MATLAB Service. For information on choosing a TCP/IP socket port, see “Choosing TCP/IP Socket Ports” on page 1-18.

Specify 'socket', 0 or 'socket', '0'.

Specify 'time', 'sec' or omit the parameter. This is the default time value resolution.

Specify 'time', 'int64'.

Specify the 'tclcmd', 'command' property name and value pair. Command must be a valid Tcl command but cannot include commands that load an Incisive simulator project or modify the simulator state.

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 Incisive simulator prompt:

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

The following is an example of a compound Tcl command used with `hdldaemon`:

```
hdldaemon('tclcmd','{force filter2d_v.clk_enable 1
-after 0ns;
force filter2d_v.reset 1 -after 0 ns 0 -after 1 ns;
puts {Running Simulink Cosimulation block};
puts [clock format [clock seconds]]}')'
```

mvl2dec

Purpose Convert multivalued logic to decimal

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

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

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

Examples The following function call returns the decimal value 23:

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

The following function call returns NaN:

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

The following function call returns the decimal value -9:

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

See Also `dec2mvl`

Purpose Start and configure Incisive simulators for use with Link for Cadence Incisive software

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

Description `nclaunch('PropertyName', 'PropertyValue'...)` starts the Incisive simulator for use with the MATLAB and Simulink features of the Link for Cadence Incisive software. The initial directory in the Incisive simulator matches your MATLAB current directory if no explicit `rundir` parameter is specified.

After you call this function, you can use HDL Simulator Tcl Commands to do interactive debug setup.

The property name/property value pair settings allow you to customize the Tcl commands used to start the Incisive simulator, the `ncsim` executable to be used, the path and name of the Tcl script that stores the start commands, and for Simulink applications, details about the mode of communication to be used by the applications. You must use a property name/property value pair with `nclaunch`.

Property Name/Property Value Pairs

`'hdlsimdir', 'pathname'`

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

`'hdlsimexe', 'simexename'`

Specifies the name of an Incisive simulator executable. By default, this function uses `'ncsim'`. You can specify a custom-built simulator executable with `'simexename.'`

`'libdir', 'directory'`

Specifies the directory containing MATLAB shared libraries. This property creates an entry in the startup Tcl file that points to the directory with the shared libraries needed for the Incisive

simulator to communicate with MATLAB when the Incisive simulator is running on a machine that does not have MATLAB.

'rundir', 'dirname'

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

- If `dirname` is specified and the directory exists, the HDL simulator is run in the specified directory.
- If no `rundir` property/value pair is specified or if `dirname` is empty, the HDL simulator is run in the current working directory.
- If the value of `dirname` is “TEMPDIR”, the function creates a temporary directory in which it runs the HDL simulator.
- If `dirname` is specified and the directory does *not* exist, you will get an error.

'startupfile', 'pathname'

Specifies a Tcl script that defines the behavior of the Incisive simulator commands `hdlsimmatlab` and `hdlsimulink`. The Tcl script consists of some general-purpose Tcl commands for launching the Incisive simulator and any commands you specify with the 'tclstart' property. If you omit this property, the function creates a temporary file each time the Incisive simulator starts. If you specify a name for the Tcl script, later you can use the file to start the Incisive simulator from a system shell as shown in the following syntax:

```
tclsh tcl_scriptname
```

'socketsimulink', 'tcp_spec'

Specifies TCP/IP socket communication for links between the Incisive simulator and Simulink. For TCP/IP socket communication on a single computing system, the `tcp_spec` can consist of just a TCP/IP port number or service name. If you are setting up communication between computing systems, you must

also specify the name or Internet address of the remote host. The following table lists different ways of specifying `tcp_spec`.

Format	Example
<code><port-num></code>	4449
<code><port-alias></code>	matlabservice
<code><port-num>@<host></code>	4449@compa
<code><host>:<port-num></code>	compa:4449
<code><port-alias>@<host-ia></code>	matlabservice@123.34.55.23

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

If the Incisive simulator 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.

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

Determines whether the Incisive simulator is launched. The default is yes, which launches the Incisive simulator and creates a startup Tcl file. If `starthdlsim` is set to no, the Incisive simulator is not launched, but a startup Tcl file is still created.

This startup Tcl file contains pointers to MATLAB and Simulink shared libraries. To run the Incisive simulator manually, see “Setting Up Link for Cadence Incisive Software for Use with the Incisive Simulator on a Separate Machine from MATLAB” on page 1-24.

`'tclstart', 'tcl_commands'`

Specifies one or more Tcl commands to execute before the Incisive simulator launches. Specify a command string or a cell array

of command strings. You must specify at least one command; otherwise, no action occurs.

Note You must put “exec” in front of non-Tcl system shell commands. For example:

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

Examples

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

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

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

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

The arguments being passed with `input` (`matlabtb` and `run`) are executed in the `ncsim` Tcl shell. In this example, `matlabcp` associates the m-function `vlogmatlabcp` to the module instance `u_matlab_component`.

It assumes that the `hdldaemon` in MATLAB is listening on port 32864. `run` will run 50 resolution units (ticks).

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

The following example demonstrates using the property `startupfile` to designate a Tcl script that is then used to start the HDL simulator from the Tcl shell.

In MATLAB:

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

In Tcl shell:

```
shell> tclsh mytclscript
```


HDL Simulator Tcl Commands — Alphabetical List

hdlsimmatlab

Purpose	Load instantiated HDL design for verification with MATLAB
Syntax	<code>hdlsimmatlab <instance> [<ncsim_args>]</code>
Arguments	<code><instance></code> Specifies the instance of an HDL design to load for verification. <code><ncsim_args></code> Specifies one or more <code>ncsim</code> command arguments. For details, see the description of <code>ncsim</code> in the Incisive simulator documentation.
Description	<p>The <code>hdlsimmatlab</code> command loads the specified instance of an HDL design for verification and sets up the Incisive simulator so it can establish a communication link with MATLAB. The Incisive simulator opens a simulation workspace as it loads the HDL design.</p> <p>This command may be run from the HDL simulator prompt or from a Tcl script shell (<code>tclsh</code>).</p>
Examples	<p>The following command loads the module instance <code>parse</code> from library <code>work</code> for verification and sets up the Incisive simulator so it can establish a communication link with MATLAB:</p> <pre>tclshell> hdlsimmatlab work.parse</pre>

Purpose	Load instantiated HDL design for cosimulation with Simulink
Syntax	<pre>hdlsimulink [<ncsim_args>] <instance> [-socket <tcp_spec>]</pre>
Argument	<p><ncsim_args> Specifies one or more <code>ncsim</code> command arguments. At a minimum, either <code>-gui</code> or <code>-tcl</code> is required. If you specify <code>-gui</code>, the Simulink GUI will be launched when the HDL design is loaded. If you specify <code>-tcl</code>, a Tcl script shell is launched instead. If you do not specify either of these arguments, the HDL simulator runs the simulation without Simulink. Other valid <code>ncsim</code> arguments may be specified in addition to <code>-gui</code> or <code>-tcl</code>. For more information on <code>-gui</code>, <code>-tcl</code>, or other <code>ncsim</code> arguments, see the description of <code>ncsim</code> in the Incisive simulator documentation.</p> <p><instance> Specifies the instance of an HDL design to load for cosimulation.</p> <p>-socket <tcp_spec> Specifies TCP/IP socket communication for the link between the Incisive simulator and MATLAB. This setting overrides the setting specified with the MATLAB <code>nclaunch</code> function. The <code><tcp_spec></code> 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 <code>matlab-service</code>.</p> <p>For more information on choosing TCP/IP socket ports, see “Choosing TCP/IP Socket Ports” on page 1-18.</p> <p>If the Incisive simulator and MATLAB are running on the same computer, you have the option of using shared memory for communication. Shared memory is the default mode of communication and takes effect if you omit <code>-socket <tcp_spec></code> from the command line.</p>

Note The communication mode that you specify with the `hdlsimulink` command must match what you specify for the communication mode when you configure Link for Cadence Incisive blocks in your Simulink model.

For more information on modes of communication, see “Modes of Communication” on page 1-9. For more information on establishing the Simulink end of the communication link, see “Configuring the Communication Link” on page 4-32.

Description

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

Examples

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

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

Purpose Associate MATLAB component function with instantiated HDL design

Syntax

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

Arguments

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

Do not specify an instance of an HDL design that has already been associated with a MATLAB test bench function (via matlabb).

