Link for Incisive®

For Use with MATLAB® and Simulink®

- Computation
- Visualization
- Programming
- Simulation

User's Guide

Version 1

How to Contact The MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support

`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Link for Incisive User's Guide

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.

Incisive® is a registered trademark of Cadence Design Systems.

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	New for Version 1 (Release 2006b+)

Contents

1 Getting Started

2 Coding a Link for Incisive MATLAB Application

3 Starting and Controlling MATLAB Link Sessions

4 Modeling and Verifying an HDL Design with Simulink

1

Getting Started

What Is Link for Incisive? (p. 1-2)	Identifies typical applications and expected users, lists key product features, describes the Link for Incisive cosimulation environment, and provides overviews of how you work with the integrated tool environment.
Installation and Setup (p. 1-12)	Explains how to install and set up Link for Incisive.
Getting Help with Link for Incisive (p. 1-25)	Identifies and explains how to gain access to available documentation online help, demo, and tutorial resources.

What Is Link for Incisive?

Link for Incisive® is a cosimulation interface 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

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

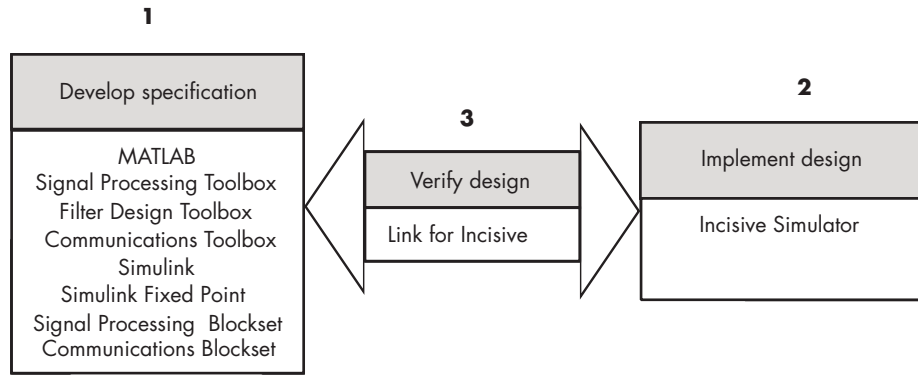
The following sections discuss

Typical Applications

Link for Incisive 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 Incisive connects tools that are traditionally used discretely to accomplish specific steps in the design process. By connecting the tools, Link for Incisive 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 Incisive 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 Incisive 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 Incisive enables engineers to cosimulate and verify a design directly between the specification and implementation, eliminating the need for manual comparisons. Link for Incisive 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 Incisive assumes users have a moderate level of prerequisite knowledge in the following subject areas:

- Hardware design and system integration
- 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 Incisive include

- Ability to link the HDL simulator to MATLAB and Simulink for bidirectional cosimulation, verification, and visualization
- Support for Linux and Solaris platforms
- Full Verilog support and support for VHDL and mixed language via Verilog I/O
- 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 Incisive, providing shared memory (for faster performance) and TCP/IP sockets (for versatility)
- A Simulink block for cosimulating HDL models 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

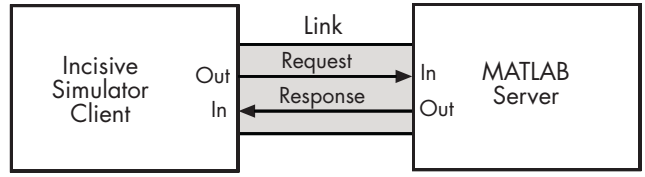
- Multiple simulation options from MATLAB, including connection of multiple MATLAB components or test benches to one or more MATLAB servers

The Cosimulation Environment

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

MATLAB and HDL Simulator Links

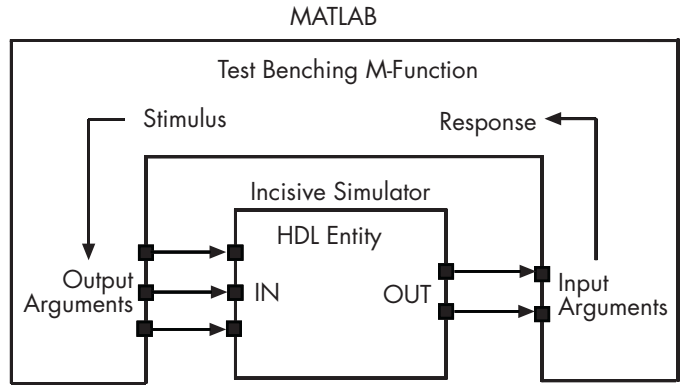
When linked with MATLAB, the HDL simulator functions as the client, as the following figure shows.



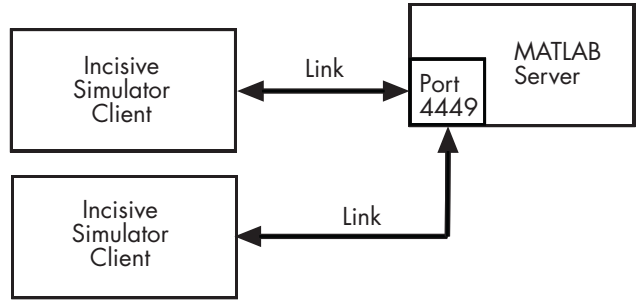
In this scenario, a MATLAB server function waits for service requests that it receives from an Incisive simulation session. After receiving a request, the server establishes a communication link and invokes a specified MATLAB function wrapper that computes data for, verifies, or visualizes the HDL model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

Note You cannot initiate Link for 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.

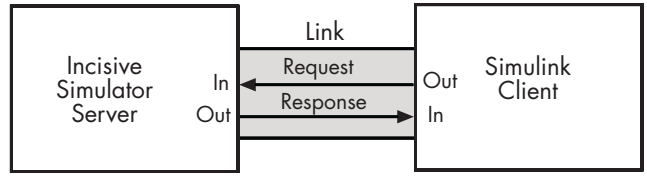


The MATLAB server can service multiple simultaneous HDL simulator sessions and HDL models. However, you should follow recommended guidelines to ensure the server can track the I/O associated with each model and session. The following figure shows a multiple-client scenario connecting to the server at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



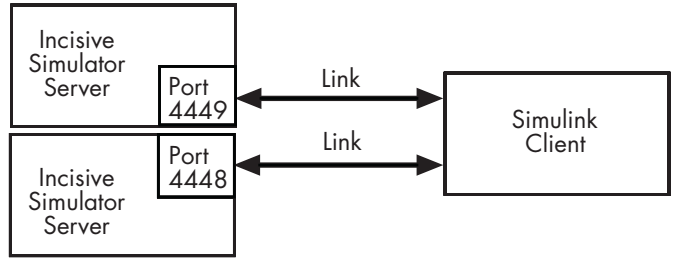
Simulink and HDL Simulator Links

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 Link for Incisive uses for a link 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 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 Link for Incisive, see "Setting Up the HDL Simulator for Use with Link for Incisive" on page 1-21.

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 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 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 1-9). 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 Link for Incisive, see “Setting Up the HDL Simulator for Use with Link for Incisive” on page 1-21.

Link for Incisive equips the HDL simulator with a set of Link for 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 Incisive 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 Incisive application environment. Topics include

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 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	
☐ Will your application use multiple communication links?	"Deciding on a Configuration" on page 1-14
☐ How many instances of the MATLAB server are required?	"Deciding on a Configuration" on page 1-14
☐ 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 1-14

Environment Requirements Checklist (Continued)

Requirement	For More Information, See...
☐ How many MATLAB functions do you need to write to model your HDL implementation?	"Deciding on a Configuration" on page 1-14
☐ 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 1-14
☐ To how many HDL simulator sessions will your Simulink model connect?	"Deciding on a Configuration" on page 1-14
Mode of Communication	
☐ 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 1-8
☐ Does your application require only one communication link (channel) on a single computing system?	"Modes of Communication" on page 1-8
☐ Is configuration flexibility a high priority for your application? Does the application have growth potential?	"Modes of Communication" on page 1-8
☐ Do you prefer to use the TCP/IP socket mode of communication for a single-computer configuration? If so, do you want Link for Incisive to identify an available socket port on the system or do you want to choose a socket port yourself?	"Choosing TCP/IP Socket Ports" on page 1-17
Network Configurations	
☐ Have you identified the computer systems that will function as Link for Incisive servers?	"Identifying a Server in a Network Configuration" on page 1-16
☐ 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 1-16
☐ Do you want Link for Incisive 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 1-17

Environment Requirements Checklist (Continued)

Requirement	For More Information, See...
Related Software	
☐ Is the HDL simulator installed on all systems as needed for your application?	"Installing Related Application Software" on page 1-20
☐ 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 1-20
☐ 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 1-20
☐ Will you be using the Simulink component of Link for Incisive? 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 1-20
HDL Simulator Setup	
☐ Will you be running the HDL simulator on a machine that does not have MATLAB installed?	"Setting Up the HDL Simulator for Use with Link for Incisive" on page 1-21

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 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 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 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 Incisive connections

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

Choosing TCP/IP Socket Ports

To use the TCP/IP socket communication, you must choose a TCP/IP socket port number that is available in your computing environment for use by the Link for 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 alias) is a shared resource. To avoid potential collisions, particularly on servers, you should use caution when choosing a port number for your application. Consider the following guidelines:

- If you are setting up a link for MATLAB, consider the Link for 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 Incisive interface to fail to make a connection.

