



# Model-Based Design with Simulink, HDL Coder, and Xilinx System Generator for DSP

By Kiran Kintali and Yongfeng Gu

---

## Contents

<b>Introduction</b> .....	<b>3</b>
<b>Required Software</b> .....	<b>3</b>
<b>Setting Up the MATLAB Environment for HDL Code Generation and Xilinx System Generator Integration</b> .....	<b>4</b>
<b>Example 1: Design with Blocks from Both Simulink and Xilinx System Generator for DSP</b> .....	<b>5</b>
Preparing a Model for HDL Code Generation.....	6
Creating a Xilinx System Generator Subsystem.....	6
Matching Port Data Types.....	7
Setting Code Generation Options for Xilinx System Generator.....	8
Setting Code Generation Options for HDL Coder.....	9
Generating HDL.....	10
Generating HDL Test Bench and Simulation Scripts.....	10
<b>Example 2: Generate a Xilinx System Generator for DSP Black Box Block from a MATLAB HDL Design</b> .....	<b>11</b>
Creating a New Folder and Copying Relevant Files.....	11
Simulating the Design.....	12
Creating a New Project from the Command Line.....	12
Generating a Xilinx System Generator for DSP Black Box.....	12
Running Fixed-Point Conversion and Generating Code.....	13
Examining the Generated Model and Configuration File.....	13
Deleting the Temporary Files.....	15
<b>Further Reading</b> .....	<b>15</b>

## Introduction

MATLAB® and Simulink® for Model-Based Design provide signal, image, and video processing engineers with a development platform that spans design, modeling, simulation, code generation, and implementation. Engineers who use Model-Based Design to target FPGAs or ASICs can design and simulate systems with MATLAB, Simulink, and Stateflow® and then generate bit-true, cycle-accurate, synthesizable Verilog® and VHDL® code using HDL Coder™.

Alternatively, engineers who specifically target Xilinx® FPGAs can use a Xilinx library of bit- and cycle-true blocks to build a model in Simulink. They can then use Xilinx System Generator for DSP™, a plug-in to Simulink code generation software, to automatically generate synthesizable hardware description language (HDL) code mapped to pre-optimized Xilinx algorithms.

Table 1 summarizes the complementary features and benefits of HDL Coder and System Generator.

Used independently, each approach provides an effective FPGA design flow. Some projects, however, benefit from a mixture of approaches – a workflow that combines the native Simulink workflow, device-independent code, and code readability offered by HDL Coder, with the Xilinx FPGA-specific features and optimizations offered by Xilinx System Generator.

This paper describes two workflows. In the first, the design is created with blocks from both Simulink and System Generator for DSP. In the second, a Xilinx System Generator for DSP Black Box is generated from a MATLAB HDL design.

Prior experience with MATLAB, Simulink, and Xilinx System Generator will help you make the most of the examples in this paper.

## Required Software

The example models described in this paper are from two examples included with HDL Coder: “Using Xilinx System Generator for DSP with HDL Coder” and “Generate Xilinx System Generator for DSP Black Box from MATLAB HDL Design”.

Simulation and code generation from the models have been tested with the following versions of the software:

- MATLAB (R2015b)
- Simulink
- HDL Coder
- MATLAB® Coder™
- Fixed-Point Designer™
- Xilinx Vivado® System Edition Design Suite 2014.4

Feature	HDL Coder	System Generator	Benefit
MATLAB to HDL	X		Rapidly prototype algorithmic MATLAB
Floating- to Fixed-Point Conversion	X		Shorten design cycles
Design exploration	X		Rapidly explore hardware solution space
Software and hardware code generation	X		Partition algorithms between processors and hardware
Access to Simulink block library	X		Rapidly assemble system models using existing blocks
Support for native Simulink blocks	X		Easily migrate from system model to hardware
Automatic test generation	X		Verify hardware against system models
Transaction Level Model (TLM) component generation	X		Support system level modeling
Readable, traceable HDL code	X		Streamline standards compliance and reporting
Access Xilinx IP in Simulink		X	Generate implementations optimized for Xilinx targets
Hardware co-simulation		X	Verify hardware implementations on Xilinx development boards
Analog data acquisition		X	Verify algorithms to real world analog data
Hardware deployment		X	Deploy designs in hardware without FPGA design experience

Table 1. HDL Coder and System Generator complementary features and benefits.