<time-specs>
Specifies a combination of time specifications consisting of any or all of the following:

<code><timen>,...</code>	Specifies one or more discrete time values at which the specified MATLAB function is called. Each time value is relative to the current simulation time. The MATLAB function is always called once at the start of the simulation, even if you do not specify a time.
<code>-repeat <time></code>	Specifies that the MATLAB function be called repeatedly based on the specified <code><timen>,...</code> pattern. The time values are relative to the value of <code>tnow</code> at the time the MATLAB function is initially called.
<code>-cancel <time></code>	Specifies a time at which the specified MATLAB function stops executing. The time value is relative to the value of <code>tnow</code> at the time the MATLAB function is initially called. If you do not specify a cancel time, the command calls the MATLAB function.

Note Time specifications must be placed after the `matlabcp` instance and before any additional command arguments; otherwise the time specifications are ignored.

`-socket <tcp_spec>`
Specifies TCP/IP socket communication for the link between the Incisive simulator and MATLAB. For TCP/IP socket communication on a single computer, the `<tcp_spec>` can consist of just a TCP/IP port number or service name (alias). If you are setting up communication between computers, you must also specify the name or Internet address of the remote host that is

running the MATLAB server (hdldaemon). The following table lists different ways of specifying <tcp_spec>.

Format	Example
<port-num>	4449
<port-alias>	matlabservice
<port-num>@<host>	4449@compa
<host>:<port-num>	compa:4449
<port-alias>@<host-ia>	matlabservice@123.34.55.23

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

If the Incisive simulator and MATLAB are running on the same computer, you have the option of using shared memory for communication. Shared memory is the default mode of communication and takes effect if you omit -socket <tcp_spec> from the command line.

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

For more information on modes of communication, see “Modes of Communication” on page 1-9. For more information on establishing the MATLAB end of the communication link, see “Starting the MATLAB Server” on page 3-7.

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

Indicates that the specified MATLAB function is called on the rising edge (transition from '0' to '1') of any of the specified

signals. Specify `-rising` with the pathnames of one or more signals defined as a logic type.

`-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.

`-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 module you specify for instance. If you omit this argument, `matlabcp` attaches the module to a MATLAB function that has the same name as the module. For example, if the module is `myfirfilter`, `matlabcp` associates the module with the MATLAB function `myfirfilter`. If you omit this argument and `matlabcp` does not find a MATLAB function with the same name, the command generates an error message.

Description

The `matlabcp` command has the following characteristics:

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

the command `matlabcp` for instance `foo`, all previously scheduled events initiated by `matlabcp` on `foo` are canceled.

Note For the Incisive simulator to establish a communication link with MATLAB, the MATLAB server, `hdldaemon`, must be running with the same communication mode and, if appropriate, the same TCP/IP socket port as you specify with the `matlabcp` command.

MATLAB component functions simulate the behavior of the HDL model. A stub entity or module (providing port definitions only) in the HDL design 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 a MATLAB Component Function” on page 2-33.

Examples

The following command starts the Incisive simulator client component of the Link for Cadence Incisive software. The `-mfunc` option specifies the m-function to connect to and `-socket` option specifies the port number for socket connection mode.

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

matlabtb

Purpose Initiate MATLAB test bench session for instantiated HDL design

Syntax

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

Arguments <instance>
Specifies the instance of an HDL design 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.

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

<time-specs>
Specifies a combination of time specifications consisting of any or all of the following:

<code><timen>,...</code>	Specifies one or more discrete time values at which the specified MATLAB function is called. Each time value is relative to the current simulation time. Even if you do not specify a time, the command calls the MATLAB function once at the start of the simulation.
<code>-repeat <time></code>	Specifies that the MATLAB function be called repeatedly based on the specified <code><timen>,...</code> pattern. The time values are relative to the value of <code>tnow</code> at the time the MATLAB function is initially called.
<code>-cancel <time></code>	Specifies a time at which the specified MATLAB function stops executing. The time value is relative to the value of <code>tnow</code> at the time the MATLAB function is initially called. If you do not specify a cancel time, the command calls the MATLAB function.

Note Time specifications must be placed after the `matlabtb` instance and before any additional command arguments; otherwise the time specifications are ignored.

`-socket <tcp_spec>`
Specifies TCP/IP socket communication for the link between the Incisive simulator and MATLAB. For TCP/IP socket communication on a single computer, the `<tcp_spec>` can consist of just a TCP/IP port number or service name (alias). If you are setting up communication between computers, you must also specify the name or Internet address of the remote host that is

running the MATLAB server (hd1daemon). The following table lists different ways of specifying <tcp_spec>.

Format	Example
<port-num>	4449
<port-alias>	matlabservice
<port-num>@<host>	4449@compa
<host>:<port-num>	compa:4449
<port-alias>@<host-ia>	matlabservice@123.34.55.23

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

If the Incisive simulator and MATLAB are running on the same computer, you have the option of using shared memory for communication. Shared memory is the default mode of communication and takes effect if you omit -socket <tcp_spec> from the command line.

Note The communication mode that you specify with the matlabtb command must match what you specify for the communication mode when you issue the hd1daemon command in MATLAB.

For more information on modes of communication, see “Modes of Communication” on page 1-9. For more information on establishing the MATLAB end of the communication link, see “Starting the MATLAB Server” on page 3-7.

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

Indicates that the specified MATLAB function is called on the rising edge (transition from '0' to '1') of any of the specified

signals. Specify `-rising` with the pathnames of one or more signals defined as a logic type.

`-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.

`-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 HDL design.

`-mfunc <name>`

The name of the MATLAB function that is attached to the module you specify for `instance`. If you omit this argument, `matlabtb` attaches the module to a MATLAB function that has the same name as the module. For example, if the module is `myfirfilter`, `matlabtb` associates the module with the MATLAB function `myfirfilter`. If you omit this argument and `matlabtb` does not find a MATLAB function with the same name, the command generates an error message.

Description

The `matlabtb` command has the following characteristics:

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

Note For the Incisive simulator to establish a communication link with MATLAB, the MATLAB server, `hdldaemon`, must be running with the same communication mode and, if appropriate, the same TCP/IP socket port as you specify with the `matlabtb` command.

Examples

The following command starts the Incisive simulator client component of the Link for Cadence Incisive software, associates an instance of the module `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:

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

The following command starts the Incisive simulator client component of the Link for Cadence Incisive software, and initiates a remote TCP/IP socket-based session using remote MATLAB host `compa` 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.

```
ncsim> matlabtb myfirfilter 10 ns -rising top.fclk  
-sensitivity top.din -socket 4449@compa
```

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

```
ncsim>matlabtb osc_top -sensitivity osc_top.sine_out  
-socket 4448 -mfunc hosctb
```

Purpose	Call specified MATLAB function for immediate execution on behalf of instantiated HDL design
Syntax	matlabtbeval <instance> [-socket <tcp_spec>] [-mfunc <name>]
Arguments	<p><instance> Specifies the instance of an HDL design 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.</p> <p>-socket <tcp_spec> Specifies TCP/IP socket communication for the link between the Incisive simulator and MATLAB. For TCP/IP socket communication on a single computer, the <tcp_spec> can consist of just a TCP/IP port number or service name (alias). If you are setting up communication between computers, you must also specify the name or Internet address of the remote host. The following table lists different ways of specifying <tcp_spec>.</p>

Format

<port-num>

<port-alias>

<port-num>@<host>

<host>:<port-num>

<port-alias>@<host-ia>

Example

4449 on this computer

matlabservice on this
computer

4449@compa

compa:4449

matlabservice@123.34.55.23

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

If the Incisive simulator and MATLAB are running on the same computer, you have the option of using shared memory for communication. Shared memory is the default mode of communication and takes effect if you omit `-socket <tcp-spec>` from the command line.

Note The communication mode that you specify with the `matlabtbeval` command must match what you specify for the communication mode when you call the `hdldaemon` command to start the MATLAB server.

For more information on modes of communication, see “Modes of Communication” on page 1-9. For more information on establishing the MATLAB end of the communication link, see “Starting the MATLAB Server” on page 3-7.

`-mfunc <name>`

The name of the MATLAB function that is attached to the module you specify for instance. If you omit this argument, `matlabtbeval` attaches the module to a MATLAB function that has the same name as the module. For example, if the module is `myfirfilter`, `matlabtbeval` associates the module with the MATLAB function `myfirfilter`. If you omit this argument and `matlabtbeval` does not find a MATLAB function with the same name, the command displays an error message.

Description

The `matlabtbeval` command has the following characteristics:

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

Note For the Incisive simulator to establish a communication link with MATLAB, the MATLAB `hdldaemon` must be running with the same communication mode and, if appropriate, the same TCP/IP socket port as you specify with the `matlabtbeval` command.

