

System Generator for DSP

Getting Started Guide

Release 10.1 March, 2008





Xilinx is disclosing this Document and Intellectual Property (hereinafter "the Design") to you for use in the development of designs to operate on, or interface with Xilinx FPGAs. Except as stated herein, none of the Design may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Any unauthorized use of the Design may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes.

Xilinx does not assume any liability arising out of the application or use of the Design; nor does Xilinx convey any license under its patents, copyrights, or any rights of others. You are responsible for obtaining any rights you may require for your use or implementation of the Design. Xilinx reserves the right to make changes, at any time, to the Design as deemed desirable in the sole discretion of Xilinx. Xilinx assumes no obligation to correct any errors contained herein or to advise you of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or technical support or assistance provided to you in connection with the Design.

THE DESIGN IS PROVIDED "AS IS" WITH ALL FAULTS, AND THE ENTIRE RISK AS TO ITS FUNCTION AND IMPLEMENTATION IS WITH YOU. YOU ACKNOWLEDGE AND AGREE THAT YOU HAVE NOT RELIED ON ANY ORAL OR WRITTEN INFORMATION OR ADVICE, WHETHER GIVEN BY XILINX, OR ITS AGENTS OR EMPLOYEES. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DESIGN, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT OF THIRD-PARTY RIGHTS.

IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOST DATA AND LOST PROFITS, ARISING FROM OR RELATING TO YOUR USE OF THE DESIGN, EVEN IF YOU HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE TOTAL CUMULATIVE LIABILITY OF XILINX IN CONNECTION WITH YOUR USE OF THE DESIGN, WHETHER IN CONTRACT OR TORT OR OTHERWISE, WILL IN NO EVENT EXCEED THE AMOUNT OF FEES PAID BY YOU TO XILINX HEREUNDER FOR USE OF THE DESIGN. YOU ACKNOWLEDGE THAT THE FEES, IF ANY, REFLECT THE ALLOCATION OF RISK SET FORTH IN THIS AGREEMENT AND THAT XILINX WOULD NOT MAKE AVAILABLE THE DESIGN TO YOU WITHOUT THESE LIMITATIONS OF LIABILITY.

The Design is not designed or intended for use in the development of on-line control equipment in hazardous environments requiring fail-safe controls, such as in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support, or weapons systems ("High-Risk Applications"). Xilinx specifically disclaims any express or implied warranties of fitness for such High-Risk Applications. You represent that use of the Design in such High-Risk Applications is fully at your risk.

© 2002-2008 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc. PowerPC is a trademark of IBM, Inc. All other trademarks are the property of their respective owners.

Table of Contents

Preface: About This Guide

Guide Contents	7
System Generator PDF Doc Set	7
Additional Resources	8
Conventions	8
Typographical	8
Online Document	9

Chapter 1: Introduction

The Xilinx DSP Block Set	12
FIR Filter Generation	13
Support for MATLAB	14
System Resource Estimation	15
Hardware Co-Simulation	16
System Integration Platform	17

Chapter 2: Installation

Downloading	19
Hardware Co-Simulation Support	19
Installing	19
Software Prerequisites	19
Using the ISE Design Suite Installer	20
Hardware Co-Simulation Installation	20
Compiling Xilinx HDL Libraries	21
Configuring the System Generator Cache	21
Displaying and Changing Versions of System Generator	21

Chapter 3: Release Information

Release Notes 10.1	23
System Generator Enhancements	23
Xilinx DSP Blockset Enhancements	24
Tool Flow and Integration	24
Release Notes 9.2.01	25
System Generator Enhancements	25
Xilinx DSP Blockset Enhancements	29
Tool Flow and Integration	29
Known Issues	29
Release Notes 9.2.00	30
System Generator Enhancements	30
Xilinx DSP Blockset Enhancements	33
Tool Flow and Integration	34
Known Issues	34
Release Notes 9.1.01	35
System Generator Enhancements	35
Xilinx DSP Blockset Enhancements	35
Tool Flow and Integration	36
Migrating Designs Created in Previous Versions of Software	36

Upgrading a Xilinx System Generator Model	37
Upgrading v2.x and Prior Models	37
Upgrading v3.x, v6.x and v7.x Models	37
Examples	38

Chapter 4: Getting Started

Introduction	39
Lesson 1 - Design Creation Basics	40
The System Generator Design Flow	40
The Xilinx DSP Blockset	41
Defining the FPGA Boundary	42
Adding the System Generator Token	43
Creating the DSP Design	44
Generating the HDL Code	45
Model-Based Design using System Generator	46
Creating Input Vectors using MATLAB	47
Lesson 1 Summary	48
Lab Exercise: Using Simulink	48
Lab Exercise: Getting Started with System Generator	48
Lesson 2 - Fixed Point and Bit Operations	49
Fixed-Point Numeric Precision	49
System Generator Fixed-Point Quantization	50
Overflow and Round Modes	51
Bit-Level Operations	52
The Reinterpret Block	53
The Convert Block	54
The Concat Block	55
Slice Block	56
The BitBasher Block	57
Lesson 2 Summary	58
Lab Exercise: Signal Routing	58
Lesson 3 - System Control	59
Controlling a DSP System	59
The MCode Block	60
The Xilinx "xl_state" Data Type	61
State Machine Example	62
The Expression Block	63
Reset and Enable Ports	64
Bursty Data	65
Lesson 3 Summary	66
Lab Exercise: System Control	66
Lesson 4 - Multi-Rate Systems	67
Creating Multi-Rate Systems	67
Up and Down Sampling Blocks	68
Rate Changing Functional Blocks	69
Viewing Rate Changes in Simulink	70
Debugging Tools	71
Sample Period "Rules"	72
Lab Exercise: Multi-Rate Systems	73
Lesson 5 - Using Memories	74
Block vs. Distributed RAM	74
Initializing RAMs and ROMs	75

System Generator RAM Blocks	76
System Generator ROM Blocks	77
The Delay Block	78
The FIFO Block	79
Lab Exercise: Using Memories	80
Lesson 6 - Designing Filters	81
Introduction	81
The Virtex DSP48 Math Slice	82
FIR Compiler Block	83
Creating Coefficients with FDATool	84
Using FDA Tool Coefficients	85
Lab Exercise: Designing Filters	86
Additional Examples and Tutorials	87
Black Box Examples	87
ChipScope Examples	87
DSP Examples	88
M-Code Examples	89
Processor Examples	89
Shared Memory Examples	90
Timing Analysis Examples	91
Miscellaneous Examples	91
System Generator Demos	92
Index	93

About This Guide

This Getting Started Guide introduces you to **System Generator for DSP**, then provides installation and configuration instructions, release information, and six mini-training modules that highlight the main features of the product. Each module starts with a lesson of 8-10 slides that explain important concepts, followed by a lab exercise that take about 30 minutes to complete. Because this introductory training is part of the tool, you can progress through the material at your own pace and on your own time schedule

Guide Contents

This Getting Started Guide contains the following topics:

- Introduction
- Installation
- Release Information
- Getting Started
 - a. Design Creation Basics
 - b. Fixed Point and Bit Operations
 - c. System Control
 - d. Multi-Rate Systems
 - e. Using Memories
 - f. Designing Filters
 - g. Additional Examples and Tutorials

System Generator PDF Doc Set

This Getting Started Guide can be found in the System Generator Help system and is also part of the System Generator Doc Set that is provided in PDF format. The content of the doc set is as follows:

- *System Generator for DSP Getting Started Guide*
- *System Generator for DSP User Guide*
- *System Generator for DSP Reference Guide*

Note: Hyperlinks across these PDF documents work only when the PDF files reside in the same folder. After clicking a Hyperlink in the Adobe Reader, you can return to the previous page by pressing the Alt key and the left arrow key (←) at the same time.

Additional Resources

To find additional documentation, see the Xilinx website at:

<http://www.xilinx.com/literature>.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

<http://www.xilinx.com/support>.

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
Courier font	Messages, prompts, and program files that the system displays	speed grade: - 100
Courier bold	Literal commands that you enter in a syntactical statement	ngdbuild <i>design_name</i>
Helvetica bold	Commands that you select from a menu	File →Open
	Keyboard shortcuts	Ctrl+C
Italic font	Variables in a syntax statement for which you must supply values	ngdbuild <i>design_name</i>
	References to other manuals	See the <i>Development System Reference Guide</i> for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are <i>not</i> connected.
Square brackets []	An optional entry or parameter. However, in bus specifications, such as bus [7 : 0] , they are required.	ngdbuild [<i>option_name</i>] <i>design_name</i>
Braces { }	A list of items from which you must choose one or more	lowpwr = { on off }
Vertical bar	Separates items in a list of choices	lowpwr = { on off }

Convention	Meaning or Use	Example
Vertical ellipsis .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...	Repetitive material that has been omitted	allow block <i>block_name loc1 loc2 ... locn;</i>

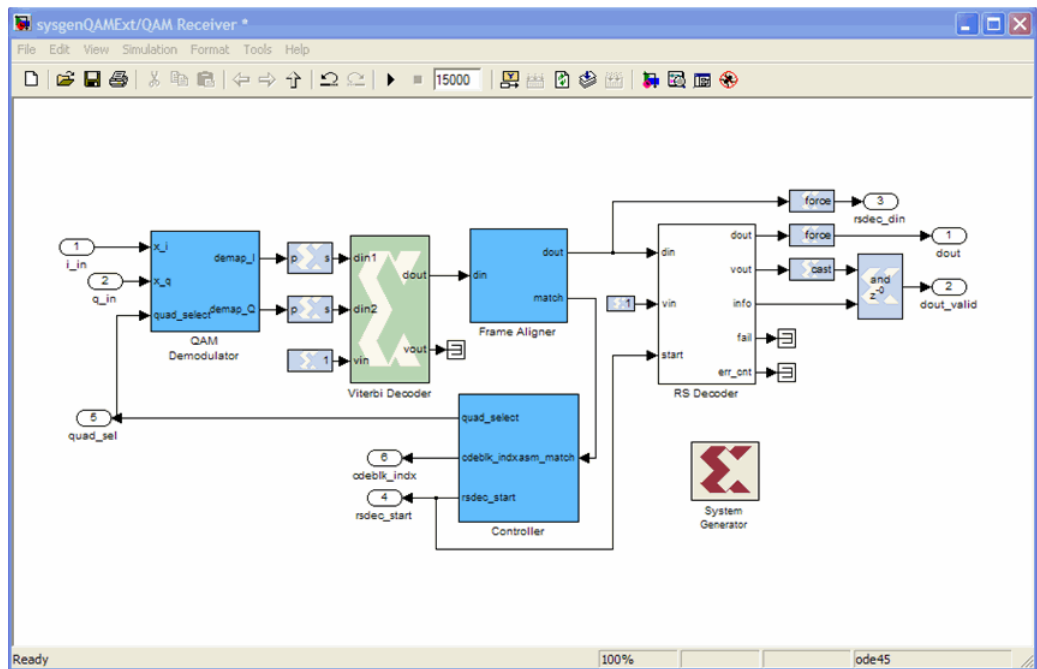
Online Document

The following conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the topic " Additional Resources " for details. Refer to " Title Formats " in Chapter 1 for details.
Red text	Cross-reference link to a location in another document	See Figure 2-5 in the <i>Virtex-II Platform FPGA User Guide</i> .
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

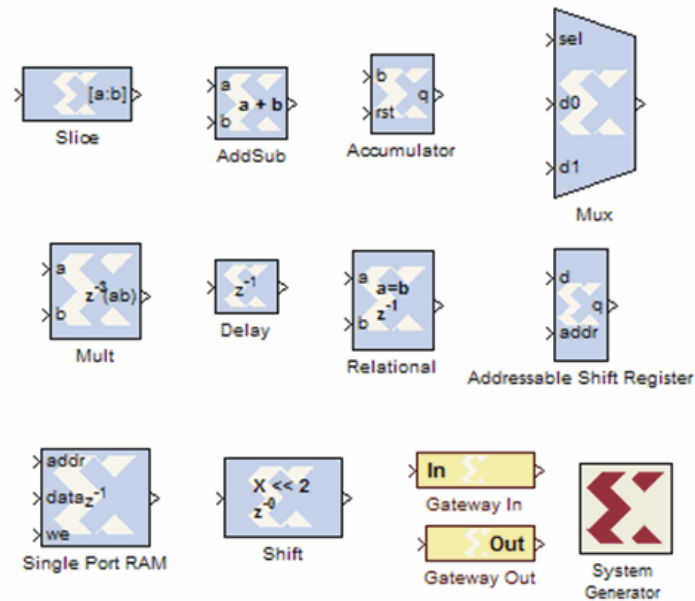
Introduction

System Generator is a DSP design tool from Xilinx that enables the use of The Mathworks model-based design environment Simulink for FPGA design. Previous experience with Xilinx FPGAs or RTL design methodologies are not required when using System Generator. Designs are captured in the DSP friendly Simulink modeling environment using a Xilinx specific blockset. All of the downstream FPGA implementation steps including synthesis and place and route are automatically performed to generate an FPGA programming file.



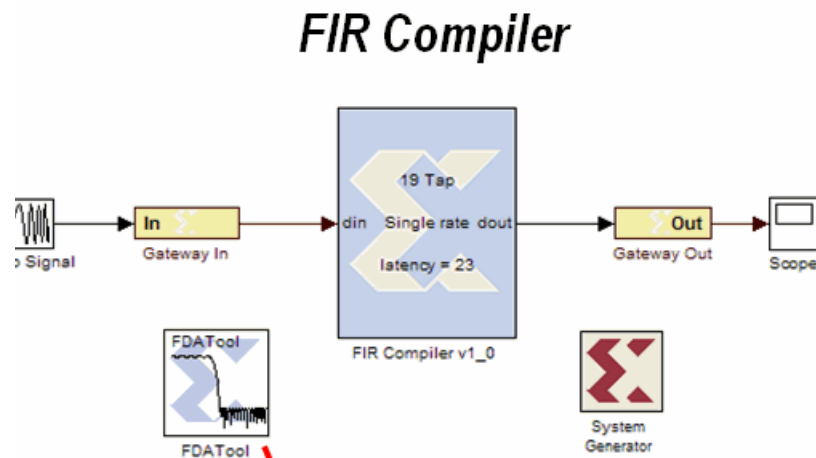
The Xilinx DSP Block Set

Over 90 DSP building blocks are provided in the Xilinx DSP blockset for Simulink. These blocks include the common DSP building blocks such as adders, multipliers and registers. Also included are a set of complex DSP building blocks such as forward error correction blocks, FFTs, filters and memories. These blocks leverage the Xilinx IP core generators to deliver optimized results for the selected device.

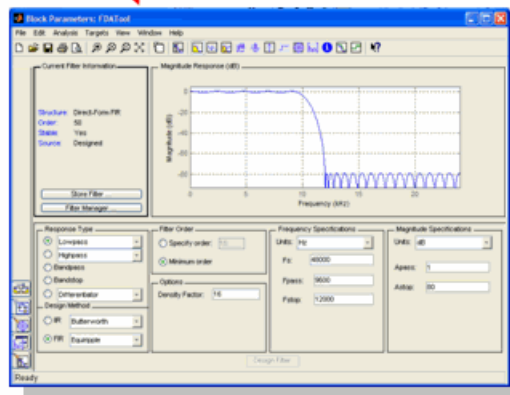


FIR Filter Generation

System Generator includes a FIR Compiler block that targets the dedicated DSP48 hardware resources in the Virtex4 and Virtex5 devices to create highly optimized implementations that can run in excess of 500 Mhz. Configuration options allow generation of direct, polyphase decimation, polyphase interpolation and oversampled implementations. Standard MATLAB functions such as fir2 or The Mathworks FDAtool can be used to create coefficients for the Xilinx FIR Compiler.

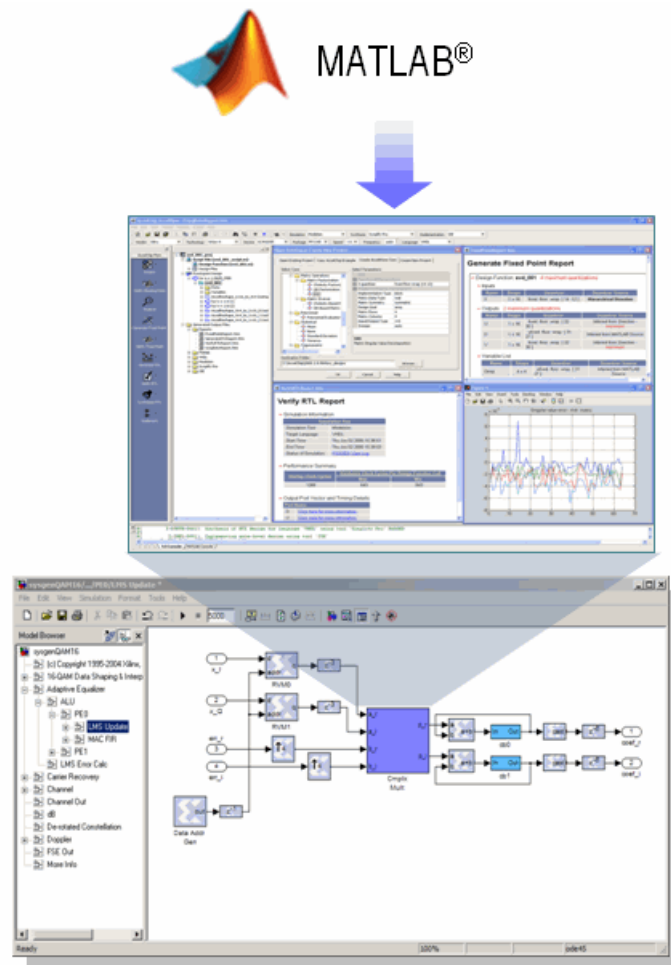


FDA Tool



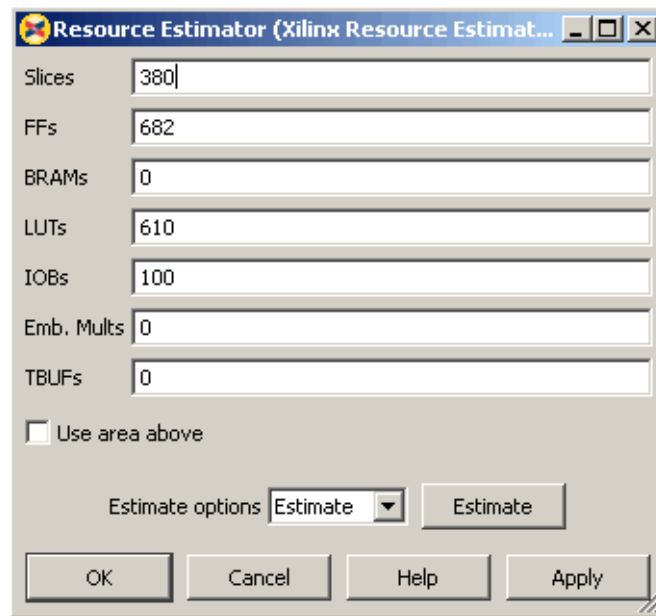
Support for MATLAB

Algorithmic MATLAB models can be incorporated into System Generator through AccelDSP. AccelDSP includes powerful algorithmic synthesis that takes floating-point MATLAB as input and generates a fully scheduled fixed-point model for use with System Generator. Features include floating- to fixed-point conversion, Automatic IP insertion, design exploration and algorithmic scheduling. Also included in System Generator is an MCode block that allows the use of non-algorithmic MATLAB for the modeling and implementation of simple control operations.