## Setting Up the MATLAB Environment for HDL Code Generation and Xilinx System Generator Integration

Before you can begin working on your model, you need to ensure that the MATLAB environment is aware of System Generator and that System Generator is configured to work with MATLAB.

The System Generator MATLAB Configurator is installed as part of the Vivado Design Suite package (Figure 1). You'll need to run this to make System Generator aware of the version of MATLAB you plan to use.

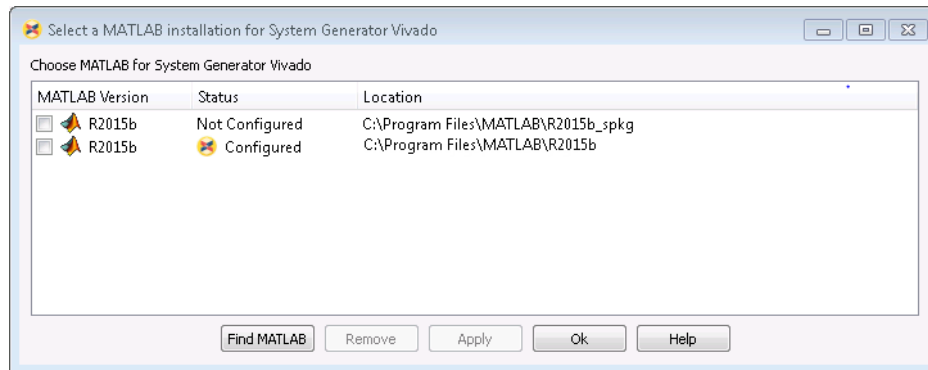


Figure 1. MATLAB Configurator window.

Note that you need to run the Configurator only once, prior to the first use of a model containing System Generator elements.

Your MATLAB environment also needs to be made aware of the ISE Design Suite installation. This is accomplished through the `hdlsetuptoolpath` command in MATLAB. The command below shows the typical path for a Windows PC ISE installation.

```
>> hdlsetuptoolpath('ToolName','Xilinx Vivado','ToolPath',' c:\Xilinx\Vivado\2014.4\bin\vivado.bat');
```

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

### Example 1: Design with Blocks from Both Simulink and Xilinx System Generator for DSP

The example model (`hdlcoder_slsysgen.slx`) performs image filtering. The top level of the design contains two subsystems, one implemented with Xilinx blocks and the other with Simulink blocks (Figure 2). Because they are designed for synthesis on Xilinx devices, Xilinx blocks will yield an optimized implementation of this 5x5 image filter on a Xilinx FPGA. Users can also explore various optimizations through HDL Coder on the Simulink part of the design.

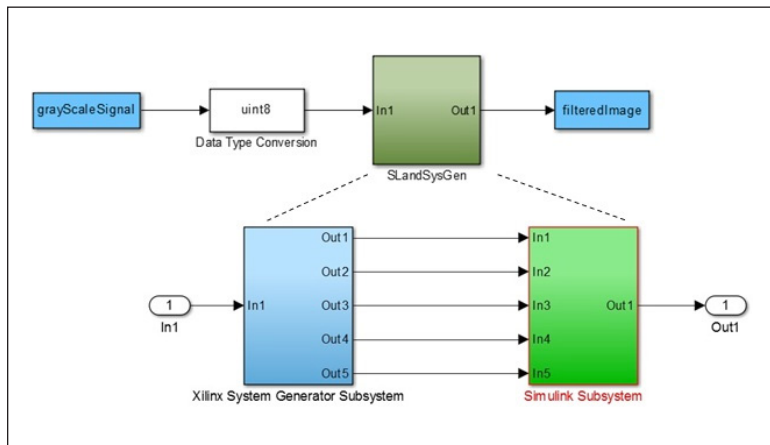


Figure 2. Simulink model of an image filtering system (hdlcoder\_slsysgen.slx).

### Preparing a Model for HDL Code Generation

In a typical development workflow, engineers model and simulate a design in Simulink, completing multiple iterations to identify and eliminate design problems in preparation for implementation. Before using Simulink HDL Coder and Xilinx System Generator to generate code, however, you must prepare the model by:

- Creating a Xilinx System Generator subsystem
- Matching port data types
- Setting code generation options for Xilinx System Generator
- Setting code generation options for HDL Coder

### Creating a Xilinx System Generator Subsystem

To create a Xilinx System Generator subsystem:

1. Put all the Xilinx blocks in one subsystem. (In general, you can create multiple System Generator subsystems, and all Xilinx blocks must be contained in these subsystems. For this example, you will use a single System Generator subsystem.)
2. . Ensure the subsystem's Architecture parameter is set to Module, which is the default value. (See Figure 7 for more on this setting.)
3. Place a System Generator token at the top level of the subsystem (Figure 3). You can have a subsystem hierarchy in a Xilinx System Generator Subsystem, but there must be a System Generator token at the top level of the hierarchy.