Examples

The following command starts the Incisive simulator client component of the Link for Cadence Incisive software, associates an instance of the module `myfirfilter` with the function `myfirfilter.m`, and uses a local TCP/IP socket-based communication link to TCP/IP port 4449 to execute the function `myfirfilter.m`:

```
ncsim> matlabtbeval myfirfilter -socket 4449
```

The following command starts the Incisive simulator client component of the Link for Cadence Incisive software, associates an instance of the module `filter` with the function `myfirfilter.m`, and uses a remote TCP/IP socket-based communication link to host `compa` and TCP/IP port 4449 to execute the function `myfirfilter.m`

```
ncsim> matlabtbeval myfirfilter -socket 4449@compa
```

nomatlabtb

Purpose	Terminate active MATLAB test bench and MATLAB component sessions
Syntax	nomatlabtb
Description	The nomatlabtb command terminates all active MATLAB test bench and MATLAB component sessions that were previously initiated by matlabtb or matlabcp commands.
Examples	The following command terminates all MATLAB test bench and MATLAB component sessions: <pre>ncsim> nomatlabtb</pre>
See Also	matlabcp, matlabtb

Simulink Blocks — Alphabetical List

HDL Cosimulation

Purpose

Cosimulate a hardware component by communicating with an HDL model executing in Incisive simulator

Library

[Link for Cadence Incisive](#)

Description



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

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

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

The **Ports** pane provides fields for mapping signals of your HDL design to input and output ports in your block. The signals can be at any level of the HDL design hierarchy. Simulink deposits an input port signal on an Incisive simulator signal at the signal's sample rate. Conversely, Simulink reads an output port signal from a specified Incisive simulator signal at the specified sample rate.

In general, Simulink handles port sample periods as follows:

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

In addition to specifying output port sample times, you can force the fixed point data types on output ports. For example, setting the **Data Type** property of an 8-bit output port to Signed and setting its **Fraction Length** property to 5 would force the data type to `sfixed8_E5`.

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

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

Input/output ports can be used here as well; specify port as both input and output.

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

HDL Cosimulation

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

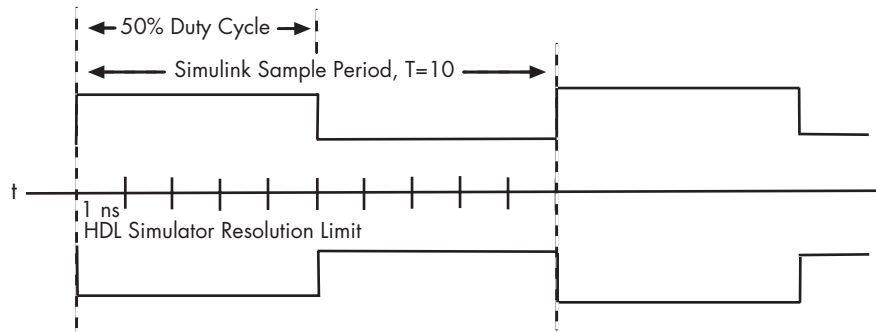
- If you specify an explicit clock period, you must enter a sample time equal to or greater than 2 resolution units (ticks).
- If the clock period (whether explicitly specified or defaulted) is not an even integer, Simulink cannot create a 50% duty cycle, and therefore the Link for Cadence Incisive interface creates the falling edge at

$$\text{clockperiod} / 2$$

(rounded down to the nearest integer).

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

Rising Edge Clock



Falling Edge Clock

For more information on calculating relative and absolute timing modes, see “Defining the Simulink and HDL Simulator Timing Relationship” on page 4-9 in Chapter 4, “Modeling and Verifying an HDL Design with Simulink”.

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

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

Dialog Box

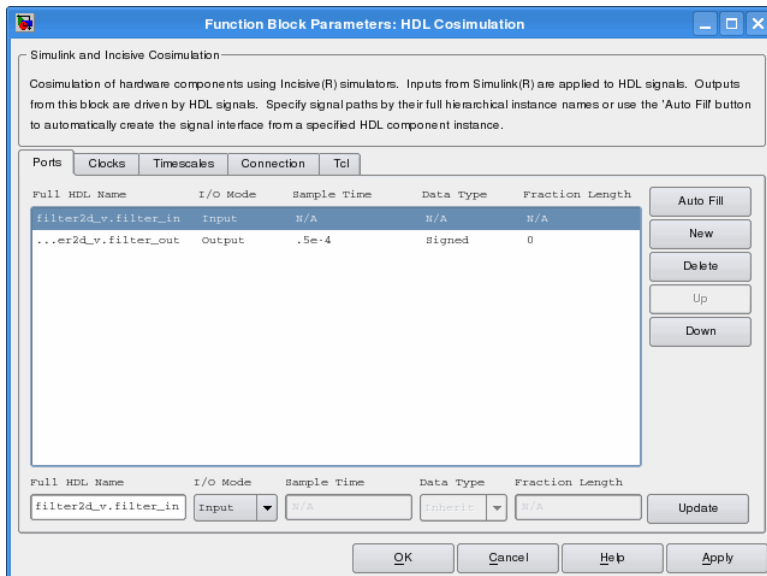
The Block Parameters dialog box consists of four tabbed panes of configuration options:

- “Ports Pane” on page 7-6

HDL Cosimulation

- “Connection Pane” on page 7-12
- “Timescales Pane” on page 7-15
- “Clocks Pane” on page 7-18
- “Tcl Pane” on page 7-20

Ports Pane



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

Maintain this list with the buttons on the right of the pane:

- **Auto Fill** — Transmit a port information request to the Incisive simulator. The port information request returns port names and information from an HDL model under simulation in the Incisive simulator, and automatically enters this information into the ports list. See “Obtaining Signal Information Automatically from the

Incisive Simulator” on page 4-27 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.
- **Update** — Update the displayed values in the list for the selected signal. Note that this affects only the signal list. To commit edits to the Simulink model, you must also click **Apply**.

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

Full HDL Name

Specifies the signal pathname, using the Incisive simulator 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 HDL Cosimulation block port corresponding to the signal is labeled with the **Full HDL Name**.

Specifying port/signal and module paths. These rules are for signal/port and module path specifications for Simulink. Other specifications may work but are not guaranteed to work in this or future releases. For MATLAB path specifications, see “Specifying Port/Signal and Module Paths” on page 2-4.

- If the top level is Verilog:
 - Path specification must start with a top-level module name.
 - Path specification can include ".", "/", or ":" as path delimiters, but cannot include a mixture.
 - The leaf module or signal may be either VHDL or Verilog.

HDL Cosimulation

The following are valid signal and module path specification examples:

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

- If the top level is VHDL:
 - Path specification can include the top-level module or begin with ":" to represent the top-level module.
 - Path specification can include ".", "/", or ":" as path delimiters, but cannot include a mixture.
 - The leaf module or signal may be either VHDL or Verilog.

The following are valid signal and module path specification examples:

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

The following are invalid signal and module path specification examples:

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

Note You can copy signal pathnames directly from the HDL simulator **wave** window and paste them into the **Full HDL Name** field, using the standard copy and paste commands in the Incisive simulator and Simulink (as long as you use the 'Path.Name' view and not 'Db::Path.Name' view). After pasting a signal pathname into the **Full HDL Name** field, you must click the **Update** button to complete the paste operation and update the signal list.

I/O Mode

Select either Input, Output, or both.

Input designates signals of your HDL model that are to be driven by Simulink. Simulink deposits values on the specified the Incisive simulator signal at the signal's sample rate.

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

Output designates signals of your HDL 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 Date Type and Fraction Length in a following section).

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.

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 7-15). See also “Representation of Simulation Time” on page 4-8.

Data Type

Fraction Length

These two related parameters apply only to output signals.

