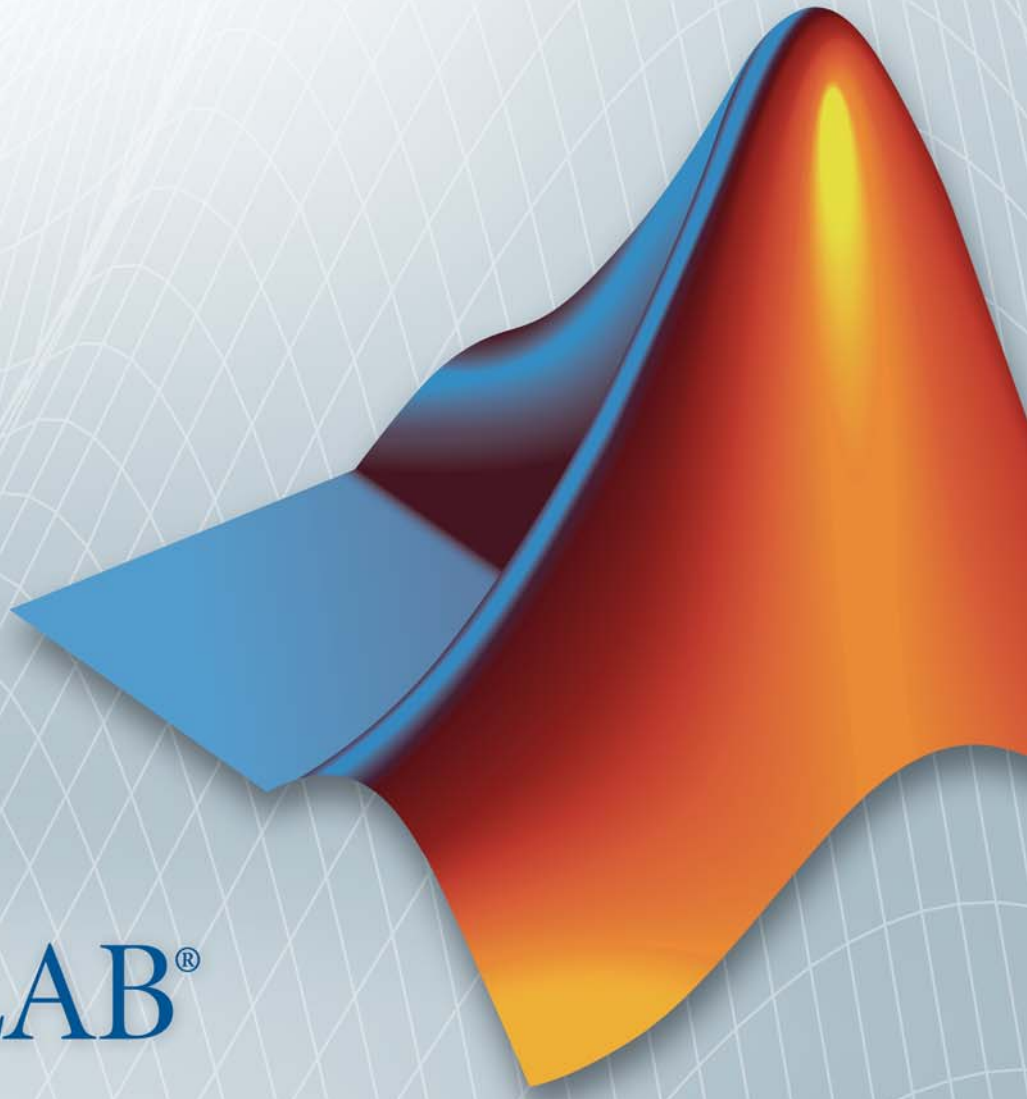
HDL Coder™

Getting Started Guide

R2012a



MATLAB®

MathWorks®

How to Contact MathWorks

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

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

HDL Coder™ Getting Started Guide

© COPYRIGHT 2012 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)

Contents

1 Getting Started with HDL Coder

2 Tutorials

Examples

A

Index

1

Getting Started with 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® and Altera® 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 HDL 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 System objects and Stateflow charts
- Mealy and Moore finite-state machines and control logic implementations
- Workflow advisor for programming Xilinx and Altera application boards
- Resource sharing and retiming for area-speed tradeoffs
- Code-to-model and model-to-code traceability for DO-254
- Legacy code integration

Installation

For instructions on installing MathWorks® products, see the MATLAB installation documentation for your platform. If you have installed MATLAB and want to see which other MathWorks products are installed, in the MATLAB Command Window, enter `ver`.

Toolbox Setup

In this section...

VHDL and Verilog Language Support

The generated HDL code complies with the following standards:

- VHDL versions 93 and 02
- Verilog-2001 (IEEE 1364-2001) or later

Setting Up the C/C++ Compiler

Before using HDL Coder, you must set up your C/C++ compiler by running the `mex -setup` command, as described in the documentation for `mex` in the MATLAB Function Reference. You must run this command even if you use the default C compiler that comes with the MATLAB product for Microsoft® Windows® platforms. You can also use `mex` to choose and configure a different C/C++ compiler, as described in "Building MEX-Files" in the MATLAB External Interfaces documentation.

For a list of supported compilers, see at `http://www.mathworks.com/support/compilers/current_release/`.

Supported Third Party Synthesis Tools

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.

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

- Xilinx ISE 13.1
- Altera Quartus II 11.0
- Xilinx ISE 10.1 is supported only for compatibility with Speedgoat FPGA target devices.

  Speedgoat IO301, IO303, and IO311 FPGA IO boards, which use Xilinx Virtex-II FPGAs, are tested with Xilinx ISE version 10.1. Before you select one of these Speedgoat devices in the HDL Workflow Advisor, make sure that you have installed Xilinx ISE 10.1. See also "Workflow for Speedgoat FPGA IO Boards and xPC Target™" for more information.

Setting Up the Synthesis Tool Path

If you plan to use HDL Coder with one of the supported third-party FPGA synthesis tools, you need to add the tools to your system path.

To permanently add a synthesis tool to your path:

1. Add the synthesis tool directory to your system path using `setenv`.
2. Close MATLAB.
3. Reopen MATLAB.

To add a synthesis tool to your path for the current MATLAB session:

1. Close open projects.
2. Close MATLAB.
3. Reopen MATLAB.
4. Use `hdlsetuptoolpath` to add the synthesis tool.

The syntax and operation of `hdlsetuptoolpath` are as follows:

```
hdlsetuptoolpath ('ToolName', TOOLNAME, 'ToolPath', TOOLPATH)
```

The input property-value pairs are:

- `'ToolName', ['Xilinx ISE' | 'Altera Quartus II']`: specify the synthesis tool name.
- `'ToolPath','path'`: specify the full path to the synthesis tool executable.

For example, the following command sets the synthesis tool path to point to an installed Xilinx ISE 13.1 executable.

```
 hdlsetuptoolpath('ToolName','Xilinx ISE', ...
'ToolPath', 'C:\Xilinx\13.1\ISE_DS\ISE\bin\nt64\ise.exe');
```

To see your Xilinx ISE synthesis tool path, use the following command:

```
!which ise
```

To see your Altera Quartus synthesis tool path, use the following command:

```
!which quartus
```

Tip `hdlsetuptoolpath` changes the system path and system environment variables for the current MATLAB session only. To execute `hdlsetuptoolpath` automatically when MATLAB starts, add `hdlsetuptoolpath` to your `startup.m` script.

Software Requirements for Demos

To operate some demos shipped with this release, the following related products are required:

- DSP System Toolbox™
- Filter Design HDL Coder™
- HDL Verifier™
- Communications System Toolbox™ (required to use Viterbi Decoder demo)
- Image Processing Toolbox™ (required to use Image Reconstruction demos)

2

Tutorials

HDL Code Generation from MATLAB Code

In this section...

About the Algorithm in This Example

For the purpose of this example, you generate and synthesize HDL code for a MATLAB algorithm that implements a simple filter. However, you can use HDL Coder to generate HDL code from MATLAB algorithms for many applications.

This tutorial uses these files:

- `mlhdlc_sfir.m` — Simple filter function from which you generate HDL code.
- `mlhdlc_sfir_tb.m` — Test bench that the HDL Coder project uses to exercise the filter using a representative input range.

mlhdlc_sfir Function Code

The following code provides the complete `mlhdlc_sfir` function definition.

```
%#codegen
function [y_out, delayed_xout] = mlhdlc_sfir(x_in, h_in1, h_in2, h_in3, h_in4)
% Symmetric FIR Filter

persistent ud1 ud2 ud3 ud4 ud5 ud6 ud7 ud8;
if isempty(ud1)
    ud1 = 0; ud2 = 0; ud3 = 0; ud4 = 0; ud5 = 0; ud6 = 0; ud7 = 0; ud8 = 0;
end

a1 = ud1 + ud8; a2 = ud2 + ud7;
a3 = ud3 + ud6; a4 = ud4 + ud5;

m1 = h_in1 * a1; m2 = h_in2 * a2;
m3 = h_in3 * a3; m4 = h_in4 * a4;

a5 = m1 + m2; a6 = m3 + m4;

% filtered output
y_out = a5 + a6;
% delayout input signal
delayed_xout = ud8;

% update the delay line
ud8 = ud7;
ud7 = ud6;
ud6 = ud5;
ud5 = ud4;
ud4 = ud3;
ud3 = ud2;
ud2 = ud1;
ud1 = x_in;

end
```

mlhdlc_sfir_tb.m Test Bench

The `mlhdlc_sfir_tb` test bench creates an input signal and calls the `mlhdlc_sfir` filter, passing in the input data.

```
clear all;

% input signal with noise
x_in = cos(2.*pi.*(0:0.001:2).*(1+(0:0.001:2).*75)).';

% filter coefficients
h1 = -0.1339; h2 = -0.0838; h3 = 0.2026; h4 = 0.4064;

len = length(x_in);
y_out = zeros(1,len);
x_out = zeros(1,len);

for ii=1:len
    data = x_in(ii);
    % call to the design 'mlhdlc_sfir' that is targeted for hardware
    [y_out(ii), x_out(ii)] = mlhdlc_sfir(data, h1, h2, h3, h4);
end

figure('Name', [mfilename, '_plot']);
subplot(2,1,1); plot(1:len,x_in);
subplot(2,1,2); plot(1:len,y_out);
```

Copying Files Locally

Before you begin generating code, set up a working folder and copy the tutorial files to this folder.

1. Start MATLAB.
2. Create a folder named `filter_sfir`, for example:

   ```
   mkdir filter_sfir
   ```

   The folder must not be within the MATLAB directory structure. You must be able to write to this folder.

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

Setting Up Your C Compiler

Before using HDL Coder to generate HDL code, you must set up your C compiler.

Note If your installation does not include a default compiler, for a list of supported compilers for the current release of MATLAB, see at http://www.mathworks.com/support/compilers/current_release/. Install a compiler that is suitable for your platform.

To set up the installed compiler:

1 At the MATLAB command line, enter:

```
mex -setup
```

2 Enter `y` to see the list of installed compilers.

3 Select a supported compiler.

4 Enter `y` to verify your choice.

Checking Your Synthesis Tool Setup

Before using HDL Coder to synthesize HDL code, you must set up your synthesis tool path.

To view your Xilinx ISE synthesis tool path, use the following command:

```
!which ise
```

To view your Altera Quartus synthesis tool path, use the following command:

```
!which quartus
```

If you do not have a synthesis tool set up, see “Setting Up the Synthesis Tool Path” on page 1-5.

Testing the Original MATLAB Algorithm

Before generating HDL code for this MATLAB algorithm, simulate your MATLAB design to verify that it runs, and to provide a baseline for comparison with the generated HDL code.

