HDL Verifier™
Getting Started Guide
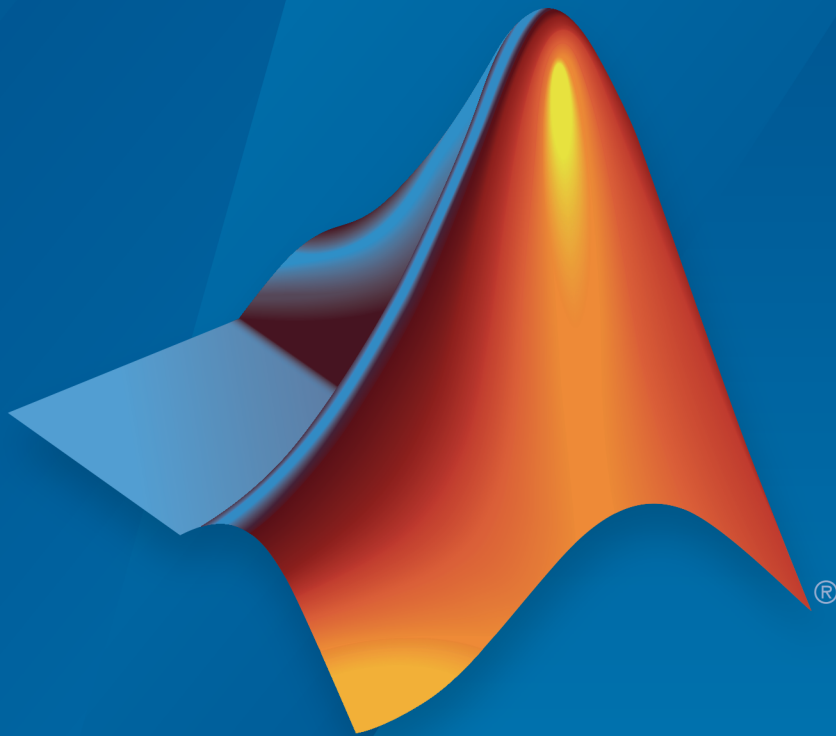
MATLAB®&SIMULINK®

R2015a

MathWorks®

How to Contact MathWorks

Latest news: www.mathworks.com

Sales and services: www.mathworks.com/sales_and_services

User community: www.mathworks.com/matlabcentral

Technical support: www.mathworks.com/support/contact_us

Phone: 508-647-7000

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

Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See www.mathworks.com/trademarks for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

Patents

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

Contents

3

Third-Party Product Requirements

4

System Objects

Introduction

HDL Verifier Product Description
Verify VHDL and Verilog using HDL simulators and FPGA-in-the-loop test benches

HDL Verifier™ automates Verilog® and VHDL® design verification using HDL simulators and FPGA hardware-in-the-loop. It provides interfaces that link MATLAB® and Simulink® with Cadence Incisive®, Mentor Graphics® ModelSim®, and Mentor Graphics Questa® HDL simulators. It also supports FPGA-in-the-loop verification with Xilinx® and Altera® FPGA boards.

HDL Verifier automates verification by using MATLAB or Simulink to stimulate your HDL code and analyze its response. This approach eliminates the need to author standalone Verilog or VHDL test benches.

Key Features

- Cosimulation support for Cadence Incisive and for Mentor Graphics ModelSim and Questa
- FPGA-in-the-loop verification using Xilinx and Altera FPGA boards
- MATLAB functions and Simulink blocks
- Generation of IEEE® 1666 SystemC TLM 2.0 compatible transaction-level models
- Interactive or batch-mode cosimulation and debugging
- Single-machine, multiple-machine, and cross-network cosimulation

2

About HDL Verifier

HDL Cosimulation

HDL Cosimulation with MATLAB or Simulink

The HDL Verifier software consists of MATLAB functions, a MATLAB System object™, and a library of Simulink blocks, all of which establish communication links between the HDL simulator and MATLAB or Simulink.

HDL Verifier software streamlines FPGA and ASIC development by integrating tools available for the following processes:

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.
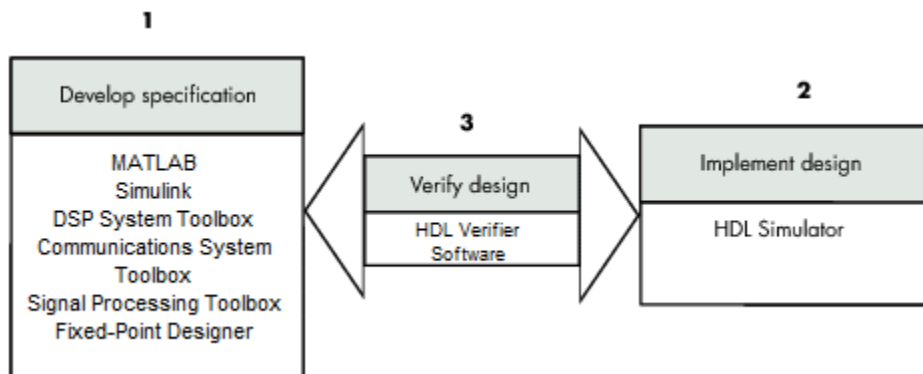
As the figure shows, HDL Verifier software connects tools that traditionally have been used discretely to perform specific steps in the design process. By connecting these tools, the link simplifies verification by allowing you to cosimulate the implementation and original specification directly. This cosimulation results in significant time savings and the elimination of errors inherent to manual comparison and inspection.

