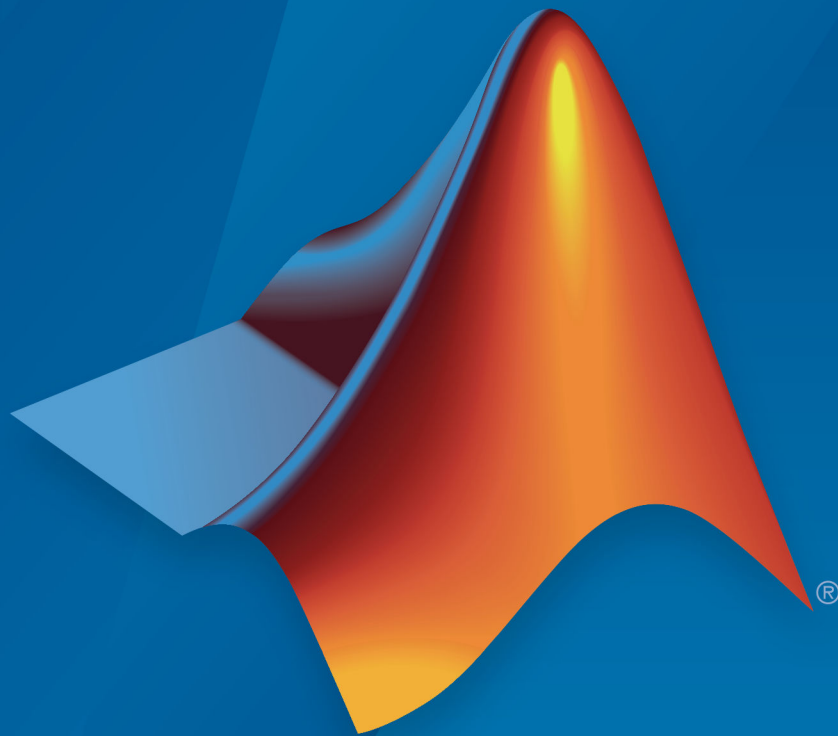


HDL Coder™

Getting Started Guide



MATLAB® & SIMULINK®

R2018b



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 Coder™ Getting Started Guide

© COPYRIGHT 2012–2018 by The MathWorks, Inc.

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

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

Trademarks

MATLAB 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.

Revision History

March 2012	Online only	New for Version 3.0 (Release 2012a)
September 2012	Online only	Revised for Version 3.1 (Release 2012b)
March 2013	Online only	Revised for Version 3.2 (Release 2013a)
September 2013	Online only	Revised for Version 3.3 (Release 2013b)
March 2014	Online only	Revised for Version 3.4 (Release 2014a)
October 2014	Online only	Revised for Version 3.5 (Release 2014b)
March 2015	Online only	Revised for Version 3.6 (Release 2015a)
September 2015	Online only	Revised for Version 3.7 (Release 2015b)
October 2015	Online only	Rereleased for Version 3.6.1 (Release 2015aSP1)
March 2016	Online only	Revised for Version 3.8 (Release 2016a)
September 2016	Online only	Revised for Version 3.9 (Release 2016b)
March 2017	Online only	Revised for Version 3.10 (Release 2017a)
September 2017	Online only	Revised for Version 3.11 (Release 2017b)
March 2018	Online only	Revised for Version 3.12 (Release 2018a)
September 2018	Online only	Revised for Version 3.13 (Release 2018b)

About HDL Coder

1

HDL Coder Product Description	1-2
Key Features	1-2
Supported Third-Party Tools and Hardware	1-3
Third-Party Synthesis Tools and Version Support	1-3
FPGA-in-the-Loop Hardware	1-3
Simulink Real-Time FPGA I/O: Speedgoat Target Hardware ..	1-4
Generic ASIC/FPGA Hardware	1-4
IP Core Generation Hardware	1-5
FPGA Turnkey Hardware	1-6
VHDL and Verilog Language Support	1-8
HDL Coder Supported Hardware	1-9

Getting Started with HDL Coder

2

Tool Setup	2-2
Synthesis Tool Path Setup	2-2
HDL Simulator Setup	2-3
Xilinx System Generator Setup for ModelSim Simulation	2-4
Altera DSP Builder Setup	2-5
FPGA Simulation Library Setup	2-5
C/C++ Compiler Setup	2-6

HDL Code Generation and FPGA Synthesis from a MATLAB Algorithm	3-2
About the Algorithm in This Example	3-2
Create Local Copy of Design and Testbench Files	3-2
Set Up Synthesis Tool Path	3-3
Test the Original MATLAB Algorithm	3-3
Set Up a Project Using HDL Coder App	3-4
Open the HDL Coder Workflow Advisor	3-6
Create Fixed-Point Versions of the Algorithm and Test Bench	3-7
Generate HDL Code	3-9
Verify Generated HDL Code	3-9
FPGA Synthesis and Implementation	3-9
Create HDL-Compatible Simulink Model	3-11
Open Model and HDL Coder Library	3-11
Develop Design and Test Bench	3-12
Configure Model for HDL Compatibility	3-15
Check Subsystem for HDL Compatibility	3-16
Run Model Advisor Checks for HDL Coder	3-18
Generate HDL Code	3-19
Generate HDL Code from Simulink Model	3-20
The sfir_fixed Model	3-20
Generate HDL Code	3-22
View HDL Code Generation Files	3-24
HDL Code Generation and FPGA Synthesis Using the HDL Workflow Advisor	3-26
About HDL Workflow Advisor	3-26
Set Up Tool Path	3-27
Open the HDL Workflow Advisor	3-27
Generate HDL Code	3-28
Perform FPGA Synthesis and Analysis	3-30
Run Workflow at Command Line with a Script	3-31
Verify Generated Code from Simulink Model Using HDL Test Bench	3-33
How to Verify the Generated Code	3-33
What is a HDL Test Bench?	3-33

Generate HDL Test Bench	3-34
View HDL Test Bench Files	3-35
Run Simulation and Verify Generated HDL Code	3-36

About HDL Coder

- “HDL Coder Product Description” on page 1-2
- “Supported Third-Party Tools and Hardware” on page 1-3
- “VHDL and Verilog Language Support” on page 1-8
- “HDL Coder Supported Hardware” on page 1-9

HDL Coder Product Description

Generate VHDL and Verilog code for FPGA and ASIC designs

HDL Coder generates portable, synthesizable VHDL[®] and Verilog[®] code from MATLAB[®] functions, Simulink[®] models, and Stateflow[®] charts. The generated HDL code can be used for FPGA programming or ASIC prototyping and design.

HDL Coder provides a workflow advisor that automates the programming of Xilinx[®], Microsemi[®], and Intel[®] FPGAs. You can control HDL architecture and implementation, highlight critical paths, and generate hardware resource utilization estimates. HDL Coder provides traceability between your Simulink model and the generated Verilog and VHDL code, enabling code verification for high-integrity applications adhering to DO-254 and other standards.

Key Features

- Target-independent, synthesizable VHDL and Verilog code
- Code generation support for MATLAB functions, System objects and Simulink blocks
- Mealy and Moore finite-state machines and control logic implementations using Stateflow
- Workflow advisor for programming Xilinx, Microsemi, and Intel application boards
- Resource sharing and retiming for area-speed tradeoffs
- Code-to-model and model-to-code traceability for DO-254
- Legacy code integration

Supported Third-Party Tools and Hardware

In this section...

“Third-Party Synthesis Tools and Version Support” on page 1-3

“FPGA-in-the-Loop Hardware” on page 1-3

“ Simulink Real-Time FPGA I/O: Speedgoat Target Hardware” on page 1-4

“Generic ASIC/FPGA Hardware” on page 1-4

“IP Core Generation Hardware” on page 1-5

“FPGA Turnkey Hardware” on page 1-6

Third-Party Synthesis Tools and Version Support

The HDL Workflow Advisor is tested with the following third-party FPGA synthesis tools:

- Intel Quartus Prime Standard Edition 17.1
- Xilinx Vivado® Design Suite 2017.4
- Microsemi Libero® SoC 11.8
- Xilinx ISE 14.7

To use third-party synthesis tools with HDL Coder, a supported synthesis tool must be installed, and the synthesis tool executable must be on the system path. For details, see “Tool Setup” on page 2-2.

FPGA-in-the-Loop Hardware

The FPGAs supported for FPGA-in-the-loop simulation with HDL Verifier™ are listed in the HDL Verifier documentation.

You can also add custom FPGA boards using the FPGA Board Manager. See “FPGA Board Customization” for details.

For FPGA-in-the-Loop or Customization for USRP® Device using the HDL Workflow Advisor, a supported synthesis tool must be installed, and the synthesis tool executable must be on the system path. For details, see “Tool Setup” on page 2-2.

Simulink Real-Time FPGA I/O: Speedgoat Target Hardware

Speedgoat I/O Module	FPGA Device	Synthesis Tool
IO342	Xilinx Kintex UltraScale	For more information and to learn about the synthesis tool requirements, see Xilinx HDL Support with Speedgoat IO Modules.
IO333-325K, IO334, IO325	Xilinx Kintex-7	
IO332, IO397	Xilinx Artix-7	
IO332, IO331, IO331-6	Xilinx Spartan-6	

Generic ASIC/FPGA Hardware

The following hardware is supported for the Generic ASIC/FPGA workflow:

Synthesis Tool	Device Family
Xilinx Vivado	Kintex7
	Artix7
	Kintex UltraScale+
	KintexU
	Spartan7
	Virtex UltraScale+
	Virtex7
	VirtexU
	Zynq
Zynq UltraScale+	
Xilinx ISE	Virtex6
	Virtex5
	Virtex4
	Spartan-3A DSP
	Spartan 3E

Synthesis Tool	Device Family
	Spartan3
	Spartan6
Altera® Quartus II	Cyclone® III
	Cyclone IV
	Arria® II GX and GZ
	Stratix® IV
	Stratix V
	Cyclone III
	Arria 10
	Arria V GX
	MAX 10
Microsemi Libero SoC	SmartFusion2 SoC
	RTG4
	IGLOO2

IP Core Generation Hardware

The following hardware is supported for the IP Core Generation workflow:

Synthesis Tool	Target Platform
Xilinx Vivado	Zedboard and with FMC-HDMI-CAM and FMCOMMS2/3/4/
	ZC706 and with FMC-HDMI-CAM and FMCOMMS2/3/4/
	ZC702 with FMC-HDMI-CAM
	Zynq ZC706 evaluation kit
	Zynq ZC702 evaluation kit
	PicoZed FMC-HDMI-CAM
Altera Quartus II	Arria 10 SoC development kit

Synthesis Tool	Target Platform
	Cyclone V SoC development kit Rev. C and Rev. D
	Arrow DECA Max 10 FPGA development board
	Arrow SoCKit development board
	Arria 10 GX FPGA development kit

FPGA Turnkey Hardware

The following hardware is supported for the FPGA Turnkey workflow:

- Altera Arria II GX FPGA development kit
- Altera Cyclone III FPGA development kit
- Altera Cyclone IV GX FPGA development kit
- Altera DE2-115 development and education board
- XUP Atlys Spartan-6 development board
- Xilinx Spartan-3A DSP 1800A development board
- Xilinx Spartan-6 SP605 development board
- Xilinx Virtex-4 ML401 development board
- Xilinx Virtex-4 ML402 development board
- Xilinx Virtex-5 ML506 development board
- Xilinx Virtex-6 ML605 development board

For FPGA development boards that have more than one FPGA device, only one such device can be used with FPGA Turnkey.

Supported FPGA Device Families for Board Customization

You can also add custom FPGA boards using the FPGA Board Manager. HDL Coder supports the following FPGA device families for board customization; that is, when you create your own board definition file. See “FPGA Board Customization” (HDL Verifier).