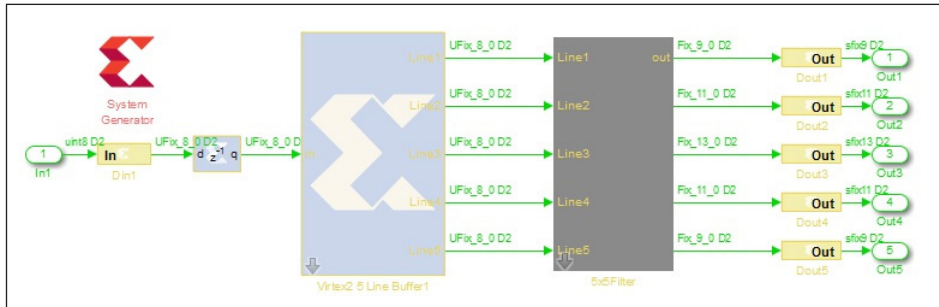


Figure 3. The image processing system's Xilinx System Generator subsystem.

You can use the following MATLAB command to open the System Generator subsystem in the example model.

```
>> open_system('hdlcoder_slsysgen/SLandSysGen/Xilinx System Generator Subsystem');
```

### Matching Port Data Types

In each Xilinx System Generator subsystem, you must connect input and output ports directly to Gateway In and Gateway Out blocks.

Gateway In blocks must not do non-trivial data type conversion. For example, a Gateway In block can convert between uint8 and UFix\_8\_0, but changing data sign, word length, or fraction length is not allowed (Figure 4).

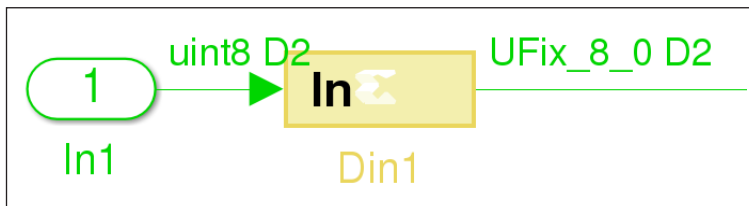


Figure 4. A Gateway In block with matching data types.

In Gateway Out blocks, the Propagate Data Type To Output option must be selected (Figure 5).

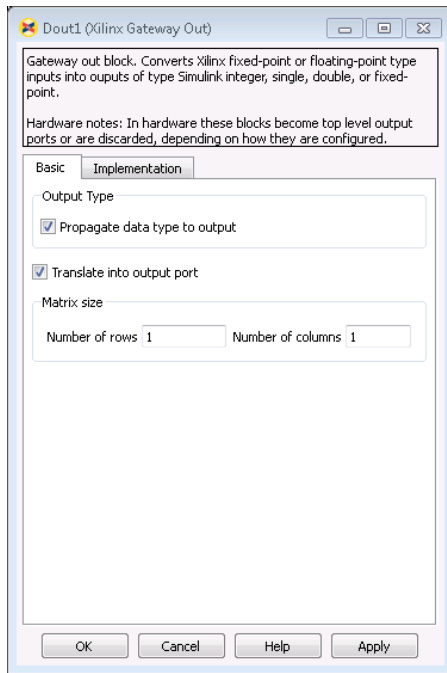


Figure 5. Configuration options for a Gateway Out block.

### Setting Code Generation Options for Xilinx System Generator

With HDL Coder, multiple Xilinx System Generator subsystems can be used in a design, but all Xilinx System Generator tokens must have the same port settings.

HDL Coder supports Xilinx System Generator code generation with the following settings only (Figure 6):

- Compilation must be HDL Netlist.
- Hardware description language must be the same as Target Language setting in HDL Coder.
- Create testbench must be unchecked.
- Multirate implementation must be Clock Enables.
- Synthesis strategy must be Vivado Synthesis\*.
- Implementation strategy must be Vivado Implementation Defaults\*.
- Provide clock enable clear pin must be Checked.



