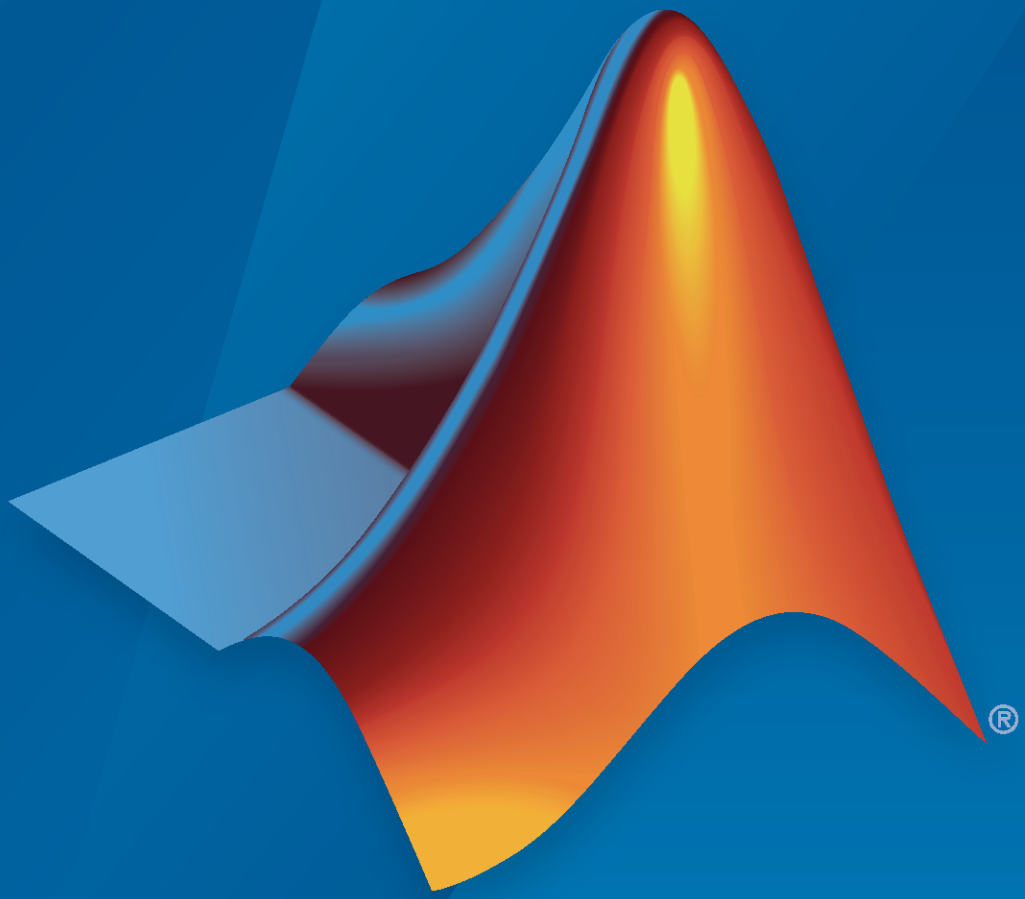
HDL Coder™

Getting Started Guide



MATLAB® & SIMULINK®

R2020a

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.
1 Apple Hill Drive
Natick, MA 01760-2098

HDL Coder™ Getting Started Guide

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)
March 2019	Online only	Revised for Version 3.14 (Release 2019a)
September 2019	Online only	Revised for Version 3.15 (Release 2019b)
March 2020	Online only	Revised for Version 3.16 (Release 2020a)

Contents

1 About HDL Coder

2 Getting Started with HDL Coder

3 Tutorials

About HDL Coder

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.

Support for industry standards is available through IEC Certification Kit (for ISO 26262 and IEC 61508).

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

HDL Language Support and Supported Third-Party Tools and Hardware

In this section...

VHDL and Verilog Language Support

The generated HDL code complies with the following standards:

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

Third-Party Synthesis Tools and Version Support

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

- Intel Quartus Prime 18.1
- Intel Quartus Pro 19.2
- Xilinx Vivado® Design Suite 2019.1
- 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	**Warning** `Speedgoat IO331` and its variant `Speedgoat IO331-6` will no longer be supported in R2020b with the `Simulink Real-Time FPGA I/O` workflow. You can still run the workflow for these IO modules in R2020a. In R2020b, use `Speedgoat IO333-325K` or a later Speedgoat IO module that is based on Xilinx Vivado.
IO333, IO334, IO335	Xilinx Kintex-7	
IO332, IO397	Xilinx Artix-7	
IO323, IO331	Xilinx Spartan-6	For more information and to learn about the synthesis tool requirements, see Xilinx HDL Support with Speedgoat IO Modules.

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
	Spartan3
	Spartan6
Altera® Quartus® II	Cyclone® III
	Cyclone IV
	Arria® II GX and GZ
	Stratix® IV
	Stratix V

Synthesis Tool	Device Family
	Cyclone III
	Arria 10
	Arria V GX
	MAX 10
Intel Quartus Pro	Arria 10
	Cyclone 10 GX
	Stratix 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/ and FMCOMMS5
	ZC702 with FMC-HDMI-CAM
	Zynq ZC706 evaluation kit
	Zynq ZC702 evaluation kit
	PicoZed FMC-HDMI-CAM
	Zynq UltraScale+ MPSoC ZCU102 Evaluation Kit
Altera Quartus II	Arria 10 SoC development kit
	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
	Artix7
	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

Getting Started with HDL Coder

Tool Setup

In this section...

Synthesis Tool Path Setup

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 Prime synthesis tool setup in MATLAB, try launching the tool with the following command:

```
!quartus
```

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

```
!qpro
```

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 `compxlib`.

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 `compxlib` 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

Create Simulink Model for HDL Code Generation

In this section...

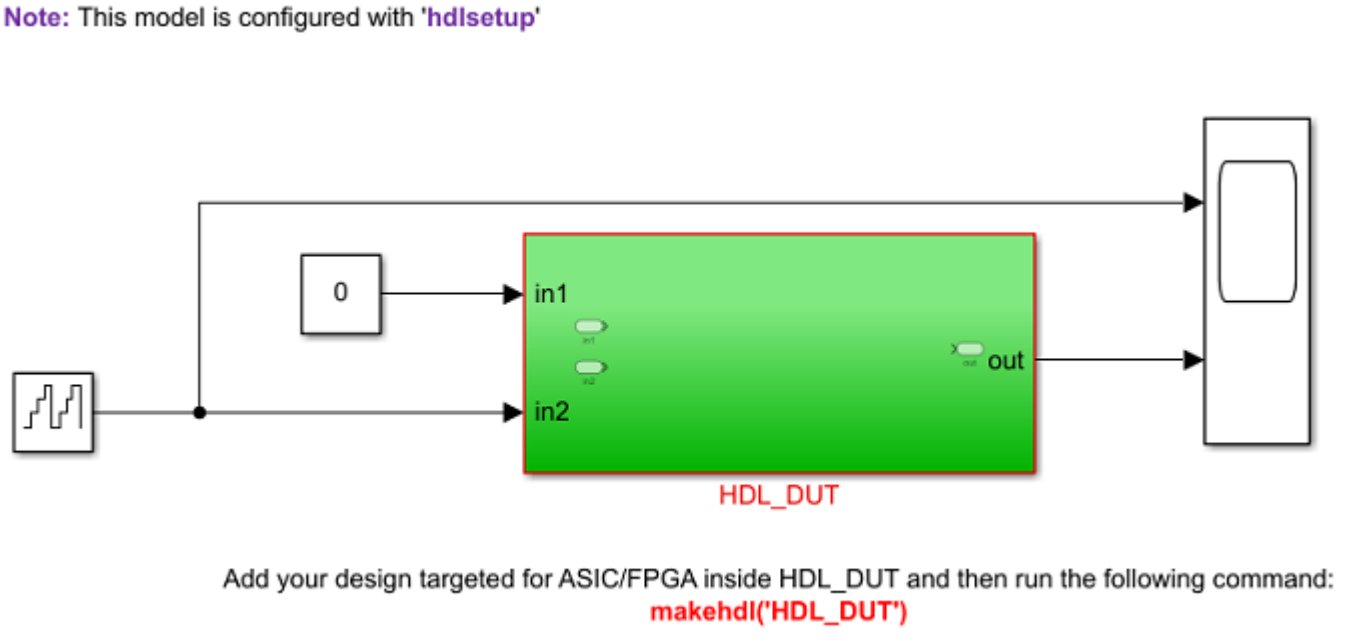
HDL Coder can generate VHDL and Verilog code from MATLAB code, Simulink models, and Stateflow charts. You can then verify that the generated code matches your original algorithm, and deploy it on the target hardware.

This example illustrates how you can create a Simulink model for HDL code generation. The model is a simple up counter algorithm that wraps back to zero after it reaches the upper limit that you specify.

Open Model and HDL Coder Library

1. Start MATLAB. From the MATLAB toolstrip, click the **Simulink** button  . Then, in the **HDL Coder** section, select the **Blank DUT** template.

   Selecting this template opens a Simulink model that is preconfigured for HDL code generation. Save the model with a file name such as `hdlcoder_simple_up_counter.slx` in a working folder that is writable.