Device Family	
Xilinx	Kintex7
	Spartan-3A DSP
	Spartan3
	Spartan3A and Spartan3AN
	Spartan3E
	Spartan6
	Virtex4
	Virtex5
	Virtex6
	Virtex7
Altera	Cyclone III
	Cyclone IV
	Arria II
	Stratix IV
	Stratix V

See Also

More About

- “Tool Setup” on page 2-2

VHDL and Verilog Language Support

The generated HDL code complies with the following standards:

- VHDL-1993 (IEEE® 1076-1993) or later
- Verilog-2001 (IEEE 1364-2001) or later

HDL Coder Supported Hardware



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

Support Package	Vendor	Earliest Release Available	Last Release Available
Intel FPGA Boards	Intel	R2013b	Current
Intel SoC Devices	Intel	R2014b	Current
Xilinx FPGA Boards	Xilinx	R2013b	Current
Xilinx Zynq Platform	Xilinx	R2013a	Current

For a complete list of support packages, see [Hardware Support](#).

In addition to these packages, HDL Coder includes built-in support for:

- FPGA-in-the-loop simulation with HDL Verifier
- Simulink Real-Time™ FPGA I/O hardware
- Custom FPGA boards using the FPGA Board Manager

For details, see “Supported Third-Party Tools and Hardware” on page 1-3.

Getting Started with HDL Coder

Tool Setup

In this section...

“Synthesis Tool Path Setup” on page 2-2

“HDL Simulator Setup” on page 2-3

“Xilinx System Generator Setup for ModelSim Simulation” on page 2-4

“Altera DSP Builder Setup” on page 2-5

“FPGA Simulation Library Setup” on page 2-5

“C/C++ Compiler Setup” on page 2-6

Synthesis Tool Path Setup

- “hdlsetuptoolpath Function” on page 2-2
- “Add Synthesis Tool for Current HDL Workflow Advisor Session” on page 2-2
- “Check Your Synthesis Tool Setup” on page 2-3
- “Supported Tool Versions” on page 2-3

hdlsetuptoolpath Function

To use HDL Coder with one of the supported third-party FPGA synthesis tools, add the tool to your system path using the `hdlsetuptoolpath` function. Add the tool to your system path before opening the HDL Workflow Advisor. If you already have the HDL Workflow Advisor open, see “Add Synthesis Tool for Current HDL Workflow Advisor Session” on page 2-2.

Add Synthesis Tool for Current HDL Workflow Advisor Session

Simulink to HDL Workflow

- 1 At the MATLAB command line, use the `hdlsetuptoolpath` function to add the synthesis tool.
- 2 In the HDL Workflow Advisor, in the **Set Target > Set Target Device and Synthesis Tool** step, to the right of **Synthesis tool**, click **Refresh**.

The synthesis tool is now available.

MATLAB to HDL Workflow

- 1 At the MATLAB command line, use the `hdlsetuptoolpath` function to add the synthesis tool.
- 2 In the HDL Workflow Advisor, in the **Select Code Generation Target** step, to the right of **Synthesis tool**, click **Refresh list**.

The synthesis tool is now available.

Check Your Synthesis Tool Setup

To check your Altera Quartus synthesis tool setup in MATLAB, try launching the tool with the following command:

```
!quartus
```

To check your Xilinx Vivado synthesis tool setup in MATLAB, try launching the tool with the following command:

```
!vivado
```

To check your Xilinx ISE synthesis tool setup in MATLAB, try launching the tool with the following command:

```
!ise
```

To check your Microsemi Libero SoC synthesis tool setup in MATLAB, try launching the tool with the following command:

```
!libero
```

Supported Tool Versions

For supported tool versions, see “Third-Party Synthesis Tools and Version Support” on page 1-3.

HDL Simulator Setup

To open the HDL simulator from MATLAB, enter these commands:

MATLAB Command to Open HDL Simulator

HDL Simulator	Command to Open the Simulator
Cadence Incisive®	nclaunch
Mentor Graphics® ModelSim®	vsim

For example, to open the Mentor Graphics ModelSim simulator, enter this command:

```
vsim('vsimdir', 'C:\Program Files\ModelSim\questasim\10.5c\win64\vsim.exe')
```

To learn more about how to set up ModelSim, Questa®, or Incisive® for HDL simulation, or for cosimulation with HDL Verifier, see “HDL Simulator Startup” (HDL Verifier).

Add Simulation Tool for Current HDL Workflow Advisor Session

MATLAB to HDL Workflow

- 1 Set up your simulation tool.
- 2 In the HDL Workflow Advisor, in the **HDL Verification > Verify with HDL Test Bench** task, click **Refresh list**.

The simulation tool is now available.

Xilinx System Generator Setup for ModelSim Simulation

To generate ModelSim simulation scripts for a design containing Xilinx System Generator blocks, you must:

- Have compiled Xilinx simulation libraries.
- Specify the path to your compiled libraries.

Required Libraries for Vivado and ISE

To generate ModelSim simulation scripts, you must have the following compiled Xilinx simulation libraries for your EDA simulator and target language:

- unisim
- simprim
- xilinxcorelib

To learn how to compile these libraries, refer to the Xilinx documentation.

- For Vivado, see `compile_simlib`.
- For ISE, see `compplib`.

Specify Path to Required Libraries

Specify the path to your compiled Xilinx simulation libraries by setting the `XilinxSimulatorLibPath` parameter for your model.

For example, you can use `hdlset_param` to set `XilinxSimulatorLibPath`:

```
libpath = '/apps/Xilinx_ISE/XilinxISE-13.4/Linux/ISE_DS/ISE/vhdl/  
mti_se/6.6a/lin64/xilinxcorelib';  
hdlset_param (bdroot, 'XilinxSimulatorLibPath', libpath);
```

Altera DSP Builder Setup

To generate code for a design containing both Altera DSP Builder and Simulink blocks, you must open MATLAB with Altera DSP Builder. For details, refer to the Altera DSP Builder documentation.

FPGA Simulation Library Setup

To map your design to an Altera or a Xilinx FPGA simulator library:

- Use Xilinx LogiCORE® IP Floating-Point Operator v5.0 or Altera floating-point megafunction IP cores.
- Specify the compiled simulation library and the target language for your EDA simulator. Use `XilinxCoreLib` simulation library for Xilinx LogiCORE IP and the EDA simulation library compiler for Altera megafunction IP.

To learn how to compile this library, refer to the Xilinx `compplib` documentation .

- Specify the path to your compiled Altera or Xilinx simulation libraries. Altera provides the simulation model files in `\quartus\eda\sim_lib` folder. Set the `SimulationLibPath` parameter for your DUT.

For example, you can use `hdlset_param` to set `SimulationLibPath`:

```
myDUT = gcb;  
libpath = '/apps/Xilinx_ISE/XilinxISE-13.4/Linux/ISE_DS/ISE/vhdl/  
mti_se/6.6a/lin64/xilinxcorelib';  
hdlset_param (myDUT, 'SimulationLibPath', libpath);
```

You can also specify the simulation library path from the **HDL Code Generation > Test Bench** pane in the Configuration Parameters dialog box.

C/C++ Compiler Setup

HDL Coder locates and uses a supported installed compiler. For most platforms, a default compiler is supplied with MATLAB. For a list of supported compilers, see at https://www.mathworks.com/support/compilers/current_release/.

See Also

More About

- “Third-Party Synthesis Tools and Version Support” on page 1-3

Tutorials

- “HDL Code Generation and FPGA Synthesis from a MATLAB Algorithm” on page 3-2
- “Create HDL-Compatible Simulink Model” on page 3-11
- “Generate HDL Code from Simulink Model” on page 3-20
- “HDL Code Generation and FPGA Synthesis Using the HDL Workflow Advisor” on page 3-26
- “Verify Generated Code from Simulink Model Using HDL Test Bench” on page 3-33

HDL Code Generation and FPGA Synthesis from a MATLAB Algorithm

In this section...

- “About the Algorithm in This Example” on page 3-2
- “Create Local Copy of Design and Testbench Files” on page 3-2
- “Set Up Synthesis Tool Path” on page 3-3
- “Test the Original MATLAB Algorithm” on page 3-3
- “Set Up a Project Using HDL Coder App” on page 3-4
- “Open the HDL Coder Workflow Advisor” on page 3-6
- “Create Fixed-Point Versions of the Algorithm and Test Bench” on page 3-7
- “Generate HDL Code” on page 3-9
- “Verify Generated HDL Code” on page 3-9
- “FPGA Synthesis and Implementation” on page 3-9

This example illustrates how you can use HDL Coder to generate and synthesize HDL code for a MATLAB algorithm that implements a simple filter.

About the Algorithm in This Example

This tutorial uses these files:

- `mlhdlc_sfir.m` — Simple filter function from which you generate HDL code. To see the MATLAB code for the FIR filter algorithm, at the command-line, enter:

```
edit('mlhdlc_sfir')
```

- `mlhdlc_sfir_tb.m` — Test bench that the HDL Coder project uses to simulate the filter using a representative input range. To see the MATLAB code for the FIR filter test bench, at the command-line, enter:

```
edit('mlhdlc_sfir_tb')
```

Create Local Copy of Design and Testbench Files

Before you begin generating code, In the MATLAB path, navigate to a folder that is writable, and then create a working folder to store the design and test bench files.

- 1 In your current working folder, create a folder called `filter_sfir`.

```
mkdir filter_sfir;
```

- 2 Copy the tutorial files, `mlhdlc_sfir.m` and `mlhdlc_sfir_tb.m`, to this folder.

```
mlhdlc_demo_dir = fullfile(matlabroot, 'toolbox', 'hdlcoder', ...
    'hdlcoderdemos', 'matlabhdlcoderdemos');
copyfile(fullfile(mlhdlc_demo_dir, 'mlhdlc_sfir.m'), 'filter_sfir');
copyfile(fullfile(mlhdlc_demo_dir, 'mlhdlc_sfir_tb.m'), 'filter_sfir');
```

Set Up Synthesis Tool Path

If you want to synthesize the generated HDL code, before you use HDL Coder to generate code, set up your synthesis tool path. To set up the path to your synthesis tool, use the `hdlsetuptoolpath` function. For example, if your synthesis tool is Xilinx Vivado

```
hdlsetuptoolpath('ToolName', 'Xilinx Vivado', 'ToolPath', ...
    'C:\Xilinx\Vivado\2017.2\bin\vivado.bat');
```

To check your Xilinx Vivado synthesis tool setup, launch the tool with the following command:

```
!vivado
```

If you are using another Synthesis tool, to see how to set up the tool path and the right tool version to use, see “Synthesis Tool Path Setup” on page 2-2.

Test the Original MATLAB Algorithm

To verify the functionality of your MATLAB algorithm, before generating HDL code, simulate your MATLAB design.