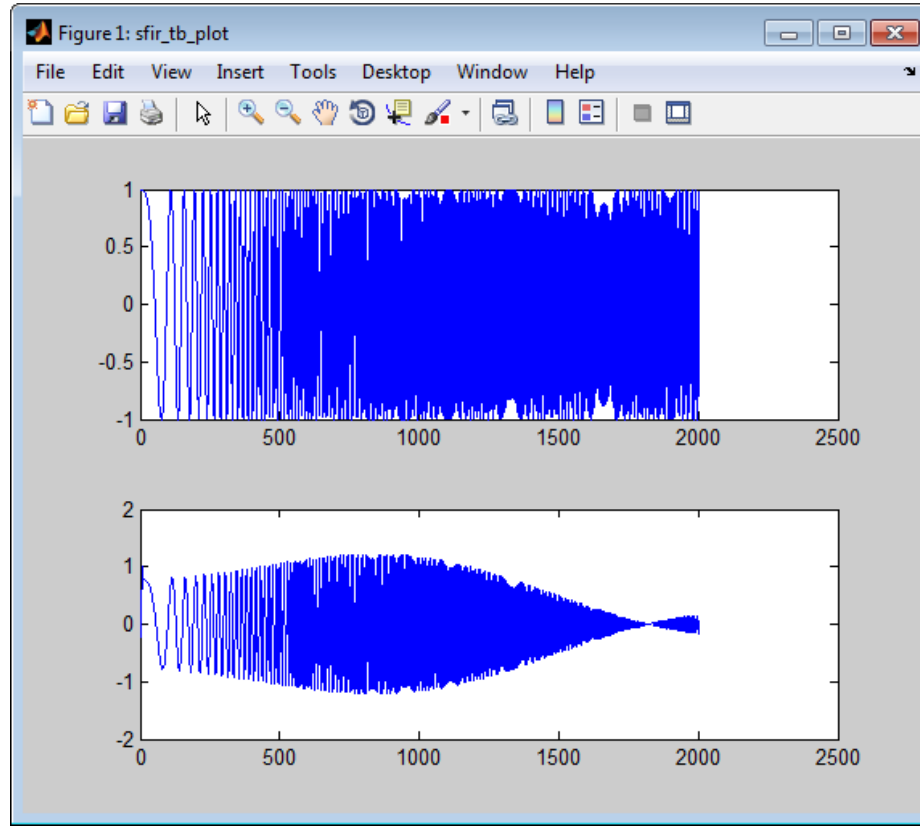
1. Make the `filter_sfir` folder your working folder, for example:

   ```
   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.

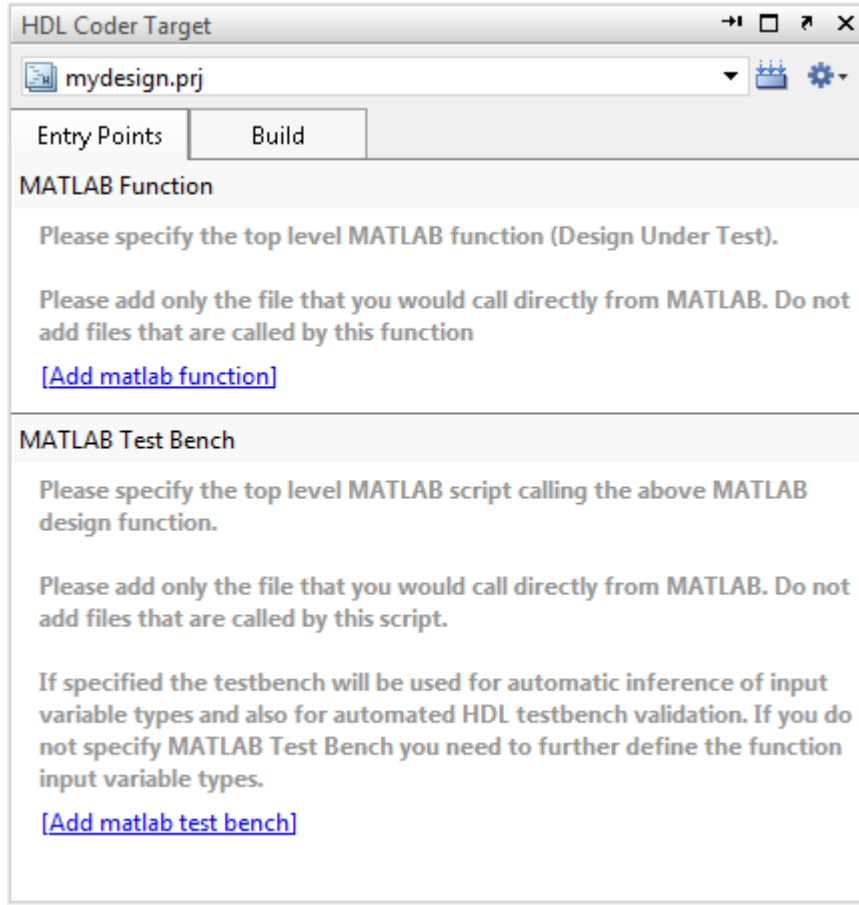


Setting Up an HDL Coder Project

1 Create a new project named `mydesign`:

```
coder -hdlcoder -new mydesign
```

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



2. On the project **Entry Points** tab, under **MATLAB Function**, click **Add MATLAB function**.

3. In the **Add Files** dialog box, select `mlhdlc_sfir.m` and click **Open**.

   HDL Coder adds the file to the project.

4 On the project **Entry Points** tab, under **MATLAB Test Bench**, click **Add MATLAB test bench**.

5 In the **Add Files** dialog box, select `mlhdlc_sfir_tb.m` and click **Open**.

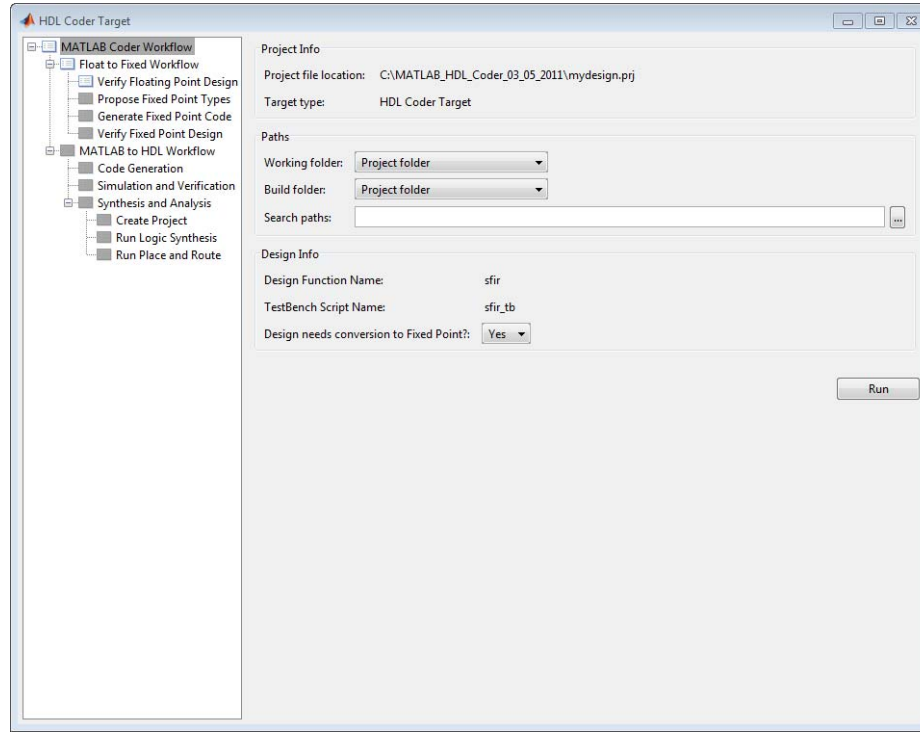
HDL Coder adds the test bench file to the project.

You are now ready to run through the float-to-fixed workflow.

Creating Fixed-Point Versions of the Algorithm and Test Bench

1 Click the project **Build** tab.

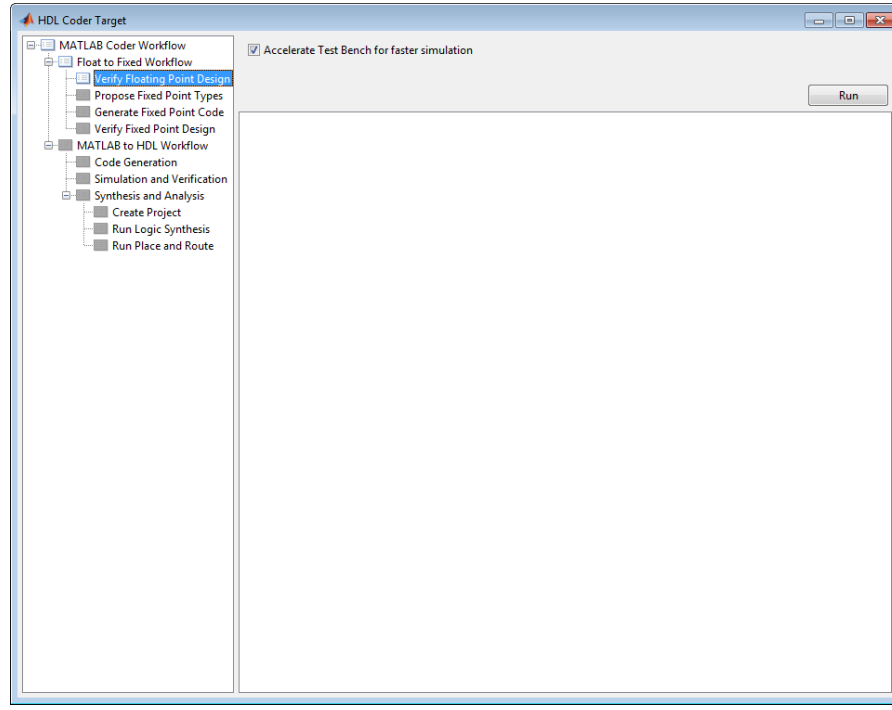
2 On the **Build** tab, click the **Advisor** button to open the HDL Coder Workflow Advisor.



3 In the Workflow Advisor left pane, the **Float to Fixed Workflow** folder is open by default.

In this tutorial, run each task in this folder individually.

a Select the **Verify Floating-Point Design** task and click the **Run** button.

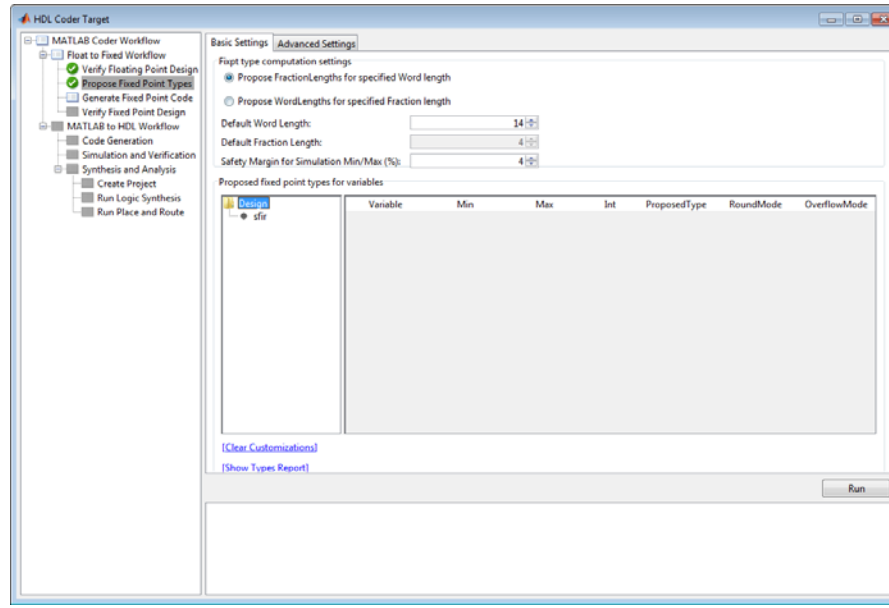


This task verifies that the floating-point algorithm is suitable for code generation by generating a MEX function. It then runs the generated MEX function to check for run-time errors. By default, **Accelerate Test Bench for faster simulation** is selected to accelerate test bench simulation by generating a MEX function for the test bench code. If you do not select this option, the **Verify Fixed-Point Design** task might take a very long time.

The Workflow Advisor generates a build log. If the log contains errors, the Workflow Advisor provides a link to the code generation report.

b Select the **Propose Fixed-Point Types** task.

The advisor displays the **Basic Settings** tab.



By default, HDL Coder proposes fraction lengths for the specified word length that generate no overflows for the specified input range.

When proposing 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 `4%`. 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.

Note HDL Coder uses `hdlfimath` as the default `fimath` value. To change this setting, click the **Advanced Settings** tab and enter the new value in the **fimath** field.

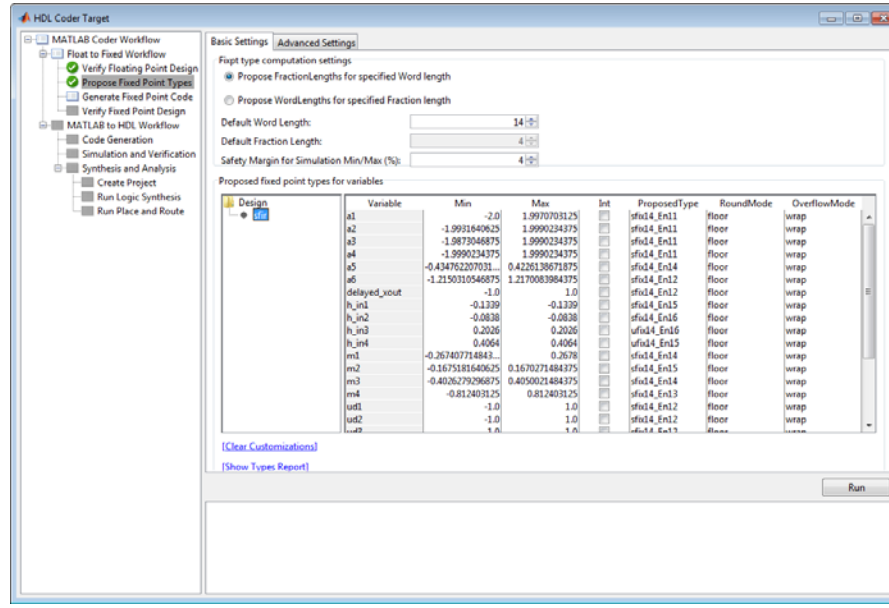
---

c Click the **Run** button.

The task runs and the software proposes fixed-point data types for each variable in the MATLAB code.

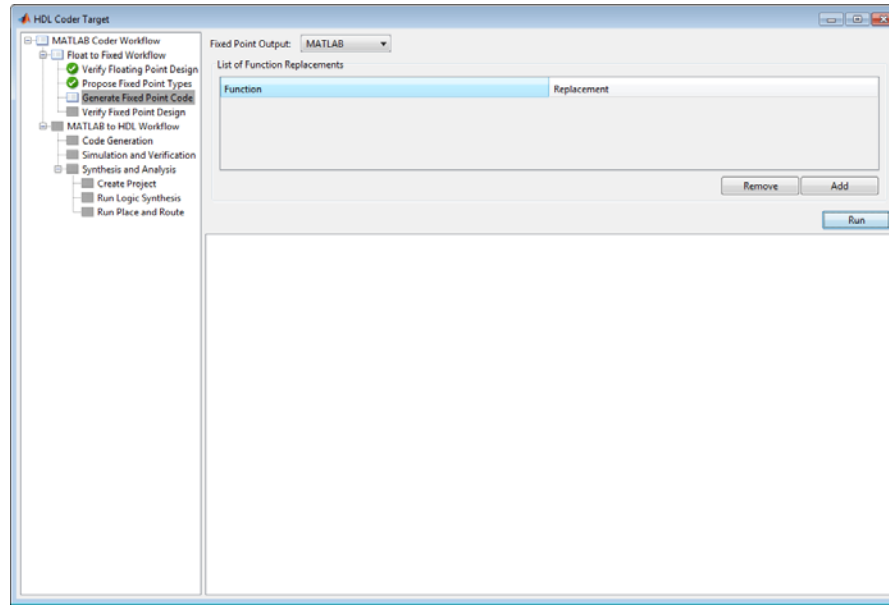
The advisor provides the following information for each variable under **Proposed fixed-point types for variables**, in the **Basic Settings** tab. You can manually modify the information.

Column Heading	Description
Min	Simulation minimum value. If you edit this value, the Workflow Advisor uses the new value as the design minimum for this variable.
Max	Simulation maximum value. If you edit this value, the Workflow Advisor uses the new value as the design maximum for this variable.
Int	The Workflow Advisor checks this box for variables that were pure integers during simulation.
ProposedType	The proposed fixed-point data type for this variable based on simulation data. To override a proposed data type, select the type and enter a new value.
RoundMode	The rounding mode used in the simulation. To propose data types for a different rounding mode, change this setting in the results table. From the drop-down list, selecting a new mode and then rerun the task.
OverflowMode	Whether the **OverflowMode** was `wrap` or `saturate` during simulation. To propose data types for a different overflow mode, change this setting in the results table. From the drop-down list, selecting a new mode and then rerun the task.



d Click **Show Types Report** to view the instrumentation results in a report.

e Select and run **Generate Fixed-Point Code**.



This task generates fixed-point MATLAB code for the `mlhdlc_sfir` function, `mlhdlc_sfir_FixPt`, and for the test bench, `mlhdlc_sfir_tb_FixPt`, using the default fimath, `hdlfimath`, and the data types proposed in the previous task.

Click **mlhdlc_sfir_FixPt** to see the fixed-point MATLAB code for the `mlhdlc_sfir` function.