  In such cases the error messages displayed by the Link for Incisive 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 Incisive choose an available port number for you.

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

Checking Product Requirements

Link for Incisive requires the following:

Platform	Linux 32 & 64 Solaris 32 Windows 32 (for MATLAB and Simulink)
Application software	Incisive HDL Simulator, Incisive Design Team Simulator, or Incisive Enterprise Specman Simulator. Visit the MathWorks Link for Incisive requirements page for specific versions supported with the current release of Link for Incisive. 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 of the Link for Incisive demos require one or more of the above.
Platform-specific software	The Link for Incisive shared libraries (`liblfihdls*.so`, `liblfihdlc*.so`) are built using the `gcc` included in the Incisive 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 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 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 Link for Incisive

Based on your configuration decisions, identify systems on which you need to install Link for Incisive. Install Link for Incisive 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 Incisive, see the MATLAB installation instructions.

Setting Up the HDL Simulator for Use with Link for Incisive

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 Incisive commands for use with Incisive simulators. See "Setting Up Link for Incisive for Use with the Incisive Simulator on the Same Machine as MATLAB" on page 1-21.
- If you choose to use a different machine, follow the instructions in "Setting Up Link for Incisive for Use with the Incisive Simulator on a Separate Machine from MATLAB" on page 1-22.

Setting Up Link for Incisive 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 Link for Incisive, use the `nclaunch` command with the following arguments.

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

Where `lfiinit` is the name you choose for the Tcl startup script. The property name/value pair `'starthldsim'` 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 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 Incisive 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 Incisive product page.

Copying Libraries and Creating Simulation Requirements.

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

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

Where `arch` is the system type of the platform running the HDL simulator: `glnx86`, `glnxa64`, or `sol2`.

Note If you are running `ncsim` in 32-bit mode on a 64-bit Linux platform, copy the libraries from `glnx86`.

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 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/liblfihdlc_IUS_VERSION:matlabclient
-input filename
```

   b. For the link to Simulink:

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

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

Note The Link for Incisive shared libraries were built against the GCC libraries included with the Incisive 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 `glnxa64` architecture in a csh:

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

Getting Help with Link for Incisive

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

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 Incisive MATLAB Application	Explains how to code HDL models and MATLAB functions for Link for Incisive MATLAB applications. Provides details on how the Link for 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 Incisive functions for use with MATLAB.

Title	Description
HDL Simulator Tcl Commands — Alphabetical List	Describes Link for Incisive Tcl commands for use with the HDL simulator.
Simulink Blocks — Alphabetical List	Describes Link for 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 Incisive product link in the browser's Contents.
- M-help for Link for 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

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

```
>> demos
```

Select **Toolboxes** then “Link for Incisive” from the navigational pane.

2

Coding a Link for Incisive MATLAB Application

Overview

Link for Incisive 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. La La LA.

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 Incisive. To program the HDL component of a Link for Incisive application, you must perform the following tasks:

1. Code the HDL model for MATLAB verification.
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 HDL Designs for MATLAB Verification

The most basic element of communication in the Link for Incisive interface is the HDL model. The interface passes all data between the HDL simulator and MATLAB as port data. Link for Incisive works with any existing HDL model. However, when coding an HDL design 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:

Note Link for Incisive currently supports only Verilog, although mixed-language simulations should be possible as long as all cosimulation signals are in Verilog modules.

Steps for Coding HDL Models

To code an HDL model for verification in the MATLAB environment, perform the following steps:

1. Choose an HDL model name.

   Consider choosing a model name that can be used as a valid MATLAB function name. By default, the Link for Incisive interface assumes that an HDL model and its simulation function share the same name. If the model and function names do not match, you must specify the MATLAB function name explicitly when you initialize a MATLAB link session (a MATLAB test bench or component function) with the HDL simulator `matlabtb`, `matlabtbeval`, or `matlabcp` command.

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

2. Specify required ports.

3. Specify an HDL data type that is supported by the Link for Incisive interface for each port.

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

- `reg`
- `integer`
- `wire`

For details on how Link for Incisive converts data types for the MATLAB environment, see “Verilog Data Type Conversions” on page 2-7.

Note If you use unsupported types, Link for Incisive issues a warning and ignores the port at run time.

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 Verilog compilation, elaboration, and simulation.

The following Incisive simulator command compiles and elaborates the HDL 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 platform documentation for complete details on compiling and elaborating your HDL designs. For more examples, see the Link for 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

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

1. Understand how Link for Incisive converts HDL model data (Verilog) 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.

Verilog Data Type Conversions

The Link for Incisive interface 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 Incisive converts supported Verilog data types to MATLAB types. Only scalar data types are supported for Verilog.

Verilog-to-MATLAB Data Type Conversions

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

Array Indexing Differences Between MATLAB and HDL

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 HDL.

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 Incisive interface assumes the name for a MATLAB function matches the name of the HDL model that the function verifies or visualizes.

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 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 3-10 and "Controlling Callback Timing from a MATLAB Test Bench or Component Function" on page 3-11. For more information on port data, see "Gaining Access to and Applying Port Information" on page 2-9.

Gaining Access to and Applying Port Information

The Link for 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...
field1	`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
field2	*portname*	The name of the port	All port types
field3	`type`	The port type For Verilog, `'verilog_logic'` identifies port types `reg`, `wire`, `integer`	All port types
	`size`	(Verilog) The size of the bit vector containing the data	All port types
	`label`	(Verilog) The string `'01ZX'`	(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 bit vector to an unsigned integer	Use the `bin2dec` function to convert the data to an unsigned decimal value. For example: `uval = bin2dec(oport.val')` This example assumes the bit vector is composed of the character literals `'1'` and `'0'` only. These are the only two values that can be converted to an integer equivalent.

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

Converting Data for Return to the HDL Simulator

If your simulation MATLAB function needs to return data to the HDL simulator, it may be necessary for you to first convert the data to a type supported by the Link for Incisive interface. The following tables list circumstances requiring such conversions for Verilog.

Verilog Conversions for Incisive Simulators

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

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 3-11.

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 3-11.

- `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 2-9.

For more information on using `tnext` and `tnow` for simulation scheduling, see "Deciding on MATLAB Link Session Scheduling Options" on page 3-10.

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
D:\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.

3

Starting and Controlling MATLAB Link Sessions

Running and Monitoring a MATLAB Link Session (p. 3-19)	Explains how to run and monitor a MATLAB link session.
Stopping a MATLAB Link Session (p. 3-20)	Explains how to stop a MATLAB link session.

Overview

Link for Incisive 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 3-9.)
4. Schedule invocations of the MATLAB test bench or component function. (See "Deciding on MATLAB Link Session Scheduling Options" on page 3-10.)
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 3-11.)
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 3-12.)
7. Apply MATLAB link session stimuli. (See "Applying Stimuli with the HDL Simulator force Command" on page 3-17.)

8 Run and monitor the MATLAB link session. (See “Running and Monitoring a MATLAB Link Session” on page 3-19.)

9 Stop a MATLAB link session. (See “Stopping a MATLAB Link Session ” on page 3-20.)

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:

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

If the server is not running, the message reads

```
HDLDaemon is NOT running
```

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

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

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

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

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

Starting the MATLAB Server

Start the MATLAB server as follows:

1. Start MATLAB.
2. In the MATLAB Command Window, call the `hdldaemon` function with property name/property value pairs that specify whether Link for Incisive is to use shared memory or TCP/IP socket communication.

   Use the following syntax:

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

   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 the `matlabtb`, `matlabtbeval`, or `matlabcp` HDL simulator command.

   In addition, if you specify TCP/IP socket mode, the socket port that you specify with this function and the HDL simulator command must match.

Link for Incisive returns time values in seconds.

The following function call starts the server in TCP/IP socket mode, using port number 4449.

```
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 1-17. 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 3-12.

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 Incisive. See “Setting Up Link for Incisive for Use with the Incisive Simulator on the Same Machine as MATLAB” on page 1-21.

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 Incisive 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 Link for Incisive 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 3-11. For information on setting other scheduling parameters, see “Initializing the HDL Simulator for a MATLAB Link Session” on page 3-12.

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. `tnext = 1e-9`
`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. `tnext=int64(100)`

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

Initializing the 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 an instance of an 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 1-17.

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

For more information on modes of communication, see “Modes of Communication” on page 1-8. For information on choosing socket ports, see “Choosing TCP/IP Socket Ports” on page 1-17. For more information on starting the MATLAB end of the communication link, see “Starting the MATLAB Server” on page 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	`time`[, ...]	`time` represents one of `n` time values, past time 0, at which the MATLAB function executes. For example: `matlabtb` *`entity`* `10 ns, 10 ms, 10 s -mfunc` *`function`* The MATLAB function executes when time equals 0 and then 10 nanoseconds, 10 milliseconds, and 10 seconds from time zero.