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

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

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

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

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

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

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

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

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

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

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

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

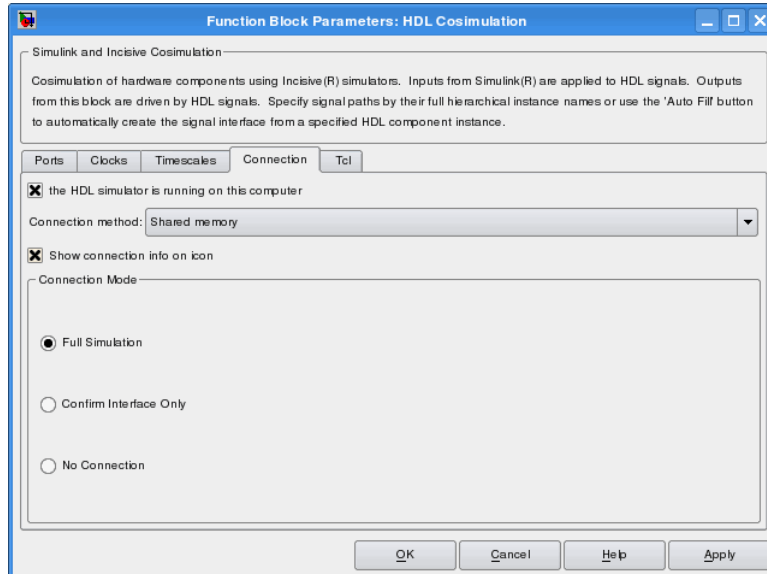
- Select **Unsigned** from the **Data Type** list if you want to explicitly assign an unsigned fixed point data type. When **Unsigned** is selected, the **Fraction Length** edit field is enabled. The port is assigned a fixed point type `ufixN_EnF`, where N is the signal width and F is the **Fraction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

HDL Cosimulation

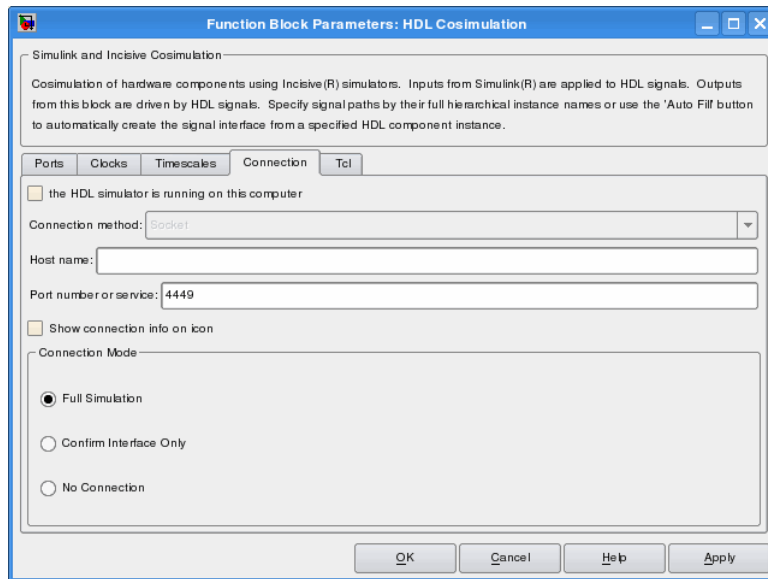
For example, if you specify **Data Type** as Unsigned and a **Fraction Length** of 5 for a 16-bit signal, Simulink forces the data type to `ufix16_En5`. For the same signal with a **Data Type** set to Unsigned and **Fraction Length** of -5, Simulink forces the data type to `ufix16_E5`.

Connection Pane

This figure shows the default configuration of the **Connection** pane. By default, the block is configured for shared memory communication between Simulink and the Incisive simulator, running on a single computer.



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



the HDL Simulator is running on this computer

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

Connection method

This list is enabled when **the HDL Simulator is running on this computer** is selected. Select Socket if you want Simulink and the Incisive simulator to communicate via a designated TCP/IP socket. Select Shared memory if you want Simulink and the Incisive simulator to communicate via shared memory. For more information on these connection methods, see “Configuring the Communication Link” on page 4-32.

Host name

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

Port number or service

Indicate a valid TCP socket port number or service for your computer system (if not using shared memory). For information on choosing TCP socket ports, see “Choosing TCP/IP Socket Ports” on page 1-18.

Show connection info on icon

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

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

Connection Mode

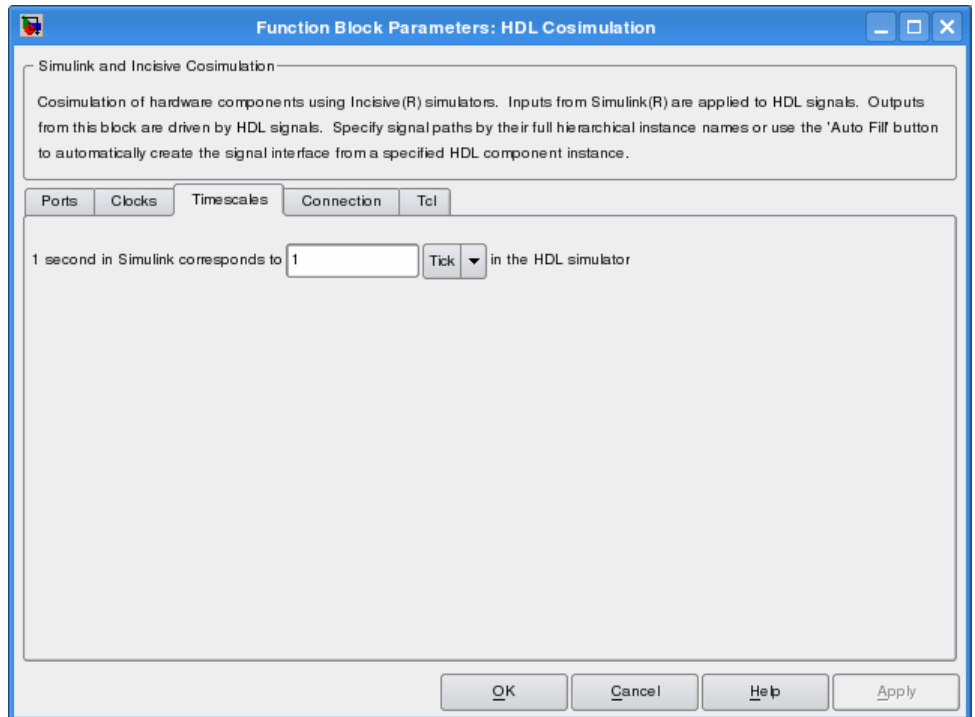
If you want to bypass the HDL simulator when running a Simulink simulation, use these options to specify what type of simulation connection you want. Select one of the following:

- **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, the Link for Cadence Incisive interface does not communicate with the HDL simulator during Simulink simulation.

Timescales Pane

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

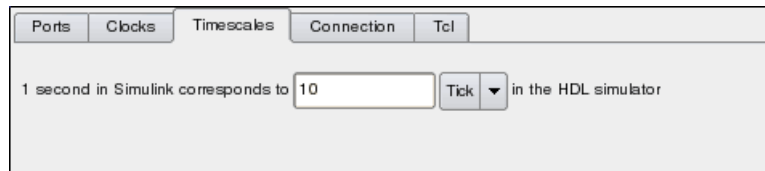


The **Timescales** pane specifies a correspondence between one second of Simulink time and some quantity of Incisive simulator time. This quantity of Incisive simulator time can be expressed in one of the following ways:

HDL Cosimulation

- In *relative* terms (i.e., as some number of Incisive 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 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 Incisive simulator ticks to 1 Simulink second.



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

To use absolute mode, select a unit of absolute time (available units are fs, ps, ns, us, ms, s) from the list on the right. Then enter a scale factor in the left-side edit box. For example, in the figure below the **Timescales** pane is configured for an absolute timing correspondence of 1 Incisive simulator 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 Cadence Incisive software 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 the Link for Cadence Incisive software creates the falling edge at

