



Xilinx System Generator v2.1 for Simulink

Setting up the Tools

*Introduction to Simulink and the
Xilinx Gateway*

Software Overview

Black Box

Multiplier Accumulator

The Costas Loop

Filter

Image Enhancement Example

Combination Lock State Machine

Introductory Tutorials

About the Tutorials

This set of tutorials is a beginner's guide for designers unfamiliar with the features of the Xilinx System Generator software, v2.1 and later. These tutorials show the features and capabilities of the System Generator tools, using simple designs and examples.

Tutorials Contents

This set of tutorials contains the following chapters.

- Chapter 1, *Setting up the Tools*, gives instructions for installing the software and lists software dependencies.
- Chapter 2, *Introduction to Simulink and the Xilinx Gateway*, provides a simple introduction to Simulink which will ensure correct installation of the tools, and provides basic information on the gateways for using the Xilinx Blockset.
- Chapter 3, *Software Overview*, walks you through a simple integrator sample model and exhibits some of the behavior of the Xilinx Blockset. This chapter also gives a very basic flow through the downstream Xilinx tools, using the Foundation ISE Project Navigator.
- Chapter 4, *Black Box*, shows you how the System Generator lets you create your own blocks and have them included in the generated code.
- Chapter 5, *Multiplier Accumulator*, gives directions and some hints for creating a new model using the Xilinx Blockset.
- Chapter 6, *The Costas Loop*, is another example project provided with the System Generator. If time allows, you may want to run the Costas Loop simulations and look at a larger design that uses elements from the Xilinx Blockset as well as Simulink blocks.
- Chapter 7, *Filter*, demonstrates the effects of applying a filter to a random signal. You will run MATLAB console commands to set up your design to automatically preload coefficients and use the MATLAB "SPTool" utility to define filter coefficients.
- Chapter 8, *Image Enhancement Example*, demonstrates an image run through a filter, also showing the benefits of simulating the entire system, including the Xilinx blocks, in Simulink.
- Chapter 9, *Combination Lock State Machine*, demonstrates how to design a state machine for a combination lock using the Xilinx Mealy State Machine block.

Additional Resources

For additional information, see the *System Generator Reference Guide* and *Quickstart Guide*, supplied with your installation of the System Generator. These files are installed, by default, into `$MATLAB/toolbox/xilinx/sysgen/help`.

Contents

Chapter 1	Setting up the Tools.....	4
	Software Dependencies	
	Using the System Generator installer	
	Compiling IP libraries for simulation	
Chapter 2	Introduction to Simulink and the Xilinx Gateways.....	8
Chapter 3	Software Overview.....	16
	Simulink Design Flow	
	Customizing the Xilinx Blockset Elements in a Design	
	Simulation	
	System Generator Block	
	Files Produced by System Generator Code Generation	
	Testbench Generation	
	Implementation within the Xilinx Design Environment	
Chapter 4	Black Boxes.....	30
Chapter 5	Multiplier Accumulator.....	32
	Incomplete Design	
	Complete the Multiplier/Accumulator	
Chapter 6	The Costas Loop.....	36
	Design Overview	
Chapter 7	Filter.....	39
	<i>SPTool</i> Utility	
	Design Overview	
Chapter 8	Image Enhancement.....	45
	Design Overview	
	Process the Design in Simulink	
	Generate VHDL and Testbench Vectors	
	Simulation and Implementation	
Chapter 9	Combination Lock State Machine.....	52
	State Machine Library	
	Design Overview	
	Implementation	
	Simulation	

Setting Up the Tools

This chapter describes the software dependencies, setup, and installation process of the Xilinx System Generator.

Software Dependencies

This section describes the product dependencies for the System Generator. You must have the following software installed on your computer before you install the System Generator.

- R12 or R12.1 of The MathWorks tools, including:
 - ◆ R12: MATLAB v6.0 and Simulink v4.0
 - ◆ R12.1: MATLAB v6.1 and Simulink v4.1
- Xilinx ISE v4.1i software, including
 - ◆ Xilinx CORE Generator (comes standard with Foundation and Alliance ISE software tools)
 - ◆ Software Service Pack 1
 - ◆ 4.x IP Update #1
 - ◆ The environment variable XILINX must be set and must point to your Xilinx ISE 4.1i software installation directory.

The correct Service Pack and IP Update may be downloaded from the appropriate sections of the 4.1i Software Updates Center:

http://support.xilinx.com/support/software/install_info.htm

The screenshot shows the Xilinx Support website interface. At the top, there's a navigation bar with the Xilinx logo and 'SUPPORT.XILINX.COM'. Below that, a menu contains links for HOME, PRODUCTS, SUPPORT, EDUCATION, BUY ONLINE, CONTACT, and SEARCH. A secondary menu lists Troubleshoot, Hardware, Software, Download, Documentation, Design, and Services. The main heading is '4.1i Software Updates Center', dated 10/22/2001. The content lists several updates:

- Service Packs - Service Pack 2 is now available.** Service Packs are cumulative for 4.1i Software and contain the latest updates for all Xilinx software tools except CORE Generator, IP Updates, and System Generator which must be downloaded and installed separately.
 - If you are already registered for Service Pack downloads:
 - [Download PC Service Pack 2](#)
 - [Download Solaris Service Pack 2](#)
 - [Download HP Service Pack 2](#)
 - [I need to Register for Service Pack Downloads](#)
 - Service Pack 3 is scheduled for release December 12, 2001.**
- Unisim or Simprim Libraries** may need to be recompiled after a Service Pack. See [Answer 12628](#) for more information on compiling simulation libraries.
- Intellectual Property: IP Updates - IP Update #1 is now available.** IP Updates include new and updated IP, as well as software updates to the Xilinx CORE Generator.
 - [Download 4.1i IP Update #1](#)
 - MXE users be sure to read the item below.**
 - IP Update #2 is scheduled for Release Q1 2002.**
- MXE Users** - When upgrading your IP, you will also need to download the latest pre-compiled ModelSim XE Libraries in order to simulate any new or updated Core in the 4.1i IP Update.
 - New MXE libraries for IP Update #1 are now available .**
 - [Download VHDL E-IP #1 MXE Libraries.](#)

Figure 1-1 Service Packs and Updates on Xilinx Web Site

To simulate, synthesize, and implement the VHDL and cores generated from the System Generator, you must also have

- a VHDL (behavioral) simulator, such as
 - ◆ MXE (Modelsim Xilinx Edition) 4.1i, available with the Xilinx Foundation ISE 4.1i software tools
 - ◆ ModelSim PE or SE v5.5 or similar version from Model Technology
- a synthesis compiler, such as
 - ◆ XST (Xilinx Synthesis Technology), available in the Xilinx Foundation ISE 4.1i software tools
 - ◆ Synplicity: Synplify Pro v6.2.4 or v7.0.1
 - ◆ Exemplar: LeonardoSpectrum v2000.1b
- the Xilinx implementation tools, available in the Xilinx Foundation or Alliance ISE 4.1i software tools

Software Download

The System Generator v2.1 is available only via download from a Xilinx web page. You may purchase, register, and download the System Generator software from the

site at: <http://www.xilinx.com/products/software/sysgen.htm>

On this web page, you will see the following sequence of steps:

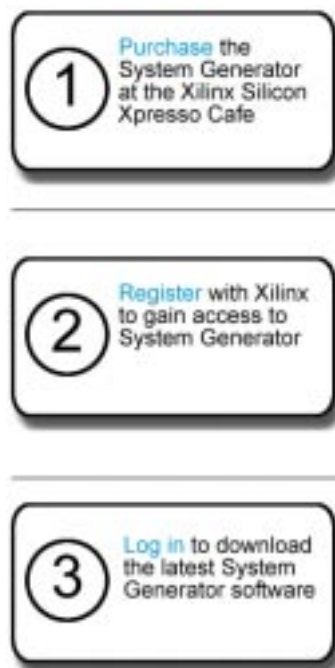


Figure 1-2 Interface for Downloading System Generator

After you have purchased and registered your copy of the System Generator, you will be given a user ID and password. These will allow you to log in and download the System Generator software from Step 3.

Using the System Generator installer

The System Generator installer is now contained in a single MATLAB file: `setup.dll`.

Download `SysgenInstall_v2_1.exe` from the Xilinx web site and execute it. This extracts `setup.dll` and `README.txt` to a temporary directory. Since `setup.dll` is a MATLAB file, you will need to install the software from within MATLAB. Open the MATLAB console, then change directories (`cd`) to the temporary directory where you extracted `setup.dll`. Type:

```
>> setup
```

at the MATLAB console prompt. This will launch the System Generator installer.

Uninstalling previous System Generator directories

If you have previously installed the System Generator tools, the installer will ask if you wish to install System Generator v2.1 to the same location. If so, it will warn you that your old copy will be removed. If you have opened any System Generator designs in your current MATLAB session, you must close and re-open MATLAB before uninstalling can proceed.

Note that the System Generator will remove everything in your previously installed System Generator directory and subdirectories. If you have added any files to the

installed System Generator area, they will be removed. We suggest that you back up your System Generator designs into another directory, such as the `$MATLAB/work` directory.

If you wish to uninstall System Generator v2.1 or previous versions by hand, you may manually remove the entire directory, starting at the top level of the System Generator installed area. This is located by default at `$MATLAB/toolbox/xilinx`.

Recommended Documentation

The *System Generator Reference Guide* is included with your installation and contains a comprehensive description of System Generator blocks, interfaces, and design methodology.

To use the System Generator project flow from design through FPGA implementation, you should be familiar with Xilinx and The MathWorks tools. It is recommended that you refer to the manuals *Using Simulink* from The MathWorks, and *ISE 4.1i User's Guide* from Xilinx. These manuals explain the Simulink environment, as well as the Xilinx implementation tools.

Compiling IP libraries for simulation

You must compile your IP (cores) libraries with ModelSim before you can simulate.

ModelSim (PE or EE/SE)

To compile your IP with ModelSim (PE or EE/SE) you will need to download a TCL/TK script from the Xilinx web site, and run it to compile these libraries:

- Xilinx Simprim
- Unisim
- XilinxCoreLib

Xilinx supplies two sets of instructions for compiling your IP libraries using TCL/TK scripts. The instructions can be found at the following locations:

<http://support.xilinx.com/techdocs/2561.htm>

<http://support.xilinx.com/techdocs/8066.htm>

MXE libraries

If you plan to use ModelSim XE (Xilinx Edition), download the MXE pre-compiled libraries from the Xilinx web site. You may find the latest libraries at:

http://support.xilinx.com/support/software/install_info.htm

Unzip these MXE libraries into your MXE installed directory (usually `$MXE/xilinx/vhdl/xilinxcorelib`). This is the location where MXE expects to find your Xilinx compiled libraries, so you do not need to make any changes to your `modelsim.ini` file. This file should point to the correct installed location.

Introduction to Simulink and Xilinx Gateway Blocks

The purpose of this chapter is to introduce Simulink and the Xilinx Gateway blocks.

Introduction to Simulink

Simulink, which runs in MATLAB, is an interactive tool for modeling, simulating, and analyzing dynamical systems. The Xilinx System Generator, a high-performance design tool, runs as part of Simulink. The System Generator elements bundled as the Xilinx Blockset, appear in the Simulink library browser.

This section provides steps to implement a sample model using Simulink blocks.

1. Open the MATLAB command window by double-clicking on the MATLAB icon on your desktop, or by launching it from the Start menu on your PC.
2. You may navigate to product directories by typing a `cd` command in the MATLAB command window. Type `ls` to see directory contents. Many UNIX shell commands work from the MATLAB command window. `cd` to the examples directory installed with the System Generator. (This is generally `$MATLAB/toolbox/xilinx/sysgen/examples`.)
3. Launch Simulink by typing `simulink` at the MATLAB command prompt, or bring up the Simulink library browser by clicking on the corresponding button on the MATLAB toolbar.

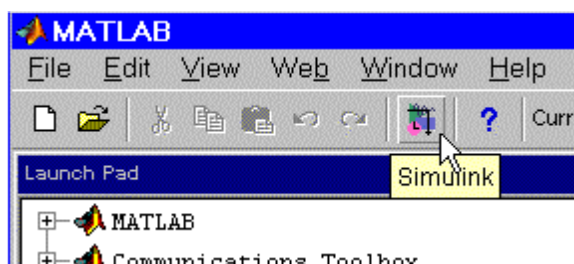


Figure 2-1 Simulink button, available on MATLAB console toolbar

4. Look at the blocks available in the Simulink library browser. The following elements should appear, among others:
 - ◆ Simulink (sources and sinks)
 - ◆ DSP Blockset
 - ◆ Xilinx Blockset

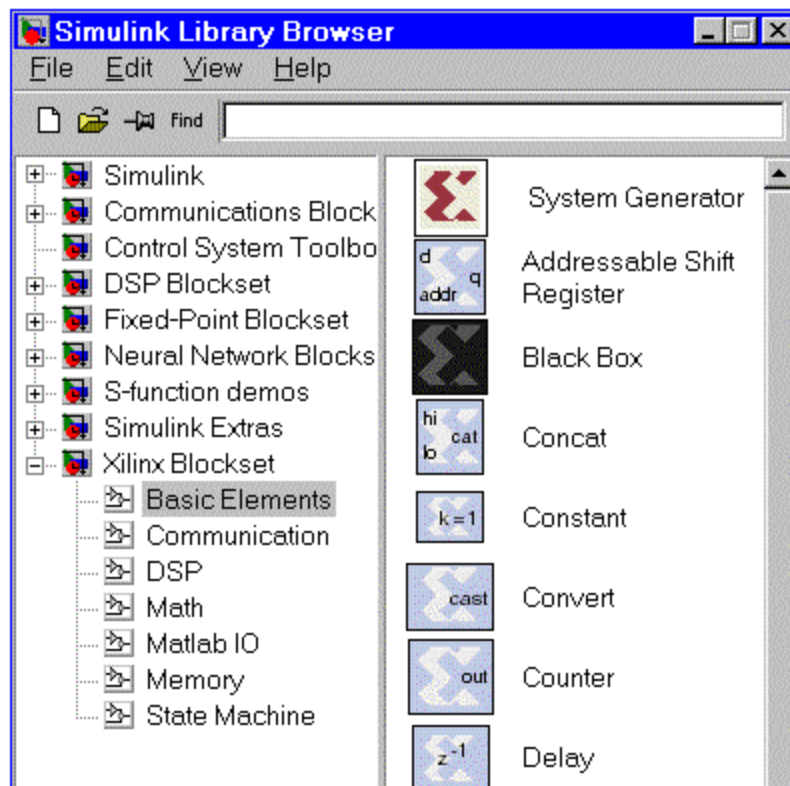


Figure 2-2 Simulink Library Browser window, showing Xilinx Blockset

5. Right-mouse-click on any block in the library browser and choose help from the MATLAB menu. This brings up details about the block. This also works on the Xilinx Blockset elements.

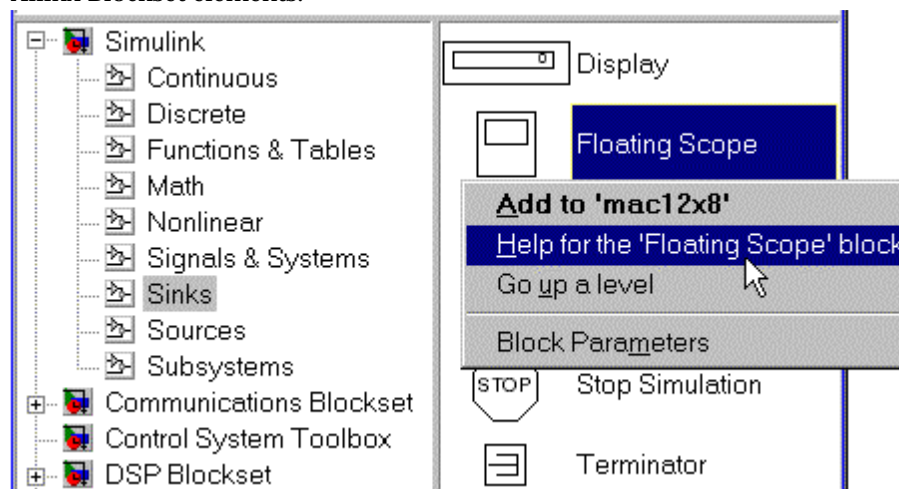


Figure 2-3 Opening a Help window on any block

6. Create a blank sheet "new model" using the button on the Simulink library browser.

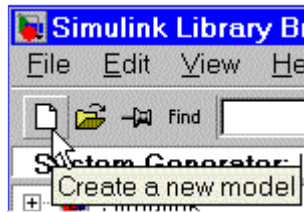


Figure 2-4 Create a new Simulink model via button on Simulink Browser window

7. Add the following 2 blocks to your project sheet. From Simulink Sources, add a sine wave. From Simulink Sinks, add a scope block. Drag and drop the blocks from the Simulink Library Browser into your new model. Draw a wire from the sine wave to the scope block.