Simulation Scheduling Options (Continued)

To Specify MATLAB Function Execution...	Include...	Where...
At a combination of explicit times and repeatedly at an interval	`time[, ...] -repeat n`	`time` represents a time value at which the MATLAB function executes and the `n` specified with `-repeat` represents an interval between MATLAB function executions. For example: `matlabtb entity 5 ns -repeat 10 ns -mfunc function` 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 `x` time value is used.
Stop the executions after *x* amount of time	`time[, ...] n -repeat x -cancel`	For example: `matlabtb entity 5 ns -repeat 10 ns -cancel 1 us -mfunc function` This cancellation stops the execution after 100 microseconds.
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[, ...]`	`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.

Simulation Scheduling Options (Continued)

To Specify MATLAB Function Execution...	Include...	Where...
		Note Use of this option for `INOUT` ports can result in double calls.

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:

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

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

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

If a connection exists, the function returns the message

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

Applying Stimuli with the 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's 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.

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.

4

Modeling and Verifying an HDL Design with Simulink

Using Frame-Based Processing in Cosimulation (p. 4-40)	Explains how to improve the performance of your cosimulation by using frame-based signals.
Using a Value Change Dump File for Design Verification (p. 4-42)	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 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.

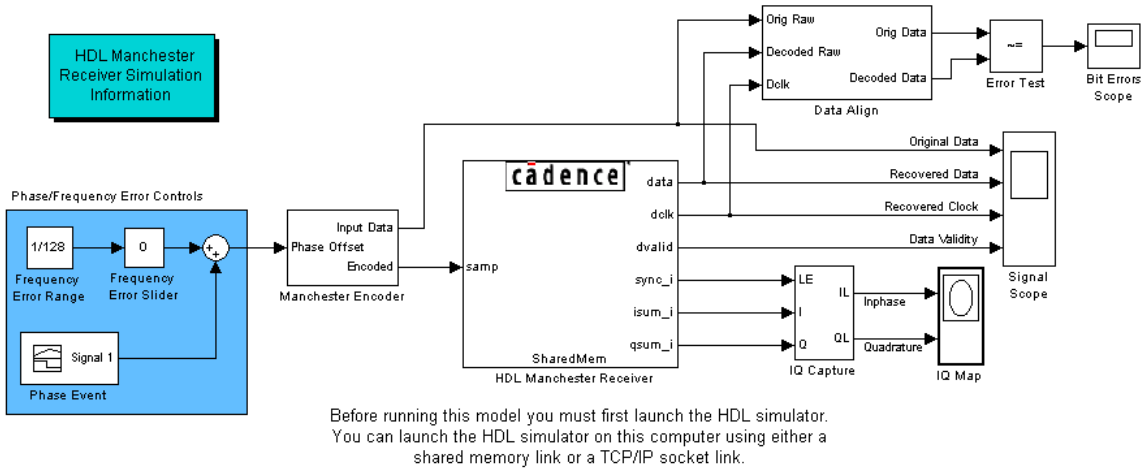
Creating a Hardware Model Design in Simulink

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.



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

- Frequency Error Range block, Frequency Error Slider block, and Phase Event block
- Manchester encoder subsystem
- Data alignment subsystem
- Inphase/Quadrature (I/Q) capture subsystem
- Error Rate Calculation block from the Communications Blockset
- 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 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 Incisive cosimulation application, you should be familiar with how signal values are handled across simulators. See the following topics:

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, Simulink applies the resolution function at each time step defined by the signal's Simulink sample rate. Depending on the other drivers, the Simulink value may or may not get applied. Furthermore, Simulink has no control over signal changes that occur between its sample times.

Representation of Simulation Time

The representation of simulation time differs significantly between 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:

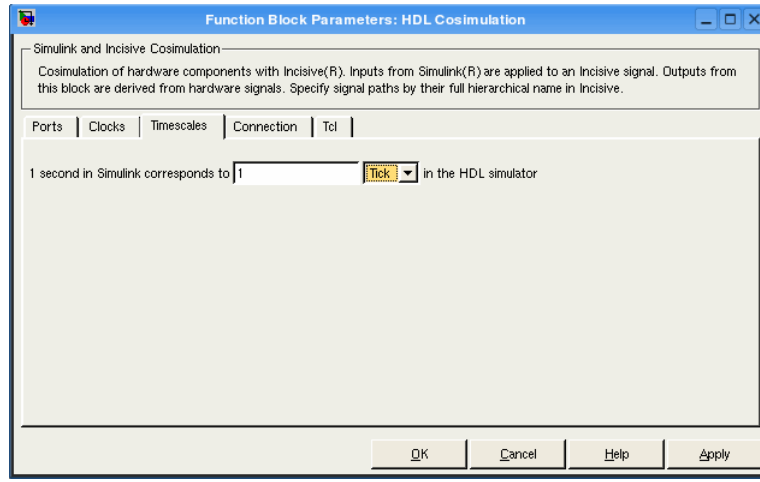
- Total simulation time
- Input port sample times
- Output port sample times
- Clock periods

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.

Link for Incisive 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

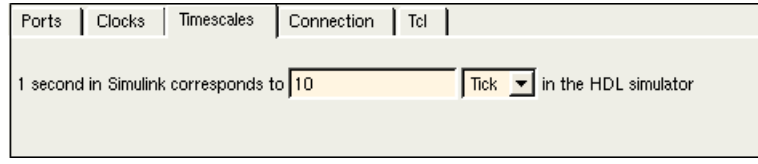
Relative timing mode lets you define the timing relationship between Simulink and the HDL simulator in terms of relative time units and a scale factor, e.g., *One second* in Simulink corresponds to *N ticks* in the HDL simulator, where `N` is a scale factor.

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:

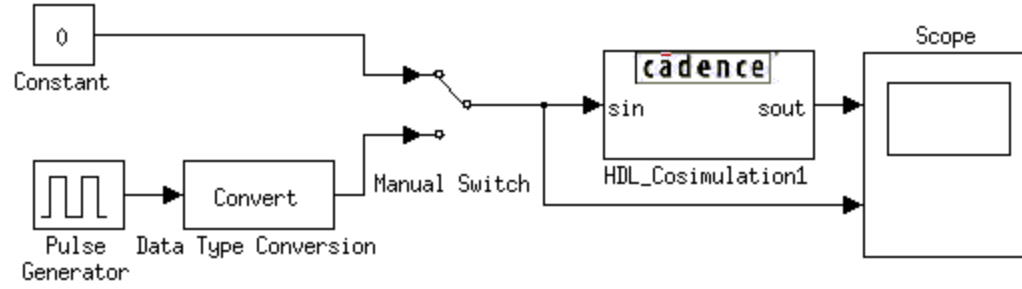
- If the total simulation time in Simulink is specified as `N` seconds, then the HDL simulation runs for exactly `N` ticks (i.e., `N` ns at the default resolution limit).
- Similarly, if Simulink computes the sample time of an HDL Cosimulation block input port as *Tsi* seconds, new values are deposited on the HDL input port at exact multiples of *Tsi* ticks. If an output port has an explicitly

specified sample time of *Tso* seconds, values are read from the HDL simulator at multiples of *Tso* ticks.