```
%#codegen
function [y_out,delayed_xout] = mlhdlc_sfir_FixPt(x_in,h_in1,h_in2,h_in3,h_in4)

fm = hdlfimath;
% Symmetric FIR Filter
persistent ud1 ud2 ud3 ud4 ud5 ud6 ud7 ud8
if isempty( ud1 )
    ud1 = fi(0, 1, 14, 12, fm);
    ud2 = fi(0, 1, 14, 12, fm);
    ud3 = fi(0, 1, 14, 12, fm);
    ud4 = fi(0, 1, 14, 12, fm);
    ud5 = fi(0, 1, 14, 12, fm);
    ud6 = fi(0, 1, 14, 12, fm);
    ud7 = fi(0, 1, 14, 12, fm);
    ud8 = fi(0, 1, 14, 12, fm);
end
a1 = fi(ud1 + ud8, 1, 14, 11, fm);
a2 = fi(ud2 + ud7, 1, 14, 11, fm);
a3 = fi(ud3 + ud6, 1, 14, 11, fm);
a4 = fi(ud4 + ud5, 1, 14, 11, fm);
m1 = fi(h_in1*a1, 1, 14, 14, fm);
m2 = fi(h_in2*a2, 1, 14, 15, fm);
m3 = fi(h_in3*a3, 1, 14, 14, fm);
m4 = fi(h_in4*a4, 1, 14, 13, fm);
a5 = fi(m1 + m2, 1, 14, 14, fm);
a6 = fi(m3 + m4, 1, 14, 12, fm);
% filtered output
y_out = fi(a5 + a6, 1, 14, 12, fm);
% delayout input signal
delayed_xout = fi(ud8, 1, 14, 12, fm);
ud8 = fi(ud7, 1, 14, 12, fm);
ud7 = fi(ud6, 1, 14, 12, fm);
ud6 = fi(ud5, 1, 14, 12, fm);
ud5 = fi(ud4, 1, 14, 12, fm);
ud4 = fi(ud3, 1, 14, 12, fm);
ud3 = fi(ud2, 1, 14, 12, fm);
ud2 = fi(ud1, 1, 14, 12, fm);
ud1 = fi(x_in, 1, 14, 12, fm);
end
```

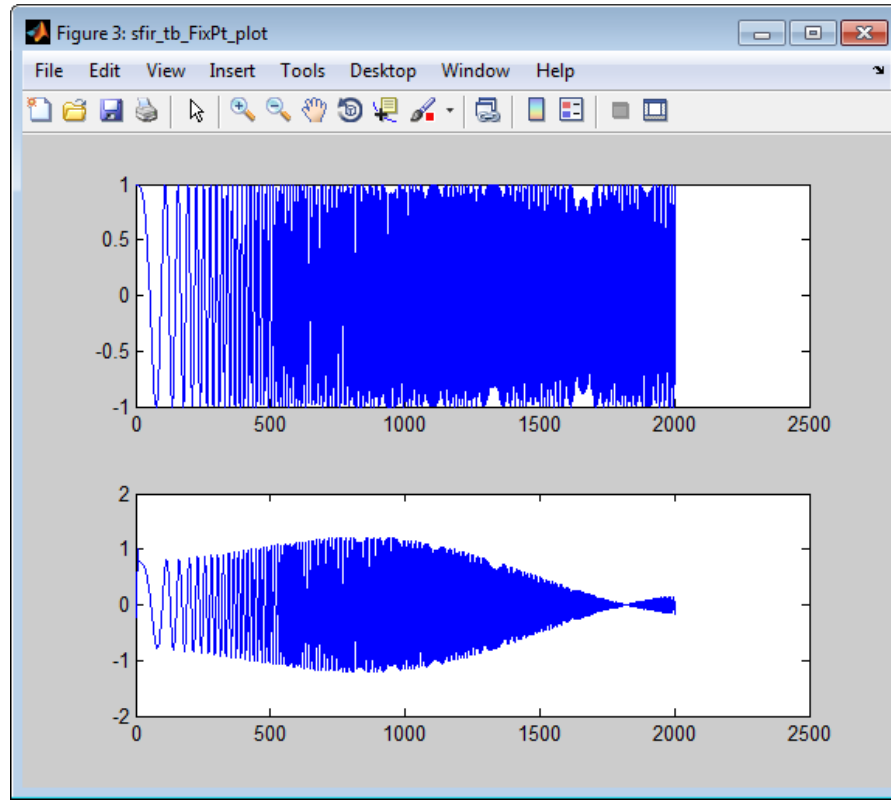
Click **mlhdlc_sfir_tb_FixPt** to see the fixed-point code for the test bench.