System Resource Estimation

System Generator provides a Resource Estimator block that quickly estimates the area of a design prior to place and route. This can be a valuable aid in the hardware / software partitioning process by helping system designers take full advantage of the FPGA resources which include up to 640 multiply/accumulate (or DSP) blocks in the Virtex5 devices.



Resource Estimator (Xilinx Resource Estimat...)

Slices	380
FFs	682
BRAMs	0
LUTs	610
IOBs	100
Emb. Mults	0
TBUFs	0

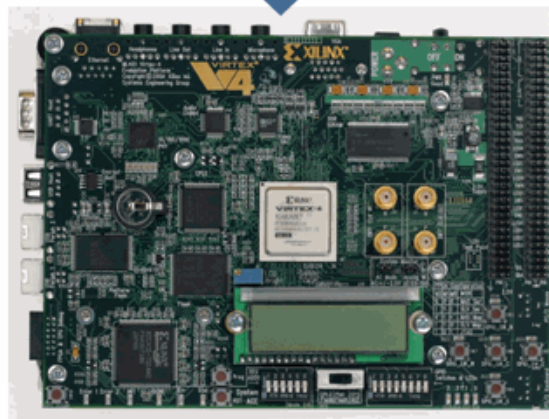
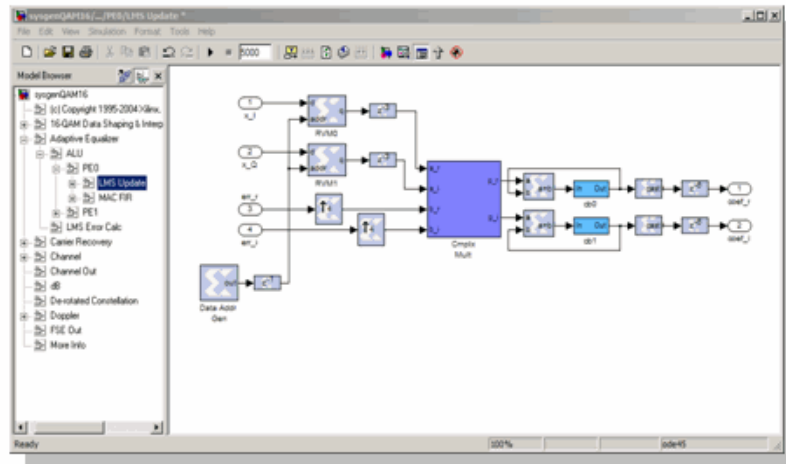
Use area above

Estimate options: Estimate [v] Estimate

OK Cancel Help Apply

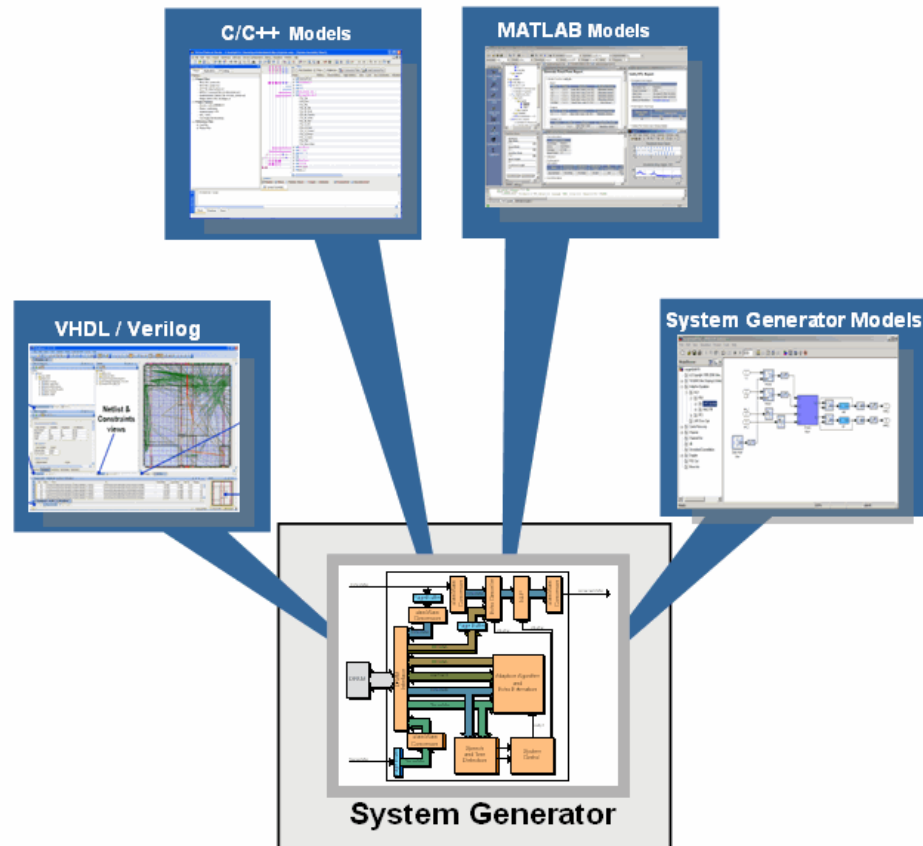
Hardware Co-Simulation

System Generator provides accelerated simulation through hardware co-simulation. System Generator will automatically create a hardware simulation token for a design captured in the Xilinx DSP blockset that will run on one of over 20 supported hardware platforms. This hardware will co-simulate with the rest of the Simulink system to provide up to a 1000x simulation performance increase.



System Integration Platform

System Generator provides a system integration platform for the design of DSP FPGAs that allows the RTL, Simulink, MATLAB and C/C++ components of a DSP system to come together in a single simulation and implementation environment. System Generator supports a black box block that allows RTL to be imported into Simulink and co-simulated with either ModelSim or Xilinx ISE Simulator. System Generator also supports the inclusion of a MicroBlaze embedded processor running C/C++ programs.



Installation

Downloading

System Generator is only available via download from the Xilinx web page. You may purchase, register, and download the System Generator software from the site at:

http://www.xilinx.com/ise/optional_prod/system_generator.htm

Note: In special circumstances, System Generator can be delivered on a CD. Please contact your Xilinx distributor if your circumstances prohibit you from downloading the software via the web.

Hardware Co-Simulation Support

If you have an FPGA development board, you may be able to take advantage of System Generator's ability to use FPGA hardware co-simulation with Simulink simulations. The System Generator software includes support for the XtremeDSP Development Kit, the MicroBlaze Multimedia Demonstration boards, the MVI hardware platform, the ML402 Virtex-4 platform, the ML506 Virtex-5 platform, and the Spartan-3A DSP 1800 starter platform and 3400 development platform. Additional System Generator board support packages provide support for additional hardware co-simulation platforms. System Generator board support packages can be downloaded from the following URL:

http://www.xilinx.com/technology/dsp/thirdparty_devboards.htm

Installing

Software Prerequisites

You must have the following software installed before running System Generator.

- One of the following versions of MATLAB from The MathWorks Inc.:
- MATLAB v7.4/Simulink v6.6 (R2007a)
- MATLAB v7.5/Simulink v7.0 (R2007b)
Note: MATLAB must be installed in a directory with no spaces (e.g., C:\MATLAB\R2007a).
- Xilinx ISE Foundation version 10.1

Some features in System Generator require the following software to be installed:

- A logic synthesis tool. System Generator is fully compatible with Xilinx XST (included in the ISE Foundation bundle) and Synplify Pro v8.6.2 or v8.9 from Synplicity, Inc.
- A hardware description language (HDL) simulator is required only for co-simulating HDL modules within Simulink using System Generator. System Generator HDL co-simulation interfaces are compatible with the Xilinx ISE Simulator, ModelSim Xilinx

Edition MXE (an option with ISE Foundation), and ModelSim PE or SE, version v6.3c, from Model Technology Inc.

Note: The Microsoft Windows environment variable \$XILINX must be set and point to your ISE software installation directory.

ISE software service packs may be downloaded from the Xilinx Download Center:

http://www.xilinx.com/xlnx/xil_sw_updates_home.jsp

Using the ISE Design Suite Installer

Before invoking the ISE Design Suite Installer, it is a good idea to make sure that all instances of MATLAB are closed. When all instances of MATLAB are closed, launch the installer and follow the directions on the screen.

Choose MATLAB Version for System Generator

As the last step of the System Generator installation, click the check box of the MATLAB installation you wish to associate with this version of System Generator, then click **Apply**.

If you don't see a valid version of MATLAB listed, for example a version installed on a network device, click the **Add Version** button, browse to the MATLAB root directory of the unlisted version, then click **Add**. If you wish to associate this version of MATLAB with System Generator, click the check box of the newly listed MATLAB installation, then click **Apply**.

If you have no version of MATLAB available, click **Choose Later** to continue with the installation. At a later time, after you have installed MATLAB, you can associate that version of MATLAB with System Generator by executing the Windows menu item **Start > All Programs > Xilinx ISE Design Suite 10.1 > DSP Tools > Select MATLAB version for Xilinx System Generator**.

Hardware Co-Simulation Installation

This topic provides links to hardware and software installation procedures for hardware co-simulation. If you do not plan to use hardware co-simulation, you may skip this topic.

Ethernet-Based Hardware Co-Simulation

[Installing an ML402 Board for Ethernet Hardware Co-Simulation](#)

[Installing an ML506 Board for Ethernet Hardware Co-Simulation](#)

[Installing a Spartan-3A DSP 1800A Starter Platform for Ethernet Hardware Co-Simulation](#)

[Installing a Spartan-3A DSP 3400A Development Platform for Ethernet Hardware Co-Simulation](#)

JTAG-Based Hardware Co-Simulation

[Installing an ML402 Board for JTAG Hardware Co-Simulation](#)

Third-Party Hardware Co-Simulation

As part of the Xilinx XtremeDSP™ Initiative, Xilinx works with distributors and many OEMs to provide a variety of DSP prototyping and development platforms. Please refer to the following Xilinx web site page for more information on available platforms:

http://www.xilinx.com/technology/dsp/thirdparty_devboards.htm

Compiling Xilinx HDL Libraries

If you intend to simulate System Generator designs using ModelSim, you must compile your IP (cores) libraries. This topic describes the procedure.

ModelSim (PE or EE/SE)

The Xilinx tool that compiles libraries for use in ModelSim PE or EE/SE is named `complib`. The following command can, for example, be used to compile all the VHDL and Verilog libraries with ModelSim SE:

```
complib -s mti_se -f all -l all
```

Complete instructions for running `complib` can be found in the ISE Software Manual titled "Synthesis and Simulation Design Guide".

MXE Libraries

If you plan to use ModelSim XE (Xilinx Edition), download the MXE precompiled libraries from the Xilinx web site. Unzip these MXE libraries into the directory in which you have MXE installed, e.g., `c:/Modeltech_XE/`. This is the location where MXE expects to find your Xilinx compiled libraries, so it is not necessary to change your `modelsim.ini` file. This file should point to the correct installed location.

Configuring the System Generator Cache

Both the System Generator simulator and the design generator incorporate a disk cache to speed up the iterative design process. The cache does this by tagging and storing files related to simulation and generation, then recalling those files during subsequent simulation and generation rather than rerunning the time consuming tools used to create those files.

Setting the Size

By default, the cache will use up to 500 MB of disk space to store files. To specify the amount of disk space the cache should use, set the `SYSGEN_CACHE_SIZE` environment variable to the size of the cache in megabytes. Set this number to a higher value when working on several large designs.

Setting the Number of Entries

The cache entry database stores a fixed number of entries. The default is 20,000 entries. To set size of the cache entry database, set the `SYSGEN_CACHE_ENTRIES` environment variable to the desired number of entries. Setting this number too small will adversely affect cache performance. Set this number to a higher value when working on several large designs.

Displaying and Changing Versions of System Generator

It is possible to have several versions of System Generator installed. The MATLAB command `xlVersion` displays which versions are installed, and makes it possible to switch from one to another. `xlVersion` is useful when upgrading a model to run in the latest version of System Generator.

Entering "`xlVersion`" in the MATLAB console displays the versions of System Generator that are installed, and entering "`xlVersion <version>`" switches to the specified

version. For example, if the versions that are installed are 9.2.01 and 10.1, and the currently selected version is 10.1, then entering "xlVersion" displays

```
Available System Generator installations:  
Version 9.2.01 in C:/Xilinx/9.2.01/DSP_Tools/sysgen  
Version 10.1 in C:/Xilinx/10.1/DSP_Tools/sysgen  
Current version of System Generator is 10.1.
```

Entering "xlVersion 9.2.01" switches the System Generator version to 9.2.01.

Occasionally, it is necessary to restart MATLAB to make it possible to switch. In this case, the response to entering "xlVersion 10.1" looks like the following:

```
Please restart MATLAB and run xlVersion 10.1 again to switch.
```

When the switch succeeds, xlVersion prints the following:

```
Your System Generator has been switched. Please restart MATLAB.
```

If you install System Generator 10.1 after you install 9.2.01, you need to install 10.1 again in order to make xlVersion work.

Once you switch System Generator version, you need to switch to the right version of ISE in order to make System Generator work correctly.

Release Information

Release Notes 10.1

System Generator Enhancements

System Generator / Project Navigator Integration

System Generator designs can now be more easily incorporated into a larger design inside of Project Navigator by using a new source type in Project Navigator. The System Generator design can also be launched from Project Navigator.

DCM Support

System Generator now provides the option to automatically include a DCM in a design. Although the optional DCM is abstracted away from the designer, the generated design will leverage DCMs available in the silicon.

An alternative option exposes the clock ports at the top level for manual connection to a DCM.

Dual Asynchronous-Clock Support for PLB46

This capability gives the designer additional flexibility by allowing the DSP and embedded processing portions of a design to run at different clock rates.

Run Time Speed Improvements

- Up to 2x faster first time initialization of a simulation
- >10x faster initialization when loading the Xilinx Blockset in the Simulink Library Browser

M-Based HW Co-Simulation

System Generator models compiled for HW Co-Simulation can now be embedded, configured and utilized in a MATLAB M-code script; allowing for calls into hardware to be made from MATLAB.

Xilinx DSP Blockset Enhancements

FFT 5.0

Update to existing block which now includes cyclic prefix insertion

FIR Compiler 3.2

Update which now include support for Virtex II and Spartan 3A.

Reset Generator

New block that produces synchronized downsampled reset signals which eliminates the need to manually create these signals.

CIC Compiler 1.1

New block now available in System Generator.

Tool Flow and Integration

System Generator 10.1 is compatible with the following tools:

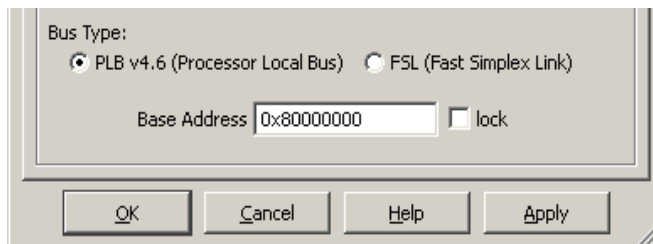
Tool	Version
The Mathworks MATLAB® and Simulink	2007a and 2007b
Mentor Graphics ModelSim® SE	6.3c
Synplicity Synplify Pro®	8.8.0.4 (Requires a floating license for Hardware Co-Simulation)
Xilinx® AccelDSP	10.1
Xilinx® ChipScope Pro	10.1
Xilinx® EDK	10.1
Xilinx® ISE	10.1
Xilinx® ISE IP Update	10.1 IP Update 1
Xilinx® ISE Simulator	10.1

Release Notes 9.2.01

System Generator Enhancements

EDK 9.2 SP1, PLBv46 & MB 7.0 Support

This release of System Generator supports Embedded Design Kit (EDK) Release 9.2SP1 and the MicroBlaze PLB v4.6 bus interface. As shown in the figure below, when you export a System Generator design as a MicroBlaze processor core (pcore) using the EDK Processor block, you can choose to connect the pcore to the MicroBlaze v4.6 bus or the previously supported FSL (Fast Simplex Link). The PLB v4.6 is now the default choice.



When you select the PLB v4.6 option, the target MicroBlaze processor must have a PLB v4.6 bus properly connected to the DPLB interface and a *proc_sys_reset* module connected to the system reset pin. Also, both the pcore PLB memory map and the PLB bus should run at the same operating frequency. These requirements will be in place if you use the *XPS Base System Builder* to build the MicroBlaze processor.

