Vivado Design Suite Tutorial:

Model-Based DSP Design using System Generator

UG948 (v2014.3) October 28, 2014

This tutorial document has been validated for the following software versions: Vivado Design Suite 2014.3 and 2014.4.
## Revision History

The following table shows the revision history for this document.

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>10/28/2014</td>
<td>2014.3</td>
<td>Lab 2 renamed &quot;Working with Data Types&quot;; Lab 11 renamed &quot;Modeling Blocks with HDL&quot;; tutorial re-validated with release 2014.3.</td>
</tr>
<tr>
<td>12/18/2013</td>
<td>2013.4</td>
<td>No technical updates. Re-published for this release</td>
</tr>
<tr>
<td>10/02/2013</td>
<td>2013.3</td>
<td>Added Lab 10 titled AXI4-Lite Interface Synthesis</td>
</tr>
<tr>
<td>6/19/2013</td>
<td>2013.2</td>
<td>Added Lab 9 titled &quot;Including a System Generator Design as a Module in an IP Integrator Design&quot;</td>
</tr>
<tr>
<td>3/20/2013</td>
<td>2013.1</td>
<td>Initial Xilinx Release</td>
</tr>
</tbody>
</table>
Table of Contents

Vivado Design Suite Tutorial: ................................................................. 1
  Revision History .................................................................................. 2
System Generator for DSP Overview ...................................................... 6
  Introduction .......................................................................................... 6
  Software Requirements ....................................................................... 7
  Hardware Requirements ...................................................................... 7
  Configuring MATLAB to the Vivado® Design Suite......................... 7
  Locating and Preparing the Tutorial Design Files ......................... 8
Lab 1: Using Simulink .......................................................................... 9
  Lab 1: Introduction ............................................................................ 9
  Step 1: Introduction to Simulink ......................................................... 10
  Step 2: Analyzing the Sampling Period Effect .................................. 16
  Step 3: Creating a Simple Filter Design Using Simulink Blocks ....... 17
  Step 4: Creating a Subsystem ......................................................... 18
  Summary ............................................................................................ 19
Lab 2: Working with Data Types .......................................................... 20
  Lab 2 Introduction ............................................................................ 20
  Step 1: Simulate the Executable Specification .................................. 21
  Step 2: Duplicate the Design using Xilinx Blocks ......................... 22
  Step 3: Implement the Xilinx Design .............................................. 25
  Step 4: Explore Different Hardware Architectures ....................... 27
  Summary ............................................................................................ 30
Lab 3: Signal Routing ........................................................................ 31
  Lab 3 Introduction ............................................................................ 31
  Step 1: Designing Padding Logic ..................................................... 32
  Step 2: Designing Unpadding Logic ................................................. 34
  Summary ............................................................................................ 34
Lab 4: System Control ....................................................................... 35
<table>
<thead>
<tr>
<th>Lab 10: Packaging a Synthesized AXI4-Lite Interface</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab 9: Including a System Generator Design as a Module in an IP Integrator Design</td>
<td>77</td>
</tr>
<tr>
<td>Lab 8: Importing C/C++ Source Files into System Generator by Leveraging Integration with Vivado HLS</td>
<td>57</td>
</tr>
<tr>
<td>Lab 7: Including a System Generator Model in a Vivado IDE Design</td>
<td>49</td>
</tr>
<tr>
<td>Lab 6: Using Memories</td>
<td>45</td>
</tr>
<tr>
<td>Lab 5: Multi-Rate Systems</td>
<td>39</td>
</tr>
<tr>
<td>Lab 4: Introduction</td>
<td>35</td>
</tr>
<tr>
<td>Step 1: Designing Padding Logic</td>
<td>35</td>
</tr>
<tr>
<td>Step 2: Changing Sample Rate: Down Sample</td>
<td>39</td>
</tr>
<tr>
<td>Step 3: Changing Sample Rate: Up Sample</td>
<td>41</td>
</tr>
<tr>
<td>Step 4: Using Serial-to-Parallel Blocks</td>
<td>43</td>
</tr>
<tr>
<td>Step 5: Using Parallel-to-Serial Blocks</td>
<td>44</td>
</tr>
<tr>
<td>Summary</td>
<td>48</td>
</tr>
<tr>
<td>Lab 3: Including a System Generator Package from Vivado HLS</td>
<td>45</td>
</tr>
<tr>
<td>Step 1: Creating a new Vivado IDE Project with DSP Sources</td>
<td>49</td>
</tr>
<tr>
<td>Step 2: Explore the DSP Module from the Vivado IDE Cockpit</td>
<td>53</td>
</tr>
<tr>
<td>Summary</td>
<td>56</td>
</tr>
<tr>
<td>Lab 2: Including a Vivado HLS Package in a System Generator Design</td>
<td>57</td>
</tr>
<tr>
<td>Step 1: Packaging IP with Port/Pin Interfaces</td>
<td>66</td>
</tr>
<tr>
<td>Step 2: Packaging the IP with Port-Name Interface Inference</td>
<td>67</td>
</tr>
<tr>
<td>Step 3: Programming an AXI4 Interface from Software</td>
<td>71</td>
</tr>
<tr>
<td>Step 4: Programming an AXI4-Lite Interface</td>
<td>74</td>
</tr>
<tr>
<td>Lab 1: Designing Padding Logic</td>
<td>77</td>
</tr>
<tr>
<td>Lab 10 Introduction</td>
<td>77</td>
</tr>
<tr>
<td>Requirements</td>
<td>77</td>
</tr>
<tr>
<td>Section</td>
<td>Page</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>Step 1</td>
<td>78</td>
</tr>
<tr>
<td>Conclusion</td>
<td>86</td>
</tr>
<tr>
<td>Lab 11: Modeling Blocks with HDL</td>
<td>87</td>
</tr>
<tr>
<td>Lab 11 Introduction</td>
<td>87</td>
</tr>
<tr>
<td>Step 1: Import RTL as a Black Box</td>
<td>87</td>
</tr>
<tr>
<td>PLEASE READ: Important Legal Notices</td>
<td>92</td>
</tr>
<tr>
<td>Notice of Disclaimer</td>
<td>92</td>
</tr>
</tbody>
</table>
System Generator for DSP Overview

Introduction

The System Generator for DSP is a design tool in the Vivado® Design Suite that enables you to use the MathWorks® model-based Simulink® design environment for FPGA design. Previous experience with Xilinx® FPGA devices or RTL design methodologies is not required when using System Generator. Designs are captured in the Simulink™ modeling environment using a Xilinx-specific block set. Downstream FPGA implementation steps including RTL synthesis and Place and Route are automatically performed to produce an FPGA programming bitstream.

Over 80 building blocks are included in a Xilinx-specific DSP block set for Simulink. These blocks include common building blocks such as adders, multipliers and registers. Also included are complex DSP building blocks such as forward-error-correction blocks, FFTs, filters and memories. These blocks leverage Xilinx LogiCORE™ IP to produce optimized results for the selected target device.

In this tutorial, you will do the following:

- Lab 1: Use Simulink to create a simple design, create a subsystem, and then simulate.
- Lab 2: Build a basic design in System Generator, simulate the design, and then generate an FPGA bitstream for a target Xilinx technology.
- Lab 3: Learn how signal routing blocks are used to re-define or modify fixed-point numbers at the bit-level.
- Lab 4: Create a finite state machine using the Mcode block in System Generator.
- Lab 5: Change sample rates in a multi-rate DSP design and then convert a serial data stream to a parallel data word and a parallel data word to a serial data stream.
- Lab 6: Use a Xilinx ROM block to implement a trig or math function such as arcsin.
- Lab 7: Include a System Generator model within a Vivado IDE design and combine that model with other RTL sources.
- Lab 8: Import C/C++ source files into a System Generator model by leveraging the tool integration with Vivado High-Level Synthesis (HLS).
- Lab 9: Include a System Generator Design as a Module in an IP Integrator Design
- Lab 10: AXI4-Lite Interface Synthesis
- Lab 11: Black Box Examples
Software Requirements

The lab exercises in this tutorial require the installation of MATLAB 2014a (or later) and Vivado Design Environment 2014.3 (or later).