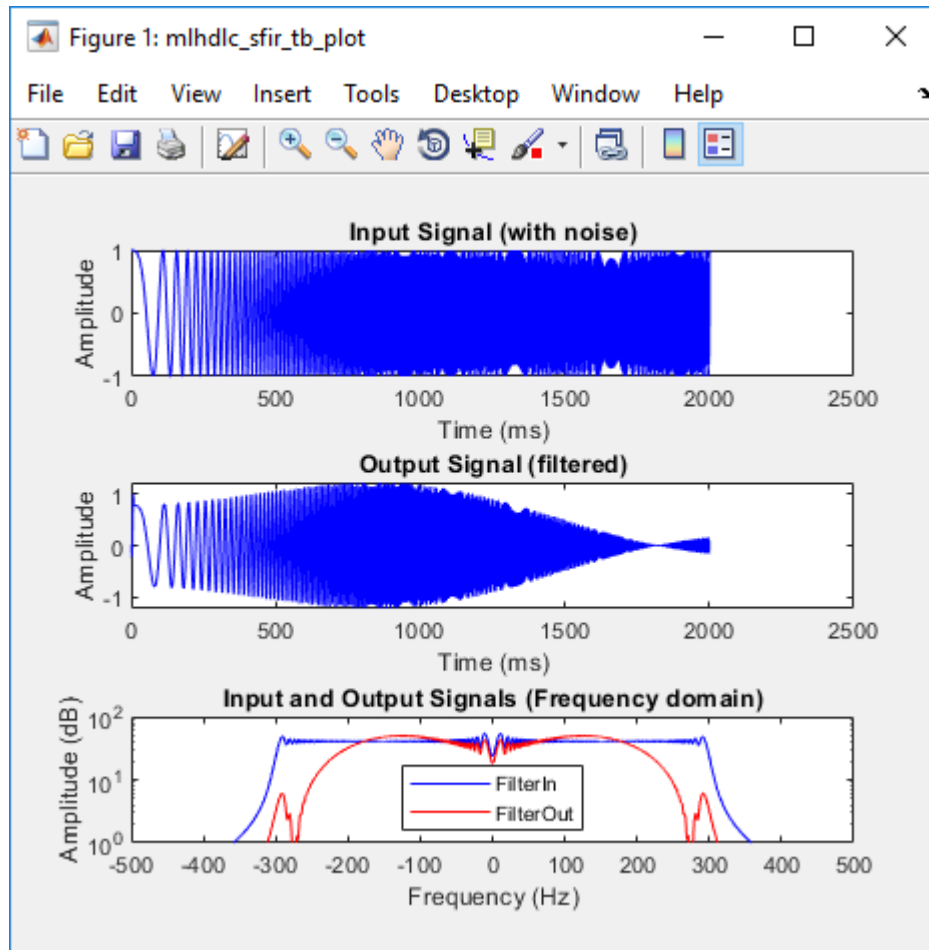
- 1 Make the `filter_sfir` folder your working folder, for example:

```
cd filter_sfir
```

- 2 Run the test bench. At the MATLAB command line, enter:

```
mlhdlc_sfir_tb
```

The test bench runs and plots the input signal and the filtered output.

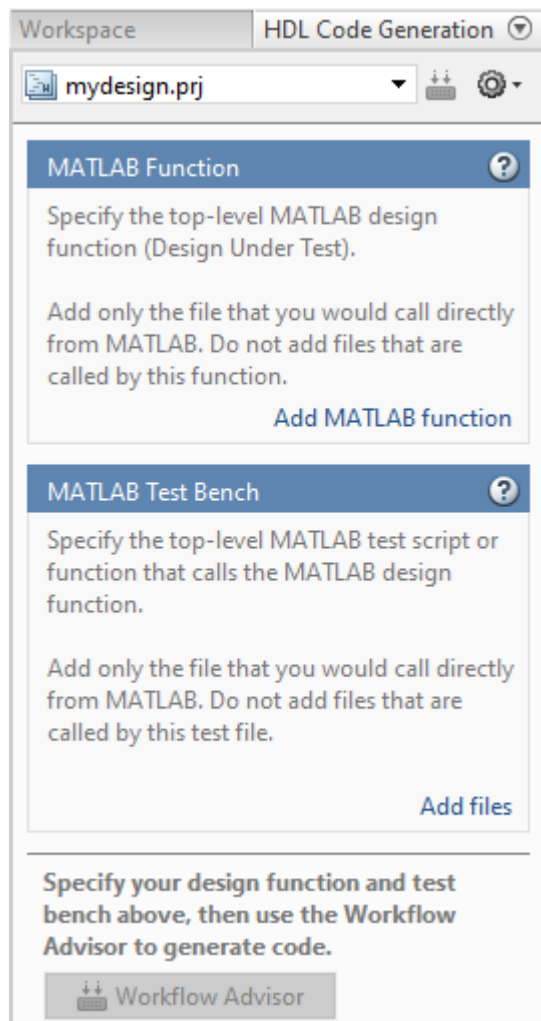


Set Up a Project Using HDL Coder App

- 1 Open the **HDL Coder** App.
 - To open the App from the UI, in MATLAB, on the **Apps** tab, in the **Code Generation** section, select **HDL Coder**. You can add this App to your favorites.
 - To open the App from the command line, enter:
`hdlcoder`

- 2 Specify the project name, for example, enter mydesign.

HDL Coder creates the project, `mydesign.prj`, in the local working folder, and opens the project in the right side of the MATLAB workspace.



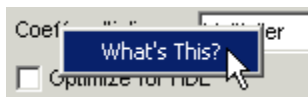
- 3 Add the design and test bench files. For **MATLAB Function**, add the `mlhdlc_sfir.m` file, and for **MATLAB Test Bench**, add the `mlhdlc_sfir_tb.m` file.

- 4 To have the App automatically define the data types of the signals, when you add the **MATLAB Function**, select **Autodefine types**. Select the **MATLAB Test Bench** file `mlhdlc_sfir_tb.m`, and then run the test bench.

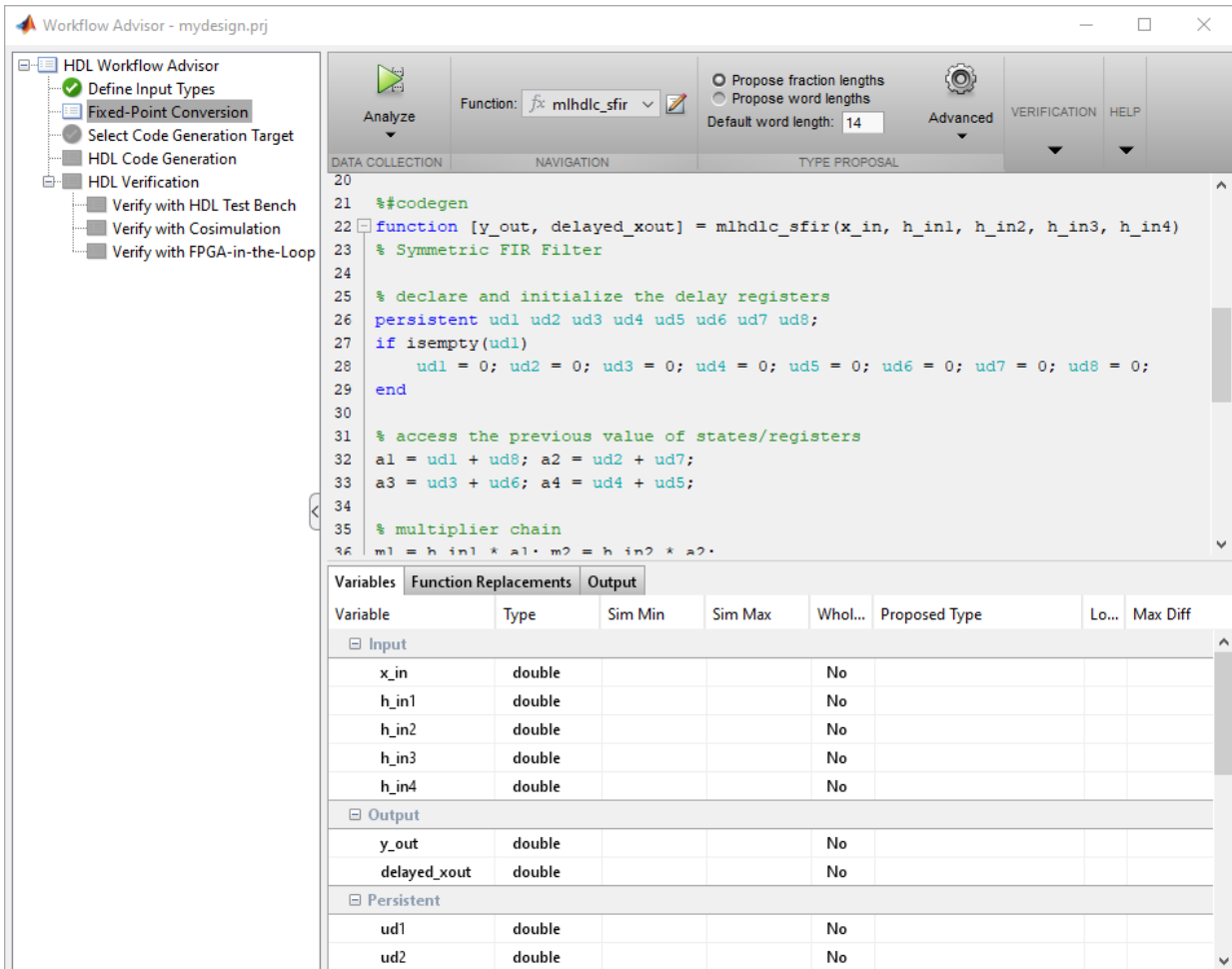
HDL Coder simulates the algorithm and test bench, and automatically defines input types. Select **Use these types**.

Open the HDL Coder Workflow Advisor

Use the HDL Coder Workflow Advisor to convert your algorithm to fixed-point, generate synthesizable HDL code, and then deploy the code to a target platform. To learn more about each individual task in the HDL Workflow Advisor, right-click that task, and select **What's This?**.



To open the Workflow Advisor, in the project, at the bottom of the pane, select the **Workflow Advisor** button. You see that the **Define Input Types** task has passed.



Workflow Advisor - mydesign.pj

HDL Workflow Advisor

- Define Input Types
- Fixed-Point Conversion
- Select Code Generation Target
- HDL Code Generation
- HDL Verification
 - Verify with HDL Test Bench
 - Verify with Cosimulation
 - Verify with FPGA-in-the-Loop

Analyze Function: `fx mlhdlc_sfir`

Propose fraction lengths
Propose word lengths
Default word length: 14

Advanced VERIFICATION HELP

DATA COLLECTION NAVIGATION TYPE PROPOSAL

```

20
21 %%codegen
22 function [y_out, delayed_xout] = mlhdlc_sfir(x_in, h_in1, h_in2, h_in3, h_in4)
23 % Symmetric FIR Filter
24
25 % declare and initialize the delay registers
26 persistent ud1 ud2 ud3 ud4 ud5 ud6 ud7 ud8;
27 if isempty(ud1)
28     ud1 = 0; ud2 = 0; ud3 = 0; ud4 = 0; ud5 = 0; ud6 = 0; ud7 = 0; ud8 = 0;
29 end
30
31 % access the previous value of states/registers
32 a1 = ud1 + ud8; a2 = ud2 + ud7;
33 a3 = ud3 + ud6; a4 = ud4 + ud5;
34
35 % multiplier chain
36 m1 = h_in1 * a1; m2 = h_in2 * a2;

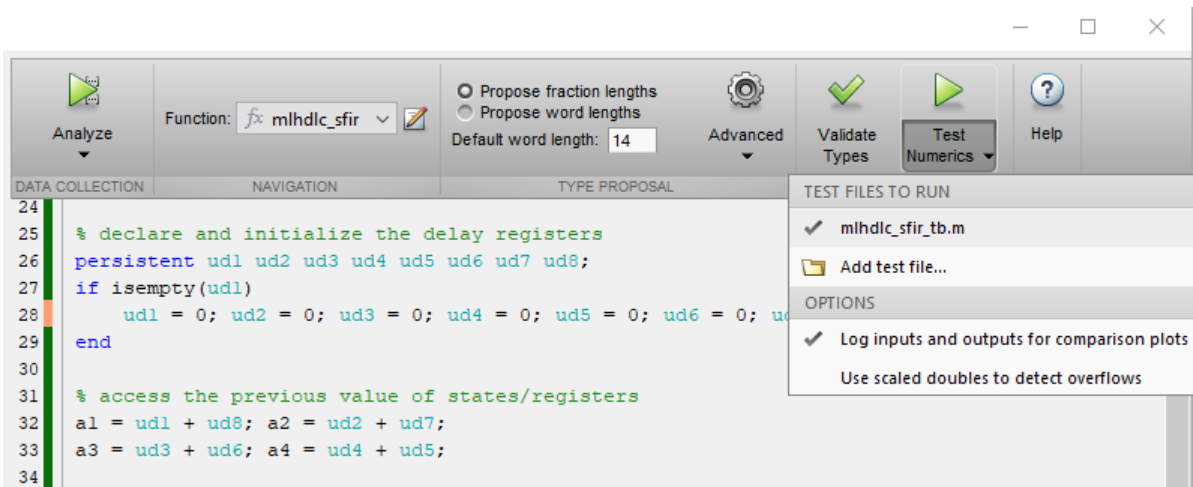
```

Variable	Type	Sim Min	Sim Max	Whol...	Proposed Type	Lo...	Max Diff
Input							
x_in	double			No			
h_in1	double			No			
h_in2	double			No			
h_in3	double			No			
h_in4	double			No			
Output							
y_out	double			No			
delayed_xout	double			No			
Persistent							
ud1	double			No			
ud2	double			No			

Create Fixed-Point Versions of the Algorithm and Test Bench

When you run fixed-point conversion, to propose fraction lengths for floating-point data types, HDL Coder uses the **Default word length**. In this tutorial, the **Default word length** is 14. The advisor provides a default **Safety Margin for Simulation Min/Max** of 0%. The advisor adjusts the range of the data by this safety factor. For example, a value of 4 specifies that you want a range of at least 4 percent larger.