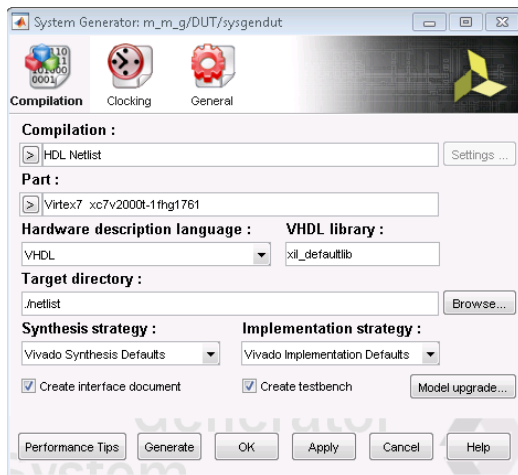


Figure 6. Compilation options for the Xilinx System Generator token.

Note: The Multirate Implementation and Provide Clock Enable Clear Pin Settings are on a different tab and are not shown in Figure 6.

You do not need to configure these settings yourself. HDL Coder will modify and restore these settings on the Xilinx System Generator token during code generation.

### Setting Code Generation Options for HDL Coder

In addition to the settings described above for HDL code generation, HDL Coder requires the Architecture parameter of Xilinx System Generator subsystems to be set to Module (Figure 7). If code generation is performed with HDL Workflow Advisor, the device settings for Workflow Advisor and Xilinx System Generator tokens must be identical, as shown in Figures 6 and 8.

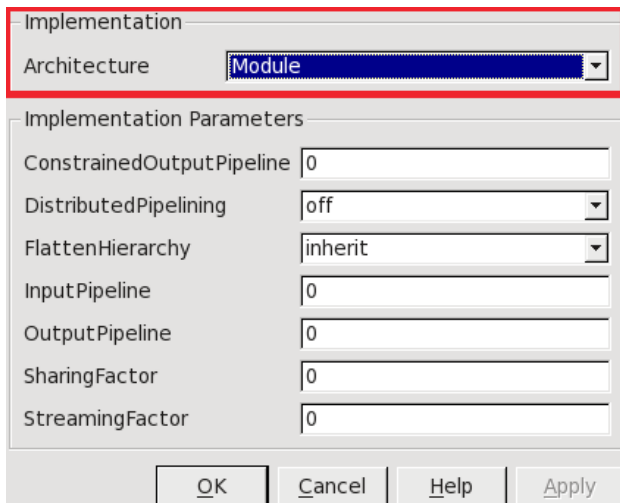


Figure 7. Xilinx System Generator subsystem implementation parameters in HDL Coder.

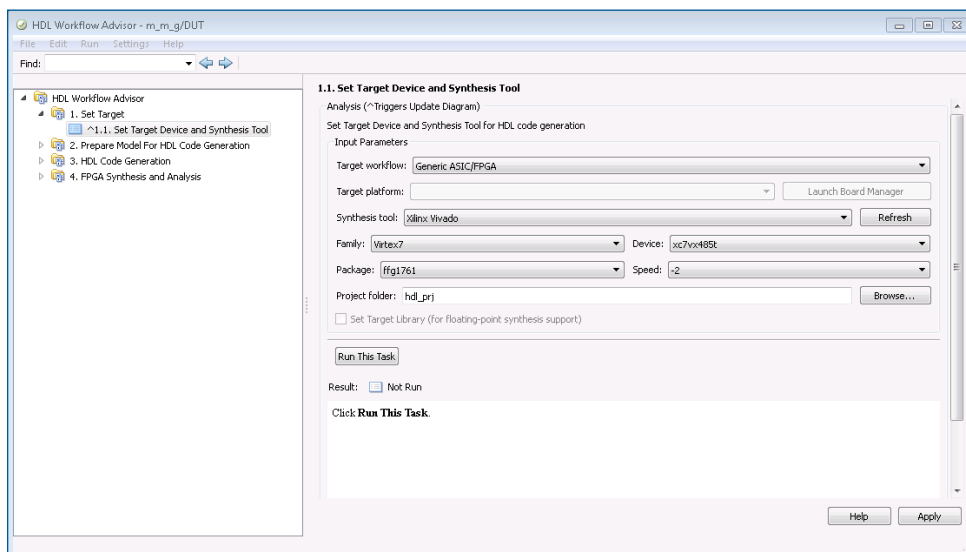


Figure 8. Tool and device options in HDL Workflow Advisor.