Hardware Requirements

The supported Operating Systems include Redhat 5.6 Linux 64, and Windows 7 (32 and 64 bit). Xilinx recommends a minimum of 2 GB of RAM when using the Vivado Design Suite.

Configuring MATLAB to the Vivado® Design Suite

Before you begin, you should verify that MATLAB is configured to the Vivado Design Suite. Do the following:

1. Select Start > All Programs > Xilinx Design Tools > Vivado 2014.x > System Generator > System Generator 2014.x MATLAB Configurator

2. Click the check box of the version of MATLAB you want to configure and then click OK.

On Windows OS systems you may need to launch the MATLAB configurator as Administrator. When MATLAB configurator is selected in the menu, use the mouse right-click to select Run as Administrator.
Locating and Preparing the Tutorial Design Files

There are separate project files and sources for each of the labs in this tutorial. You can find the design files for this tutorial under **Vivado Design Suite -2014.3 Tutorials** on the Xilinx.com website.

1. **Download** the **ug948-design-files.zip** file from the Xilinx website.
2. **Extract** the zip **file contents** into any write-accessible location on your hard drive or network location.

---

**RECOMMENDED:** You will modify the tutorial design data while working through this tutorial. You should use a new copy of the **ug948-design-files** directory each time you start this tutorial.
Lab 1: Using Simulink

Lab 1: Introduction

In this lab exercise, you will learn the basics of using Simulink. You will use a Simulink block set to generate a simple design and take it through simulation. You will then change the sampling settings to see its effect on the output. You will then learn how to create a subsystem.

Objectives

After completing this lab, you will be able to:

- Use the Simulink tool to create a simple design
- Create a subsystem and simulate
- Describe the effect of the sampling period

Procedure

This exercise has four primary parts.

- In Step 1, you are introduced to the Simulink environment.
- In Step 2, you will analyze the effect of the sampling period.
- In Step 3, you will create a simple filter design using a Simulink block set.
- Finally, in Step 4, you will create a subsystem of the design and perform simulation.
Step 1: Introduction to Simulink

In this Step you become familiar with the MATLAB and Simulink environments (software tools from The MathWorks suite). You will start with a blank worksheet, add a Sine Wave source element, add a Scope sink element, and wire the two together.


2. Navigate to the lab1 folder: cd C:/ug948-design-files/lab1.
   You can view the directory contents in the MATLAB Current Directory window, or type ls at the command line prompt. Many UNIX-type shell commands work from the MATLAB command window.

3. Type simulink at the MATLAB command prompt, or click the Simulink button in the MATLAB toolbar to open the Simulink Library Browser.

4. Examine the available blocks in the Simulink Library Browser.
   The following elements, among others, display:
   - Simulink (sources and sinks)
   - Xilinx Blockset
   - Xilinx Reference Blockset
This provides details about the block.
Step 1: Introduction to Simulink

You can also select Help from the Xilinx Blockset elements as shown below.

5. Click the **New model** button in the Simulink Library Browser to create a new model blank sheet. The name of the new model worksheet is untitled by default. Later you will change the name of the model.

6. In the Library Browser window, expand the **Simulink Library** and click **Sources**.

7. Scroll through the library to find the Sine Wave source. Select and drag Sine Wave onto the worksheet.

8. From the Simulink Browser, select Simulink Sinks, add the **Scope** block, and draw a wire from the Sine Wave to the Scope block. An automatic block connection tip appears.

   *Note: To draw a wire, move the cursor to the source output port (the cursor will become a cross-hair). Click and drag your mouse to an input port of destination.*

### Assigning the Frequency

Next, assign a $2\pi(1/150)$ frequency to the Sine Wave element and show port data types. Change the stop time of the simulation to 150 and set the **Solver options**.

1. Double-click the **Sine Wave** block.
   
   The Block Parameters dialog box opens.

2. Change the frequency to $2\pi(1/150)$, as shown below and then click **OK** to close the dialog box.

3. On the worksheet, select **Display > Signal & Ports > Port Data Types**.
Step 1: Introduction to Simulink

The signal type displays above the wire, as shown below:

![Simulink Model](image)

4. From the project sheet, select **Simulation > Model Configuration Parameters**.

5. From the Configuration Parameters dialog box, enter **150.0** in the **Stop time** field, and make the following selections in the **Solver** and **Task** mode options:
   - **Type**: Fixed-step
   - **Solver**: discrete (no continuous states)
   - **Tasking mode for periodic sample times**: SingleTasking

Setting these parameters allows your simulation to run for 150 time units.

6. Click **OK**.
Parameterizing the Scope Block

Next, you parameterize the Scope block.

1. Double-click the Scope block.
2. Click the Scope Parameters button.

3. In the Scope Parameters dialog box, set the Time range to 150 and then click OK.

Next, you run the simulation.
Running the Simulation

1. From your Simulink project worksheet, select **Simulation > Run** or click the **Run** simulation button.

2. On the Scope display, click the **Autoscale** button so that the output fits into the Scope.

3. View the Scope output.

   A smooth sine wave should fit into your scope window, which is what you would expect because you are running a double-precision software simulation.
Step 2: Analyzing the Sampling Period Effect

Now you will run the simulation at a slower sampling period (sample period of 5) of the sine wave source and analyze the sampling period effect. You will then change the stop time to 500, run the simulation, and observe the output. Finally, you will change the stop time back to 150.

1. Double-click the Sine Wave block to open its parameter dialog box.
2. Change the Sample time of the Sine Wave source to 5, click OK, and run the simulation.
   
   *Note: As the Sample time is increased (i.e. fewer inputs are sampled), the quantization error is increased.*

3. Type 500 in the worksheet toolbar window and press Enter to change the simulation stop time.

4. Run the simulation and observe the output in the scope window.
   
   You might need to click the Autoscale button in the Scope block GUI to see three complete Sine wave cycles.

5. Change the simulation stop time back to 150.

6. Change the Sample time of the Sine Wave source back from 5 to 0 and click OK.
Step 3: Creating a Simple Filter Design Using Simulink Blocks

In this step, you build a simple design to implement the following function by using the appropriate blocks from the Simulink block set: \( Y(n+1) = X(n+1) + 3 \times X(n-1) \).

1. From the Simulink Library browser, select the Discrete library.
2. Select and add the Delay block to the design.
3. Double-click the Delay block and verify the Delay length set to 2.
4. From the Simulink Library browser, select the Math Operations library.
5. Select and add the Gain block to the design.
6. Double-click the Gain block and change the Gain to 3.
7. From the Simulink Library browser, select the Math Operations library.
8. Select and add the Add block to the design.
9. Connect the blocks as shown below:

![Diagram of Simulink blocks](image)

10. Run the simulation and observe the output.
Step 4: Creating a Subsystem

In this step you will select all the blocks between the source and sink, and then create a subsystem. You will name the subsystem **Filter**, run the simulation and verify that the output is still the same. You then save the model as **lab1.slx** in the current working directory and then exit MATLAB.

1. Click in a white space and create a rectangle enclosing all relevant blocks and connections to select all the blocks between the source and sink.