In addition to the preceding design scenario, HDL Verifier software enables you to work with tools in the following ways:

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

---

Note: You can cosimulate a module using SystemVerilog, SystemC or both with MATLAB or Simulink using the HDL Verifier software. Write simple wrappers around the SystemC and make sure that the SystemVerilog cosimulation connections are to ports or signals of data types supported by the link cosimulation interface.

More discussion on how cosimulation works can be found in the following sections:

- "Linking with MATLAB and the HDL Simulator" on page 2-3
- "Linking with Simulink and the HDL Simulator" on page 2-5
- "The HDL Cosimulation Wizard" on page 2-6

Linking with MATLAB and the HDL Simulator

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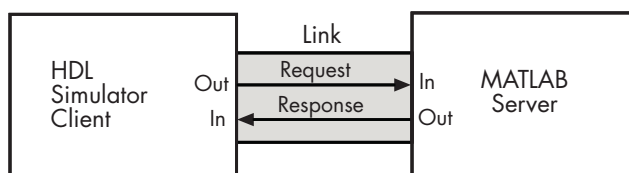
In this scenario, a MATLAB server function waits for service requests that it receives from an HDL simulator session. After receiving a request, the server establishes a communication link and invokes a specified MATLAB function that computes data for, verifies, or visualizes the HDL module (coded in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

After the server is running, you can start and configure the HDL simulator or use with MATLAB with the supplied HDL Verifier function:

- `nclaunch` (Incisive)
- `vsim` (ModelSim)

The following figure shows how a MATLAB test bench function wraps around and communicates with the HDL simulator during a test bench simulation session.
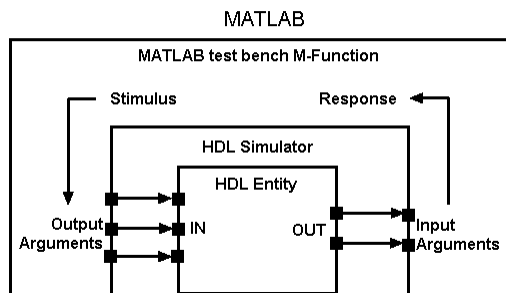
The following figure shows how a MATLAB component function is wrapped around by and communicates with the HDL simulator during a component simulation session.
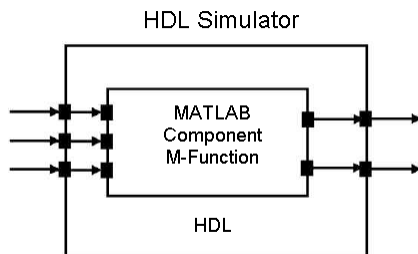
When you begin a specific test bench or component session, you specify parameters that identify the following information:

• The mode and, if applicable, TCP/IP data for connecting to a MATLAB server

• The MATLAB function that is associated with and executes on behalf of the HDL instance

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

Linking with Simulink and the HDL Simulator

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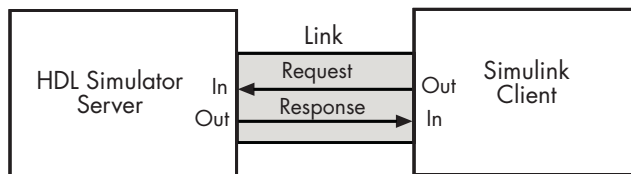
In this case, the HDL simulator responds to simulation requests it receives from cosimulation blocks in a Simulink model. You begin a cosimulation session from Simulink. After a session is started, you can use Simulink and the HDL simulator to monitor simulation progress and results. For example, you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

Using the Block Parameters dialog box for an HDL Cosimulation block, you can configure the following:

• Block input and output ports that correspond to signals (including internal signals) of an HDL module. 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 module. You can individually specify the period of each clock.

• Tcl commands to run before and after the simulation.

HDL Verifier software equips the HDL simulator with a set of customized functions. For ModelSim, when you use the function `vsimulink`, you execute the HDL simulator with an instance of an HDL module for cosimulation with Simulink. After the module is

loaded, you can start the cosimulation session from Simulink. Incisive users can perform the same operations with the function `hdlsimulink`.

HDL Verifier software also includes a block for generating value change dump (VCD) files. You can use VCD files generated with this block to perform the following tasks:

- View Simulink simulation waveforms in your HDL simulation environment
- Compare results of multiple simulation runs, using the same or different simulation environments
- Use as input to post-simulation analysis tools

The HDL Cosimulation Wizard

HDL Verifier contains the Cosimulation Wizard feature, which uses existing HDL code to create a customized MATLAB function (test bench or component), MATLAB System object, or Simulink HDL Cosimulation block. For more information, see "Import HDL Code With the HDL Cosimulation Wizard".