$$\text{clockperiod} / 2$$

(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

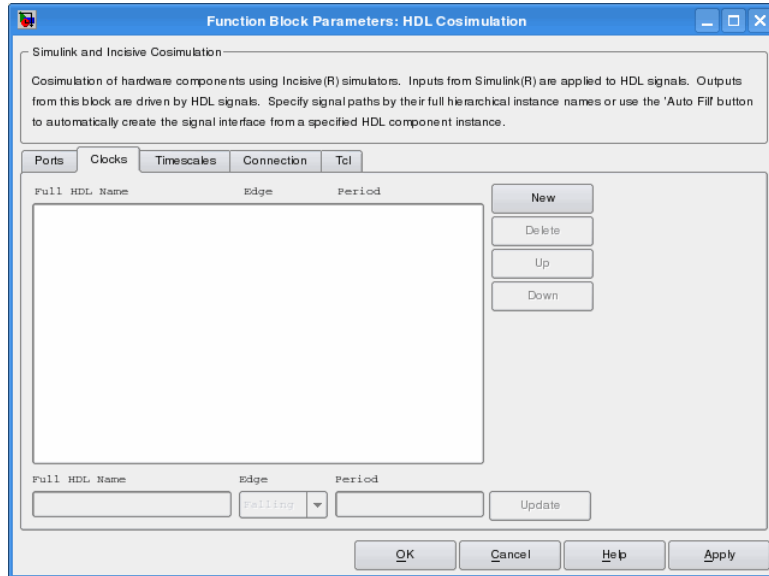
1 Simulink seconds = 3 HDL simulator seconds
1 Simulink second = 3x108 HDL simulator ticks
```

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 more information on calculating relative and absolute timing modes, see “Defining the Simulink and HDL Simulator Timing Relationship” on page 4-9 in Chapter 4, “Modeling and Verifying an HDL Design with Simulink”.

For detailed information on the relationship between Simulink and the Incisive simulator during cosimulation, and on the operation of relative and absolute timing modes, see “Representation of Simulation Time” on page 4-8 in Chapter 4, “Modeling and Verifying an HDL Design with Simulink”.

Clocks Pane



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

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

- **New** — Add a new clock signal to the list and select it for editing.
- **Delete** — Remove a clock signal from the list.
- **Up** — Move the selected clock signal up one position in the list.
- **Down** — Move the selected clock signal down one position in the list.
- **Update** — Update the displayed values in the list for the selected clock signal. Note that this affects only the signal list. To commit edits to the Simulink model, you must also click **Apply**.

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

Full HDL Name

Specify each clock as a signal pathname, using the Incisive simulator 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 7-6.

Note You can copy signal pathnames directly from the HDL simulator **wave** window and paste them into the **Full HDL Name** field, using the standard copy and paste commands in the Incisive simulator and Simulink (as long as you use the 'Path.Name' view and not 'Db::Path.Name' view). After pasting a signal pathname into the **Full HDL Name** field, you must click the **Update** button to complete the paste operation and update the signal list.

Edge

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

Period

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

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,

HDL Cosimulation

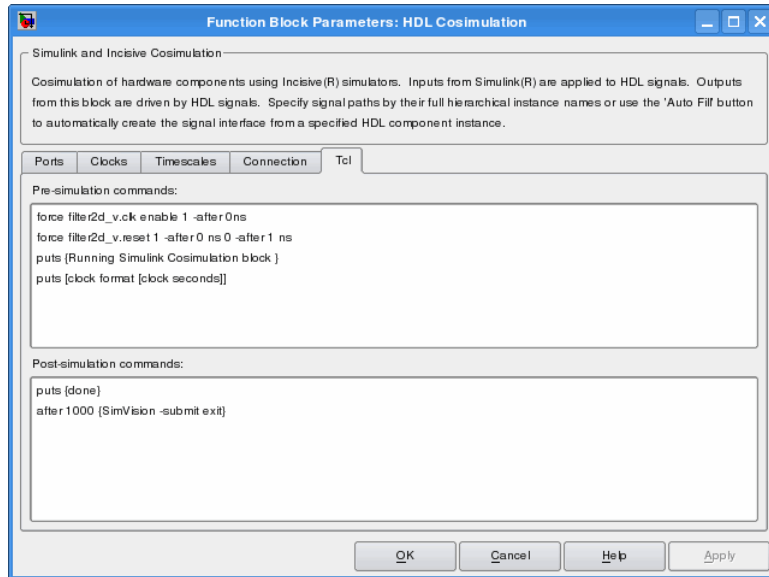
and therefore the Link for Cadence Incisive software creates the falling edge at

$$\text{clockperiod} / 2$$

(rounded down to the nearest integer).

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

Tcl Pane



Pre-simulation commands

Tcl commands to be executed before the Incisive simulator simulates the HDL component of your Simulink model. You can specify one Tcl command per line in the text box, or enter multiple

commands per line by appending each command with a semicolon (;), the standard Tcl concatenation operator.

Alternatively, you can create an Incisive simulator Tcl script that lists Tcl commands and then specify that file with the Incisive simulator source command as follows:

```
source mycosimstartup.script_extension
```

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 Tcl script that you specify for this parameter cannot include commands that load an Incisive simulator project or modify simulator state. For example, they cannot include commands such as run, stop, or reset.

Post-simulation commands

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

Alternatively, you can create an Incisive simulator Tcl script that lists Tcl commands and then specify that file with the Incisive simulator source command as follows:

```
source mycosimcleanup.script_extension
```

Notes

- You can include the `exit` command in an after simulation Tcl script to force the Incisive simulator 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 `exit` at the end of the command string or Tcl script.

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

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

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