Specifying a PLB v4.6 Base Address When you select the **PLB v4.6 (Processor Local Bus) option**, the bus address space will be automatically adjusted and minimized. If you know where you want the bus address space to start, you enter the address and click **Lock**. Otherwise, the base address will be automatically determined for you. This Base Address option is not used with the FSL Bus Type.

Note: Software simulation is disabled in this release. You can simulate the processor subsystem within System Generator and Simulink by connecting a supported hardware platform to your host computer and using Hardware Co-Simulation. See the topic [Using Hardware Co-Simulation](#) for details.

Pcore Export Enhancements

The EDK Export Tool has been enhanced to provide the following new features:

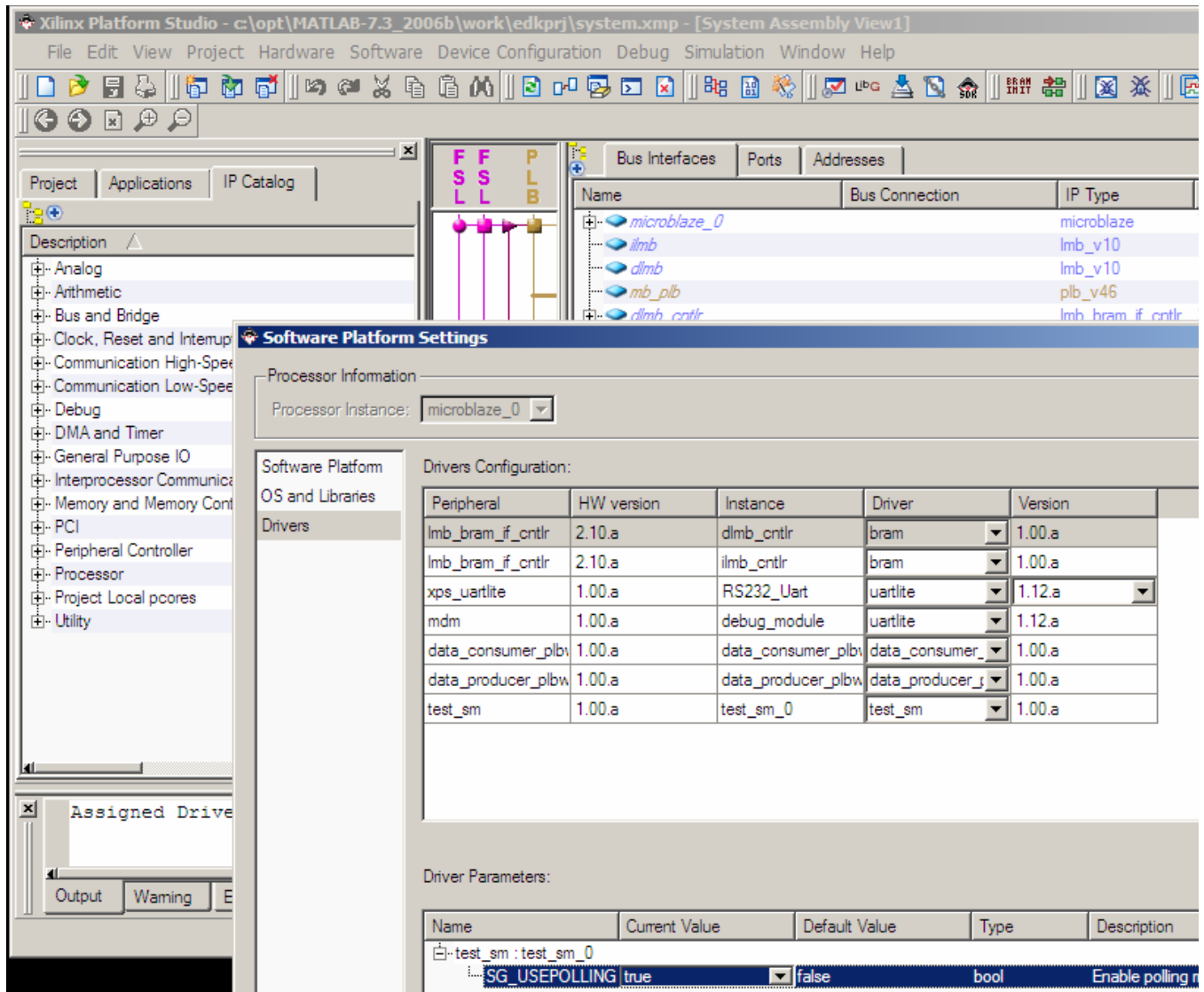
Asynchronous Software Drivers for FSLs

FSL-based pcores connect to the MicroBlaze processor using FIFOs. It is possible for users in XPS, to connect one clock to the processor and another to the System Generator pcore. In the past, if you did this there was a chance that data would be lost.

In the release, you can set your software drivers to work in *polling mode*. Polling drivers will keep retrying (for a programmable number of times) to read or write data. Polling drivers are less efficient, but they are tolerant to clock differences between the Processor and Pcore.

To configure the software driver in Xilinx Platform Studio (XPS), you select the menu **Software > Software Platform Settings** to get the following dialog box shown below. You

click on **Drivers** and if Sysgen-based FSL pcores are available, driver parameters will be present as shown below. Select **SG_POLLING** to be true to enable polling drivers.



The screenshot shows the Xilinx Platform Studio interface with the **Software Platform Settings** dialog box open. The **Drivers** tab is selected, displaying a table of driver configurations. The **SG_USEPOLLING** parameter is highlighted in blue.

Peripheral	HW version	Instance	Driver	Version
lmb_bram_if_cntrl	2.10.a	dlmb_cntrl	bram	1.00.a
lmb_bram_if_cntrl	2.10.a	ilmb_cntrl	bram	1.00.a
xps_uartlite	1.00.a	RS232_Uart	uartlite	1.12.a
mdm	1.00.a	debug_module	uartlite	1.12.a
data_consumer_plbv	1.00.a	data_consumer_plbv	data_consumer_	1.00.a
data_producer_plbw	1.00.a	data_producer_plbw	data_producer_	1.00.a
test_sm	1.00.a	test_sm_0	test_sm	1.00.a

Name	Current Value	Default Value	Type	Description
test_sm : test_sm_0	SG_USEPOLLING true	false	bool	Enable polling n

Export as Pcore Under Development

This feature works for both FSL- and PLB-based pcore export. When a pcore is marked as **Pcore under development**, XPS will not cache the HDL produced for this pcore. This is useful when you are developing pcores in System Generator and testing them out in XPS. You can just enable this checkbox, make changes in System Generator and compile in XPS. XPS always compiles the generated pcore, so you don't have to empty the XPS cache which may contain caches of other peripherals, thus slowing down the compile of the final bitstream.

Enable Custom Bus Interfaces

This feature works for both FSL- and PLB-based pcore export and allows you to create custom bus interfaces that will be understood in XPS.

Improved EDK Processor Bitstream Support

When you perform bitstream compilation on a System Generator design with an EDK Processor block, the imported EDK project and the shared memories sitting between the System Generator design and MicroBlaze processor are netlisted along with the System Generator design and included in the resulting bitstream.

System Generator also attempts to compile any active software programs inside the imported EDK project. If the compilation of active software programs succeeds, System Generator invokes the **data2bram** utility to include the compiled software programs into the resulting bitstream.

Note: No error or warning message is issued when System Generator encounters failures during software program compilation or when System Generator updates the resulting bitstream with the compiled software programs.

Once the bitstream is generated, you can modify the software programs in the imported EDK project and use the following command to compile the software programs, and update the System Generator bitstream without re-running Place & Route:

```
xlProcBlockCallbacks('updatebitstream', [], xmp_file, bit_file, bmm_file);
```

where

- **xmp_file** is the pathname to the imported EDK project file
- **bit_file** is the pathname to the Sysgen bitstream file
- **bmm_file** is the pathname of the back-annotated BMM file produced by Sysgen during bitstream compilation

If the imported EDK project contains a BMM file named `imported_edk_project.bmm`, System Generator creates a back-annotated BMM file named `imported_edk_project_bd.bmm`. You should provide the later back-annotated BMM file to the above command in order to update the bitstream properly.

Spartan-3A DSP 1800A Starter Platform Support

System Generator now supports the Spartan-3A DSP 1800A Starter Platform for Ethernet Point-to-Point Hardware Co-Simulation.

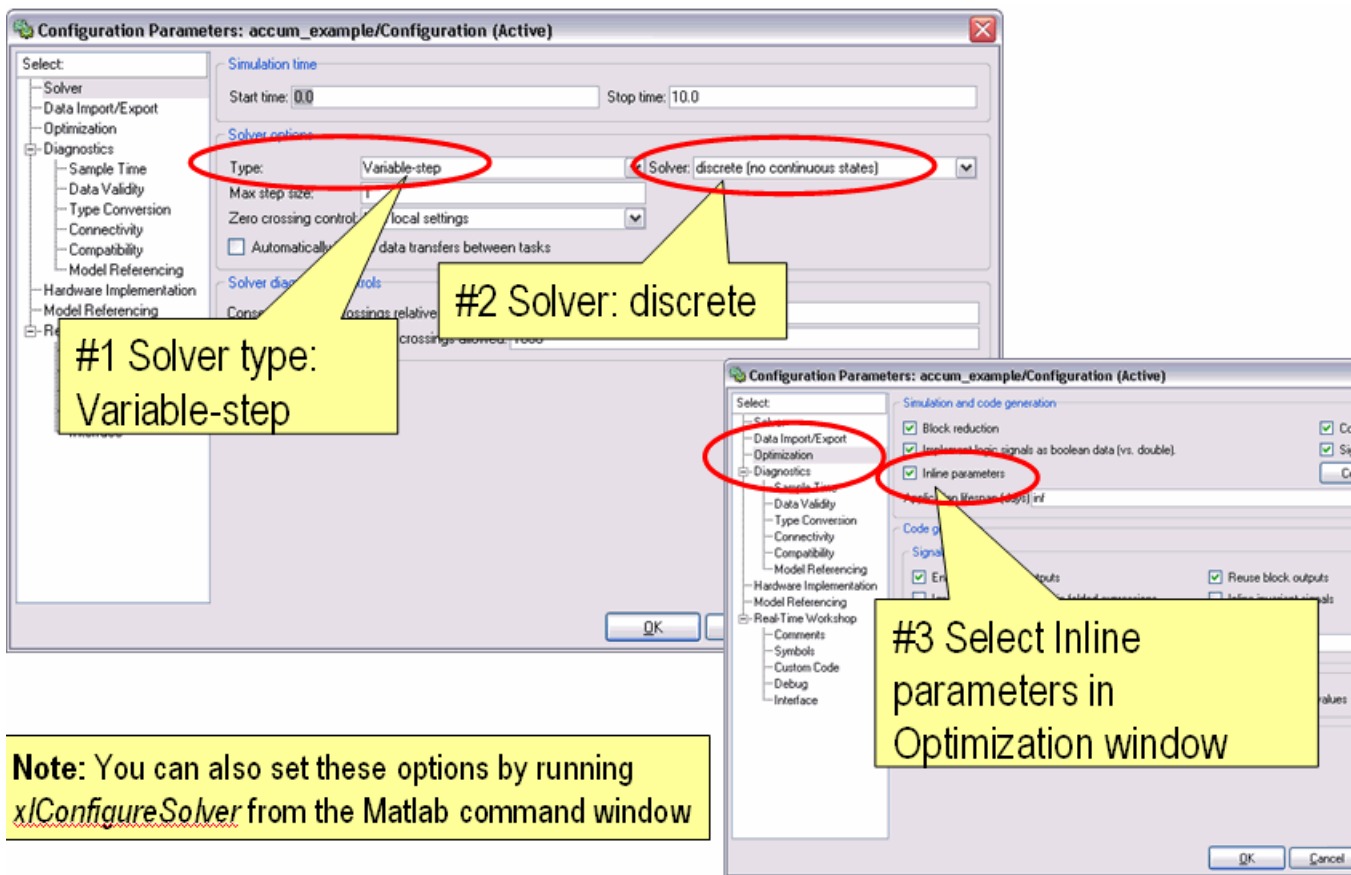
Spartan-3A DSP 3400A Development Platform Support

System Generator now supports the Spartan-3A DSP 3400A Development Platform for Hardware Co-Simulation. Both the Point-to-point Ethernet configuration as well as the Network-Based Ethernet configuration are supported.

Simulation Speed Improvements

Simulation speed for designs created with the Xilinx DSP Blockset has been improved substantially in System Generator 9.2.01. This speedup is relative to the number of blocks in the design with large designs over 2000 blocks showing a 5x to 10x run time improvement. The following Simulink solver settings must be used to achieve the

maximum simulation performance improvement. These solver settings can be automatically configured by typing the “xlConfigureSolver” command at the MATLAB console.



RTL Improvements

In this release, if two or more Simulink Sub-systems are exactly the same, including input data types and mask parameters, only a single VHDL entity or Verilog module will be generated, then instantiated multiple times in the HDL code generated by System Generator. This optimization improves HDL logic synthesis runtime while reducing the amount of memory consumed by XST.

New JTAG Cable Sharing Option

A new option called **Shared cable for concurrent access** has been added to the properties dialog box for the JTAG Co-Simulation block. This option allows the JTAG cable to be shared with EDK XMD during a JTAG co-simulation.

When the option is checked, the JTAG co-simulation engine only acquires a lock on the cable access and then immediately releases the lock when the access completes. Otherwise, the JTAG co-simulation engine holds the lock throughout the simulation. Due to the significant overhead on locking and unlocking the cable, this cable sharing option is disabled by default and only enabled when you check the box.

Xilinx DSP Blockset Enhancements

Convolutional Encoder v6.1

- Support has been added for Spartan-3A DSP

Reed-Solomon Decoder v6.1

- Support has been added for Spartan-3A DSP
- The Field Polynomial entry in the GUI is now entered as a decimal number rather than a binary string.

Reed-Solomon Encoder v6.1

- Support has been added for Spartan-3A DSP
- The Field Polynomial entry in the GUI is now entered as a decimal number rather than a binary string.

Viterbi Decoder v6.1

- Support has been added for Spartan-3A DSP

Tool Flow and Integration

System Generator 9.2.01 is compatible with the following tools:

Tool	Version
The Mathworks MATLAB® and Simulink	2006b and 2007a
Mentor Graphics ModelSim® SE	6.1f
Synplicity Synplify Pro®	8.8.0.4 (Requires a floating license for Hardware Co-Simulation)
Xilinx® AccelDSP	9.2.01
Xilinx® ChipScope Pro	9.2.03i
Xilinx® EDK	EDK 9.2 SP1
Xilinx® ISE	9.2.03i
Xilinx® ISE IP Update	9.2i IP Update 1
Xilinx® ISE Simulator	9.2.03i

Known Issues

Known issues with this release can be found on the Xilinx web site at the following address:

http://www.xilinx.com/xlnx/xil_ans_display.jsp?getPagePath=29595

Release Notes 9.2.00

System Generator Enhancements

Single DSP Tools Installer using Xilinx Unified Installer

All Xilinx DSP Tools now use a Unified Installer for software installation and configuration.

System Generator MATLAB Selector

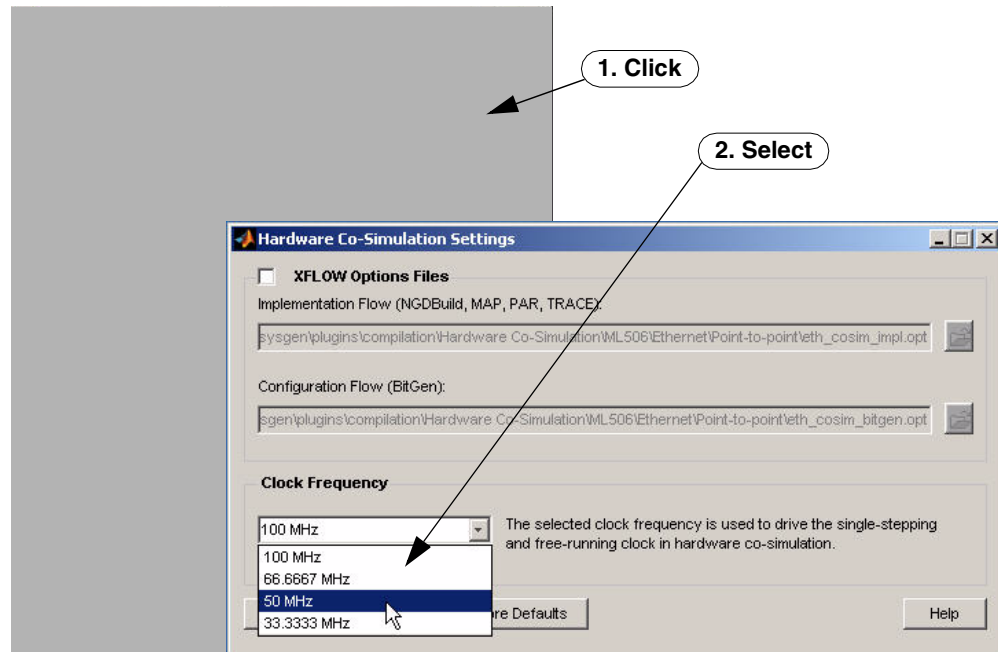
As part of the unified installation process, this configuration feature allows the user additional flexibility to specify which version of MATLAB/Simulink should be associated with a particular version of System Generator. This is helpful for users who have more than one version of MATLAB/Simulink or System Generator on their machine.