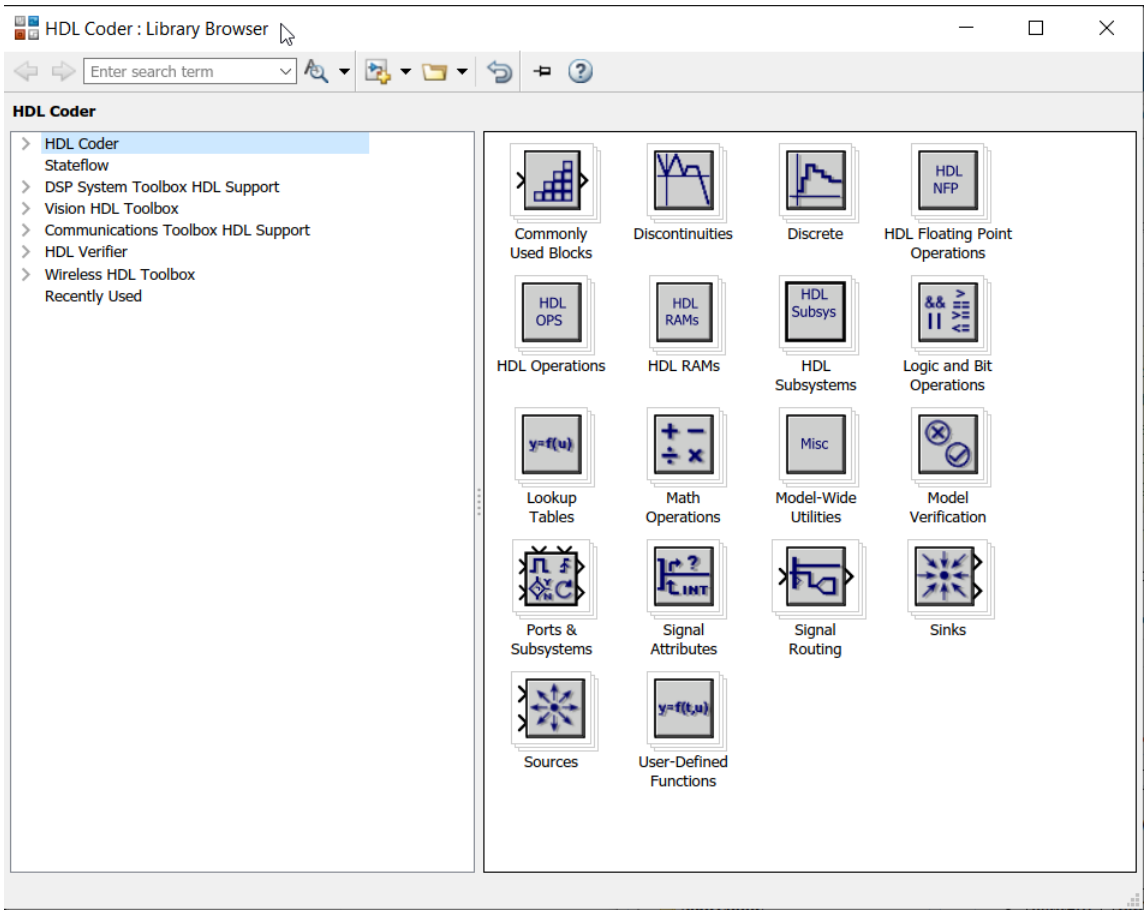


When you create a model for HDL code generation, you partition the model into a Design-Under-Test (DUT) and a test bench. The DUT is a Subsystem that is mostly at the top level of your

model, and contains the algorithm for which you generate HDL code. Blocks outside this Subsystem form the test bench, and contains inputs to the Subsystem and output values that are logged. The test bench ensures that the DUT functionality is as expected.

For the test bench, you can use blocks that are not supported for HDL code generation. In the **Blank DUT** template, the model has a **HDL_DUT** Subsystem that corresponds to the DUT. Blocks outside the **HDL_DUT** Subsystem form the test bench.

2 Open the **HDL Coder** Block Library for designing your counter algorithm. To filter the Simulink Library Browser to show block libraries that support HDL code generation, in the **Apps** tab, select **HDL Coder**. The **HDL Code** tab appears. Select **HDL Block Properties > Open HDL Block Library**.



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 HDL-supported blocks in these block libraries:

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

To restore the Library Browser to the default view, in the Library Browser, click the button.

Alternatively, at the MATLAB command-line, enter:

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

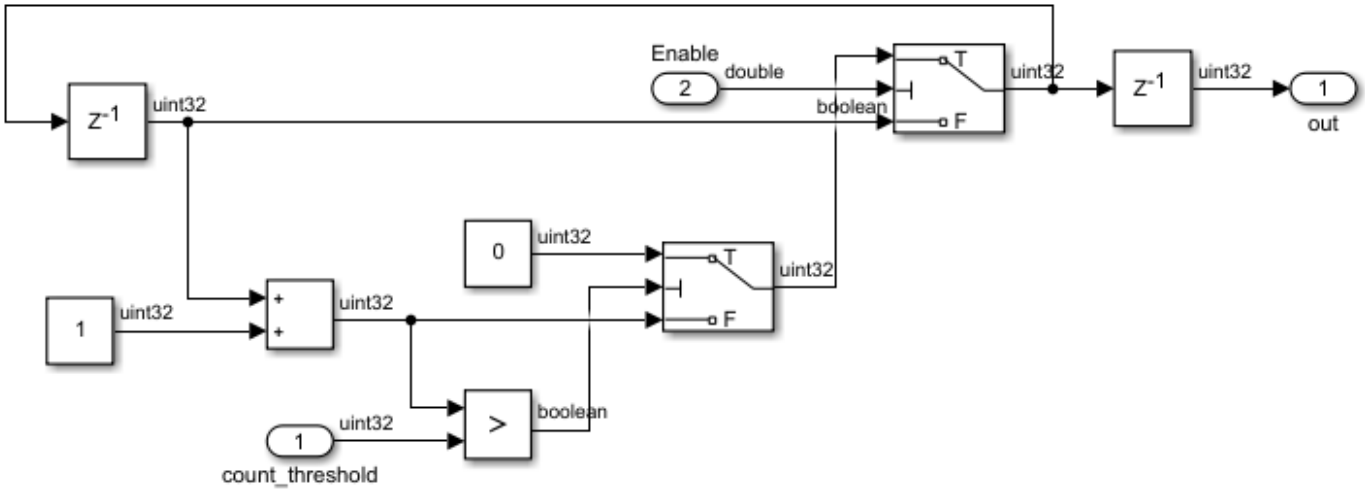
Note The set of supported blocks tend to change each release. Rebuild your supported blocks library each time you install a new version of this product.

Develop Design and Test Bench

1 Double-click the **HDL_DUT** Subsystem. Drag blocks from the **HDL Coder** library and add them to your model. This table illustrates the blocks to add to your model for designing the up counter. To learn about what a block does, and to specify the block parameters for that block, double-click the block.

Block	Library	Number of Blocks	Block Parameters
Constant	Sources	2	• Constant values: 1 and 0 • Output data type: uint32
Switch	Signal Routing	2	Criteria for passing first input: u2 > Threshold
Delay	Discrete	2	Delay length: 1
Add	Math Operations	1	Accumulator data type: Inherit: Same as first input
Relational Operator	Logic and Bit Operations	1	Relational operator: >

2 Rename the input ports `In1` and `In2` to `count_threshold` and `Enable` respectively. Place the blocks in your model and connect them as shown in figure.



The `Enable` signal specifies whether the counter should count upwards from the previous value. When the `Enable` signal is logic high, the counter counts up from zero to the `count_threshold` value. When the value of `out` becomes equal to the `count_threshold` value, the counter wraps back to zero and starts counting again. When the `Enable` signal becomes logic low, the counter holds the previous value.

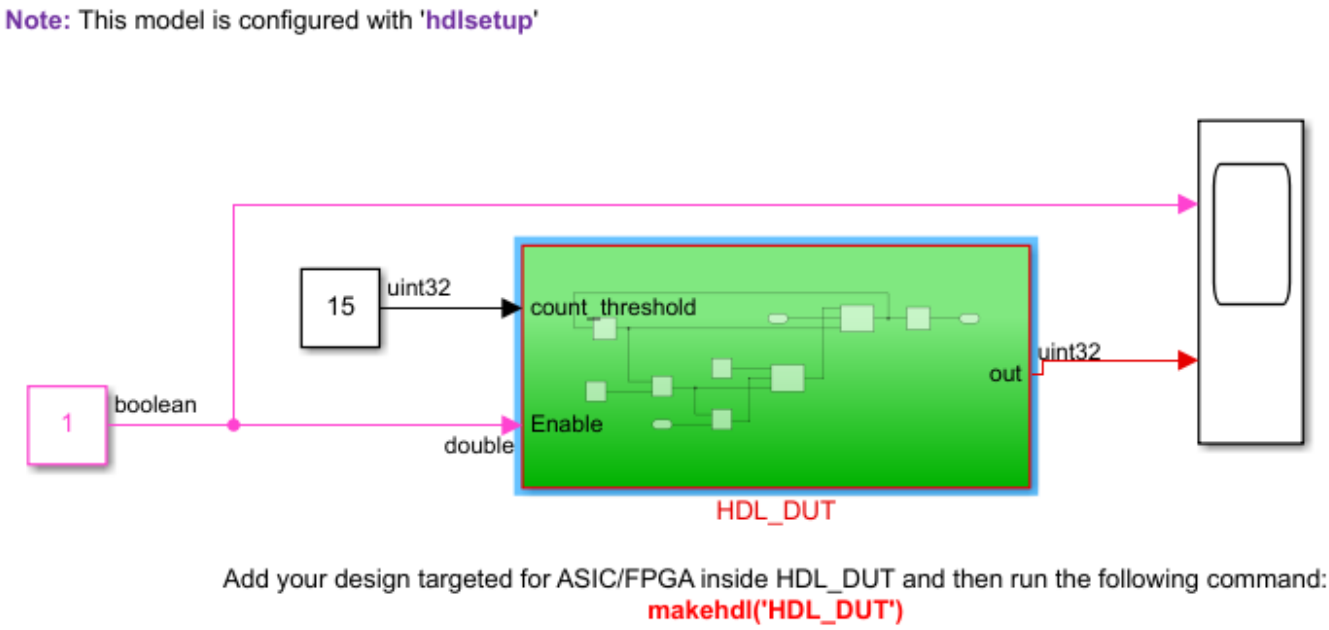
3 Navigate to the top level of the model and change the input settings.