```
after 1000 {exit}
```

- With the exception of `exit`, the command string or Tcl script that you specify cannot include commands that load an Incisive simulator project or modify simulator state. For example, they cannot include commands such as `run`, `stop`, or `reset`.
-

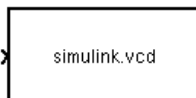
Purpose

Generate a value change dump (VCD) file

Library

Link for Cadence Incisive

Description



To VCD File

The To VCD File block generates a VCD file that contains information about changes to signals connected to the block's input ports and names the file with the specified file name. 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. Examples of postprocessing include the extraction of data pertaining to a particular section of a design hierarchy or data generated during a specific time interval.

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

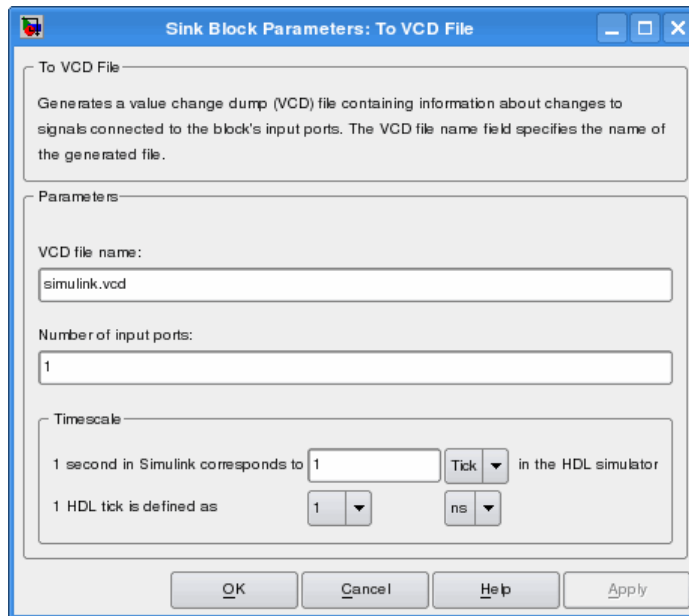
- The file name to be used for the generated file
- The number of block input ports that are to receive signal data

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 4-47.

To VCD File

Dialog Box



VCD file name

The file name to be used for the generated VCD file. If you specify a file name only, Simulink places the file in your current MATLAB directory. Specify a complete 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.

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

Caution 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.

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 the HDL simulator.

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

- In *relative* terms (i.e., as some number of Incisive 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

To VCD File

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** .

A

- Absolute timing mode 4-12
- addresses, Internet 1-18
- application software 1-20
- application specific integrated circuits (ASICs) 1-2
- applications 1-3
 - coding Link for Cadence Incisive software overview of 2-2
 - programming Link for Cadence Incisive software overview of 2-2
- arguments
 - for hdlstimmatlab command 6-2
 - for hdlstimulink command 6-3
 - for matlabcp command 6-5
 - for matlabbtb command 6-10
 - for matlabbtbeval command 6-15
- array indexing
 - differences between MATLAB and VHDL 2-11
- arrays
 - converting to 2-21
 - indexing elements of 2-11
 - of VHDL data types 2-6
- ASICs (application specific integrated circuits) 1-2
- Auto fill
 - in Ports pane of HDL Cosimulation block 7-2
 - using in Ports pane 4-23

B

- behavioral model 1-3
- BIT data type 2-6
 - conversion of 2-11
- bit vector
 - converting for MATLAB 2-20
- BIT_VECTOR data type 2-6
 - conversion of 2-11

- block input ports parameter
 - description of 7-2
 - mapping signals with 4-23
- block latency 4-16
- block library
 - description of 4-19
 - Link for Cadence Incisive software 1-5
- block output ports parameter
 - description of 7-2
 - mapping signals with 4-23
- block Parameters dialog
 - for HDL Cosimulation block 4-23
- Block Parameters dialog
 - for To VCD File block 4-44
- block ports
 - mapping signals to 4-23
 - requirements for HDL Cosimulation blocks 4-20
- blocks
 - HDL Cosimulation
 - applying configuration settings for 4-40
 - configuring 4-20
 - description of 7-2
 - To VCD File
 - configuring 4-44
 - description of 7-23
 - generating VCD files with 4-44
- blocksets
 - for creating hardware models 4-5
 - for EDA applications 4-5
 - installing 1-22
- breakpoints 3-20

C

- callback specification 2-16
- callback timing 3-12
 - cancel option 6-10
- CHARACTER data type 2-6
 - conversion of 2-11

- checklists
 - environment requirements 1-13
 - HDL Cosimulation block requirements 4-20
- client
 - for MATLAB and HDL simulator links 1-7
 - for Simulink and HDL simulator links 1-8
- client/server environment 1-7
- clocks
 - requirements for HDL Cosimulation blocks 4-20
 - specifying for HDL Cosimulation blocks 4-34
- Clocks pane
 - configuring block clocks with 4-34
 - description of 7-2
- column-major numbering 2-11
- comm status field
 - checking with `hdldaemon` function 3-5
 - description of 5-3
- commands, HDL simulator 6-1
 - See also* HDL simulator commands
- commands, Incisive simulator 6-1
- communication
 - configuring for blocks 4-32
 - features 1-5
 - initializing for HDL simulator and MATLAB session 3-13
 - modes of 1-9
 - requirements for HDL Cosimulation blocks 4-20
 - socket ports for 1-18
- communication channel
 - checking identifier for 3-5
- communication modes
 - checking 3-5
 - specifying for HDL Cosimulation block 4-20
 - specifying with `hdldaemon` function 3-7
- Communications Blockset
 - as optional software 1-20
 - using for EDA applications 4-5
- components 1-5
- composite data types
 - conversions of 2-11
 - VHDL 2-6
- configurations
 - deciding on 1-15
 - multiple-link 1-15
 - single-system 1-15
 - valid for MATLAB and the HDL simulator 1-16
 - valid for Simulink and the HDL simulator 1-17
- Connection pane
 - configuring block communication with 4-32
 - description of 7-2
- connections status field
 - checking with `hdldaemon` function 3-5
 - description of 5-3
- connections, link
 - checking number of 3-5
 - TCP/IP socket 1-18
- Continue button, MATLAB 3-20
- Continue option 3-20
- continuous signals 4-8
- cosimulation 1-5
 - configuring a HDL Cosimulation block for 4-20
 - controlling MATLAB 3-1
 - overview of 3-3
 - logging changes to signal values during 4-44
 - requirements for 4-20
 - starting MATLAB 3-1
 - overview of 3-3
 - starting with Simulink 4-41
- cosimulation block 4-20
 - See also* HDL Cosimulation block
- cosimulation environment 1-7
- Cosimulation timing
 - absolute mode 7-2
 - relative mode 7-2

D

data types
 conversions of 2-11
 converting for HDL simulator 2-21
 converting for MATLAB 2-20
 unsupported VHDL 2-6
 VHDL port 2-6

dbstop function 3-20

dec2mvl function
 description of 5-2

delta time 4-16

demos 1-28

deposit
 changing signals with 4-7
 for iport parameter 2-16
 with force commands 3-18

design process, hardware 1-3

dialogs
 for HDL Cosimulation block 7-2
 for To VCD File block 7-23

discrete blocks 4-8

do command 4-37

DO files
 specifying for HDL Cosimulation blocks 4-37

documentation overview 1-27

double values
 as representation of time 3-12
 converting for HDL simulator 2-21
 converting for MATLAB 2-20

dspstartup M-file 4-18

duty cycle 4-34

E

EDA (Electronic Design Automation) 1-2

Electronic Design Automation (EDA) 1-2

End Simulation option, HDL simulator 3-22

entities
 coding for MATLAB verification 2-3
 naming 2-4

sample definition of 2-8
 specifying ports for 2-6

enumerated data types 2-6
 conversion of 2-11
 converting to 2-21

environment requirements 1-13

environment, cosimulation 1-7

examples 4-5
 dec2mvl function 5-2
 hdldaemon function 5-3
 hdlstimatlab command 6-2
 hdlstimulink command 6-3
 matlabcbp command 6-5
 matlabtb command 6-10
 matlabtbeval command 6-15
 mvl2dec function 5-10
 nclaunch function 5-11
 nomatlabtb command 6-18
 test bench function 2-26
See also Manchester receiver Simulink model

F

-falling option 6-10
 specifying scheduling options with 3-13

falling-edge clocks
 creating for HDL Cosimulation blocks 4-34
 description of 7-2
 specifying as scheduling options 3-11
 specifying for HDL Cosimulation block 4-20

Falling-edge clocks parameter
 specifying block clocks with 4-34

features, product 1-5

field programmable gate arrays (FPGAs) 1-2

files
 generating VCD 4-44
 VCD 4-47

force command
 applying simulation stimuli with 3-18

- resetting clocks during cosimulation
 - with 4-41
- FPGAs (field programmable gate arrays) 1-2
- Frame-based processing 4-42
 - in cosimulation 4-42
 - performance improvements gained from 4-42
 - requirements for use of 4-42
 - restrictions on use of 4-42
- functions 5-1
 - resolution 4-7
 - See also* MATLAB functions

G

- Go Until Cursor option, MATLAB 3-20

H

- hardware description language (HDL) 1-2
- hardware design process 1-3
- hardware model design
 - creating in Simulink 4-5
- HDL (hardware description language) 1-2
- HDL Cosimulation block
 - adding to a Simulink model 4-19
 - applying configuration settings for 4-40
 - black boxes representing 4-5
 - configuration requirements for 1-15
 - configuring 4-20
 - configuring clocks for 4-34
 - configuring communication for 4-32
 - configuring ports for 4-23
 - configuring Tcl commands for 4-37
 - description of 7-2
 - design decisions for 4-5
 - handling of signal values for 4-7
 - in Link for Cadence Incisive simulation
 - environment 1-7
 - opening Block Parameters dialog for 4-23
 - scaling simulation time for 4-8

- valid configurations for 1-17
- HDL Cosimulation block output ports 4-28
- HDL design 4-3
- HDL designs
 - using port information for 2-17
 - validating 2-17
- HDL entities or modules
 - specifying ports for 2-6
- HDL models 1-3