- Clocks operate in a similar fashion. Where a clock has a period of *T* seconds:
  - If *T* is even, the clock signal is forced in 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 HDL code for the inverter:

```
module inverter_clock_vl(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_vl.sin`) sample time: `N/A`
  - Output port (`inverter_clock_vl.sout`) sample time: `1 s`
  - Clock (`inverter_clock_vl.clk`) period: `5 s`
- HDL simulator resolution limit: `1 ns`

The previous example was excerpted from the Link for Incisive Inverter tutorial. For more information, see the Link for Incisive demos.

Absolute Timing Mode

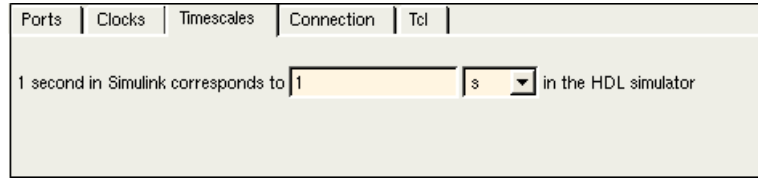
Absolute timing mode lets you define the timing relationship between Simulink and the HDL simulator in terms of absolute time units and a scale factor, e.g., *One second* in Simulink corresponds to *(N * Tu) seconds* in the HDL simulator, where *Tu* is an absolute time unit (e.g., ms, ns, etc.) and *N* 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 = (12ns * (1s / 1ns)) = 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 a 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 Incisive 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.

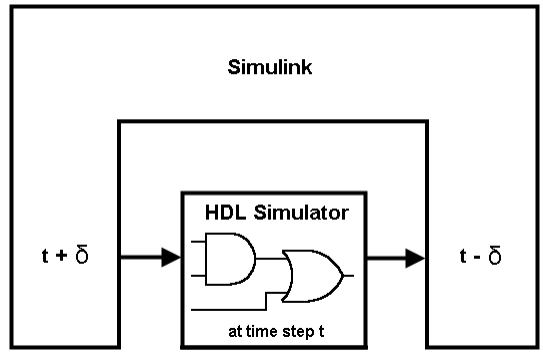
To avoid such race conditions, Link for Incisive delays all such clocks by ½ clock period, in effect inverting the sense of the rising or falling edge. The delay provides a setup and hold time for input data, ensuring that data inputs are always applied before the driving clock edge is applied. For example,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

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

Block Simulation Latency

Simulink and Link for 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 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 + δ) and block output reflects signal values just before the current HDL simulator step time (t - δ).



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 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:

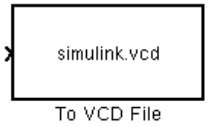
Parameter	Default Setting
`'SingleTaskRateTransMsg'`	`'error'`
`'Solver'`	`'fixedstepdiscrete'`
`'SolverMode'`	`'singletasking'`
`'StartTime'`	`'0.0'`
`'StopTime'`	`'inf'`
`'FixedStep'`	`'auto'`
`'SaveTime'`	`'off'`
`'SaveOutput'`	`'off'`
`'AlgebraicLoopMsg'`	`'error'`

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 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 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 4-42.

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.

Note In a mixed-language HDL model (one that contains both VHDL and Verilog components), a cosimulation block can access signals only with the language of the top-level module instance or component. Currently, that language must be Verilog.

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 Link for Incisive
- **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?

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	
☐ 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

HDL Cosimulation Block Requirements Checklist (Continued)

Requirement	For More Information, See...
☐ 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
☐ 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
☐ 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
☐ If the block is block is modeling a source device, do you want to specify an output sample time other than two clock ticks? If you do not specify an input port, the block uses a default sample time of two clock ticks.	"Mapping HDL Signals to Block Ports" on page 4-23
Timing	
☐ What is the optimal timing relationship between Simulink and the Incisive simulator for your cosimulation?	"Representation of Simulation Time" on page 4-8
☐ 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 4-29
☐ 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 4-29
Communication	
☐ Is it critical that communication performance be as optimal as possible?	"Configuring the Communication Link" on page 4-31

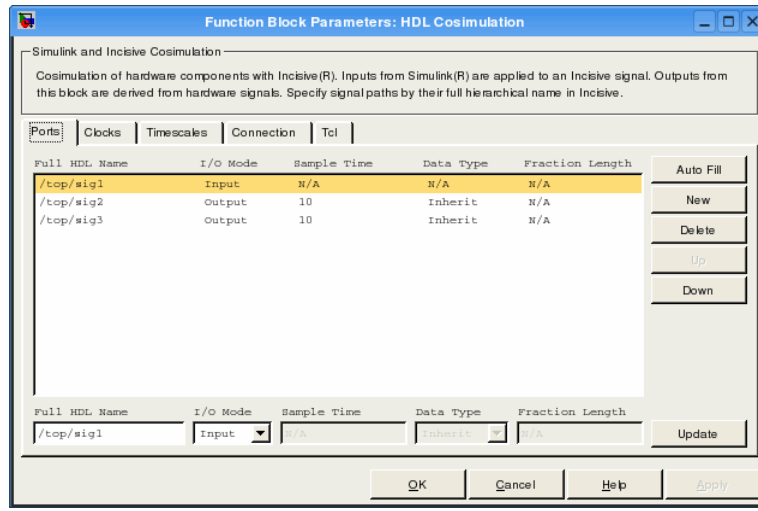
HDL Cosimulation Block Requirements Checklist (Continued)

Requirement	For More Information, See...
☐ Are you running the Incisive simulator and Simulink on the same computer?	"Configuring the Communication Link" on page 4-31
☐ 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 4-31
☐ 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 4-31
☐ 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 4-31
Clocks	
☐ Do you want to create a rising-edge clock to apply stimuli to your cosimulation model?	"Creating Optional Clocks" on page 4-33
☐ Do you want to create a falling-edge clock to apply	"Creating Optional Clocks" on page
☐ Do you want to specify the period for rising/falling edge clocks specified in the model?	"Creating Optional Clocks" on page 4-33
Tcl	
☐ 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 4-36
☐ 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 4-36

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 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.

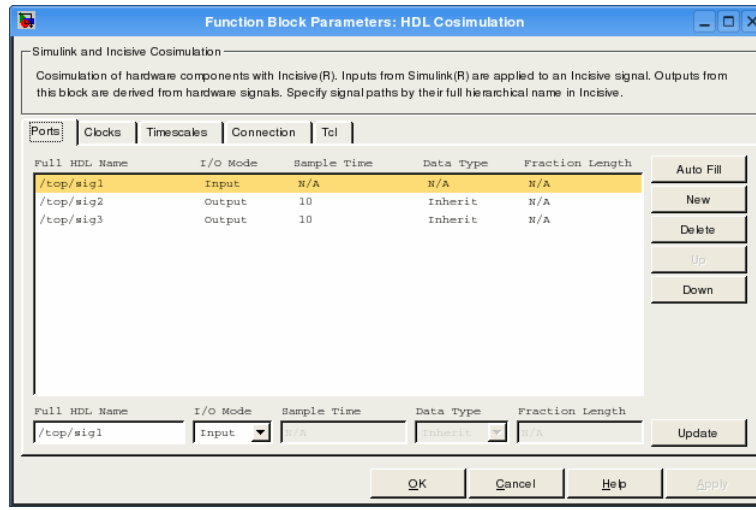
Entering Signal Information Manually

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 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.