2. Select **Diagram > Subsystem & Model Reference > Create Subsystem from Selection**, or press **Ctrl+G**.
3. Name the subsystem **Filter** by selecting and typing over the title.

![Diagram](image1)

You can adjust block placements so that the design appears lined up. The design should look similar to the following figure.

![Diagram](image2)

4. Click **File > Save** and enter **lab1.slx** as the filename.
5. Click **Save** to save the file in the current directory.

   Notice that the worksheet name at the top-left corner changes from **untitled** to **lab1**.

6. Type **exit** in the MATLAB command window to close MATLAB.

---

**Summary**

In this lab, you learned the basic design flow involved in Simulink using the Simulink block set. You observed the effect of a sampling period. You simulated the design using the Simulink simulator and observed the output using the Scope block. Finally, you created a subsystem. The solution to this lab can be found in the following location:

   C:/ug948-design-files/lab1/solution.
Lab 2: Working with Data Types

Lab 2 Introduction

This lab exercise introduces you to the basic concepts of creating a design using System Generator for DSP within the model-based design flow provided through Simulink. The design is a simple multiply-add circuit.

Objectives

After completing this lab, you will be able to:

- Simulate a design in System Generator for DSP
- Run the System Generator token to generate a Xilinx FPGA bitstream
- Create a subsystem
- Improve performance using dedicated Xilinx FPGA math functions

Procedure

This lab has four primary Steps.

- In Step 1, you open and simulate a Simulink block set-based design that serves as an executable specification.
- In Step 2, you re-create the Simulink design using the Xilinx block set.
- In Step 3, you take the Xilinx executable specification through the full implementation flow.
- Finally, in Step 4, you explore different hardware architectures to achieve the best performance.
Step 1: Simulate the Executable Specification

1. Launch the MATLAB program and change the working directory to:
   C:/ug948-design-files/lab2

2. Open the file lab2.slx, and observe the following design.

   ![Simulink Diagram]

   **Note:** This design is an executable specification created in Simulink using the standard Simulink block set. It is a simple multiply-add circuit but serves to demonstrate many of the key concepts of model-based design. You are going to design a Xilinx FPGA to this specification.

3. Simulate the design for 100 cycles by pressing the **play** button on the toolbar.

4. View the waveform by double-clicking on the **Scope block**.
Step 2: Duplicate the Design using Xilinx Blocks

1. From the Simulink® Library browser, open the Xilinx® Blockset Library and the Index sub-library to access the blocks.

2. Create a Xilinx version of the multiply/add design using Xilinx blocks.

**IMPORTANT:** Remember you must use the Xilinx Gateway In / Gateway Out blocks to define the FPGA boundary and you must place a Xilinx System Generator™ token in the design, as shown in the following figure.

3. Leave all the block settings at their default values, except the AddSub block. Double-click on the AddSub block and change the latency on the AddSub block to 0, to ensure the circuit matches the figure below.

4. Simulate the design and view the waveform on the Scope attached to the Xilinx implementation.
Notice that the waveforms have square edges. This is because the System Generator block set forces a discrete sampling of the input signals that represents the behavior of the actual hardware that operates on synchronous clock edges.

5. Compare the results from the executable specification versus the Xilinx implementation using a **Subtractor** from the Simulink/Math Operations library as shown below. This is an important model-based design concept.

6. Simulate the design and view the waveform on the Scope

As shown in the following figure, the Scope output shows the outputs from both paths are different.
Step 2: Duplicate the Design using Xilinx Blocks

The reason for these differences is that the elements in the Xilinx block set introduce delays.

7. Double-click on the Mult block to confirm the latency is 3.

This latency models that the multiplication will require 3 clock cycles to complete. To ensure both paths have the same behavior in time, you must balance the delays.

8. Add a Delay block from the Discrete Library at the output of the Product.

9. Double-click on the Delay block and set the Delay Length to 3 and Sample time to 1.

The design should now look like the following figure:

10. Re-simulate the design.

The design is now functionally matching the executable specification. The next part is to perform the FPGA implementation steps that include RTL generation, RTL synthesis, and Place and Route.
Step 3: Implement the Xilinx Design

1. Double-click the **System Generator** token and set the **Compilation** target to **HDL Netlist**, as shown below. Also, set the **Part** to **Kintex7 xc7k325t-2ffg900**.

2. Click the **Generate** button to initiate the netlisting process.

   System Generator automatically executes the HDL netlist generation and displays the following message when finished.

3. Invoke the Vivado Integrated Design Environment (IDE):

   **Start > All Programs > Xilinx Design Tools > Vivado 2014.x > Vivado 2014.x**

4. Click **Open Project** and then navigate to the folder

   C:/ug948-data-files/lab2/netlist/hdl_netlist.
5. Double-click the file `lab2.xpr` and the Vivado IDE invokes the generated project file as shown:

6. Click **Run Implementation**, select **OK** when prompt for missing synthesis results (synthesis will be run) and let the Synthesis and Implementation processes complete.

   You now have an initial implementation.

7. Close the Vivado IDE and the lab2.xpr project.

   The remainder of this lab focuses on some common techniques used to explore different hardware architectures.
Step 4: Explore Different Hardware Architectures

1. Select all the Xilinx components, including the System Generator token and place them into a subsystem by pressing Ctrl-G. The diagram should look like the following figure.

You now have a Simulink diagram that contains two subsystems each with a System Generator token. This represents two FPGAs or two blocks of a single FPGA within a larger DSP system.

2. Copy the subsystem to create a second subsystem and connect it to the design as shown in the following figure.
Step 4: Explore Different Hardware Architectures

Each System Generator token creates a top-level entity from the subsystem from which it is associated. It will not merge with the other subsystem.

**TIP:** Creating subsystems can be a useful technique when exploring hardware architectures for a given design.

3. Push into the copy of the subsystem (subsystem1) and modify the design to implement the same function using the Xilinx DSP48 Macro 3.0 block.

4. Select and delete the Mult and AddSub blocks.

5. Right-click, select Xilinx BlockAdd and type dsp in the Add block field.

6. Double-click the DSP48 Macro 3.0 block and position the block as shown below.

Using this block allows improved control over the hardware implementation. The DSP48 macro forces the use of the DSP48 primitives in the final netlist. Port c on the DSP 48 Macro 3.0 block requires an input quantization of Fix_48_28.

7. Double-click the Gateway In2 block:
   a. Set the Number of bits to 48
   b. Set the Binary point to 28
   c. Click OK.

8. To view the port quantization values as shown above, execute the following pulldown menus: Display > Signals & Ports > Port Data Types.
Step 4: Explore Different Hardware Architectures

9. Double-click the **DSP48 3.0** Macro block to bring up the properties dialog box editor.
10. From the Instructions tab, verify the equation to be A*B+C, as shown below.

![Image of DSP48 Macro 3.0 properties dialog box with A*B+C equation highlighted]

11. Observe the other implementation options, leaving them at their default values.
12. Click the Pipeline Options tab and set the **Pipeline Options** to **Expert**, as shown in the following figure.
13. As shown, click the following:
   a. A, B, and C boxes in Tier 1
   b. M and C boxes in Tier 5
14. Un-click all other boxes, then click OK.
15. Re-simulate the design to insure functional correctness.
16. Double-click the System Generator token to re-generate the HDL Netlist. After the netlist generation completes, save the lab2.slx design model and exit from MATLAB.

   NOTE: If you get an error, it usually means that you forgot to close the lab2.xpr project file at the end of the previous major Step.

