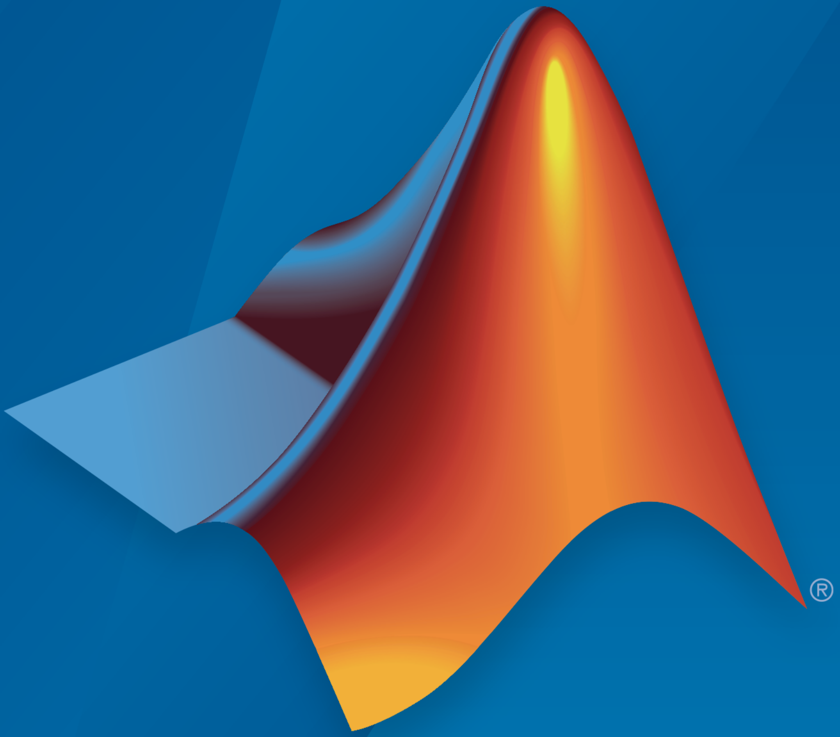
HDL Verifier™

Getting Started Guide



MATLAB® & SIMULINK®

R2016b

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

HDL Verifier™ Getting Started Guide

Revision History

August 2003	Online only	New for Version 1 (Release 13SP1)
February 2004	Online only	Revised for Version 1.1 (Release 13SP1)
June 2004	Online only	Revised for Version 1.1.1 (Release 14)
October 2004	Online only	Revised for Version 1.2 (Release 14SP1)
December 2004	Online only	Revised for Version 1.3 (Release 14SP1+)
March 2005	Online only	Revised for Version 1.3.1 (Release 14SP2)
September 2005	Online only	Revised for Version 1.4 (Release 14SP3)
March 2006	Online only	Revised for Version 2.0 (Release 2006a)
September 2006	Online only	Revised for Version 2.1 (Release 2006b)
March 2007	Online only	Revised for Version 2.2 (Release 2007a)
September 2007	Online only	Revised for Version 2.3 (Release 2007b)
March 2008	Online only	Revised for Version 2.4 (Release 2008a)
October 2008	Online only	Revised for Version 2.5 (Release 2008b)
March 2009	Online only	Revised for Version 2.6 (Release 2009a)
September 2009	Online only	Revised for Version 3.0 (Release 2009b)
March 2010	Online only	Revised for Version 3.1 (Release 2010a)
September 2010	Online only	Revised for Version 3.2 (Release 2010b)
April 2011	Online only	Revised for Version 3.3 (Release 2011a)
September 2011	Online only	Revised for Version 3.4 (Release 2011b)
March 2012	Online only	Revised for Version 4.0 (Release 2012a)
September 2012	Online only	Revised for Version 4.1 (Release 2012b)
March 2013	Online only	Revised for Version 4.2 (Release 2013a)
September 2013	Online only	Revised for Version 4.3 (Release 2013b)
March 2014	Online only	Revised for Version 4.4 (Release 2014a)
October 2014	Online only	Revised for Version 4.5 (Release 2014b)
March 2015	Online only	Revised for Version 4.6 (Release 2015a)
September 2015	Online only	Revised for Version 4.7 (Release 2015b)
September 2016	Online only	Revised for Version 5.1 (Release 2016b)

Contents

3 Third-Party Product Requirements

4 System Objects

1

Introduction

HDL Verifier Product Description

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

HDL Verifier™ automatically generates test benches for Verilog® and VHDL® design verification. You can use MATLAB® or Simulink® to directly stimulate your design and then analyze its response using HDL cosimulation or FPGA-in-the-loop with Xilinx® and Altera® FPGA boards. This approach eliminates the need to author standalone Verilog or VHDL test benches.