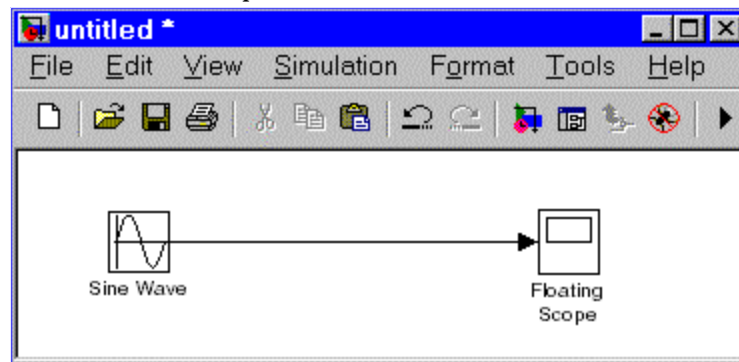


Figure 2-5 Sine wave block wired to Scope output block

8. Double click on the sine wave block. A Block parameters dialog box will open. Change the frequency to **pi/150**.



Figure 2-6 Frequency option on the Sine Wave block parameters dialog

9. From your project sheet, pull down the **Format** menu, and select **port data types**. Now, on your Simulink sheet, you can see that the signal is double precision.
10. From your project sheet, pull down the **Simulation** menu and select **Simulation parameters**. From the **Simulation Parameters** dialog box, change the stop time to **inf**. This will allow your simulation to run to infinity (until you manually stop the simulation).

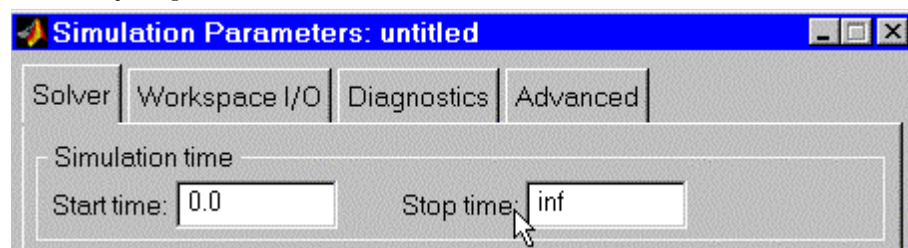


Figure 2-7 Modifying simulation parameters

11. Double click on the scope block. The scope display will open. Click on the Scope Parameters button. In the Scope Parameters box, set the time range to **500**. This is the range that will now be displayed in the scope.

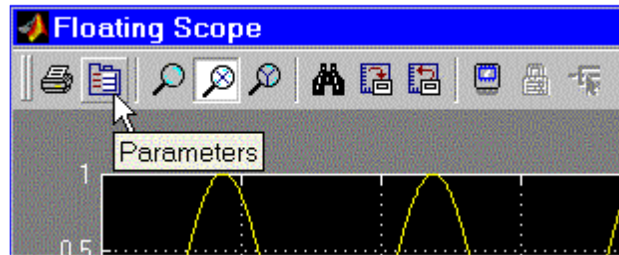
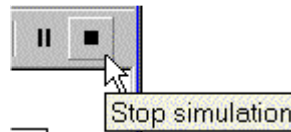
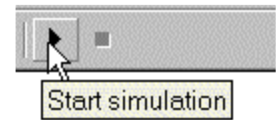


Figure 2-8 Modifying the Scope display parameters

12. Run the simulation. From your Simulink project sheet, click on the Start simulation button (or you can pull down the Simulation menu and select start).
13. On the scope display, click the autoscale button so the output will fit into the scope. The autoscale button looks like a pair of binoculars.
14. Look at the scope output. A *smooth* sine wave should fit into your scope window. This is what you would expect, since you are running a double-precision software simulation.
15. Stop the simulation.



Precision and the Xilinx Gateways

Now that you have seen some of the inputs (sources) and outputs (sinks) available in Simulink, you will create your first design using System Generator blocks. All System Generator designs start with the Xilinx Gateway Blocks. The Xilinx Gateway In and Gateway Out blocks provide an interface to the Xilinx Blockset in Simulink. If you want to add an FPGA design to your Simulink model, the Xilinx Gateway In block represents an input port into the FPGA. The Gateway Out block is an output port from the FPGA.

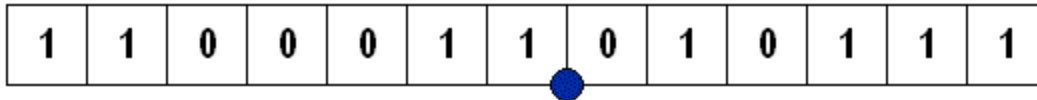


Figure 2-9 The Xilinx Gateway blocks

MATLAB uses double-precision floating-point and the Xilinx portion of the design uses fixed-point precision. Xilinx Gateway blocks handle the type conversions. In the following manual exercises, you will consider the number of bits necessary to represent fractional numbers through the Gateway In block:

Precision exercises

- The System Generator uses a `Fix` notation which shows the total number of bits in a number, followed by the position of the binary point. Using this notation, define the format of the following 2's Complement binary fraction and calculate the value it represents:



Format `<Fix_ _ >`
 Value =

- Represent 8.4375 as a 2's Complement binary fraction in the box below. You must also show the position of the binary point:

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

- Represent the following useful coefficient in `<Ufix_8_8>` format.



- Now determine the next size format that will provide greater precision for the number above.

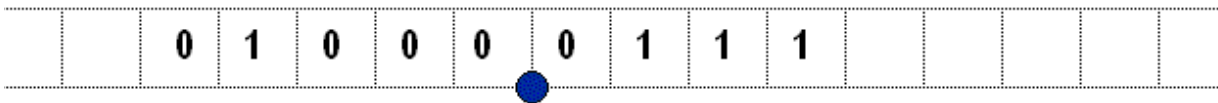
--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

The answers to the precision exercises 1-4 are below.

ANSWERS to precision exercises:

- Format `<Fix_13_6>`. Value = $(-1853/64) = -28.640625$

-

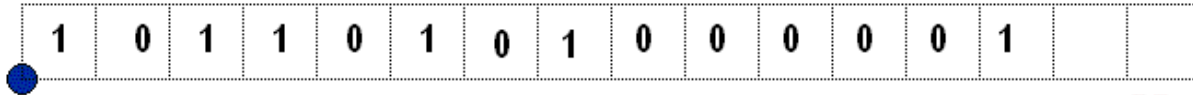


3.

$$\frac{1}{\sqrt{2}} = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\ \hline \end{array}$$

$181/256 = 0.70703125$ gives 0.01% error.

4.



$11585/16384 = 0.70709228515$ gives 0.002% error. The format must increase to `<Ufix_14_14>` before any improvement is gained.

Add Xilinx Gateways to Sine Wave example

Now, considering the means for representing different precision within the fixed-point data type, follow the steps to modify your previous sine-wave design by sending the signal through Xilinx Gateway blocks.

1. From the Xilinx Blockset (in the Simulink library browser), go to MATLAB I/O and drag the Gateway In block onto your sheet. Drop it on the connection between the sine wave and the output scope. It will automatically insert itself.
2. Also from MATLAB I/O, drag a Gateway Out block onto the sheet, and drop it between the Gateway In block and the output scope block.
3. We would like to compare the double-precision sine wave with the fixed-point sine wave that has gone through the Xilinx Gateways. To see both of these plots on the same scope, we will combine the outputs through a Simulink MUX block. From the Simulink Signals & Systems block set, drag a MUX and drop it between the Gateway Out and the scope.
4. Now add an additional net between the sine wave and the MUX. This way the scope will display both the double-precision sine wave and the sine wave that has gone into and back out of the Xilinx Gateways.

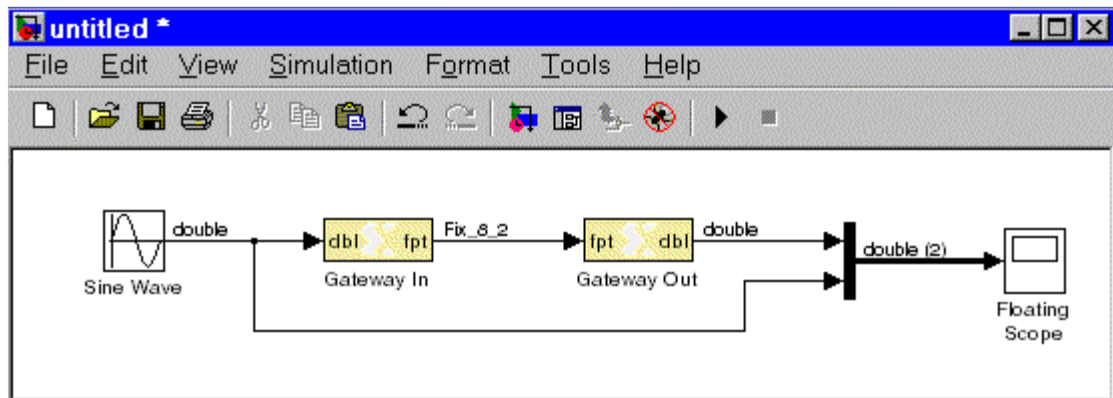


Figure 2-10 Sine wave example with Xilinx Gateway blocks inserted

5. To view the number of signals in the MUX, select the options under the **Format** menu:
6. Go to the **Edit** menu and select **Update Diagram**. Now look at your port types. See that the **Gateway In** block has changed the signals from double precision to fixed point types. Fixed point looks like `Fix_8_2` in this case.
7. Double click on the **Gateway In** block. A block parameters dialog box will open. Keep this box open for the rest of this lab; we will examine the effects of changing some of the parameters.
8. Run the simulation (click the **go** button). You will see a jagged sine wave next to the smooth sine wave that is not going through the Xilinx blocks. You are seeing quantization effects (the difference between the double precision floating point of MATLAB and the fixed point `Fix_8_2` of the Xilinx block).

- ✓ **Wide nonscalar lines**
- ✓ **Signal dimensions**
- ✓ **Port data types**

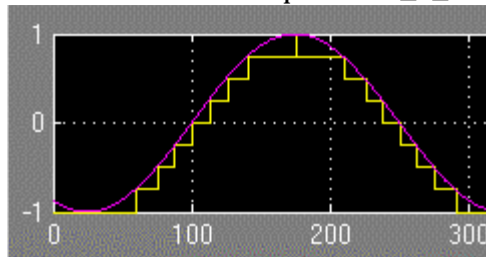


Figure 2-11 Quantization effects of double precision vs fixed point

9. Now change the **Gateway** block input to **Unsigned** (instead of `2's complement`). Click **Apply** on the **Gateway In** dialog box. Notice the output scope now displays unsigned results. (After you click **Apply** you may need to click the **Autoscale** button on the output scope again.)
10. Now that the value is unsigned, you have some overflow (the negative part of the sine wave). Since **Overflow** is set to **wrap** on the **Gateway In** dialog, you can see that the negative portion is wrapping on your output scope.
11. Change the **Overflow** option to **saturate** and click **Apply**. See the different results on the output scope. The negative part of the sine wave is saturated to 0.

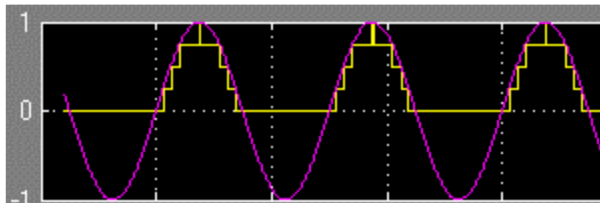
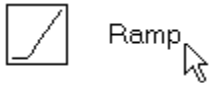


Figure 2-12 An effect of changing overflow option on Gateway In block

12. Change the input back to `2's complement` and click **Apply**.
13. Change the quantization to **Round** and click **Apply**. You'll see the sine wave is rounded up to the peak value.
14. Finally, change the quantization to **Truncate** and click **Apply**. Now instead of rounding up, the effect of quantization error is to truncate the peak value.
15. Remove some of the quantization error by changing the binary point. Instead of 2, increase the binary point to 6 and click **Apply**. Now you will see a smoother sine

wave, since more of the quantization error has been removed. The number of fractional bits was increased from 2 to 6.

16. Stop the simulation.
17. Now we will examine the effects of the Sample Period box. Instead of a sample period of 1, change it to 5 and click `Apply`.
18. Restart the simulation. Since it is sampling fewer times, you will see more quantization error.
19. Stop the simulation.
20. Go to your Simulink sheet and remove the sine wave. Replace it with a Ramp function from the Simulink Sources menu. 
21. Open the Simulation Parameters dialog box.
22. Change the stop time to 100.
23. Change the binary point to 0 and the sample period to 10, in the Gateway In block parameters dialog box.
24. Start the simulation. See that it is only sampling the ramp input every 10th clock period.
25. Now change the sample period to 1 in the Gateway In block parameters dialog.
26. Start the simulation. Notice that the ramp is smoother since you are sampling the block every clock period.
27. Stop the simulation.

At this point, you should be able to create a new System Generator design in Simulink. You know how to interface between the Xilinx Gateway blocks and Simulink blocks. The next step is to place Xilinx Blockset blocks within the Gateways and thus create the FPGA portion of your design. The remaining tutorials installed with the System Generator consist of designs created with the Xilinx Blockset, within these Gateway blocks.

Software Overview

This chapter tells how to use Simulink for modeling a system design with the System Generator. It also tells us how to simulate and implement the design using Xilinx implementation tools.

This chapter contains the following sections.

- Introduction
- Basic Simulink Functionality
- Customizing the Xilinx Blockset elements in a design
- Simulation
- System Generator Token
- Files produced by System Generator code generation
- Testbench Generation
- Implementation within Xilinx design environment

Introduction

This tutorial uses a simple design to illustrate the basics of the System Generator.

This tutorial walks you step-by-step through the design flow to illustrate the basics of System Generator. The primary goal is to provide enough information that you can quickly begin using System Generator on your own. A secondary goal is to give you an overview of the capabilities of the Xilinx tools.

The design example consists of a simple digital integrator, implemented using elements of the Xilinx Blockset, with test bench consisting of other Simulink blocks. As can be seen in the figure below, the input to the integrator is a scaled sinusoid, modified by additive noise and a linear ramp.

The integrator acts as a simple low-pass filter, which smooths the additive noise. The modulated input signal is further scaled for viewing in the Signal scope block, which also shows the integrator output, and the quantization error at the integrator output.

The example design, as seen in Simulink, is shown below.

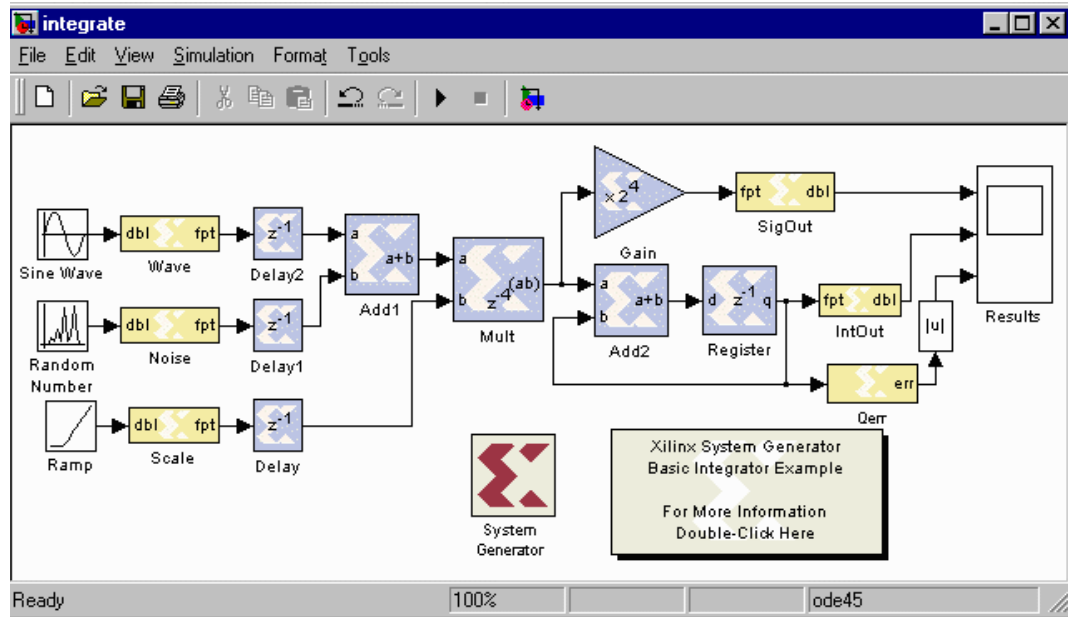


Figure 3-1 Example Design as seen in Simulink

Simulink Design Flow

Opening the Design Example

The default directory location of the example designs is:

`$MATLAB\toolbox\xilinx\sysgen\examples`. The design example for this tutorial is in the directory: `integrate`.

1. To bring up the tutorial design, start MATLAB by double-clicking on the desktop MATLAB icon (as shown on the right), or by launching it from the Start menu on your PC.



The MATLAB console window will open.