Summary
In this lab, you learned the basic design flow involved in Simulink using the Simulink block set. You observed the effect of a sampling period. You simulated the design using the Simulink simulator and observed the output using the Scope block. Finally, you created a subsystem. The complete solution to this lab is in the following location:

C:/ug948-design-files/lab2/solution
Lab 3: Signal Routing

Lab 3 Introduction

This lab exercise introduces you to the System Generator features that you use to:

- Convert fixed-point numbers from floating-point
- Re-define the fixed-point format
- Perform bit slice, pad and unpad operations.

You will also design and verify the padding and unpadding logic using the System Generator signal routing blocks.

Objectives

After completing this lab, you will be able to:

- Understand how signal routing blocks can be used to redefine or modify a fixed-point number at the bit level
- Convert a fixed-point number into a new fixed-point number
- Slice bits from a fixed-point number
- Pad and Unpad a fixed-point number

Procedure

In this lab you will create an initial design and use the Reinterpret and Concat blocks to convert a fixed-point number to a floating point number. In Step 2, you use the Slice and Reinterpret blocks to convert the numbers in the opposite direction.
Step 1: Designing Padding Logic

1. Launch the MATLAB program and change the working directory to:
   
   `C:/ug948-design-files/lab3`

2. Launch the Simulink library browser by clicking the **Simulink** icon on the MATLAB toolbar:

3. As shown in the following figure, click the **New** model button in the **Simulink** Library Browser to create a new model blank sheet.

![Simulink Library Browser](image)

4. Create the design shown below.
   
   a. Use a Constant block from the Simulink Sources block set and a Display block from the Simulink **Sinks** block set
   
   b. Set the value of the constant to `.5`
   
   c. Add a Xilinx **Gateway In** block quantized to **fixed_8_6** (signed 2’s comp)
   
   d. Add a Xilinx **Gateway Out** block

5. Connect the blocks already added in the design model as shown below.

   **TIP:** Remember that a System Generator token is also required in this diagram.

6. To display the signal, type **Fix_8_6** as shown below by selects the work sheet menu item **Display >Signals & Ports >Port Data Types**, and then run a simulation.
The objective of this lab is to convert the binary representation of the number .5 when quantized to fixed [8 6] to the number .007813 when quantized to ufixed [12 12].

To accomplish this you are going to have to zero pad the MSBs and reinterpret the number. This requires the use of the Concat and Reinterpret blocks. Review the Help text for these blocks to gain a greater understanding of how they work.

7. Now modify the block diagram previously shown to convert the input constant value of .5 to an output value of .007813. You will first need to use the Reinterpret block to convert the number to ufix [8 0]. You then use a Constant and Concat block to convert to ufix [12 0], and then use another Reinterpret block to convert ufix [12 12].

The solution to this exercise can be found in the following location:
C:/ug948-design-files/lab3/solution/lab3_step1_solution.slx.
Step 2: Designing Unpadding Logic

You are going to perform an exercise similar to Step 1, but in the other direction.

Here the input is the constant .007813 and you want the output to be converted to .5 through bit manipulation, not arithmetic.

1. Create the design as shown in the following figure. The input constant should be set to .007813 and the input gateway can be set to fix [12 12].

   Remember to include the System Generator token in the diagram.

2. Use the **Slice** and **Reinterpret** blocks to manipulate the binary number to achieve an output of .5.
   a. First, use the **Slice** block to convert the number to ufix [8 0]
   b. Then use the **Reinterpret** block to get fix [8 6]. Refer to the binary diagrams in Step 1 of this lab.

   The solution to this exercise can be found in the following location:
   C:/ug948-design-files/lab3/solution/lab3_step2_solution.slx.

Summary

In this lab, you:

- Learned how to convert between signal data types.
- Learned how the slice, concatenation and reinterpret blocks are used to manipulate signal types.

Solutions to this lab can be found in the following location:
C:/ug948-design-files/lab3/solution.
Lab 4 Introduction

In this lab exercise you will be creating a simple finite state machine (FSM) using the MCode block to detect a sequence of binary values 1011. The FSM needs to be able to detect multiple transmissions as well, such as 1011011.

Objectives

After completing this lab, you will be able to create a finite state machine using the Mcode block in System Generator

Procedure

In this lab exercise you will create the control logic for a Finite State Machine using M-code. You will then simulate the final design to confirm the correct operation.

Step 1: Designing Padding Logic

1. Launch the MATLAB program and change the working directory to:
   C:/ug948-design-files/lab4.
2. Open the file lab4.slx.
   You see the following uncompleted diagram.
   ![Diagram](image)
3. Add an MCode block from the Xilinx Blockset/ Index library. Do not wire up the block yet; first, edit the MATLAB function to create the correct ports and function name.
4. Double-click the **MCode** block and as shown below, click the **Edit M-file** option.

5. Edit the default MATLAB function to include the function name `state_machine` and the input `din` and output `matched`.

6. You can now delete the sample M-code.

7. After you make the edits, save the MATLAB file as `state_machine.m` to the `lab4` folder, and use the **Browse** button ensure that the **MCode** block is referencing the local MCode file.
8. In the MCode Dialog Box, click **OK**.
You will see the **MCode** block assume the new ports and function name.

9. Now connect the **MCode** block to the diagram as shown below:

You are now ready to start coding the state machine. The bubble diagram for this state machine is shown in the following figure. This FSM has five states and is capable of detecting two sequences in succession.
10. Edit the MCode file, state_machine.m, and define the state variable using the Xilinx xl_state data type as shown below. This requires that the variable be declared as a persistent variable. The xl_state function requires two arguments: the initial condition and a fixed-point declaration.

Because you need to count up to 4, you need 3 bits.

```matlab
persistent state, state = xl_state(0,{xlUnsigned, 3, 0});
```

11. Use a switch-case statement to define the FSM states shown. A small sample is provided below to get you started.

   **Note**: you need an otherwise statement as your last case.

```matlab
switch state
    case 0
        if din == 1
            state = 1;
        else
            state = 0;
        end
        matched = 0;
    end
end
```

12. Save the MCode file and run the simulation. The waveform should look like the following figure.

   You should notice two detections of the sequence.

---

**Summary**

In this exercise you learned how to create control logic using M-Code. The final design may be used to create an HDL netlist, in the same manner as designs created using the Xilinx BlockSets. The complete solution to this lab is in the following location:

C:/ug948-design-files/lab4/solution
Lab 5 Introduction

In this lab exercise you will explore the effects of the rate changing blocks available in System Generator including the **Up Sample**, **Down Sample**, **Serial to Parallel** and **Parallel to Serial** blocks.

- **Upsampling** is the process of increasing the sampling rate of a signal.
- **Down sampling** is the process of decreasing the sampling rate of a signal. It is common practice in signal processing systems to change the sample rate of a signal to simplify the hardware or processing tasks.

Objectives

After completing this lab, you will be able to:

- Change the sample rates in a DSP System
- Convert a serial stream of data to a parallel word
- Convert a parallel word of data into a serial stream

Procedure

In this lab exercise you will first learn to understand how the sampling rate impacts the operation of the design by using the Down Sample and Up Sample blocks. In steps 3 and 4 you then use this understanding of the sample rate to create some rate changing designs (serial and parallel converts).

Step 1: Changing Sample Rate: Down Sample

1. Launch the MATLAB program, and change the working directory to:
   
   C:/ug948-design-files/lab5.

2. Open a new Simulink model and save it as lab5.slx in the current working directory.

3. Create the simple diagram as shown in the following figure.

   a. Use the **Counter Limited** block from the **Simulink/Sources** library and set the upper limit of the counter to 10.

   b. Set the quantization of the **Gateway In** block to fix [8 0].
4. Simulate the counter for 10 simulation cycles and observe the results.
5. As shown in the following figure, add a **Down Sample** block from the Xilinx **Blockset/Index** library between the **Gateway In** and **Gateway Out** blocks, then re-simulate the design.