 - adding to Simulink models 4-19
 - compiling 2-9
 - configuring Simulink for 4-18
 - cosimulation 1-3
 - porting 4-44
 - running in Simulink 4-41
 - testing in Simulink 4-41
 - verifying 1-3
 - verifying port direction modes for 2-17
 - See also* HDL models
- HDL simulator
 - handling of signal values for 4-7
 - initializing for MATLAB session 3-13
 - quitting 3-22
 - setting up during installation 1-23
 - simulation time for 4-8
 - specifying version of 3-10
 - starting from MATLAB 3-10
 - working with MATLAB links to 1-10
 - working with Simulink links to 1-11
- HDL simulator commands
 - force
 - applying simulation stimuli with 3-18
 - resetting clocks during cosimulation
 - with 4-41
 - hdlsimmatlab
 - description of 6-2
 - matlabtb
 - initializing HDL simulator with 3-13
 - matlabtbeval
 - initializing HDL simulator with 3-13

- specifying scheduling options with 3-11
 - vcd2wlf 4-44
 - HDL support
 - Verilog 1-5
 - VHDL 1-5
 - hdldaemon function
 - checking link status of 3-5
 - configuration restrictions for 1-15
 - description of 5-3
 - starting 3-7
 - hdlsimdir property
 - specifying with nclaunch function 3-10
 - hdlsimmatlab command
 - description of 6-2
 - hdlsimulink command
 - description of 6-3
 - help 1-27
 - Host name parameter
 - description of 7-2
 - specifying block communication with 4-32
 - hostnames
 - identifying Incisive simulator server 4-32
 - identifying MATLAB server 3-13
 - identifying server with 1-17
- I**
- IN direction mode 2-6
 - verifying 2-17
 - Incisive simulator
 - as required software 1-20
 - in Link for Cadence Incisive simulation environment 1-7
 - installing 1-22
 - Incisive simulator commands
 - hdlsimmatlab
 - description of 6-2
 - hdlsimulink
 - description of 6-3
 - matlabcp
 - description of 6-5
 - matlabtb
 - description of 6-10
 - matlabtbeval
 - description of 6-15
 - nomatlabtb 6-18
 - Incisive simulator running on this computer
 - parameter
 - description of 7-2
 - specifying block communication with 4-32
 - inout data type 2-6
 - inout direction mode 2-6
 - INOUT direction mode 2-6
 - verifying 2-17
 - INOUT ports
 - specifying 7-2
 - input 2-6
 - See also* input ports
 - input data type 2-6
 - input direction mode 2-6
 - input ports
 - attaching to signals 4-7
 - for HDL model 2-6
 - for MATLAB component function 2-33
 - for test bench function 2-16
 - mapping signals to 4-23
 - simulation time for 4-8
 - specifying block 4-20
 - installation
 - of Link for Cadence Incisive software 1-23
 - of related software 1-22
 - installation of Link for Cadence Incisive software 1-13
 - integer data type
 - conversion of 2-11
 - INTEGER data type 2-6
 - conversion of 2-11
 - Internet address 1-18
 - identifying server with 1-17

- specifying 3-13
- interprocess communication identifier 3-5
- ipc_id status field
 - checking with hdldaemon function 3-5
 - description of 5-3
- iport parameter 2-16

K

- kill option
 - description of 5-3

L

- latency, block 4-16
- Link for Cadence Incisive software
 - block library 1-5
 - using to add HDL to Simulink with 4-19
 - blocks 1-15
 - definition of 1-2
 - installing 1-23
 - setting up the HDL simulator for 1-23
- link status
 - checking MATLAB server 3-5
 - function for acquiring 5-3
- links
 - MATLAB and the HDL simulator 1-7
 - Simulink and the HDL simulator 1-8

M

- MATLAB
 - as required software 1-20
 - in Link for Cadence Incisive simulation environment 1-7
 - installing 1-22
 - quitting 3-22
 - working with HDL simulator links to 1-10
- MATLAB component functions
 - adding to MATLAB search path 2-35
 - defining 2-33

- specifying required parameters for 2-33
- MATLAB data types
 - conversion of 2-11
- MATLAB functions 5-1
 - coding for HDL verification 2-10
 - dbstop 3-20
 - dec2mvl
 - description of 5-2
 - defining 2-16
 - hdldaemon 3-7
 - description of 5-3
 - mvl2dec
 - description of 5-10
 - naming 2-15
 - nclaunch
 - description of 5-11
 - programming for HDL verification 2-10
 - sample of 2-26
 - scheduling invocation of 3-11
 - specifying required parameters for 2-16
 - test bench 1-7
 - which 2-35
- MATLAB link sessions
 - controlling 3-20
 - overview 3-3
 - logging changes to signal values during 4-44
 - monitoring 3-20
 - scheduling invocation of 3-11
 - starting
 - overview 3-3
 - stopping 3-22
- MATLAB search path 2-35
- MATLAB server
 - checking link status with 3-5
 - configuration restrictions for 1-15
 - configurations for 1-16
 - function for invoking 1-7
 - identifying in a network configuration 1-17
 - starting 3-7
- matlabcp command

- description of 6-5
- matlabtb command
 - description of 6-10
 - initializing HDL simulator for MATLAB session 3-13
 - specifying scheduling options with 3-11
- matlabtbeval command
 - description of 6-15
 - initializing HDL simulator for MATLAB session 3-13
 - specifying scheduling options with 3-11
- mfunc option
 - specifying test bench or component function with 3-13
 - with matlabcp command 6-5
 - with matlabtb command 6-10
 - with matlabtbeval command 6-15
- Mixed-HDL model support
 - limitations 1-5
- models
 - compiling. *See* HDL models
 - getting port information of 2-16
- modes
 - communication 3-7
 - port direction 2-17
- module names
 - specifying paths
 - in MATLAB 2-4
 - in Simulink 7-2
- modules
 - coding for MATLAB verification 2-3
 - naming 2-4
 - specifying ports for 2-6
- multirate signals 4-15
- mv12dec function
 - description of 5-10

N

- names

- for test bench functions 2-15
- for Verilog modules 2-4
- for VHDL entities 2-4
- shared memory communication channel 3-5
- verifying port 2-17
- NATURAL data type 2-6
 - conversion of 2-11
- nclaunch function
 - description of 5-11
 - starting HDL simulator with 3-10
- nclaunchdir property
 - with nclaunch function 5-11
- network configuration 1-17
- network environment 1-7
- nomatlabtb command 6-18
- Number of input ports parameter 7-23
 - configuring To VCD File block with 4-44
- Number of output ports parameter
 - configuring To VCD File block with 4-44
 - description of 7-23
- numeric data
 - converting for HDL simulator 2-21
 - converting for MATLAB 2-20

O

- online help 1-27
- oport parameter 2-16
- options
 - for hdlstimulink command 6-3
 - for matlabcp command 6-5
 - for matlabtb command 6-10
 - for matlabtbeval command 6-15
 - kill 5-3
 - property
 - with hdldaemon function 5-3
 - with nclaunch function 5-11
 - status 5-3

- OS platform requirements. *See* Link for Cadence Incisive product requirements page on The MathWorks web site
- OS platforms. *See* Link for Cadence Incisive product requirements page on The MathWorks web site
- OUT direction mode 2-6
 - verifying 2-17
- output data type 2-6
- output direction mode 2-6
- output ports
 - for HDL model 2-6
 - for MATLAB component function 2-33
 - for test bench function 2-16
 - mapping signals to 4-23
 - simulation time for 4-8
 - specifying block 4-20
- Output sample time parameter
 - description of 7-2
 - specifying sample time with 4-23
- P**
- parameters
 - for HDL Cosimulation block 7-2
 - for To VCD File block 7-23
 - required for MATLAB component functions 2-33
 - required for test bench functions 2-16
- path specification
 - for ports/signals and modules
 - in MATLAB 2-4
 - in Simulink 7-2
- phase, clock 4-34
- platform support 1-5
 - required 1-20
- port names
 - specifying paths
 - in MATLAB 2-4
 - in Simulink 7-2
 - verifying 2-17
- Port number or service parameter
 - description of 7-2
 - specifying block communication with 4-32
- port numbers 1-18
 - checking 3-5
 - specifying for HDL simulator 3-11
 - specifying for MATLAB server 3-7
- portinfo parameter 2-16
- portinfo structure 2-17
- ports
 - getting information about 2-16
 - specifying direction modes for 2-6
 - specifying for HDL entities or modules 2-6
 - specifying VHDL data types for 2-6
 - using information about 2-17
 - verifying data type of 2-17
 - verifying direction modes for 2-17
 - Verilog data types 2-6
- Ports pane
 - Auto fill option 7-2
 - configuring block ports with 4-23
 - description of 7-2
 - using Auto fill 4-23
- ports, block
 - mapping signals to 4-23
 - requirements for 4-20
- Post– simulation command parameter
 - specifying block Tcl commands with 4-37
- postprocessing tools 4-44
- Post–simulation command parameter
 - description of 7-2
- Pre– simulation command parameter
 - specifying block simulation Tcl commands with 4-37
- Pre–simulation command parameter
 - description of 7-2
- properties
 - for `hdldaemon` function 5-3
 - for `nclaunch` function 5-11

- for starting MATLAB server 3-7
- nclaunchdir
 - with nclaunch function 5-11
- socket 5-3
- socketsimulink 5-11
- startupfile 5-11
- tclstart
 - with nclaunch function 5-11
- time
 - description of 5-3
- property option
 - for hdldaemon function 5-3
 - for nclaunch function 5-11

R

- rate converter 4-15
- real data
 - converting for HDL simulator 2-21
 - converting for MATLAB 2-20