HDL Verifier also generates components that reuse MATLAB and Simulink models natively in simulators from Cadence®, Mentor Graphics®, and Synopsys®. These components can be used as verification checker models or as stimuli in more complex test-bench environments such as those that use the Universal Verification Methodology (UVM).

Key Features

- Cosimulation with Cadence Incisive®, Mentor Graphics ModelSim®, or Questa®
- FPGA-in-the-loop verification using Xilinx and Altera FPGA boards
- SystemVerilog DPI component generation from MATLAB functions and Simulink blocks
- Generation of IEEE® 1666 SystemC TLM 2.0 compatible transaction-level models
- Automated verification workflow with HDL Coder™

About HDL Verifier

HDL Cosimulation

In this section...

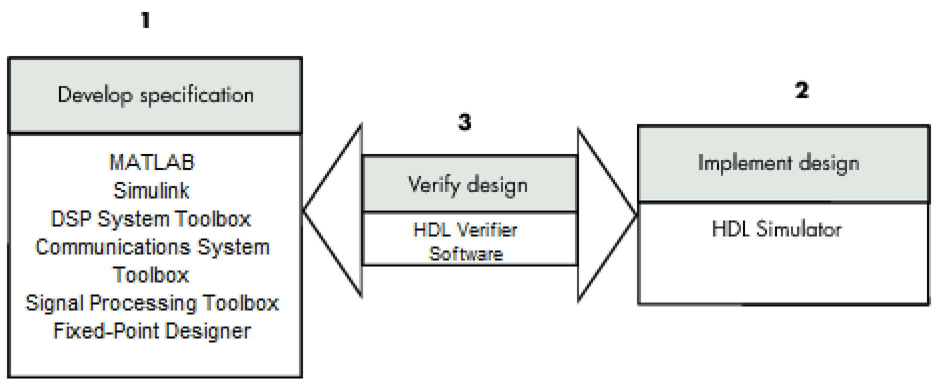
HDL Cosimulation with MATLAB or Simulink

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

---

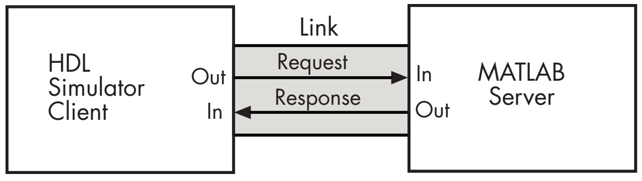
Note: You can cosimulate a module using SystemVerilog, SystemC or both with MATLAB or Simulink using the HDL Verifier software. Write simple wrappers around the SystemC and make sure that the SystemVerilog cosimulation connections are to ports or signals of data types supported by the link cosimulation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

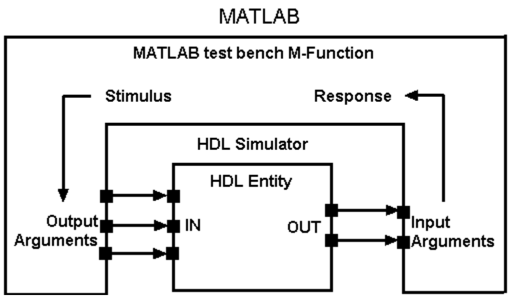


In this scenario, a MATLAB server function waits for service requests that it receives from an HDL simulator session. After receiving a request, the server establishes a communication link and invokes a specified MATLAB function that computes data for, verifies, or visualizes the HDL module (coded in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

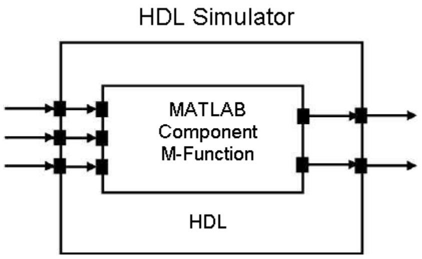
After the server is running, you can start and configure the HDL simulator or use with MATLAB with the supplied HDL Verifier function:

- `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.