Selectable Block Frequency for Hardware Co-Simulation

If you are using a Xilinx ML402 or ML506 platform, at netlist time System Generator allows you to choose a clock frequency for the target design that is equal to or less than the system clock frequency. The following table outlines the frequencies that are available:

Platform	Interface	System Clock Frequency	Available Frequencies
Xilinx ML402	JTAG, Point-to-point Ethernet, Network-based Ethernet	100 MHz	100 MHz 66.7 MHz 50 MHz 33.3 MHz
Xilinx ML506	Point-to-point Ethernet, Network-based Ethernet	200 MHz	100 MHz 66.7 MHz 50 MHz 33.3 MHz

As shown below, you set the target clock frequency at compilation time by clicking the **Settings** button on the System Generator block dialog box, then select the frequency in the pulldown menu.



Clock Enable Fanout Reduction

A new mapping algorithm has been implemented that uses register duplication and placement based on recursive partitioning of loads on high fanout nets. This means improved FMAX on System Generator designs with large CE fanout.

Although this feature is enabled in System Generator by default, the fanout reduction occurs downstream during the ISE mapping operation and the following MAP options must be turned on:

- **Perform Timing-Driven Packing and Placement:** on
- **Map Effort Level:** High
- **Register Duplication:** on

If you are using the ISE Project Navigator flow, these MAP options are also on by default. However, if you are using the default System Generator netlisting flow, you must turn these MAP options on by modifying the bitstream .opt file or by providing your own .opt file.

Shared Memory Stitching

Starting with this release, if two **Shared Memory** blocks with the same name exist anywhere in the design hierarchy, then during netlisting, the two blocks will be stitched into a single **Shared Memory** block. If a single **Shared Memory** block exists, then the input and output ports of the block are pushed to the top-level of the design.

If more than two **Shared Memory** blocks with the same name exist, then an error occurs.

This new stitching feature also applies to **To FIFO** and **From FIFO** block pairs with the same name and **To Register** and **From Register** block pairs with the same name.

For backward compatibility, you can set the MATLAB global variable `xlSgSharedMemoryStitch` to “off” to bring System Generator back to the netlisting behavior before the 9.2 release. For example, from the MATLAB command line, enter the following:

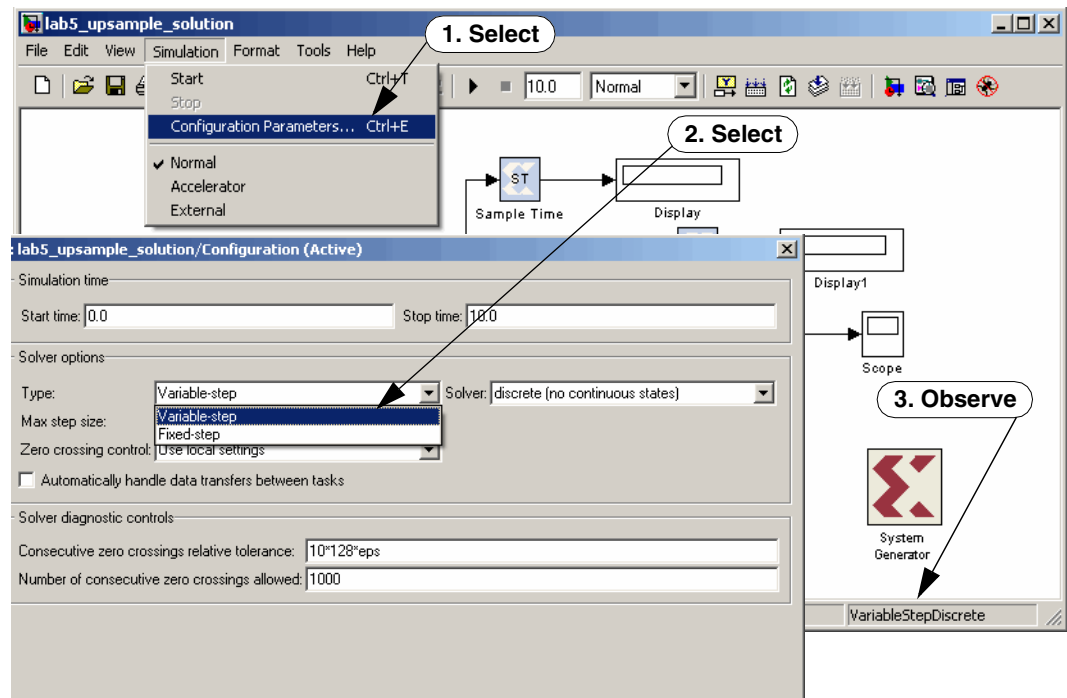
```
global xlSgSharedMemoryStitch;
xlSgSharedMemoryStitch = 'off';
```

System Generator Requires a Simulink Variable-step Solver

Simulink divides simulation solvers into two types: fixed-step and variable-step. Both types of solvers compute the next simulation time as the sum of the current simulation time and a quantity known as the step size. With a fixed-step solver, the step size remains constant throughout the simulation. By contrast, with a variable-step solver, the step size can vary from step to step, depending on the model's dynamics. In particular, a variable-step solver reduces the step size when a model's states are changing rapidly to maintain accuracy and increases the step size when the system's states are changing slowly in order to avoid taking unnecessary steps.

System Generator requires that you use a **Variable-step** solver for every Simulink simulation. The use of a **Fixed-step** solver is not supported and will be explicitly disallowed in a future release. Using a **Fixed-step** solver might result in an inaccurate System Generator simulation.

As shown below, to choose a **Variable-step** solver, select **Simulation > Configuration Parameters...** from the Simulink pulldown menu, then choose **Variable-step** from the Solver options.



Xilinx DSP Blockset Enhancements

DDS Compiler v2_0

- Supports LogiCORE DDS Compiler 2.0
- Support has been added for Spartan-3A DSP
- New controls have been introduced to expose the **rfd** and **rdy** output ports. This feature has also been added to DDS Compiler v1.1
- **channel** input port has been renamed **channelse1** to avoid confusion with the channel output port
- Core initialization times have been significantly reduced both for DDS Compiler v1.1 and DDS Compiler v2.0

FIR Compiler v3_1

- Supports LogiCORE FIR Compiler 3.1
- Faster simulation speeds when using re-loadable coefficients
- Support for Rounding in the FIR Compiler core through the following new parameters on the Advanced Pane of the Parameters dialog box:
 - ◆ **Rounding mode:** List box with the following options:
 - Full_Precision
 - Truncated_LSBS
 - Non_Symmetric_Rounding_Down
 - Non_Symmetric_Rounding_Up
 - Convergent_Rounding_To_Even
 - Convergent_Rounding_To_Odd
 - Symmetric_Rounding_To_Zero
 - Symmetric_Rounding_To_One
 - Symmetric_Rounding_To_Infinity
 - ◆ **Output Width:** An editbox specifying the output width which is activated only if the Rounding mode is set to a value other than Full_Precision
 - ◆ **Allow Rounding Approximation:** Check box that specifies if approximations can be allowed to save resources when using symmetric rounding

Help System Improvements

The System Generator Help System has gone through a major upgrade that includes a new HTML browser, a new comprehensive index and full text search capability.

Tool Flow and Integration

System Generator 9.2.00 is compatible with the following tools:

Tool	Version
The Mathworks MATLAB® and Simulink	2006b and 2007a
Mentor Graphics ModelSim® SE	6.1f
Synplicity Synplify Pro®	8.8.0.4 (Requires a floating license for Hardware Co-Simulation)
Xilinx® AccelDSP	9.2.00
Xilinx® ChipScope Pro	9.2.02i
Xilinx® EDK	EDK support is not available in this release of System Generator, and is expected to return in the next service pack.
Xilinx® ISE	9.2.02i
Xilinx® ISE IP Update	9.2i IP Update 1
Xilinx® ISE Simulator	9.2.02i

Known Issues

Known issues with this release can be found on the Xilinx web site at the following address:

http://www.xilinx.com/xlnx/xil_ans_display.jsp?getPagePath=29110

Release Notes 9.1.01

System Generator Enhancements

New Technologies Supported

Spartan™-3A DSP. The new Spartan™-3A DSP series targets cost-sensitive, high-performance signal processing applications.

Xilinx DSP Blockset Enhancements

FIR Compiler v3_0

- Supports LogiCORE FIR Compiler 3.0
- Support has been added for Spartan-3A DSP
- Supports interpolated filter implementations
- Maximum number of channels increased to 64
- Maximum integer rate change increased to 64
- Now exploits symmetry when interpolating by an even rate with an odd number of coefficients, reducing resource utilization

DDS Compiler v1_1

- Supports LogiCORE DDS Compiler 1.1
- Support added for Virtex™-5 and Spartan™-3A

DSP48 Macro

- Support added for DSP48A slice.
- The DSP48 Macro block provides a device independent abstraction for the DSP48, DSP48E and DSP48A slices.

DSP48A

- Supports DSP48A slice
- The DSP48A slice is unique to the Spartan™-3A DSP family of FPGAs. The DSP48A slices support many independent functions, including multiplier, multiplier accumulator (MACC), preadder/subtractor followed by a multiply-accumulator, multiplier followed by an adder, wide bus multiplexers, magnitude comparator, or wide counter.

Tool Flow and Integration

System Generator is compatible with the following tools:

Tool	Version
The Mathworks MATLAB® and Simulink	2006a, 2006b
Mentor Graphics ModelSim® SE	6.1f
Synplicity Synplify Pro®	8.6.2 (Requires a floating license for Hardware Co-Simulation)
Xilinx® AccelDSP	9.1.01
Xilinx® ChipScope Pro	9.1.03i
Xilinx® EDK	9.1.01i
Xilinx® ISE	9.1.03i
Xilinx® ISE IP Update	9.1i IP Update 2
Xilinx® ISE Simulator	9.1.03i

Migrating Designs Created in Previous Versions of Software

You must update v7.1, or earlier models to 9.1.01. Conversion instructions are provided in the next section that explain the process in detail. To update a model, you run a MATLAB command `x1UpdateModel` that invokes a conversion script.

Please be advised that the conversion script does not automatically save an old version of your model as it updates the design nor save a new version of your model after conversion. You can either make a back up copy of your model before running the conversion script, or you can save the updated model with a new name.

Some models may require some manual modification after running the conversion script. The script will point out any necessary manual changes.

Upgrading a Xilinx System Generator Model

Upgrading v2.x and Prior Models

If you are upgrading from versions of System Generator earlier than v3.1, you must obtain System Generator v7.x and update your models to v7.x before you can update them to v9.1.01.

Upgrading v3.x, v6.x and v7.x Models

This section describes the process of upgrading a Xilinx System Generator v3.x, v6.x or v7.x model to work with v9.1.01.

Note: Any reference to v3.x or v6.x in this section can be used interchangeably with v7.x.

The basic steps for upgrading a v7.x model to v9.1.01 is as follows: 1) Save a backup copy of your v7.1 model and user-defined libraries that your model uses 2) Run `xlUpdateModel` on any libraries first and then on your model 3) Read the report produced by `xlUpdateModel` and follow the instructions 4) Check that your model runs under v9.1.01.

These steps are described in greater detail below.

1. Save a backup copy of your v7.1 model and user-defined libraries that your model uses.
2. Run the `xlUpdateModel` Function

From the MATLAB console, `cd` into the directory containing your model. If the name of your model is `designName.mdl`, type `xlUpdateModel('designName')`.

The `xlUpdateModel` function performs the following tasks:

- ◆ Updates each block in your v7.x design to a corresponding v9.1.01 block with equivalent settings.
- ◆ Writes a report explaining all of the changes that were made. This report enumerates changes you may need to make by hand to complete the update.

In most cases, `xlUpdateModel` produces an equivalent v9.1.01 model. However, there are a few constructs that may require you to edit your model. It is important that you read the report and follow the remaining steps in this section.

3. Read the `xlUpdateModel` report and Follow the Instructions

If the report contains the issues listed below, manual intervention will be required to complete the conversion.

- a. Xilinx System Generator v7.x models containing removed blocks

The following blocks have been removed from System Generator: CIC, Clear Quantization Error, Digital Up Converter, J.83 Modulator, Quantization Error, Sync.

- b. Xilinx System Generator v7.x Models that Contain Deprecated Blocks

The DDSv4.0 block still exist in System Generator, but has been deprecated:

- c. Xilinx System Generator v7.x Models Utilizing Explicit Sample Periods

The explicit sample period fields have been removed from most non-source blocks in System Generator v9.1.01. Source blocks (e.g., Counter block) continue to allow the specification of explicit sample periods. When upgrading models containing

feedback loops, Assert blocks must typically be added by hand after `xlUpdateModel` has been run. This is necessary in order to help System Generator determine appropriate rates and types for the path. The following error message is an indication that an Assert block is required:

“The data rates could not be established for the feedback paths through this block. You may need to add Assert blocks to instruct the system”

In such a case, you should augment each feedback loop with an Assert block, and specify rates and types explicitly on this block.

The update script will annotate the converted model wherever the v7.1 model asserted an explicit period. In the converted model, you will most often not need to insert Assert blocks. To find out where you need them, try to update the diagram (the Update Diagram control is under the Edit menu). If rates do not resolve, you will need to insert one or more Assert blocks.

The update script can be configured to automatically insert Assert blocks immediately following blocks configured with an explicit sample period setting. To use this option, run the following command:

```
xlUpdateModel(designName, 'assert')
```

4. Save and Close the updated model.

If you did not previously make a backup copy of the old model, you can save the updated model under a new name to preserve the old model.

5. Verify that Your model Runs Under System Generator v9.1.01.

If you have followed the instructions in the previous steps, your model should run with System Generator v9.1.01. Open the model with System Generator v9.1.01 and run it.

Examples

Example 1:

```
>> xlUpdateModel('my_model_name');
```

Update the file `my_model_name.mdl` that is located in the current MATLAB working directory.

Example 2:

```
>> xlUpdateModel('my_model_name', 'lib');
```

Update the file `my_model_name.mdl` that is located in the current MATLAB working directory, along with the libraries that are associated with the model.

Example 3:

```
>> xlUpdateModel('my_model_name', 'assert');
```

Update the file `my_model_name.mdl` that is located in the current MATLAB working directory. Add **Assert** blocks where necessary.

Getting Started

Introduction

This Getting Started training consists of six short lessons that introduce you to major features of System Generator for DSP. Each lesson takes less than 10 minutes to read and is followed by one or more hands-on lab exercises. The lab exercise folders are located in the System Generator software tree and contain data files and step-by-step instructions.

If you have System Generator installed on your computer, you can complete each lab exercise at your own pace and on your own time schedule. If you do not have System Generator installed, you can access this free training in a recorded e-learning format through the Xilinx web site at the following location:

<http://www.xilinx.com/support/training/rel/system-generator.htm>

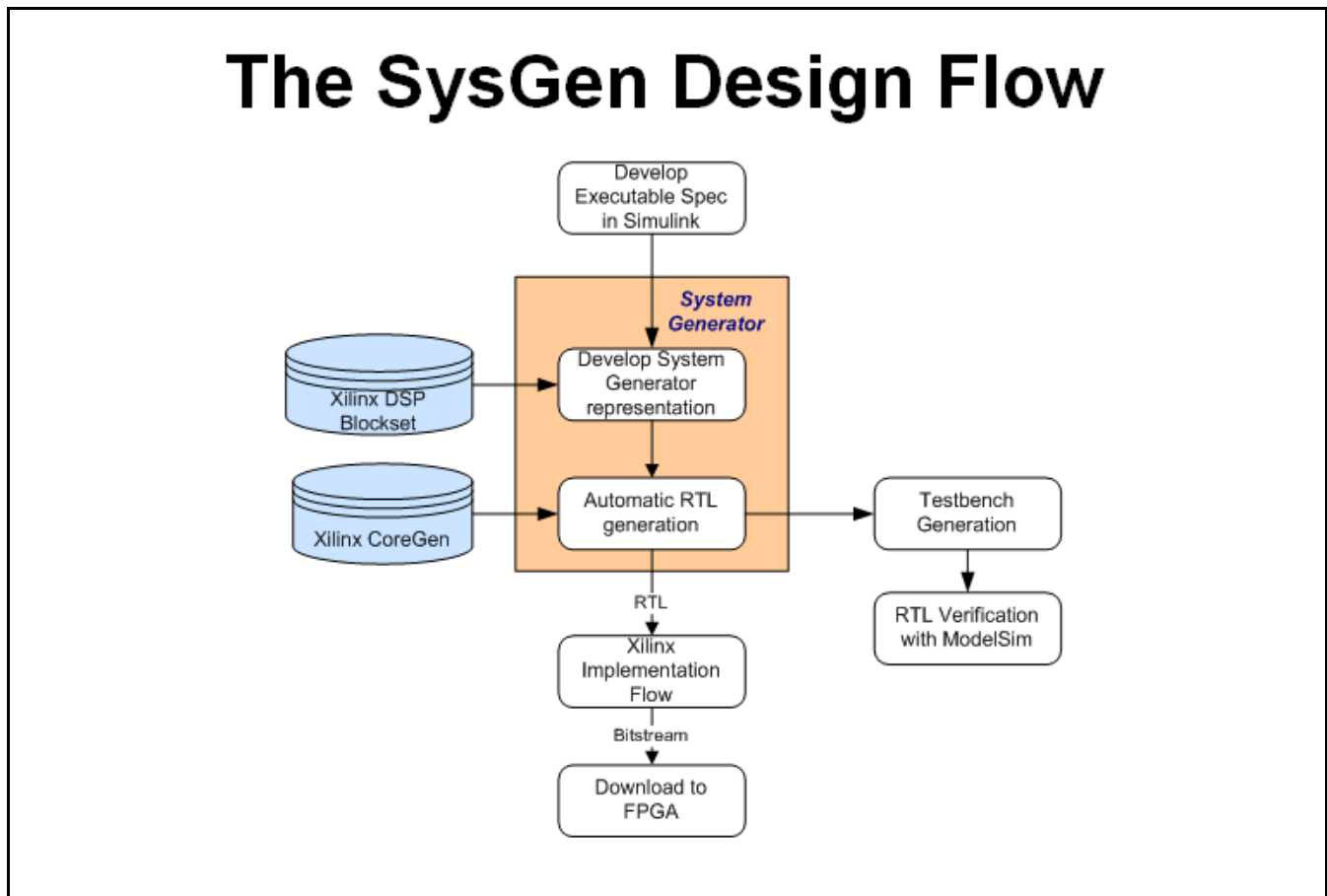
The lessons contained in this Getting Started are as follows:

- **Lesson 1 - Design Creation Basics:** Introduces the basics of creating and implementing a DSP design using System Generator.
- **Lesson 2 - Fixed Point and Bit Operations:** Covers the use of the System Generator routing blocks for extracting and manipulating the individual bits of a fixed-point signal.
- **Lesson 3 - System Control:** Covers the preferred methods for using System Generator to create finite state machines, logical control conditions, and the handling of bursty data typical of FFT and filtering operations.
- **Lesson 4 - Multi-Rate Systems:** Shows the proper way to create multi-rate systems using upsampling and downsampling of data.
- **Lesson 5 - Using Memories:** Covers proper usage of the Xilinx block RAM resources and the DSP blocks available for building DSP designs targeting Xilinx RAMs.
- **Lesson 6 - Designing Filters:** Discusses methods for creating efficient FIR filters in the Xilinx devices, use of the FIR Compiler block for filter implementation, and use of the FDATATool for filter design.