What do you observe?

---

**Step 2: Changing Sample Rate: Up Sample**

6. Replace the **Down Sample** block with an **Up Sample** block and re-simulate the design.

   The System Generator token is going to generate an error that indicates your sample rate is incorrect.

7. Double-click the **System Generator** token and change the **Simulink System Period** to $\frac{1}{2}$ as the message suggests.

8. Re-simulate the design.

9. Add **Sample Time** probes from the Xilinx **Blockset/Index** library before and after the Up Sample block and connect the outputs of the probes to Gateway Out blocks and the Display block from the Simulink/Sinks as shown in the following figure.

   These probes do not add any hardware to the design, but offer a powerful debugging tool for complex multi-rate systems.
10. Re-simulate the design to observe the sample rate in the Display sinks.

In the next two steps, you explore the rate changing effects of using the **Serial to Parallel** and **Parallel to Serial** blocks from the Xilinx **Blockset**.
Step 3: Using Serial-to-Parallel Blocks

1. Open a new blank model, and then create the design shown in the following figure.

![Diagram of Serial-to-Parallel Blocks](image)

2. Set the **Upper limit** on the **Counter Limited** block to 1. This is simply going to generate the sequence 1010101010.

3. Set the output of the Serial to Parallel block to **Unsigned [8 0]**.

The **Serial to Parallel** block imposes a rate change on the system equal to the number of output bits / number of input bits. In this example, you have 8 output bits and 2 input bits so the rate change will be set to 4.
Step 4: Using Parallel-to-Serial Blocks

4. In the Block dialog box, click **OK**, and then add sample rate probes to the input and output of the **Serial to Parallel** block.

5. Re-simulate the design and observe the sample rates.
   
   Input Sample Rate _______
   
   Output Sample Rate _______

6. Change the output quantization of the **Serial to Parallel** block to `fix [16 0]` and re-simulate. What are the sample rates now?
   
   Input Sample Rate _______
   
   Output Sample Rate _______

---

**Step 4: Using Parallel-to-Serial Blocks**

1. Replace the **Serial to Parallel** block with the **Parallel to Serial** block. Leave the output quantization at the default `ufix [1 0]`.

2. Change the sample rate in the **System Generator** token from 1 to ½, then click **OK**.

3. Re-simulate the design and record the input and output sample rates.
   
   Input Sample Rate _______
   
   Output Sample Rate _______

---

**Summary**

In this exercise you learned how to work with the system sample rate to accurately create designs in which the rate of computation changes. The complete solution to this lab is in the following location:

C:/ug948-design-files/lab5/solution.
Lab 6: Using Memories

Lab 6 Introduction

In this lab exercise you learn how to use a Xilinx ROM block to implement a LUT-based operation such as an arcsin using block or distributed RAM. This provides an efficient implementation for trig and math functions with inputs that can be quantized to 10 bits or less.

Objectives

After completing this lab, you will be able to use a Xilinx ROM block to implement a trig or math function such as arcsin.

Procedure

In this exercise you will first examine a standard simulink look-up-table and then implement similar version using Xilinx block RAM.

Step 1: Using ROM for a Look-Up-Table

1. Launch the MATLAB program and change the working directory to:
   C:/ug948-design-files/lab6.
2. As shown in the following figure, open the Simulink executable specification named lab6.slx. Double-click the Lookup Table block to see how the arcsine function is defined.
3. Simulate the counter for 201 simulation cycles and observe the results. The design is using the Simulink X Y Graph to plot the output data as a function of the input data.

![XY Graph](image)

*Note:* This plot is a Simulink representation of the MATLAB arcsine example that is displayed when you type `doc asin` from the MATLAB command line prompt.

```matlab
x = -1:.01:1;
plot(x,asin(x)), grid on
```

4. Add a Xilinx **Gateway In, Gateway Out, System Generator** token, and a **ROM block**, as shown in the following figure:

![System Generator Diagram](image)

5. Double-click the **ROM** block and set the **initialization vector** equal to `asin([-1:.01:1])` and the **depth** to 256.

   This is the same initialization as the Simulink Lookup Table block.
The Xilinx memory blocks are going to require depths to fall on power of 2 boundaries. The MATLAB statement used to initialize the ROM is only going to set 201 locations. The other locations will be uninitialized.

6. Simulate the design for 201 clock cycles.

You get an error indicating an incorrect quantization at the input of the ROM block. You left the quantization of the Gateway In block to the default value of fixed [16 14].

To address a Xilinx memory, the quantization must be ufix with no fractional bits. Because you have a 256 element address space, you are going to need an input quantization of ufix [8 0].

7. Change the gateway and re-simulate.

8. View the waveform in XY Graph1. Notice that it is incorrect.

The reason the waveform does not match is that the quantization of the Gateway In is truncating the fractional bits and the sign bit. It is not as simple as just re-specifying the quantization. The arcsine value for the input -1 is stored at address location zero and the arcsine value for 1 is stored in location 201.

To match the behavior of the MATLAB LUT block, which can accept negative and fractional numbers as inputs, you need to convert the input data to an appropriate RAM address.

9. Add a second From Workspace block to the diagram and configure the block to generate outputs from 0 to 201. This is the simplest approach.
10. Specify the **data** field as follows:

\[1:201;0:1:200]\'

11. Re-simulate. You should see correct results.

---

**Summary**

In this exercise you learned how to implement a memory. The complete solution to this lab is in the following location: `C:/ug948-design-files/lab6/solution`. 
Lab 7: Including a System Generator Model in a Vivado IDE Design

Lab 7 Introduction

Typically, a System Design Engineer has to include hardware design sources from a variety of different formats and put them together into a final system-level design. This lab exercise provides an overview of how you can include a System Generator model within a Vivado® IDE design and combine that model with other RTL sources.

Objectives

After completing this lab, you will be able to integrate designs created with in the Simulink environment with other blocks in a system using the Vivado IDE.

Procedure

In this exercise you will first create a Vivado IDE project for a DSP design using a design file from MATLAB, along with an associated HDL wrapper and constraint file. In step 2, you will review how these sources can be analyzed, modified and synthesized into an FPGA implementation.

Step 1: Create a new Vivado IDE Project with DSP Sources

1. Invoke the Vivado Integrated Design Environment (IDE):
   
   Start > All Programs > Xilinx Design Tools > Vivado 2014.x > Vivado 2014.x

2. Select Create New Project from the Welcome screen.

3. On the first screen of the New Project wizard, click Next.

4. As shown in the following figure, specify the project name as vivado_dsp and the project location as C:/ug948-data-files/lab7/vivado_dsp_source, then click Next.
5. Select the **RTL Project** option, as shown in the following figure. Make sure that the option **Do not specify sources at the time** is unchecked, and then click **Next**.
Step 1: Create a new Vivado IDE Project with DSP Sources

6. As shown in the following figure, click the Add Files button, navigate to folder rtl_sources, select top_level.vhd, click OK, then click Next.

7. In the Add Existing IP dialog box, click the Add Files button, navigate to the dsp_source folder, select rgb2gray.slx and click Next.