## Generating HDL

You can generate HDL code from the configured model with the command line interface or with the GUI, as with any other model. For this example model, the command to generate code is:

```
>> makehdl('hdlcoder_slsysgen/SLandSysGen');
```

Refer to the “HDL Code Generation from a Simulink Model” tutorial that ships with HDL Coder for details on how to generate code using the GUI.

## Generating HDL Test Bench and Simulation Scripts

If you want to simulate the primitives used by HDL code from Xilinx System Generator, you must compile Xilinx Simulation libraries with `compile_simlib`, a tool from Xilinx. For example, the following command compiles all libraries for all FPGA device families, for both VHDL and Verilog, and for ModelSim® SE:

```
% compile_simlib -s mti_se -arch all
```

When generating simulation scripts, HDL Coder uses `compile_simlib` to locate the compiled libraries. If `compile_simlib` fails to find the compiled libraries, HDL Coder gives a warning about incomplete simulation scripts. You can manually provide the location of the compiled libraries location, using the `XilinxSimulatorLibPath` option via the command line and adding the path information for your compiled libraries (the default from the command line is the current directory you are in when running the script):

```
>> makehdltb('hdlcoder_slsysgen/SLandSysGen', 'XilinxSimulator-  
LibPath', '<user to insert path to compiled libraries>');
```

Alternatively, you can set it as a model parameter and generate a test bench:

```
>> hdlset_param('hdlcoder_slsysgen', 'XilinxSimulatorLibPath',
'<user to insert path to compiled libraries>');

>> makehdltb('hdlcoder_slsysgen/SLandSysGen');
```

## Example 2: Generate a Xilinx System Generator for DSP Black Box Block from a MATLAB HDL Design

After designing an algorithm in MATLAB for HDL code generation, you can integrate it into a larger system as a Xilinx System Generator Black Box block. HDL Coder can generate the System Generator Black Box block and configuration file from your MATLAB HDL design and place the generated Black Box block in a Xilinx System Generator subsystem.

The MATLAB code (mlhdlc\_fir.m) in this example implements a simple finite impulse response (FIR) filter. The example also includes a MATLAB test bench (mlhdlc\_fir\_tb.m) that exercises the filter.

To use this design and experiment with generating a Xilinx System Generator for DSP Black Box block, the main steps are:

- Creating a New Folder and Copying Relevant Files
- Simulating the Design (optional)
- Creating a New Project
- Generating the Xilinx System Generator for DSP Black Box
- Running Fixed-Point Conversion and Generating Code
- Examining the Generated Model and Configuration File
- Deleting the Temporary Files

### Creating a New Folder and Copying Relevant Files

Execute the following lines of code to copy the necessary example files into a temporary folder.

```
>> mlhdlc_demo_dir = fullfile(matlabroot, 'toolbox', 'hdlcoder',
'hdlcoderdemos', 'matlabhdlcoderdemos');
>> mlhdlc_temp_dir = [tempdir 'mlhdlc_fir'];
% Create a temporary folder and copy the MATLAB files
>> cd(tempdir);
>> [~, ~, ~] = rmdir(mlhdlc_temp_dir, 's');
>> mkdir(mlhdlc_temp_dir);
>> cd(mlhdlc_temp_dir);
>> copyfile(fullfile(mlhdlc_demo_dir, design_name), mlhdlc_temp_
dir);
>> copyfile(fullfile(mlhdlc_demo_dir, testbench_name), mlhdlc_temp_
dir);
```

## Simulating the Design

It is a good idea to simulate the design with the test bench to make sure there are no runtime errors before code generation. Use the following command to start the simulation:

```
>> mlhdlc_fir_tb
```

## Creating a New Project from the Command Line

To create a new project, execute the following command:

```
>> coder -hdlcoder -new fir_project
```

Next, add the mlhdlc\_fir.m file to the project as the MATLAB Function and add mlhdlc\_fir\_tb.m as the MATLAB Test Bench.

Click the Workflow Advisor button to launch the HDL Workflow Advisor.

## Generating a Xilinx System Generator for DSP Black Box

To generate a Xilinx System Generator Black Box from a MATLAB HDL design, you must first configure Xilinx System Generator.

On the Advanced tab of the Workflow Advisor, select the Generate Xilinx System Generator Black Box option (Figure 9).