- REAL data type 2-6
 - conversion of 2-11
- real values, as time 3-12
- reg data type
 - conversion of 2-11
- Relative timing mode 4-10
- repeat option 6-5
 - specifying scheduling options with 3-13
- requirements
 - application software 1-20
 - checking product 1-20
 - environment 1-13
 - for HDL Cosimulation block 4-20
 - platform 1-20
- resolution functions 4-7
- resolution limit 2-17
- rising option 6-5
 - specifying scheduling options with 3-13
- rising-edge clocks
 - creating for HDL Cosimulation blocks 4-34

- description of 7-2
 - specifying as scheduling options 3-11
 - specifying for HDL Cosimulation block 4-20
- Rising-edge clocks parameter
 - specifying block clocks with 4-34
- run command 3-20
- Run option, MATLAB 3-20

S

- sample periods 4-5
 - See also* sample times
- sample times 4-16
 - design decisions for 4-5
 - handling across simulation domains 4-7
 - specifying for block output ports 4-23
- Sample-based processing 4-42
- Save and Run option, MATLAB 3-20
- scalar data types
 - conversions of 2-11
 - VHDL 2-6
- scheduling options 3-11
- script
 - HDL simulator setup 1-23
- search path 2-35
- sensitivity lists 3-11
- sensitivity option 6-5
 - specifying scheduling options with 3-13
- server activation 5-3
- server shutdown 5-3
- server, MATLAB
 - checking link status of MATLAB 3-5
 - for MATLAB and HDL simulator links 1-7
 - for Simulink and HDL simulator links 1-8
 - identifying in a network configuration 1-17
 - starting MATLAB 3-7
- service names
 - TCP/IP 1-18
- Set/Clear Breakpoint option, MATLAB 3-20
- shared memory communication 1-9

- as a configuration option 1-15
- specifying for HDL Cosimulation blocks 4-32
- specifying with `hdl1daemon` function 3-7
- Shared memory parameter
 - description of 7-2
 - specifying block communication with 4-32
- signal names
 - specifying paths
 - in MATLAB 2-4
 - in Simulink 7-2
- signal pathnames
 - displaying 4-23
 - specifying for block clocks 4-34
 - specifying for block ports 4-23
- Signal Processing Blockset
 - as optional software 1-20
 - using for EDA applications 4-5
- signals
 - continuous 4-8
 - defining ports for 2-6
 - driven by multiple sources 4-7
 - exchanging between simulation domains 4-7
 - handling across simulation domains 4-7
 - how Simulink drives 4-7
 - logging changes to 4-44
 - logging changes to values of 4-44
 - mapping to block ports 4-23
 - multirate 4-15
 - read/write access for cosimulation 4-23
 - read/write access required 7-2
- signed data 2-20
- simulation analysis 4-44
- simulation time 2-16
 - guidelines for 4-8
 - representation of 4-8
 - scaling of 4-8
- simulations
 - comparing results of 4-44
 - ending 3-22
 - logging changes to signal values during 4-44
 - quitting 3-22
- simulator resolution limit 2-17
- simulators
 - handling of signal values between 4-7
- Simulink
 - as optional software 1-20
 - configuration restrictions for 1-15
 - configuring for HDL models 4-18
 - creating hardware model designs with 4-5
 - driving cosimulation signals with 4-7
 - in Link for Cadence Incisive simulation environment 1-7
 - installing 1-22
 - simulation time for 4-8
 - using with HDL simulator 4-1
 - working with HDL simulator links to 1-11
- Simulink Fixed Point
 - as optional software 1-20
 - using for EDA applications 4-5
- Simulink models
 - adding HDL models to 4-19
- sink device
 - adding to a Simulink model 4-19
 - specifying block ports for 4-23
 - specifying clocks for 4-34
 - specifying communication for 4-32
 - specifying Tcl commands for 4-37
- socket numbers 3-5
 - See also* port numbers
- socket option
 - specifying TCP/IP socket with 3-13
 - with `hdlsimulink` command 6-3
 - with `matlabcp` command 6-5
 - with `matlabtb` command 6-10
 - with `matlabtbeval` command 6-15
- socket port numbers 1-18
 - as a networking requirement 1-17
 - checking 3-5
 - specifying for HDL Cosimulation blocks 4-32
 - specifying with -socket option 3-13

- socket property
 - description of 5-3
 - specifying with `hdldaemon` function 3-7
- sockets 1-9
 - See also* TCP/IP socket communication
- socketsimulink property
 - description of 5-11
- software
 - installing Link for Cadence Incisive 1-23
 - installing related applications 1-22
 - optional 1-20
 - required 1-20
- source device
 - adding to a Simulink model 4-19
 - specifying block ports for 4-23
 - specifying clocks for 4-34
 - specifying communication for 4-32
 - specifying Tcl commands for 4-37
- standard logic data 2-20
- standard logic vectors
 - converting for HDL simulator 2-21
 - converting for MATLAB 2-20
- start time 4-8
- startup commands, HDL simulator 3-10
- startupfile property
 - description of 5-11
 - specifying with `nclaunch` function 3-10
- status option
 - checking value of 3-5
 - description of 5-3
- status, link 3-5
- STD_LOGIC data type 2-6
 - conversion of 2-11
- STD_LOGIC_VECTOR data type 2-6
 - conversion of 2-11
- STD_ULONGIC data type 2-6
 - conversion of 2-11
- STD_ULONGIC_VECTOR data type 2-6
 - conversion of 2-11
- Step button

- in MATLAB 3-20
- Step-In button, MATLAB 3-20
- Step-Out button, MATLAB 3-20
- stimuli, block internal 4-34
- stop time 4-8
- strings, time value 3-12
- subtypes, VHDL 2-6

T

- Tcl commands
 - added to script for execution at startup 1-23
 - added to startup script via `nclaunch` 5-11
 - configuring for block simulation 4-37
 - for HDL simulator 6-1
 - `hdlsimmatlab` 6-2
 - `hdlsimulink` 6-3
 - `matlabcp` 6-5
 - `matlabtb` 6-10
 - `matlabtbeval` 6-15
 - `nomatlabtb` 6-18
 - passed to simulator from `hdldaemon` 5-3
 - post-simulation
 - using `set_param` 4-37
 - pre-simulation
 - using `set_param` 4-37
 - requirements for HDL Cosimulation
 - blocks 4-20
 - specified in Tcl pane of HDL Cosimulation
 - block 7-2
 - specifying for HDL Cosimulation block 4-20
 - specifying with `nclaunch` function 3-10
 - specifying with `tclstart` property 3-10
 - when used with MATLAB 1-10
 - when used with Simulink 1-11
- Tcl pane
 - description of 7-2
- `tclstart` property
 - specifying with `nclaunch` function 3-10
 - with `nclaunch` function 5-11

- TCP/IP alias. *See* service names
 - TCP/IP networking protocol 1-9
 - as a networking requirement 1-17
 - See also* TCP/IP socket communication
 - TCP/IP socket communication
 - as a communication option 1-15
 - feature 1-5
 - mode 1-9
 - specifying with `hdldaemon` function 3-7
 - using service names 1-18
 - TCP/IP socket ports 1-18
 - specifying for HDL Cosimulation blocks 4-32
 - specifying with `-socket` option 3-13
 - test bench functions
 - adding to MATLAB search path 2-35
 - coding for HDL verification 2-10
 - defining 2-16
 - naming 2-15
 - sample of 2-26
 - specifying required parameters for 2-16
 - test bench sessions. *See* MATLAB link sessions
 - test benches 1-5
 - See also* test bench functions
 - time 4-8
 - callback 2-16
 - delta 4-16
 - simulation 2-16
 - guidelines for 4-8
 - representation of 4-8
 - See also* time values
 - TIME data type 2-6
 - conversion of 2-11
 - time property
 - description of 5-3
 - setting return time type with 3-7
 - time scale, VCD file 4-47
 - time units 3-13
 - time values 3-13
 - specifying as scheduling options 3-11
 - specifying with `hdldaemon` function 3-7
 - Timescales pane
 - description of 7-2
 - timing errors 4-8
 - Timing mode
 - absolute 4-30
 - configuring for cosimulation 4-30
 - relative 4-30
 - `tnext` parameter 2-16
 - controlling callback timing with 3-12
 - specifying as scheduling options 3-11
 - time representations for 3-12
 - `tnow` parameter 2-16
 - To VCD File block 1-5
 - configuring 4-44
 - description of 7-23
 - generating VCD files with 4-44
 - uses of 1-11
 - Tool Command Language. *See* Tcl commands
 - tools, postprocessing 4-44
 - `tscale` parameter 2-17
 - tutorials 1-28
- ## U
- unsigned data 2-20
 - unsupported data types 2-6
 - users, Link for Cadence Incisive software 1-4
- ## V
- value change dump (VCD) files. *See* VCD files
 - VCD file name parameter
 - configuring To VCD File block with 4-44
 - description of 7-23
 - VCD files 1-5
 - format of 4-47
 - generating 4-44
 - using 4-44
 - `vcd2wlf` command 4-44
 - vectors

- converting for MATLAB 2-20
- converting to 2-21
- verification
 - coding functions for 2-10
 - hardware model 1-5
- verification sessions
 - logging changes to signal values during 4-44
 - monitoring 3-20
 - running 3-20
 - stopping 3-22
- Verilog data types
 - conversion of 2-11
- Verilog models 1-3
 - See also* HDL models
- Verilog modules
 - coding for MATLAB verification 2-3
 - naming 2-4
- VHDL data types
 - conversion of 2-11
- VHDL entities

- coding for MATLAB verification 2-3
- naming 2-4
- sample definition of 2-8
- visualization
 - coding functions for 2-10
 - overview of 2-10

W

- Wave Log Format (WLF) files 4-44
- wave window, Incisive simulator 4-23
- waveform files 4-44
- which function 2-35
- wire data type
 - conversion of 2-11
- WLF files 4-44

Z

- zero-order hold 4-8