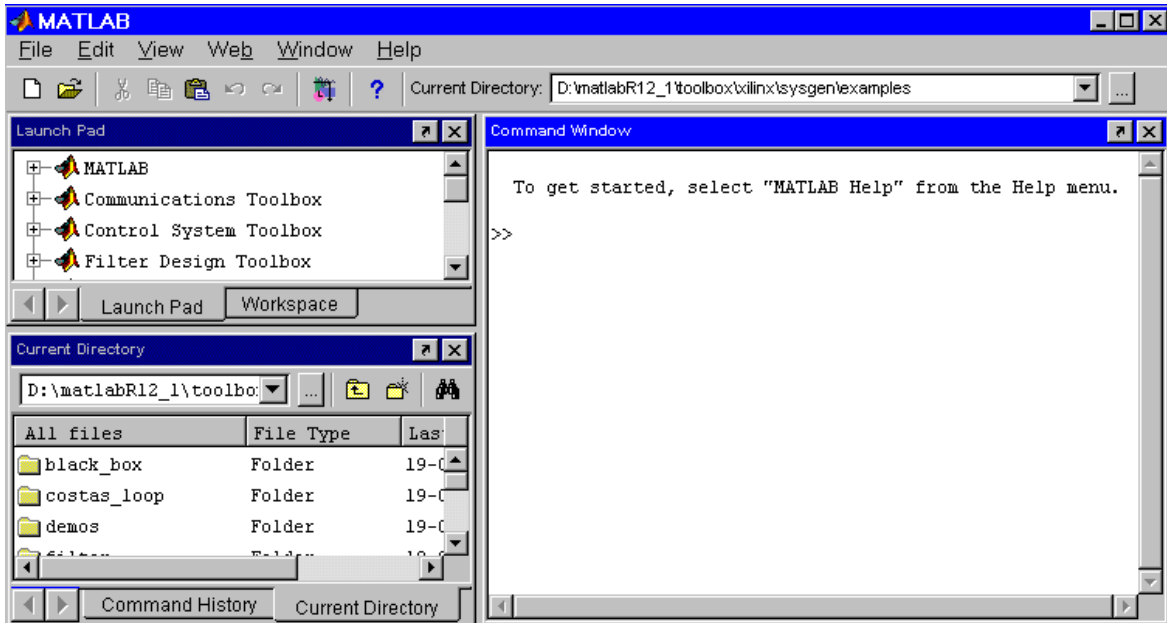
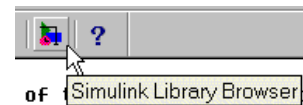


Figure 3-2 The MATLAB Command Window

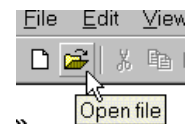
2. Now you need to launch Simulink. This can be done in two ways:

- ◆ Type `simulink` at the MATLAB console prompt, or
- ◆ Click the Simulink Library Browser button on the MATLAB toolbar



3. From the Simulink Library Browser, you can browse to the example design from the Open File button. You also can launch the design directly.

- ◆ Open the design from the MATLAB console by navigating to it. For example, if you have installed the tools to the default location:
- ◆ `>> cd C:\MATLAB\toolbox\xilinx\sysgen\examples\integrate`
- ◆ `>> ls` (this will show you the contents of the design directory)
- ◆ `>> integrate` (this will launch the integrate.mdl file)
- ◆ You can also browse to the design from the File>Open menu choice, or from the Open File button on the MATLAB console.



The Simulink Library Browser

The Simulink Library Browser is shown below. Depending on which Simulink blocksets you have installed, your library may contain different blocksets than those shown here.

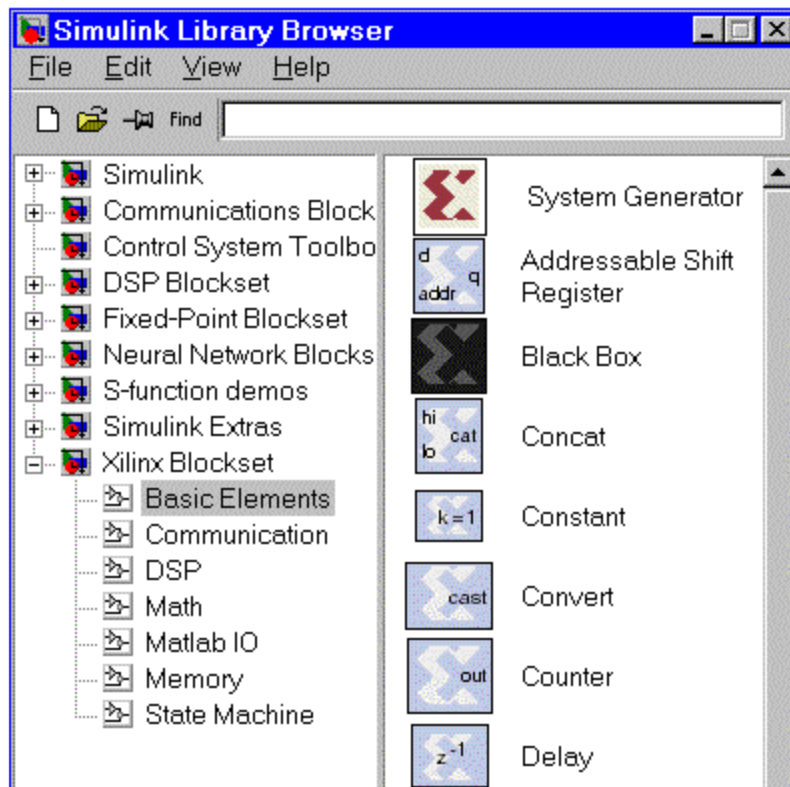


Figure 3-3 The Simulink Library Browser

The Simulink Library Browser contains the blocks necessary for you to create a Simulink subsystem.

In the figure above, note that the Xilinx Blockset *Basic Elements* library is selected. The blocks available in the Basic Elements library are therefore viewable in the pane on the right side.

4. You can expand the different categories of blocksets and see which individual elements are available, both in the Xilinx Blockset and in the other Simulink blocksets.

You will also be interested in the `Simulink>Sources` blocks, which contain sources you can use in your simulation models. The `integrator` example design uses the Sine Wave, Random Number, and Ramp blocks from this library.

5. A description of each Simulink block is available by right-clicking on the library element and selecting Help from the resulting menu.

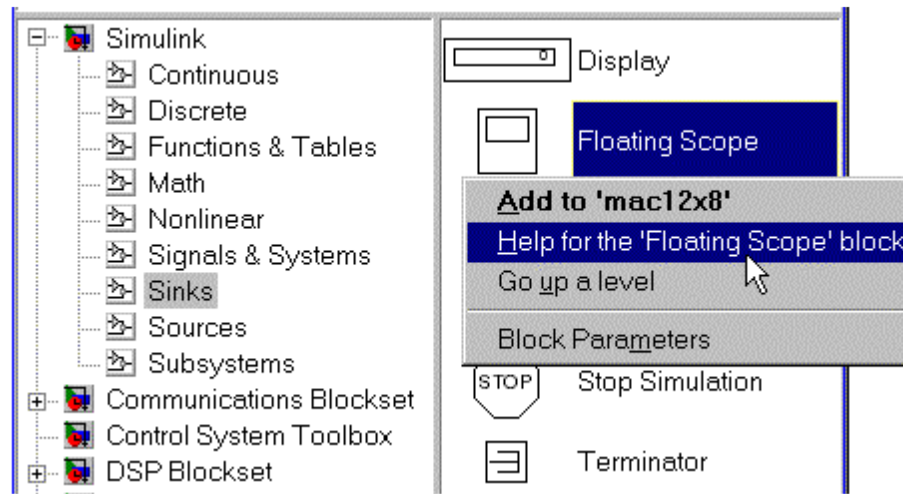


Figure 3-4 Selecting Help for any Simulink block

The Simulink Model Window

6. When you open the integrator example design, it will come up in the Simulink model window.

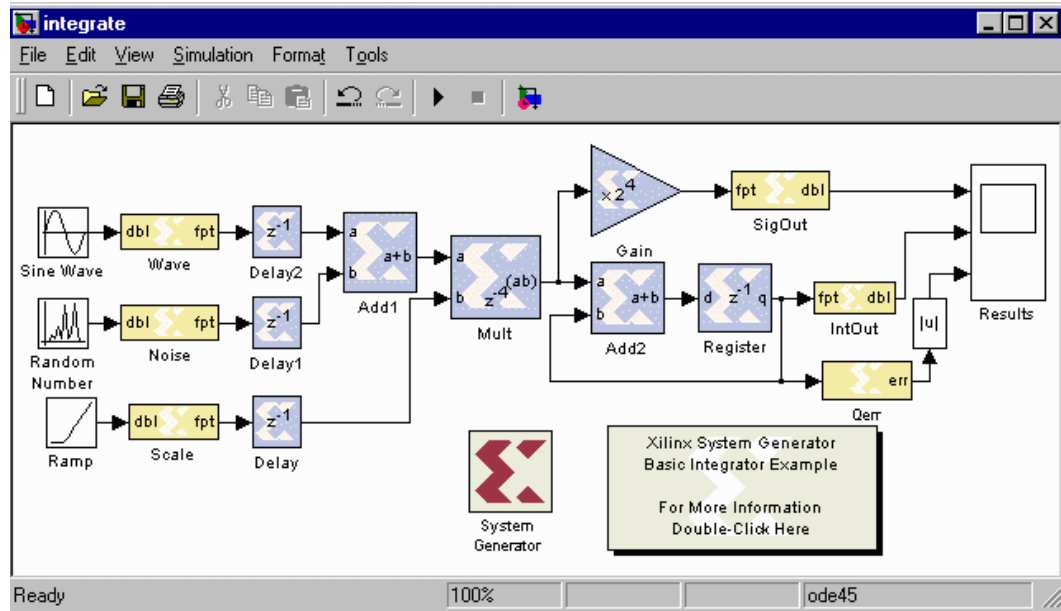


Figure 3-5 Integrator Example Design

You can drag or copy blocks into the Simulink model window from the Library Browser. You can wire them together as you would in a block editor.

The design example consists of a simple digital integrator, implemented using elements of the Xilinx Blockset, with test bench consisting of other Simulink blocks. The input to the integrator is a scaled sinusoid, modified by additive noise and a linear ramp.

The integrator acts as a simple low-pass filter, which smooths the additive noise. The modulated input signal is further scaled for viewing in the Signal scope block, which also shows the integrator output, and the quantization error at the integrator output.

The design uses input and output blocks from Simulink, with the remaining blocks all from the Xilinx Blockset. Each Xilinx block can be recognized by its Xilinx *shadow X* pattern. The Xilinx blocks can be translated into VHDL, targeting FPGA hardware.

Customizing the Xilinx Blockset Elements in a Design

- Like any Simulink block, each Xilinx block can be customized through the Block Parameters window that opens when you double-click on the block. For example, click on the Adder “Add2” and view its block parameters dialog box.

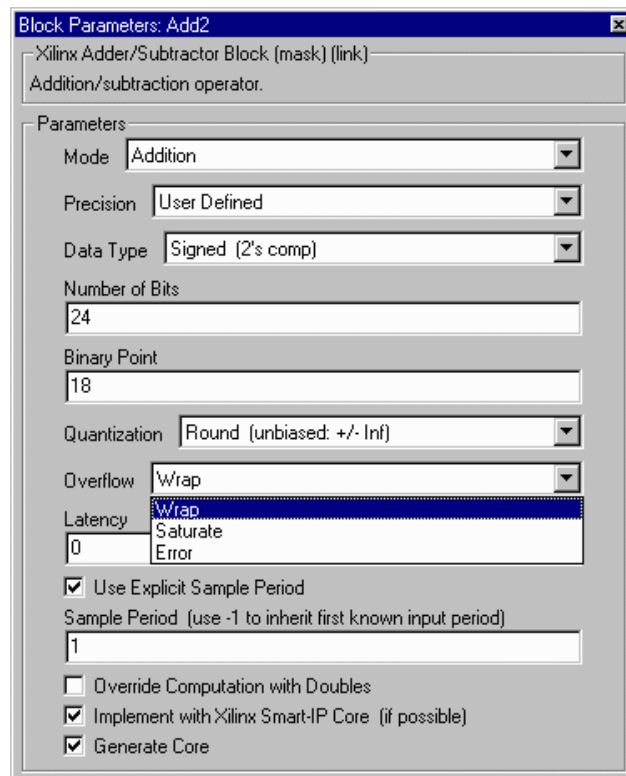


Figure 3-6 Adder block parameters dialog box

In this example, you may set Full or User Defined precision. User Defined precision expands the window to include all the precision options shown here.

- Double-click on some of the other Xilinx blocks on the Simulink model. View the options available in their block parameters boxes.

- Each block's parameters can be viewed by putting the pointer over the top of the block. A pop-up window shows what parameters have been set. For example, for the adder block just shown, the pop-up window is shown:

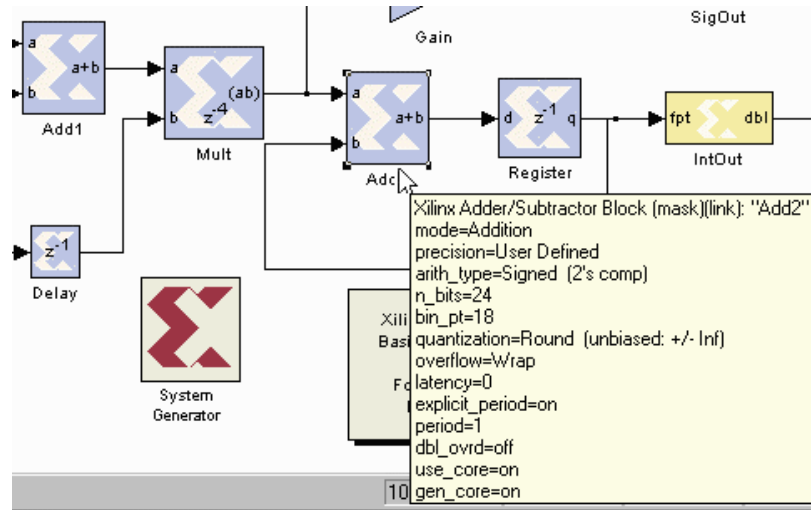


Figure 3-7 Viewing a Block's Parameters

- The Simulink model window has several options available under its **Format** menu. For example, choosing the **Port Data Types** menu item will display the precision of each element's ports on the design sheet.

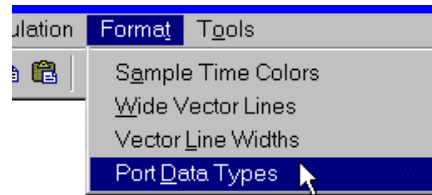


Figure 3-8 Format Menu Options

Gateway Blocks and Precision

The fundamental scalar data type used in Simulink is double precision floating point, which won't be translated into FPGA hardware. To bring part of the Simulink model into the FPGA world, signals pass through *Gateway blocks* that convert double precision floating point into a Xilinx fixed-point.

- Note the port precision for the **Wave** Gateway block in the design.

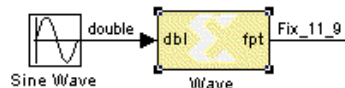


Figure 3-9 Input/Output Precision on Gateway Block

The **Fix_11_9** indicates this bus is a signed fixed-point number of 11 bits with binary point 9 bits from the right.

The Xilinx fixed-point type supports quantization (truncation or rounding to positive or negative infinity) and overflow (saturation or wrapping).

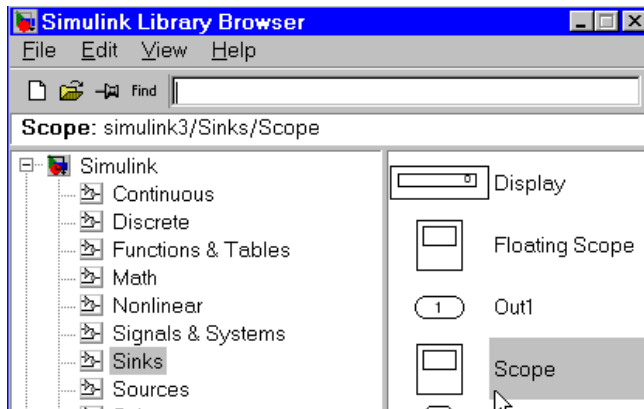
In the *integrate* example, the *Gateway-In* blocks control the initial precision. The first adder and multiplier produce full precision outputs. The integrator's output

precision allows only five signed bits, with saturation on overflow. The register inherits its precision from the block that drives it.

Simulation

The design also has three output scopes in the Results box. Results opens in its own window.

Output scopes can be dragged into your Simulink model from the Simulink>Sinks section of the Simulink Library Browser.



12. To simulate the design and view the results in the scopes, click on the Start/Pause Simulation button in the Simulink model window toolbar.



13. To scale the waveforms for easier viewing in each scope block, click on the Autoscale button in the scope block's toolbar:



The Signal scope shows the sine wave plus random noise, multiplied by a ramp signal. The Integrated Signal shows the signal after having run through the low-pass filter. The third scope shows the quantization error. (Every Xilinx signal has an associated double-precision floating point value that can be observed via the quantization error block, in the MATLAB IO section of the Xilinx Blockset.)

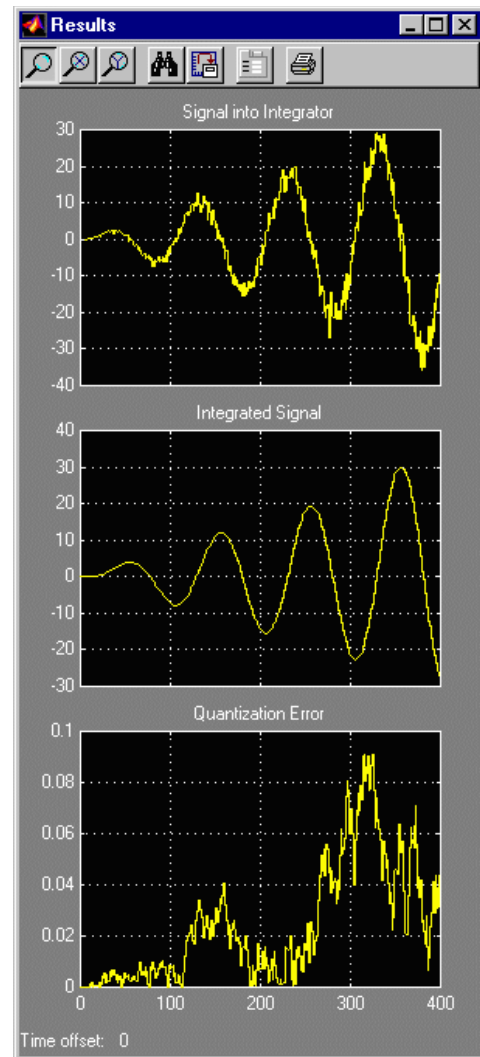


Figure 3-10 Output Scopes

- The design is set up to simulate from time 0 to 400. To simulate for a longer time, change the parameters in the Simulation menu pulldown in the Simulink model window.