Lesson 1 - Design Creation Basics

The System Generator Design Flow

System Generator works within the Simulink model-based design methodology. Often an executable spec is created using the standard Simulink block sets. This spec can be designed using floating-point numerical precision and without hardware detail. Once the functionality and basic dataflow issues have been defined, System Generator can be used to specify the hardware implementation details for the Xilinx devices. System Generator uses the Xilinx DSP blockset for Simulink and will automatically invoke Xilinx Core Generator to generate highly-optimized netlists for the DSP building blocks. System Generator can execute all the downstream implementation tools to product a bitstream for programming the FPGA. An optional testbench can be created using test vectors extracted from the Simulink environment for use with ModelSim or the Xilinx ISE Simulator.

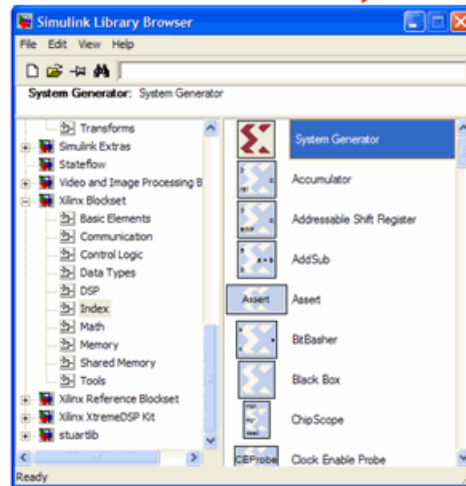


The Xilinx DSP Blockset

The Xilinx DSP blockset is accessed via the Simulink Library browser which can be launched from the standard MATLAB toolbar. The blocks are separated into sub-categories for easier searching. One sub-category, "Index" includes all the block and is often the quickest way to access a block you are already familiar with. Over 90 DSP building blocks are available for constructing your DSP system.

The Xilinx DSP Blockset

- Over 90 DSP building blocks available
- Blocks are accessed through the Simulink Library Browser
 - This can be launched from the MATLAB toolbar

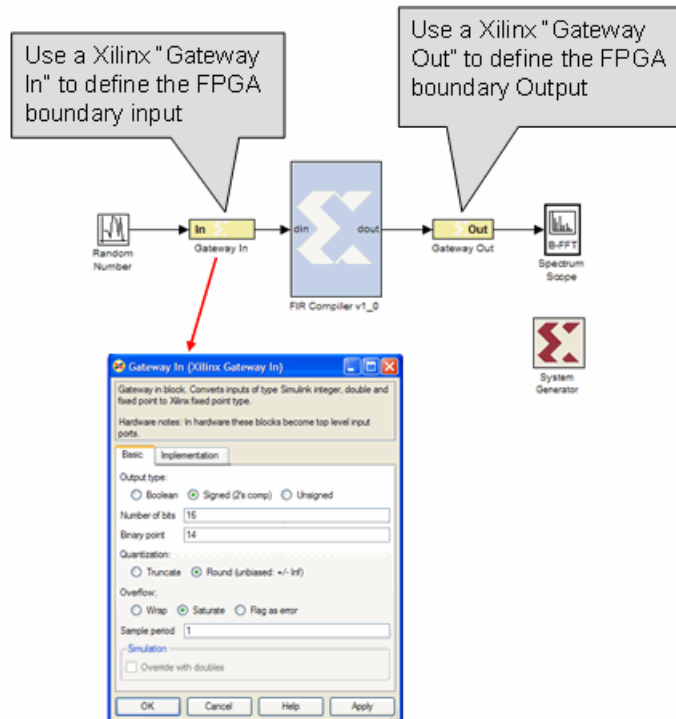


Defining the FPGA Boundary

System Generator works with standard Simulink models. Two blocks called “Gateway In” and “Gateway Out” define the boundary of the FPGA from the Simulink simulation model. The Gateway In block converts the floating point input to a fixed-point number. You double-click on the block to bring up the properties editor which is where the fixed-point number can be fully specified.

Defining the FPGA Boundary

- The FPGA boundary is defined by the Xilinx “Gateway In” and “Gateway Out” blocks
- The “Gateway In” block converts the floating-point input to fixed-point
 - Saturation and rounding modes are defined
- The “Gateway Out” block converts the FPGA outputs back to double precision

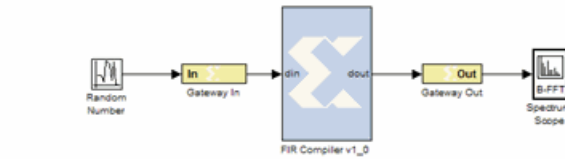


Adding the System Generator Token

Every System Generator diagram requires that at least one System Generator token be placed on the diagram. This block is not connected to anything but serves to drive the FPGA implementation process. The property editor for this block allows specification of the target netlist, device, performance targets and system period. System Generator will issue an error if this block is absent.

Adding The System Generator Token

- Every design must include a System Generator Token
- Sets the global netlisting options required for FPGA implementation
 - Target device
 - VHDL / Verilog RTL
 - Clock performance requirements
 - Downstream toolflow



Each System Generator must include the "System Generator" Token

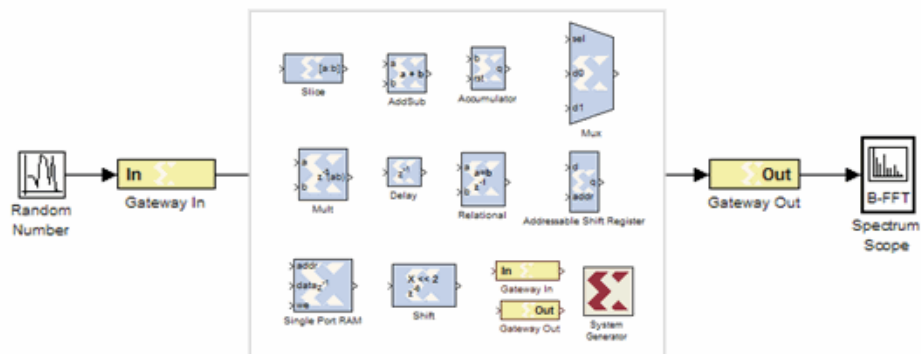
The Simulink System Period must be set correctly for simulation to work

Creating the DSP Design

Once the FPGA boundaries have been established using the Gateway blocks, the DSP design can be constructed using blocks from the Xilinx DSP blockset. Standard Simulink blocks are not supported for use within the Gateway In / Gateway out blocks. You will find a rich set of filters, FFTs, FEC cores, memories, arithmetic, logical and bitwise blocks available for use in constructing DSP designs. Each of these blocks are cycle and bit accurate.

Creating the DSP Design

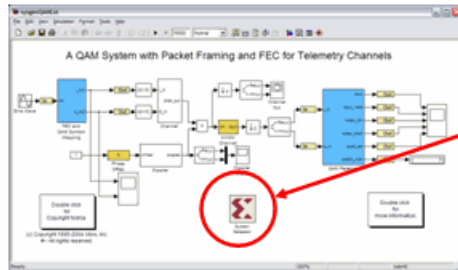
- All the blocks in the Xilinx DSP blockset are available for creating DSP designs targeting FPGAs
 - Over 90 blocks are available
 - Basic building blocks such as arithmetic and logical operators
 - Complex IP such as FIR filters, FFTs and FEC blocks
- These blocks leverage Xilinx IP generators to produce optimal results for Xilinx devices



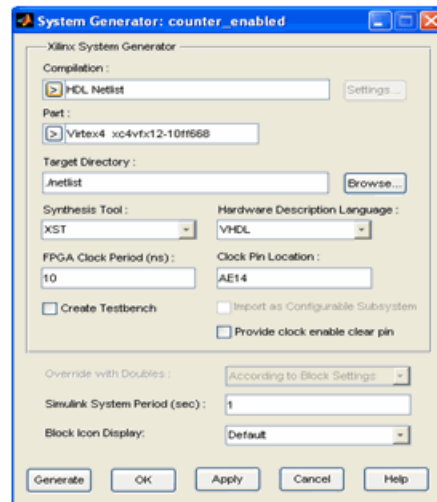
Generating the HDL Code

Once the design is completed, the hardware implementation files can be generated using the **Generate** button available on the System Generator token properties editor. One option is to select **HDL Netlist** which allows the FPGA implementation steps of RTL synthesis and place and route to be performed interactively using tool specific user interfaces. Alternatively, you can select **Bitstream** as the Compilation target and System Generator will automatically perform all implementation steps.

Generating the HDL Code



Once complete, double-click the System Generator token



- Select **HDL Netlist** as the compilation mode
- Select the target part
- Set HDL language
- Set the **FPGA Clock Period**
- Check **Create Testbench**
- **Generate** the HDL

If the **Create Testbench** option is selected, then System Generator will save and write test vector files that are extracted from the Simulink simulation and generate an HDL testbench and script files for ModelSim. This is an optional step that simply verifies that the generated hardware is functionally equivalent to the Simulink simulation. The script files must be used with ModelSim interactively.

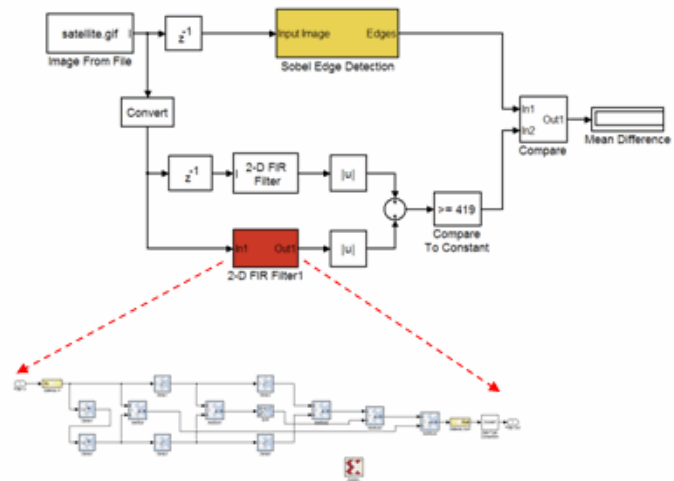
Model-Based Design using System Generator

Model-based design refers the design practice of creating a high-level executable specification using the standard Simulink blocksets or MATLAB first to define the desired functional behavior with minimal hardware detail. This executable spec is then used as a reference model while the hardware representation is specified using the Xilinx DSP blockset.

Model-Based Design using System Generator

System Generator extends the model based design environment of Simulink for FPGA Design

- First develop a high-level executable spec using standard Simulink blocks
- Create an FPGA specific implementation using System Generator
- Use Simulink to compare for functional and fixed-point differences

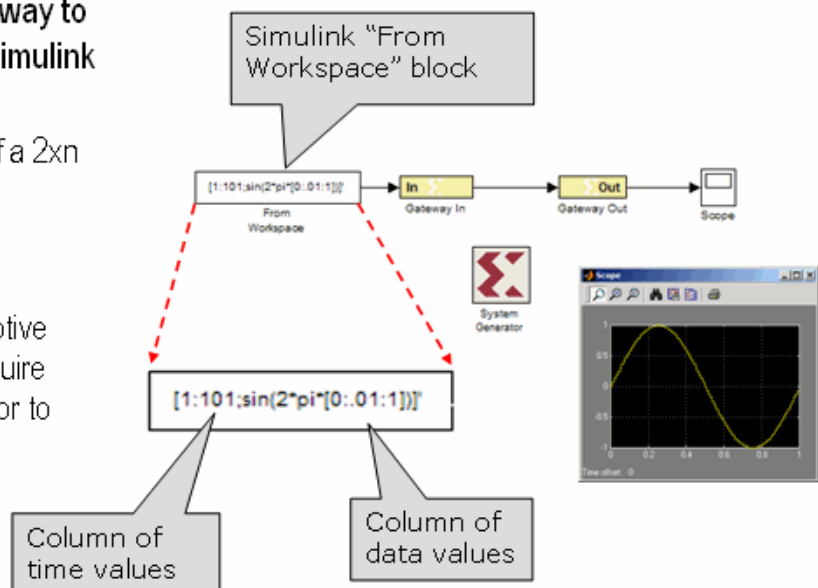


Creating Input Vectors using MATLAB

Simulink is built on top of MATLAB allowing the use of the full MATLAB language for input signal generation and output analysis. You can use the "From Workspace" and "To Workspace" blocks from the Simulink Source and Sink libraries. Input values must be specified as an n rows \times 2 column matrix where the first column is the simulation time and the second column includes the input values. This is a very popular way to generate input vectors for System Generator designs.

Creating Input Vectors using MATLAB

- The Simulink "From Workspace" block provides a convenient way to generate input stimulus for Simulink designs
 - Data must be in the form of a $2 \times n$ matrix
 - Column 1 = time values
 - Column 2 = data values
 - Often this is a more descriptive approach and does not require sourcing a MATLAB file prior to simulation



Lesson 1 Summary

- You partition the FPGA design from the Simulink “system” using Gateway In / Gateway Out blocks.
- You always include a System Generator token on each sheet
- You should only use blocks from the Xilinx DSP blockset between the gateway blocks
- You should consider using the From / To workspace blocks to use MATLAB for input generation and output analysis

Lab Exercise: Using Simulink

In this lab, you will learn the basics of Simulink. You will use a Simulink blockset 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.

The lab instructions are located in the System Generator software tree at the following pathname:

```
...<sysgen_tree>/examples/getting_started_training/lab1/lab1.pdf
```

Lab Exercise: Getting Started with System Generator

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

The lab instructions and lab design are located in the System Generator software tree at the following pathname:

```
...<sysgen_tree>/examples/getting_started_training/lab2/
```

Lab Instructions

Lab Design

Lesson 2 - Fixed Point and Bit Operations

Fixed-Point Numeric Precision

System Generator supports three data types, **Unsigned** for positive only DSP operations, **Signed** which is two's complement used for DSP operations that involve negative numbers and **Boolean** for 1-bit control signals. Each block will typically have quantization parameters. The initial quantization is defined by the Gateway In blocks.

Fixed-Point Numeric Precision

- The Xilinx “Gateway In” block will convert the Simulink “double” datatype to fixed-point numeric precision
 - Fixed-point arithmetic trades off numerical precision for hardware efficiency
- System Generator supports unsigned (ufixed) and two's complement (fixed)
 - Use “fixed” for negative numbers
 - Reduced dynamic range



UNSIGNED

Decimal	Bit Pattern
15	1111
14	1110
13	1101
12	1100
11	1011
10	1010
9	1001
8	1000
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000

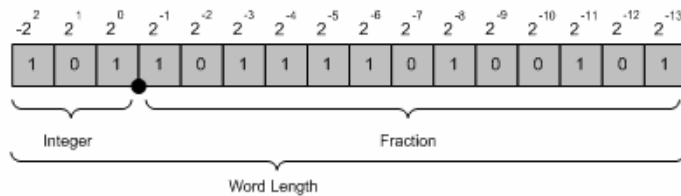
TWO'S COMPLEMENT

Decimal	Bit Pattern
7	0111
6	0110
5	0101
4	0100
3	0011
2	0010
1	0001
0	0000
-1	1111
-2	1110
-3	1101
-4	1100
-5	1011
-6	1010
-7	1001
-8	1000

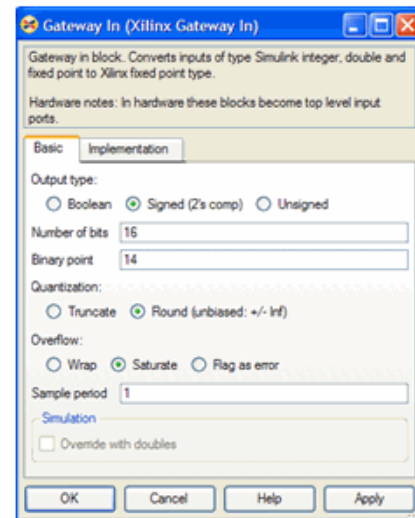
System Generator Fixed-Point Quantization

Xilinx fixed-point data types are defined by specifying the total number of bits then specifying the location of the binary point. The difference, which represents the number of bits to the left of the binary point, are the integer bits for unfixed numbers and the integer bits plus sign bit for signed numbers. Xilinx FPGAs do not require that fixed-point numbers fall in pre-defined 8 bit boundaries as is the case with DSP processors. The logic can grow bit-by-bit to accommodate the required fixed-point precision.