- Constant block input to `count_threshold`: This input decides the maximum value up to which the counter should count. This example illustrates how to design a 4-bit up counter. Therefore, set the **Constant value** to `15` (`2^4 - 1`), and set the **Output data type** to `uint32`.

  Note Make sure that the output data type of this Constant block matches the output data type of the Constant blocks inside the **HDL_DUT** Subsystem.

- Counter Free-Running block input to `Enable`: For this example, remove the Counter Free-Running block. Replace this block with a Constant block that has a value of `1` and **Output data type** set to `boolean`.

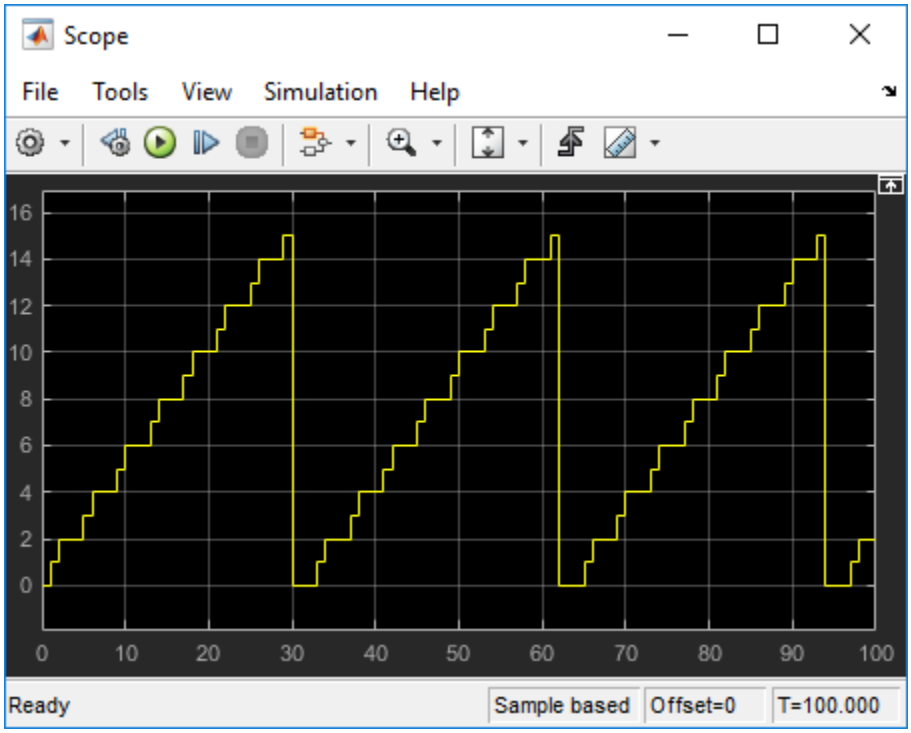
This figure shows the top level of your model after you have applied these settings.



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

Simulate and Verify Functionality of Design

Simulate your model by pressing the  button. To see the simulation results, open the Scope block at the top level of your model. The simulation results display the `Enable` signal at the top generating a constant value of `1` and the `out` signal counting from `0` to `15`, then wrapping back to zero, and then counting up again. This figure displays the waveform of the output signal `out`.



Check Model for HDL Compatibility

You have now simulated the model and verified the functionality of your design. Before you generate HDL code, you must verify that the model settings are compatible for HDL code generation. To make your design compatible for HDL code generation, you use the HDL Code Advisor. To learn how to use the HDL Code Advisor, see “Check HDL Compatibility of Model Using HDL Code Advisor” on page 3-7.

See Also

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

More About

- “Use Simulink Templates for HDL Code Generation”
- “Generate HDL Code from Simulink Model” on page 3-13
- “Verify Generated Code from Simulink Model Using HDL Test Bench” on page 3-17
- “HDL Code Generation and FPGA Synthesis Using Simulink HDL Workflow Advisor” on page 3-22

Check HDL Compatibility of Model Using HDL Code Advisor

In this section...

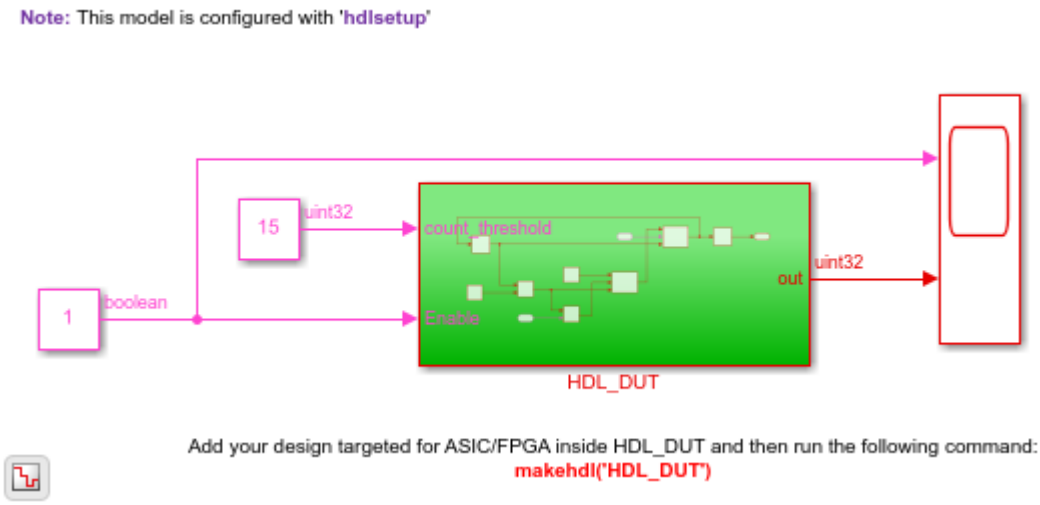
Before you can generate HDL code, it is recommended that you verify the compatibility of your algorithm modeled in Simulink for HDL code generation. To verify model compatibility, you use the HDL Code Advisor. The HDL Code Advisor verifies and updates your Simulink model or subsystem for compatibility with HDL code generation. The Code Advisor checks for model configuration settings, ports and subsystem settings, block settings, support for native floating point, and conformance to the industry-standard rules. The Code Advisor produces a report that lists suboptimal conditions or settings, and then proposes better model configuration settings.

This example shows how you can update a simple up counter model for HDL compatibility. To learn more about the counter algorithm and how you can create this model, see “Create Simulink Model for HDL Code Generation” on page 3-2.

Simple Up Counter Model

Open this model to see a simple up counter. The model counts up from zero to a threshold value and then wraps back to zero. In this model, the threshold value is set to `15`. You can change the threshold value by changing the value of the Constant block that is input to the `count_threshold` port. The Enable signal specifies whether the counter should count up or hold the previous value. The Enable signal is set to `1` which means that the counter counts upwards continuously.

```
open_system('hdlcoder_simple_up_counter.slx')
set_param('hdlcoder_simple_up_counter', 'SimulationCommand', 'Update')
```



Open the HDL Code Advisor

To open the HDL Code Advisor, in the **Apps** tab, select **HDL Coder**. The **HDL Code** tab appears. Select the DUT Subsystem and then click **HDL Code Advisor**.

Note You open the HDL Code Advisor and then run checks for the DUT Subsystem that you want to generate code for. The top level model can contain blocks that are not compatible for HDL code generation. Running the HDL Code Advisor for the entire model can flag these blocks and your model as incompatible for HDL code generation.

In the HDL Code Advisor, the left pane lists the folders in the hierarchy. Each folder represents a group or category of related checks. Expanding the folders shows available checks in each folder. From the left pane, you can select a folder or an individual check. The HDL Code Advisor displays information about the selected folder or check in the right pane. The content of the right pane depends on the selected folder or check. The right pane has a **Result** subpane that contains a display area for status messages and other task results.