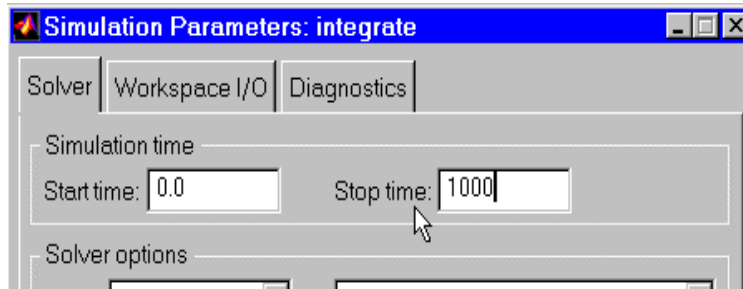


Figure 3-11 Changing Simulation Parameters

- You can also change the scope display properties by clicking on the Properties button in the scope window toolbox.

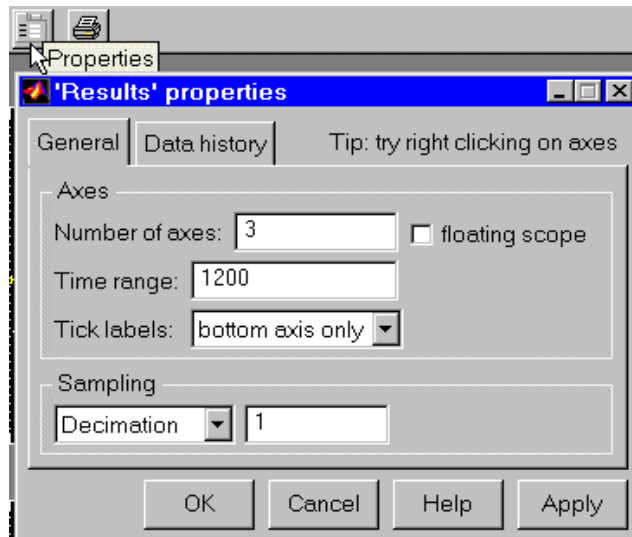


Figure 3-12 Displaying Properties

Simulation past time 400 shows increased quantization error as the integrated signal begins to overflow. (Remember to click on the Autoscale button to scale signals.)

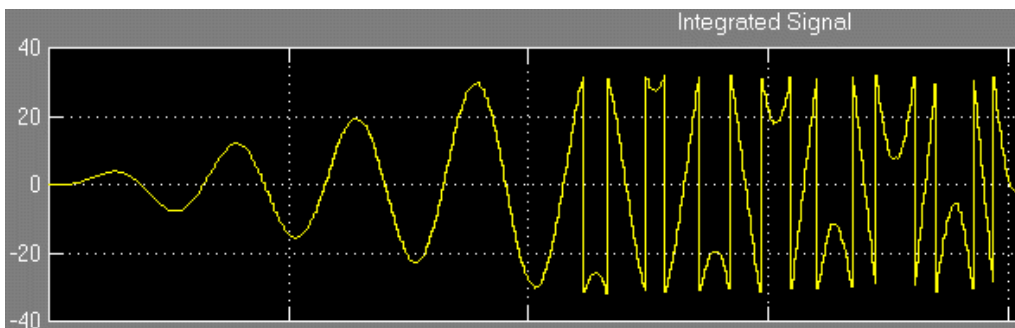


Figure 3-13 Quantization Error

You can see a 90-degree phase shift (the integral of a sine is a cosine) as the additive noise has been filtered out. The design is initially configured so the accumulator's adder will "wrap" on overflow.

16. Change the accumulator's adder to "saturate" and run the simulation again. Observe the difference in the overflow integrated signal and in the quantization error. (To change to "saturate" on overflow, double-click on the Add2 block, change the Overflow pulldown, and Apply the change.)
17. Experiment with changing the precision and overflow on other blocks. Try to predict the simulation outputs.

System Generator block

After you have finished modeling the system, you are ready to generate VHDL and cores for a Xilinx FPGA. We do this with the System Generator token from the Xilinx blockset.



If your design has hierarchy, drag the System Generator block to the highest level for which you want to generate hardware files. (For examples of designs with hierarchy, see the demonstration designs in the `examples/demos` directory.)

18. To generate VHDL and cores, double-click on the System Generator icon. This opens its parameters dialog box..

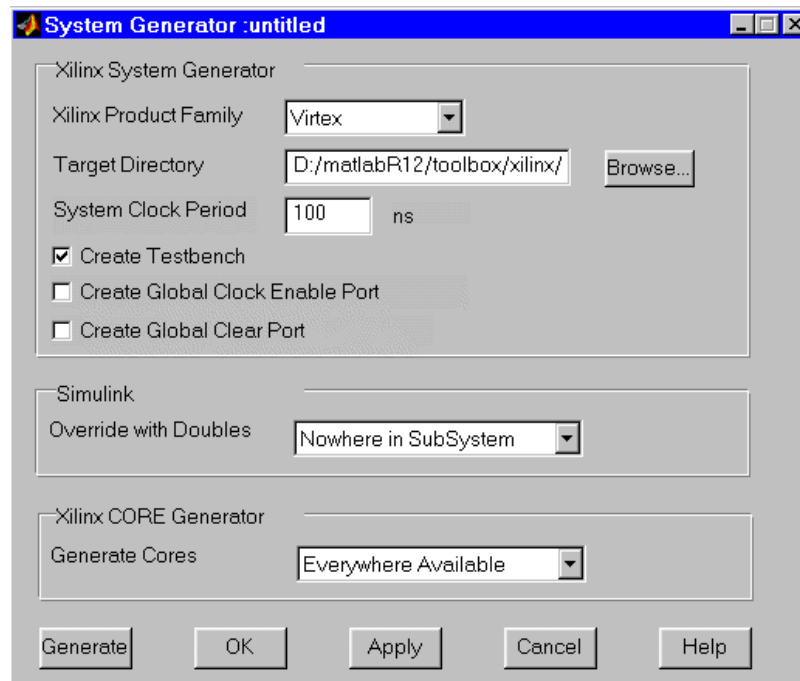


Figure 3-14 System Generator block parameters dialog box

Here you can choose

- ◆ FPGA target device family
- ◆ target directory in which all generated files will be written
- ◆ whether to generate testbench vectors
- ◆ where to use double-precision values in lieu of fixed point (this option applies only to simulation of the model within Simulink)
- ◆ whether to generate cores with the Xilinx CORE Generator (for your final implementation, you will want to invoke the CORE Generator; however,

when debugging, you may not want to spend the time needed by the CORE Generator)

- ◆ you can also specify a system clock period and global CLK or global CLR for constraining your design

The `Apply` button will save your selections and leave the window visible. The `OK` button will save your selections and close the window. Invoking the `Generate` button generates your VHDL and cores.

Files Produced by System Generator Code Generation

19. From the System Generator block parameters dialog, choose a target directory, choose the Virtex device, make sure the `Create Testbench` box is checked, and click the `Generate` button. You will see taskbars showing the System Generator, then the Xilinx CORE Generator running.
20. Now (in Windows Explorer or another file browser) view the files that have been written to your target directory. You will see the following files (among others):
 - ◆ `integrate.vhd` - the top level VHDL file for your project. There are additional VHDL files included when your design has more hierarchy.
 - ◆ `integrate_xlmult_core1` - files associated with the generated multiplier core, such as the behavioral simulation models and EDIF file.
 - ◆ `corework` - subdirectory containing the CORE Generator log file.
 - ◆ `integrate.npl` - project file for opening the design in Xilinx ISE 4.1i Project Navigator, using the XST synthesis compiler and ModelSim simulator.
 - ◆ `integrate_testbench.vhd` - the top level VHDL testbench file, associated with the top level VHDL source file in the project.
 - ◆ `integrate_<gateways>.dat` - stimulus files for inputs to testbenches, or predicted outputs of testbenches. The `.dat` files are generated by Simulink simulation and saved for running in Xilinx testbenches to verify design behavior. In this example, `<gateways>` refers to the names of the Xilinx gateway blocks, which collect and save the data.
 - ◆ `integrate_synplicity.prj` - a project file for running this design in Synplify (synthesis tools from Synplicity).
 - ◆ `integrate_leon.tcl` - a project file for running this design in Leonardo Spectrum (synthesis tools from Exemplar).

For a complete description of all of the files produced during code generation, please see Chapter 4 of the *System Generator Reference Guide*.

Testbench Generation

Testbench files were generated if you chose `Create Testbench` on the System Generator parameters dialog.

The testbench VHDL file, `integrate_testbench.vhd`, is a *wrapper* for your top level. The System Generator also generates `.dat` files. These contain test vectors representing inputs and expected outputs, as generated and observed in the Simulink simulation.

21. View the `integrate_testbench.vhd` file, and you will see references to the `.dat` files.

Implementation within the Xilinx Design Environment

After code generation, you are ready to simulate your design in a behavioral simulator, then synthesize it using any of the synthesis compilers that support Xilinx devices. In this tutorial, we will use MXE (Modelsim Xilinx Edition) simulation and XST (Xilinx Synthesis Technology) synthesis through the Xilinx Foundation ISE 4.1i Project Navigator tool.

The System Generator has created a basic Foundation ISE project file for you. By opening this project file, you can import your System Generator design into the ISE Project Navigator, and from there, you can continue to work on the design in the Xilinx 4.1i software tools environment.

22. Double-click on the `integrate.npl` file that was created. The ISE Project Navigator environment will open and will read in your System Generator project.

When first opening your System Generator project, you will receive a warning indicating that you have not set up a device package. This is because System Generator did not require that you enter a device package before generating VHDL.

23. You may now configure the rest of your Xilinx design by opening the Project Navigator properties dialog. Right-click on the device and default package at the top of the module view, and select `Properties`.

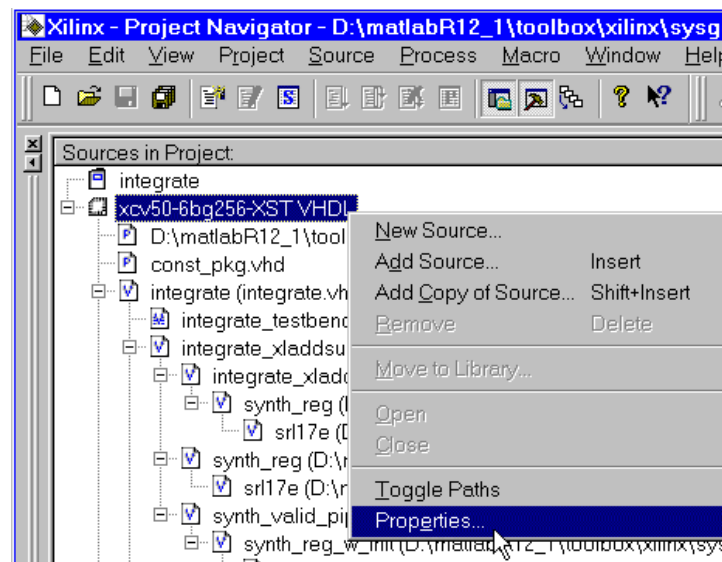


Figure 3-15 Opening Project Navigator properties on your design

From within the Properties dialog, you can choose other device families, speed grades, packages, and VHDL compilers. For now, we will use the defaults that have been set up already.

Behavioral Simulation

The System Generator project is already set up to run your behavioral simulation with the ModelSim simulator from the Project Navigator. It will use a custom "do" file called `pn_behavioral.do`. This file was produced by System Generator and has been associated with your behavioral simulation for this project in the ISE Project Navigator.

24. Select the `integrate_testbench.vhd` file in the Project Navigator sources module view. When you select the testbench, you will see that the processes window changed to show available simulation processes. Double-click on Simulate Behavioral VHDL Model.

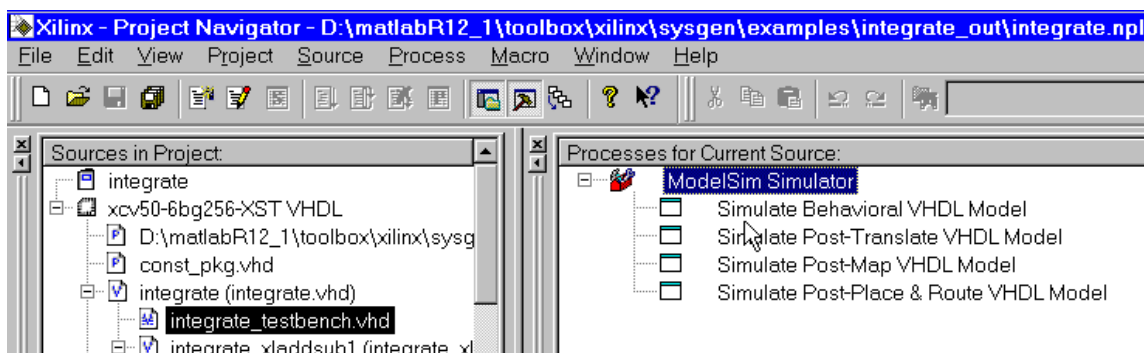


Figure 3-16 Simulation processes associated with `integrate_testbench.vhd`

25. The ModelSim console will open, and the `pn_behavioral.do` file (created by System Generator) will run. It will compile and run the same System Generator simulation that you ran in Simulink. To examine the results graphically, you will look at the ModelSim debug windows. (You may view all of the debug windows by choosing **View All** from the console pulldown menu. Further instruction on the ModelSim environment can be found in the *Xilinx Foundation ISE 4.1i* documentation.) After verifying your behavioral simulation, you may close ModelSim.

Implementing your design

You have many options within Project Navigator for working on your project. You can open any of the Xilinx software tools such as the Floorplanner, Constraints Editor, report viewers, etc. To implement your design, you can simply instruct Project Navigator to run your design all the way from synthesis to bitstream.

26. In the Sources window, select the top-level VHDL module in your design. Now you will notice that the Process window shows you all available processes that can be run on the top-level VHDL module.

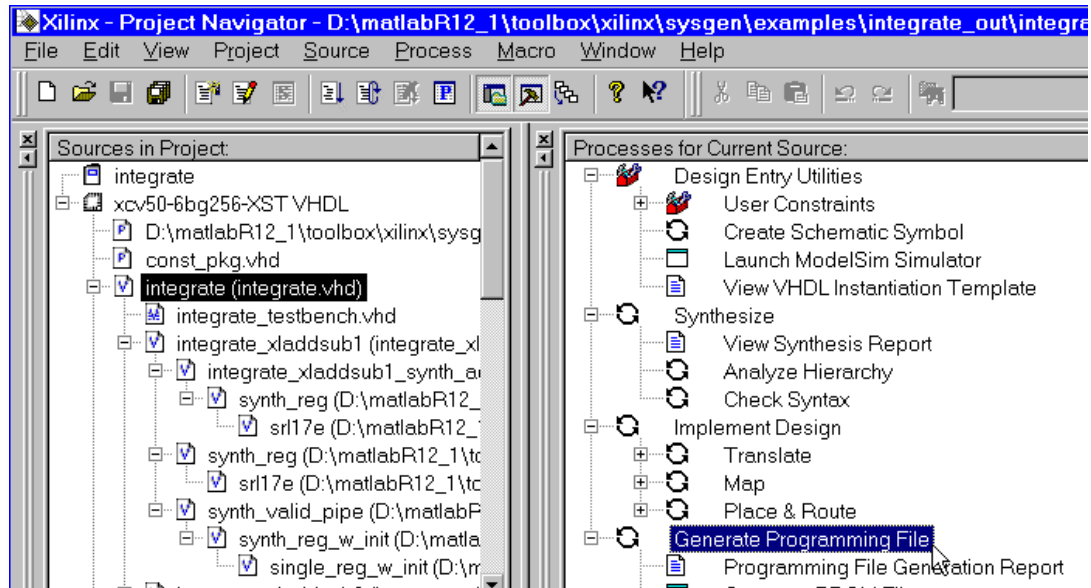


Figure 3-17 Processes available to run on top-level integrate.vhd

27. In the Process window, right-click on Generate Programming File and select Run. You are instructing Project Navigator to run through whatever processes are necessary to produce a programming file (FPGA bitstream) from the selected VHDL source. In the messages console window, you will see that Project Navigator is synthesizing, translating, mapping, routing, and generating a bitstream for your design.

Now that you have generated a bitstream for your design, you have access to all the files that were produced on the way to bitstream creation.

28. For example, if you wish to see how your design was placed on the Xilinx FPGA, select the FloorPlanner view underneath the Place & Route option in the Process window. The Floorplanner window will open, showing your implemented design. .