Communications for HDL Cosimulation

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

TCP/IP socket mode is more versatile. You can use it for single-system and network configurations. This option offers the greatest scalability. For more on TCP/IP socket communication, see "TCP/IP Socket Ports".

Hardware Description Language (HDL) Support

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

HDL Verifier software also supports mixed-language HDL models (models with both Verilog and VHDL components), allowing you to cosimulate VHDL and Verilog signals

simultaneously. Both MATLAB and Simulink software can access components in different languages at any level.

HDL Cosimulation Workflows

The HDL Verifier User Guide provides instruction for using the verification software with supported HDL simulators for the following workflows:

- Simulating an HDL Component in a MATLAB Test Bench Environment
- Replacing an HDL Component with a MATLAB Component Function
- Simulating an HDL Component in a Simulink Test Bench Environment
- Replacing an HDL Component with a Simulink Algorithm
- Recording Simulink Signal State Transitions for Post-Processing

Product Features and Platform Support

Product Feature	Required Products	Recommended Products	Supported Platforms
MATLAB and HDL simulator cosimulation (function)	MATLAB	Fixed-Point Designer™, Signal Processing Toolbox™	Windows® 32- and 64-bit; Linux® 64-bit
MATLAB and HDL simulator cosimulation (System object)	MATLAB and Fixed-Point Designer	Communications System Toolbox™, DSP System Toolbox™	Windows 32- and 64-bit; Linux 64-bit
Simulink and HDL simulator cosimulation	Simulink, Fixed-Point Designer	Signal Processing Toolbox, DSP System Toolbox	Windows 32- and 64-bit; Linux 64-bit

FPGA Verification

FPGA Verification with HDL Verifier and HDL Coder

HDL Verifier works with Simulink or MATLAB and HDL Coder™ and the supported FPGA development environment to prepare your automatically generated HDL Code for implementation in an FPGA. FPGA-in-the-Loop (FIL) simulation allows you to run a Simulink or MATLAB simulation with an FPGA board strictly synchronized with this software. This process lets you get real world data into your design while accelerating your simulation with the speed of an FPGA.

You can generate a FIL programming file in one of the following ways:

• With the HDL Verifier FIL Wizard.
• With the HDL Coder Workflow Advisor.

The FIL Wizard uses any synthesizable HDL code including code automatically generated from Simulink models by HDL Coder software. When you use FIL in the Workflow Advisor, HDL Coder uses the loaded design to create the HDL code. Either way, this HDL code is then augmented by customized code for FIL communication with your design and assembled into an FPGA project. The applicable downstream tools are used to process that project to create a programming file that is automatically downloaded to the FPGA device on a development board for verification.

HDL Verifier supports the use of a FIL block in a model reference block and a System object in conjunction with a MATLAB program.

Product Features and Platform Support

• "Preregistered FPGA Devices for FIL Simulation" on page 2-9
• "Supported FPGA Device Families for Clock Module Generation" on page 2-9
• "JTAG Cable Support for Altera Boards" on page 2-10

Product Feature	Required Products	Recommended Products	Supported Platforms
FPGA-in-the-Loop	For FIL simulation with MATLAB: MATLAB, Fixed-Point Designer For FIL simulation with Simulink: Simulink, Fixed-Point Designer	HDL Coder	Windows 32- and 64-bit; Linux 64-bit

Preregistered FPGA Devices for FIL Simulation

HDL Verifier supports FIL simulation on the devices as described in "Supported FPGA Devices for FIL Simulation" on page 3-5. The FPGA board support packages contain the definition files for all supported boards. You may download one or more vendor-specific packages, but you must download one of the packages before you can use FIL or customize your own board definition file using the New FPGA Board Wizard (see "Create Custom FPGA Board Definition").

Visit the Hardware Support Catalog for a list of currently supported devices and boards. Download an FPGA board support package with the `supportPackageInstaller` command in MATLAB.

Supported FPGA Device Families for Clock Module Generation

For project generation with Filter Design HDL Coder™, see Xilinx documentation for a full list of supported FPGA families in ISE.

With the current release, clock module generation is supported for the following device families:

- Spartan-3
- Spartan-3A and Spartan-3AN
- Spartan-3A DSP
- Spartan-3E
- Spartan-6
- Virtex-4
- Virtex-5

JTAG Cable Support for Altera Boards

- Windows

  Requires Quartus II version 13.0 or higher, Quartus II executable directory must be on system path

- Linux

  Requires Quartus II version 13.1 or higher, Quartus II library directory must be on LD_LIBRARY_PATH *before* starting MATLAB, only 64-bit Quartus are supported

TLM Component Generation

Generating TLM Components for Virtual Platform Development

HDL Verifier lets you create a SystemC Transaction Level Model (TLM) that can be executed in any OSCI-compatible TLM 2.0 environment, including a commercial virtual platform.

When used with virtual platforms, HDL Verifier joins two different modeling environments: Simulink for high-level algorithm development and virtual platforms for system architectural modeling. The Simulink modeling typically dispenses with implementation details of the hardware system such as processor and operating system, system initialization, memory subsystems, device configuration and control, and the particular hardware protocols for transferring data both internally and externally.