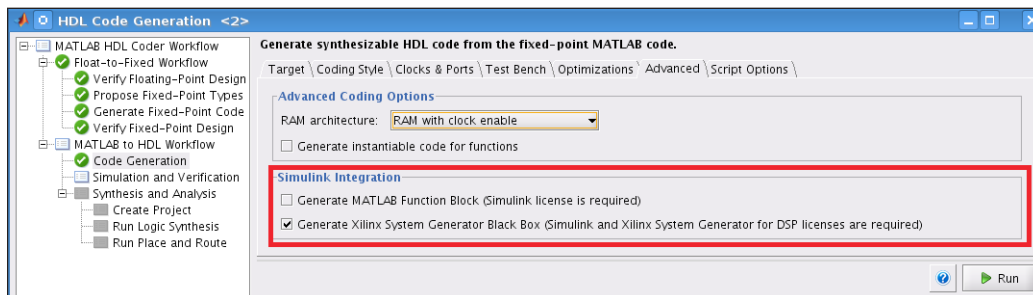


Figure 9. The Generate Xilinx System Generator Black Box code generation option.

To generate code compatible with a Xilinx System Generator Black Box block, set the following options (Figure 10):

- Set Clock Input Port to clk
- Set Clock Enable Input Port to ce
- Set Drive Clock Enable At to DUT base rate

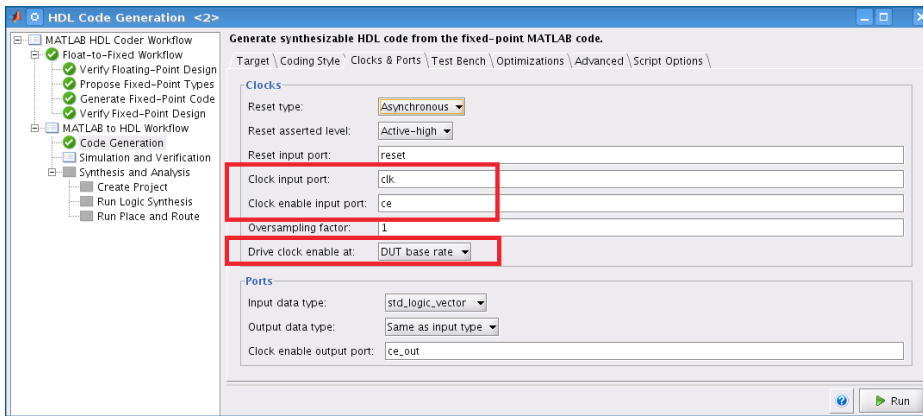


Figure 10. HDL code generation options.

### Running Fixed-Point Conversion and Generating Code

Right-click the Code Generation step on the left and choose Run To Selected Task to run all the steps from the beginning through HDL code generation.

### Examining the Generated Model and Configuration File

After HDL code generation, a new model opens. It contains a subsystem named DUT at the top level. The DUT subsystem has a Xilinx System Generator subsystem named SysGenSubSystem (Figure 11), which contains:

- A Xilinx System Generator Black Box block
- A System Generator block
- Gateway In blocks
- Gateway Out blocks