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

How to Run Checks In the HDL Code Advisor

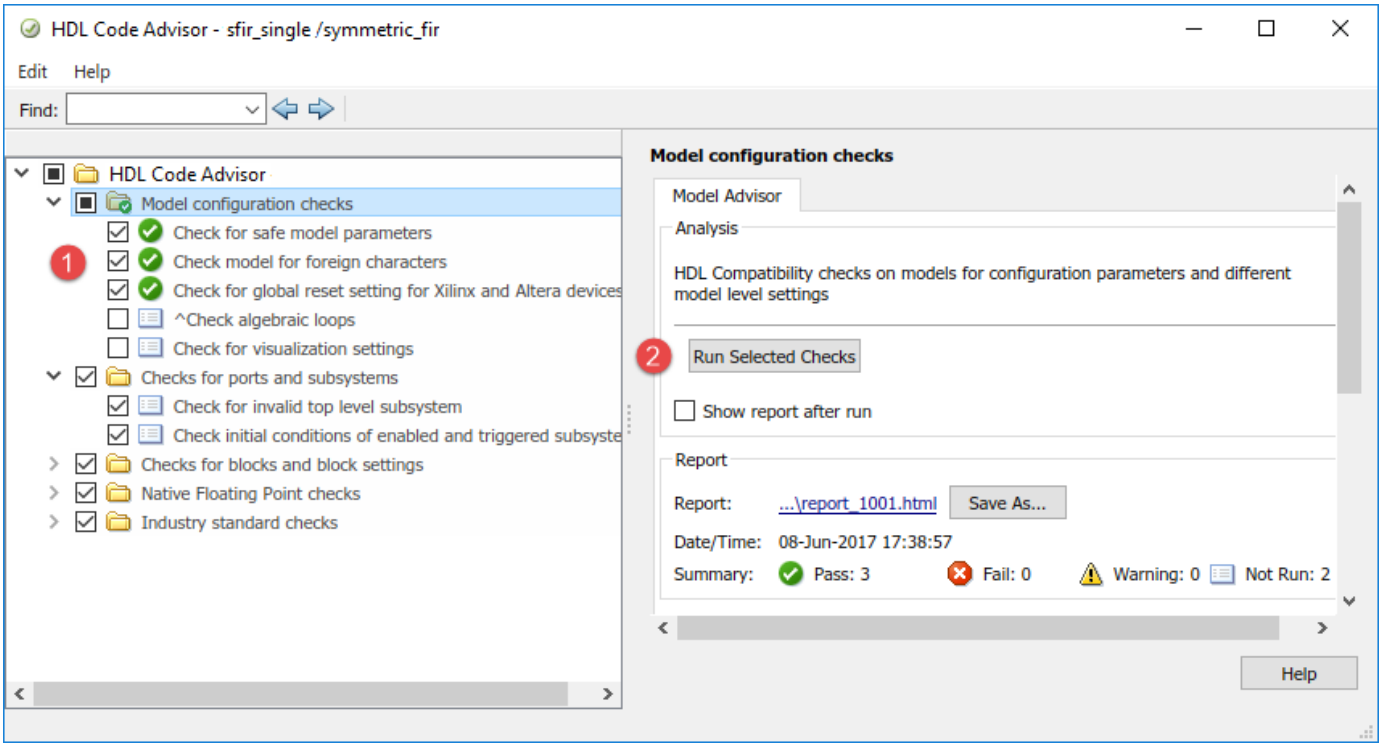
In the HDL Code Advisor window, you can run individual checks or a group of checks. To run a check, **Select** that check and then click **Run This Check**. For example, to run the **Check for safe model parameters**, select the check box, and then click **Run This Check**.

In the HDL Code Advisor window, you can run a group of checks within a folder.

1 Select the checks that you want to run.

2 Select the folder that contains these checks and then click **Run Selected Checks**.

This example shows how to run selected checks in the **Model configuration checks** folder.



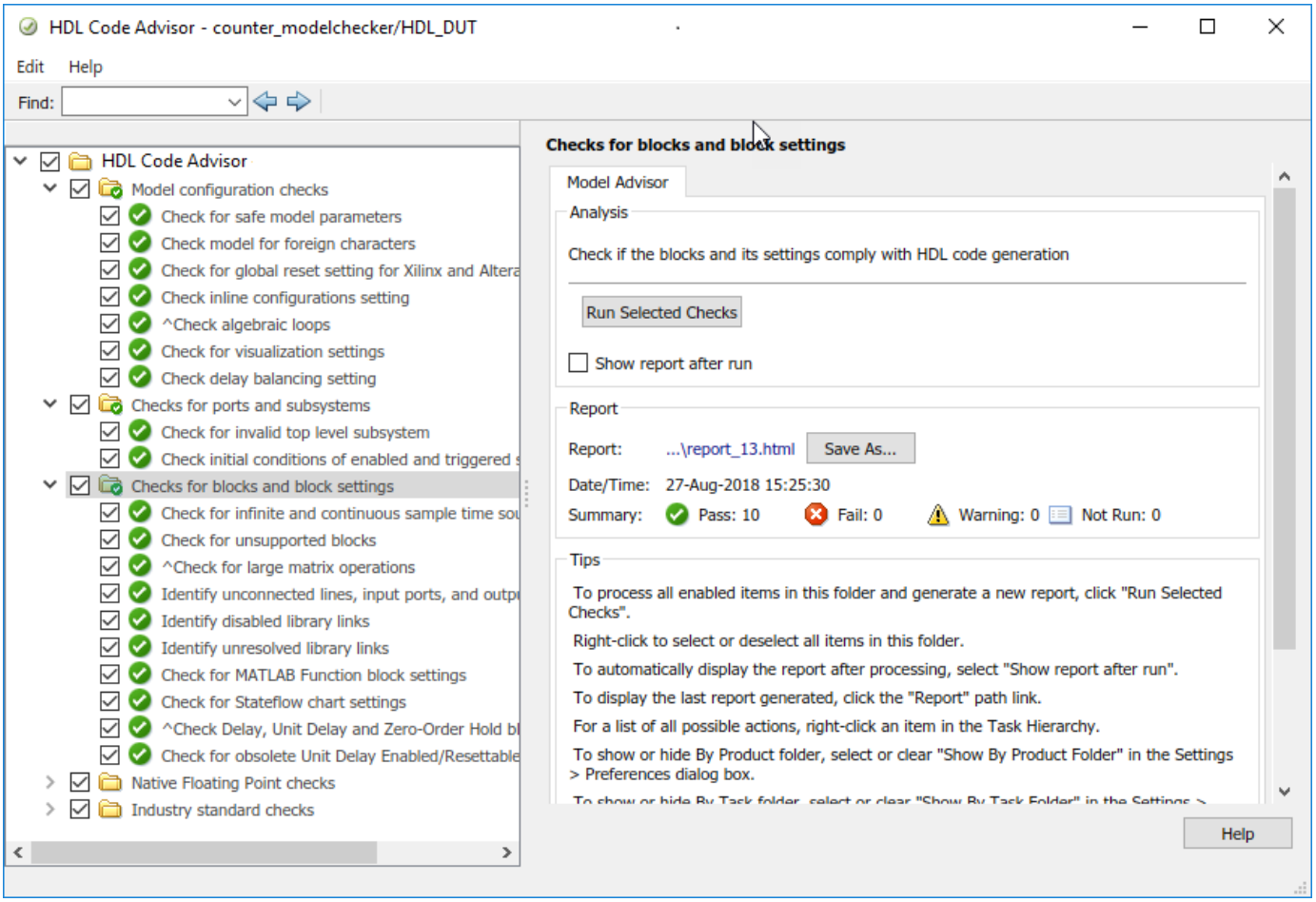
Run Checks for Counter Model

By using this approach, for your counter model, run all checks in these folders:

- **Model Configuration checks**
- **Checks for ports and subsystems**
- **Checks for blocks and block settings**

For this example, you do not need to run the checks in **Native Floating Point checks** and **Industry standard checks** folders. To learn more about these checks, see “HDL Code Advisor Checks”.

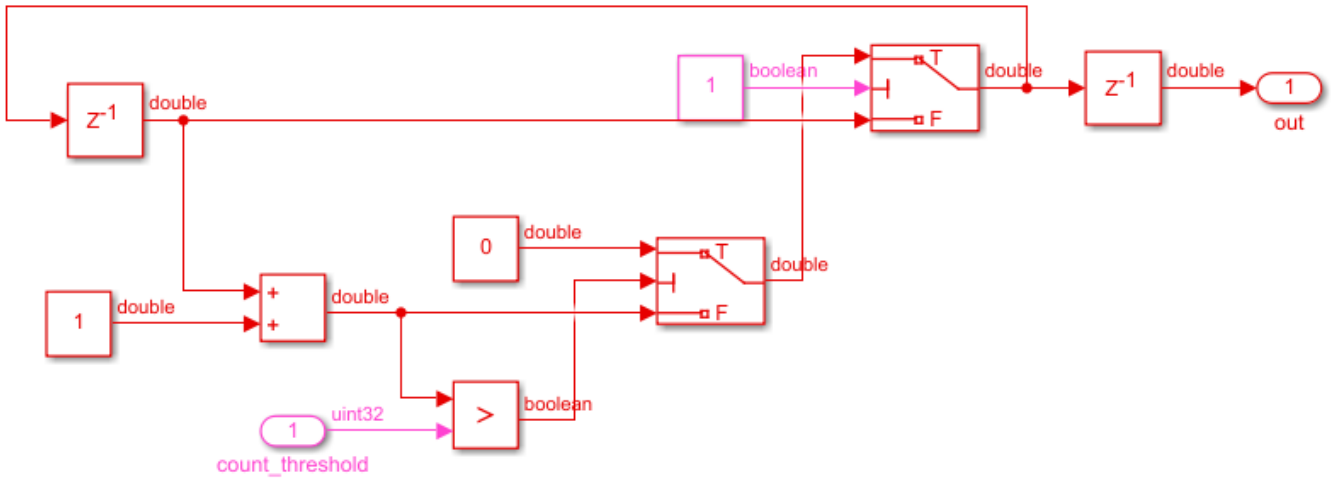
For the counter model, the checks display the results as **Passed**, which means that the model is compatible for HDL code generation.