The virtual platform is a simulation environment that is concerned about the hardware details: it has components that map to hardware devices such as processors, memories, and peripherals, and a means to model the hardware interconnect between them.

Although many goals could be met with a virtual platform model, the ideal scenario for virtual platforms is to allow for software development—both high level application software and low-level device driver software—by having fairly abstract models for the hardware interconnect that allow the virtual platform to run at near real-time speeds, as demonstrated in the following diagram.
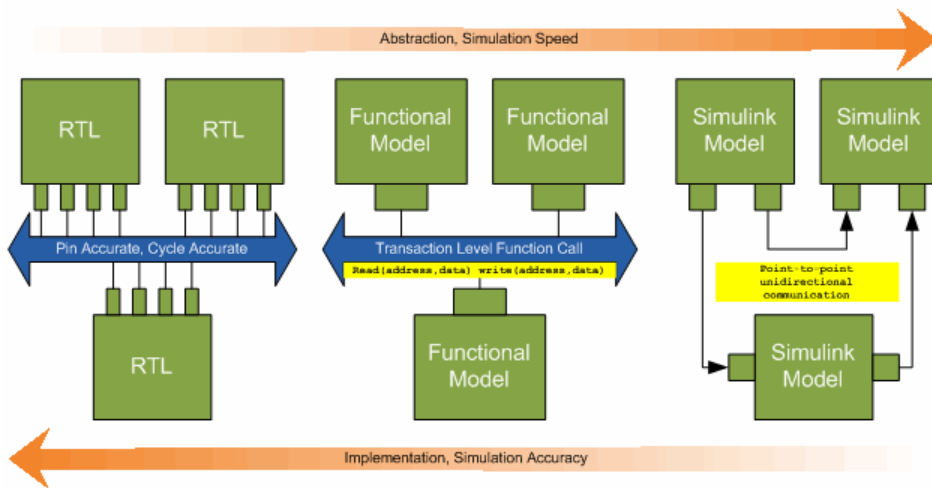
The functional model provides a sort of halfway point between the speed you can achieve with abstraction and the accuracy you get with implementation.

Typical Users and Applications

Using HDL Verifier and Simulink, you can create a TLM-2.0-compliant SystemC Transaction Level Model (TLM) that can be executed in any OSCI-compatible TLM 2.0 environment, including a commercial virtual platform.

Typical users and applications include:

- System-level engineers designing electronic system models that include architectural characteristics
- Software developers who want to incorporate an algorithm into a virtual platform without using an instruction set simulator (ISS).
- Hardware functional verification engineers. In this case, the algorithm represents a piece of hardware going into a chip.

Product Feature and Platform Support

Product Feature	Required Products	Recommended Products	Supported Platforms
TLM Generator	Simulink Coder™	Embedded Coder® (Simulink Coder is also required)	Windows 32-bit and 64–bit; Linux 64-bit

SystemVerilog DPI Component Generation

Export Simulink Subsystem or MATLAB Function With DPI Interface

Export a Simulink subsystem or MATLAB function with a DPI interface for Verilog or SystemVerilog Simulation. You can wrap generated C code with a DPI wrapper that communicates with a SystemVerilog thin interface function in a SystemVerilog simulation.

For Simulink, this feature is available in the Model Configuration Parameters dialog. For MATLAB, you generate the component using the dpigen function.

Workflows

For use with Simulink:

- "Generate SystemVerilog DPI Component and Test Bench"
- "Customize Generated SystemVerilog Code"

For use with MATLAB:

- "DPI Component Generation for MATLAB Function"

Product Feature and Platform Support

Product Feature	Required Products	Recommended Products	Supported Platforms
SystemVerilog DPI Component Generator	*For DPI component generation with Simulink:*	Embedded Coder	Windows 32-bit and 64–bit; Linux 64-bit

Product Feature	Required Products	Recommended Products	Supported Platforms
	Simulink and Simulink Coder *For DPI component generation with MATLAB:* MATLAB and MATLAB Coder		

HDL Verifier Supported Hardware

As of this release, HDL Verifier supports the following hardware.

Support Package	Vendor	Platforms	Earliest Release Available	Last Release Available
Altera FPGA Boards	Altera	Windows, Linux	R2013a	Current
BEEcube miniBEE Platform	BEEcube	Windows, Linux	R2013b	Current
Xilinx FPGA Boards	Xilinx	Windows, Linux	R2013a	Current

For a complete list of supported hardware, see Hardware Support.

Third-Party Product Requirements

Supported EDA Tools and Hardware

In this section...
"Cosimulation Requirements" on page 3-2
"FPGA Verification Requirements" on page 3-3
"TLM Generation System Requirements" on page 3-8

Cosimulation Requirements

- "Cadence Incisive Requirements" on page 3-2
- "Mentor Graphics Questa and ModelSim Usage Requirements" on page 3-2

Cadence Incisive Requirements

MATLAB and Simulink support Cadence® verification tools using HDL Verifier. Use one of these recommended versions, which have been fully tested against the current release:

- INCISIV 13.2 p002
- INCISIV 13.1 s006
- INCISIV 12.2 s007

The HDL Verifier shared libraries (`liblfihdls*.so`, `liblfihdlc*.so`) are built using the gcc included in the Cadence Incisive simulator platform distribution. Before you link your own applications into the HDL simulator, first try building against this gcc. See the HDL simulator documentation for more details about how to build and link your own applications.