```
%clc;
clear all ;
load( 'tb_data.mat' );
x_in = tbdata{ 1 };
h1 = -0.1339;
h2 = -0.0838;
h3 = 0.2026;
h4 = 0.4064;
len = length( x_in );
y_out = zeros( 1, len );
x_out = zeros( 1, len );
for ii = 1:len
    data = x_in( ii );
    % call to the design 'mlhdlc_sfir' that is targeted for hardware
    fm = hdlfimath;
    x_in_in = fi(data, 1, 14, 12, fm);
    h_in1_in = fi(h1, 1, 14, 15, fm);
    h_in2_in = fi(h2, 1, 14, 16, fm);
    h_in3_in = fi(h3, 0, 14, 16, fm);
    h_in4_in = fi(h4, 0, 14, 15, fm);

    [y_out_out,delayed_xout_out] = mlhdlc_sfir_FixPt( x_in_in, h_in1_in,
                                    h_in2_in, h_in3_in, h_in4_in );

    y_out( ii ) = double( y_out_out );
    x_out( ii ) = double( delayed_xout_out );

end
figure( 'Name', [ mfilename, '_plot' ] );
subplot( 2, 1, 1 );
plot( [ 1:len ], x_in );
subplot( 2, 1, 2 );
plot( [ 1:len ], y_out );
```

f Select and run **Verify Fixed-Point Design**.

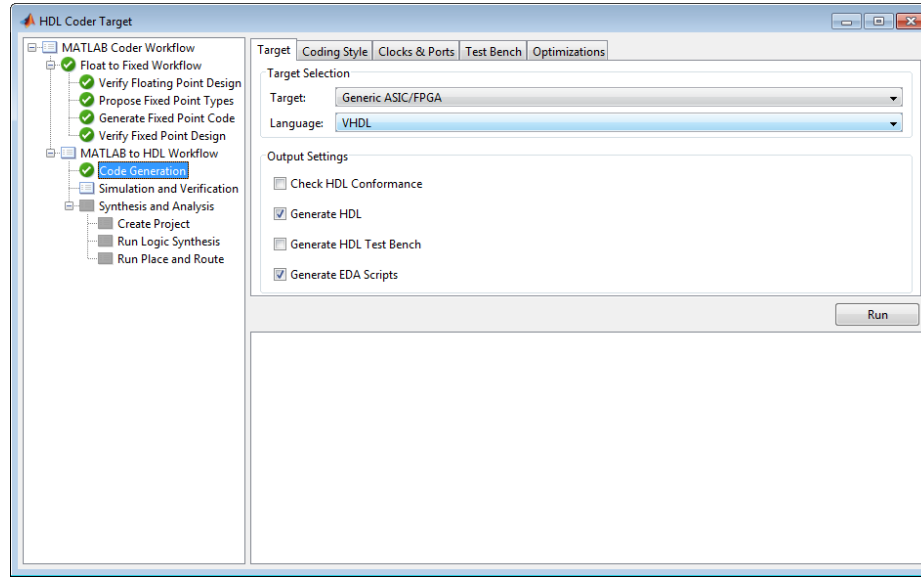
The task runs a fixed-point simulation and plots the results of using the fixed-point version of the filter. Compare this plot to the plot of the floating-point results obtained earlier in the **Verify Floating-Point Design** step to check that the floating-point and fixed-point algorithms are functionally equivalent.

Note This task verifies that the generated fixed-point code is compilable. In the **Verify floating-point design** task, if you did not select **Accelerate Test Bench for faster simulation**, this task might take a very long time.



Generating HDL Code

1 In the Workflow Advisor left pane, select **Code Generation** from the **MATLAB to HDL Workflow** folder.



Tip You can use the **Target**, **Coding Style**, **Clocks and Ports**, **Test Bench**, and **Optimizations** tabs to set code generation options. For more information, see the HDL Coder Workflow Advisor reference.

2 On the **Target** tab, select **Generate HDL test bench**.

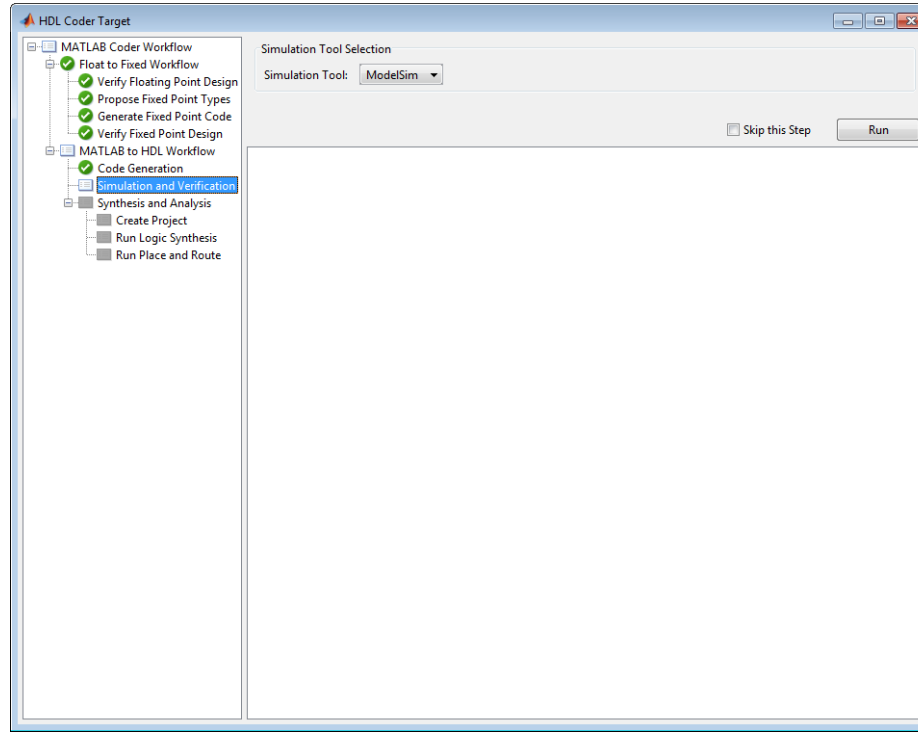
If you want to simulate your design in the next task, you must generate an HDL test bench.

Note **Generate HDL** and **Generate EDA scripts** are already selected by default.

3 Run the task.

HDL Coder generates VHDL code and provides links to this code and to a resource utilization report.

4 Select **Simulation and Verification**.

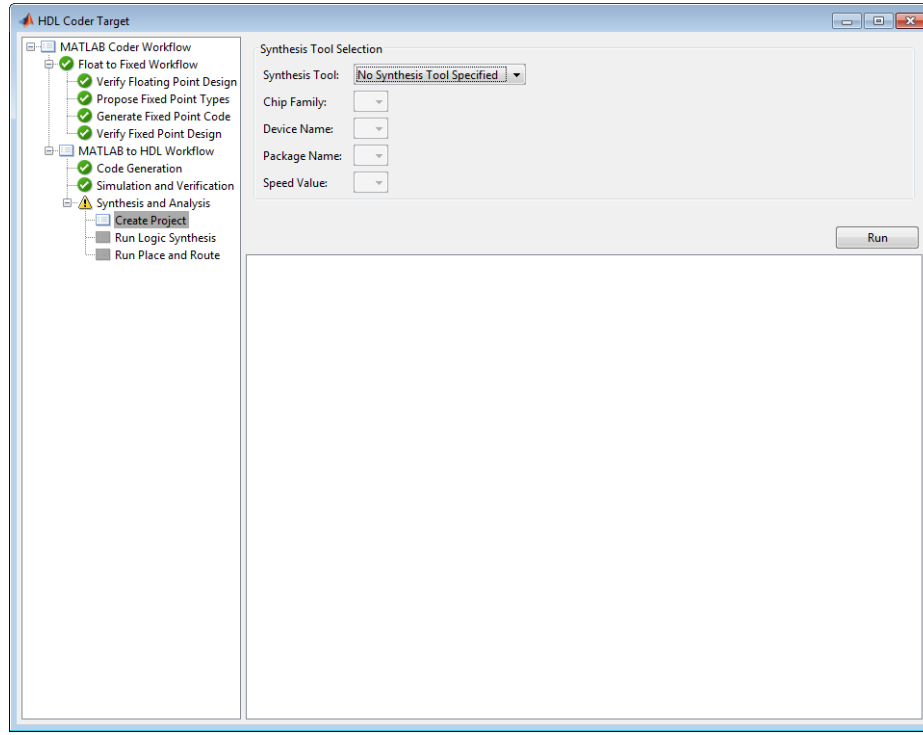


a On the right pane, select the **Simulation tool**.

b Run the task.

The task simulates the fixed-point design using the selected simulation tool and generates a compilation report and a simulation report.

5 Select **Create Project** in the **Synthesis and Analysis** folder.



a Select **Create Project**.

b On the **Synthesis Tool Selection** pane, select a **Synthesis tool** from the list.

c Run the task.

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

6 Select and run **Run Logic Synthesis**.

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.

7 Select and run **Place and Route**.

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.
- Also emits pre- and post-routing timing information for use in critical path analysis and back annotation of your source model.
- Displays results.

HDL Code Generation from a Simulink Model

In this section...

Before You Generate Code

The exercises in this introduction use a preconfigured demo model. The blocks in this demo model support HDL code generation, and the parameters of the model itself have been configured properly for HDL code generation.

After you complete the exercises, you will probably proceed to generating HDL code from your existing models, or newly constructed models. Before you generate HDL code from your own models, you should do the following:

- Use the `hdllib` utility to create a library of blocks that are currently supported for HDL code generation, as described in "Supported Blocks Library". By constructing models with blocks from this library, your models will be HDL compatible.

  The set of supported blocks will change in future releases, so you should rebuild your supported blocks library each time you install a new version of this product.

- Use the **Run Compatibility Checker** option (described in "Selecting and Checking a Subsystem for HDL Compatibility" on page 2-41) to check HDL compatibility of your model or DUT and generate an HDL Code Generation Check Report.

  Alternatively, you can invoke the `checkhdl` function (see checkhdl) to run the compatibility checker.

- Before generating code, use the `hdlsetup` utility (described in "Initializing Model Parameters with hdlsetup" on page 2-27) to set up your model for HDL code generation quickly and consistently.

Overview of Exercises

The coder supports HDL code generation in your choice of environments:

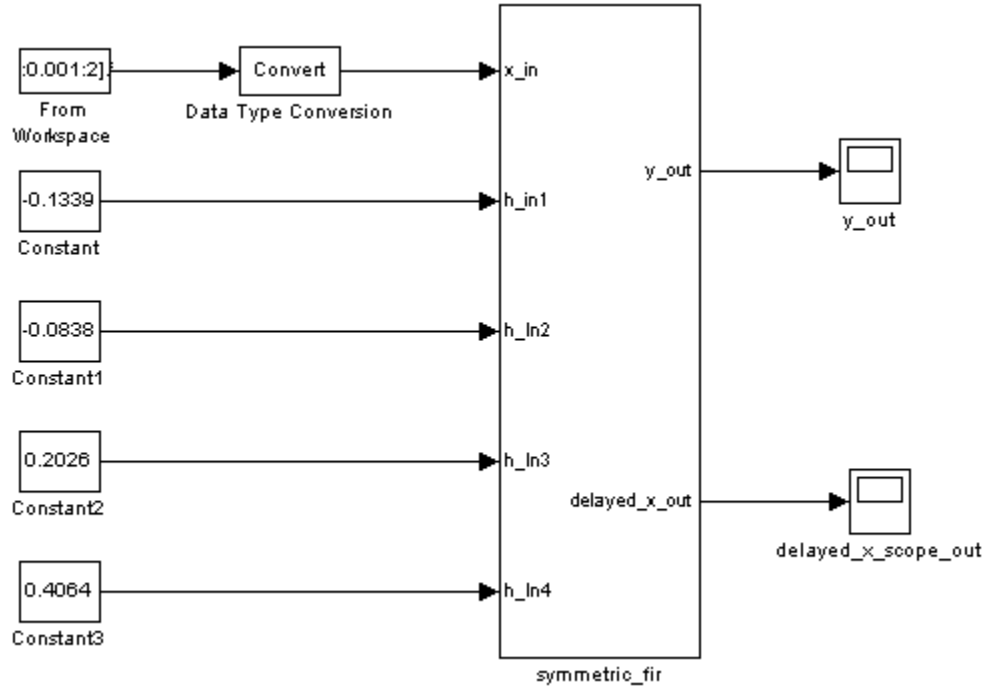
- The MATLAB Command Window supports code generation using the `makehdl`, `makehdltb`, and other functions.
- The Simulink GUI (the Configuration Parameters dialog box and/or Model Explorer) provides an integrated view of the model simulation parameters and HDL code generation parameters and functions.

The hands-on exercises in this chapter introduce you to the mechanics of generating and simulating HDL code, using the same model to generate code in both environments. In a series of steps, you will

- Configure a simple model for code generation.
- Generate VHDL code from a subsystem of the model.
- Generate a VHDL test bench and scripts for the Mentor Graphics® ModelSim® simulator to drive a simulation of the model.
- Compile and execute the model and test bench code in the simulator.
- Generate and simulate Verilog code from the same model.
- Check a model for compatibility with the coder.

The sfir_fixed Model

These exercises use the `sfir_fixed` model as a source for HDL code generation. The model simulates a symmetric finite impulse response (FIR) filter algorithm, implemented with fixed-point arithmetic. The following figure shows the top level of the model.



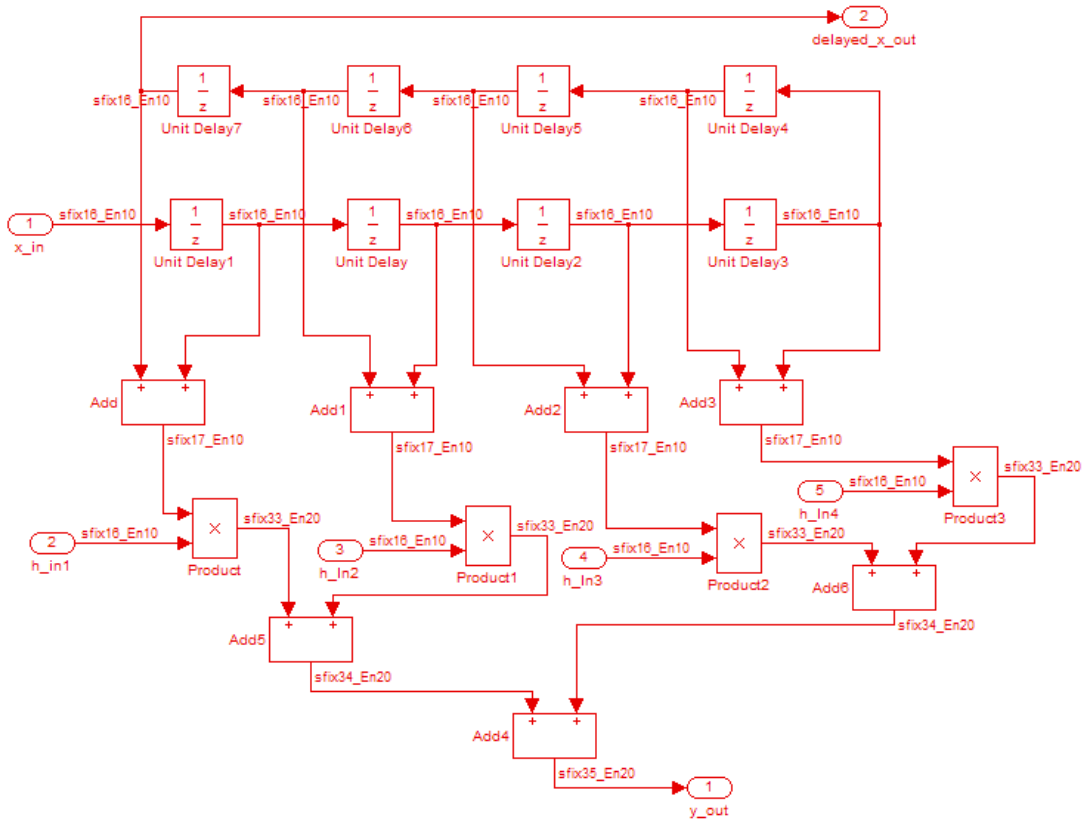
This model uses a division of labor that is helpful in HDL design:

- The `symmetric_fir` subsystem, which implements the filter algorithm, is the device under test (DUT). An HDL entity will be generated, tested, and eventually synthesized from this subsystem.
- The top-level model components that drive the subsystem work as a test bench.

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 (stimulus) signal for the filter. The four Constant blocks provide filter coefficients.

The Scope blocks are used in simulation only. They are virtual blocks, 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 (for example, the Product blocks).

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.

In the exercises that follow, you generate VHDL code that implements the `symmetric_fir` subsystem as an entity. You then generate a test bench from the top-level model. The test bench drives the generated entity, for the required number of clock steps, with stimulus data generated from the Signal From Workspace block.

Generating HDL Code Using the Command Line Interface

Overview

This exercise provides a step-by-step introduction to code and test bench generation commands, their arguments, and the files created by the code generator. The exercise assumes that you have familiarized yourself with the demo model (see "The sfir_fixed Model" on page 2-23).

Creating a Folder and Local Model File

Make a local copy of the demo model and store it in a working folder, as follows.

1. Start the MATLAB software.
2. Create a folder named `sl_hdlcoder_work`, for example:

```
mkdir C:\work\sl_hdlcoder_work
```

The `sl_hdlcoder_work` folder will store a local copy of the demo model and to store folders and code generated by the coder. The location of the folder does not matter, except that it should not be within the MATLAB tree.

3. Make the `sl_hdlcoder_work` folder your working folder, for example:

   ```
   cd C:\work\sl_hdlcoder_work
   ```

4. To open the demo model, type the following command at the MATLAB prompt:

   ```
   demos
   ```

   The **Help** window opens.

5. In the **Contents** tab on the left, select **HDL Coder > Demos > Signal Processing > Symmetric FIR Filter**.

6. In the right pane, click **Open this model**. The `sfir_fixed` model opens.

7. In Simulink, select **File > Save As** and save a local copy of `sfir_fixed.mdl` to your working folder.

8. Leave the `sfir_fixed` model open and proceed to the next section.

Initializing Model Parameters with hdlsetup

Before generating code, you must set some parameters of the model. Rather than doing this manually, use the `hdlsetup` command. The `hdlsetup` command uses the `set_param` function to set up models for HDL code generation quickly and consistently.

To set the model parameters:

1. At the MATLAB command prompt, type

   ```
   hdlsetup('sfir_fixed')
   ```

2. Select **Save** from the **File** menu, to save the model with its new settings.

Before continuing with code generation, consider the settings that `hdlsetup` applies to the model.

`hdlsetup` configures the **Solver** options that are recommended or required by the coder. These are

- **Type**: `Fixed-step`. (The coder currently supports variable-step solvers under limited conditions. See hdlsetup.)
- **Solver**: `Discrete (no continuous states)`. Other fixed-step solvers could be selected, but this option is usually the best one for simulating discrete systems.
- **Tasking mode**: `SingleTasking`. The coder does not currently support models that execute in multitasking mode.

  Do not set **Tasking mode** to `Auto`.

`hdlsetup` also configures the model start and stop times and fixed-step size as follows:

- **Start Time**: `0.0` s
- **Stop Time**: `10` s
- **Fixed step size (fundamental periodic sample time)** : `auto`

If **Fixed step size** is set to `auto` the step size is chosen automatically, based on the sample times specified in the model. In the demo model, only the Signal From Workspace block specifies an explicit sample time (1 s); the other blocks inherit this sample time.

The model start and stop times determine the total simulation time. This in turn determines the size of data arrays that are generated to provide stimulus and output data for generated test benches. For the demo model, computation of 10 seconds of test data does not take a significant amount of time. Computation of sample values for more complex models can be time consuming. In such cases, you may want to decrease the total simulation time.

The remaining parameters set by `hdlsetup` control error severity levels, data logging, and model display options. If you want to view the complete set of

model parameters affected by `hdlsetup`, open `hdlsetup.m` in the MATLAB Editor.

The model parameter settings provided by are intended as useful defaults, but they may not be optimal for your application. For example, `hdlsetup` sets a default **Simulation stop time** of 10 s. A total simulation time of 1000 s would be more realistic for a test of the `sfir_fixed` demo model. If you would like to change the simulation time, enter the desired value into the **Simulation stop time** field of the Simulink window.

See the "Model Parameters" table in the "Model and Block Parameters" section of the Simulink documentation for a summary of user-settable model parameters.

Generating a VHDL Entity from a Subsystem

In this section, you will use the `makehdl` function to generate code for a VHDL entity from the `symmetric_fir` subsystem of the demo model. `makehdl` also generates script files for third-party HDL simulation and synthesis tools.

`makehdl` lets you specify numerous properties that control various features of the generated code. In this example, you will use the `makehdl` property defaults.

Before generating code, make sure that you have completed the steps described in "Creating a Folder and Local Model File" on page 2-26 and "Initializing Model Parameters with hdlsetup" on page 2-27.

To generate code:

1. Select **Current Folder** from the **Desktop** menu in the MATLAB window. This displays the MATLAB Current Folder browser, which lets you easily access your working folder and the files that will be generated within it.

2. At the MATLAB prompt, type the command

   ```
   makehdl('sfir_fixed/symmetric_fir')
   ```

   This command directs the coder to generate code from the `symmetric_fir` subsystem within the `sfir_fixed` model, using default property values.

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

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

Observe that the names of generated files in the progress messages are hyperlinked. After code generation completes, you can click these hyperlinks to view the files in the MATLAB Editor.

`makehdl` compiles the model before generating code. Depending on model display options (such as port data types, etc.), the appearance of the model may change after code generation.

4 By default, `makehdl` generates VHDL code. Code files and scripts are written to a *target folder*. The default target folder is a subfolder of your working folder, named `hdlsrc`.

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.

5 The files that `makehdl` has generated in the `hdlsrc` folder are

- `symmetric_fir.vhd`: VHDL code. This file contains an entity definition and RTL architecture implementing the `symmetric_fir` filter.
- `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 "Code Tracing Using the Mapping File").

6 To view the generated VHDL code in the MATLAB Editor, double-click the `symmetric_fir.vhd` file icon in the Current Folder browser.

At this point it is suggested that you study the `ENTITY` and `ARCHITECTURE` definitions while referring to "HDL Code Generation Defaults" in the makehdl reference documentation. The reference documentation describes the default naming conventions and correspondences between the elements

of a model (subsystems, ports, signals, etc.) and elements of generated HDL code.

7. Before proceeding to the next section, close files you have opened in the editor. Then, click the Go Up One Level button in the Current Folder browser, to set the current folder back to your `sl_hdlcoder_work` folder.

8. Leave the `sfir_fixed` model open and proceed to the next section.

Generating VHDL Test Bench Code

In this section, you use the test bench generation function, `makehdltb`, to generate a VHDL test bench. The test bench is designed to drive and verify the operation of the `symmetric_fir` entity that was generated in the previous section. A generated 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.

In addition, `makehdltb` generates Mentor Graphics ModelSim scripts to compile and execute the test bench.

This exercise assumes that your working folder is the same as that used in the previous section. This folder now contains an `hdlsrc` folder containing the previously generated code.

To generate a test bench:

1. At the MATLAB prompt, type the command