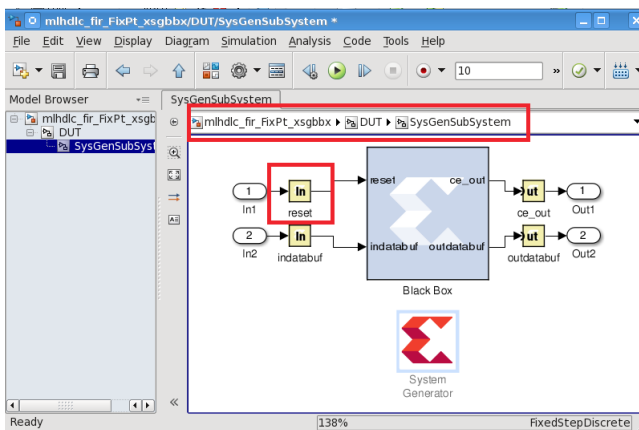
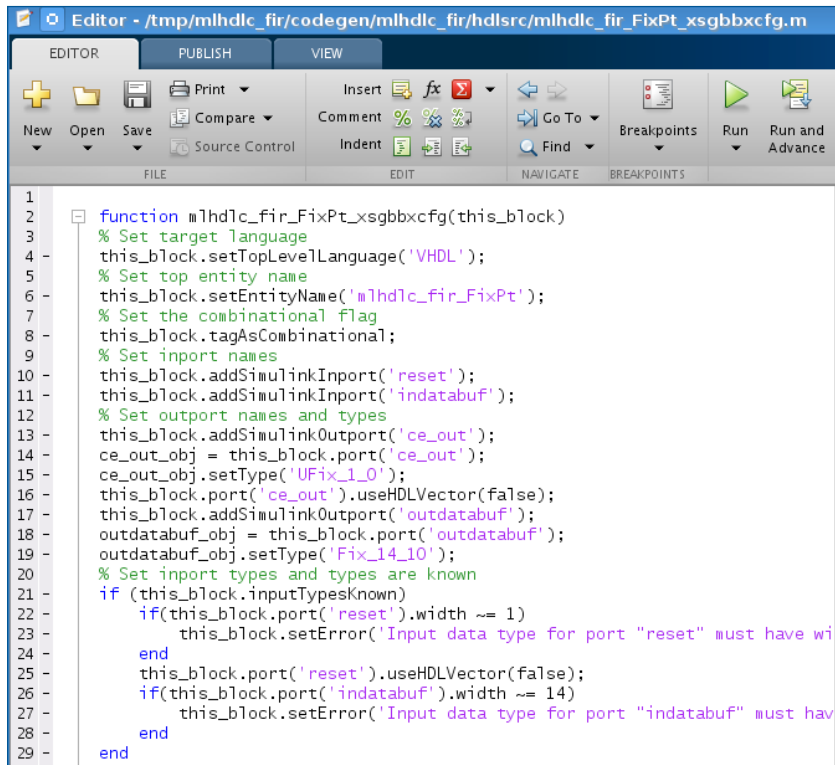


Figure 11. Xilinx System Generator subsystem containing a generated Black Box block.

Notice that in addition to the data ports, there is a reset port on the Black Box interface. The other common control signals `clk` (clock) and `ce` (clock enable) are handled differently. These two inputs are registered to System Generator by the Black Box configuration file.

The configuration file (Figure 12) and your new model are saved in the same directory as the generated HDL code. You can open the configuration file by executing the following command:

```
>> edit('codegen/mlhdlc_fir/hdlsrc/mlhdlc_fir_FixPt_xsgbbxcfg.m');
```



```

1
2 function mlhdlc_fir_FixPt_xsgbbxcfg(this_block)
3 % Set target language
4 this_block.setTopLevelLanguage('VHDL');
5 % Set top entity name
6 this_block.setEntityName('mlhdlc_fir_FixPt');
7 % Set the combinational flag
8 this_block.tagAsCombinational;
9 % Set inport names
10 this_block.addSimulinkInport('reset');
11 this_block.addSimulinkInport('indatabuf');
12 % Set output names and types
13 this_block.addSimulinkOutport('ce_out');
14 ce_out_obj = this_block.port('ce_out');
15 ce_out_obj.setType('UFix_1_0');
16 this_block.port('ce_out').useHDLVector(false);
17 this_block.addSimulinkOutport('outdatabuf');
18 outdatabuf_obj = this_block.port('outdatabuf');
19 outdatabuf_obj.setType('Fix_14_10');
20 % Set inport types and types are known
21 if (this_block.inputTypesKnown)
22     if(this_block.port('reset').width ~= 1)
23         this_block.setError('Input data type for port "reset" must have width 1');
24     end
25     this_block.port('reset').useHDLVector(false);
26     if(this_block.port('indatabuf').width ~= 14)
27         this_block.setError('Input data type for port "indatabuf" must have width 14');
28     end
29 end

```

Figure 12. Xilinx System Generator Black Box configuration file.

You can now use the generated Xilinx System Generator Black Box block and configuration file in a larger system design.

### Deleting the Temporary Files

When you are finished with this example, you can run the following commands to clean up the temporary project folder.

```
>> mlhdlc_demo_dir = fullfile(matlabroot, 'toolbox', 'hdlcoder',  
'hdlcoderdemos', 'matlabhdlcoderdemos');  
>> mlhdlc_temp_dir = [tempdir 'mlhdlc_fir'];  
>> clear mex;  
>> cd (mlhdlc_demo_dir);  
>> rmdir(mlhdlc_temp_dir, 's');
```

### Further Reading

For more HDL Coder videos and examples, visit <http://www.mathworks.com/products/hdl-coder/examples.html>.