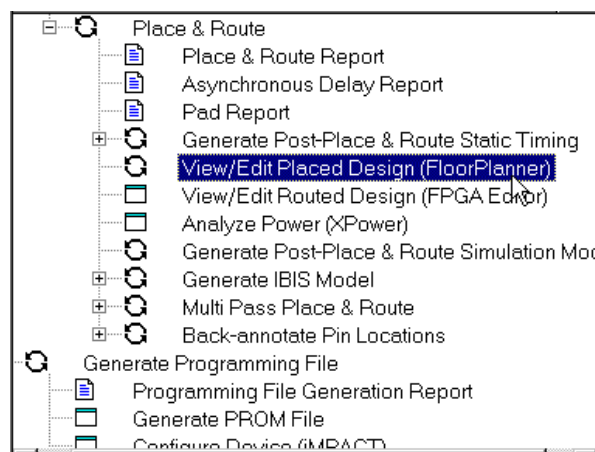


Figure 3-18 Click to run Floorplanner view on implemented design

Black Boxes

This chapter tells how to use black boxes in System Generator. Sometimes a design must include subsystems that cannot be realized with Xilinx blocks. For example, the design might require a FIR filter whose capabilities differ from those in the filter supplied in the Xilinx Blockset. Black boxes provide a way to include such subsystems in designs otherwise built from Xilinx blocks. To add a black box to a design, do the following:

- Implement the subsystem (your black box) in Simulink. The subsystem can contain any combination of Xilinx and non-Xilinx blocks.
- Place the Xilinx Black Box token at the top level sheet of the subsystem. This indicates to System Generator that the user will provide the VHDL or Verilog HDL necessary to implement that subsystem.
- Double-click on the token to open the Black Box block parameters dialog box. Enter the information that describes the black box.
- You must manually enter your VHDL or Verilog HDL black box files into your downstream software tools project after you run the System Generator code-generation step.

A Black Box Example Model

The directory: `/xilinx/sysgen/examples/black_box`, ordinarily stored in `$MATLAB/toolbox`, contains an example showing how to use black boxes.

1. For this example to run correctly, you must change your directory (`cd` within the MATLAB console window) to this directory before launching the example model.

The files contained in this directory are:

- `black_box.mdl` - the Simulink model with an example black box
 - `bit_reverse.m` - a MATLAB function for reversing bit order
 - `bit_reverse.vhd` - VHDL code for reversing bit order. This file is the actual black box that must be passed to the Xilinx implementation tools. It imitates the behavior of the MATLAB function.
2. Open the design example by typing `black_box`.

The example project displays three windows:

- The top-level model (a model with black box instantiated in it),
- The black box (a new Simulink model), and
- The output simulation scopes.

- Run the simulation from the top-level model, and you can see the bits reverse in the output scope. This simulation is running the MATLAB function `bit_reverse`.

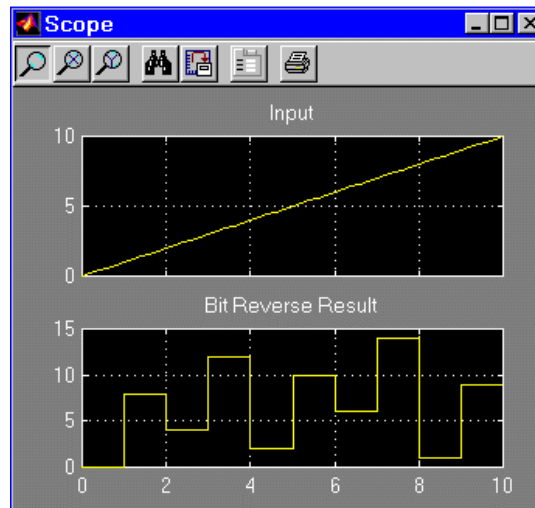


Figure 4-1 “Bit-Reverse” Black Box Simulation

The Black Box Window

The Xilinx Black Box token identifies the top level of your black box.

- Double-clicking on this token brings up a window that allows you to configure the black box.
- Open the file `bit_reverse.vhd` in an editor and view the code. You will see the name of the component (`bit_reverse`) is the same name that you assigned in the Black Box configuration window. The user-defined generic (`n_bits`) is defined there as well. The others are default generics that correspond to the ports (`DIN` and `BRN`) on the black box. You must make sure the VHDL code you write has component and generic names matching those entered in the configuration window.



Notice the
`main : process (DIN)`
 section near the bottom of the VHDL file. This is where the actual bit-reversing functionality takes place.

Multiplier Accumulator

This tutorial shows how to create a multiplier/accumulator and use it as a block within a FIR filter created out of Xilinx blocks.

You will start with an incomplete Simulink model and complete it by finishing one of the model sheets.

Incomplete Design

1. Open the incomplete model `mac_fir_tutorial`. You will find this model in the `$MATLAB\toolbox\xilinx\examples\mac_fir` directory. Four sheets will open, plus a scope window.

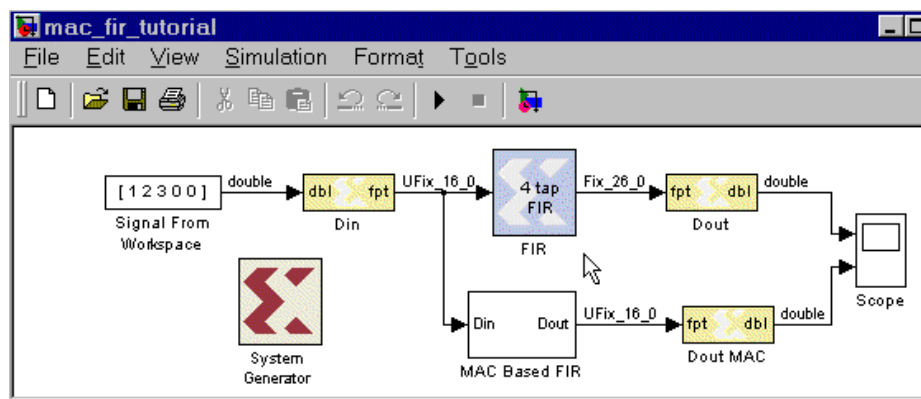


Figure 5-1 MAC FIR Tutorial Screen

The inputs to the model are defined in the Simulink Signal From Workspace (from the DSP Sources Blockset).

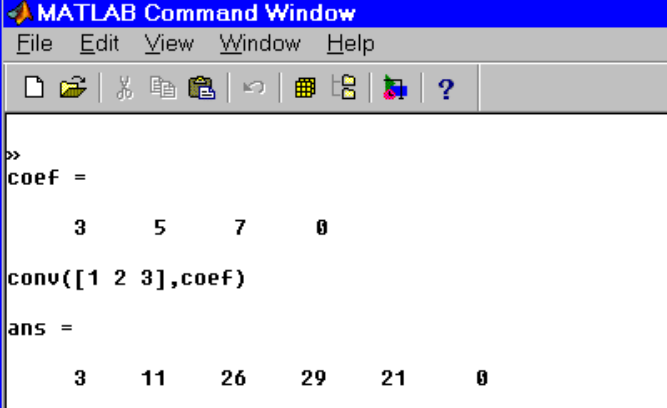
You will see there are two FIR filters in this model. The first is a single Xilinx block that uses the Xilinx FIR core. The second is MAC-based FIR assembled from several Xilinx blocks, and is incomplete.

2. When you open the model, you will also see that a set of coefficients has been set (in the MATLAB console window). In MATLAB M-Code, you can convolve the input signal (shown on the Simulink sheet as the input to the model) with these coefficients.
3. In the MATLAB console window, type:


```
conv([1 2 3],coef)
```

 and note the results. Note that `coef(4)=0`, so you could really create this design

with a 3-tap FIR filter, but the MAC FIR design we are using assumes the number of taps is a power of 2. This simplifies the addressing of memories.



```

MATLAB Command Window
File Edit View Window Help
[Icons]
>>
coef =
      3      5      7      0
conv([1 2 3],coef)
ans =
      3     11     26     29     21      0

```

Figure 5-2 MATLAB Command Window

- At the top-level sheet, simulate the design (click on the “Start/Pause Simulation” button in the Simulink toolbar).

Note: The result of the Xilinx FIR (on the scope) matches the answer in the MATLAB console.

The results appear after several cycles (period 4) because the Xilinx FIR contains a pipeline. Since the MAC-based FIR is incomplete, its results are meaningless. Your goal is to complete the MAC-based FIR so that its simulation results match those of the Xilinx FIR.

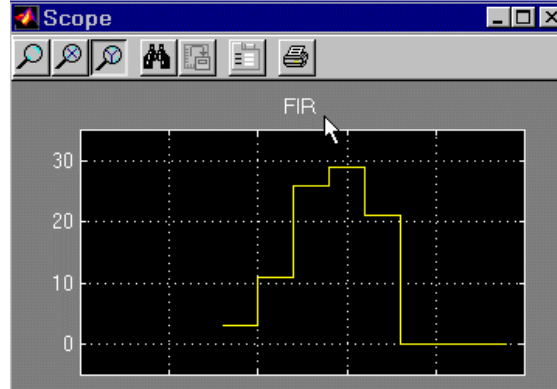


Figure 5-3 Xilinx FIR on Scope

- Push into the “MAC Based FIR” block. The subsequent sheet contains two additional blocks to push into. Push into “MAC.” This is the block you must complete.

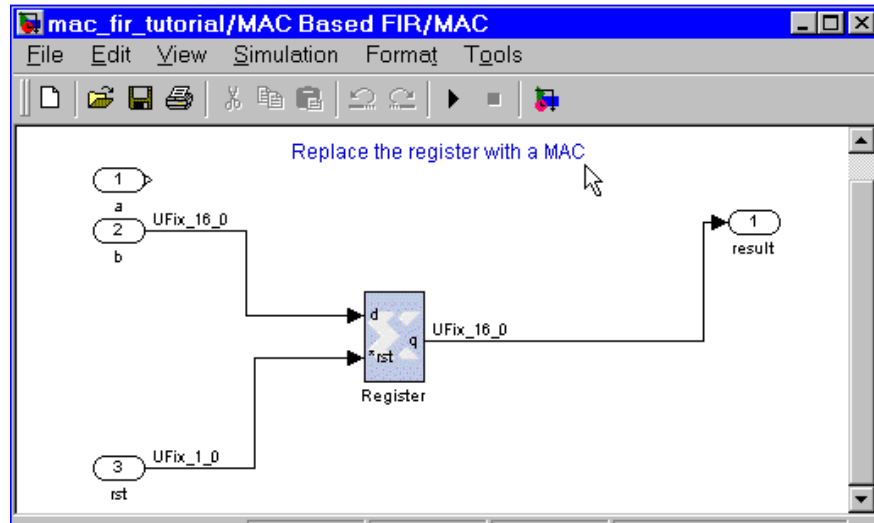


Figure 5-4 Incomplete Block

Complete the Multiplier/Accumulator

- Replace the register (a temporary place-holder in the MAC block) with a multiplier/accumulator (MAC). Requirements for the MAC are:
 - ◆ Two inputs (plus reset)
 - ◆ One output
 - ◆ Multiply the two inputs and put the result in an accumulator
 - ◆ Build the MAC such that there is zero latency between the inputs and the output
- Begin work on the MAC before you read the hints below. Don't read the hints before you have worked on the MAC!
- Hints:
 - ◆ The MAC has three blocks (a multiplier, an adder, and a register).
 - ◆ Use the Xilinx multiplier with latency of 0.
 - ◆ Build the accumulator out of an adder and a register.
 - ◆ The output of the multiplier feeds one input of the adder.
 - ◆ For the accumulator, the output of the adder returns into the register and then back into the second input of the adder.
 - ◆ Because of the adder's feedback loop, you will get the error: "input port data type mismatch" if you use the adder's default of Full Precision. The loop of causes the Full Precision setting to create mismatched port types. To get past this error, in the adder's configuration GUI you must select "User-Defined" precision and set the adder's output precision to the same precision as the feedback loop input precision.

- ◆ You may see the error: “sample periods at one or more inputs can't be determined.” Blocks in feedback loops need explicit sample periods. In the adder's configuration GUI, check the checkbox: “Use Explicit Sample Period” and choose -1. This choice of -1 is a convention specific to Xilinx blocks. It dictates that the block will inherit its first known input sample period. In this case, the adder will inherit its period from the multiplier. You could also set the period explicitly to 1, since that is in fact the cycle period at which this particular MAC is to be run.
7. The answer (the complete project) is available in the same directory where you found the tutorial, if you want to compare your answer. You will find the answer in `$MATLAB\toolbox\xilinx\examples\mac_fir\mac_fir_answer`.

More Fun With the MAC

(Optional Section)

8. If you need another challenge, set up the MAC so that the multiplier can be pipelined in hardware. Just checking the check-box in the GUI is the easy part, but you will then find that you need to add latency to the multiplier. This is a bit tricky, as you have to adjust delays elsewhere to synch up.
 - ◆ Hint #1: You need to add delay to the reset signal before it is delivered to the accumulator, and you also need to adjust the delay before the down sampler at the MAC output. This latter delay element is used to synch the final sum from the accumulator to the sample frame grabbed by the down sampler.
 - ◆ Hint #2: You need to add latency (a delay line) between the MAC and the down sampler to compensate for the pipeline latency and still have the down sampler read the correct data frame.
9. The two scopes that plot the comparison of the MAC-based FIR and the Xilinx FIR may show the correct answer but not at the same time period. Why is that? How can you figure this out by looking at the design?

(Answer: the two implementations have different latencies. They can be balanced by adding a delay line to the output of the MAC FIR. To view the latencies, you can look at the configuration GUIs for both the Xilinx FIR and the components of the MAC FIR.)

The Costas Loop

This chapter introduces the Costas Loop, an example provided with the Xilinx System Generator. There is a version of the model in the `sysgen/examples/demos` directory under `$MATLAB/toolbox/xilinx`.

Note: The Communications Toolbox (includes the Communications Blockset for Simulink), required to run the example, is used within the Costas Loop.

Design Overview

The Costas loop is a portion of the communication system described below.

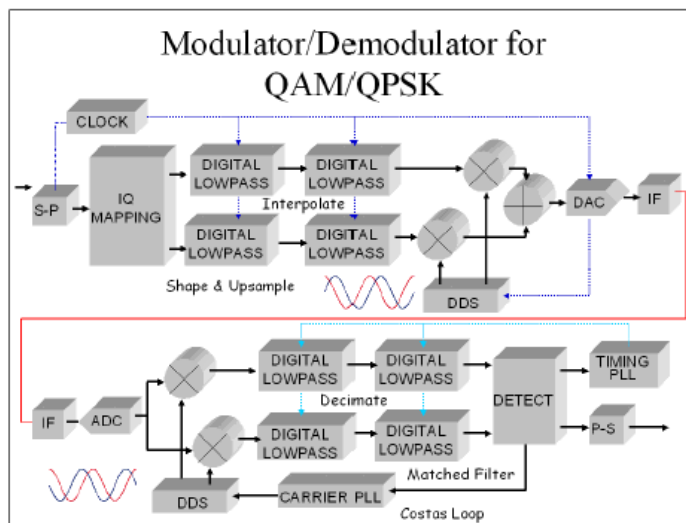


Figure 6-1 Example of a Communication System Showing a Transmitter/Receiver

This system works as follows. A signal is presented to the transmitter, mapped into symbols, fed through to a match filter and a polyphase interpolator in preparation for up conversion to a digital IF. The values are then converted to analog form and sent along the channel. When they get to the receiver, the samples are brought to baseband by complex heterodyne, fed through a match filter and a polyphase decimator to adjust the sample rate to the channel bandwidth. Finally, the values are presented to the detector where the symbol decisions are made.

The communication system must run a carrier phase lock loop (PLL) in order to generate a version of the local oscillator that is matched in both frequency and phase to the oscillator employed in the transmitter. Typically, a PLL is implemented as a Costas loop. Although both the transmitter and receiver can be implemented in a Xilinx FPGA, for the purpose of this tutorial, we will focus only on the Costas loop.

The Costas Loop design in System Generator

1. Open the Costas Loop model from the MATLAB console window. First, `cd` into the directory
`$MATLAB\toolbox\xilinx\sysgen\examples\demos`
2. Type `sysgenCostasLoop`, which is the name of the Simulink model.

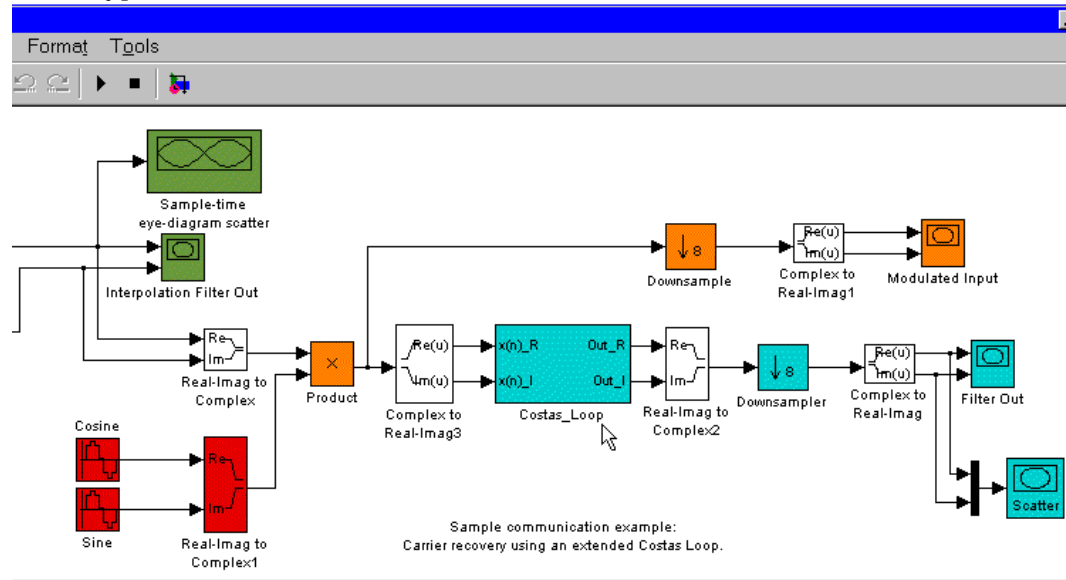


Figure 6-2 Top Level Sheet of Costas Loop

The system shown in the model is the simple transmitter and a Doppler shift to verify the operation of the Costas Loop.