Select the **Fixed-Point Conversion** task. The Fixed-Point Conversion tool opens in the right pane.



- 1 At the top left, select **Analyze**.

After the simulation, HDL Coder displays the input signal, filtered output signal, and the frequency domain plot of input and output signals. If you navigate to **Fixed-Point Conversion** tool in the Workflow Advisor Window, you see that each input, output, and persistent variable has a **Sim Min**, **Sim Max**, and **Proposed Type** in the table.

This example uses the simulation ranges to infer fixed-point types. You can use **Compute Derived Ranges** to obtain the range using static range analysis. To learn more, see “Automated Fixed-Point Conversion”.

- 2 At the top, in the Verification section, click **Validate Types**.

HDL Coder validates the build with the proposed fixed-point types and generates a fixed-point design.

- 3 At the top, in the Verification section, click the down-arrow for **Test Numerics** and select **Log inputs and outputs for comparison plots**. Click the top part of the **Test Numerics** button.

HDL Coder simulates the fixed-point design with the original test bench, compares the output to the original floating-point design output, and then displays the difference as an error signal.

- 4 In the bottom, you see a **Verification Output** tab. The tab displays a link to the report `mlhdlc_sfir_fixed_report.html`. To explore the fixed-point code for the `mlhdlc_sfir` function, open the report.

To see the fixed-point code in the MATLAB Editor, in the `filter_sfir` folder, you see a `codegen` folder. When you navigate this folder, you see a `mlhdlc_sfir_fixpt` file. Open this file.

Generate HDL Code

- 1 If you want to synthesize your design on a target FPGA platform, select the **Select Code Generation Target** task. Leave **Workflow** to **Generic ASIC/FPGA** and specify the **Synthesis tool**. If you don't see the synthesis tool, select **Refresh list**.
- 2 Before generating code, to customize code generation options, in the **HDL Code Generation** task, use the **Target, Coding Style, Clocks and Ports, Optimizations, Advanced, and Script Options** tabs
- 3 To generate HDL code, in the **HDL Code Generation** task, select **Run**.

The message window has a links to the generated HDL code and the resource report. Click the links to view the code and resource report.

Verify Generated HDL Code

- 1 In the HDL Workflow Advisor left pane, select **HDL Verification > Verify with HDL Test Bench** task.
- 2 Enable **Generate HDL test bench** and disable **Skip this step**. Enable **Simulate generated HDL test bench** and select a simulation tool. Click **Run**.

The task generates an HDL test bench, then simulates the fixed-point design using the selected simulation tool, and generates a compilation report and a simulation report.

FPGA Synthesis and Implementation

- 1 Select **Synthesis and Analysis** and disable **Skip this step**. Run the **Create Project** task.

This task creates a synthesis project for the HDL code. HDL Coder uses this project in the next task to synthesize the design.

- 2 Select and run **Run Synthesis** task. This task:
 - Launches the synthesis tool in the background.
 - Opens the synthesis project created in the previous task, compiles HDL code, synthesizes the design, and emits netlists and related files.
 - Generates a synthesis report.
- 3 Select and run **Run Implementation** task. This task:
 - Launches the synthesis tool in the background.
 - Runs a Place and Route process that takes the circuit description produced by the previous mapping process, and emits a circuit description suitable for programming an FPGA.
 - Emits pre- and post-routing timing information for use in critical path analysis and back annotation of your source model.

See Also

Related Examples

- “Basic HDL Code Generation with the Workflow Advisor”
- “Getting Started with MATLAB to HDL Workflow”
- “Generate HDL Code from MATLAB Code Using the Command Line Interface”

Create HDL-Compatible Simulink Model

In this section...

- “Open Model and HDL Coder Library” on page 3-11
- “Develop Design and Test Bench” on page 3-12
- “Configure Model for HDL Compatibility” on page 3-15
- “Check Subsystem for HDL Compatibility” on page 3-16
- “Run Model Advisor Checks for HDL Coder” on page 3-18
- “Generate HDL Code” on page 3-19

This example illustrates how you can model a simple up counter and configure the counter for compatibility with HDL code generation. The counter wraps back to zero after it reaches the upper limit that you specify.

Open Model and HDL Coder Library

- 1 Open a blank model in the Simulink Editor. To open a new model, at the command line, enter:

```
simulink
```

For this example, select the **Blank Model** template.

Note On the Simulink Start Page, in the **HDL Coder** section, you see templates that are preconfigured for HDL code generation. These models have their Configuration Parameters and solver settings set up for HDL code generation. To learn more, see “Use Simulink Templates for HDL Code Generation”.

Save the model with a filename such as `counter.slx` in a working folder that is writable.

- 2 Open the Simulink Library Browser and then open the **HDL Coder** Block Library.

To filter the Library Browser to show the block libraries that support HDL code generation, use the `hdlLib` function:

```
hdlLib
```

In the **HDL Coder** library, you see several blocks that are pre-configured for HDL code generation. Blocks in this library are available with Simulink. If you do not have HDL Coder, you can simulate the blocks in your model, but cannot generate HDL code.

You can find additional blocks in these block libraries:

- **DSP System Toolbox HDL Support**
- **Communications Toolbox HDL Support**
- **Vision HDL Toolbox**
- **LTE HDL Toolbox**

To restore the Library Browser to the default view, enter:

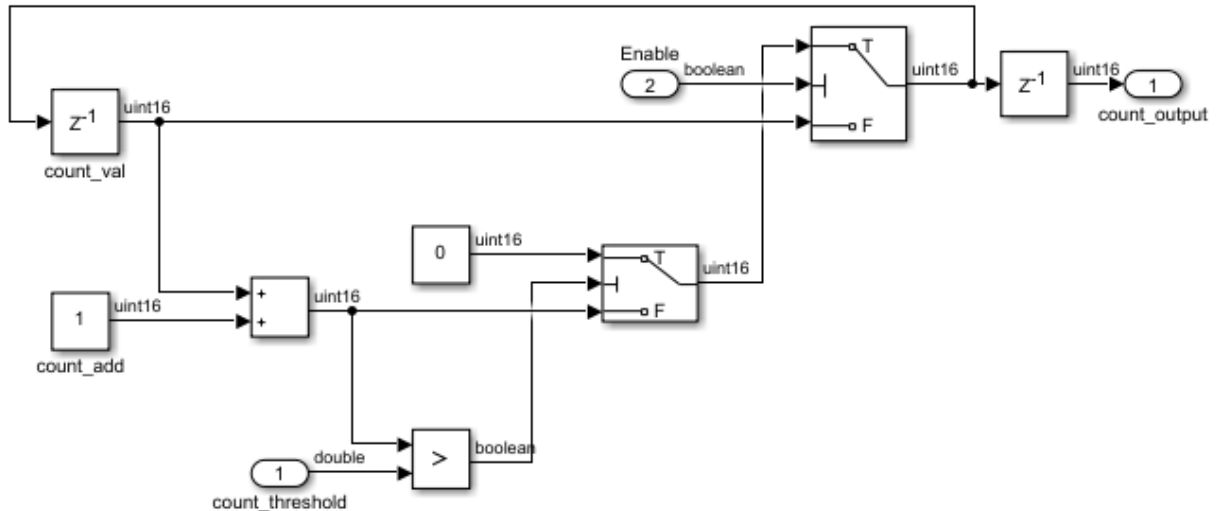
```
hdllib('off')
```

Note The set of supported blocks tend to change each release. Rebuild your supported blocks library each time you install a new version of this product.

To learn more, see `hdllib`.

Develop Design and Test Bench

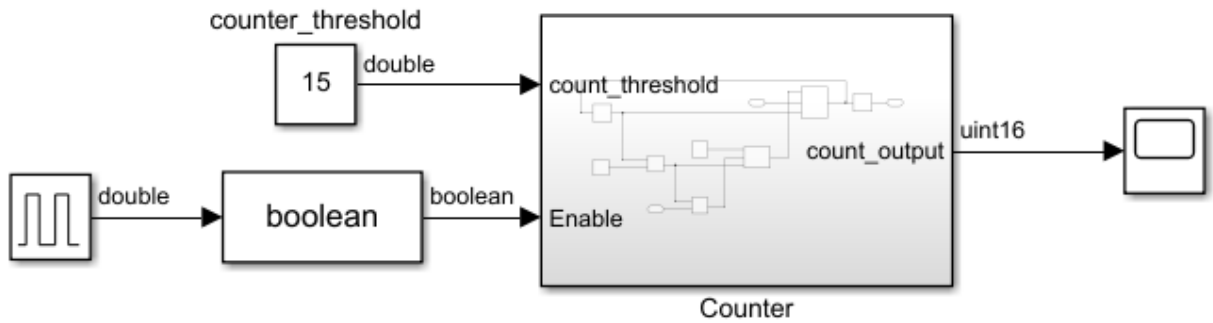
- 1 Drag the blocks from the **HDL Coder** library to the model window and connect them to develop your algorithm. This figure shows an example of how to model a counter.



The model has two input ports, `count_threshold` and `Enable`, and one output port, `count_output`. When the `Enable` signal is logic high, the counter counts up from zero to the `count_threshold` value and then wraps back to zero. When the `Enable` signal becomes logic low, the counter holds the previous value.

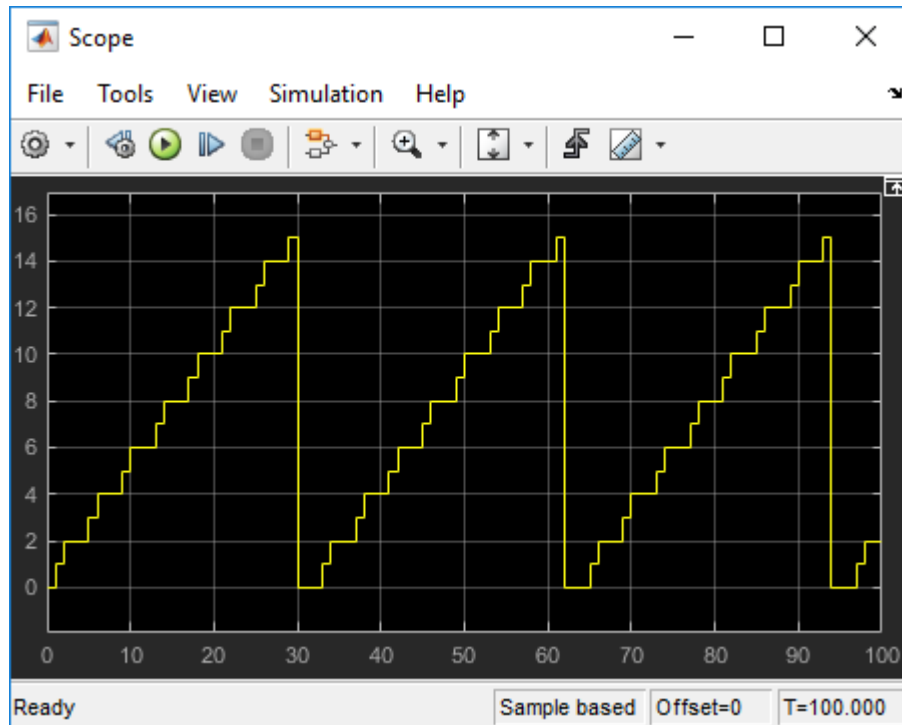
To learn more about how to create a model, see “Create a Simple Model” (Simulink).