System Generator Fixed-Point Quantization



- System Generator supports the following fixed-point data types
 - Signed (2's Complement)
 - Required for negative numbers
 - Unsigned
 - Provides a greater range with same hardware when numbers are all positive
- To optimize the dynamic range of a number
 - Use a minimal # of integer bits to accommodate the range of possible values
 - Use a minimal # of fraction bits to accommodate acceptable precision



Fractional Bits	Fractional Values Available
1	0, 0.5
2	0, 0.25, 0.5, 0.75
3	0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875

Overflow and Round Modes

System Generator supports the overflow modes **Wrap**, **Saturate** and **Flag as error**. **Wrap** is the default because it has the least cost in hardware. **Saturate** requires System Generator to insert logic to perform that operation and therefore should only be used when necessary for the application

System Generator supports **Truncate** and **Round** of the LSB during the quantization process. Similar to the **Wrap** mode for overflow mode, **Truncate** has minimal hardware cost and is the default. Specifying the **Round** mode requires System Generator to insert extra logic and should be used when only necessary for the application.

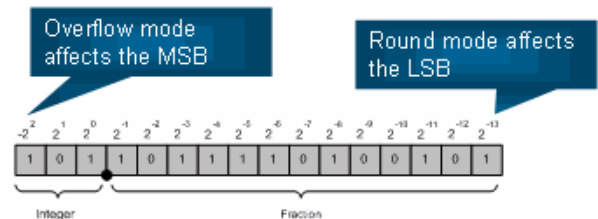
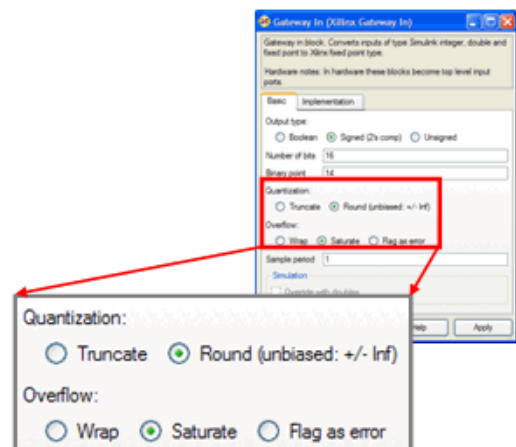
Overflow and Round Modes

Overflow

- Defines how System Generator handles the MSB of a number
 - When a number is too large to be represented by the integer bits of a number
- **Wrap** – the MSB's are dropped
 - Simple to implement in hardware
- **Saturate** – The result is set to the maximum value
 - Requires additional logic

Round Mode

- Defines how System Generator handles the LSB of a number
 - When a floating-point number is converted to fixed-point, "unnecessary" precision is lost
- Users must decide to "cut the precision off" (truncate) or to round to the nearest precision value (round)



Bit-Level Operations

In a real DSP hardware system, not all operations can be expressed mathematically. Often a signal must be accessed by its individual bits. System Generator supports a set of bit-level operations that allow the reinterpret, combining, conversion and extraction of the individual bits of a signal. This can be used to pad, unpad and slice off the bits of a signal with a high degree of control. These blocks do not use any hardware resources

Bit-level Operations

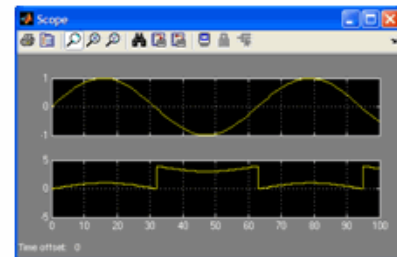
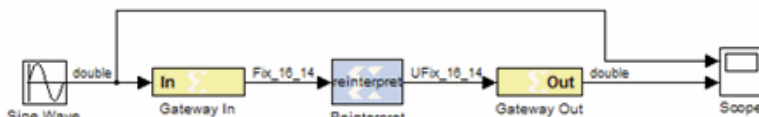
- Implementing a DSP design in hardware will typically require some operations to be performed at the bit level
- System Generator support blocks to perform the following bit-level operations:
 - **Reinterpret** unsigned data as signed or the converse
 - **Combine** two data buses together to form a new bus
 - **Convert** a fixed-point data type to a new fixed-point data type
 - **Extract** certain bits of data, especially when there is bit growth

The Reinterpret Block

The Reinterpret block forces the bits of a signal to a new type without regard for the numerical value or location of the decimal point. This block does not change the number of bits of a signal but simply reinterprets the data type. For example if the number 4 is represented as an unsigned [4 1] it is 1000. If this number is reinterpreted to be unsigned [4 0], the 1000 is now 8.

Reinterpret Block

- Forces the output to a new type without regard for the numerical value represented by the input
- The total number of bits in = total number of bits out
- Allows unsigned data to be reinterpreted as signed data and the converse
- Allows scaling of the data through repositioning of the binary point
 - '1000' quantized to unsigned [4 1] = 4
 - '1000' reinterpreted to unsigned [4 0] = 8

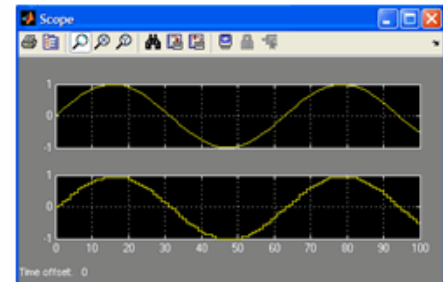
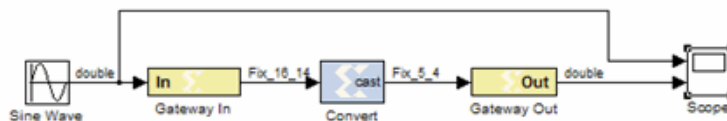


The Convert Block

The Convert block changes the quantization of a number but not the value. This block can alter the number of bits used to represent a number. It can be used to convert a signed type to an unsigned type and visa versa. Often the Convert block is used to truncate the output fractional bits after a multiplication operation.

Convert Block

- converts each input sample into a number of a desired arithmetic type
 - Converts a number to a signed (twos complement), unsigned value, or Boolean
 - The total number of bits and the binary point are user specified
 - Overflow and quantization options apply to the output value

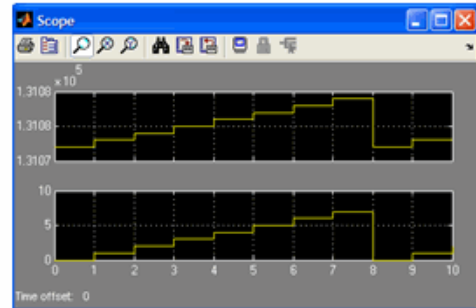
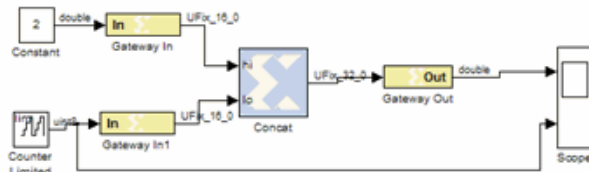


The Concat Block

The Concat block concatenates two inputs into a single output at the bit level. This block has two input ports that are labeled hi and lo. The hi port occupies the MSB's and the lo input occupies the LSB's of the output signal. This block is useful for zero padding the MSBs or LSBs of a signal.

Concat Block

- concatenates two inputs up to 16 bits
- All inputs must be unsigned integers
 - That is, unsigned numbers with binary points at position zero
- The Reinterpret block provides signed-to-unsigned conversion capabilities that can extend the functionality of the concat block

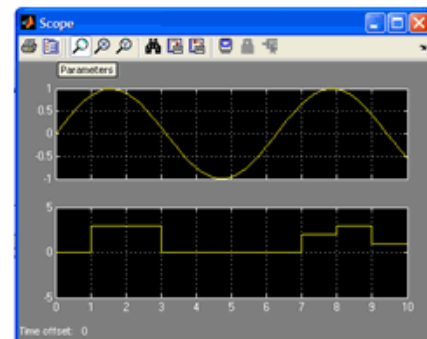
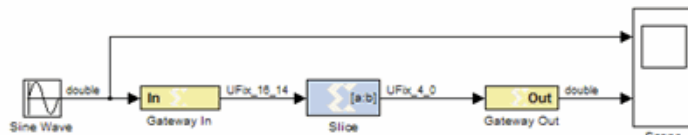


Slice Block

The Slice block is used to access individual bits of a quantized number. This block provides several mechanisms by which the sequence of bits can be specified. If the input type is known at the time of parameterization, the various mechanisms do not offer any gain in functionality. If, however, a Slice block is used in a design where the input data width or binary point position are subject to change, the variety of mechanisms becomes useful. For example, the block can be configured to always extract only the top bit of the input, or only the integer bits, or only the first three fractional bits.

Slice Block

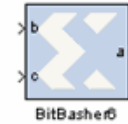
- Slices off a sequence of bits from the input data to create a new data value
- The output data type is unsigned, with its binary point at zero
 - One bit slices can be set to type “boolean”



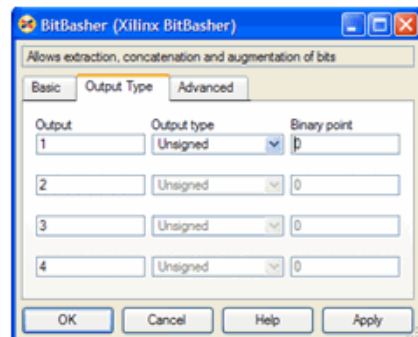
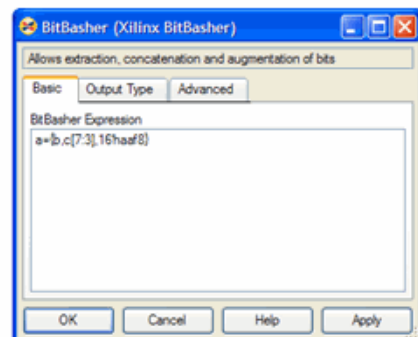
The BitBasher Block

The BitBasher block provides a textual method, based on Verilog syntax, for working with the signals at the bit level. This block supports concatenation and slicing if the input signal to create an output. It also allows for augmentation with constants. The BitBasher block supports up to 4 outputs that are inferred by the expressions

BitBasher Block



- Bit manipulation and augmentation through textual specification
 - Based on Verilog syntax
 - Supports Concatenation, Slicing and Repeat operators
 - Allows augmentation with constants specified as binary, decimal, octal or hex
 - At least one of the inputs must be from input port
 - Supports up to four outputs
 - Number of Output type and Binary point fields available in Output Type tab depends on number of output equations in Basic tab



Lesson 2 Summary

- Quantization and overflow options are available when the output of a block is user defined
- Quantization occurs when the number of fractional bits is insufficient to represent the fractional portion of a value
- Overflow occurs when a value lies outside the representable range
- Bit picking blocks allow combining of multiple buses into a single bus, force a conversion of data type without changing the number of bits, extract bits, and convert the number into different format
- The BitBasher block allows bit manipulation and augmentation through textual specification based in Verilog

Lab Exercise: Signal Routing

In this lab you will design and verify padding and unpadding logic using the System Generator signal routing blocks

The lab instructions are located in the System Generator software tree at the following pathname:

```
...<sysgen_tree>/examples/getting_started_training/lab3/lab3.pdf
```

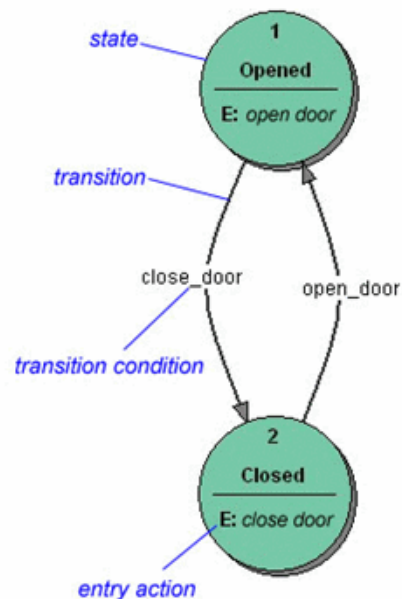
Lesson 3 - System Control

Controlling a DSP System

When you develop a DSP system in hardware, some level of control is usually required. This may include state dependent behavior or simply performing operations such as filter coefficient updating. System-level control may also be needed for controlling bursty data such as non-streaming FFTs.

Controlling a DSP System

- Real hardware will require some level of operational System control
- System Generator supports the following control mechanisms
 - Finite State Machines
 - Bursty data flow control
 - Reset and Clock Enable pins
 - Logical expressions

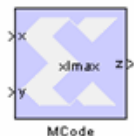


The MCode Block

The MCode block supports the use of MATLAB for implementing state dependent and branch conditional control operations. This block is not suitable for MATLAB that describes an algorithmic operation such as a FIR filter or Matrix inverse. The Xilinx AccelDSP tool can be used in these cases. The MCode block provides a convenient and efficient method for implementing state machines and complex muxing conditions. This is the recommended way to implement a finite state machine in System Generator.

The MCode Block

- System Generator includes an “MCode Block” that supports using MATLAB for modeling low-level hardware control structures
 - MATLAB is translated in VHDL during hardware generation
 - The MCode block does not support algorithmic MATLAB - use AccelDSP
- Recommended for implementing state machines in System Generator
 - A state variable is declared with the MATLAB keyword *persistent* and must be initially assigned with an *xl_state* function call
- Restrictions
 - All block inputs and outputs must be Xilinx fixed-point type
 - The block must have at least one input port and one output port



```
function q = accum(din, rst)
init = 0;
persistent s, s=xl_state(init, x1Signed,4,0);
q=s;
if rst
    s=init;
else
    s=s+din;
end
```

The Xilinx “xl_state” Data Type

When implementing a state machine using the MCode block, a Xilinx-provided MATLAB function called “xl_state” must be used to initialize a persistent variable. This function has two arguments, the first is the initial condition, the second is the quantization of the assigned variable. For example, if your state machine has 6 states, you need a quantization of 4-bits unsigned.

The Xilinx “xl_state” Datatype

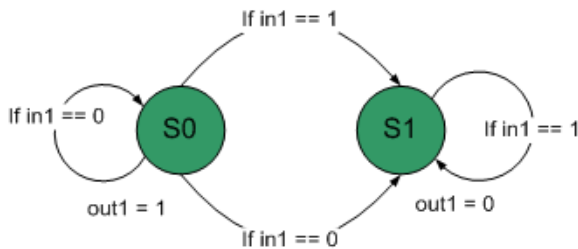
- “xl_state” is a Xilinx provided MATLAB datatype that can be used with the MCode Block to specify FSM state variables
 - Vectors specified with “xl_state” will be efficiently implemented hardware
 - `v = xl_state(init, precision)`
 - Init = initial value of state register after reset
 - Precision = a Xilinx fixed-point datatype defined for the MCode block:
 - `xlUnsigned(<word length>, <binary point>)`
 - `xlSigned(<word length>, <binary point>)`

State Machine Example

The figure below shows a simple 2-state FSM. This can be easily extended to more states. Notice that a variable called "state" is declared to be persistent and is initialized to 2 bits, unsigned using the "xl_state" function. A switch-case statement is then used to decode the inputs, branch to the next state and assign the outputs.

State Machine Example

- The following simple FSM example shown how the MCode block can be used to implement a finite state machine
- This example can be easily extended to include more complex behavior



```
function [out1] = fsm(in1)

persistent state,
state=xl_state(0,{xlUnsigned,2,0});

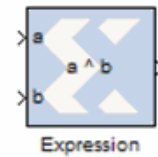
switch double(state)
    case 0
        if in1==1;
            out1=0;
            state=1
        else
            out1=0;
            state=0;
        end
    case 1
        if in1==0
            out1=1;
            state=0;
        else
            out1=0;
            state=1;
        end
    otherwise
        state=0;
        out1=0;
end
```

The Expression Block

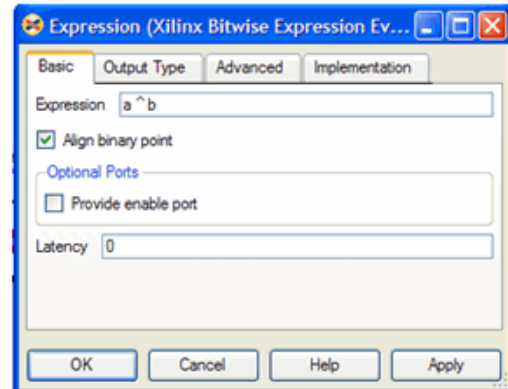
The Expression block performs a bitwise not, and, or & xor on two input signals. The inputs can have a word length greater than 1. In cases where the two inputs have different word lengths, the binary points are matched up and then an element-by-element boolean operation is performed. This block provides a useful way to implement logical control in a DSP system

The Expression Block

- The expression block provides an easy way to implement logical control using expressions
- Number of input ports is inferred from the expression
 - “a & b | c” = 3 inputs



Operator	Symbol
Precedence	()
NOT	~
AND	&
OR	
XOR	^

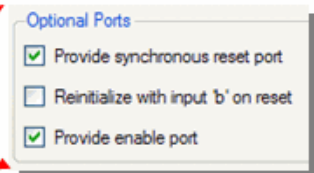
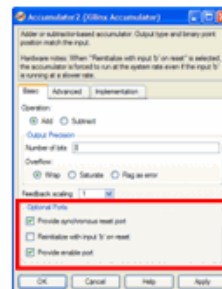
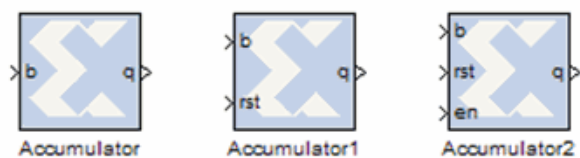


Reset and Enable Ports

Most System Generator blocks that include memory or storage provide options to expose the reset and clock enable ports. If un-selected, these ports are automatically connected to the final hardware's global reset and clock enable or DCM schemes. Exposing these ports on the System Generator block creates a condition where the block is reset or enabled when either the global signals or the local signals assert TRUE. You should use these ports if greater control over these functions is required in the DSP system.