In practice, the Doppler shift is associated with movement between the transmit and receive platforms, as might be the case with a cellular handset being used in a moving car.

The top left section of this model generates a QPSK modulation stream. The bottom left section is the introduction of channel impairment, i.e., Doppler shift.

Exploring the design

3. Explore the hierarchy of the model. In this model, the top level is composed of Simulink blocks, with Xilinx blocks making up the second level in the subsystem named `Costas_Loop`.
4. Simulation outputs scopes results in five output windows. Can you explain the results and how they were obtained through the Xilinx Blockset elements as well as the Simulink blocks in the design?

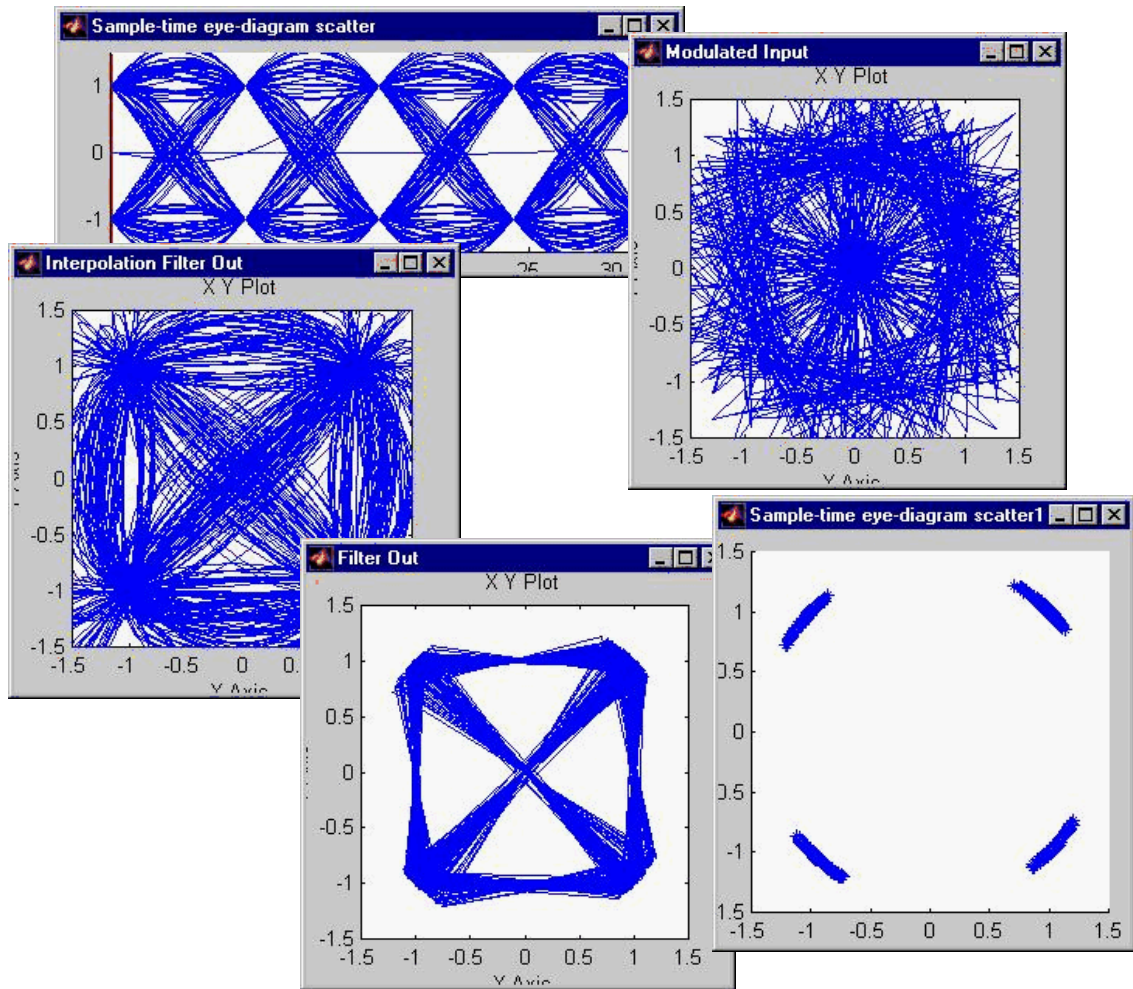


Figure 6-3 Output Scope Windows

Note: The default simulation time is 500. To increase the simulation time to 3000, which is required to see the output plots shown above, change “stop time” to 3000 in the Simulation Parameters dialog. This dialog is opened from the Simulation menu on your Simulink window.

Simulation results

Output scopes show the following:

- Constellation from the QPSK modulation stream
- Rotational nature of the constellation diagram, indicating that there is some frequency offset introduced
- Rotated data after presentation to our simple receiver

Chapter 7

Filter

This tutorial demonstrates the effects of applying a filter to a random signal. It also shows how to set up your Simulink model and coefficients by using some MATLAB console commands.

The model for this tutorial can be found in the `sysgen/examples/filter` directory under your `$MATLAB/toolbox/xilinx` directory.

SPTool Utility

This tutorial utilizes the SPTool interactive GUI. SPTool is available only with MATLAB's Signal Processing Toolbox, a collection of tools built on the MATLAB numeric computing environment.

Design Overview

The design is a complex random source that is filtered and run into a 16-point FFT used as a spectral analyzer, where the different frequency components can be observed. The unfiltered signal is also run into another FFT so the outputs can be compared.

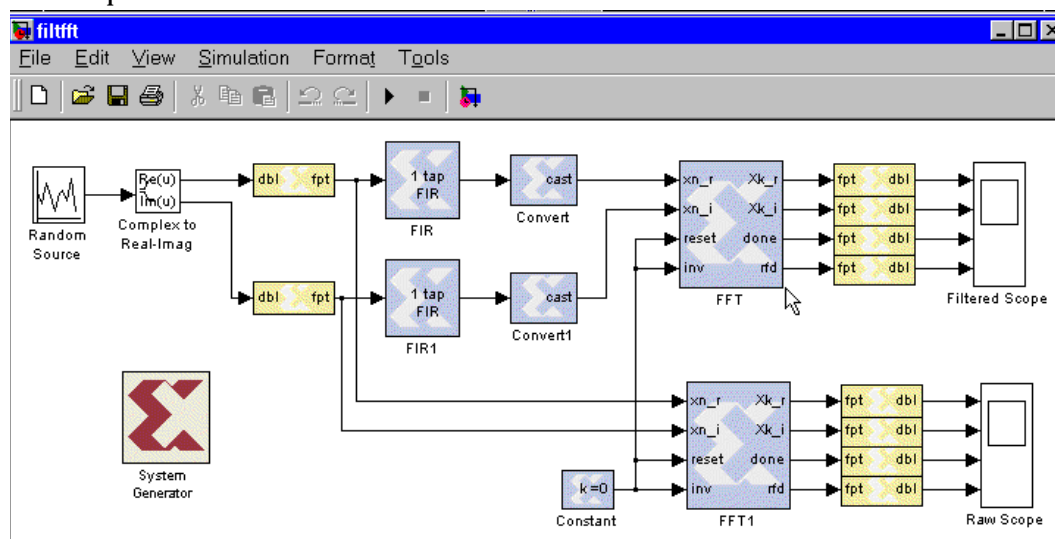


Figure 7-1 Filter 'filtfft'

5. Open the Filter model from the MATLAB console window. First, "cd" into the directory `$MATLAB/toolbox/xilinx/sysgen/examples/filter`.
6. Type `filtfft`, the name of the Simulink model.

You can see that the design consists of a complex random signal source that is fed into two identical FIR filters, the outputs of which are used as inputs to a 16-point FFT

(spectral analyzer). The FFT outputs are fed to a scope so you can see the frequency components of the filtered signal.

Observe that in the beginning the filters are 1-tap. In this tutorial, you will make the filters more interesting.

7. Simulate the model in Simulink. You will notice a Simulink error, saying that the DA FIR filter core cannot accommodate a 1-tap filter. You will fix this error by building better filters.
8. Cancel the error window, and observe in the scopes that the outputs of the filters have the same frequency characteristics as the unfiltered versions.

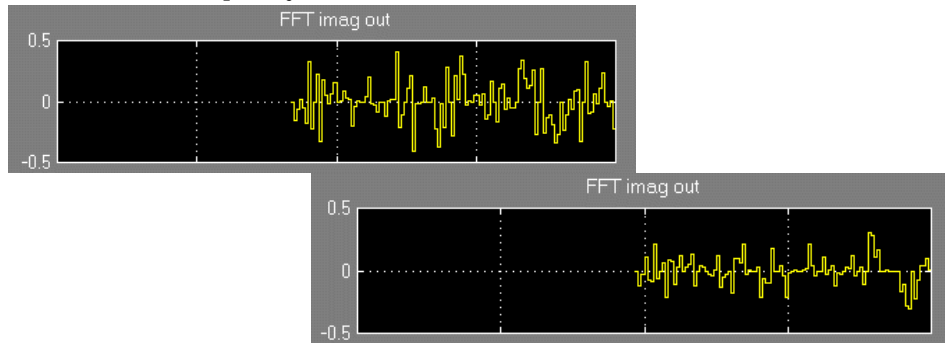


Figure 7-2 Unfiltered Scope Output

9. Double-click on the FIR filter block in your Simulink model window. This will open the FIR Filter mask GUI. Observe that the coefficients for this filter are defined by a vector called **h**. This vector has been defined in the file `filtInit.m`. The file was automatically loaded when you opened the Simulink model.

Preloading init Files

(Optional Information)

Note: This section contains instructions for preloading items (such as variables defining coefficients) into your Simulink models.

- Select the first FIR filter in your model and then type `gcb` in the MATLAB console. (`gcb` means “get the full block path name of the current Simulink block.”)

```
>> gcb
```

You will see that “ans = filtfft/FIR”

- You can get a handle to the top-level model by defining a variable as the “parent.” At the MATLAB console prompt, type

```
>> p=get_param(gcb,'parent');
```

- Now type

```
>> get_param(p,'PreloadFcn')
```

and you will see that the M-code file `filtInit.m` will be loaded whenever the model is opened. If this has not already been done for this model, you could set the pre-loaded file by typing

```
>> set_param(p,'PreloadFcn','filtInit');
```

This tells Simulink to load `filtInit.m` whenever you open this model.

Using this technique, you can define filter coefficients once (perhaps by using SPTool as you will do next), and then save them into a file and have them always loaded with your model.

Using SPTool

Next, you will create a low-pass filter using SPTool.

10. In the MATLAB console, type

```
>> sptool
```

The SPTool start-up window will open.

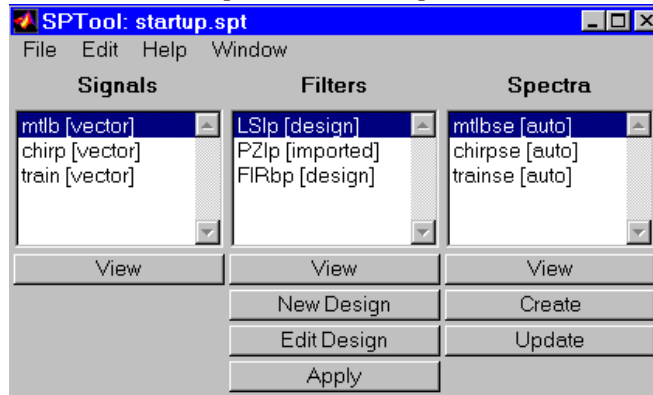


Figure 7-3 SPTool Start-up Window

11. In the SPTool start-up window, click on the “New Design” button. This pops up the Filter Designer with a sample “Equiripple FIR” low-pass filter with a -20dB stopband and the following settings:

- ◆ Sampling frequency: 8192
- ◆ Passband frequency: $F_p = 409.6$
- ◆ Passband ripple: $R_p = 3$
- ◆ Stopband frequency: $F_s = 614.4$
- ◆ Stopband ripple: $R_s = 20$

- ◆ Filter order = 22 (minimum order)

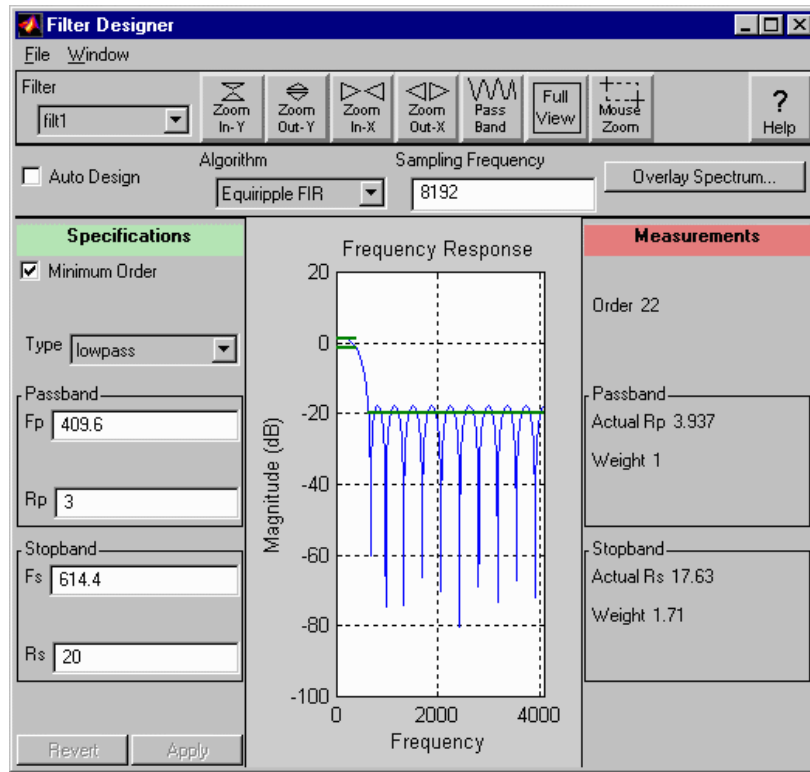


Figure 7-4 Filter Designer GUI

- From the SPTool start-up window, choose File->export from the File menu. This will pop up the Export window.

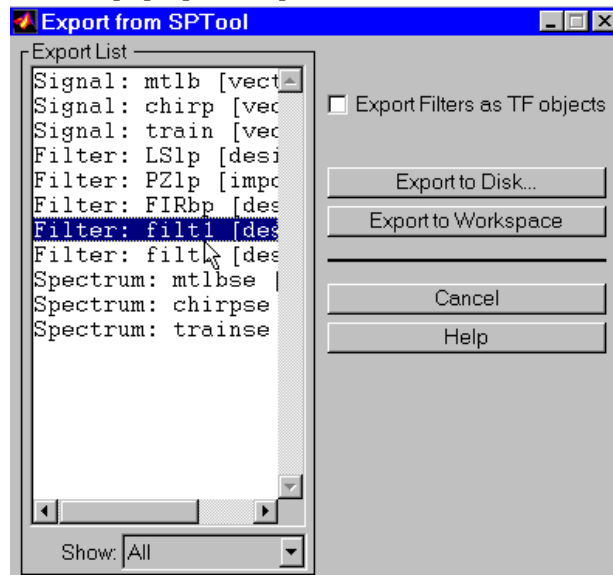


Figure 7-5 SPTool Export Window

- In the Export window, select Filter: filt1[design] and click the Export to Workspace button. You have now exported this new filter to the MATLAB console workspace.

14. Go back to the MATLAB console and type

```
>> filt1
```

Note: The `whos` command will display all the variables that you currently have in the workspace. Typing `whos` at this point will show you the vector `h`, preloaded by the model, as well as `filt1`, which you have just exported.

You can see that `filt1` is a structure.

```
filt1 =

    tf: [1x1 struct]
    ss: []
    zpk: []
    sos: []
    imp: []
    step: []
    t: []
    H: []
    G: []
    f: []
    specs: [1x1 struct]
    Fs: 8192
    type: 'design'
    lineinfo: []
    SPTIdentifier: [1x1 struct]
    label: 'filt1'
```

Its coefficients can be extracted and bound to the filters in our example model from the transfer function. (As this is a FIR filter, the denominator is 1.)

15. Transfer the coefficients from `filt1` to our vector `h` by typing the following in the MATLAB console:

```
>> h=filt1.tf.num;
```

Now, if you list the contents of the vector `h` (just type the variable name to see its contents), you will see that `h` has 23 elements.

16. Return to your Simulink model window, and with your cursor in the window, type `Ctrl-D`. This will recompile your Simulink model using the new vector `h`, and thus importing 23 coefficients into your filters. Notice that the number of taps in your FIR filters has changed to 23.

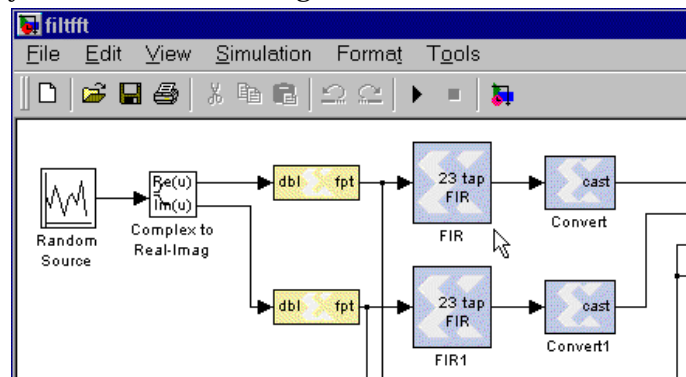


Figure 7-6 Changed Filter Window

17. Run the simulation again and observe that the filter is now greatly reducing the high frequency components in the FFT frames.