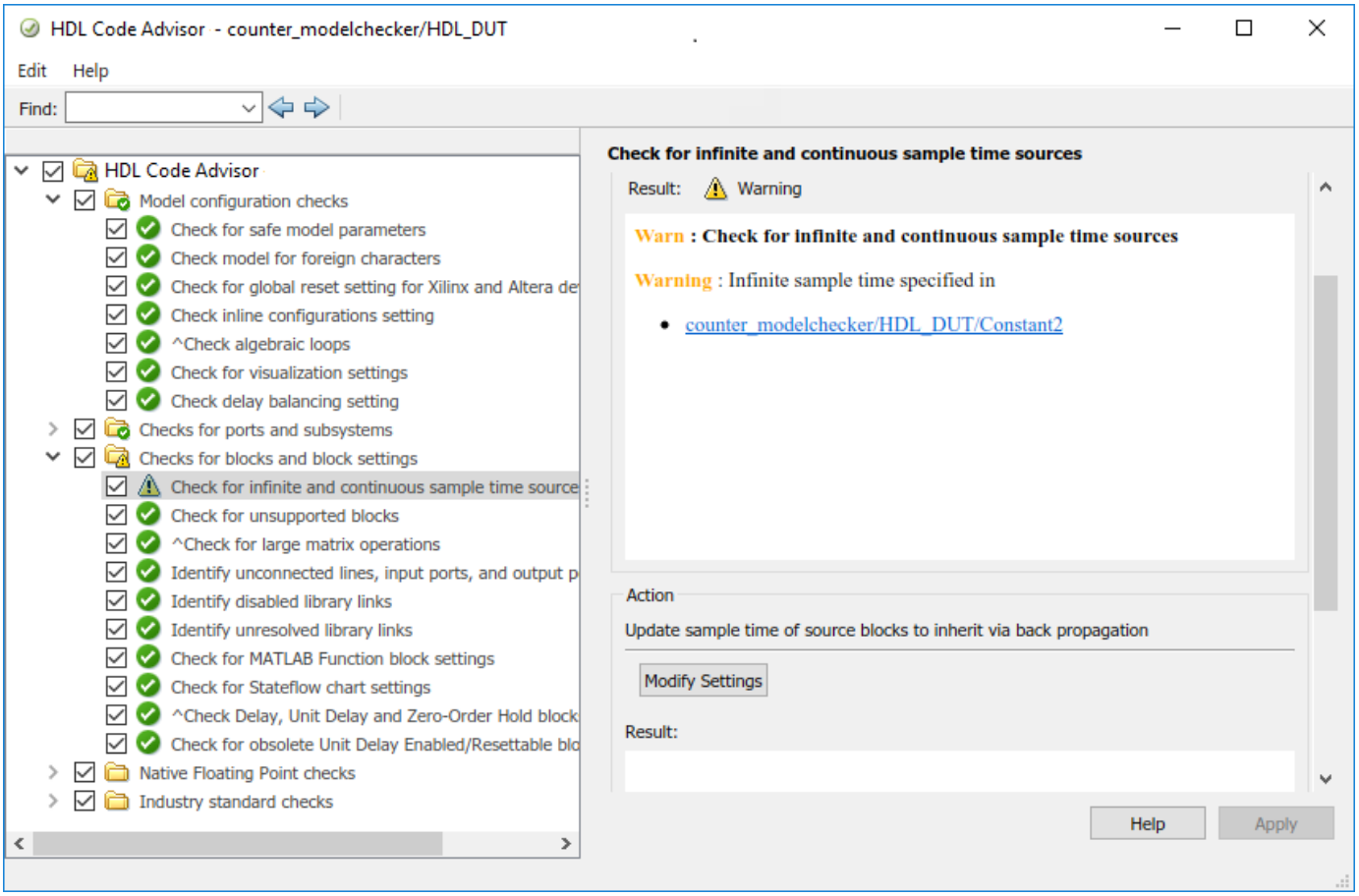
Fix HDL Code Advisor Warnings or Failures

In the HDL Code Advisor, if a check fails, the right pane shows the warning or failure information in a **Result** subpane. The **Result** subpane displays model settings that are not compliant. For some tasks, use the **Action** subpane to apply the Code Advisor recommended settings.

For example, inside the **HDL_DUT** Subsystem, consider that you remove the `Enable` port and replace this port with a Constant input that has a value of `1`.



Now, if you run the **Check for infinite and continuous sample time sources** check, the HDL Code Advisor displays this warning.



To apply the model configuration settings that the code generator reported in the **Result** subpane, click the **Modify Settings** button. After you click **Modify Settings**, the **Result** subpane reports the changes that were applied. In this example, the **Sample time** of the Constant block is reset to `-1`. You can now run this check.

Caveats

- If you reference one model in another by using a Model block, the HDL Code Advisor checks the model configurations or settings of the parent model. To check whether the referenced model is compatible with HDL code generation, open the HDL Code Advisor for the referenced model, and then run the checks.
- If you run the checks on masked library blocks in your Simulink model, the Code Advisor cannot verify whether the blocks inside the library blocks have HDL-compatible settings.
- When you apply Model Advisor checks to your model, it increases the likelihood that your model does not violate certain modeling standards or guidelines. However, it does not guarantee that the design is ready for HDL code generation. Make sure that you verify the design by using multiple methods for HDL code generation readiness.

Generate HDL Code

The counter model is now compatible for HDL code generation. You can generate HDL code for the **HDL_DUT** Subsystem, which contains the counter algorithm. To learn how to generate code, see “Generate HDL Code from Simulink Model” on page 3-13.

See Also

`checkhdl` | `hdlmodelchecker` | `hdlsetup`

More About

- “HDL Code Advisor Checks”
- “Verify Generated Code from Simulink Model Using HDL Test Bench” on page 3-17
- “HDL Code Generation and FPGA Synthesis Using Simulink HDL Workflow Advisor” on page 3-22

Generate HDL Code from Simulink Model

In this section...

This example shows how you can generate HDL code for a simple `Counter` model. The model is a simple up counter that counts up from zero to a threshold value and then wraps back to zero. Before you generate HDL code for this model, it is recommended that you verify the HDL compatibility of your model by using the HDL Code Advisor. To learn how to:

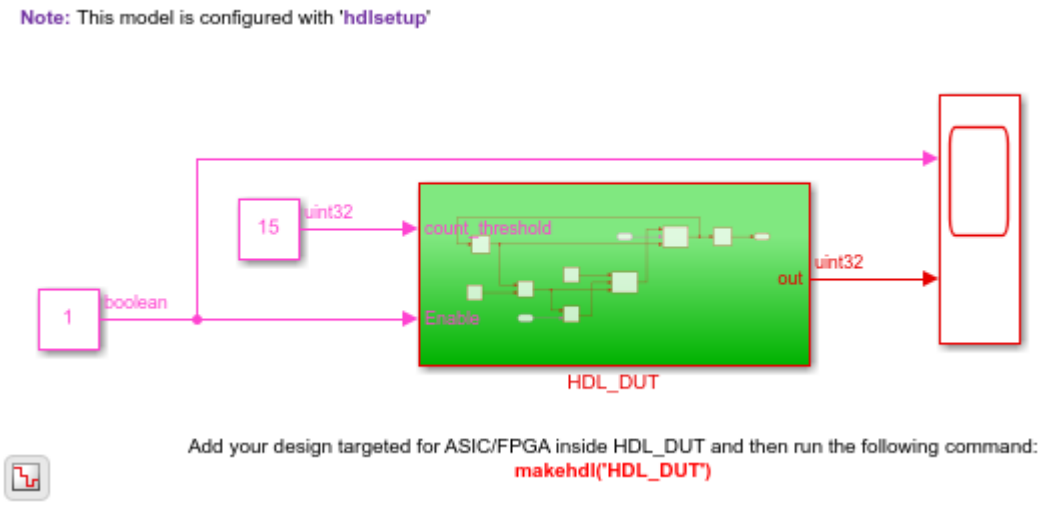
- Create this counter model, see “Create Simulink Model for HDL Code Generation” on page 3-2.
- Check HDL compatibility of the counter model, see “Check HDL Compatibility of Model Using HDL Code Advisor” on page 3-7.

By default, HDL Coder creates an `hdlsrc` folder in the current working folder to generate the HDL files. Therefore, before you proceed to generate HDL code, make sure that your current working folder is writeable.

Simple Up Counter Model

Open this model to see a simple up counter. The model counts up from zero to a threshold value and then wraps back to zero. In this model, the threshold value is set to `15`. You can change the threshold value by changing the value of the Constant block that is input to the `count_threshold` port. The Enable signal specifies whether the counter should count up or hold the previous value. The Enable signal is set to `1` which means that the counter counts upwards continuously.

```
open_system('hdlcoder_simple_up_counter.slx')
set_param('hdlcoder_simple_up_counter', 'SimulationCommand', 'Update')
```



Generate HDL Code

To generate code, you use the **HDL Code** tab in the To open the HDL Code Advisor, Select the DUT Subsystem and then click **HDL Code Advisor**.