8. In the Add Constraints dialog box, select Add Files, and then navigate to the constraints directory.
Step 1: Create a new Vivado IDE Project with DSP Sources

9. As shown in the following figure, add the kc705.xdc file, then click **Next**.

![Add Constraint Files](image1)

10. In the Project Pane, select **Boards** and then select **Kintex-7 KC705 Evaluation Platform** as shown in the following figure:

![New Project](image2)

11. Click **Next** and then click **Finish**.
Step 2: Explore the DSP Module from the Vivado IDE Cockpit

The previous steps setup a project. The following figure shows the hierarchy of the project in Vivado IDE Hierarchy Browser:

---

Step 2: Explore the DSP Module from the Vivado IDE Cockpit

1. Open the DSP Design created in MATLAB by double clicking on the file `rgd2gray.slx` file in the Design Source top-level model as shown in the following figure.

This opens the System Generator model in MATLAB/Simulink.
Step 2: Explore the DSP Module from the Vivado IDE Cockpit

You can make modifications to this Simulink model if necessary.

2. Exit MATLAB and transfer control back to the Vivado IDE.

3. Double-click on the file `top_level.vhd` inside the Design Sources as shown in the following figure.

This file instances the DSP module created using System Generator for DSP into an RTL design. To perform this instantiation, you must include a component declaration (lines 92-101). The name of the component should be the same as the name of the DSP module, as indicated in the following code snippet.

```markdown
component rgb2gray is
  port (  
    blue: in std_logic_vector(7 downto 0);  
    clk: in std_logic; -- clock period = 10.0 ns (100.0 Mhz) 
    green: in std_logic_vector(7 downto 0);  
    red: in std_logic_vector(7 downto 0);  
    rst: in std_logic;  
    grey_output: out std_logic_vector(31 downto 0)  
  );
```
end component;

Lines 118-126 instantiate the module:

```vhdl
-- System Generator Design #1
u_rgb2gray :rgb2gray  
port map (  
  blue   => blue,  
  clk    => clk,  
  green  => green,  
  red    => red,  
  rst    => rst,  
  grey_output  => grey_output
);  
```

4. Using the Vivado **Flow Navigator** (left-hand side of the GUI), click **Open Elaborated Design** in the **RTL Analysis** group.

This action recognizes that the System Generator model has not yet been generated, and automatically triggers generation. All files generated by the System Generator model are included in the synthesis and implementation steps automatically. You will see the MATLAB/Simulink tool that is on your path launched and the System Generator model is generated and elaborated as shown in the schematic below.

You can now invoke other downstream tool steps, such as **Run Synthesis** and **Run Implementation**, just like any other RTL design in the Vivado IDE.
Summary

In this exercise you learned how to incorporate a MATLAB design file into an RTL project and learned how a top-level RTL wrapper is required, in which to instantiate the MATLAB design. You have seen how these sources can be viewed and modified from within the Vivado IDE and how synthesis is performed.

This exercise showed you how to import a MATLAB design into the Vivado IDE and use the Vivado IDE to synthesize the design into an FPGA implementation.
Lab 8: Importing C/C++ Source Files into System Generator by Leveraging Integration with Vivado HLS

Lab 8 Introduction

The System Edition of the Vivado® Design Environment includes the feature called Vivado HLS which has the ability to transform C/C++ design sources into RTL. System Generator has a block called Vivado HLS located in the Control and Index libraries that enables you to bring in C/C++ source files into a System Generator model.

Objectives

After completing this lab, you will be able to incorporate a design, synthesized from C, C++ or SystemC using Vivado HLS, as a block into your MATLAB design.

Procedure

In this exercise you will first synthesize a C file using Vivado HLS Vivado IDE project for a DSP design using a design file from MATLAB, along with an associated HDL wrapper and constraint file. In step 2, you incorporate the output from Vivado HLS into MATLAB and use the rich simulation features of MATLAB to verify that the C algorithm correctly filters an image.
Step 1: Creating a System Generator Package from Vivado HLS

1. Invoke Vivado HLS: Start > All Programs > Xilinx Design Tools > Vivado 2014.x > Vivado HLS > Vivado HLS 2014.x.

2. Select Open Project in the welcome screen and navigate to the Vivado HLS project directory C:\ug948-design-files\lab8\hls_project as shown in the following figure.

3. Click OK to open the project.
Step 1: Creating a System Generator Package from Vivado HLS

4. **Expand** the **Source folder** in the Explorer pane (left-hand-side) and **double-click** on file MedianFilter.cpp to view the contents of the C file as shown in the following figure.

   ![MedianFilter.cpp code snippet](image)

   This file implements a 2-Dimensional median filter on 3x3 window size.

5. Synthesize the source file by right-clicking on **solution1** and selecting **C Synthesis > Active Solution** as shown in the following figure.

   ![Synthesis options](image)

   After the synthesis is complete you can package the source for use in System Generator for DSP.

6. Right click on **solution1** and select **Export RTL**.
7. Set **Format Selection** to **System Generator for DSP** as shown in the following figure and click **OK**.

![Export RTL Dialog](image)

8. When the Export RTL process completes and Vivado HLS displays the message “@I [LIC-101] Checked in feature [VIVADO_HLS]” **Exit** Vivado HLS.
Step 2: Including a Vivado HLS Package in a System Generator Design

1. Launch System Generator and open the `sysgenHLSMedianFilter.slx` file in the `lab8` folder. This should open the model as shown in the following figure.

2. Add a Vivado HLS block by right-clicking anywhere on the canvas workspace.
3. Select Xilinx **BlockAdd** and scroll down to the Vivado HLS block as shown in the following figure.

4. **Double-click** on the **Vivado HLS** block to add a block called Vivado HLS as shown in the following figure.
5. **Double-click** the Vivado HLS block and use the **Browse** button to select the solution created by Vivado HLS `C:\ug948-design-files\lab8\hls_project\solution1` as shown in the following figure, then click **OK**.

6. Connect the input and output ports of the block as shown in the following figure.
8. Navigate into the **Noise Image** sub-system and **double-click** on the **Image From File** block jena.png to open the Source Block Parameters dialog box.

9. Use the Browse button to ensure the file name correctly point to file lena.png as shown below.

10. Press OK to exit the Source Block Parameters dialog box.

11. **Simulate** the design and verify the image is filtered, as shown in the following figures.
Summary

In this lab exercise you were able to take a filter written in C, synthesize it with Vivado HLS and incorporate the design into MATLAB. This process allows you to use any C, C++ or SystemC design and create a custom block for use in your designs.

This exercise showed you how to import a MATLAB design from Vivado HLS and use the design inside MATLAB.
Lab 9 Introduction

System Generator has a feature that lets you export a design as an IP module into the Vivado® IP Catalog. From the IP Catalog, you can then import the System Generator design as IP into a Vivado IP integrator design. For example, you can create a data-path module in System Generator and then add the data-path to a larger processor-centric design in IP integrator.

There are two IP creation modes in System Generator and a combination of these modes is supported as well:

- **Port/Pin Interface Packaging.** When used in this manner, the port names on the IP match exactly the Pin names of the Gateway In and Gateway Out blocks used in System Generator.
- **Port-Name-Based Interface Inference.** In this mode of operation, multiple Gateway In and Gateway Out blocks are grouped into a single port on the IP.
Step 1: Packaging IP with Port/Pin Interfaces

Do the following:

1. Invoke System Generator, navigate to the lab9/raw_interface design data folder and open the file named raw_interfaces.slx.

2. Double-click the System Generator token and verify that the IP Catalog compilation target is selected.
When you select the IP Catalog compilation target and generate the design, the following output products are placed in the `<target_directory>/ip_catalog` directory:

- `<Vendor>_<Library>_<design_name>_<Version>.zip`: This is the IP created from System Generator.
- `<design_name>.xpr`: This is an example project file that instantiates the IP that is created. It contains an IP integrator block diagram with an example instantiation of the IP.