```
makehdltb('sfir_fixed/symmetric_fir')
```

This command generates a test bench that is designed to interface to and validate code generated from `symmetric_fir` (or from a subsystem with a functionally identical interface). By default, VHDL test bench code, as well as scripts, are generated in the `hdlsrc` target folder.

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

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

3 To view generated test bench and script files, double-click the `hdlsrc` folder icon in the Current Folder browser. Alternatively, you can click the hyperlinked names of generated files in the code test bench generation progress messages.

The files generated by `makehdltb` are:

- `symmetric_fir_tb.vhd`: VHDL test bench code and generated test and output data.
- `symmetric_fir_tb_compile.do`: Mentor Graphics ModelSim compilation script (`vcom` commands). This script compiles and loads both 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.

4 If you want to view the generated test bench code in the MATLAB Editor, double-click the `symmetric_fir.vhd` file icon in the Current Folder browser. You may want to study the code while referring to the makehdltb reference documentation, which describes the default actions of the test bench generator.

5 Before proceeding to the next section, close files you have opened in the editor. Then, click the Go Up One Level button in the Current Folder browser, to set the current folder back to your `sl_hdlcoder_work` folder.

Verifying Generated Code

You can now take the previously generated code and test bench to an HDL simulator for simulated execution and verification of results. See "Simulating and Verifying Generated HDL Code" on page 2-47 for an example of how to use generated test bench and script files with the Mentor Graphics ModelSim simulator.

Generating a Verilog Module and Test Bench

The procedures for generating Verilog code differ only slightly from those for generating VHDL code. This section provides an overview of the command syntax and the generated files.

Generating a Verilog Module. By default, `makehdl` generates VHDL code. To override the default and generate Verilog code, you must pass in a property/value pair to `makehdl`, setting the `TargetLanguage` property to `'verilog'`, as in this example.

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

The previous command generates Verilog source code, as well as scripts for the simulation and the synthesis tools, in the default target folder, `hdlsrc`.

The files generated by this example command are:

- `symmetric_fir.v`: Verilog code. This file contains a Verilog module implementing the `symmetric_fir` subsystem.
- `symmetric_fir_compile.do`: Mentor Graphics ModelSim compilation script (`vlog` command) to compile the generated Verilog 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 "Code Tracing Using the Mapping File").

Generating and Executing a Verilog Test Bench. The `makehdltb` syntax for overriding the target language is exactly the same as that for `makehdl`. The following example generates Verilog test bench code to drive the Verilog module, `symmetric_fir`, in the default target folder.

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

The files generated by this example command are

- `symmetric_fir_tb.v`: Verilog test bench code and generated test and output data.
- `symmetric_fir_tb_compile.do`: Mentor Graphics ModelSim compilation script (`vlog` commands). This script compiles and loads both the entity to be tested (`symmetric_fir.v`) and the test bench code (`symmetric_fir_tb.v`).
- `symmetric_fir_tb_sim.do`: Mentor Graphics ModelSim script to initialize the simulator, set up **wave** window signal displays, and run a simulation.

The following listing shows the commands and responses from a test bench session using the generated scripts:

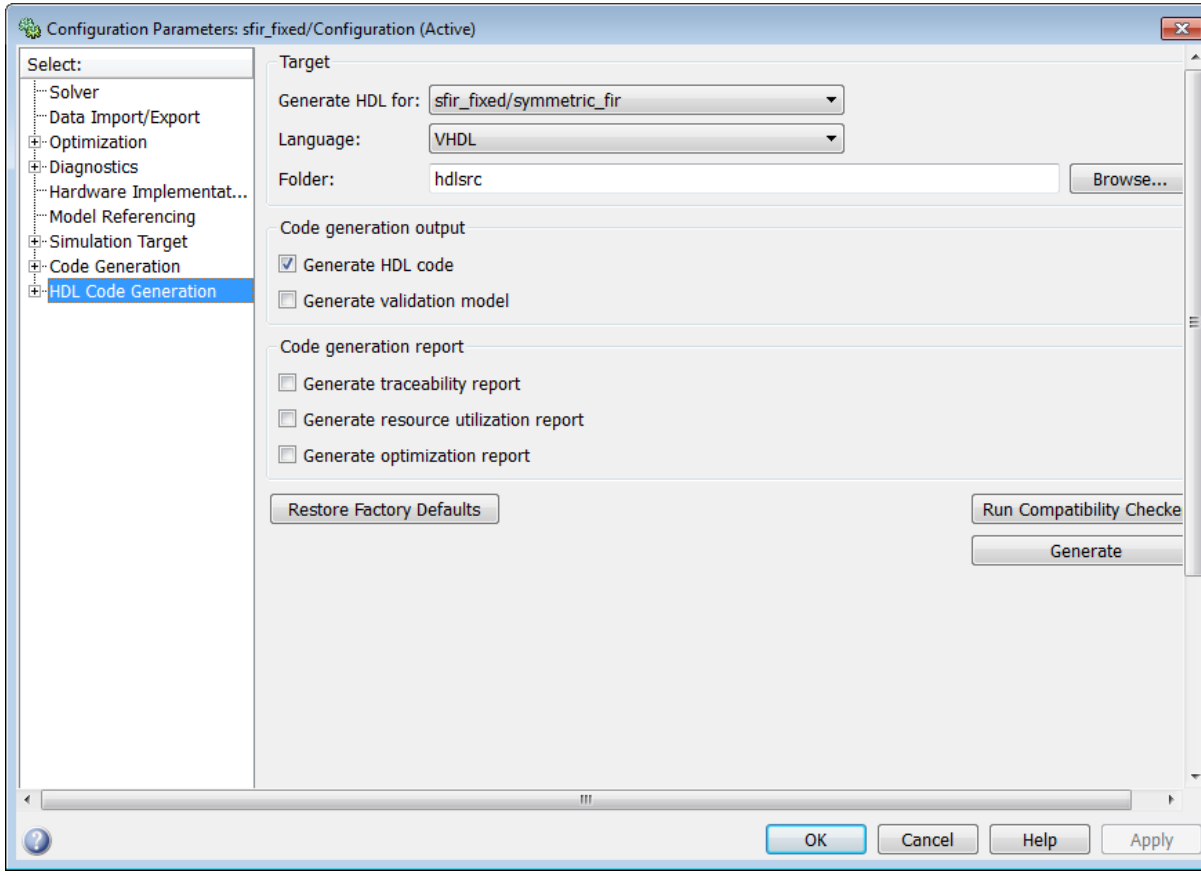
```
ModelSim> do symmetric_fir_tb_compile.do
# Model Technology ModelSim SE vlog 6.0 Compiler 2004.08 Aug 19 2004
# -- Compiling module symmetric_fir
#
# Top level modules:
#  symmetric_fir
# Model Technology ModelSim SE vlog 6.0 Compiler 2004.08 Aug 19 2004
# -- Compiling module symmetric_fir_tb
#
#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

Generating HDL Code Using the GUI

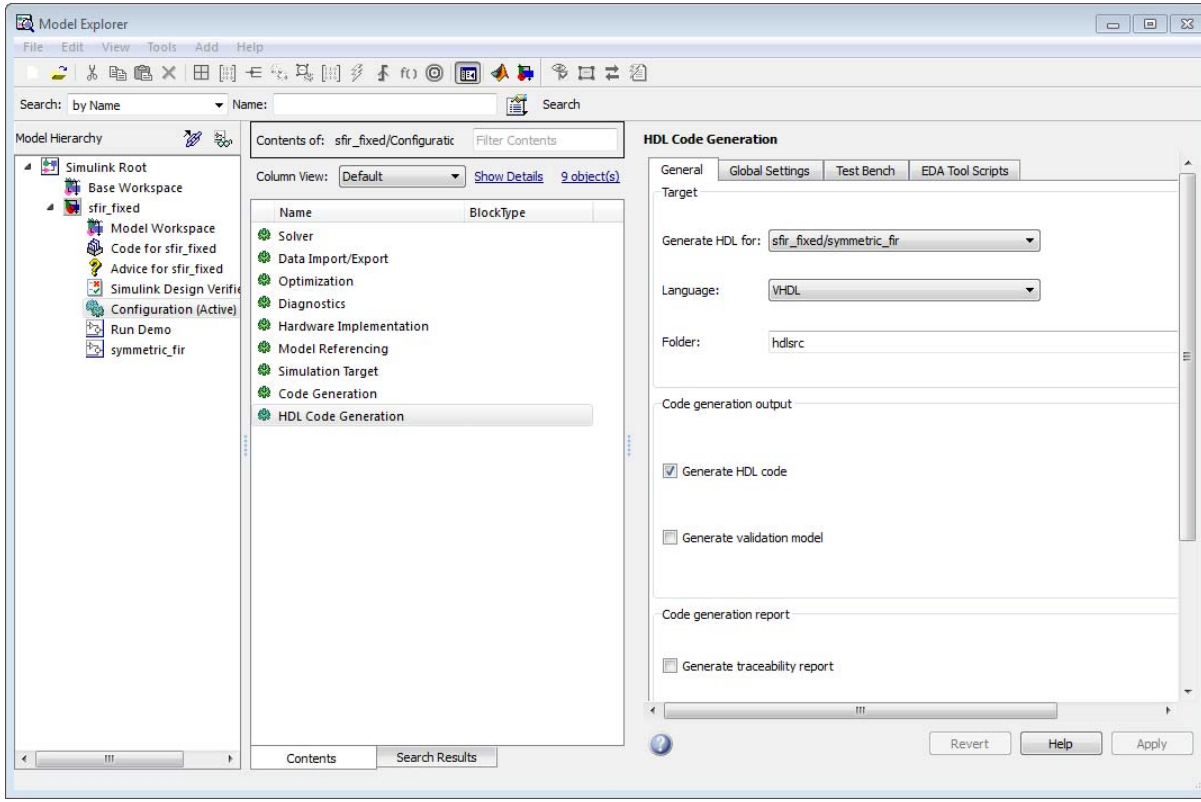
HDL Coder GUI Overview

You can view and edit options and parameters that affect HDL code generation in the Configuration Parameters dialog box, or in the Model Explorer.

The following figure shows the top-level **HDL Code Generation** pane in the Configuration Parameters dialog box.



The following figure shows the top-level **HDL Code Generation** options pane in the Model Explorer.



If you are not familiar with Simulink configuration sets, or how to view and edit them in the Configuration Parameters dialog box, see the following documentation:

- “Manage a Configuration Set”
- “Configuration Parameters Dialog Box”

If you are not familiar with the Model Explorer, see “Exploring, Searching, and Browsing Models”.

In the hands-on code generation exercises that follow, you use the Configuration Parameters dialog box to view and set the coder options and

controls. The exercises use the `sfir_fixed` model (see "The sfir_fixed Model" on page 2-23) in basic code generation and verification steps.

Creating a Folder and Local Model File

In this section you will setup the folder and a local copy of the demo model.

Creating a Folder. Start by setting up a working folder:

1. Start MATLAB.
2. Create a folder named `sl_hdlcoder_work`, for example:

   ```
   mkdir C:\work\sl_hdlcoder_work
   ```

   You will use `sl_hdlcoder_work` to store a local copy of the demo model and to store folders and code generated by the coder. The location of the folder does not matter, except that it should not be within the MATLAB folder tree.
3. Make the `sl_hdlcoder_work` folder your working folder, for example:

   ```
   cd C:\work\sl_hdlcoder_work
   ```

Making a Local Copy of the Model File. Next, make a copy of the `sfir_fixed` model:

1. To open the model, type the following command at the MATLAB prompt:

   ```
   sfir_fixed
   ```
2. Save a local copy of `sfir_fixed.mdl` to your working folder.
3. Leave the `sfir_fixed` model open and proceed to the next section.

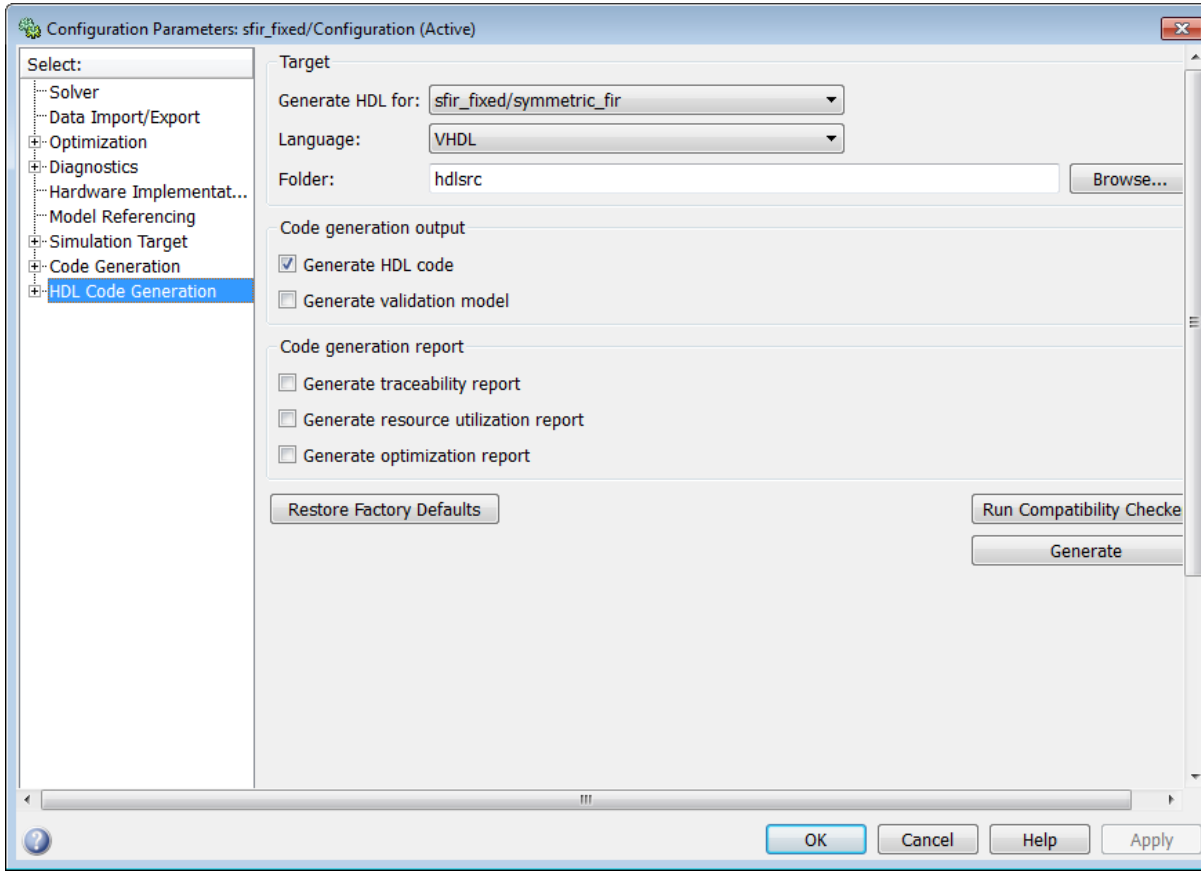
Viewing Coder Options in the Configuration Parameters Dialog Box

The coder option settings are displayed as a category of the model's active configuration set. You can view and edit these options in the Configuration

Parameters dialog box, or in the Model Explorer. This discussion uses the Configuration Parameters dialog box.