For the up counter model, the **HDL_DUT** Subsystem is the DUT. To generate code for the DUT:

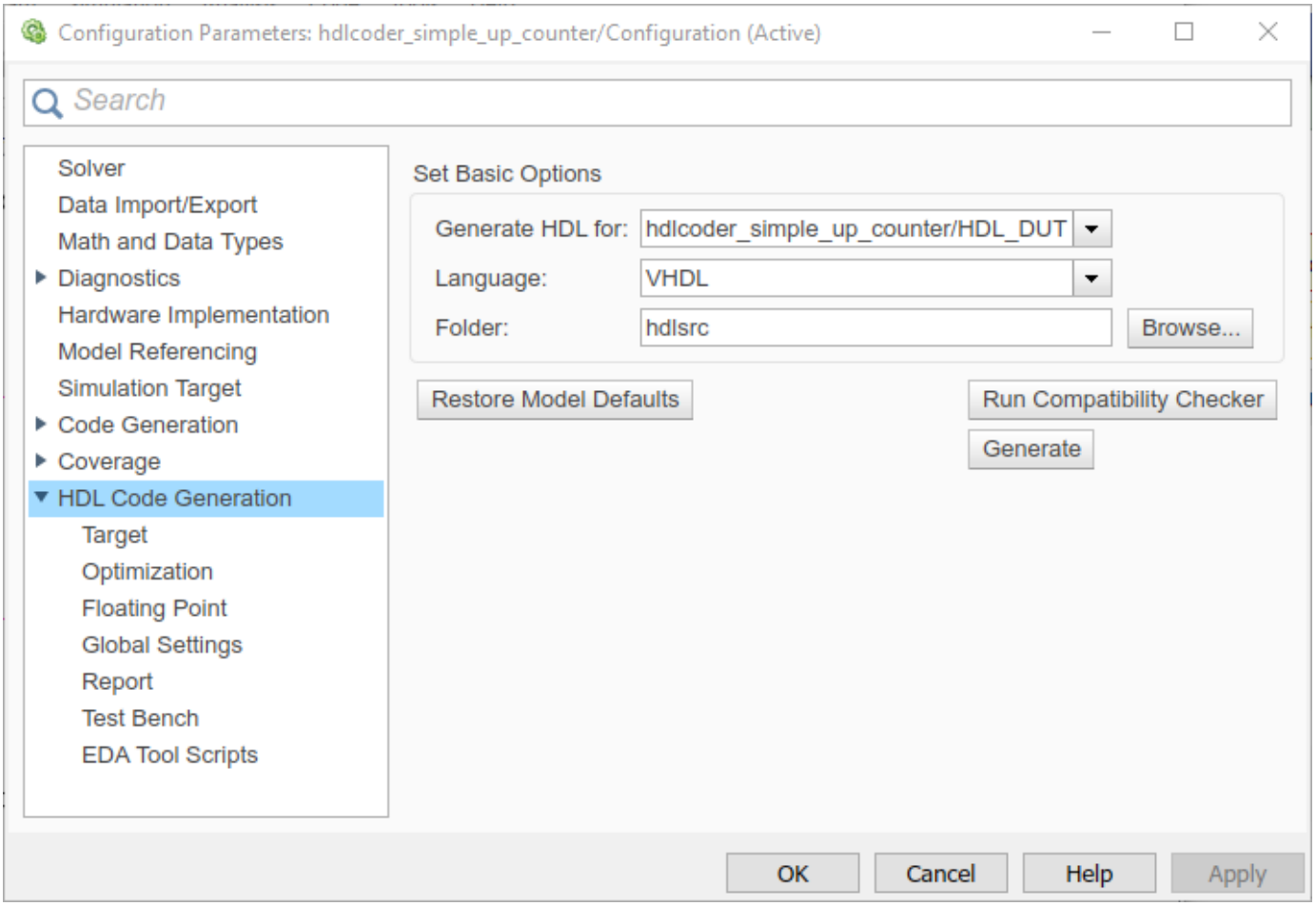
1. In the **Apps** tab, select **HDL Coder**. The **HDL Code** tab appears.
2. Select the DUT Subsystem in your model, and make sure that this Subsystem name appears in the **Code for** option. To remember the selection, you can pin this option. Click **Generate HDL Code**.

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

If you want to generate Verilog code, you can specify this setting in the **HDL Code Generation** pane of the Configuration Parameters dialog box. Before you generate code, you can also customize model-level settings for your design such as enable native floating-point support, generate resource and traceability reports, use model-level optimizations, and modify other global settings.

To generate Verilog code for the counter model:

1. In the **HDL Code** tab, click **Settings**.
2. In the **HDL Code Generation** pane, for **Language**, select `Verilog`. Leave other settings to the default. Click **Apply** and then click **Generate**.



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 in the MATLAB command line with:

- Link to the Configuration 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.

The process completes with the message:

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

View HDL Code Generation Files

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:

- `HDL_DUT.vhd`: This file is the VHDL code that contains the entity definition and RTL architecture implementing the up counter that you designed.

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

- `HDL_DUT_compile.do`: Mentor Graphics ModelSim compilation script. To invoke this script and compile the generated VHDL code, you use the `vcom` command.
- `HDL_DUT_synplify.tcl`: This file is a Synplify® synthesis TCL script.
- `HDL_DUT_map.txt`: This report file is a mapping file that generated entities or modules to the subsystems that generated them. See “Trace Code Using the Mapping File”.
- `HDL_DUT_report.html`: This file is a HDL Code Generation Check report displays the status of HDL code generation and any warnings or messages. If HDL code generation fails, you see the cause of failure in the Check report.
- `gm_hdlcoder_simple_up_counter.slx`: This file is a generated model that behaviorally represents the HDL code in the Simulink modeling environment. For more information, see “Generated Model and Validation Model”.

Verify Generated HDL Code

Before you proceed to deploy your design on the target hardware, you must verify the generated HDL code. 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 3-17.

See Also

`hdlset_param` | `hdlsetup` | `makehdl`

More About

- “Create Simulink Model for HDL Code Generation” on page 3-2
- “HDL Code Generation and FPGA Synthesis Using Simulink HDL Workflow Advisor” on page 3-22

Verify Generated Code from Simulink Model Using HDL Test Bench

In this section...

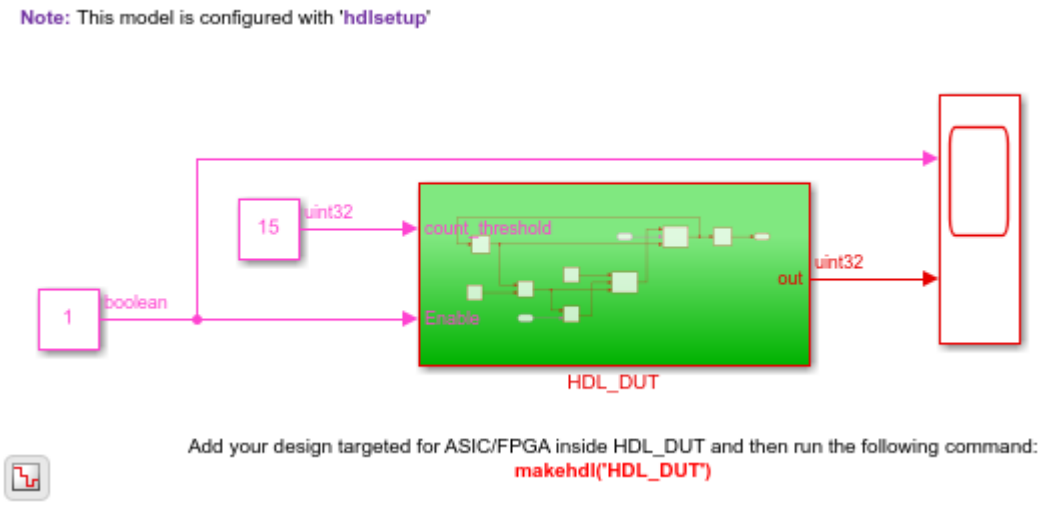
This example shows how to generate a HDL test bench and verify the generated code for your design. The example assumes that you have already generated HDL code for your model. This example illustrates how to verify the generated code for a simple up counter. To learn more about this counter model and how to generate HDL code, see “Generate HDL Code from Simulink Model” on page 3-13.

Note If you haven't already generated HDL code, you can still open this model and generate the HDL test bench. Before generating the test bench, HDL Coder runs code generation to make sure that there is at least one successful code generation run before generating the testbench.

Simple Up Counter Model

Open this model to see a simple up counter. The model counts up from zero to a threshold value and then wraps back to zero. In this model, the threshold value is set to 15. You can change the threshold value by changing the value of the Constant block that is input to the `count_threshold` port. The Enable signal specifies whether the counter should count up or hold the previous value. The Enable signal is set to 1 which means that the counter counts upwards continuously.

```
open_system('hdlcoder_simple_up_counter.slx')
set_param('hdlcoder_simple_up_counter', 'SimulationCommand', 'Update')
```



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”
SystemVerilog DPI Test Bench	“SystemVerilog DPI Test Bench”
FPGA-in-the-Loop	“FPGA-in-the-Loop”

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, 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.