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.

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

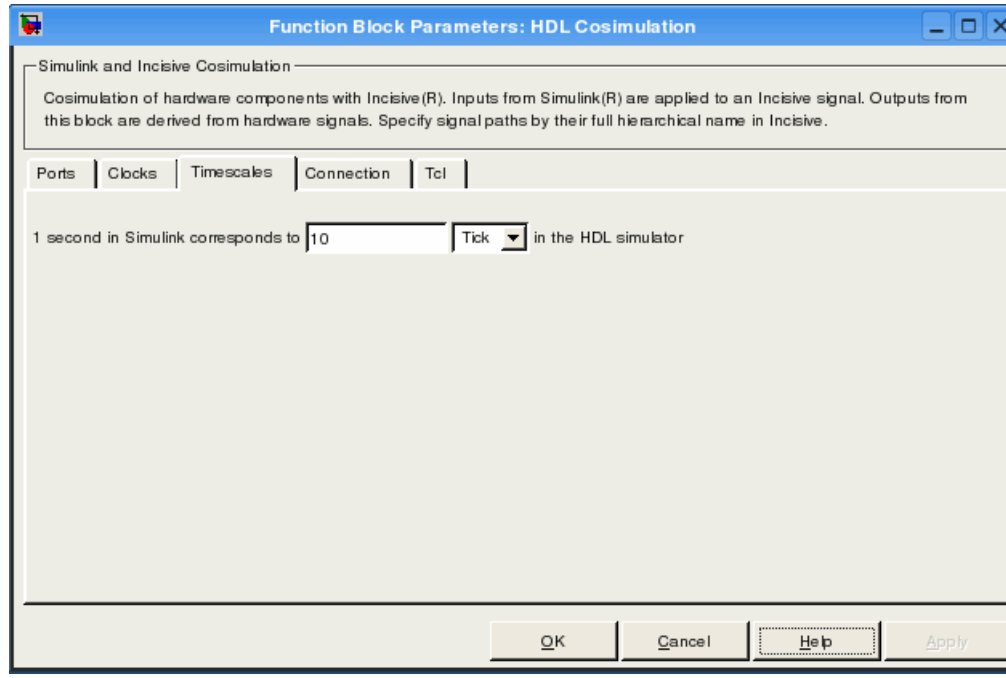
To configure relative timing mode for a cosimulation:

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

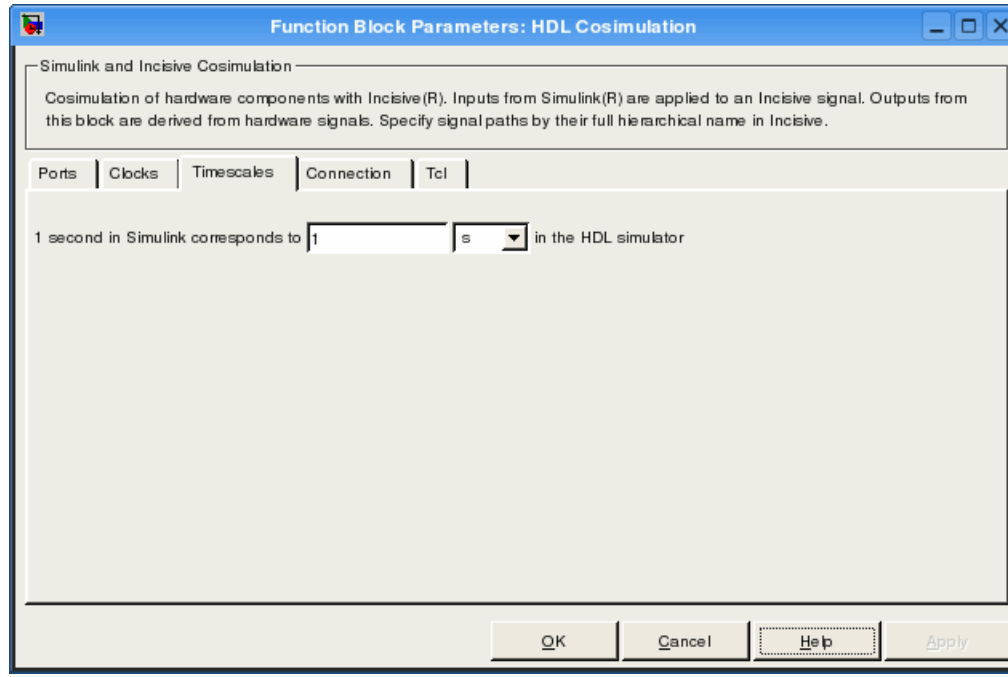
To configure absolute timing mode for a cosimulation:

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 Incisive simulator in the **Host name** text field. In the **Port number or service** text field, specify a valid port number or service for your computer system. For information on choosing TCP/IP socket ports, see "Choosing TCP/IP Socket Ports" on page 1-17. Skip to step 5.

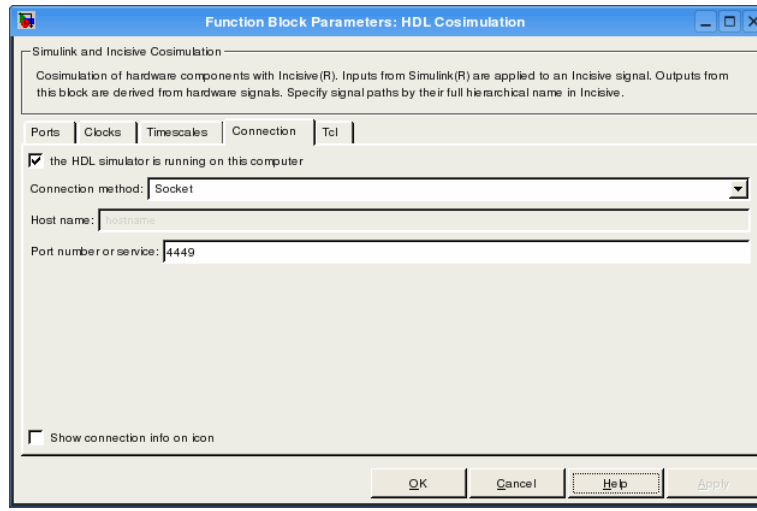
4 If the Incisive simulator and Simulink are running on the same computer, decide whether you are going to use shared memory or TCP/IP sockets for the communication channel. For information on the different modes of communication, see "Modes of Communication" on page 1-8.

   If you choose TCP/IP socket communication, specify a valid port number or service for your computer system in the **Port number or service** text field. For information on choosing TCP/IP socket ports, see "Choosing TCP/IP Socket Ports" on page 1-17.

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

5 Click **Apply**.

The following example dialog 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.

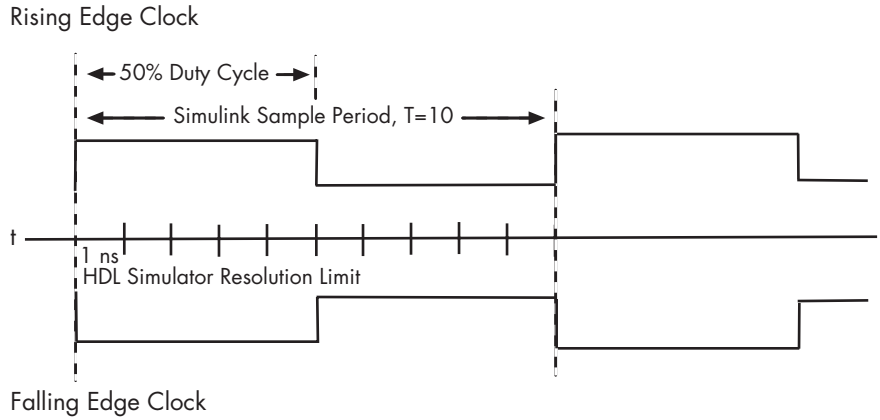


Creating Optional Clocks

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`.

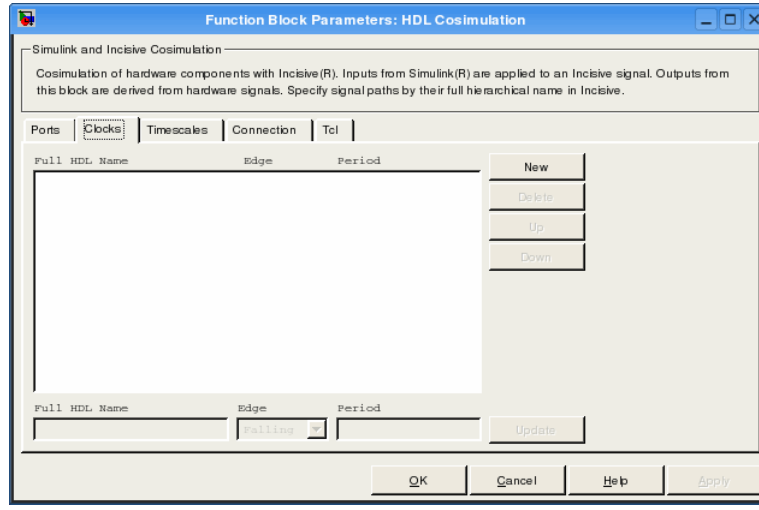
The following figure shows a timing diagram that includes rising and falling edge clocks with a Simulink sample time of `T=10` and an Incisive simulator resolution limit of 1 ns. The figure also shows that given those timing parameters, the clock duty cycle is 50%.



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.

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

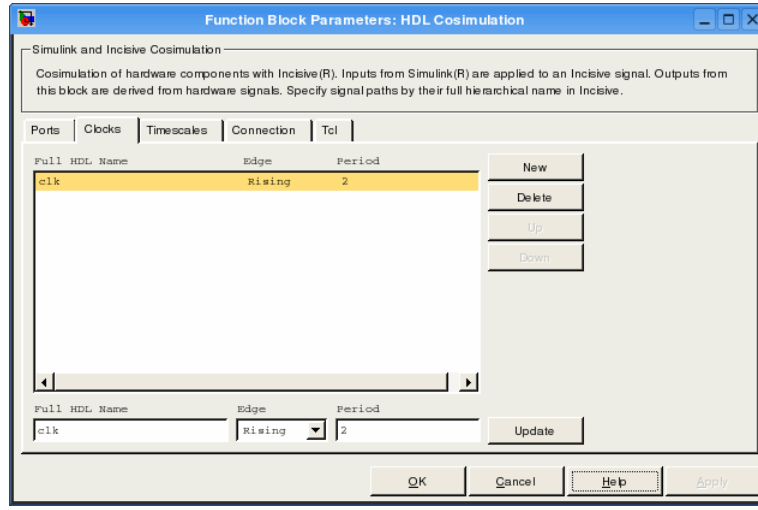
6 The **Period** field specifies the clock period. Accept the default (2), or override it by entering the desired clock period explicitly in the **Period** field.

   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`

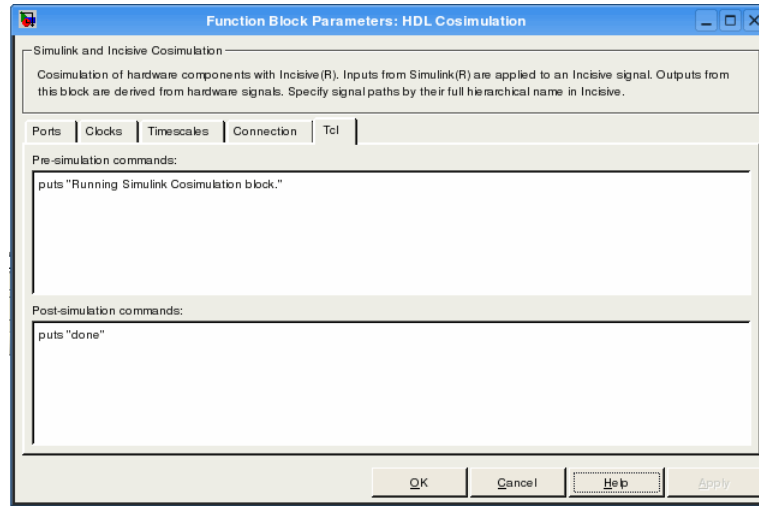
Notes

- 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/clk_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

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 4-41. 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.

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 wave form 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 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 4-42
- "VCD File Format" on page 4-45

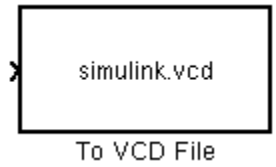
Generating a VCD File

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 Incisive library. The browser displays four types of blocks, one of which is the To VCD File block.

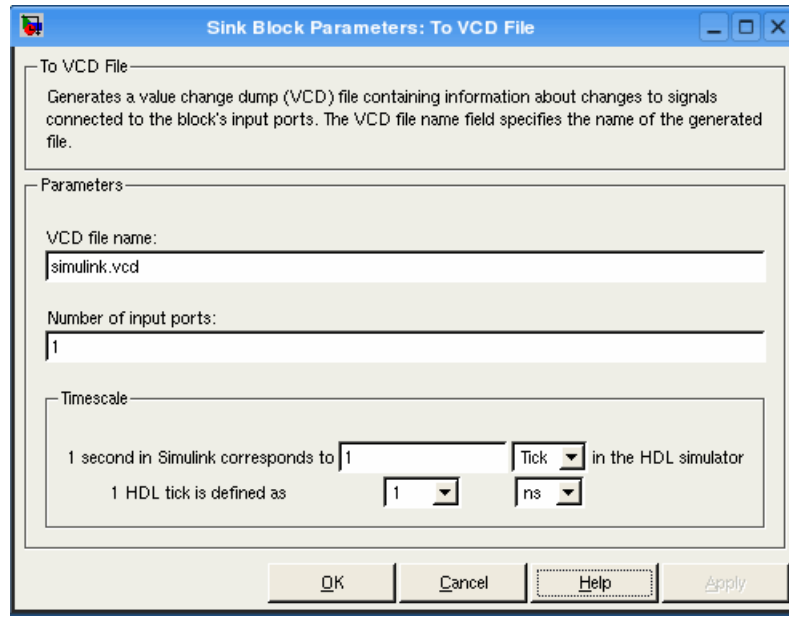


4 Copy the To VCD File block from the Library Browser to your model by clicking the block and dragging it from the browser to your model window.

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

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 4-45.

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
`$timescale 1 ns $ end`	All timestamps for VCD variable value changes are related to this single timescale.
`$scope module manchestermodel $end`	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

`$comment SL scale=1.000000 Tick;`
`HDL tick=1 ns; SL2HDL`
`Scaling Factor=1.000000 $end`
	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.

`$var wire 1 ! Original Data [0] $end`
`$var wire 1 " Recovered Clock [0] $end`
`$var wire 1 # Recovered Data [0] $end`
`$var wire 1 $ Data Validity [0] $end`
	Variable definitions. Each definition associates a signal with character identification code (symbol). The symbols are derived from printable characters in the ASCII character set from ! to ~. Variable definitions also include the variable type (wire) and size in bits.

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).

5

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 the Link for Incisive interface

Syntax

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

Description

Server Activation

`hdldaemon` starts the MATLAB server component of Link for Incisive with the shared memory communication enabled and the 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 `hldaemon` 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.

For more information on modes of communication, see “Modes of Communication” on page 1-8. 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 3-12.

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

Link Status

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

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

You can also use this function to check on the mode of communication being used, the number of existing connections, and the interprocess communication identifier (`ipc_id`) being used for a link by assigning the return value of `hdldaemon` to a variable. The `ipc_id` identifies a port number for TCP/IP socket links or the file system name for a shared memory communication channel. For example:

```
x=hdldaemon('status')
x =
            comm: 'sockets'
    connections: 0
         ipc_id: '4449'
```

This function call indicates that the server is using TCP/IP socket communication with socket port 4449 and is running with no active 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.

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 1-17.

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.

`'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...	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.

If Your Application Is to...	Do the Following...
Operate in TCP/IP socket mode, using a specific TCP/IP socket port	Specify the `'socket', tcp_spec` property name and value pair. The *tcp_spec* can be a socket port number or service name. Examples of valid port specifications include `'4449'`, `4449`, and `MATLAB Service`. For information on choosing a TCP/IP socket port, see "Choosing TCP/IP Socket Ports" on page 1-17.
Operate in TCP/IP socket mode, using a TCP/IP socket that the operating system identifies as available	Specify `'socket', 0` or `'socket', '0'`.
Execute Tcl command immediately upon simulator connection	Specify the `'tclcmd', 'command'` property name and value pair. Command must be a valid Tcl command but cannot include commands that load an Incisive simulator project or modify the simulator state.

The following function call starts the MATLAB server with shared memory communication enabled:

```
hdldaemon
```

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:

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

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]]}')
```

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`

nclaunch

Purpose

Start and configure Incisive simulators for use with Link for Incisive

Syntax

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

Description

`nclaunch('PropertyName', 'PropertyValue'...)` starts the Incisive simulator for use with the MATLAB and Simulink features of Link for Incisive. 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', 'tempdir'`
: Specifies where to run the HDL simulator. By default, the function uses the current working directory. If `'tempdir'` is specified, the function creates a temporary directory in which it runs the HDL simulator.

`'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
`<port-num>`	`4449`
`<port-alias>`	`matlabservice`
`<port-num>@<host>`	`4449@compa`

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

For more information on choosing TCP/IP socket ports, see "Choosing TCP/IP Socket Ports" on page 1-17.

If 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 Incisive for Use with the Incisive Simulator on a Separate Machine from MATLAB" on page 1-22.

`'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 Link for Incisive 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 `vlogmatlabc` 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
```

6

HDL Simulator Tcl Commands — Alphabetical List

hdlsimmatlab

Purpose

Load instantiated HDL design for verification with MATLAB

Syntax

```
hdlsimmatlab <instance> [<ncsim_args>]
```

Arguments

`<instance>`
: Specifies the instance of an HDL design to load for verification.

`<ncsim_args>`
: Specifies one or more `ncsim` command arguments. For details, see the description of `ncsim` in the Incisive simulator documentation.

Description

The `hdlsimmatlab` 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.

This command may be run from the HDL simulator prompt or from a Tcl script shell (`tclsh`).

Examples

The following command loads the module instance `parse` from library `work` for verification and sets up the Incisive simulator so it can establish a communication link with MATLAB:

```
tclshell> hdlsimmatlab work.parse
```

Purpose

Load instantiated HDL design for cosimulation with Simulink

Syntax

```
hdlsimulink [<ncsim_args>] <instance>
[-socket <tcp_spec>]
```

Argument

`<ncsim_args>`

Specifies one or more `ncsim` command arguments. At a minimum, either `-gui` or `-tcl` is required. If you specify `-gui`, the Simulink GUI will be launched when the HDL design is loaded. If you specify `-tcl`, 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 `ncsim` arguments may be specified in addition to `-gui` or `-tcl`. For more information on `-gui`, `-tcl`, or other `ncsim` arguments, see the description of `ncsim` in the Incisive simulator documentation.

`<instance>`

Specifies the instance of an HDL design to load for cosimulation.

`-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 `nclaunch` function. The `<tcp_spec>` can consist of a TCP/IP socket port number or service name (alias). For example, you might specify port number 4449 or the service name `matlabservice`.

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

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

For more information on modes of communication, see "Modes of Communication" on page 1-8. For more information on establishing the Simulink end of the communication link, see "Configuring the Communication Link" on page 4-31.

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 `matlabtb`).

`<time-specs>`

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

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

	time the MATLAB function is initially called.
`-cancel <time>`	Specifies a time at which the specified MATLAB function stops executing. The time value is relative to the value of `tnow` at the time the MATLAB function is initially called. If you do not specify a cancel time, the command calls the MATLAB function.

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 1-17.

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 1-8. 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 Link for Incisive.
- Associates a specified instance of an HDL design created in the Incisive simulator with a MATLAB function.

Note Link for Incisive currently supports only Verilog, although mixed-mode simulations should be possible as long as all cosimulation signals are in Verilog modules.

- 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 2-14.

Examples

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

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

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:

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

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 (`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 1-17.

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 `hdldaemon` command in MATLAB.

For more information on modes of communication, see "Modes of Communication" on page 1-8. For more information on establishing the MATLAB end of the communication link, see "Starting the MATLAB Server" on page 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 Link for Incisive.
- Associates a specified instance of an HDL design created in the Incisive simulator with a MATLAB function.

Note Link for Incisive currently supports only Verilog, although mixed-mode simulations should be possible as long as all cosimulation signals are in Verilog modules.

- 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.

MATLAB component functions simulate the behavior of components in 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 2-14.

Examples

The following command starts the Incisive simulator client component of Link for Incisive, 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 Link for Incisive, 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 Link for Incisive. 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

`<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.

`-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>`.

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

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

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 1-8. 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 Link for Incisive.
- 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 Link for Incisive, 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 Link for Incisive, 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
```

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:

```
ncsim> nomatlabtb
```

See Also

`matlabcp`, `matlabtb`

7

Simulink Blocks — Alphabetical List

HDL Cosimulation

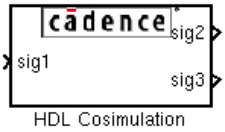
Purpose

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

Library

Link for 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 `sfix8_En5`.

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).

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 **Clocks** pane lets you create optional rising-edge and falling-edge clocks that apply stimuli to your cosimulation model. You can either specify an explicit period for each clock, or accept a default period of `2`. Simulink attempts to create a clock that has a 50% duty cycle and a predefined phase that is inverted for the falling edge case.