Reset and Enable Ports

- System Generator blocks that include storage generally offer the option to add a reset and clock enable pin
 - The signals driving these ports must be of type "boolean"



Bursty Data

Several of the more complex DSP blocks offered in the Xilinx DSP blockset result in “bursty” data. For example, the non-streaming FFT requires several clock cycles to process the input data prior to generating valid output data. In these cases, these blocks include data flow control ports that must be used in the DSP system. These ports provide basic push mode dataflow control. They consist of a `vin` port which indicates that valid data is available at the inputs and `vout` which indicates that valid data is available at the outputs.

Bursty Data

- Often the dataflow through the system is not continuous but rather comes in “bursts”
 - Non-streaming FFT
 - Resource shared FIR Filter
- In these cases the user will need to implement dataflow control into the System Generator diagram
- System Generator blocks that require extra data processing time will include two flow control ports called `vin` & `vout`



Blocks that have valid bit modeling:

- FIR (optional)
- FFT
- Reed-Solomon Encoder/Decoder
- Viterbi Decoder
- Convolutional Encoder
- Interleaver/DeInterleaver
- CIC

Lesson 3 Summary

- Use the MCode block for state machines and branch conditional logic
- Use the Expression block to implement logical control at the bit level
- Storage elements have the ability to include optional reset and clock enable pins that can be connected in System Generator
- Blocks that operate on bursty data include data flow control pins called `vin` and `vout`

Lab Exercise: System Control

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

The lab data and instructions are located in the System Generator software tree at the following pathname:

```
...<sysgen_tree>/examples/getting_started_training/lab4/
```

Lab Instructions

Lab Data

Lesson 4 - Multi-Rate Systems

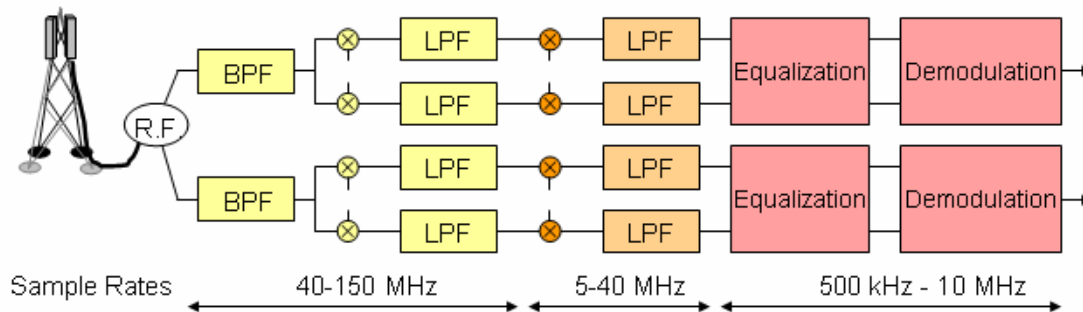
Creating Multi-Rate Systems

The following illustration shows a typical base-station receiver. The tower has multiple antennas to provide sectored coverage of the area. The diagram shows that this results in two receiver channels. In each of these channels, there is some form of complex mixing, resulting in real and imaginary channels.

Often DSP systems such as this will down sample the input signals prior to the digital filtering steps performed during equalization and demodulation. Doing so can simplify the filter design and hardware significantly. These systems are referred to as “multi-rate” systems

Creating Multi-Rate Systems

- Down-sampling and up-sampling data through a DSP system is a common approach to improving hardware efficiency
 - A common example, shown below, is a wireless base station
- System Generator supports the design of multi-rate systems through rate changing blocks



Up and Down Sampling Blocks

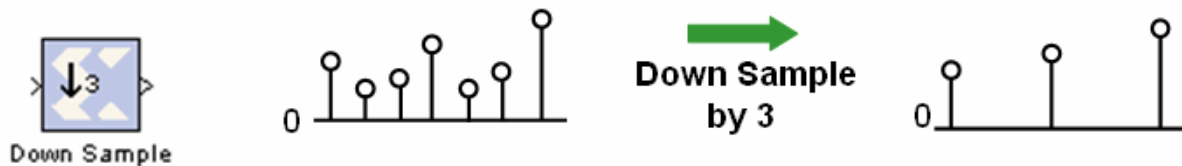
System Generator includes Up Sample and Down Sample blocks that change the system sample rate. The Up Sample block adds additional samples to the signal to achieve the desired rate change. The value of these new samples is either zero or the value of the last actual sample depending on the block options. The Down Sample block simply discards samples until it achieves the desired rate change. For example, downsample by 3 means to discard 2 out of every 3 samples.

Up and Down Sampling Blocks

- Use the “Up Sample” and “Down Sample” blocks to change the rate of a signal in System Generator
 - The **up sample** can either replicate the same number $M-1$ times or insert $M-1$ zeros to achieve the higher sampling rate



- The **down sample** “throws away” $M-1$ samples to achieve the lower sampling rate




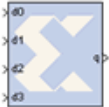


Rate Changing Functional Blocks

In addition to the straightforward “Up Sample” and “Down Sample” blocks, System Generator also provides rate changing functional blocks; that is blocks that also perform a specific function. The Parallel to Serial block will up sample, the Serial to Parallel block will down sample, the FIR Compiler, if using a resource-shared multiplier will down sample and the TDM block will up sample.

Rate Changing Functional Blocks

- The following functional blocks will also change the rate of a DSP system

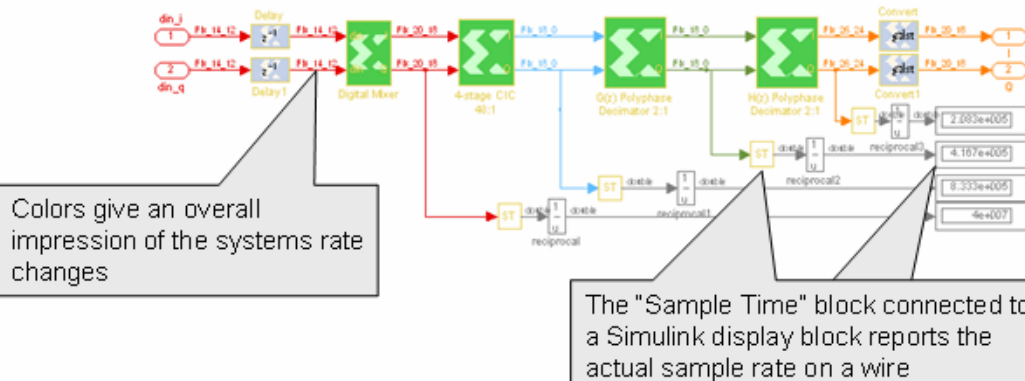
Parallel to Serial: The output rate will be M-times faster, where M is the width of the input parallel data	 Parallel to Serial
Serial to Parallel: The output rate will be M-times slower, where M is the width of the output parallel data	 Serial to Parallel
FIR and FIR Compiler: Can be used as a polyphase interpolation or decimation FIR	 FIR Compiler v1_0
The Time Division Multiplexer block multiplexes values presented at input ports into a single faster rate output. The up sample rate is determined by the number of input	 Time Division Multiplexer

Viewing Rate Changes in Simulink

Simulink supports viewing different sample times as different colors which is fully supported for System Generator blocks. To enable the **Sample Time Colors** feature, select the pulldown menu **Format > Sample Time Colors**. The Simulink tool does not automatically recolor the model with each change you make to it, so you must select **Edit > Update Diagram** to explicitly update the model coloration. To return to your original coloring, disable the sample time coloration by, again, choosing **Sample Time Colors**.

Viewing Rate Changes in Simulink

- Sample rates can be displayed in different colors using Simulink
 - Use the pull down menu command (**Format** → **Sample Time Colors**)
- The actual sample rate of a particular wire can be displayed using the “Sample Time” (ST) block in the Xilinx Blockset
 - Use the Simulink display block to view the output of the sample block
 - Does not add hardware to the design



Debugging Tools

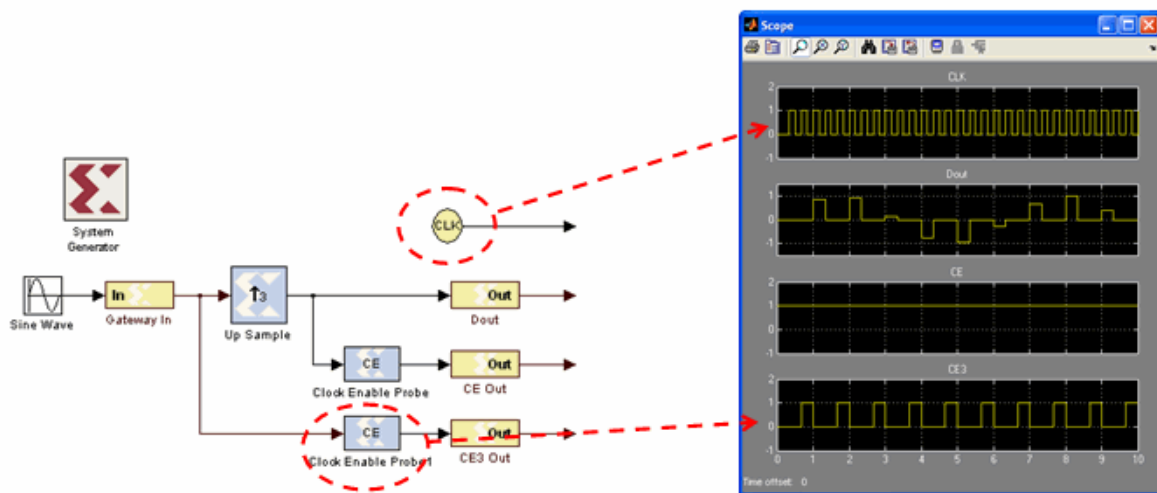
System Generator provides 3 debugging utilities to assist in debugging complex multi-rate systems.

The Sample Time (ST) probe can be connected to any System Generator signal then to a Simulink “display” block from the “Sinks” library. The sample time for the connected net will appear in the display.

The `clk` probe is not connected to any inputs but only to a scope output. It displays the master clock. This can be used with the Clock Enable Probe to display the behavior of the clock enable signal at various points in the down sampling

Debugging Tools

- The “`clk`” probe and the “clock enable probe” can be used to view the behavior of the multi-rate system
 - These blocks add no logic
 - Their outputs can be connected directly to a Simulink sink block



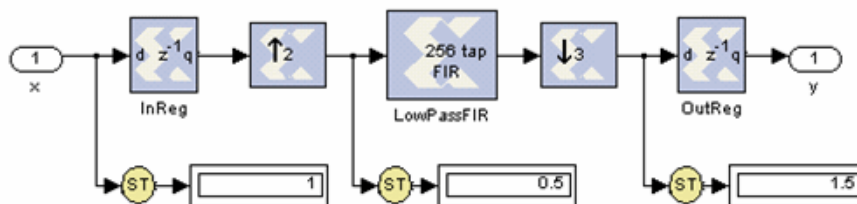
Sample Period “Rules”

The illustration below is an example of a multi-rate system that demonstrates how the Simulink System Period can be calculated and entered into the System Generator token GUI.

If you get it wrong, there is a sampling period analyzer that automatically determines the appropriate sample period and prompts you to update the GUI.

Sample Period “Rules”

- The system period is the global sample period from which all other sample periods can be derived
 - The System Period is set in the System Generator token
- Every sample period in a design must be a multiple of the system period



Block Output	X	Up Sample	Down Sample
Sample Period	1	.5	1.5
Sample Period (GCD)	2/2	1/2	3/2

Lab Exercise: Multi-Rate Systems

In this lab you will be exploring the effects of the rate changing blocks available in System Generator. These blocks include Upsample, Downsample, Serial to Parallel and Parallel to Serial.

The lab instructions and lab design are located in the System Generator software tree at the following pathname:

```
...<sysgen_tree>/examples/getting_started_training/lab5/
```

Lab Instructions

Lab Design

Lesson 5 - Using Memories

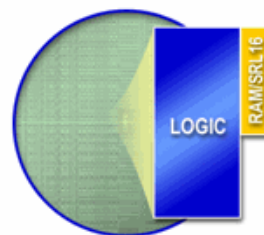
Block vs. Distributed RAM

Xilinx FPGAs offer two distinct memory options, Block RAM and Distributed RAM. Block RAM uses dedicated, on-chip, hardware resources and represents the most area-efficient RAM implementation. Block RAMs offer high performance but due to their fixed location on the chip, may incur slightly larger routing delays. Distributed RAM uses the lookup tables in the FPGA slices to implement memory and in doing so will subtract from the slices available for logical operations. Because Distributed RAM can be located anywhere throughout the chip, routing delays can be minimized and slightly higher performance can be achieved. Distributed RAM is an excellent option for small FIFOs.

Block vs. Distributed RAM

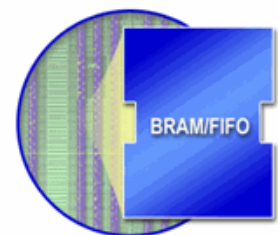
- Xilinx devices offer two implementation options for RAMs, FIFOs and ROMs
 - **Block RAM** – uses dedicated on-chip RAM resources
 - More area efficient
 - **Distributed RAM** – uses the FPGA lookup tables
 - Higher performance
 - Subtracts from available area for logic
- System Generator RAM, FIFO and ROM blocks support either implementation

Distributed RAM/SRL16



- Very efficient, localized memory
- Minimal impact on logic routing
- Great for small FIFOs

On-chip BRAM/FIFO



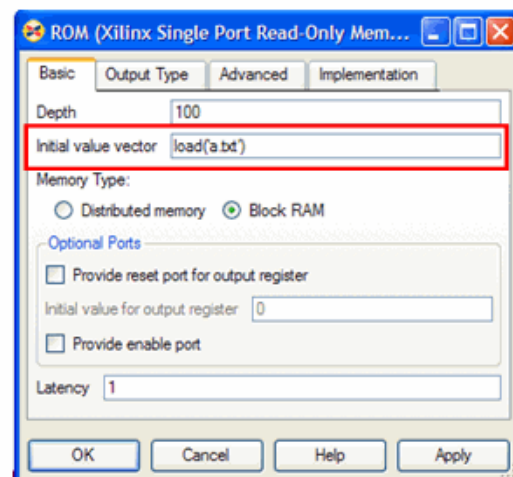
- Efficient, on-chip blocks
- Flexible + optional FIFO logic
- Ideal for mid-sized FIFOs/buffers

Initializing RAMs and ROMs

The RAM and ROM blocks can be initialized to a 1xn vector that matches the depth of the RAM. MATLAB is used to set the initial value vector. Any MATLAB statement can be used that results in a 1xn vector including the file reading commands such as `imread`, `auread`, `wavread`, and `load`.

Initializing RAMs and ROMs

- MATLAB statements are used to initialize the RAMs and ROMs.
 - Statement must create a 1xn vector
- Loading a text file
 - `load('filename.txt')`
- Using a MATLAB statement
 - `sin(pi*(0:15)/16)`
- Reading other file formats
 - `imread`
 - `auread`
 - `aviread`
 - `wavread`
 - `fread`

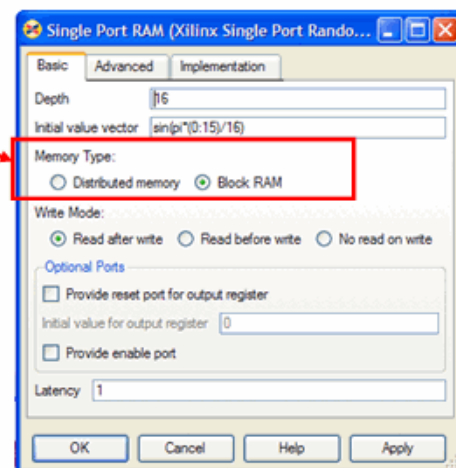
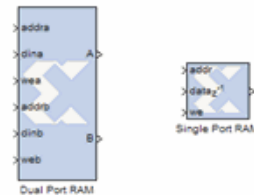


System Generator RAM Blocks

System Generator provides both a single- and dual-port RAM block. Depths up to 64K are supported. Both Distributed RAM and Block RAM implementation options are available. System Generator calls the Xilinx memory compiler to create an efficient memory structure in hardware for the given parameters, bit widths and depths. You don't need to be concerned with the hardware details of the specific Virtex block or Distributed RAM structure. Both the single- and dual-port RAM blocks support initialization. The signal connected to the address port of a RAM must be unsigned with no fractional bits.

System Generator RAM Blocks

- System Generator offers both single and dual port RAM blocks
 - Options include selection of “Block” vs. “Distributed” implementation options
- These blocks call the Xilinx IP memory generator to create efficient RAM implementations for any depth and bit widths

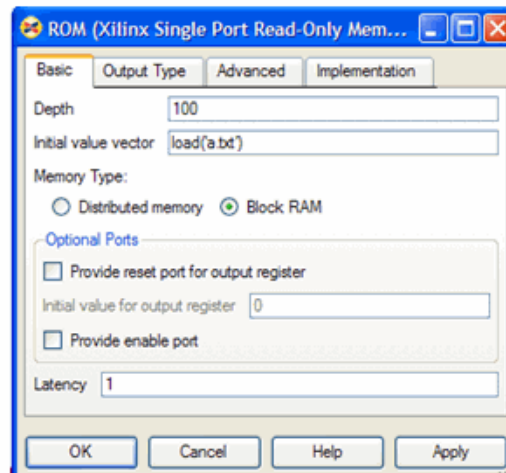


System Generator ROM Blocks

The ROM block supports an implementation in either Block- or Distributed RAM and is initialized through a MATLAB command. The signal connected to the address port must be unsigned with no fractional bits

System Generator ROM Blocks

- Offers both Block RAM vs. Distributed RAM implementation Options
- Address port must be unsigned with no fractional bits
- Depth and data widths are user configurable

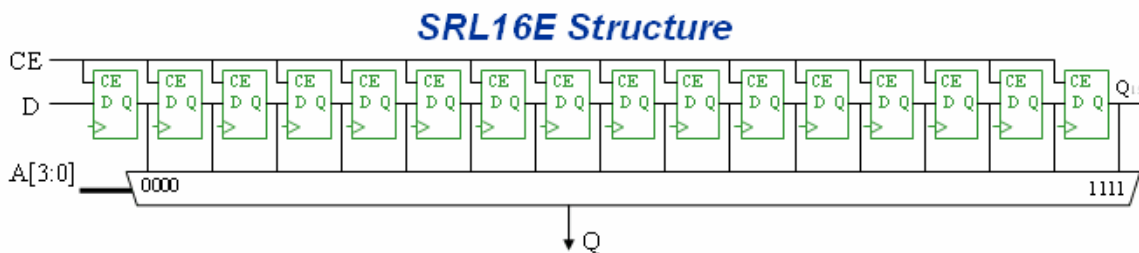


The Delay Block

The Delay block is used to synchronize dataflow through the FPGA. This block maps to a highly-efficient shift register structure built from a slice lookup table called an SRL16 that is 85% smaller than using registers.