3. Verify that the **Create testbench** option is checked.
   
   This will ensure a testbench is created and the example project contains an RTL Testbench instantiating the IP.

4. Click the **Settings** button and configure the **IP Catalog:Settings** as shown in the following figure.

5. Ensure the option **Auto Infer Interfaces** is **unselected**.
6. Click **OK**, to exit the IP Catalog:Settings

7. Click **Generate** on the System Generator token.

   Instructions on using the IP are displayed in the Compilation status dialog box, as shown below.
8. Navigate to the directory C:\ug948-design-files\lab9\raw_interface\netlist\ip_catalog and double-click project file raw_interface.xpr to open the example design project in the Vivado IDE.

9. In the Vivado Flow Navigator pane, select Run Simulation and then Run Behavioral Simulation. This demonstrates the IP can be verified with RTL simulation inside the Vivado IDE.

10. In the Vivado Flow Navigator pane, select Open Block Design to open the example IP integrator project.

11. Click on the Validate Design toolbar button to confirm the IP works within IP integrator.

12. Exit the Vivado IDE.

13. Close the raw_interfaces design in MATLAB.
Step 2: Packaging the IP with Port-Name Interface Inference

1. Navigate to the lab9/gain_control folder and open the System Generator design named gain_control.slx.

A gain block is implemented using a multiplier with one input controlled using an AXI4-Lite interface and the other input is part of the data path as shown in the following figure.

![Gain Control Block](image)

Inside the AXI4-Lite Interface subsystem is an MCode block that models a single AXI4-Lite Interface. The MCode block is attached to Gateway In and Gateway Out blocks, as shown in the following figure.

![AXI4-Lite Interface Block](image)
As you can see, all gateways are named with a the same base name, _s_axi_ and suffixes that match the signals in an AXI4-Lite interface. This naming convention ensures that the pins may be grouped into a single AXI4-Lite interface port named _s_axi_.

2. Double-click the **System Generator** token.

3. Open **Settings** and verify the **Auto Infer Interface** setting is checked.

This option will infer ports based on their pin names. The following ports are inferred from pin names provided the following are both true:

- The pins must have a common prefix name.
- The suffix must match the name shown in the following table.

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>Gateway In (Restriction)</th>
<th>Gateway Out (Restriction)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AXI4-Lite Slave</td>
<td>awaddr</td>
<td>awready (1 Bit Wide)</td>
</tr>
</tbody>
</table>
Step 2: Packaging the IP with Port-Name Interface Inference

<table>
<thead>
<tr>
<th>Interface Name</th>
<th>Gateway In (Restriction)</th>
<th>Gateway Out (Restriction)</th>
</tr>
</thead>
<tbody>
<tr>
<td>awvalid(1 Bit)</td>
<td>wdata</td>
<td>wready (1 Bit Wide)</td>
</tr>
<tr>
<td>wvalid(1 Bit)</td>
<td>bready(1 Bit)</td>
<td>bresp (2 Bits Wide)</td>
</tr>
<tr>
<td>AXI4-Stream Slave</td>
<td>tdata</td>
<td>tready</td>
</tr>
<tr>
<td>AXI4-Stream Master</td>
<td>tready(1 Bit)</td>
<td>tdata</td>
</tr>
<tr>
<td>Reset (full name)</td>
<td>aresetn</td>
<td></td>
</tr>
<tr>
<td>Clock</td>
<td>clk</td>
<td></td>
</tr>
</tbody>
</table>

4. Click OK in the IP Packager settings menu.
5. Click Generate to create the IP.
6. Navigate to C:\ug948-design-files\lab9\gain_control\netlist\ip_catalog and open the Vivado project gain_control.xpr.
7. In the Vivado Flow Navigator pane, select Open Block Design and confirm port s_axi on gain_control is an AXI4-Lite interface.
Step 3: Programming an AXI4 Interface from Software

The first step to writing software is to export this design to the Software Development Kit (SDK).

Also in the Address editor pane, you must verify that the right address range is allocated to the System Generator AXI-Lite interface. As you can see, 4K address space is reserved because while exporting from System Generator awaddr and araddr are 12 bits wide.

1. Right-click the IP integrator design and select Export Hardware to SDK as shown in the following figure, and ensure that SDK launches.

In SDK, you need to create a default Application project for the given exported hardware. This creates a test application for all peripherals.

In this case, the only major peripheral is the System Generator IP.

2. Open the xparameters.h file included in the Peripheral Test application and see the following lines:

```c
/* Definitions for peripheral GAIN_CONTROL_1 */
#define XPAR_GAIN_CONTROL_1_BASEADDR 0x44A00000
#define XPAR_GAIN_CONTROL_1_HIGHADDR 0x44A000FF
```

This can be co-related with the address map in the IP integrator design Environment and is essentially the software interface to the System Generator peripheral.
3. Modify the Test Peripheral application as shown in the following figure.

```c
#include <stdio.h>
#include "xparameters.h"
#include "xil_cache.h"
#include "xil_io.h"

int main()
{
    Xil_ICacheEnable();
    Xil_DCacheEnable();
    //
    int i;
    for (i = 0; i < 1; i++) {
        Xil_Out32(XPAR_GAIN_CONTROL_1_BASEADDR,i);
    }
    //
    Xil_DCacheDisable();
    Xil_ICacheDisable();
    return 0;
}
```

4. **Save** and **Exit**. This creates an ELF file executable.

5. Locate the ELF file and associate as shown in the following two figures. The ELF file should be located in `gain_control.sdk/SDK/SDK_Export/test/Debug/*.elf`. 
Step 3: Programming an AXI4 Interface from Software

Model-Based DSP Design using System Generator [www.xilinx.com]

UG948 (v2014.3) October 28, 2014
Lab 10: Packaging a Synthesized AXI4-Lite Interface

Lab 10 Introduction

In this lab, you learn how to package a System Generator design with a synthesized AXI4-Lite interface such that a processor can control the value of registers within the interface ports.

Requirements

To complete this lab, you must have the following tools installed:

- Vivado Design Environment System Edition with SDK (v2014.3 or greater)
- MicroZed Hardware Board or any other Xilinx supported Board

*Note: You can verify that the SDK tool is installed by navigating to the Xilinx installation directory and observing the SDK folder.*

If the SDK folder is not present, download the file named Vivado 2014.3: Standalone SDK Single File Download Image from the [www.xilinx.com](http://www.xilinx.com) download area and install the file.
Step 1

1. Invoke System Generator for DSP from the Windows Start menu, navigate to the Lab 10 folder, and then open the rgb2gray.slx model in Simulink. The model is shown in the following figure:

   ![Rgb2Gray Model Diagram]

2. Simulate the model to ensure that the results are correct. In this case, the output should be 17.

3. Do the following to map each Gateway In block to an AXI4-Lite Slave Interface so it can be controlled by a microprocessor:
   a. Open (double-click) the Gateway In block named Red.
   b. Select the Implementation tab.
   c. As shown in the following figure, set the control called Interface to AXI4-Lite.
4. Repeat the process for the Gateway In blocks named Green and Blue, and the Gateway Out block named Luminance.

You are now ready to package the design.

5. Double-click the System Generator token and note that the Compilation Target is set to IP Catalog. Additionally, for this lab, the device is set to ZynQ, the same device located on the MicroZed board.

6. Change the target directory to ./netlist_mzed, and click the Generate button on the System Generator token.

   This action packages the System Generator design as an IP module in the subdirectory called ./netlist_mzed/ip. In addition, an example Vivado IDE project file named rgb2gray.xpr is created and configured to the MicroZed board.