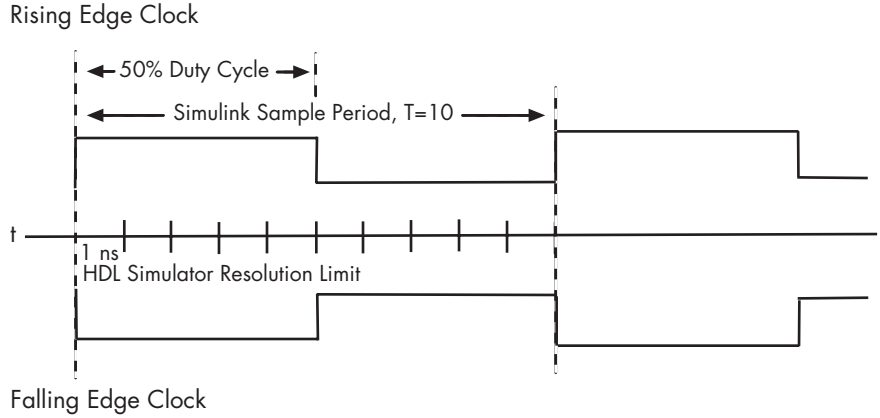
Whether you have configured the **Timescales** pane for relative timing mode or absolute timing mode, the following restrictions apply to clock periods:

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

  clockperiod / 2

  (rounded down to the nearest integer).

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



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”.

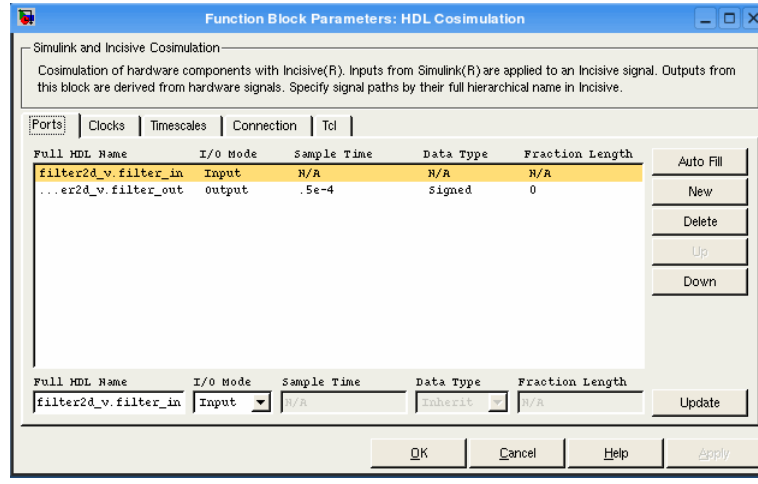
The **Tcl** pane provides a way of specifying tools command language (Tcl) commands to be executed before and after the Incisive simulator simulates the HDL component of your Simulink model. The **Pre–simulation commands** field on this pane is particularly useful for simulation initialization and startup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

Dialog Box

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

- "Ports Pane" on page 7-5
- "Connection Pane" on page 7-10
- "Timescales Pane" on page 7-12
- "Clocks Pane" on page 7-16
- "Tcl Pane" on page 7-18

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**.

You can use a VHDL delimiter (:), a Verilog delimiter (.), or a Simulink delimiter (/) in the signal pathname. You will get an error if you use an incorrect delimiter.

The following are valid signal pathnames:

- `/manchester/u_iqconv/qsum` (Simulink path delimiter)
- `manchester.u_iqconv.qsum` (default Verilog path delimiter)
- `manchester.u_iqconv.qsum[4:0]` (direct copy from HDL simulator waveform window)
- `manchester:u_iqconv:qsum` (default VHDL path delimiter, used on a Verilog signal)
- `manchester:u_iqconv:qsum[4:0]` (same, with vector specification)

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).

`Inout` designates HDL model signals that are bidirectional; that is, signals that may be either driven by Simulink or read by Simulink.

Sample Time

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

Sample Time represents the time interval between consecutive samples applied to the output port. The default sample time is `1`. The exact interpretation of the output port sample time depends on the settings of the **Timescale** pane of the HDL Cosimulation block (see "Timescales Pane" on page 7-12). 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.

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.

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

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

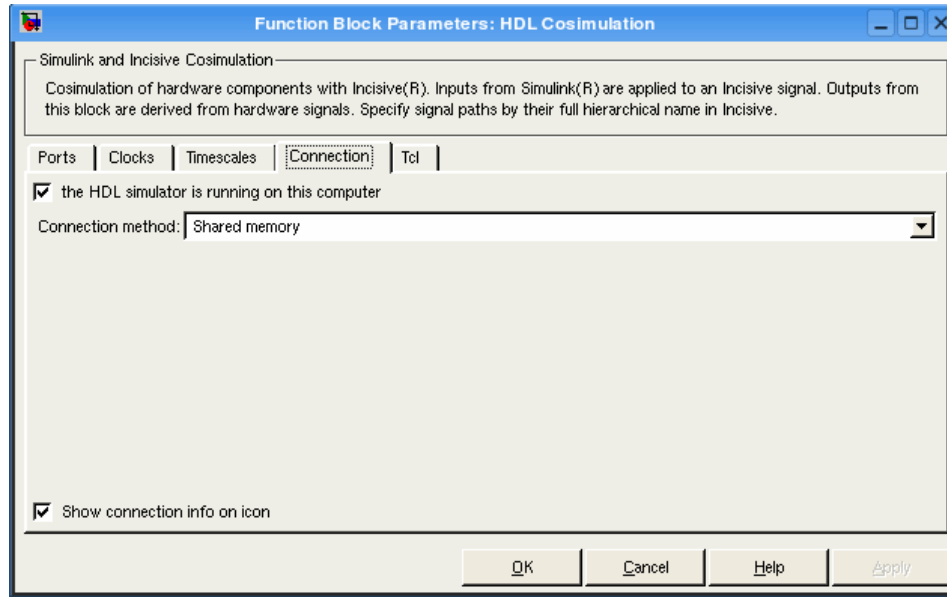
- Select `Unsigned` from the **Data Type** list if you want to explicitly assign an unsigned fixed point data type When `Unsigned` is selected, the **Fraction Length** edit field is enabled. The port is assigned a fixed point type `ufixN_EnF`, where `N` is the signal width and `F` is the **Fraction Length**.

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

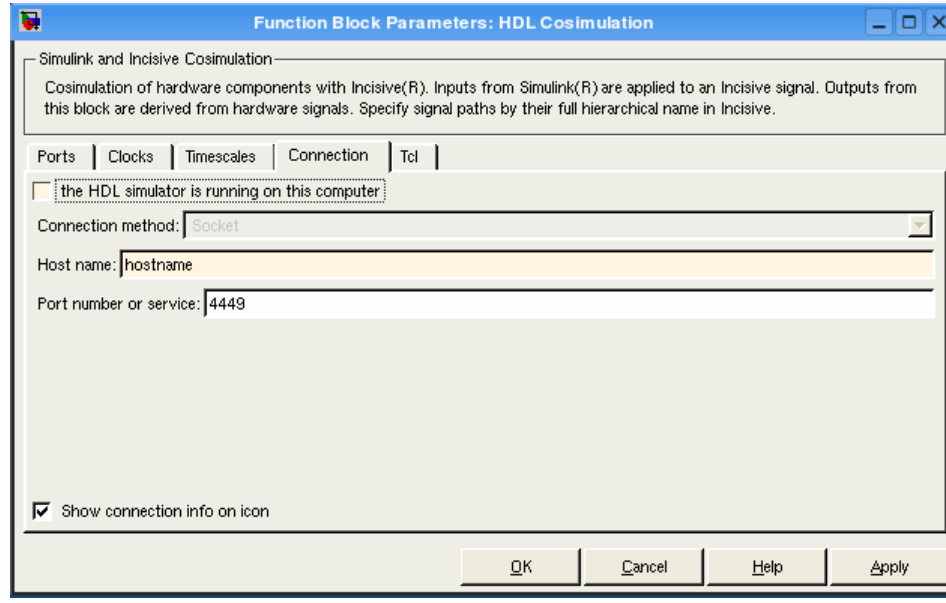
Type set to `Unsigned` and **Fraction Length** of `-5` , Simulink forces the data type to `ufix16_E5`.

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 4-31.

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.