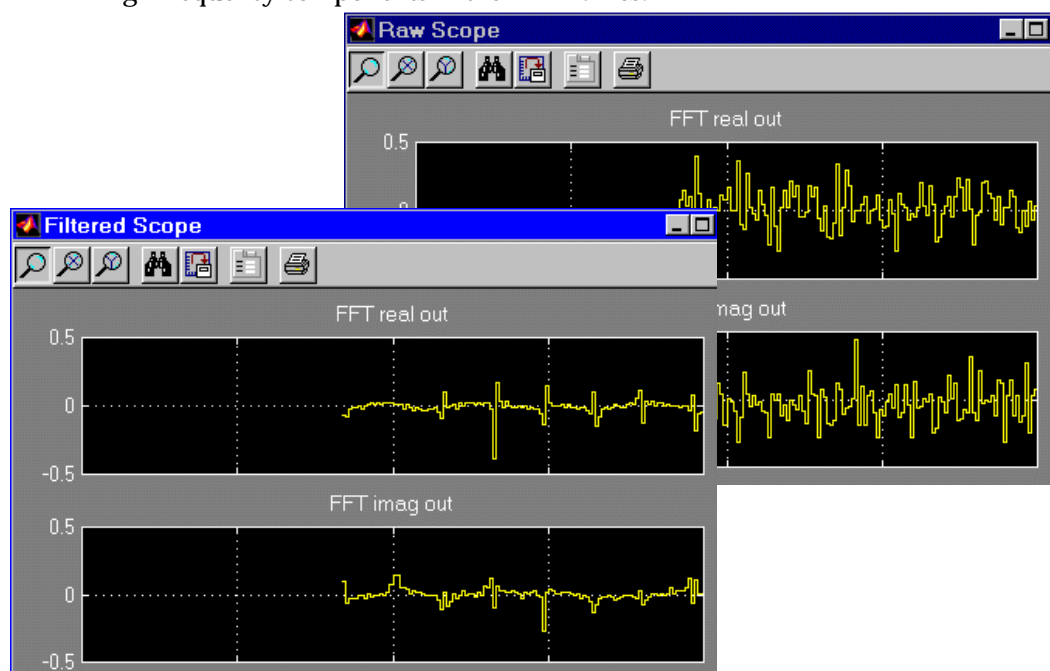
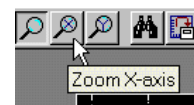


Figure 7-7 Change in Filtered Scope Output

18. To view the results in more natural datawidths, use the “Zoom X Axis” and “Zoom Y Axis” buttons on the scopes. Using the “done” pulse as a width to observe, you can view the data in sets of 16, which is more natural for FFT data when $N=16$.



19. Continue to experiment with SPTool, designing other filters (not only lowpass) and exporting the coefficients to the workspace, then importing them to your model.

Help on SPTool

Typing

```
>> help sptool
```

from the console causes MATLAB to display SPTool instructions.

Chapter 8

Image Enhancement

This chapter of the tutorial shows you how to use the Xilinx blocks to implement a simple image processing application. You will work with an image enhancement algorithm in Simulink and generate an FPGA implementation that can be simulated in an HDL simulator using test vectors created in Simulink by System Generator.

The tutorial uses the Image Processing Toolbox function `imshow()` to view images. If you have not purchased this toolbox from The MathWorks, it is possible to work through the tutorial, but you will have to view the image files some other way.

The input for this example is a medical image of a human heart. You will apply an image enhancement algorithm to increase the contrast of the image. After simulating the design in Simulink, you will generate a VHDL implementation in System Generator, simulate the VHDL in the ModelSim HDL simulator from Model Technology, synthesize the design for a Xilinx FPGA using the Synplify synthesis compiler from Synplicity, and place and route the design using the Xilinx Alliance Series software tools.

Note that although the Simulink simulation typically runs in several minutes, an HDL simulation that operates at a much lower level of abstraction (i.e. largely mapped onto gate-level primitives), may run for more than an hour. This is an indication of why modeling an FPGA design in System Generator is so powerful. With quick turnaround times, it is possible to increase dramatically the number of design iterations while developing the algorithm.

The model for this tutorial can be found in the `sysgen/examples/image_enhance` directory under your `$MATLAB/toolbox/xilinx` directory.

Design Overview

The image enhancement design consists of a two-dimensional low-pass filter and a mixer. The filter creates a blurred version of the original image, and the mixer is used to create a weighted difference between the blurred and original images. In the resultant image, the high-frequency components of the original have been amplified, having the effect of enhancing the image.

The two dimensional filter is factored into two one dimensional filters, that are sequentially applied to vertical and horizontal scan lines of the image in the `enh/Enhance/2D_LowPass/VFIR` and `enh/Enhance/2D_LowPass/HFIR` subsystems. In both subsystems, filter symmetry is exploited to halve the number of required multipliers. The mixer subsystem `enh/Enhance/Mixer` boosts high frequency components in the image by computing a weighted difference of the original image and the low-pass filtered image.

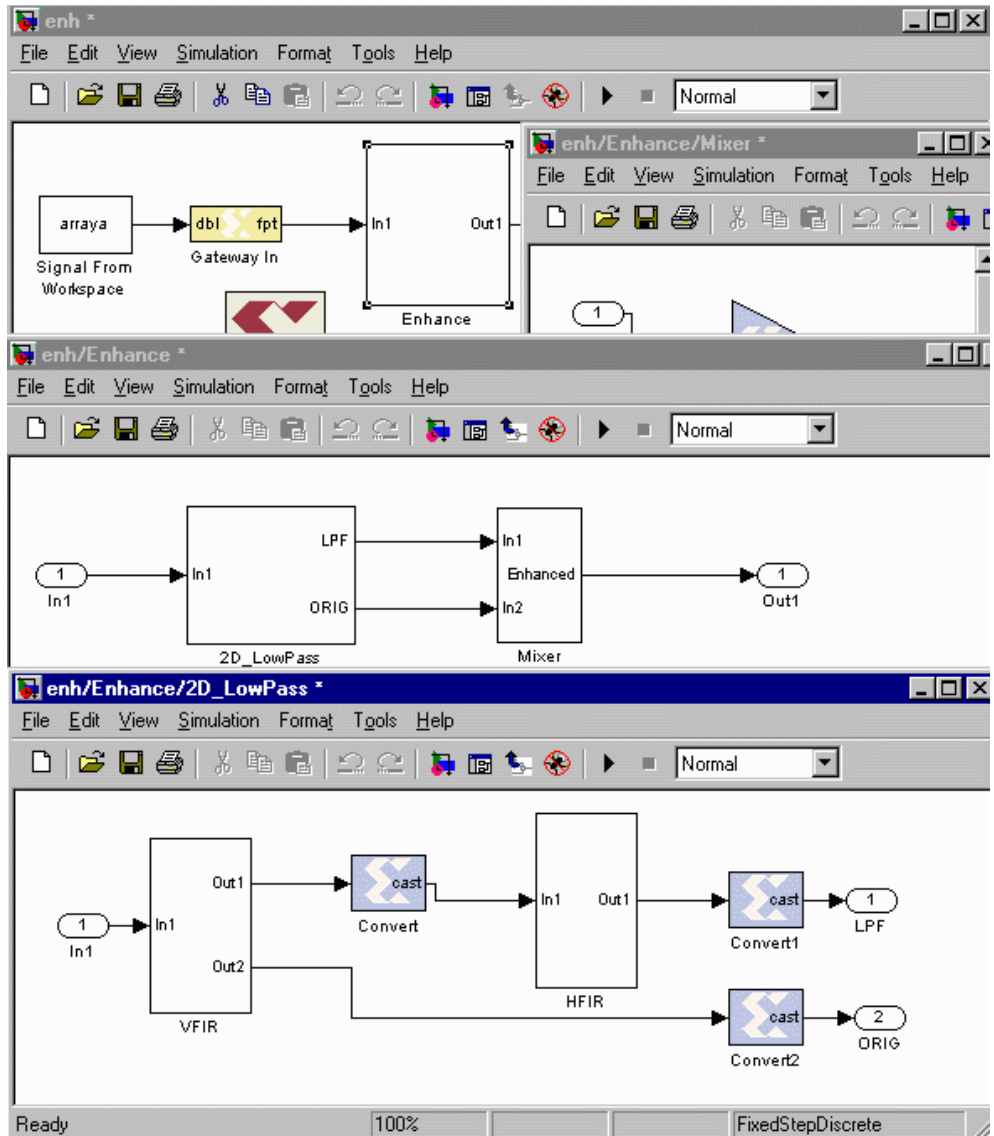


Figure 8-1 Image Enhancement Model in Simulink

Process the Design in Simulink

1. First, `cd` into the directory

`$MATLAB\toolbox\xilinx\sysgen\examples\image_enhance.`

2. Run a preprocessing script that assigns the image to a MATLAB array, by typing **PreProc_heart** at the MATLAB console prompt. Notice that this script also displays the unfiltered image of the heart for you to view, see Figure 8-2. The image can be found in the file `Heart.bmp` in your project directory.

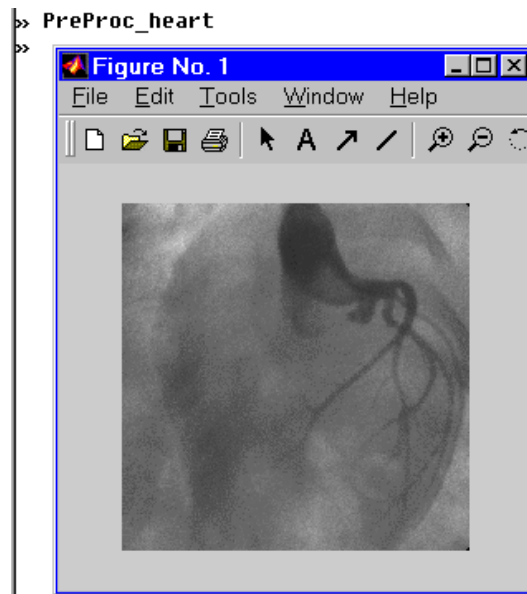


Figure 8-2 Unfiltered Image

3. Open the Simulink model by typing `enh` in the MATLAB command window. The top-level sheet will open.
4. Explore the hierarchy of the model to see how the design consists of a 2-D filter and a mixer, and how the 2-D filter has been decomposed into two 1-D filters, VFIR and HFIR.

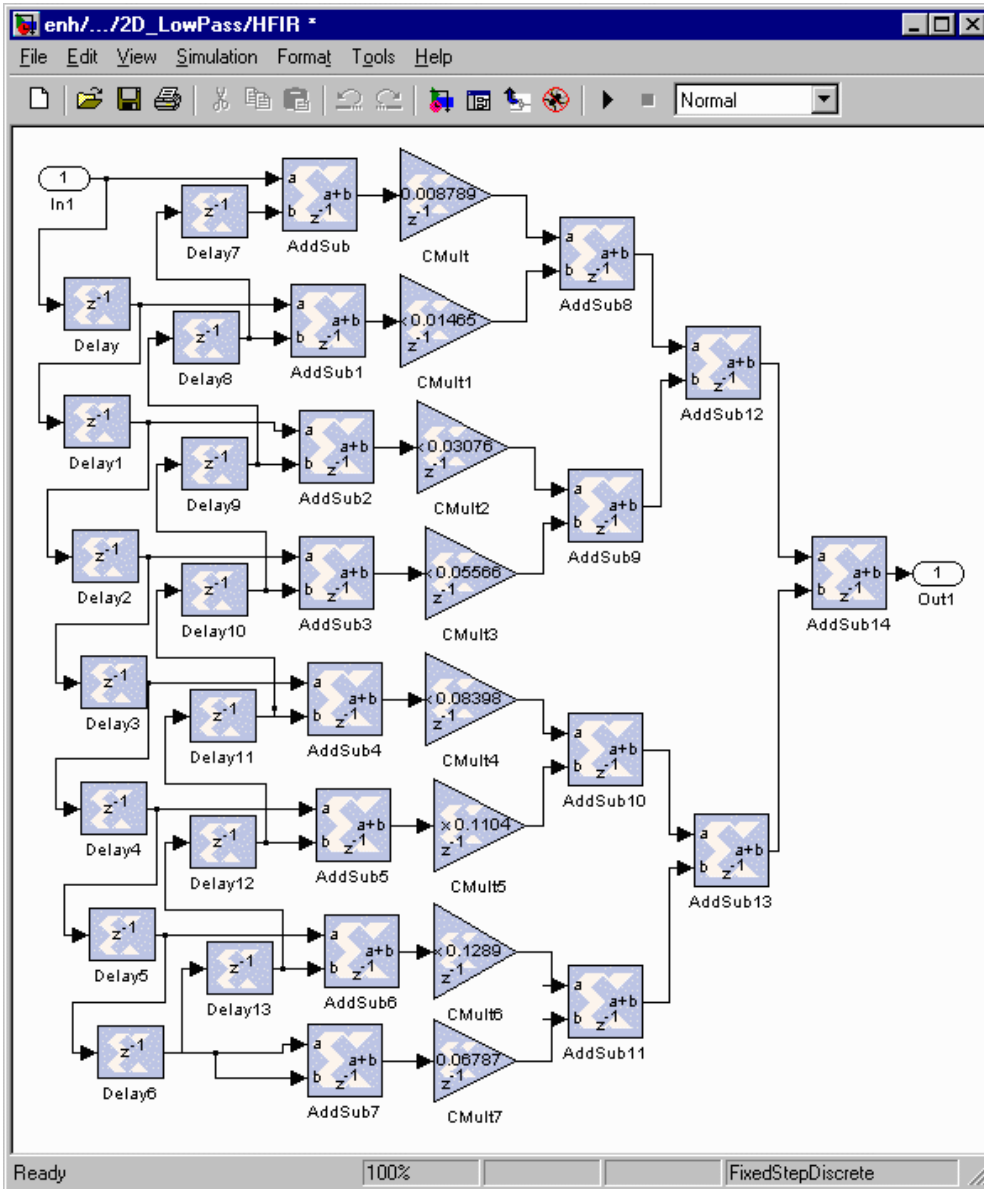


Figure 8-3 Low-Level HFIR Sheet

- Run the system simulation in Simulink by selecting *Start* from the *Simulation* pull-down menu. The simulation may take several minutes to complete. You can reduce the simulation *Stop time* parameter, but doing so will reduce the image enhancement.

A real-time system requires a processing rate of at least 30 frames per second.

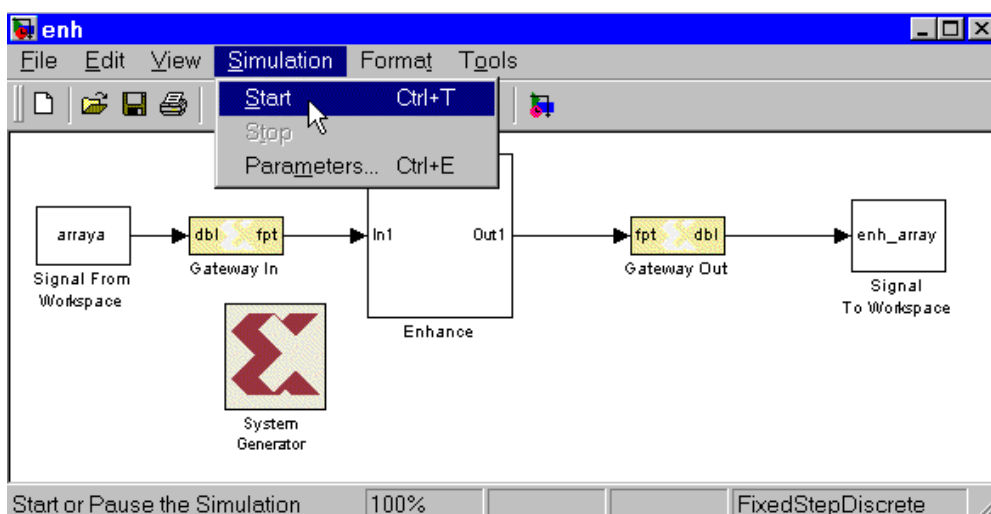


Figure 8-4 Start the Simulation

6. Return to the MATLAB console window and run a post-processing script by typing
PostProc_heart
 The original image, along with an enhanced image for comparison, will appear side by side. View the enhanced image in the comparison window. See Figure 8-5.

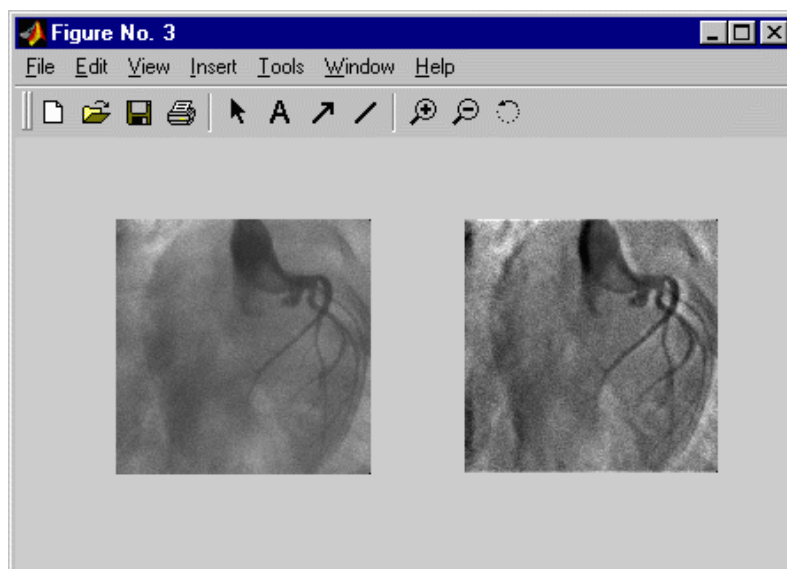


Figure 8-5 Original Image and Enhanced Version

Generate VHDL and Testbench Vectors

(Optional Section)

7. From the top level, double click on the System Generator token.

8. In the System Generator block parameters dialog box, choose an output directory and select “Create Testbench.” Click “Generate” to generate the VHDL code and testbench vectors for this design.