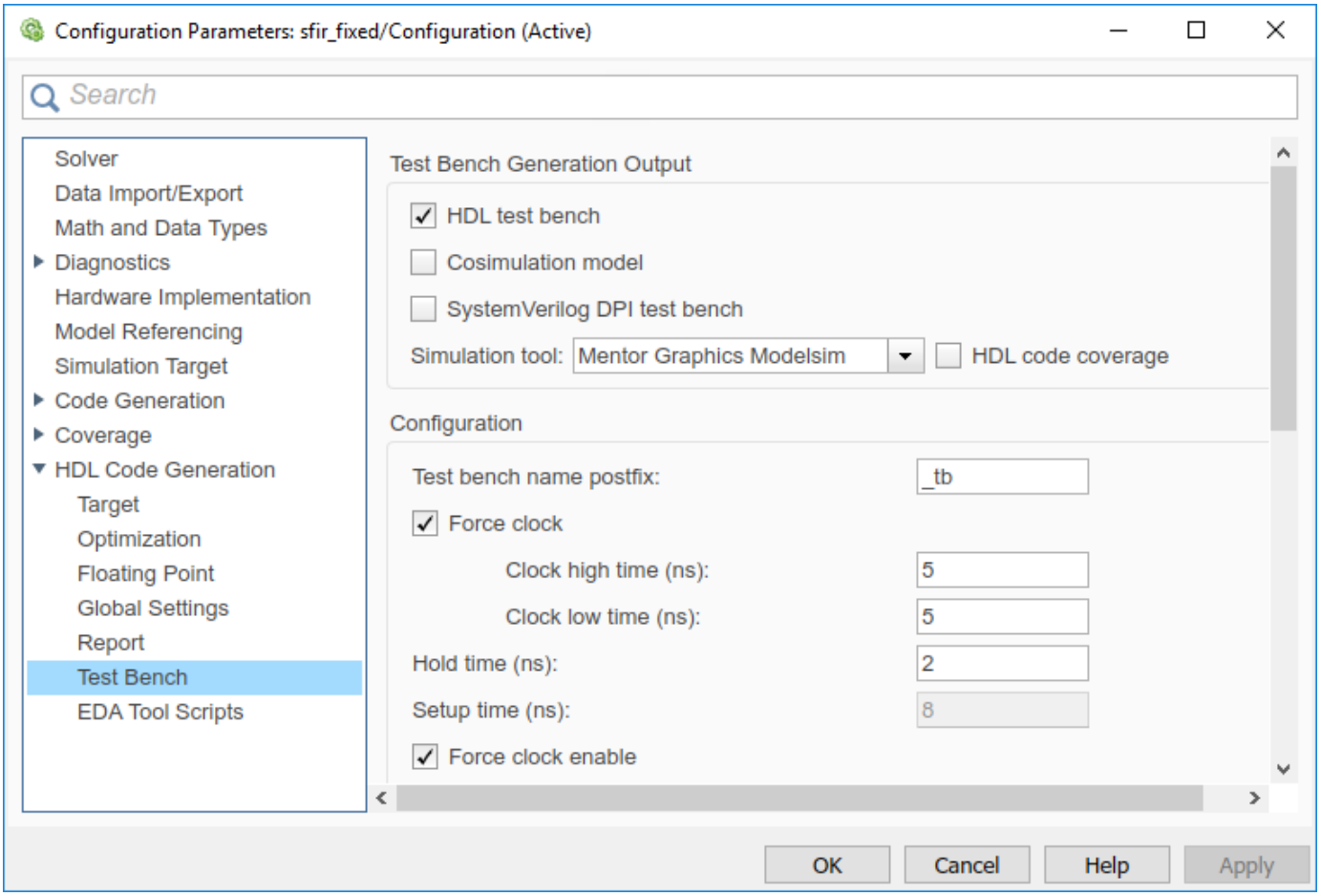
For the up counter model, the **HDL_DUT** Subsystem is the DUT. To generate the testbench, select this Subsystem. You cannot generate a HDL testbench for an entire model.

1. In the **Apps** tab, select **HDL Coder**. The **HDL Code** tab appears.
2. Select the DUT Subsystem in your model, and make sure that this Subsystem name appears in the **Code for** option. To remember the selection, you can pin this option. Click **Generate Testbench**.

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

To generate Verilog testbench code for the counter model:

1. In the **HDL Code** tab, click **Settings**.
2. In the **HDL Code Generation** pane, for **Language**, select `Verilog`. Leave other settings to the default.
3. In the **HDL Code Generation > Test Bench**pane, click **Generate Test Bench**.



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 `HDL_DUT` Subsystem in your up counter model, the folder contains:

   - `HDL_DUT_tb.vhd`: VHDL test bench code, with generated test and output data.

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

   - `HDL_DUT_tb_pkg.vhd`: Package file for VHDL test bench code. This file is not generated if you specified Verilog as the target language.
   - `HDL_DUT_tb_compile.do`: Mentor Graphics ModelSim compilation script (`vcom` commands). This script compiles and loads the entity to be tested (`HDL_DUT.vhd`) and the test bench code (`HDL_DUT_tb.vhd`).
   - `HDL_DUT_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 `HDL_DUT_tb.vhd` or `HDL_DUT_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\10.6b\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 `hdlcoder_simple_up_counter/HDL_DUT` 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 `hdlcoder_simple_up_counter/HDL_DUT` Subsystem, run this command to compile the generated code.

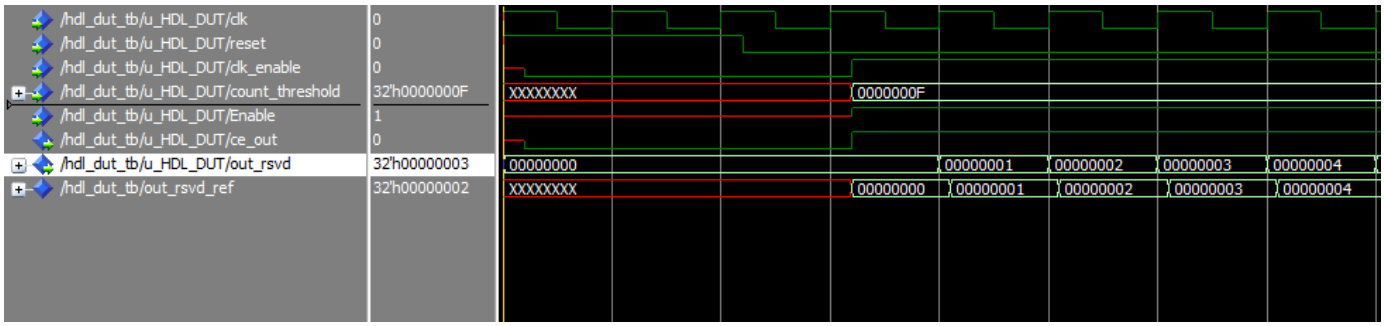
   ```
   QuestaSim>do HDL_DUT_tb_compile.do
   ```

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 `hdlcoder_simple_up_counter/HDL_DUT` Subsystem, run this command to simulate the generated code.

```
QuestaSim>do HDL_DUT_tb_sim.do
```

The simulator optimizes your design and displays the results of simulating your HDL design in a **wave** window. if you don't see the simulation results, open the **wave** window. The simulation script displays inputs and outputs in the model including the clock , reset, and clock enable signals 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.

Deploy Generated HDL Code on Target Device

After you verified the functionality of your HDL design, you can deploy the generated code on a target FPGA device. For deployment, you use the Simulink HDL Workflow Advisor. To learn how more, see “HDL Code Generation and FPGA Synthesis Using Simulink HDL Workflow Advisor” on page 3-22.

See Also

`makehdl` | `makehdltb`

More About

- “Test Bench Generation Output”
- “HDL Test Bench”
- “HDL Code Generation and FPGA Synthesis Using Simulink HDL Workflow Advisor” on page 3-22

HDL Code Generation and FPGA Synthesis Using Simulink HDL Workflow Advisor

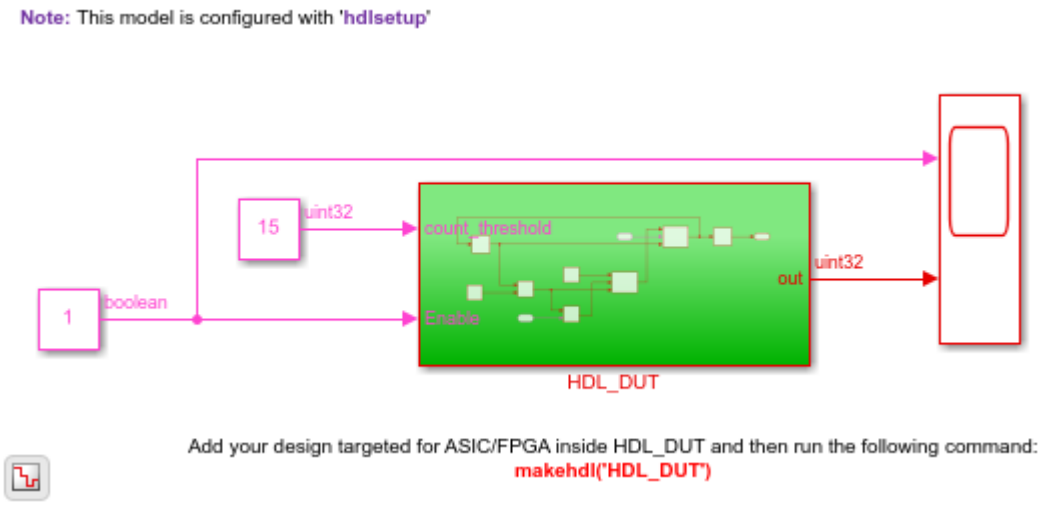
In this section...