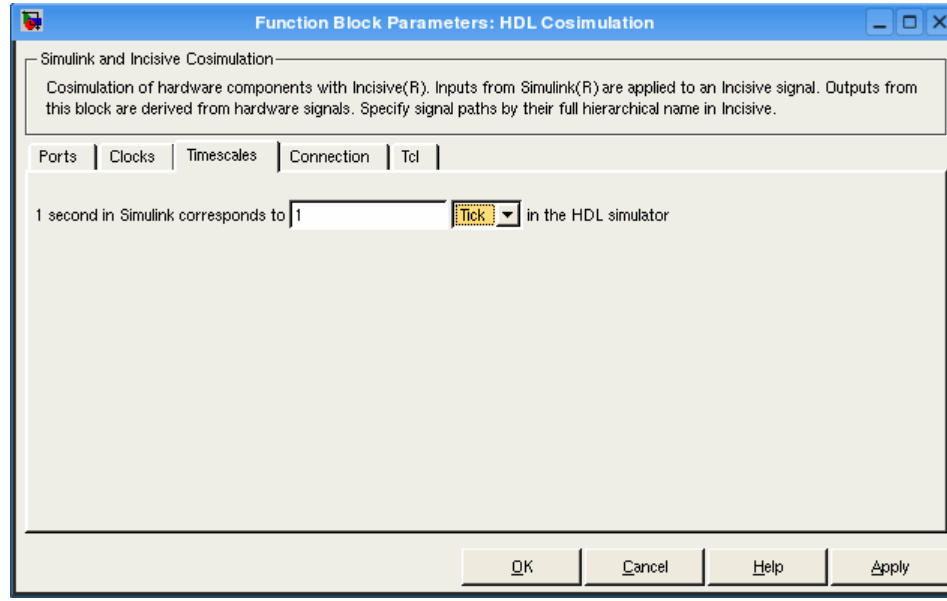
Port number or service
: Indicate a valid TCP socket port number or service for your computer system (if not using shared memory). For information on choosing TCP socket ports, see "Choosing TCP/IP Socket Ports" on page 1-17.

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.

Timescales Pane

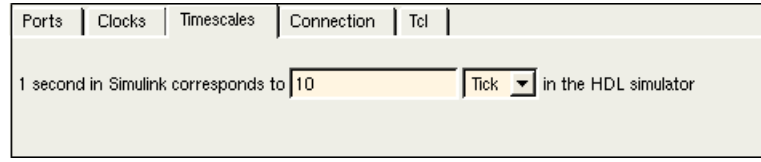
The **Timescales** pane of the HDL Cosimulation block parameters dialog lets you choose an optimal timing relationship between Simulink and the Incisive simulator. The following figure shows the default settings of the **Timescales** pane.



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:

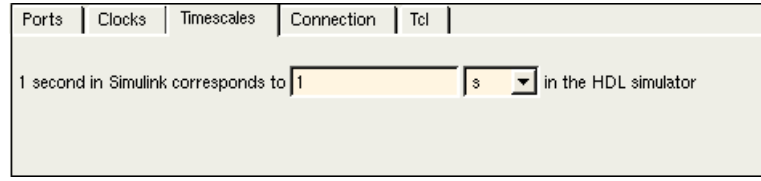
- In *relative* terms (i.e., as some number of Incisive simulator ticks). In this case, the cosimulation is said to operate in *relative timing mode*. Relative timing mode is the default.

  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 Incisive handles the timing of the falling edge when the duty cycle does not fall at 50%. The following restrictions apply to clock periods:

  - You must enter a sample time equal to or greater than 2 resolution units (ticks) (no falling edge can occur in < 2 ticks).
  - If the clock period (whether explicitly specified or defaulted) is not an even integer multiple, Simulink cannot create a 50% duty cycle, and therefore Link for Incisive creates the falling edge at

    clockperiod / 2

    (rounded down to the nearest integer).

    You must know how many ticks your selected time represents so that you know how the falling edge will occur. This next example

demonstrates how to calculate the number of HDL simulator ticks for an absolute clock period of 1 Simulink second = 3 HDL simulator seconds.

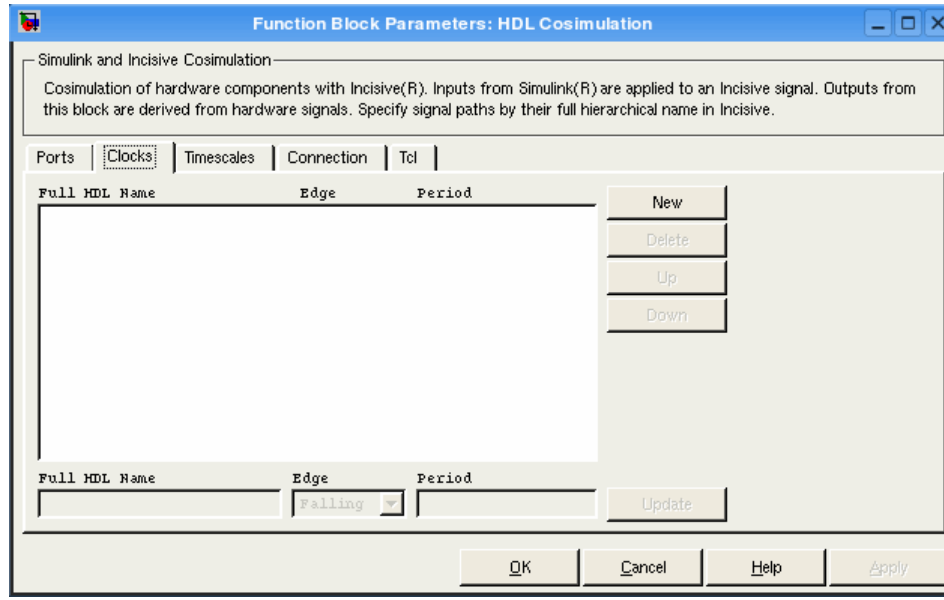
```
1 HDL simulator second = 10^9 HDL simulator ns
1 HDL simulator tick = 10 HDL simulator ns
1 HDL simulator second = (10^9/10) or 10^8 HDL simulator ticks

1 Simulink seconds = 3 HDL simulator seconds
1 Simulink second = 3x10^8 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`.

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`.

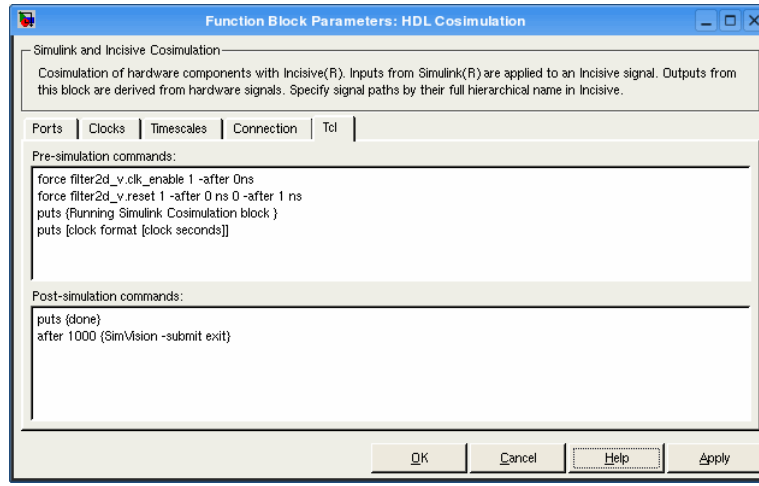
If you specify an explicit clock period, you must enter a sample time equal to or greater than 2 resolution units (ticks).

If the clock period (whether explicitly specified or defaulted) is not an even integer, Simulink cannot create a 50% duty cycle, and therefore Link for Incisive creates the falling edge at

clockperiod / 2

(rounded down to the nearest integer).

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`.

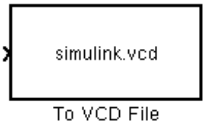
Purpose

Generate a value change dump (VCD) file

Library

Link for Incisive

Description



The To VCD File block generates a VCD file that contains information about changes to signals connected to the block's input ports and names the file with the specified file name. VCD files can be useful during design verification. Some examples of how you might apply VCD files include

- 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.

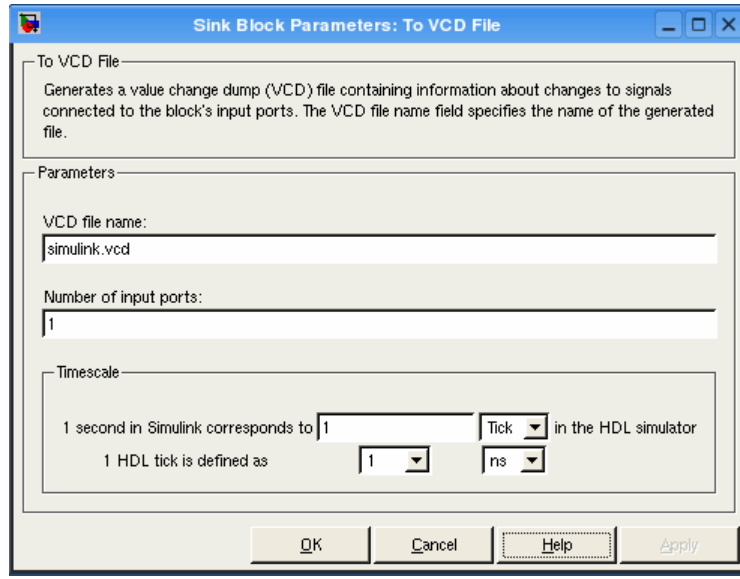
Using the Block Parameters dialog box, you can specify the following:

- 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 4-45.

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.

In some cases, a single input port maps to multiple signals (and symbols). This mapping occurs when the input port receives one of the following:

- Vector of real numbers
- Fixed-point real number

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 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** .

Index

D

E

F

I

K

L

M

R

S

T

U

V

W

Z