Mentor Graphics Questa and ModelSim Usage Requirements

MATLAB and Simulink support Mentor Graphics verification tools using HDL Verifier. Use one of the following recommended versions. Each version has been fully tested against the current release:

- QuestaSim 10.3, 10.2c, 10.2b
- ModelSim/QuestaSim PE 10.2c, 10.1a
- ModelSim/QuestaSim DE 10.1a

• ModelSim SE 10.1c

The Linux platform requires that HDL Verifier software run gcc c++ libraries (4.1 or later). You should install a recent version of the gcc c++ library on your computer. To determine which libraries are installed on your computer, type the command:

```
gcc -v
```

FPGA Verification Requirements

• "Xilinx Usage Requirements" on page 3-3
• "Altera Quartus II Usage Requirements" on page 3-3
• "Supported FPGA Board Connections for FIL Simulation" on page 3-4
• "Supported FPGA Devices for FIL Simulation" on page 3-5
• "Supported FPGA Device Families for Board Customization" on page 3-7

Xilinx Usage Requirements

MATLAB and Simulink support Xilinx design tools using HDL Verifier.

• FPGA-in-the-Loop is tested with Xilinx ISE 14.7 and Xilinx Vivado® 2013.4.

  Xilinx ISE is required for FPGA boards in the Spartan-6, Virtex-4, Virtex-5, and Virtex-6 families. For all other supported FPGA families, Xilinx Vivado is required.
• ISE 11.1 or newer is recommended
• Consult Xilinx user documentation for compatibility of ISE tools with various Linux distributions.

Note: Xilinx does not ship the Digilent® plugin with ISE 14.2. To get the plugin, see the Digilent plugin and related software download page on the Digilent web site.

Altera Quartus II Usage Requirements

MATLAB and Simulink support Altera design tools using HDL Verifier.

• FPGA-in-the-Loop is tested with Altera Quartus II 13.1.

Supported FPGA Board Connections for FIL Simulation

JTAG Connection

Vendor	Supported Devices	Required Hardware	Required Software
Altera	Any Altera FPGA board within the supported FPGA family, for example: • Cyclone III, IV, V, and V SoC • Arria II and Arria V • Stratix IV and V • Additional boards within the supported device families can be custom added with the "FPGA Board Manager".	• Altera FPGA board • USB Blaster I or USB Blaster II download cable	• Windows: Quartus II 12.1 or higher version, Quartus II executable directory must be on system path • Linux: Quartus II 13.0sp1 with a patch, or Quartus II 13.1. Quartus II library directory must be on LD_LIBRARY_PATH *before* starting MATLAB, only 64-bit Quartus are supported • Installation of USB Blaster I or II driver
Xilinx	The FPGA board must be using an FPGA device in the supported Xilinx FPGA family: Virtex 7, Kintex 7 , Artix® 7 and Zynq 7000. All Xilinx 7-series FPGA boards that HDL Verifier supports directly (as of this release) can perform simulation through an on-board Digilent JTAG cable. Additional boards using the supported FPGA devices can	The FPGA board must be using a Digilent download cable. If your board has a standard Xilinx 14 pin JTAG connector, you can obtain the HS2 cable from Digilent.	• For Windows operating systems: Xilinx Vivado 2014.2. Vivado executable directory must be on system path • For Linux operating systems: Xilinx Vivado 2014.2 and Digilent Adept2

Vendor	Supported Devices	Required Hardware	Required Software
	be added with the FPGA Board Manager.		

Ethernet Connection

- Hardware:

  - Gigabit Ethernet card
  - Cross-over Ethernet cable
  - FPGA board with supported Ethernet connection

- PHY Interface Type:

  - Gigabit Ethernet — GMII
  - Gigabit Ethernet — RGMII
  - Gigabit Ethernet — SGMII
  - Ethernet — MII

- Software:

  - There are no software requirements for an Ethernet connection, but the firewall on the host computer should not prevent UDP communication for FIL

Supported FPGA Devices for FIL Simulation

HDL Verifier supports FIL simulation on the devices shown in the following table. The board definition files for these boards are in the "Download FPGA Board Support Package". You may also add other FPGA boards for use with FIL with FPGA board customization ().