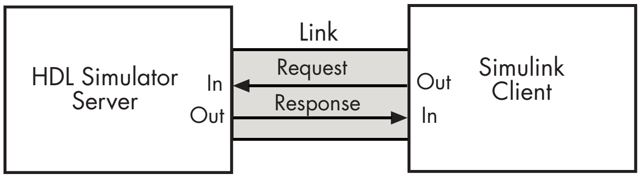


When you begin a specific test bench or component session, you specify parameters that identify the following information:

- 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 for Cosimulation”.

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

In this section...
"FPGA Verification with HDL Verifier and HDL Coder" on page 2-8
"Product Features and Platform Support" on page 2-8

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

Product Feature	Required Products	Recommended Products	Supported Platforms
FPGA-in-the-Loop	For FIL simulation with MATLAB:	HDL Coder	Windows 64-bit; Linux 64-bit

Product Feature	Required Products	Recommended Products	Supported Platforms
	MATLAB, Fixed-Point Designer For FIL simulation with Simulink: Simulink, Fixed-Point Designer		

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. To download an FPGA board support package:

- On the MATLAB **Home** tab, in the **Environment** section, click **Add-Ons** > **Get Hardware Support Packages**.

TLM Component Generation

In this section...

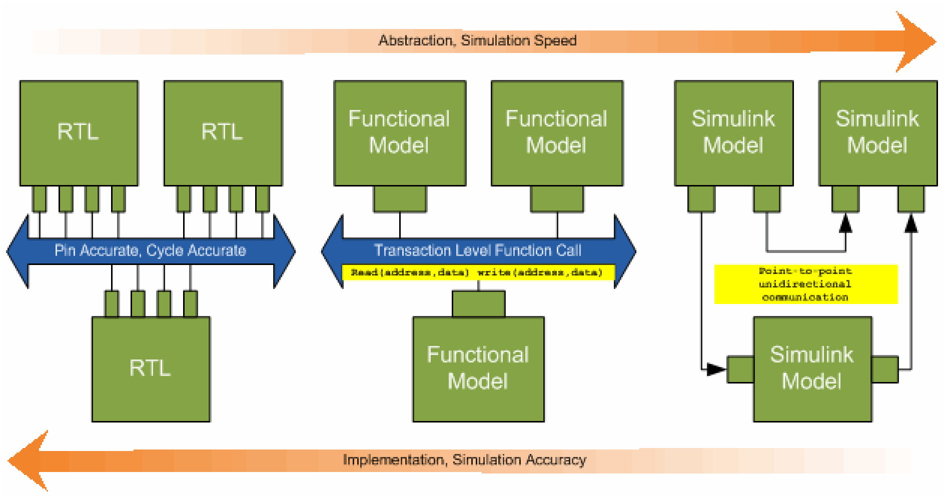
Generating TLM Components for Virtual Platform Development

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.


Abstraction, Simulation Speed
RTL
RTL
Functional Model
Functional Model
Simulink Model
Simulink Model
Pin Accurate, Cycle Accurate
Transaction Level Function Call
Read(address,data) write(address,data)
Point-to-point unidirectional communication
RTL
Functional Model
Simulink Model
Implementation, Simulation Accuracy

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-compliant SystemC Transaction Level Model (TLM) that can be executed in any OSCI-compatible TLM 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

In this section...

Export Simulink Subsystem or MATLAB Function Using DPI Interface

You can export a Simulink subsystem or MATLAB function with a DPI interface for Verilog or SystemVerilog simulation. The coder wraps generated C code with a DPI wrapper accessed through a SystemVerilog thin interface function.

- Simulink subsystem — Access this feature from the Model Configuration Parameters dialog box, under **Code Generation**. See "Generate SystemVerilog DPI Component".
- MATLAB function — Generate the component using the `dpigen` function. See "Generate DPI Component Using MATLAB".