This example shows how you can use the HDL Workflow Advisor to generate HDL code and synthesize your design on a target Xilinx FPGA. For this tutorial, you can use a simple up counter model that you created 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. To learn how to create this model, see “Create Simulink Model for HDL Code Generation” on page 3-2.

Simple Up Counter Model

Open this model to see a simple up counter. The model counts up from zero to a threshold value and then wraps back to zero. In this model, the threshold value is set to `15`. You can change the threshold value by changing the value of the Constant block that is input to the `count_threshold` port. The Enable signal specifies whether the counter should count up or hold the previous value. The Enable signal is set to `1` which means that the counter counts upwards continuously.

```
open_system('hdlcoder_simple_up_counter.slx')
set_param('hdlcoder_simple_up_counter', 'SimulationCommand', 'Update')
```



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 2019.1 executable. To learn about the latest supported tools, see “HDL Language Support and Supported Third-Party Tools and Hardware” on page 1-3.

```
hdlsetuptoolpath('ToolName','Xilinx Vivado','ToolPath',...
 'C:\Xilinx\Vivado\2019.1\bin\vivado.bat');
```

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

Open the HDL Workflow Advisor

To start the HDL Workflow Advisor from a Simulink model,

1. In the **Apps** tab, select **HDL Coder**. The **HDL Code** tab appears.
2. Select the DUT Subsystem in your model, and make sure that this Subsystem name appears in the **Code for** option. To remember the selection, you can pin this option. Click **Workflow Advisor**.

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**.

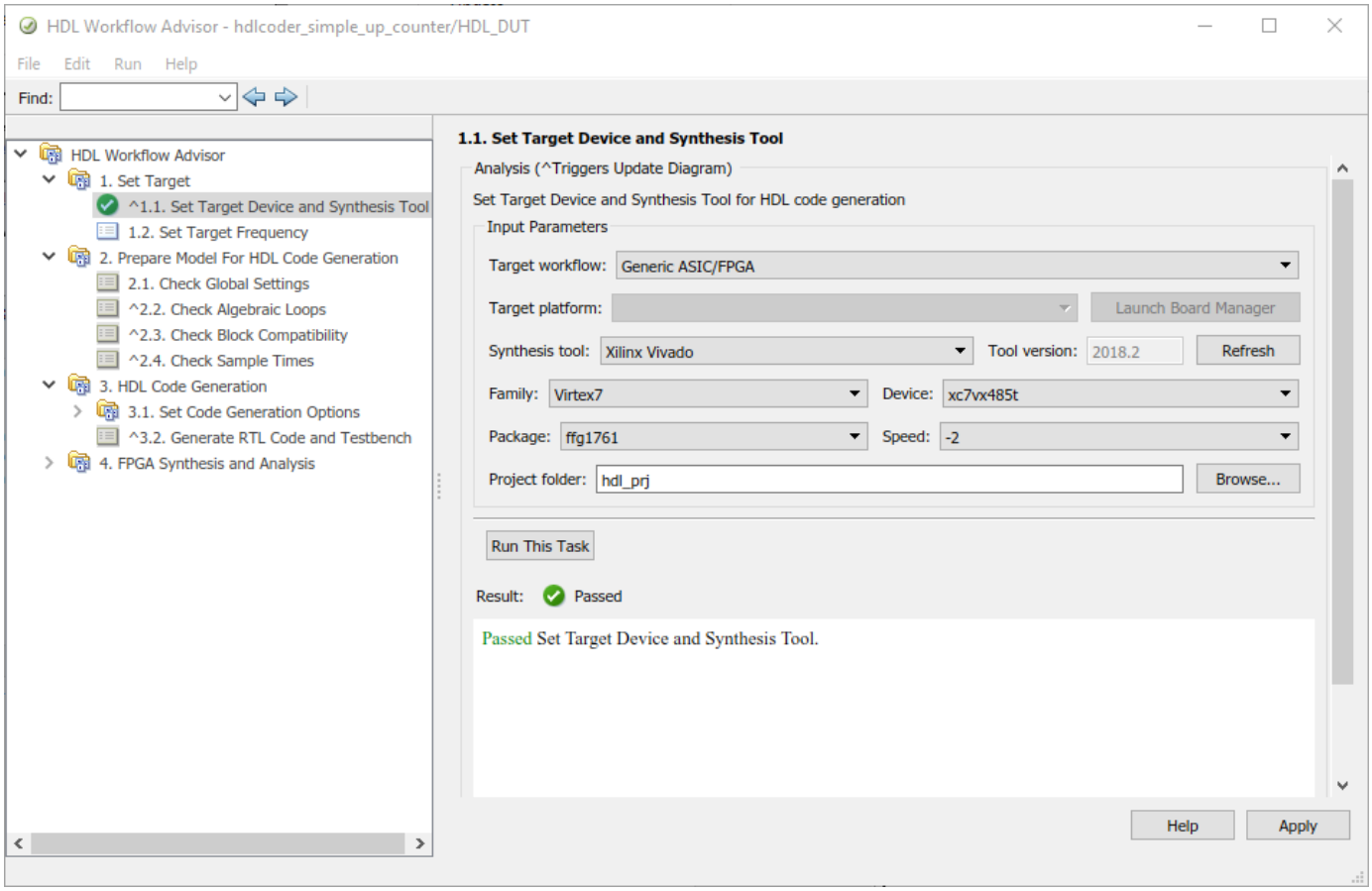
In the HDL Workflow Advisor, the left pane lists the folders in the hierarchy. Each folder represents a group or category of related tasks. Expanding the folders shows available tasks in each folder. From the left pane, you can select a folder or an individual task. The HDL Workflow Advisor displays 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”.

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. In **Set Target Frequency** task, specify a “Target Frequency” that you want the design to achieve. For this example, you can set **Target Frequency (MHz)** to `200`.
3. 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.

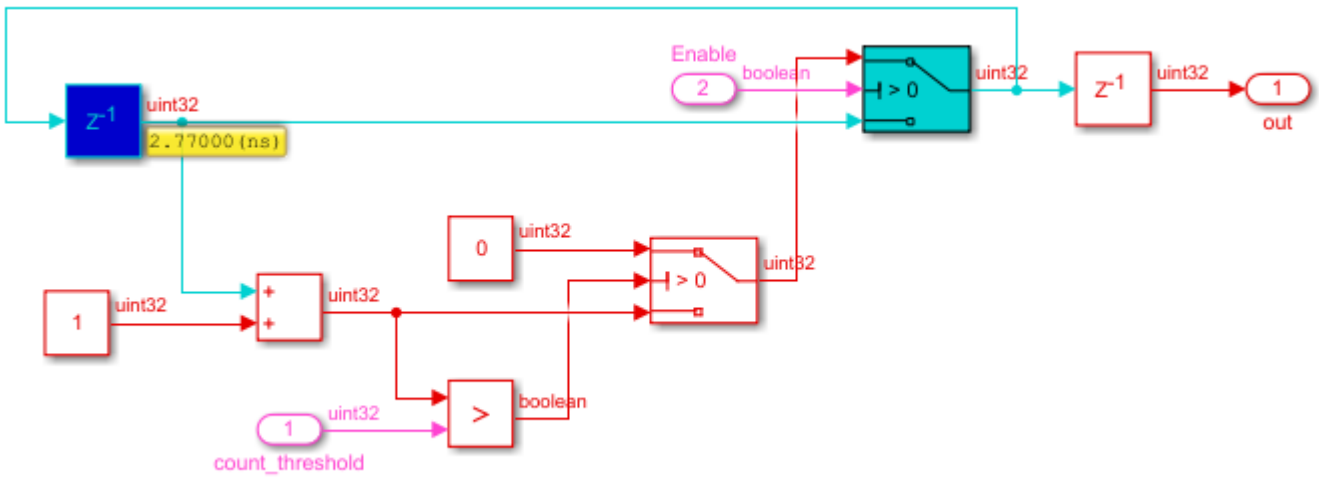
   Note If running a task generates a warning, select **Modify All**, and rerun the task.
4. To modify code generation options, use the tasks in **Set Code Generation Options**. For example, to customize the target HDL language and the target code generation folder, use the **Set Basic Options** task. After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

   Note If you want to generate an HDL test bench or a validation model, you can specify the corresponding settings in the **Generate RTL Code and Testbench** task. To specify additional test bench options, use the **Set Testbench Options** task.

Perform FPGA Synthesis and Analysis

1. In the **FPGA Synthesis and Analysis > Perform Synthesis and P/R > Perform Place and Route** task, clear **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.

Note If you select `Intel Quartus Pro` or `Microsemi Libero SoC` as the **Synthesis tool**, the **Annotate Model with Synthesis Result** task is not available. In this case, you can run the workflow to synthesis and then view the timing reports to see the critical path.



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 Simulink Model for HDL Code Generation” on page 3-2
- “Generate HDL Code from Simulink Model” on page 3-13