To access the coder settings:

1. Open the Configuration Parameters dialog box.
2. Select the **HDL Code Generation** pane.



The **HDL Code Generation** pane contains top-level options and buttons that control the HDL code generation process. Several other categories of options are available under the **HDL Code Generation** entry in the

Select tree. This exercise uses a small subset of these options, leaving the others at their default settings.

"Code Generation Options in the HDL Coder Dialog Boxes" summarizes the options available in the **HDL Code Generation** category.

Initializing Model Parameters with hdlsetup

Before generating code, you must set some parameters of the model. Rather than doing this manually, use the `hdlsetup` command. The `hdlsetup` command uses the `set_param` function to set up models for HDL code generation quickly and consistently.

To set the model parameters:

1. At the MATLAB command prompt, type:

   ```
   hdlsetup('sfir_fixed')
   ```

2. Save the model with its new settings.

Before continuing with code generation, consider the settings that `hdlsetup` applies to the model.

`hdlsetup` configures **Solver** options that are recommended or required by the coder. These options are:

- **Type**: `Fixed-step`. (The coder currently supports variable-step solvers under limited conditions. See hdlsetup.)
- **Solver**: `Discrete (no continuous states)`. Other fixed-step solvers could be selected, but this option is usually the best one for simulating discrete systems.
- **Tasking mode**: `SingleTasking`. The coder does not currently support models that execute in multitasking mode.

  Do not set **Tasking mode** to `Auto`.

`hdlsetup` also configures the model start and stop times and fixed-step size as follows:

- **Start Time**: `0.0` s
- **Stop Time**: `10` s
- **Fixed step size (fundamental periodic sample time)**: `auto`

If **Fixed step size** is set to `auto` the step size is chosen automatically, based on the sample times specified in the model. In the demo model, only the Signal From Workspace block specifies an explicit sample time (1 s); the other blocks inherit this sample time.

The model start and stop times determine the total simulation time. This in turn determines the size of data arrays that are generated to provide stimulus and output data for generated test benches. For the demo model, computation of 10 seconds of test data does not take a significant amount of time. Computation of sample values for more complex models can be time consuming. In such cases, you may want to decrease the total simulation time.

The remaining parameters set by `hdlsetup` control error severity levels, data logging, and model display options. If you want to view the complete set of model parameters affected by `hdlsetup`, open `hdlsetup.m` in the MATLAB Editor.

The model parameter settings provided by `hdlsetup` are intended as useful defaults, but they may not be optimal for your application. For example, `hdlsetup` sets a default **Simulation stop time** of 10 s. A total simulation time of 1000 s would be more realistic for a test of the `sfir_fixed` demo model. If you would like to change the simulation time, enter the desired value into the **Simulation stop time** field of the Simulink Editor.

See the "Model Parameters" table in the "Model and Block Parameters" section of the Simulink documentation for a summary of user-settable model parameters.

Selecting and Checking a Subsystem for HDL Compatibility

The coder generates code from either the current model or from a subsystem at the root level of the current model. You use the **Generate HDL for** menu to select the model or subsystem from which code is to be generated. Each entry in the menu shows the full path to the model or one of its subcomponents.

The `sfir_fixed` model is configured with the `sfir_fixed/symmetric_fir` subsystem selected for code generation. If this is not the case, make sure that the `symmetric_fir` subsystem is selected for code generation, as follows:

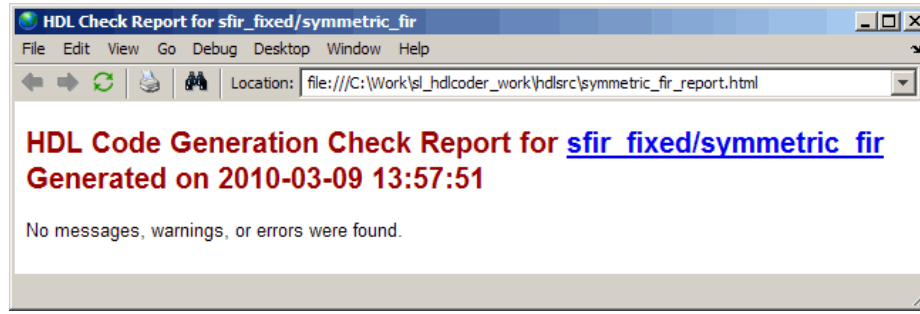
1. Select `sfir_fixed/symmetric_fir` from the **Generate HDL for** menu.
2. Click **Apply**.

To check HDL compatibility for the subsystem:

1. Click the **Run Compatibility Checker** button.
2. The HDL compatibility checker examines the system selected in the **Generate HDL for** menu for compatibility problems. In this case, the selected subsystem is fully HDL-compatible, and the compatibility checker displays the following message:

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

3. The compatibility checker also displays an HTML report in a Web browser, as shown in the following figure.

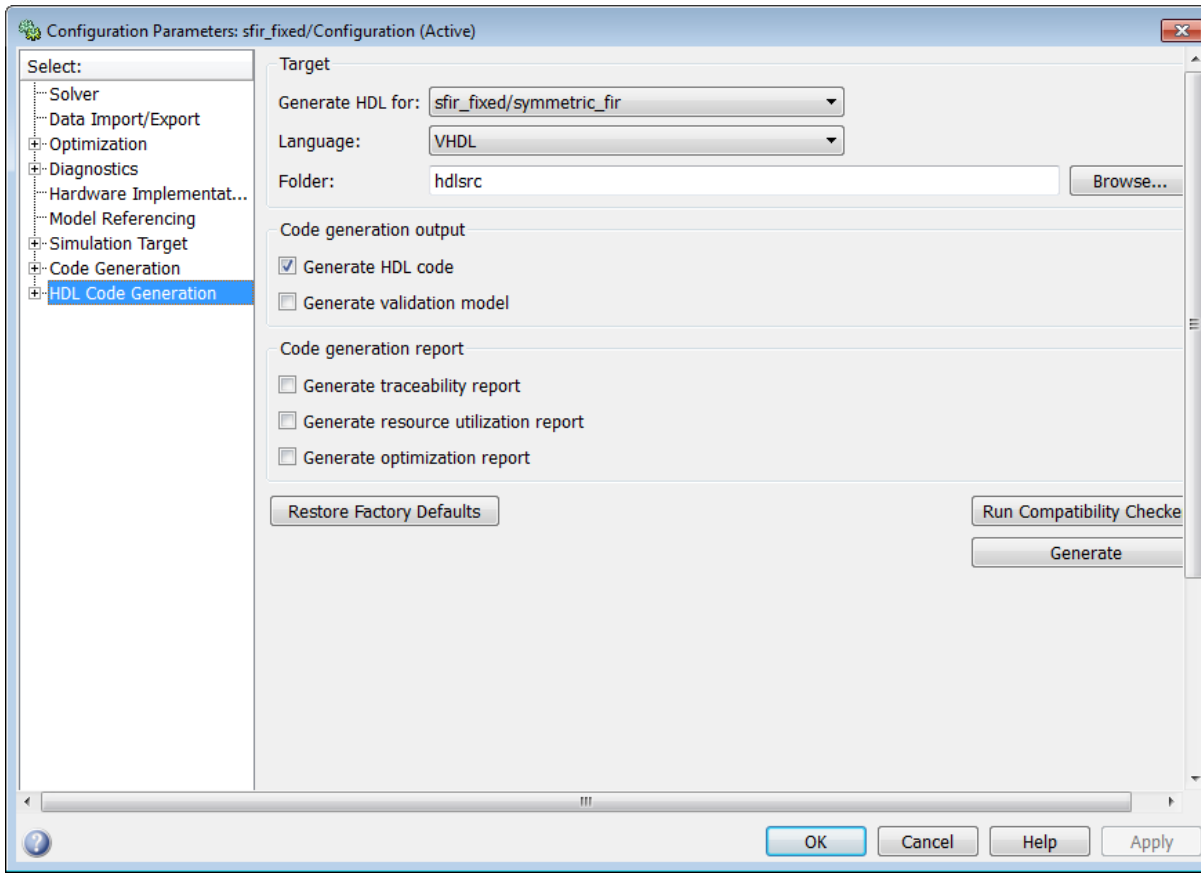


Generating VHDL Code

The top-level **HDL Code Generation** options are now set as follows:

- The **Generate HDL for** field specifies the `sfir_fixed/symmetric_fir` subsystem for code generation.

- The **Language** field specifies (by default) generation of VHDL code.
- The **Folder** field specifies a *target folder* that stores generated code files and scripts. The default target folder is a subfolder of your working folder, named `hdlsrc`.



Before generating code, select **Current Folder** from the **Desktop** menu in the MATLAB window. This displays the Current Folder browser, which lets you access your working folder and the files that will be generated within it.

To generate code:

1 Click the **Generate** button.

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

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

Observe that the names of generated files in the progress messages are hyperlinked. After code generation completes, you can click these hyperlinks to view the files in the MATLAB Editor.

The coder compiles the model before generating code. Depending on model display options (such as port data types, etc.), the appearance of the model may change after code generation.

3 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.

4 The files that were generated in the `hdlsrc` folder are:

- `symmetric_fir.vhd`: VHDL code. This file contains an entity definition and RTL architecture implementing the `symmetric_fir` filter.
- `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 "Code Tracing Using the Mapping File").

5 To view the generated VHDL code in the MATLAB Editor, double-click the `symmetric_fir.vhd` file icon in the Current Folder browser.

At this point it is suggested that you study the `ENTITY` and `ARCHITECTURE` definitions while referring to "HDL Code Generation Defaults" in the makehdl reference documentation. The reference documentation describes the default naming conventions and correspondences between the elements of a model (subsystems, ports, signals, etc.) and elements of generated HDL code.