HDL Verifier supports SystemVerilog DPI component generation with these products and platforms.

Design Format	Required Products	Recommended Products	Supported Platforms
Simulink subsystem	Simulink and Simulink Coder	Embedded Coder	• Windows 32-bit and 64-bit • Linux 64-bit
MATLAB function	MATLAB and MATLAB Coder		• Windows 64-bit • Linux 64-bit

Generate SystemVerilog DPI Test Bench in HDL Coder

If you have an HDL Coder license, you can generate a SystemVerilog DPI test bench. Use the test bench to verify your generated HDL code using C code generated from your entire Simulink model, including the DUT and data sources. To use this feature, your entire model must support C code generation with Simulink Coder. See the `GenerateSVDPITestBench` property of `makehdltb`.

HDL Verifier supports SystemVerilog DPI test bench generation in HDL Coder with these products and platforms.

Design Format	Required Products	Recommended Products	Supported Platforms
Simulink subsystem	Simulink and Simulink Coder	Embedded Coder	• Windows 32-bit and 64-bit • Linux 64-bit

More About

- “DPI Component Generation with Simulink”
- “DPI Component Generation with MATLAB”

HDL Verifier Supported Hardware

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

Support Package	Vendor	Earliest Release Available	Last Release Available
Altera FPGA Boards	Altera	R2013a	Current
Xilinx FPGA Boards	Xilinx	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

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

To get started, see "Set Up MATLAB-HDL Simulator Connection" or "Start HDL Simulator for Cosimulation in Simulink".

Cadence Incisive Requirements

MATLAB and Simulink support Cadence verification tools using HDL Verifier. Only the 64-bit version of Incisive is supported for cosimulation. Use one of these recommended versions, which have been fully tested against the current release:

- Incisive 13.2 p002
- Incisive 13.1 s006
- Incisive 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 10.4c, 10.3, 10.2c
- ModelSim/QuestaSim PE 10.4c, 10.3e, 10.2c

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 3-3
- “Supported FPGA Devices for FIL Simulation” on page 3-5
- “Supported FPGA Device Families for Board Customization” on page 3-8

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® 2015.4.

  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.

For tool setup instructions, see “Set Up FPGA Design Software Tools”.

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 15.1.

For tool setup instructions, see “Set Up FPGA Design Software Tools”.

Supported FPGA Board Connections for FIL Simulation

For board support, see “Supported FPGA Devices for FIL Simulation” on page 3-5.

Additional boards can be custom added with the “FPGA Board Manager”. See “Supported FPGA Device Families for Board Customization” on page 3-8.

JTAG Connection

Vendor	Supported Devices	Required Hardware	Required Software
Altera	The FPGA board must be using an FPGA device in the supported Altera FPGA families.	• USB Blaster I or USB Blaster II download cable	• USB Blaster I or II driver • For Windows operating systems: Quartus II executable directory must be on system path. • For Linux operating systems: versions below Quartus II 13.1 are not supported. Quartus II 14.1 is not supported. Only 64-bit Quartus II is supported. Quartus II library directory must be on `LD_LIBRARY_PATH` *before* starting MATLAB. Prepend the Linux distribution library path before the Quartus II library on `LD_LIBRARY_PATH`. For example, `/lib/x86_64-linux-gnu:$QUARTUS_PATH`.
Xilinx	The board must be using one of the following supported Xilinx FPGAs: Artix®-7, Virtex-7, Kintex®-7 or Zynq® 7000.	• 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 executable directory must be on system path. • For Linux operating systems: Digilent Adept2

Note: When simulating your FPGA design through Digilent JTAG cable with Simulink or MATLAB, you can not use any debugging software that requires access to the JTAG; for example,Vivado Logic Analyzer.

Ethernet Connection

Required Hardware	Supported Interfaces	Required Software
• Gigabit Ethernet card • Cross-over Ethernet cable • FPGA board with supported Ethernet connection	• Gigabit Ethernet — GMII • Gigabit Ethernet — RGMII • Gigabit Ethernet — SGMII • Ethernet — MII	There are no software requirements for an Ethernet connection, but ensure that the firewall on the host computer does not prevent UDP communication. **Note:** Ethernet connection to Virtex-7 VC707 not supported for Vivado versions older than 2013.4.