Device Family	Board	Comments
XilinxArtix-7	Artix-7 FPGAs	
Xilinx Spartan-6	Spartan-6 SP605 Spartan-6 SP601 XUP Atlys Spartan-6	
Xilinx Virtex-7	Virtex-7 VC707	
Xilinx Virtex-6	Virtex-6 ML605	

Device Family	Board	Comments
Xilinx Virtex-5	Virtex-5 ML505 Virtex-5 ML506 Virtex-5 ML507 Virtex-5 XUPV5–LX110T	
Xilinx Virtex-4	Virtex-4 ML401 Virtex-4 ML402 Virtex-4 ML403	
Xilinx Kintex-7	Kintex-7 KC705	
Altera Arria II	Arria II GX FPGA development kit	
Altera Arria V	Arria V SoC development kit	
Altera Cyclone IV	Cyclone IV GX FPGA development kit DE2-115 development and education board BeMicro SDK	The Altera DE2-115 FPGA development board has two Ethernet ports. FPGA-in-the-loop uses only Ethernet 0 port. Make sure that you connect your host computer with the Ethernet 0 port on the board via an Ethernet cable.
Altera Cyclone III	Cyclone III FPGA development kit Altera Nios II Embedded Evaluation Kit, Cyclone III Edition	
Altera Cyclone V	Cyclone V FPGA GX development board Cyclone V SoC development kit Cyclone V GT development kit Arrow SoC development kit	The Cyclone V SoC and Arrow SoC development kits are supported for JTAG connection only.
Altera Stratix IV	Stratix IV GX FPGA development board	
Altera Stratix V	Stratix V DSP Development Kit	

Limitations

- Ethernet PHY RGMII interface is not supported for Xilinx Spartan6 family when used with FPGA-in-the-loop.
- For FPGA development boards that have more than one FPGA device, only one such device can be used with FIL.

FPGA Board Support Packages

The FPGA board support packages contain the definition files for all supported boards. You may download one or more vendor-specific packages, but you must download one of the packages before you can use FIL or customize your own board definition file using the New FPGA Board Wizard (see "Create Custom FPGA Board Definition").

Visit the Hardware Support Catalog for a list of currently supported devices and boards. Download an FPGA board support package with the `supportPackageInstaller` command in MATLAB.

Supported FPGA Device Families for Board Customization

HDL Verifier supports the following FPGA device families for board customization; that is, when you create your own board definition file. See "FPGA Board Customization".

Device Family	
Xilinx	Virtex4
	Virtex5
	Virtex6
	Spartan6
	Kintex7
Altera	Cyclone III
	Cyclone IV
	Cyclone V
	Arria II
	Arria V
	Stratix IV
	Stratix V

TLM Generation System Requirements

With the current release, TLMG includes support for:

- Compilers:

  - Visual Studio®: VS2005, VS2008, VS2010, and VS2012
  - gcc 4.4.6

- SystemC:

  - SystemC 2.3.1 (TLM included)

You can download SystemC and TLM libraries at http://accellera.org. Consult the Accellera Systems Initiative web site for information about how to build these libraries after downloading.

System Objects

What Is a System Toolbox?

System Toolbox products provide algorithms and tools for designing, simulating, and deploying dynamic systems in MATLAB and Simulink. These toolboxes contain MATLAB functions, System objects, and Simulink blocks that deliver the same design and verification capabilities across MATLAB and Simulink, enabling more effective collaboration among system designers. Available System Toolbox products include:

- DSP System Toolbox
- Communications System Toolbox
- Computer Vision System Toolbox
- Phased Array System Toolbox

System Toolboxes support floating-point and fixed-point streaming data simulation for both sample- and frame-based data. They provide a programming environment for defining and executing code for various aspects of a system, such as initialization and reset. System Toolboxes also support code generation for a range of system development tasks and workflows, such as:

- Rapid development of reusable IP and test benches
- Sharing of component libraries and systems models across teams
- Large system simulation
- C-code generation for embedded processors
- Finite wordlength effects modeling and optimization
- Ability to prototype and test on real-time hardware

What Are System Objects?

A System object is a specialized kind of MATLAB object. System Toolboxes include System objects and most System Toolboxes also have MATLAB functions and Simulink blocks. System objects are designed specifically for implementing and simulating dynamic systems with inputs that change over time. Many signal processing, communications, and controls systems are dynamic. In a dynamic system, the values of the output signals depend on both the instantaneous values of the input signals and on the past behavior of the system. System objects use internal states to store that past behavior, which is used in the next computational step. As a result, System objects are optimized for iterative computations that process large streams of data, such as video and audio processing systems.

For example, you could use System objects in a system that reads data from a file, filters that data and then writes the filtered output to another file. Typically, a specified amount of data is passed to the filter in each loop iteration. The file reader object uses a state to keep track of where in the file to begin the next data read. Likewise, the file writer object keeps tracks of where it last wrote data to the output file so that data is not overwritten. The filter object maintains its own internal states to assure that the filtering is performed correctly. This diagram represents a single loop of the system.
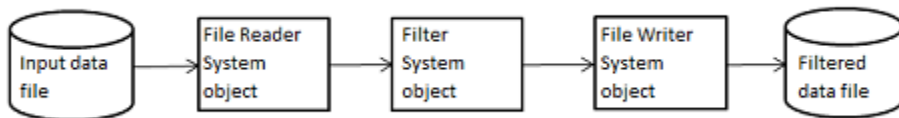
Many System objects support:

- Fixed-point arithmetic (requires a Fixed-Point Designer license)
- C code generation (requires a MATLAB Coder or Simulink Coder license)
- HDL code generation (requires an HDL Coder license)
- Executable files or shared libraries generation (requires a MATLAB Compiler™ license)

Note: Check your product documentation to confirm fixed-point, code generation, and MATLAB Compiler support for the specific System objects you want to use.