- 2 Wrap your counter algorithm in a Subsystem. This Subsystem that you want to generate HDL code for is the Design-Under-Test (DUT). Blocks outside the DUT become part of the test bench and are used for simulation. This figure shows the DUT, Counter, and the blocks that form the test bench surrounding it.



You can use any blocks inside the test bench, which includes blocks that are not supported for HDL code generation.

- 3 Verify the functionality of the design. Simulate the testbench and then open the Scope block.



Configure Model for HDL Compatibility

To configure your model for compatibility with HDL code generation, use the `hdlsetup` function. To configure your current model, enter this command:

```
hdlsetup(gcs)
```

The `hdlsetup` function specifies ASIC/FPGA as the hardware device vendor, sets the solver options, including model start and stop times, and a fixed step solver. To see the settings changed by this function, enter this command:

```
edit hdlsetup
```

Note After you run the `hdlsetup` function, it is recommended that you simulate your model.

Check Subsystem for HDL Compatibility

The compatibility checker generates a report specified system for compatibility problems, such as use of unsupported blocks, illegal data type usage, and so on.

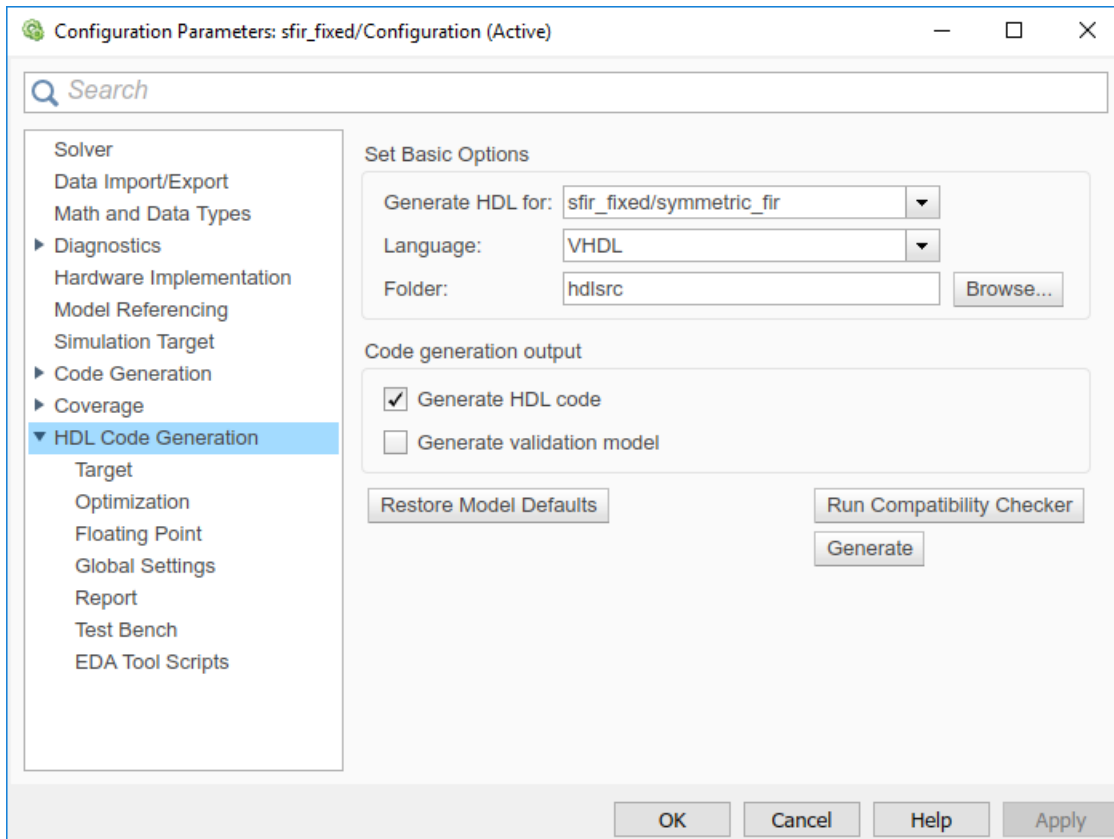
From Simulink Editor

From the Simulink Editor, right-click the DUT, and select **HDL Code > Check Subsystem for HDL compatibility**.

From Configuration Parameters Dialog Box

To customize model-level settings for your design and check compatibility of your design from the UI, use the **HDL Code Generation** pane in the Configuration Parameters dialog box or the Model Explorer.

To open the Configuration Parameters dialog box, in the Simulink Editor, on the **Simulation** tab, select **Model Configuration Parameters**.



To check HDL compatibility, in the **HDL Code Generation** pane:

- 1 For **Generate HDL for**, select the DUT Subsystem, Counter.
- 2 Click **Run Compatibility Checker**.

From Command Line

At the command line, use the `checkhdl` function. Select the DUT Subsystem and then enter this command:

```
checkhdl(gcb)
```

See also “Check Your Model for HDL Compatibility”.

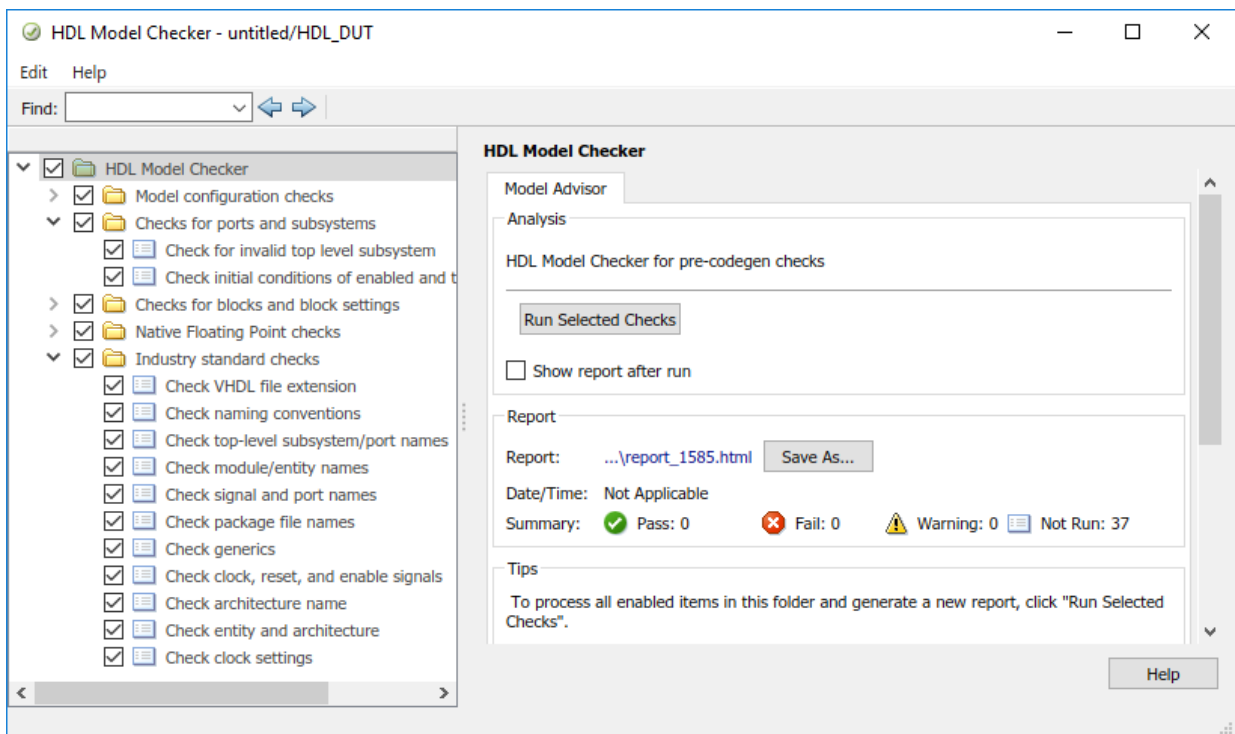
When you run this command, the HDL compatibility checker generates an HDL Code Generation Check Report. The report is stored in the target `hdlsrc` folder. If the report does not display any errors, your model is compatible for HDL code generation.

```
### Starting HDL Check.
### HDL Check Complete with 0 errors, warnings and messages.
```

Run Model Advisor Checks for HDL Coder

You can also run checks in the HDL Model Checker to verify whether your design is compatible for HDL code generation. To open the HDL Model Checker, run the `hdlmodelchecker` function:

```
hdlmodelchecker(gcb)
```



Run the checks in the HDL Model Checker and fix any warnings that are reported. To learn more, see “Getting Started with the HDL Model Checker”.

Generate HDL Code

Your model is now compatible for HDL code generation. To learn how to generate HDL code for your model, see “Generate HDL Code from Simulink Model” on page 3-20.

See Also

`checkhdl` | `hdllib` | `hdlmodelchecker` | `hdlsetup`

More About

- “Use Simulink Templates for HDL Code Generation”
- “Verify Generated Code from Simulink Model Using HDL Test Bench” on page 3-33
- “HDL Code Generation and FPGA Synthesis Using the HDL Workflow Advisor” on page 3-26

Generate HDL Code from Simulink Model

In this section...
"The <code>sfir_fixed</code> Model" on page 3-20
"Generate HDL Code" on page 3-22
"View HDL Code Generation Files" on page 3-24

For this tutorial, you can use the Counter model that you created in "Create HDL-Compatible Simulink Model" on page 3-11 as a source for HDL code generation. The model simulates an up counter that counts from zero to a threshold value and then wraps back to zero.