PCI Express

Note: FIL over PCI Express® connection is supported only for 64-bit Windows operating systems.

Device Family	Board	Required Software
Xilinx	• Kintex-7 KC705 Evaluation Kit • Virtex -7 VC707 Evaluation Kit	Vivado 2015.2
Altera	• Cyclone® V GT FPGA Development Kit • DSP Development Kit, Stratix® V Edition	AlteraQuartus II 15.0

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 can add other FPGA boards for use with FIL with FPGA board customization (“FPGA Board Customization”).

Device Family	Board	Comments
Xilinx Artix-7	Digilent Nexys™4 Artix-7	
Xilinx Kintex-7	Kintex-7 KC705	
Xilinx Kintex UltraScale™	Kintex UltraScale FPGA KCU105 Evaluation Kit	Supports JTAG connection only.
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	
Xilinx Virtex-5	Virtex ML505 Virtex ML506 Virtex ML507 Virtex XUPV5–LX110T	
Xilinx Virtex	Virtex ML401 Virtex ML402 Virtex ML403	
Xilinx Zynq	Zynq-7000 ZC702 Zynq-7000 ZC706 ZedBoard™ ZYBO™ Zynq-7000 Development Board	Xilinx Zynq boards support a JTAG connection only.
Altera Arria® II	Arria II GX FPGA Development Kit	
Altera Arria V	Arria V SoC Development Kit Arria V Starter Kit	The Altera Arria V SoC development kit supports a JTAG connection only.
Altera Arria 10	Arria 10 SoC Development Kit	

Device Family	Board	Comments
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 Starter Kit Cyclone III FPGA Development Kit Altera Nios II Embedded Evaluation Kit, Cyclone III Edition	The Altera Cyclone III FPGA starter kit supports a JTAG connection only.
Altera Cyclone V	Cyclone V GX FPGA Development Kit Cyclone V SoC Development Kit Cyclone V GT Development Kit Arrow® SoCKit Development Kit	The Cyclone V SoC and Arrow SoCKit development kits are supported for JTAG connection only.
Altera MAX® 10	Arrow MAX 10 DECA	
Altera Stratix IV	Stratix IV GX FPGA Development Kit	
Altera Stratix V	DSP Development Kit, Stratix V Edition	

Limitations

- 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 can download one or more vendor-specific packages. To use FIL, download at least one of these packages, or customize your own board definition file. See “Create Custom FPGA Board Definition”.

Visit the Hardware Support Catalog for a list of currently supported devices and boards. To download an FPGA board support package:

- On the MATLAB **Home** tab, in the **Environment** section, click **Add-Ons** > **Get Hardware Support Packages**.

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		Restrictions
Xilinx	Artix 7	
	Kintex 7	
	Kintex UltraScale	Supports JTAG connection only. Ethernet is not supported.
	Spartan 6	Ethernet PHY RGMII is not supported.
	Virtex 4	
	Virtex 5	
	Virtex 6	
	Virtex 7	Supports Ethernet PHY SGMII only.
	Zynq 7000	
Altera	Arria II	
	Arria V	
	Arria 10	
	Cyclone III	
	Cyclone IV	
	Cyclone V	
	MAX 10	
	Stratix IV	
	Stratix V	

DPI Component Generation Requirements

DPI component generation supports the same versions of Cadence Incisive and Mentor Graphics Questa and ModelSim as for cosimulation. You can generate a DPI component for use with either 64-bit or 32-bit Incisive.

TLM Generation Requirements

With the current release, TLMG includes support for:

- Compilers:
  - Visual Studio®: VS2008, VS2010, VS2012, and VS2013
  - Windows 7.1 SDK
  - 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 website for information about how to build these libraries after downloading.
- System C Modeling Library (SCML):
  - SCML 2.2

    You can download SCML from https://www.synopsys.com.

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

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.