System Objects vs. MATLAB Functions

System Objects vs. MATLAB Functions

Many System objects have MATLAB function counterparts. For simple, one-time computations use MATLAB functions. However, if you need to design and simulate a system with many components, use System objects. Using System objects is also appropriate if your computations require managing internal states, have inputs that change over time or process large streams of data.

Building a dynamic system with different execution phases and internal states using only MATLAB functions would require complex programming. You would need code to initialize the system, validate data, manage internal states, and reset and terminate the system. System objects perform many of these managerial operations automatically during execution. By combining System objects in a program with other MATLAB functions, you can streamline your code and improve efficiency.

With HDL Verifier, you can choose to create an HDL cosimulation System object or an HDL cosimulation test bench or component function, depending on the needs of your application.

- "Create a MATLAB System Object"
- "Create a MATLAB Test Bench"
- "Create a MATLAB Component Function"

System Design and Simulation in MATLAB

System objects allow you to design and simulate your system in MATLAB. You use System objects in MATLAB as shown in this diagram.
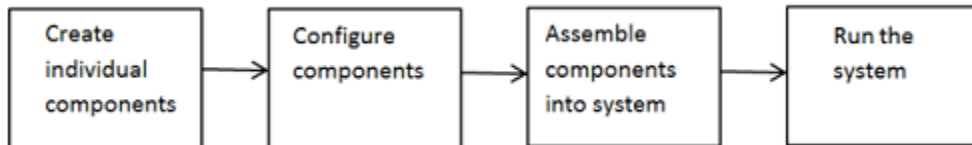
1. Create individual components — Create the System objects to use in your system. See "Create Components for Your System" on page 4-10 for information.

2. Configure components — If necessary, change the objects' property values to model your particular system. All System object properties have default values that you may be able to use without changing them. See "Configure Components for Your System" on page 4-10 for information.

3. Assemble components into system — Write a MATLAB program that includes those System objects, connecting them using MATLAB variables as inputs and outputs to simulate your system. See "Assemble Components to Create Your System" on page 4-11 for information.

4. Run the system — Run your program, which uses the `step` method to run your system's System objects. You can change tunable properties while your system is running. See "Run Your System" on page 4-12 and "Reconfigure Your System During Runtime" on page 4-12 for information.

System Objects in Simulink

System Objects in the MATLAB Function Block

You can include System object code in Simulink models using the MATLAB Function block. Your function can include one or more System objects. Portions of your system may be easier to implement in the MATLAB environment than directly in Simulink. Many System objects have Simulink block counterparts with equivalent functionality. Before writing MATLAB code to include in a Simulink model, check for existing blocks that perform the desired operation.

System Object Methods

In this section...
"What Are System Object Methods?" on page 4-7
"The Step Method" on page 4-7
"Common Methods" on page 4-8

What Are System Object Methods?

After you create a System object, you use various object methods to process data or obtain information from or about the object. All methods that are applicable to an object are described in the reference pages for that object. System object method names begin with a lowercase letter and class and property names begin with an uppercase letter. The syntax for using methods is `<method>(<handle>)`, such as `step(H)`, plus possible extra input arguments.

System objects use a minimum of two commands to process data—a constructor to create the object and the step method to run data through the object. This separation of declaration from execution lets you create multiple, persistent, reusable objects, each with different settings. Using this approach avoids repeated input validation and verification, allows for easy use within a programming loop, and improves overall performance. In contrast, MATLAB functions must validate parameters every time you call the function.

These advantages make System objects particularly well suited for processing streaming data, where segments of a continuous data stream are processed iteratively. This ability to process streaming data provides the advantage of not having to hold large amounts of data in memory. Use of streaming data also allows you to use simplified programs that use loops efficiently.

The Step Method

The `step` method is the key System object method. You use `step` to process data using the algorithm defined by that object. The `step` method performs other important tasks related to data processing, such as initialization and handling object states. Every System object has its own customized `step` method, which is described in detail on the `step` reference page for that object. For more information about the step method and other available methods, see the descriptions in "Common Methods" on page 4-8.

Common Methods

All System objects support the following methods, each of which is described in a method reference page associated with the particular object. In cases where a method is not applicable to a particular object, calling that method has no effect on the object.

Method	Description
step	Processes data using the algorithm defined by the object. As part of this processing, it initializes needed resources, returns outputs, and updates the object states. After you call the step method, you cannot change any input specifications (i.e., dimensions, data type, complexity). During execution, you can change only tunable properties. The step method returns regular MATLAB variables. Example: Y = step(H,X)
release	Releases any special resources allocated by the object, such as file handles and device drivers, and unlocks the object. For System objects, use the release method instead of a destructor.
reset	Resets the internal states of a locked object to the initial values for that object and leaves the object locked
getNumInputs	Returns the number of inputs (excluding the object itself) expected by the step method. This number varies for an object depending on whether any properties enable additional inputs.
getNumOutputs	Returns the number of outputs expected from the step method. This number varies for an object depending on whether any properties enable additional outputs.
getDiscreteState	Returns the discrete states of the object in a structure. If the object is unlocked (when the object is first created and before you have run the step method on it or after you have released the object), the states are empty. If the object has no discrete states, getDiscreteState returns an empty structure.
clone	Creates another object of the same type with the same property values
isLocked	Returns a logical value indicating whether the object is locked.