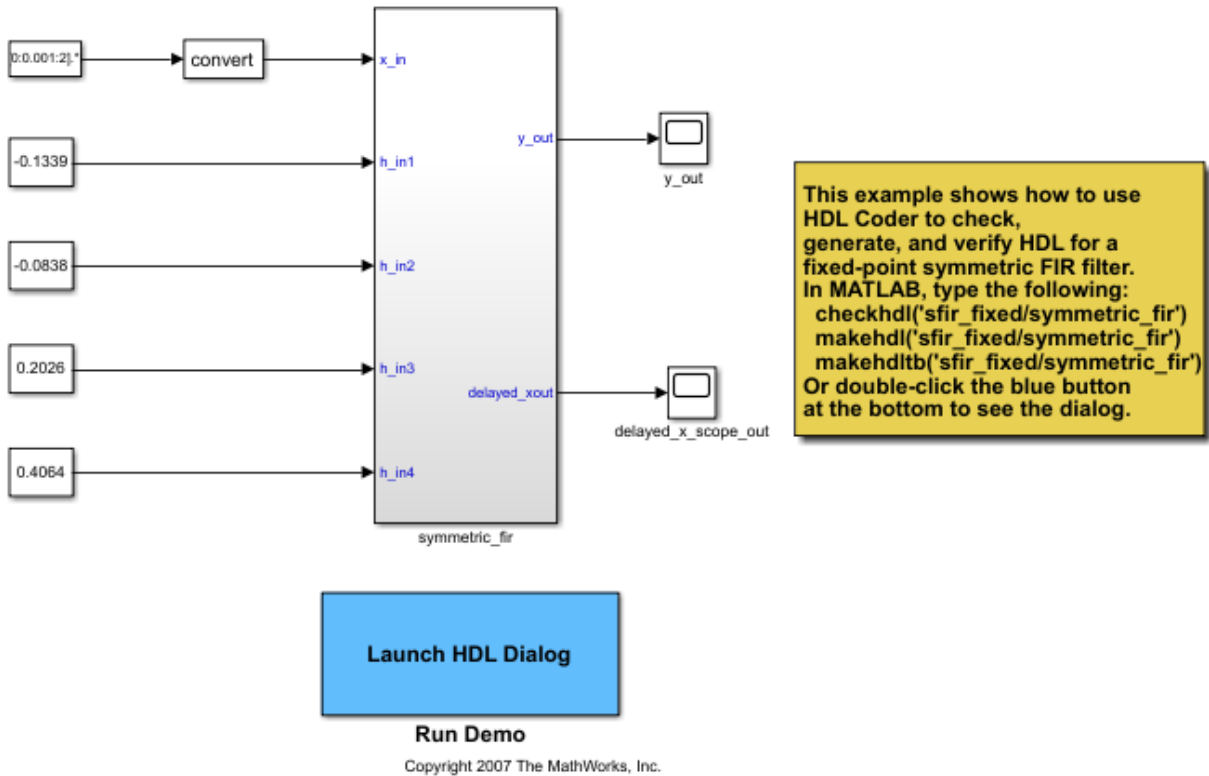
If you want to choose a different model, you can use the `sfir_fixed` model. To open this model, enter this command:

```
sfir_fixed
```

To save a local copy of the `sfir_fixed` model to your working folder, in Simulink, select **File > Save As**.

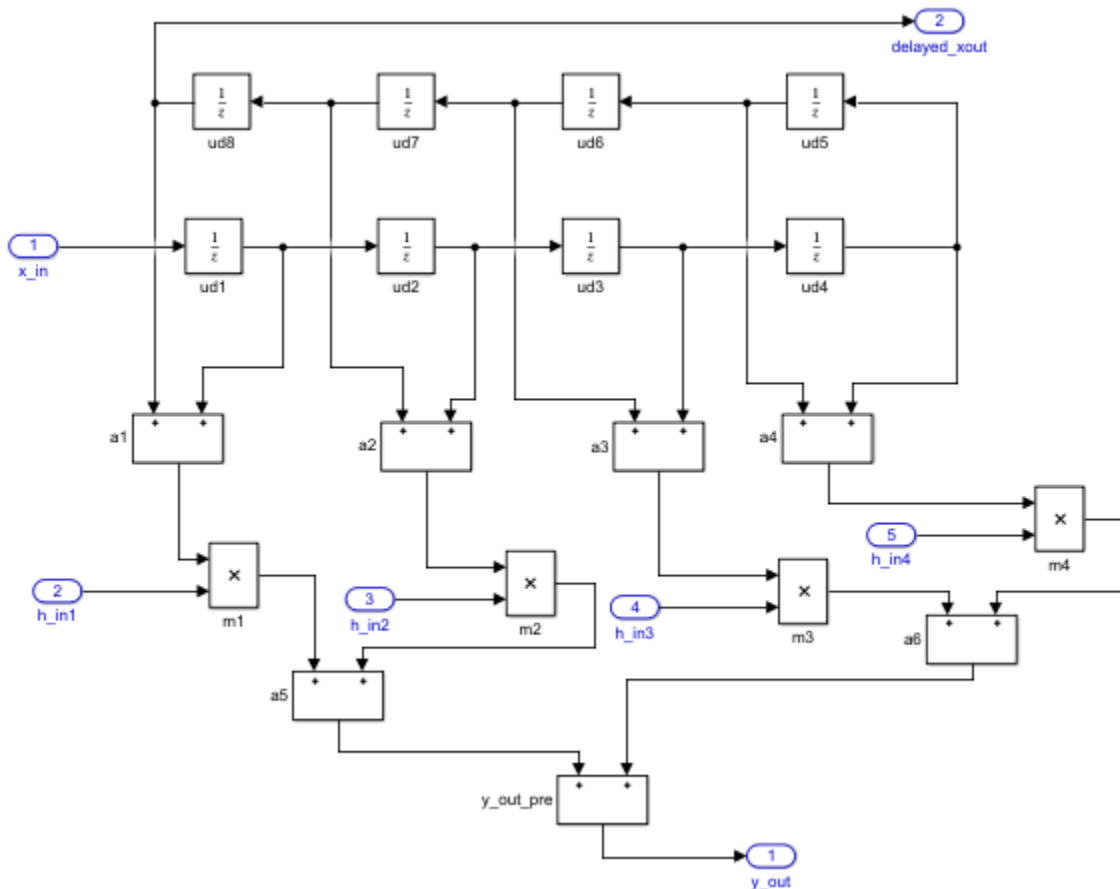
The `sfir_fixed` Model

This figure shows the `sfir_fixed` model.



The top-level model generates 16-bit fixed-point input signals for the `symmetric_fir` subsystem. The signal From Workspace block generates a test input or stimulus signal for the filter. The four Constant blocks provide filter coefficients. The Scope blocks are used in simulation, and do not generate HDL code.

The following figure shows the `symmetric_fir` Subsystem.



The fixed-point data types propagate through the subsystem. Inputs inherit the data types of the signals presented to them. Where required, internal rules of the blocks determine the output data type, given the input data types and the operation performed. The filter outputs a fixed-point result at the `y_out` port, and also replicates its input after passing it through several delay stages at the `delayed_x_out` port.

Generate HDL Code

You can generate HDL code for the DUT from the Simulink Editor, Configuration Parameters dialog box, or the command line. If you want to generate HDL code and

synthesize your design on a target FPGA platform, use the HDL Workflow Advisor. To learn more, see “HDL Code Generation and FPGA Synthesis Using the HDL Workflow Advisor” on page 3-26.

For the counter model, the counter Subsystem is the DUT. For the `sfir_fixed` model, the the `symmetric_fir` Subsystem is the DUT.

From Simulink Editor

To generate code for your DUT with the default settings, use the context menu available in the Simulink Editor. To generate code, right-click the DUT, and select **HDL Code > Generate HDL for Subsystem**.

By default, HDL Coder generates VHDL code in the target `hdlsrc` folder.

From Configuration Parameters Dialog Box

To customize model-level settings for your design and generate HDL code from the UI, use the **HDL Code Generation** pane in the Configuration Parameters dialog box or the Model Explorer. Before generating code, you can select the target HDL language and directory, specify enable native floating-point support, generate resource and traceability reports, use model-level optimizations, and modify other global settings.

To open the Configuration Parameters dialog box, in the Simulink Editor, on the **Simulation** tab, select **Model Configuration Parameters**. To generate code, in the **HDL Code Generation** pane:

- 1 For **Generate HDL for**, specify the DUT.
- 2 For **Language** and **Folder**, specify the target language of the generated HDL code and the target folder that stores generated code files and scripts.
- 3 Click **Generate**.

From Command Line

To generate HDL code from the command line, use the `makehdl` function. In your Simulink model, select the DUT that you want to generate code for, and enter this command:

```
makehdl(gcb)
```

To customize model-level settings when you generate code for your DUT, specify one or more name-value pair arguments. By default, HDL Coder generates VHDL code. To generate Verilog code, enter this command:

```
makehdl(gcb, 'TargetLanguage', 'Verilog')
```

The name-value pairs or properties that you can specify can be mapped to the corresponding parameter in the Configuration Parameters dialog box. In the dialog box, when you right-click a parameter and select **What's this?**, you see a **command-line** field that indicates the property you can specify with `makehdl`. This table shows the relationship between some of the parameters in the **HDL Code Generation** pane and the corresponding `makehdl` property.

Configuration Parameter	makehdl Property
Generate HDL for	HDLSubsystem
Language	TargetLanguage
Folder	TargetDirectory

To learn more about the name-value pairs that you can specify, see `makehdl`. The name-value pairs that you specify are not saved on the model and therefore do not persist across function calls. To save the global settings that you customize on the model, use `hdlset_param`. For example, to save Verilog as the target language on the model and then generate code, enter these commands:

```
hdlset_param(gcs, 'TargetLanguage', 'Verilog')
makehdl(gcb)
```

View HDL Code Generation Files

- 1 HDL Coder compiles the model before generating code. Depending on model display options such as port data types, the model can change in appearance after code generation.

As code generation proceeds, HDL Coder displays progress messages. The process should complete with the message

```
### HDL Code Generation Complete.
```

When generating code, HDL Coder generates a message with:

- Link to the Config Set that indicates the model for which the Configuration Parameters are applied.
 - Links to the generated files. To view the files in the MATLAB Editor, click the links.
- 2 A folder icon for the `hdlsrc` folder is now visible in the Current Folder browser. To view generated code and script files, double-click the `hdlsrc` folder icon. In the folder, you see a file containing the VHDL or Verilog code, a script to compile the generated code, a synthesis script, and a mapping file. For example, if you generated code for the `symmetric_fir` Subsystem, you see these files in the `hdlsrc` folder:
- `symmetric_fir.vhd`: VHDL code. This file contains an entity definition and RTL architecture implementing the `symmetric_fir` filter.

Note If you generated Verilog code, you get a `symmetric_fir.v` file.

- `symmetric_fir_compile.do`: Mentor Graphics ModelSim compilation script (`vcom` command) to compile the generated VHDL code.
 - `symmetric_fir_synplify.tcl`: Synplify® synthesis script.
 - `symmetric_fir_map.txt`: Mapping file. This report file maps generated entities or modules to the subsystems that generated them. See “Trace Code Using the Mapping File”.
- 3 To view the generated VHDL code in the MATLAB Editor, double-click the `symmetric_fir.vhd` file icon in the Current Folder browser.

Before you proceed to verify the generated code or deploy your design on the target hardware, from the `hdlsrc` folder, navigate to the current working folder. To learn how you can verify the generated HDL code, see “Verify Generated Code from Simulink Model Using HDL Test Bench” on page 3-33.

See Also

`hdlset_param` | `hdlsetup` | `makehdl`

More About

- “Create HDL-Compatible Simulink Model” on page 3-11
- “HDL Code Generation and FPGA Synthesis Using the HDL Workflow Advisor” on page 3-26

HDL Code Generation and FPGA Synthesis Using the HDL Workflow Advisor

In this section...
“About HDL Workflow Advisor” on page 3-26
“Set Up Tool Path” on page 3-27
“Open the HDL Workflow Advisor” on page 3-27
“Generate HDL Code” on page 3-28
“Perform FPGA Synthesis and Analysis” on page 3-30
“Run Workflow at Command Line with a Script” on page 3-31

This example shows how you can use the HDL Workflow Advisor to generate HDL code and synthesize your design on a target XilinxFPGA.

For this tutorial, you can use the Counter model that you created in “Create HDL-Compatible Simulink Model” on page 3-11 as a source for HDL code generation. The model simulates an up counter that counts from zero to a threshold value and then wraps back to zero. If you want to choose a different model, you can use the `sfir_fixed` model. To open this model, enter this command:

```
sfir_fixed
```

To learn more about the `sfir_fixed` model, see “Generate HDL Code from Simulink Model” on page 3-20.