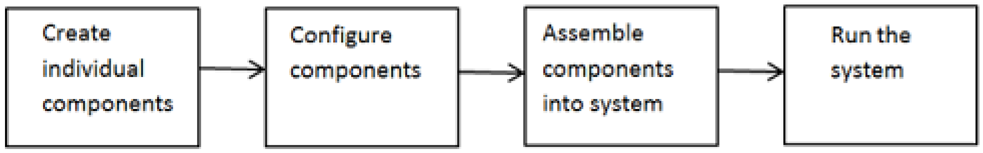
Building a dynamic system with different execution phases and internal states using only MATLAB functions would require complex programming. You would need code to initialize the system, validate data, manage internal states, and reset and terminate the system. System objects perform many of these managerial 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 4-9.

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 “Component Properties” on page 4-10.

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 “Connecting System Objects” on page 4-10.

4 Run the system — Run your program. You can change tunable properties while your system is running. See “Run Your System” on page 4-11 and “Reconfiguring Objects” on page 4-11.

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 4-6
“Running a System Object” on page 4-6
“Common Methods” on page 4-7

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. The syntax for using methods is `<method>(<system object name>)`, plus possible extra input arguments. For example, for `txfourier = dsp.FFT`, where `txfourier` is a name you assign, you call the `reset` method using `reset(txfourier)`. For more information about methods, see the descriptions in “Common Methods” on page 4-7.

Running a System Object

To run a System object and perform the operation defined by its algorithm, you call the object as if it were a function. For example, to create an FFT object that uses the `dsp.FFT` System object, specifies a length of 1024, and names it `dft`, use

```
dft = dsp.FFT;
```

To run this object on input `x`, use

```
dft(x);
```

If you run a System object without any input arguments, you must include empty parentheses. For example, `asysobj()`.

When you run a System object, it also performs other important tasks related to data processing, such as initialization and handling object states.

Note: An alternative way to run a System object is to use the `step` method. For example, for an object created using `dft = dsp.FFT`, you can run it using `step(dft,x)`.

Common Methods

All System objects support the following methods, each of which is linked from each object's reference page. In cases where a method is not applicable to a particular object, calling that method has no effect on the object.

Method	Description
Run System object like a function, or `step`	Run the object to process data using the algorithm defined by that object. Example: For `dft = dsp.FFT;`, the standard way to run the object is `y = dft(x)`. The alternative way is `y = step(dft,x)`. As part of this processing, the object initializes needed resources, returns outputs, and updates the object states. After you start running the object, you cannot change any input specifications (i.e., dimensions, data type, complexity). During execution, you can change only tunable properties. Both ways of running a System object return regular MATLAB variables.
`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 running the object. This number varies for an object depending on whether any properties enable additional inputs.
`getNumOutputs`	Returns the number of outputs expected from running the object. 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 it or after you have released the object), the states are empty. If the object has no discrete states, `getDiscreteState` returns an empty structure.

Method	Description
`clone`	Creates another object of the same type with the same property values
`isLocked`	Returns a logical value indicating whether the object is locked.
`isDone`	Applies to source objects only. Returns a logical value indicating whether the end of the data file has been reached. 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...

Predefined Components

The example in the next section shows how to use System objects that are predefined in the software. 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.

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: For FPGA-in-the-Loop simulation with MATLAB.

First, generate these objects using HDL Verifier tools, then use these objects as components in your system. Each of these System objects has default property settings, so you can create instances of them using just the object name without any input properties.

Create Components for Your System

HDL Verifier provides System objects to support HDL cosimulation and FPGA-in-the-loop (FIL) simulation. To create a System object for cosimulation, see "Create a MATLAB System Object". To create a System object for FIL simulation, see "System Object Generation with the FIL Wizard".

Component Properties

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 "Reconfiguring Objects" on page 4-11 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

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.

Connecting System Objects

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 that System objects use a two-step process. First you create the object and set its parameters and then, you run the object. Running the object initializes it and controls the data flow and state management of your system. You typically call a System object within a code loop.

You use the output from an object as the input to another object. 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 4-6.

Run Your System

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, data is processed through your objects.

What You Cannot Change While Your System Is Running

The first call to a System object initializes and then locks the 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 "Reconfiguring Objects" on page 4-11.

Reconfiguring Objects

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.