   **Note:** The board is inferred from the part settings on the System Generator token located at ./netlist_mzed/ip_catalog/rgb2gray.xpr

7. Double-click the rgb2gray.xpr project file to open the Vivado IDE project. The sources view looks like the following figure.
8. Double-click the `rgb2gray_bd_i` block design instance to open the IP integrator Block Diagram, as shown in the following figure:

9. As shown in the following figure, move to the Flow Navigator and select **IP Integrator > Generator Block Design**:

10. Right-click and press the **Generate** button on the dialog box, or, in the **Flow Navigator**, click **Generate Bitstream**.
11. As shown in the following figure, bring the Vivado IDE Sources pane to the front, right-click the `rgb2gray_bd_i` Block Design instance and select **Export Hardware to SDK**.

12. As shown in the following figure, select the Launch SDK option on the pop-up dialog box, then click **OK**.

Next, you are going to add the `rgb2gray_bd` IP to SDK, so SDK can find the appropriate drivers.
13. As shown in the following figure, in the SDK GUI, select **Xilinx Tools > Repositories** and add the full path to the directory containing the System Generator Model (`.../netlist_mzed/ip`) to the local repositories in SDK.

![Repositories in SDK](image)

14. Create a new Application Peripheral Tests Project by selecting **File>New>Application Project**.

15. Name the application `testsygen` as shown in the following figure, and click **Next**.

![New Application Project](image)

16. Select **Peripheral Tests** as shown in the following figure, and then click **Finish**.

![Peripheral Tests](image)
17. Modify the test program to exercise the System Generator Peripheral as shown below:

Include the driver header file. This is derived from the top-level name of the RTL created from System Generator for DSP design. This should be similar to the name of the Simulink model with lower-case characters as well as some mangling to ensure the name does not conflict with a reserved VHDL or Verilog words.

```c
#include "rgb2gray.h"
```

In the `main()` code, add the following lines to help with modifying frequency:

```c
{ /*
   * Writing to Red Blue and Green Ports
   */
   rgb2gray_Instance;
   rgb2gray_Initialize(&Instance, XPAR_RGB2GRAY_0_DEVICE_ID);
   rgb2gray_red_write(&Instance, 1);
   rgb2gray_blue_write(&Instance, 1);
   rgb2gray_green_write(&Instance, 1);
   /*
   * Writing to Red Blue and Green Ports
   */
   u8 luminance;
   luminance = rgb2gray_luminance_read(&Instance);

   rgb2gray_red_write(&Instance, 2);
   rgb2gray_blue_write(&Instance, 2);
   rgb2gray_green_write(&Instance, 2);
}
```

`XPAR_RGB2GRAY_0_DEVICE_ID` is defined in `xparameters.h` that was used to create the board support package.
18. Setup SDK for Debug. In the Project Explorer, right-click **testsysgen** and select **Debug As > Debug Configurations**.
19. As shown in the following figure, select **Reset entire system and Program**:

![Image of the System Debugger settings](image)

20. Select **Debug** to launch the Debugger and step through the code.

21. Verify that the value of **Luminance** is 17 and then 18.

---

**Conclusion**

A System Generator design can be packaged to include an auto-generated AXI4-Lite interface at the ports.
Lab 11 Introduction

In this lab exercise you will import an RTL design into System Generator for DSP as a black box.

- A black box allows the design to be imported into System Generator for DSP without even though the description is in Hardware Description Language (HDL) format.

Objectives

After completing this lab, you will be able to:

- Import an RTL HDL description into System Generator for DSP
- Configure the black box to ensure the design can be successfully simulated

Step 1: Import RTL as a Black Box

1. Invoke System Generator and from the MATLAB console, change the directory to: lab6/example1.

   The following files are located in this directory:
   - black_box_ex1.slx - A Simulink model containing a black box example.
   - mac.vhd - Multiply and add component used to build the transpose FIR filter.
   - transpose_fir.vhd - Top-level VHDL for a transpose form FIR filter. This file is the VHDL that is associated with the black box.

2. Open the file black_box_ex1.slx.

3. Open the subsystem named Down Converter, and then the subsystem named Transpose FIR Filter Black Box. At this point, the subsystem contains two input ports and one output port.
Step 1: Import RTL as a Black Box

The following figure shows the subsystem that will contain the black box:

4. Right-click the design canvas, select Xilinx **BlockAdd**, and add a black box block to this subsystem.

   A browser window opens that lists the VHDL source files that can be associated with the black box. From this window, select the top-level VHDL file *transpose_fir.vhd*. This is illustrated in the following figure:

   The VHD file opens in an Editor for modifications. Close the Editor for now.

5. Wire the ports of the black box to the corresponding subsystem ports and save the design.
6. Double click the black box block. The dialog box is shown in the following figure:

The following are the fields in the dialog box:

- **Block configuration M-function** - This specifies the name of the configuration M-function for the black box. In this example, the field contains the name of the function that was generated by the Configuration Wizard. By default, the black box uses the function the wizard produces. You can, however, substitute one you produce yourself.

- **Simulation mode** - There are three simulation modes:
  - **Inactive** - When the mode is Inactive, the black box participates in the simulation by ignoring its inputs and producing zeros. This setting is typically used when a separate simulation model is available for the black box, and the model is wired in parallel with the black box using a simulation multiplexer.
  - **Vivado Simulator** - When the mode is Vivado Simulator, simulation results for the black box are produced using co-simulation on the HDL associated to the black box.
  - **External co-simulator** - When the mode is External co-simulator, it is necessary to add a ModelSim HDL co-simulation block to the design, and to specify the name of the ModelSim block in the field labeled HDL co-simulator to use. In this mode, the black box is simulated using HDL co-simulation.
7. To continue this exercise, leave the parameters set as they currently are, and close the dialog box.

8. Move to the top-level of the design and run the simulation by clicking the **Simulation Run** button (green button) and then double-click the **Scope** block.

9. Notice the black box output shown in the Output Signal scope is zero. This is expected because the black box is configured to be **Inactive** during simulation.

![](image1.png)

10. From the Simulink menu, select **Display > Signals & Ports > Port Data Types** to display the port types for the black box.

11. Compile the model (**Ctrl-d**) to ensure the port data types are up to date.

   Notice that the black box port output type is **UFix_26_0**. This means it is unsigned, 26-bits wide and has a binary point 0 positions to the left of the least significant bit.

12. Open the configuration M-function `transpose_fir_config.m` and change the output type from **UFix_26_0** to **Fix_26_12**. The modified line (line 26) should read:

   ```
   out_port.setType('Fix_26_12');
   ```

13. Continue to edit the configuration **M-function** to associate an additional HDL file with the black box.

14. Locate the line 67: `this_block.addFile('transpose_fir.vhd');`

15. Immediately above this line, add the following: `this_block.addFile('mac.vhd');`

16. Save the changes to the configuration M-function and close the file.

17. Click the design canvas and recompile the model (**Ctrl-d**).
18. Your **Transpose FIR Filter Black Box** subsystem should display as follows:

19. From the Black Box block parameter dialog box, change the Simulation mode field from **Inactive** to **Vivado Simulator** and then click **OK**.

20. Move to the top-level of the design and run the simulation.

21. Examine the scope output after the simulation has completed.

   When the Simulation Mode was **Inactive**, the Output Signal scope displayed constant zero.

   Notice the waveform is no longer zero. Instead, the Output Signal shows a sine wave as the results from the Vivado Simulation.

22. Right click the **Output Signal** canvas and select **Autoscale**. You should see a display similar to that shown below.
PLEASE READ: Important Legal Notices

Notice of Disclaimer

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications:

©Copyright 2014 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Zynq, UltraScale and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.