About HDL Workflow Advisor

The HDL Workflow Advisor guides you through the stages of generating HDL code for a Simulink subsystem and the FPGA design process, such as:

- Checking the model for HDL code generation compatibility and automatically fixing incompatible settings.
- Generation of HDL code, a test bench, and scripts to build and run the code and test bench.
- Generation of cosimulation or SystemVerilog DPI test benches and code coverage (requires HDL Verifier).

- Synthesis and timing analysis through integration with third-party synthesis tools.
- Back-annotation of the model with critical path information and other information obtained during synthesis.
- Complete automated workflows for selected FPGA development target devices and the Simulink Real-Time FPGA I/O workflow, including FPGA-in-the-loop simulation.

Set Up Tool Path

If you do not want to synthesize your design, but want to generate HDL code, you do not have to set the tool path. In the HDL Workflow Advisor, on the **Set Target > Set Target Device and Synthesis Tool** step, leave the **Synthesis tool** setting to the default **No Synthesis Tool Specified**, and then run the workflow.

If you want to synthesize your design on a target platform, before you open the HDL Workflow Advisor and run the workflow, set up the path to your Synthesis tool. This example uses Xilinx Vivado, so you must have already installed Xilinx Vivado. To set the tool path, use the `hdlsetuptoolpath` function to point to an installed Xilinx Vivado 2017.2 executable.

```
hdlsetuptoolpath('ToolName','Xilinx Vivado','ToolPath',...  
    'C:\Xilinx\Vivado\2017.2\bin\vivado.bat');
```

To follow this example, you can use a different synthesis tool of your choice and use `hdlsetuptoolpath` to set the tool path.

Open the HDL Workflow Advisor

To start the HDL Workflow Advisor from a Simulink model:

- From the Simulink Editor, right-click the DUT subsystem and select **HDL Code > HDL Workflow Advisor**.
- From the command line, select the DUT subsystem, and use the `hdladvisor` function:

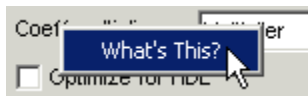
```
hdladvisor(gcf)
```

When you open the HDL Workflow Advisor, the code generator can warn that the project folder is incompatible. To open the Advisor, select **Remove slprj and continue**.

In the HDL Workflow Advisor, the left pane lists the folders in the hierarchy. Each folder represents a group or category of related tasks. Expanding the folders shows available

tasks in each folder. From the left pane, you can select a folder or an individual task. The HDL Workflow Advisor displays information about the selected folder or task in the right pane. The contents of the right pane depends on the selected folder or task. For some tasks, the right pane contains simple controls for running the task and a display area for status messages and other task results. For other tasks that involve setting code or test bench generation parameters, the right pane displays several parameter and option settings.

To learn more about each individual task, right-click that task, and select **What's This?**.

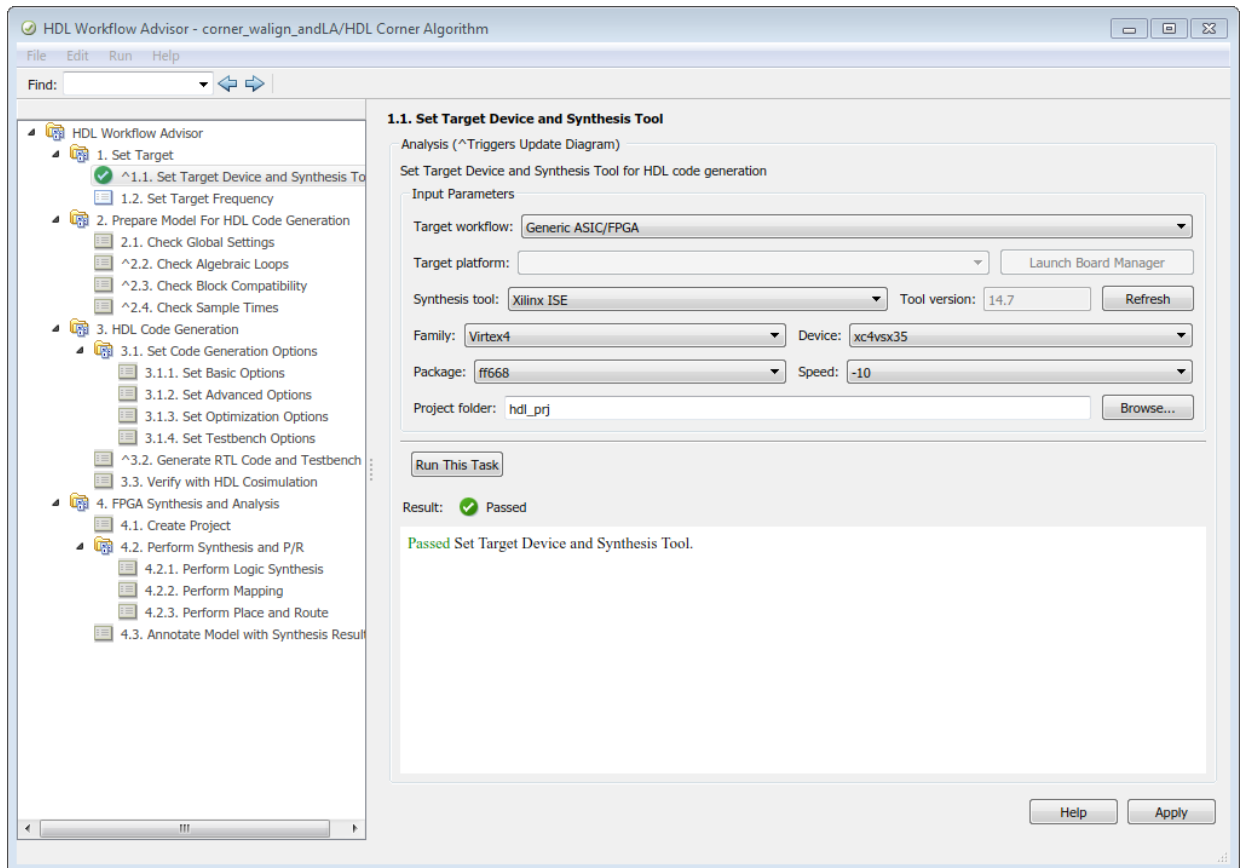


To learn more about the HDL Workflow Advisor window, see “Getting Started with the HDL Workflow Advisor”.

You can use the HDL Workflow Advisor to generate HDL code from a MATLAB script. To learn more, see “Basic HDL Code Generation with the Workflow Advisor”.

Generate HDL Code

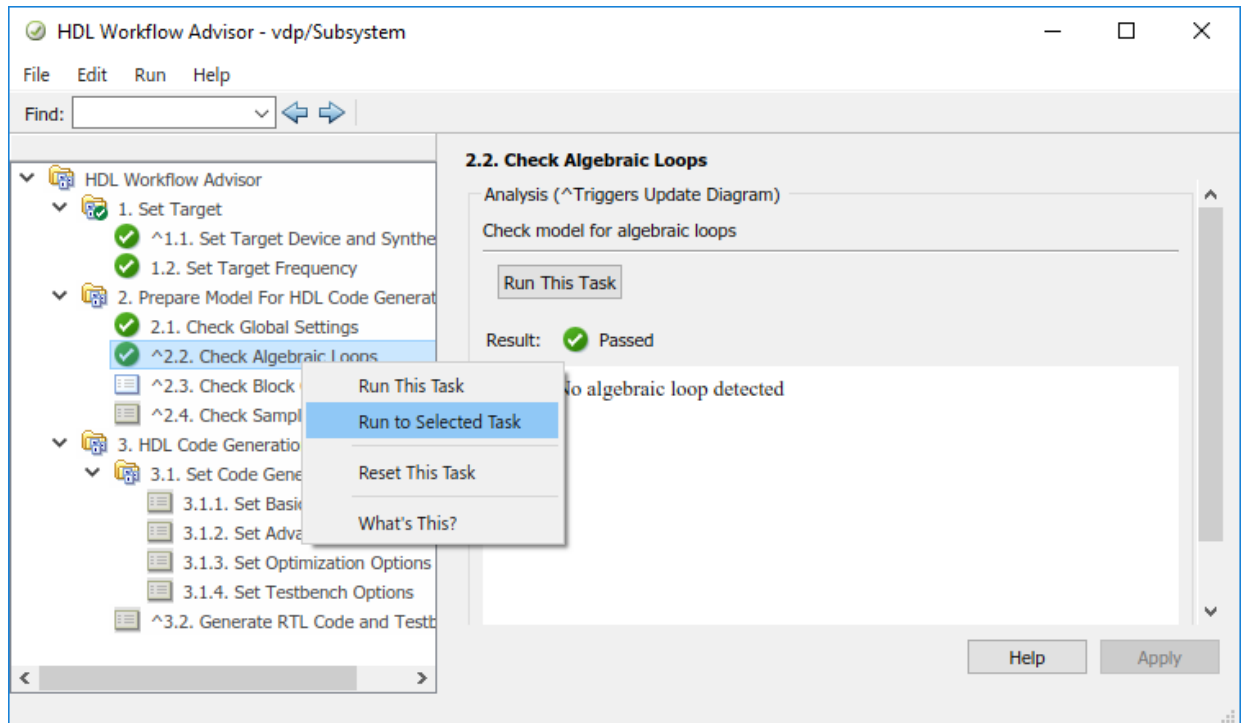
- 1 In the **Set Target > Set Target Device and Synthesis Tool** step, for **Synthesis tool**, select Xilinx Vivado and select **Run This Task**.



- 2 Leave all settings to default and right-click the **Check Sample Times** task and select **Run to Selected Task**. By running the tasks in the **Prepare Model For HDL Code Generation** folder, the HDL Workflow Advisor checks the model for code generation compatibility.

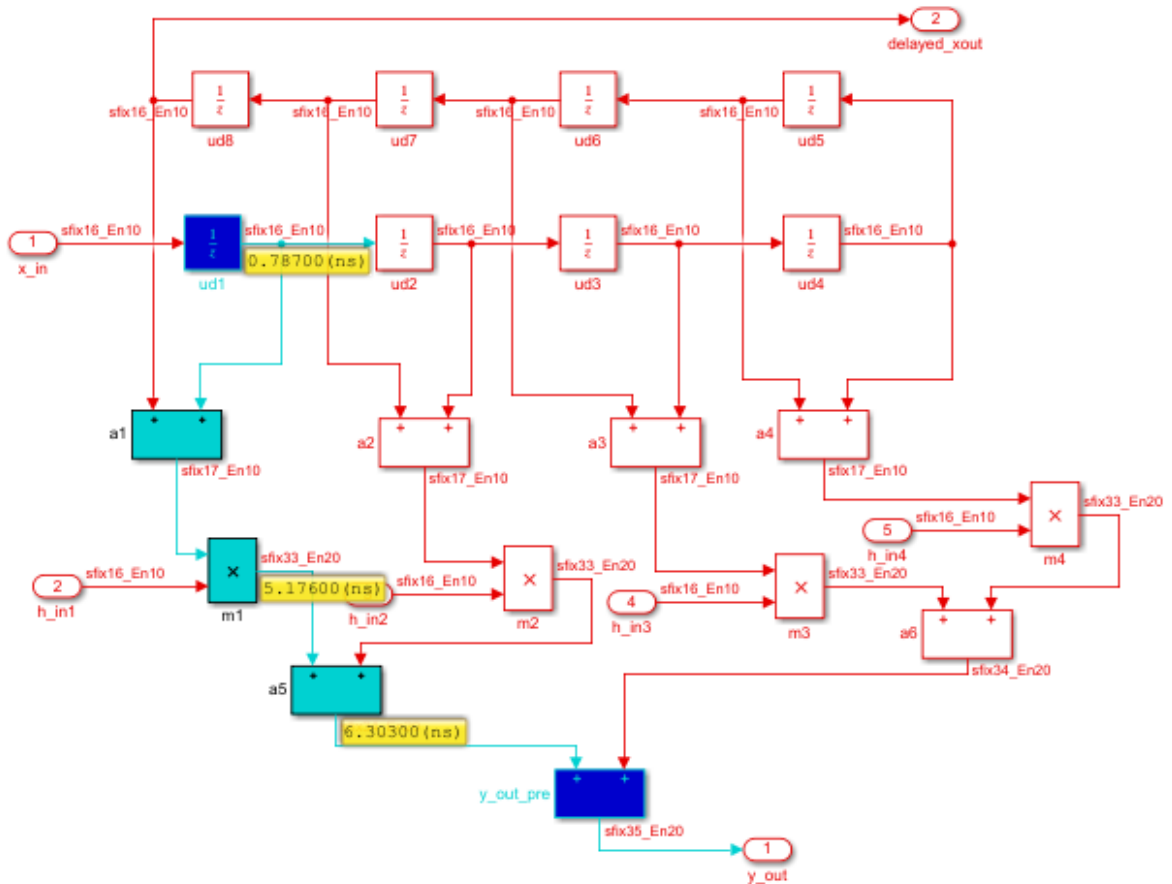
Note If running a task generates a warning, select **Modify All**, and rerun the task.