Since you have chosen to create testbench vectors, Simulink will again run the same simulation you ran earlier. This time, as it runs the simulation during the code-generation stage, it will save all of the simulation inputs and outputs into testbench vectors to be used in your VHDL and post-PAR simulation later.

Simulation and Implementation

(Optional Section)

If you like, you can simulate the design in an HDL simulator. The script files `vcom.do` and `vsim.do`, as well as testbench vectors (`.dat` files) have been created in your project output directory.

9. Open ModelSim and run `vcom.do` and `vsim.do` to verify the design’s generated VHDL. Or, if you prefer to run the design using the Xilinx ISE Project Navigator, you may open the `.npl` file that was generated by System Generator, and run the `pn_behavioral.do` file through ModelSim from the Project Navigator processes. (See tutorial Chapter 3 for details.)

Note: Running this simulation in ModelSim with the generated testbench files will take several hours. However, since you already ran the simulation in Simulink, you can be assured that your design is simulating properly. ModelSim simulation isn’t necessary, since System Generator gives you bit-true results that you already saw in Simulink.

Next, synthesize this design using Synplicity’s synthesis compiler, Synplify. System Generator has created a Synplicity project for you: it is `<project_name>_synplify.prj`.

10. Open the design in Synplify, using the project file created already.
11. Run the design through Synplify. This may take up to a half hour.
12. An EDIF file is created by Synplify. It is by default written into a subdirectory `rev1`. You may want to copy this EDIF file into another directory that you will use as the project directory for the Xilinx Implementation software tools.
13. Implement the EDIF file using the EDIF project capability of the ISE software tools. To do this, open the Project Navigator and select `File >> New Project`. A new project properties dialog will open. Select EDIF as the design flow type.

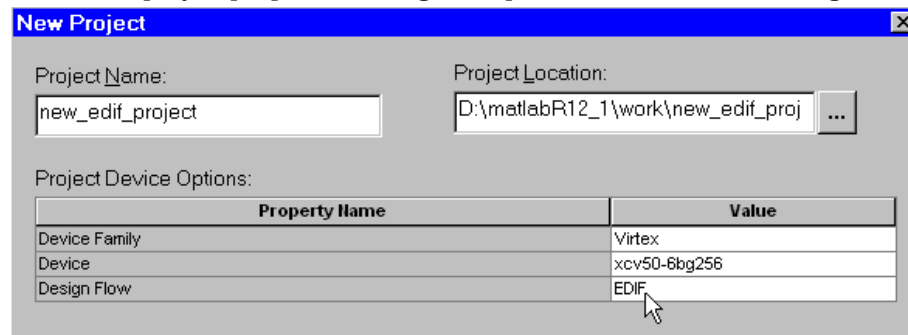


Figure 8-6: EDIF design flow in Project Navigator

14. Now you may add your EDIF files to the project as sources. From the Project Navigator pulldown menu bar, choose `Project>>Add Source`, and then browse to your EDIF files.
15. Target this design to a Virtex 600E part.
16. Translate, map, place & route your design using the Xilinx implementation tools. You can use other accessories in the Xilinx design suite to examine your design or to add attributes or constraints.

Related Information

For additional information about this particular design, see the paper: *Issues on Medical Image Enhancement* by Reza, Schley, and Turney, which can be found on the Xilinx website

http://www.xilinx.com/products/logicore/dsp/issues_enhanc_med_img.pdf

Combination Lock State Machine

In this example we design a state machine for a combination lock. We use the Xilinx Mealy State Machine block which is part of the State Machine Reference Library. The state machine is realized with block and distributed RAMs, resulting in a very fast and efficient implementation. This example uses only a single block RAM and runs at more than 150 MHz.

State Machine Library

The state machine library is available in the Xilinx Blockset.

When you start MATLAB and open the Simulink Library Browser, open the Xilinx Blockset and the State Machine section. You should see the following:

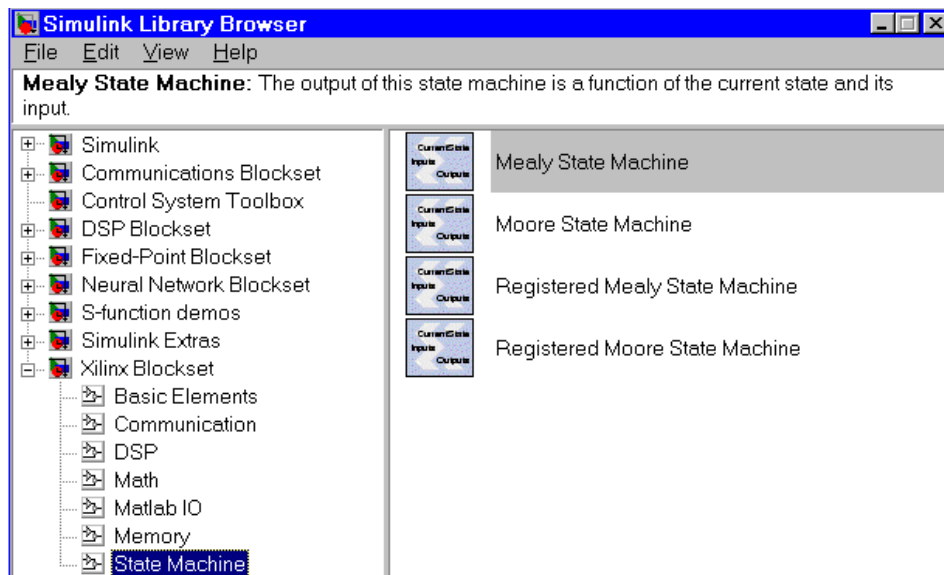


Figure 9-1 State Machine Library

Design Overview

We start by defining the combination lock state machine. It has one input x and two outputs **UNLOCK** and **HINT**. The **UNLOCK** output should be 1 if and only if x is 0 and the sequence of inputs received on x in the preceding seven cycles was '0110111'. The **HINT** output should be 1 if and only if the current value of x is the correct one to move the state machine closer to the unlocked state (with **UNLOCK** = 1).

The next state/output table for the state machine is shown below.

Meaning	Current State	If X = 0	If X = 1
Seen nothing	0	1, 01	0, 00
Seen 0	1	1, 00	2, 01
Seen 01	2	1, 00	3, 01
Seen 011	3	4, 01	0, 00
Seen 0110	4	1, 00	5, 01
Seen 01101	5	1, 00	6, 01
Seen 011011	6	4, 00	7, 01
Seen 0110111	7	1, 11	0, 00

Next State/Output Table

Figure 9-2 Table of next states and corresponding outputs

Note: The contents of the columns named 'If x = 0' and 'If x = 1' are formatted in the following way: Next State, UNLOCK HINT.

The table shows the next state and outputs that result from the current state and input x. For example, if the current state is 3 and x is 0, the next state is 4. The transition from state 3 to 4 means we have seen the sequence '0110'. During this transition, the UNLOCK output is 0 and the HINT output is 1, indicating we are one step closer to the unlocked state. The state machine moves on to the next state if we get the correct input and returns to states 0 or 1 if we get an incorrect one. State 6 is an exception; if we get the wrong input, i.e., '0', the previous three inputs might still turn out to be the beginning of the required sequence. Consequently, we go back to state 4 instead of state 1. In state 7, we have received the required sequence, so we set UNLOCK to 1 if x is 0. In each state, we set HINT to 1 for the value of x that moves the state machine closer to state 7.

Implementation

From the above description we see that we must use a Mealy state machine to implement the design, since the outputs depend on both the current state and input. A block diagram of this type of state machine is shown below.

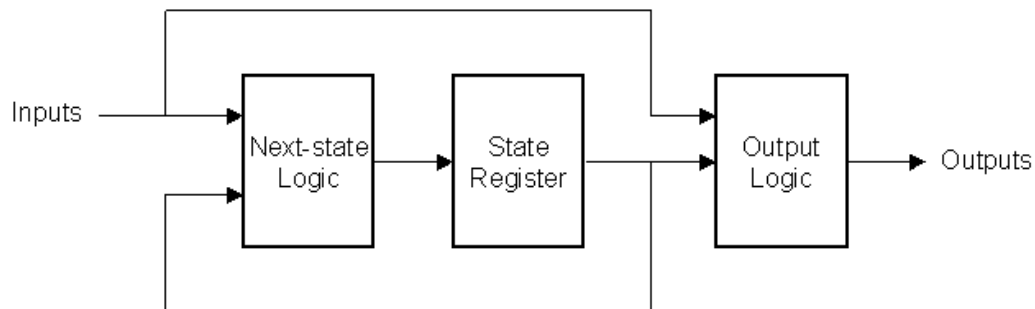


Figure 9-3 Block Diagram of Mealy State Machine

The block is configured with next state and output matrices obtained from the next state/output table discussed above. These matrices are constructed as follows:

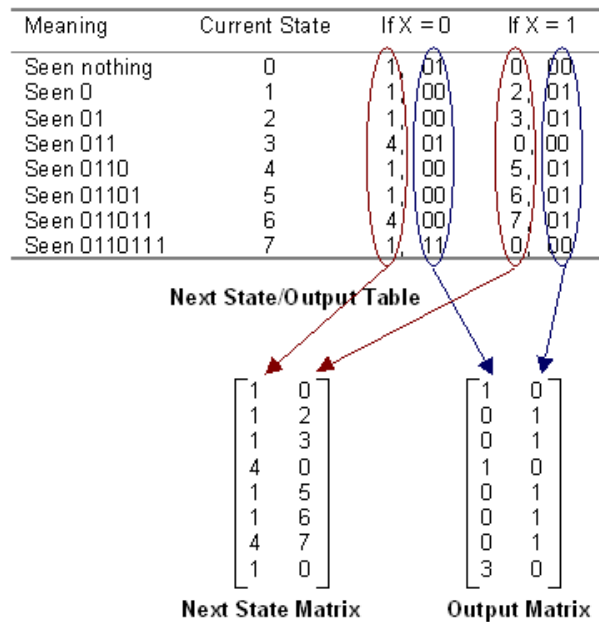


Figure 9-4 Construction of the Next State and Output Matrices

Rows of the matrices correspond to the current state, and columns correspond to the input value. The output of the state machine is an N-bit vector. (In this example N is 2.) An element of the output matrix is a decimal representation of the N-bit output vector for a particular state. In this example, the outputs UNLOCK and HINT are concatenated together to make the elements of the output matrix. For instance, when the state is 7 and x = 1, both UNLOCK and HINT are 1, so the corresponding element of the output matrix is 3 (the decimal representation of binary '11').

The next state logic and state register in this block are implemented with high speed dedicated block RAM. The output logic is implemented using a distributed RAM configured as a lookup table, and therefore has zero latency.

Simulation

Open the Simulink model lock.mdl for this example. You will see the diagram shown below.

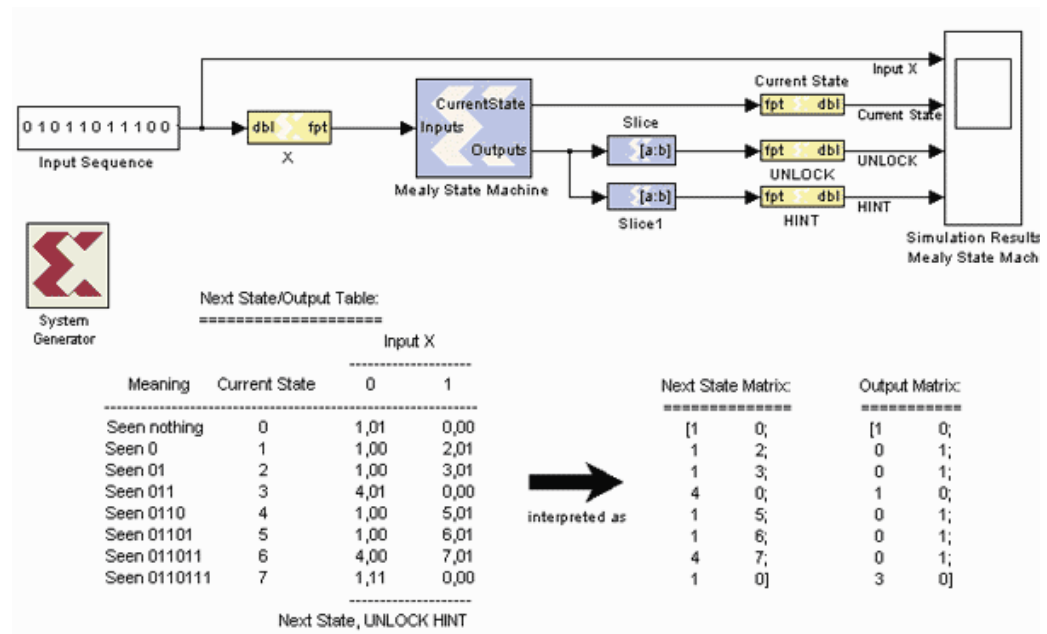


Figure 9-5 Lock Simulink Model

The block's parameters dialog box can be invoked by double-clicking on the Mealy State Machine icon in the model. In this dialog, the next state and output matrices are specified and the number of output bits is set to 2. Two slice blocks are used to extract the UNLOCK and HINT bits from the output.

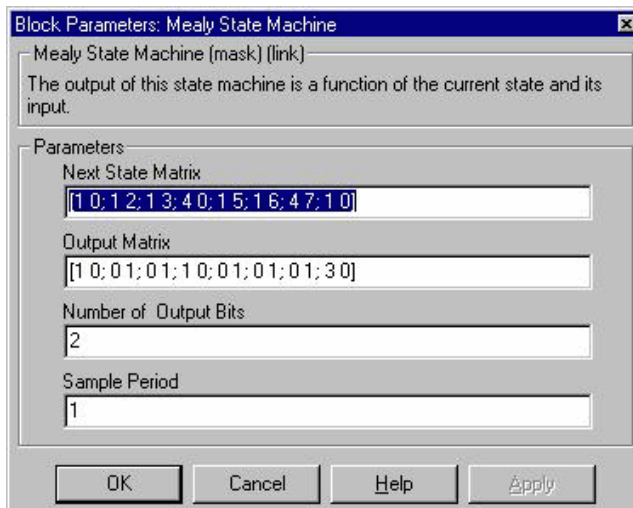


Figure 9-6 Mealy State Machine block parameters dialog box

When the design is simulated using [0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1 1 1 0 0 1 1 0 0 1 1 0 1 1 0 1 1 1] as the input vector, the sequence '0110111' is found at time offsets 20 and 31. The simulation results are shown below.

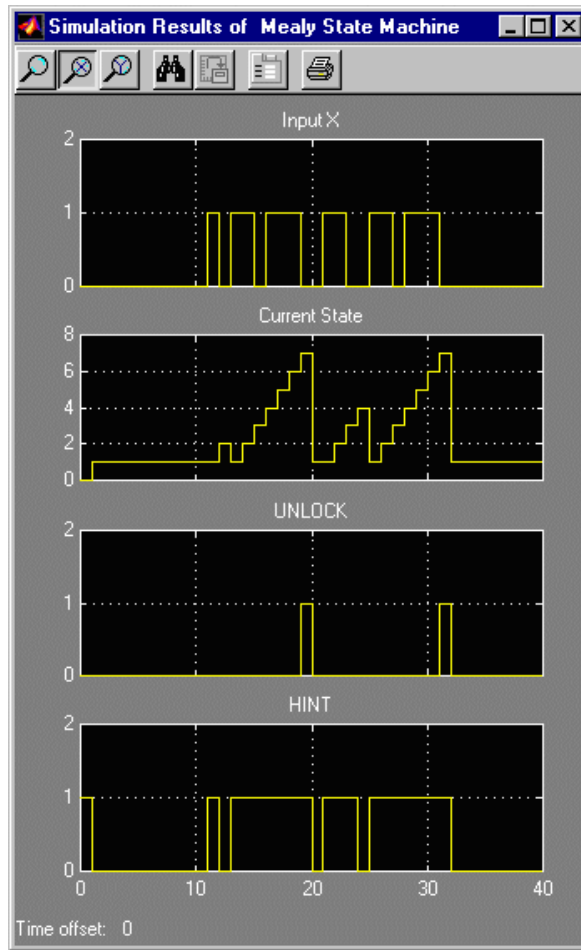


Figure 9-7 Simulation Results

More Fun With the State Machine

A MATLAB function called `calcLockMatrices` is provided and can be used to create a new next state and output matrix to detect a different sequence. To run the function, change your working directory to

```
$MATLAB\toolbox\xilinx\sysgen\examples\state_machine and then type
[next, out] = calcLockMatrices('0110101101111')
```

The next state and output matrices should be specified in the state machine block GUI as the values for 'Next State Matrix' and 'Output Matrix'. You can change the input of the design by modifying the 'Input Sequence' block. Resimulate the design and try to detect the new sequence.

References

This example is based on the "combination lock" state machine presented on pages 399-402 in *Digital Design Principles and Practices* by John F. Wakerly, Prentice Hall, 1990.