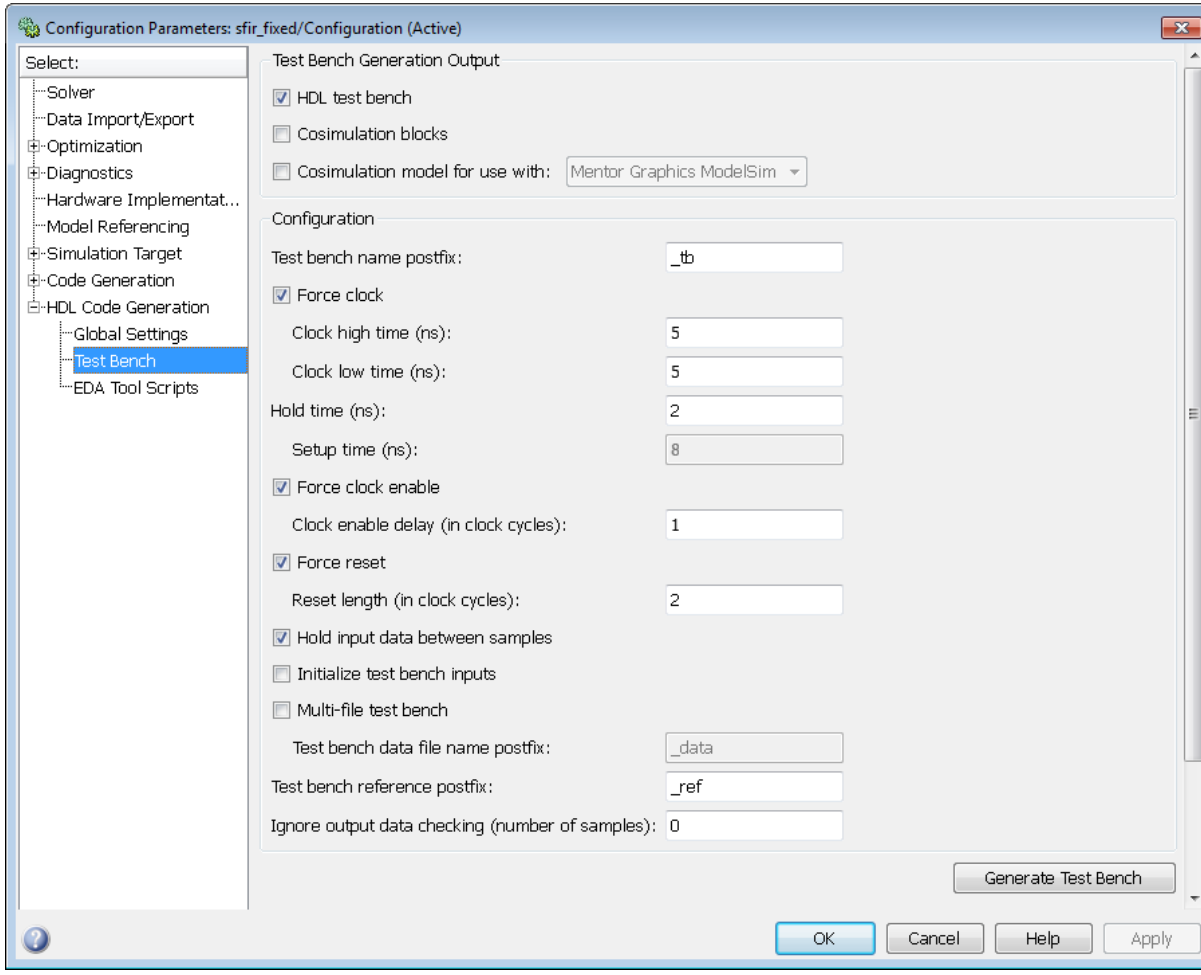
6 Before proceeding to the next section, close files you have opened in the editor. Then, click the Go Up One Level button in the Current Folder browser, to set the current folder back to your `sl_hdlcoder_work` folder.

Generating VHDL Test Bench Code

At this point, the **Generate HDL for**, **Language**, and **Folder** fields are set as they were in the previous section. Accordingly, you can now generate VHDL test bench code to drive the VHDL code generated previously for the `sfir_fixed/symmetric_fir` subsystem. The code will be written to the same target folder as before.

To generate a VHDL test bench:

1 Select the **HDL Code Generation > Test Bench** pane.



2 Select **HDL test bench**.

3 Click the **Generate Test Bench** button.

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

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

5 The generated files in the `hdlsrc` folder are:

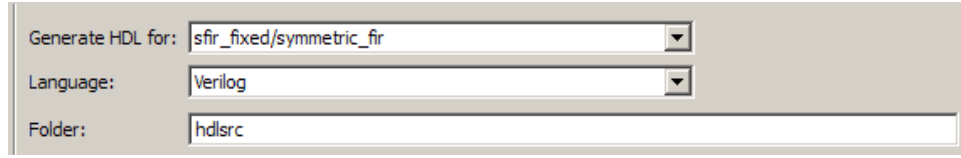
- `symmetric_fir_tb.vhd`: VHDL test bench code, with generated test and output data.
- `symmetric_fir_tb_compile.do`: Mentor Graphics ModelSim compilation script (`vcom` commands). This script compiles and loads the entity to be tested (`symmetric_fir.vhd`) and the test bench code (`symmetric_fir_tb.vhd`).
- `symmetric_fir_tb_sim.do`: Mentor Graphics ModelSim script to initialize the simulator, set up **wave** window signal displays, and run a simulation.

Verifying Generated Code

You can now take the generated code and test bench to an HDL simulator for simulated execution and verification of results. See "Simulating and Verifying Generated HDL Code" on page 2-47 for an example of how to use generated test bench and script files with the Mentor Graphics ModelSim simulator.

Generating Verilog Model and Test Bench Code

The procedure for generating Verilog code is the same as for generating VHDL code (see "Generating a VHDL Entity from a Subsystem" on page 2-29 and "Generating VHDL Test Bench Code" on page 2-31), except that you select `Verilog` from the **Language** field of the **HDL Code Generation** options.



Simulating and Verifying Generated HDL Code

Note This section requires the use of the Mentor Graphics ModelSim simulator.

This section assumes that you have generated code from the `sfir_fixed` model as described in either of the following exercises:

- "Generating HDL Code Using the Command Line Interface" on page 2-26
- "Generating HDL Code Using the GUI" on page 2-35

In this section you compile and run a simulation of the previous generated model and test bench code. The scripts generated by the coder let you do this with just a few simple commands. The procedure is the same, whether you generated code in the command line environment or in the GUI.

To run the simulation:

1. Start the Mentor Graphics ModelSim software.
2. Set the working folder to the folder in which you previously generated code.

   ```
   ModelSim>cd C:/work/sl_hdlcoder_work/hdlsrc
   ```

3. Use the generated compilation script to compile and load the generated model and text bench code. The following listing shows the command and responses.

   ```
   ModelSim>do symmetric_fir_tb_compile.do
   # Model Technology ModelSim SE vcom 6.0 Compiler 2004.08 Aug 19 2004
   # -- Loading package standard
   # -- Loading package std_logic_1164
   # -- Loading package numeric_std
   # -- Compiling entity symmetric_fir
   # -- Compiling architecture rtl of symmetric_fir
   # Model Technology ModelSim SE vcom 6.0 Compiler 2004.08 Aug 19 2004
   # -- Loading package standard
   # -- Loading package std_logic_1164
   # -- Loading package numeric_std
   # -- Compiling package symmetric_fir_tb_pkg
   # -- Compiling package body symmetric_fir_tb_pkg
   # -- Loading package symmetric_fir_tb_pkg
   # -- Loading package symmetric_fir_tb_pkg
   # -- Compiling entity symmetric_fir_tb
   # -- Compiling architecture rtl of symmetric_fir_tb
   ```

```
# -- Loading entity symmetric_fir
```

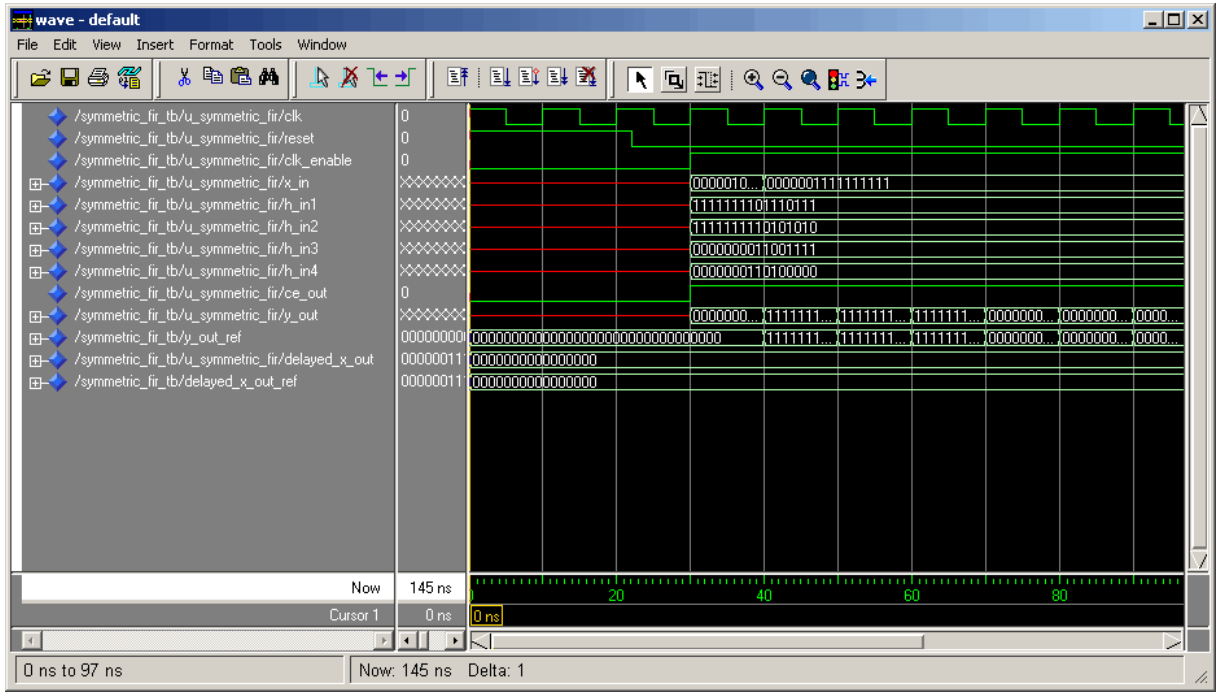
4 Use the generated simulation script to execute the simulation. The following listing shows the command and responses. The warning messages are benign.

```
ModelSim>do  symmetric_fir_tb_sim.do
# vsim work.symmetric_fir_tb
# Loading C:\Applications\ModelTech_6_0\win32/../std.standard
# Loading C:\Applications\ModelTech_6_0\win32/../ieee.std_logic_1164(body)
# Loading C:\Applications\ModelTech_6_0\win32/../ieee.numeric_std(body)
# Loading work.symmetric_fir_tb_pkg(body)
# Loading work.symmetric_fir_tb(rtl)
# Loading work.symmetric_fir(rtl)
# ** Warning: NUMERIC_STD."<": metavalue detected, returning FALSE
#    Time: 0 ns  Iteration: 0  Instance: /symmetric_fir_tb
.
.
.
# ** Warning: NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
#    Time: 0 ns  Iteration: 1  Instance: /symmetric_fir_tb
# ** Note: **************TEST COMPLETED **************
#    Time: 140 ns  Iteration: 1  Instance: /symmetric_fir_tb
```

The test bench termination message indicates that the simulation has run to completion without comparison errors.

```
# ** Note: **************TEST COMPLETED **************
```

5 The simulation script displays inputs and outputs in the model (including the reference signals `y_out_ref` and `delayed_x_out_ref`) in the Mentor Graphics ModelSim **wave** window. The following figure shows the signals displayed in the **wave** window.



6 Exit the Mentor Graphics ModelSim simulator when you finish viewing signals.

7 Close files you have opened in the MATLAB Editor. Then, click the **Go Up One Level** button in the Current Folder browser, to set the current folder back to your work folder.

A

Examples

Use this list to find examples in the documentation.

Generating HDL Code Using the Command Line Interface

Generating HDL Code Using the GUI

Verifying Generated HDL Code in an HDL Simulator

"Simulating and Verifying Generated HDL Code" on page 2-47

Index

X