The Delay Block

- Use the Delay block to synchronize the dataflow of signals through the design
- The implementation will be constructed from “SRL16E” primitives
 - Highly efficient use of the Virtex distributed RAM for implementing delay elements and shift registers



The FIFO Block

The FIFO block supports both Block RAM and Distributed RAM implementations. Depths up to 64K are supported. Three output flags are supported, `empty`, `full` and `%full`. The `%full` flag is set depending on a bit width specification. One bit will be zero until the FIFO is 50% full, then it will set to 1. Two bits will be zero until 20% full, then .25, .5 and .75.

The FIFO Block

- Can be implemented in either Block or Distributed RAM
- Supports FIFO depths up to 64K
- Supports a “full” and “percentage full” output signals
 - `%full` Specified as number of bits
 - Unsigned, fractional
 - 1 bit shows <50% or >50% full
 - 2 bits show 25% full increments
- Includes optional reset and clock enable ports



Lab Exercise: Using Memories

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

The lab instructions and lab design are located in the System Generator software tree at the following pathname:

```
...<sysgen_tree>/examples/getting_started_training/lab6/
```

Lab Instructions

Lab Design

Lesson 6 - Designing Filters

Introduction

Digital filters are a common DSP operation and especially well suited to implementation in FPGAs. High-performance applications benefit greatly from parallel filters that can return a results on every clock cycle. The Virtex 5 device includes up to 550 parallel multipliers. The FIR Compiler is designed to use these multipliers in the most efficient manner for creating commonly used FIR filters. An alternative implementation is available called “distributed arithmetic” that creates FIR filters without using multipliers by employing a shift-add technique. This can be used for smaller devices when the available multipliers have been allocated to other functions.

Introduction

- Digital filters are the most common functions found in DSP systems
- The following blocks are supported by System Generator for digital filtering
 - FIR Compiler block (DSP Blockset)
 - DAFIR block (DSP Blockset)
 - CIC block (Reference Designs Blockset)
- The digital filtering technique will depend on several factors
 - Sample rate
 - Sample width
 - Profile of the coefficients
 - Clock rate
 - Technological resources

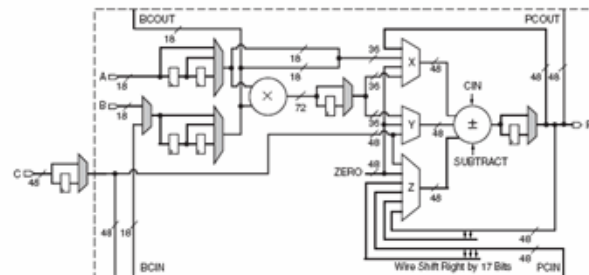
The Virtex DSP48 Math Slice

The Virtex™ family introduces a high-performance arithmetic unit along with a multiplier: the low-power DSP48 slice. The following figure is a detailed diagram of the DSP48 structure. The DSP48 slice consists of four main sections: (1) I/O registers, (2) signed multiplier, (3) three-input adder/subtractor, and (4) OPMODE multiplexers.

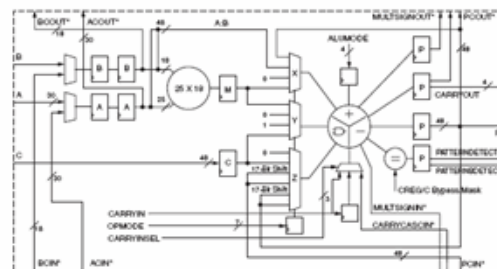
The Virtex DSP48 Math Slice

- The Virtex4 & 5 devices include up to 550 DSP48 slices
 - Performs 48 unique math operations common to DSP operations
 - Configuration set through an “opcode” input
- Efficient use of DSP48 slice is required to get high performance and efficient filters

Virtex4 DSP48 Math Slice



Virtex5 DSP48 Math Slice



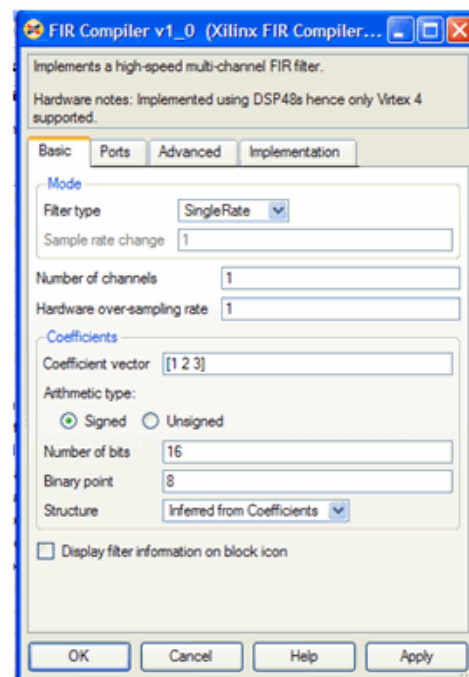
FIR Compiler Block

The Xilinx Fir Compiler v1_0 block implements a high speed MAC based FIR filter. It accepts a stream of input data and computes filtered output with a fixed delay, based on the filter configuration. The FIR Compiler supports generation of resource shared or parallel FIR structures and polyphase decimation and interpolation structures. Also supported is oversampling. Coefficients are specified using MATLAB commands.

FIR Compiler Block



- The Xilinx Fir Compiler block implements a high speed MAC based FIR filters using DSP48 primitives
- Supports polyphase decimation, polyphase interpolation and over sampling implementations

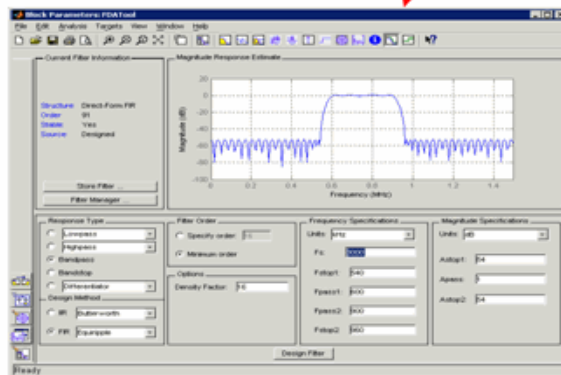
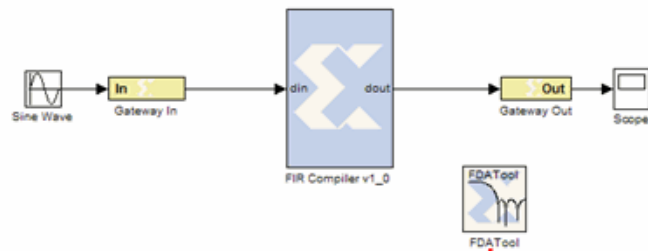


Creating Coefficients with FDATool

The Mathworks FDATool is a graphical filter design program that can be used to generate coefficients for the FIR Compiler block.

Creating Coefficients with FDATool

- Enables the use of The Mathworks FDATool for creating filter coefficients graphically
- Coefficients can be “exported” to either the MATLAB workspace or a text file



Using FDA Tool Coefficients

Once a suitable filter response has been designed, you simply export the coefficients to the workspace using the **File > Export** command. The workspace variable can then be referenced in the FIR Compiler properties editor

Using FDA Tool Coefficients

The image illustrates the process of using FDA Tool coefficients in the Xilinx Filter Compiler. It features two main windows: **Block Parameters: FDA Tool** and **FIR Compiler v1_0 (Xilinx FIR Compiler...)**.

Block Parameters: FDA Tool shows the **Export** dialog box. The **Export To** dropdown is set to **Workspace**, and the **Export As** dropdown is set to **Coefficients**. Under **Variable Names**, the **Numerator** is set to **Num**, which is circled in red. A yellow callout bubble points to this dialog with the text: "Export the coefficients from FDA Tool to the MATLAB workspace".

FIR Compiler v1_0 (Xilinx FIR Compiler...) shows the **Basic** tab of the properties editor. The **Coefficients** section has the **Coefficient vector** set to **Num**, also circled in red. A yellow callout bubble points to this field with the text: "Use the same variable name in the Xilinx Filter Compiler properties editor". A red dashed arrow connects the **Num** variable name in the FDA Tool export dialog to the **Num** variable name in the FIR Compiler properties editor.

Lab Exercise: Designing Filters

In this lab you will be using the Filter Compiler block to generate optimized filters for the Virtex4 architecture.

The lab instructions and lab design are located in the System Generator software tree at the following pathname:

```
...<sysgen_tree>/examples/getting_started_training/lab7/
```

Lab Instructions

Lab Design

Additional Examples and Tutorials

Numerous examples are used to illustrate System Generator features and functions in the System Generator documentaton. These examples are found in the directory at pathname <sysgen_tree>/examples and are listed in the table below. In addition to these examples, System Generator also includes demonstration models that can be run from the demo page. Enter the following command at the MATLAB prompt:

```
demo blocksets xilinx
```

Note: If you are using the MATLAB help browser you can open and run the examples directly from this page. To run an example, click on the link. MATLAB will change directories to the example directory and open the example model.

Black Box Examples

Topic	Description
Importing a VHDL Module	A tutorial showing how to use the black box to import VHDL into a System Generator design and how to use ModelSim to co-simulate the VHDL module.
Simulating Several Black Boxes Simultaneously	Shows how black boxes can co-simulate simultaneously, using only one ModelSim license.
Dynamic Black Boxes	A tutorial showing how to parameterize the black box.
Importing a Verilog Module	A tutorial showing how to use the black box to import Verilog into a System Generator design and how to use ModelSim to co-simulate the Verilog module.
Importing a Xilinx Core Generator Module	A tutorial showing how to import a COREGEN module as a black box.

ChipScope Examples

Topic	Description
Using ChipScope Pro Analyzer for Real-Time Hardware Debugging	This tutorial demonstrates how to connect and use the Xilinx Debug Tool called ChipScope Pro within Xilinx System Generator. The integration of ChipScope Pro in the System Generator flow allows real-time debugging at system speed.

DSP Examples

Topic	Description
DSP48 Block	Simple example demonstrating the use of the DSP48 block with the Constant block used to provide the DSP48 instruction.
DSP48 Macro Block	Simple example demonstrating how to use a DSP48 Macro block to implement a Complex Multiplier.
DSP48 Block (35-Bit Multiplier using DSP48 and Constant block)	This design demonstrates the use of the DSP48 and Constant block in implementing 35 by 35-bit multipliers at different sample rates. Three multipliers implementations are shown at 1, 2, and 4 clocks per sample.
DSP48 Macro Block (FIR filter using the DSP48 Macro block as a multiply accumulate function)	This design demonstrates the use of the DSP48 Macro block in implementing a 35 by 35 Multiplier.
DSP48 Block FIR filter examples using DSP48 block	This design demonstrates the use of the DSP48 and Constant block in FIR filter implementation. The design includes sets of parallel, semi-parallel and sequential FIR filter using Type 1 and Type 2 architectures. Each filter implements a 16-tap dsp48-based FIR filters.
DSP48 Design Techniques (DSP48-based dynamic shifter)	This design demonstrates the use of the DSP48 block in implementing a 35-bit signed right shift using 2 DSP48s.
DSP48 Design Techniques (Synthesizable FIR filter for Virtex4)	This design demonstrates how to use System Generator to implement a synthesizable FIR filter which maps efficiently to the Virtex4 architecture.
DSP48 Macro Block (FIR filter using the DSP48 Macro block as a multiply accumulate function)	This design demonstrates the use of the DSP48 Macro block when implementing a sequential FIR filter.
MAC FIR filter	This design example implements a 43 tap FIR Filter with a MAC engine and a Dual Port Ram used for data and coefficient storage.
Complex FIR filter	This example demonstrates a complex FIR filter built out of blocks from the System Generator and Simulink library.

M-Code Examples

Topic	Description
Simple Selector	This example shows how to implement a function that returns the maximum value of its inputs.
Simple Arithmetic Operations	This example shows how to implement simple arithmetic operations.
Complex Multiplier with Latency	This example shows how to build a complex multiplier with latency.
Shift Operations	This example shows how to implement shift operations.
Passing Parameters into the MCode Block	This example shows how to pass parameters into a MCode block.
Optional Input Ports	This example shows how to implement optional input ports on an MCode block.
Finite State Machines	This example shows how to implement a finite state machine.
Parameterizable Accumulator	This example shows how to build a parameterizable accumulator.
FIR Blocks and Verification	This example shows how to model FIR blocks and how to do system verification.
RPN Calculator	This example shows how to model a RPN calculator – a stack machine.
Example of disp function	This example shows how to use the disp function.

Processor Examples

Topic	Description
Designing and Exporting MicroBlaze Processor Peripherals	Demonstrates how to export a design from System Generator into Xilinx Platform Studio (EDK) by showing how to design a peripheral (pcore) for a MicroBlaze processor. An RGB to gray-scale color space converter is created and generated into a pcore using the Export to EDK compilation target.

Topic	Description
Tutorial Example - Designing and Simulating MicroBlaze Processor Systems	Demonstrates how to import a MicroBlaze processor created using Xilinx Platform Studio into System Generator. A DSP48 block is used as a co-processor to the MicroBlaze.
Designing PicoBlaze Microcontroller Applications	Demonstrates how to implement a PicoBlaze program in System Generator. The example programs the PicoBlaze to alter the output frequency of a Direct Digital Synthesizer (DDS) during an interrupt.

Shared Memory Examples

Topic	Description
Simulation across various models	Illustrates shared memories communicating across Simulink models.
Host PC Shared Memory access	Developer studio project to communicate with a shared memory.
High Speed Video Processing using Hardware Co-simulation	Discussion of a high-speed co-simulation buffering interface followed by an example in which the interface is used to support real-time processing of a video stream using a 5x5 filter kernel.
High speed I/O Buffering	Illustrates high speed Shared Memory I/O Buffering Interface for Hardware Co-simulation.
SharedMemory (Mex-function interface)	Illustrates the use of a mex-function as an interface to a shared memory.
Generating Multiple Cycle-True Islands for Distinct Clocks	An example using two asynchronous clocks.
Shared Memory, To FIFO, To Register, To Register, From Register	Demonstrates use of shared memories, FIFOs and registers to pass information.
Frame-Based Acceleration using Hardware Co-Simulation	Explains how to use frame or vector-based transfers to further accelerate simulations using FPGA hardware co-simulation.

Timing Analysis Examples

Topic	Description
Timing Analysis Tutorial	Explains how to use the System Generator Timing Analysis tool to meet timing requirements of System Generator designs. Also touches on techniques that may be used when a design does not meet timing.

Miscellaneous Examples

Topic	Description
Importing a System Generator Design into a Bigger System	Discusses how to take the VHDL netlist from a System Generator design and synthesize it in order to embed it into a larger design. Also shows how VHDL created by System Generator can be incorporated into simulation model of the overall system.
Configurable Subsystems and System Generator	Illustrates the use of Configurable Subsystems for Simulation and Generation.
Integrator	This example uses an integrator to illustrate error analysis capability.
Block RAM-Based State Machines	Demonstrates use of Mealy State Machine block from the reference library.

System Generator Demos

System Generator for DSP provides the capability to model and implement high-performance DSP systems in field-programmable gate arrays (FPGAs) using Simulink. The Xilinx Blockset contains bit and cycle-true models of arithmetic and logic functions, memories, and DSP functions for digital filtering, spectral analysis, and digital communications. System Generator converts a Simulink model of Xilinx blocks into an efficient hardware implementation that combines synthesizable VHDL and intellectual property blocks that have been hand-crafted to run efficiently in FPGAs.

Included with the tool are numerous demonstration designs that highlight key features and tool capabilities, as well as general good design practices using real-world design applications. These designs may be accessed from the System Generator demo page. Enter the following command at the MATLAB prompt:

```
demo blocksets xilinx
```

Index

A

Asynchronous Software Drivers
for FSLs 25

C

Compatibility
MATLAB 24, 29, 34, 36
ModelSim 24, 29, 34, 36
Synplify Pro 24, 29, 34, 36
Compiling
Xilinx HDL Libraries 21
Configuring
the Sysgen cache 21
Custom Bus Interfaces
for exported pcore 26

D

Downloading
System Generator 19

E

Export pcore
enable Custom Bus Interfaces 26

F

Fast Simplex Link
asynchronous software drivers for
25

H

Hardware Co-Sim
installation 20

I

Installation
Hardware Co-Sim 20
software prerequisites 20
ISE Design Suite Installer 20

M

MATLAB 24, 29, 34, 36
Memory Stitching 31
ModelSim 24, 29, 34, 36

P

Pcore
export as under development 26

S

Shared Memory Stitching 31
Software Drivers
asynchronous for FSLs 25
System Generator
Cache 21
changing versions 21
displaying versions 21
downloading the software 19
ISE Design Suite Installer 20
System Generator Utilities
xlUpdateModel 37

U

Underdevelopment
export pcore as 26

X

Xilinx HDL Libraries
compiling 21
xlUpdateModel 37