- 3 To modify code generation options, use the tasks in **Set Code Generation Options**. For example, to customize the target HDL language and the target code generation folder, use the **Set Basic Options** task. After you make changes, click **Apply**.
- 4 To generate code, right-click the **Generate RTL Code and Testbench** task, and select **Run to Selected Task**.



Perform FPGA Synthesis and Analysis

- 1 In the **FPGA Synthesis and Analysis > Perform Synthesis and P/R > Perform Place and Route** task, unselect **Skip this task** and click **Apply**.
- 2 Right-click **Annotate Model with Synthesis Result** and select **Run to Selected Task**.
- 3 View the annotated critical path in the model.



Run Workflow at Command Line with a Script

To run the HDL workflow at a command line, you can export the Workflow Advisor settings to a script. To export to script, in the HDL Workflow Advisor window, select **File > Export to Script**. In the Export Workflow Configuration dialog box, enter a file name and save the script.

The script is a MATLAB file that you can run from the command line. You can modify the script directly or, import the script into the HDL Workflow Advisor, modify the tasks, and export the updated script. To learn more, see “Run HDL Workflow with a Script”.

See Also

`hdladvisor` | `hdlsetuptoolpath` | `makehdl`

More About

- “Tool Setup” on page 2-2
- “Create HDL-Compatible Simulink Model” on page 3-11
- “Generate HDL Code from Simulink Model” on page 3-20

Verify Generated Code from Simulink Model Using HDL Test Bench

In this section...

“How to Verify the Generated Code” on page 3-33

“What is a HDL Test Bench?” on page 3-33

“Generate HDL Test Bench” on page 3-34

“View HDL Test Bench Files” on page 3-35

“Run Simulation and Verify Generated HDL Code” on page 3-36

This example shows how to generate a HDL test bench and verify the generated code for your design. The example assumes that you have already generated HDL code for your model. To learn how to generate HDL code, see “Generate HDL Code from Simulink Model” on page 3-20.

If you used the counter model to generate code, the entity to be tested using the HDL test bench is the `counter.v` or `counter.vhd` file. If you used the `sfir_fixed` model, the entity to be tested is the `symmetric_fir.vhd` or the `symmetric_fir.v` file.

How to Verify the Generated Code

This example illustrates how to generate a HDL test bench to simulate and verify the generated HDL code for your design. You can also verify the generated HDL code from your model using these methods:

Verification Method	For More Information
Validation Model	“Generated Model and Validation Model”
HDL Cosimulation (requires HDL Verifier)	“Cosimulation”
FPGA-in-the-Loop	“FPGA-in-the-Loop”
SystemVerilog DPI Test Bench	“SystemVerilog DPI Test Bench”

What is a HDL Test Bench?

To verify the functionality of the HDL code that you generated for the DUT, generate a HDL test bench. A test bench includes:

- Stimulus data generated by signal sources connected to the entity under test.
- Output data generated by the entity under test. During a test bench run, this data is compared to the outputs of the VHDL model, for verification purposes.
- Clock, reset, and clock enable inputs to drive the entity under test.
- A component instantiation of the entity under test.
- Code to drive the entity under test and compare its outputs to the expected data.

You can simulate the generated test bench and script files with the Mentor Graphics ModelSim simulator.

Generate HDL Test Bench

Depending on whether you generated VHDL or Verilog code, make sure that you can generate VHDL or Verilog test bench code. The test bench code drives the HDL code that you generated for the DUT. By default, the HDL code and the test bench code are written to the same target folder `hdlsrc` relative to the current folder.

From Configuration Parameters Dialog Box

To customize test bench settings and select the test bench that you want to use to verify the generated code, use the **HDL Code Generation > Test Bench** pane in the Configuration Parameters dialog box or the Model Explorer.

To open the Configuration Parameters dialog box, in the Simulink Editor, on the **Simulation** tab, select **Model Configuration Parameters**. To generate test bench code, in the **HDL Code Generation > Test Bench** pane, select **HDL test bench** and click **Generate Test Bench**.

From Command Line

To generate HDL test bench from the command line, use the `makehdltb` function. In your Simulink model, select the DUT that you want to generate the test bench for, and enter this command:

```
makehdltb(gcb)
```

To customize model-level settings when you generate the test bench, specify one or more name-value pair arguments. By default, HDL Coder generates VHDL code. To generate Verilog code, enter this command:

```
makehdltb('sfir_fixed/symmetric_fir','TargetLanguage','verilog')
```

With `makehdltb`, you can specify the same name-value pairs that you use with `makehdl`. To learn more about the name-value pairs that you can specify, see `makehdltb`.

These name-value pairs or properties that you can specify can be mapped to the corresponding parameter in the Configuration Parameters dialog box. In the dialog box, when you right-click a parameter and select **What's this?**, you see a **command-line** field that indicates the property you can specify with `makehdltb`. This table shows the relationship between some of the parameters in the **HDL Code Generation** pane and the corresponding `makehdltb` property.

Configuration Parameter	<code>makehdltb</code> Property
Generate HDL for	<code>HDLSubsystem</code>
Language	<code>TargetLanguage</code>
Folder	<code>TargetDirectory</code>

View HDL Test Bench Files

- 1 If you haven't already generated code for your model, HDL Coder compiles the model and generates HDL code before generating the test bench. Depending on model display options such as port data types, the model can change in appearance after code generation.

As test bench generation proceeds, HDL Coder displays progress messages. The process should complete with the message

```
### HDL TestBench Generation Complete.
```

- 2 After generating the test bench, you see the generated files in the `hdlsrc` folder. For example, if you generated a test bench for the `sfir_fixed/symmetric_fir` Subsystem, the folder contains:
 - `symmetric_fir_tb.vhd`: VHDL test bench code, with generated test and output data.

Note If you generated Verilog test bench code, the generated file is `symmetric_fir_tb.v`.

- `symmetric_fir_tb_compile.do`: Mentor Graphics ModelSim compilation script (`vcom` commands). This script compiles and loads the entity to be tested (`symmetric_fir.vhd`) and the test bench code (`symmetric_fir_tb.vhd`).

- `symmetric_fir_tb_sim.do`: Mentor Graphics ModelSim script to initialize the simulator, set up **wave** window signal displays, and run a simulation.
- 3 To view the generated test bench code in the MATLAB Editor, double-click the `symmetric_fir_tb.vhd` or `symmetric_fir_tb.v` file in the Current Folder.

Run Simulation and Verify Generated HDL Code

To verify the simulation results, you can use the Mentor Graphics ModelSim simulator. Make sure that you have already installed Mentor Graphics ModelSim.

To launch the simulator, use the `vsim` function. This command shows how to open the simulator by specifying the path to the executable:

```
vsim('vsimdir','C:\Program Files\ModelSim\questasim\10.5c\win64\vsim.exe')
```

To compile and run a simulation of the generated model and test bench code, use the scripts that are generated by HDL Coder. Following example illustrates the commands that compile and simulate the generated test bench for the `sfir_fixed/symmetric_fir` Subsystem. You can similarly compile and simulate the test bench generated for the `counter/counter` Subsystem.

- 1 Open the Mentor Graphics ModelSim software and navigate to the folder that has the generated code files and the scripts.
- 2 Use the generated compilation script to compile and load the generated model and text bench code. For example, if you generated a test bench for the `sfir_fixed/symmetric_fir` Subsystem, run this command to compile the generated code.

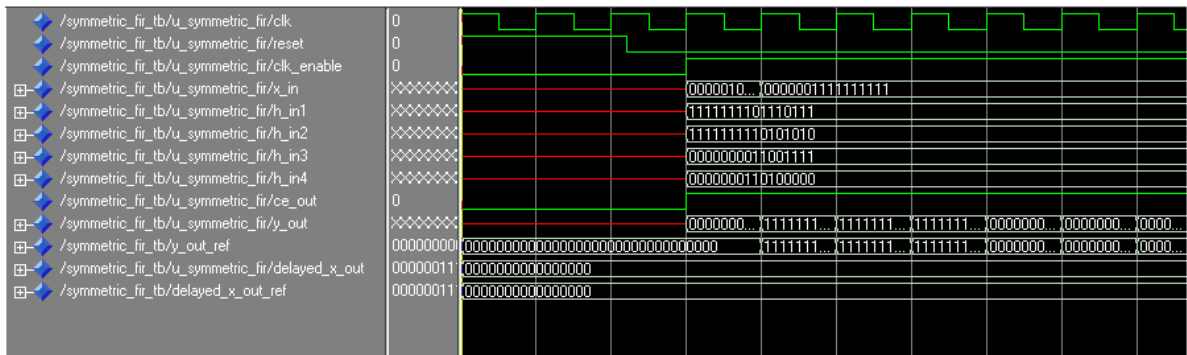
```
ModelSim> do symmetric_fir_tb_compile.do
# Model Technology ModelSim SE vlog 6.0 Compiler 2004.08 Aug 19 2004
# -- Compiling module symmetric_fir
#
# Top level modules:
#   symmetric_fir
# Model Technology ModelSim SE vlog 6.0 Compiler 2004.08 Aug 19 2004
# -- Compiling module symmetric_fir_tb
#
# Top level modules:
#   symmetric_fir_tb
```

- 3 Use the generated simulation script to execute the simulation. The following listing shows the command and responses. You can ignore any warning messages. The test bench termination message indicates that the simulation has run to completion without comparison errors. For example, if you generated a test bench for the

`sfir_fixed/symmetric_fir` Subsystem, run this command to simulate the generated code.

```
ModelSim>do symmetric_fir_tb_sim.do
# vsim work.symmetric_fir_tb
# Loading work.symmetric_fir_tb
# Loading work.symmetric_fir
# **** Test Complete. ****
# Break at
C:/work/sl_hdlcoder_work/vlog_code/symmetric_fir_tb.v line 142
# Simulation Breakpoint:Break at
C:/work/sl_hdlcoder_work/vlog_code/symmetric_fir_tb.v line 142
# MACRO ./symmetric_fir_tb_sim.do PAUSED at line 14
```

- To see the simulation results, in the Mentor Graphics ModelSim simulator, open the **wave** window. The simulation script displays inputs and outputs in the model including the reference signals `y_out_ref` and `delayed_x_out_ref` in the **wave** window.



- You can now view the signals and verify that the simulation results match the functionality of your original design. After verifying, close the Mentor Graphics ModelSim simulator, and then close the files that you have opened in the MATLAB Editor.

See Also

`makehdl` | `makehdltb`

More About

- “Test Bench Generation Output”
- “HDL Test Bench”
- “HDL Code Generation and FPGA Synthesis Using the HDL Workflow Advisor” on page 3-26