Method	Description
isDone	Applies to source objects only. Returns a logical value indicating whether the step method has reached the end of the data file. If a particular object does not have end-of-data capability, this method value returns `false`.
info	Returns a structure containing characteristic information about the object. The fields of this structure vary depending on the object. If a particular object does not have characteristic information, the structure is empty.

System Design in MATLAB Using System Objects

In this section...
"Create Components for Your System" on page 4-10
"Configure Components for Your System" on page 4-10
"Assemble Components to Create Your System" on page 4-11
"Run Your System" on page 4-12
"Reconfigure Your System During Runtime" on page 4-12

Create Components for Your System

A System object is a component you can use to create your system in MATLAB. System objects support fixed- or variable-size data. *Variable-size data* is data whose size can change at run time. By contrast, fixed-size data is data whose size is known and locked at initialization time, and therefore, cannot change at run time.

Note: If you use a function to create and use a System object, specify the object creation using conditional code. This will prevent errors if that function is called within a loop.

These examples use System objects that are predefined in the software. You can also create your own System objects (see).

The predefined components for use with HDL Verifier are:

- hdlverifier.HDLCosimulation: For HDL cosimulation between MATLAB and an HDL simulator, such as Mentor Graphics ModelSim or Cadence Incisive.
- hdlverifier.FILSimulation class: For FPGA-in-the-Loop simulation with MATLAB.

Configure Components for Your System

When to Configure Components

If you did not set an object's properties when you created it and do not want to use default values, you must explicitly set those properties. Some properties allow you to

change their values while your system is running. See "Reconfigure Your System During Runtime" on page 4-12 for information.

Most properties are independent of each other. However, some System object properties enable or disable another property or limit the values of another property. To avoid errors or warnings, you should set the controlling property before setting the dependent property.

Display Component Property Values

To display the current property values for an object, type that object's handle name at the command line (such as `audioIn`). To display the value of a specific property, type `objecthandle.propertyname` (such as `audioIn.FileName`).

Configure Component Property Values

This example shows how to set properties for HDL.

With the HDL cosimulation object created through the HDL cosim wizard or the HDL workflow advisor, you can set properties for connection, input signals, output signals, etc.

```
fft_hdl = hdlcosim_fft_hdl;
fft_hdl.Connection = ('SharedMemory';
fft_hdl.InputSignals = '/viterbi_block/In1','/viterbi_block/In2';
fft_hdl.OutputSignals = '/viterbi_block/Out1';
```

See the `hdlverifier.HdlCosimulation` ref page for full details and a list of properties.

Assemble Components to Create Your System

Connect Inputs and Outputs

After you have determined the components you need and have created and configured your System objects, assemble your system. You use the System objects like other MATLAB variables and include them in MATLAB code. You can pass MATLAB variables into and out of System objects.

The main difference between using System objects and using functions is the `step` method. The `step` method is the processing command for each System object and is customized for that specific System object. This method initializes your objects and controls data flow and state management of your system. You typically use `step` within a loop.

You use the output from an object's `step` method as the input to another object's `step` method. For some System objects, you can use properties of those objects to change the number of inputs or outputs. To verify that the appropriate number of input and outputs are being used, you can use `getNumInputs` and `getNumOutputs` on any System object. For information on all available System object methods, see "System Object Methods" on page 4-7.

Run Your System

- "How to Run Your System" on page 4-12
- "What You Cannot Change While Your System Is Running" on page 4-12

How to Run Your System

Run your code either by typing directly at the command line or running a file containing your program. When you run the code for your system, the `step` method instructs each object to process data through that object.

What You Cannot Change While Your System Is Running

The first call to the `step` method initializes and then locks your object. When a System object has started processing data, it is locked to prevent changes that would disrupt its processing. Use the `isLocked` method to verify whether an object is locked. When the object is locked, you cannot change:

- Number of inputs or outputs
- Data type of inputs or outputs
- Data type of any tunable property
- Dimensions of inputs or tunable properties, except for System objects that support variable-size data
- Value of any nontunable property

To make changes to your system while it is running, see "Reconfigure Your System During Runtime" on page 4-12.

Reconfigure Your System During Runtime

- "When Can You Change Component Properties?" on page 4-13
- "Change Input Complexity or Dimensions" on page 4-13

When Can You Change Component Properties?

When a System object has started processing data, it is locked to prevent changes that would disrupt its processing. You can use `isLocked` on any System object to verify whether it is locked or not. When processing is complete, you can use the `release` method to unlock a System object.

Some object properties are *tunable*, which enables you to change them even if the object is locked. Unless otherwise specified, System objects properties are nontunable. Refer to the object's reference page to determine whether an individual property is tunable. Typically, tunable properties are not critical to how the System object processes data.

Change Input Complexity or Dimensions

During simulation, some System objects do not allow complex data if the object was initialized with real data. You cannot change any input complexity during code generation.

You can change the value of a tunable property without a warning or error being produced. For all other changes at run time, an error occurs.