Basic Xilinx Design Capture

Objectives

After completing this module, you will be able to:

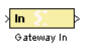
- List various blocksets available in System Generator
- Describe how signals are fed to and results are read from a System Generator based design
- List various data types supported by System Generator
- Identify steps involved in performing hardware-in-the-loop verification
- State how hardware-in-the-loop verification is beneficial for complex system designs

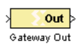
Outline

- **Gateway In/Gateway Out**
- Data Types
- Constructing Design Using Xilinx Design Capture
- System Generator Block
- HDL Co-Simulation
- Hardware Verification
- Summary

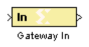
Interacting with SysGen Design

- The Simulink environment uses a “double” to represent numbers in a simulation. A double is a 64-bit twos complement floating point number
  - Because the binary point can move, a double can represent any number between +/- 9.223 x 10^{18} with a resolution of 1.08 x 10^{-19}…a wide desirable range, but not efficient or realistic for FPGAs
- The Xilinx blockset uses n-bit fixed point numbers (twos complement optional)
- Thus, a conversion is required when Xilinx blocks communicate with Simulink blocks (Xilinx Blockset $\longrightarrow$ MATLAB I/O $\longrightarrow$ Gateway In/Out)



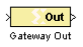


Gateway In



- The Gateway In block support parameters to control the conversion from double-precision to N-bit Boolean, signed (2's complement), or unsigned fixed-point precision
- During conversion the block provides options to handle extra bits
- Defines top-level input ports in the HDL design generated by System Generator
- Defines testbench stimuli when the Create Testbench box is checked in the System Generator block
- Names the corresponding port in the top level HDL entity

Gateway Out



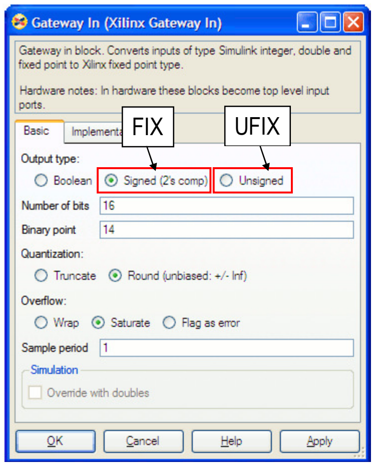
- The Gateway Out block converts data from System Generator fixed point type to Simulink double
- Defines I/O ports for the top level of the HDL design generated by System Generator
- Names the corresponding output port on the top level HDL entity provided the option is selected

Outline

- Gateway In/Gateway Out
- **Data Types**
- Constructing Design Using Xilinx Design Capture
- System Generator Block
- HDL Co-Simulation
- Hardware Verification
- Summary

FIX and UFIX Types

- FIX data type produces a signed twos complement number
- UFIX data type produces unsigned number
- When the output of a block is user defined, the number is further conditioned according to the selected Quantization and Overflow options



Gateway in block. Converts inputs of type Simulink integer, double and fixed point to Xilinx fixed point type.

Hardware notes: In hardware these blocks become top level input ports.

Output type: Boolean [✓ Signed (2's comp)] Unsigned

Number of bits 16

Binary point 14

Quantization: Truncate [✓ Round (unbiased: +/- Inf)]

Overflow: Wrap [✓ Saturate] Flag as error

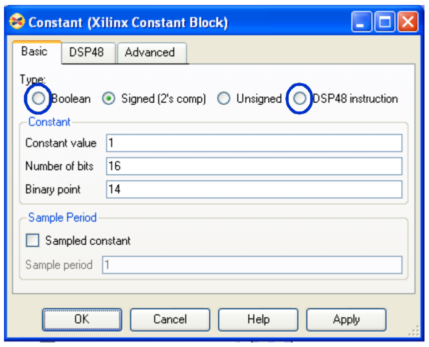
Sample period 1

Simulation

☐ Override with doubles

Boolean and DSP48 Types

- The Xilinx blockset also uses the type Boolean for control ports, such as CE and RESET
- The Boolean type is a variant of the one-bit unsigned number in that it will always be defined (High or Low).
  - A one-bit unsigned number can become invalid; a Boolean type cannot
- The DSP48 type is accessible when you parameterize a constant—it is helpful when driving the OPMODE input of the DSP48 block



Type: Boolean [✓ Signed (2's comp)] Unsigned DSP48 instruction

Constant

Constant value: 1

Number of bits: 16

Binary point: 14

Sample Period

Sampled constant

Sample period: 1

Knowledge Check

Using the technique below, convert the following fractional values

- Define the format of the following twos complement binary fraction and calculate the value it represents

1	1	0	0	0	1	1	0	1	0	1	1

●

Format = < _ _ >

Value =

- What format should be used to represent a signal that has:

a) Max value: +1
Min value: -1
Quantized to 12-bit data

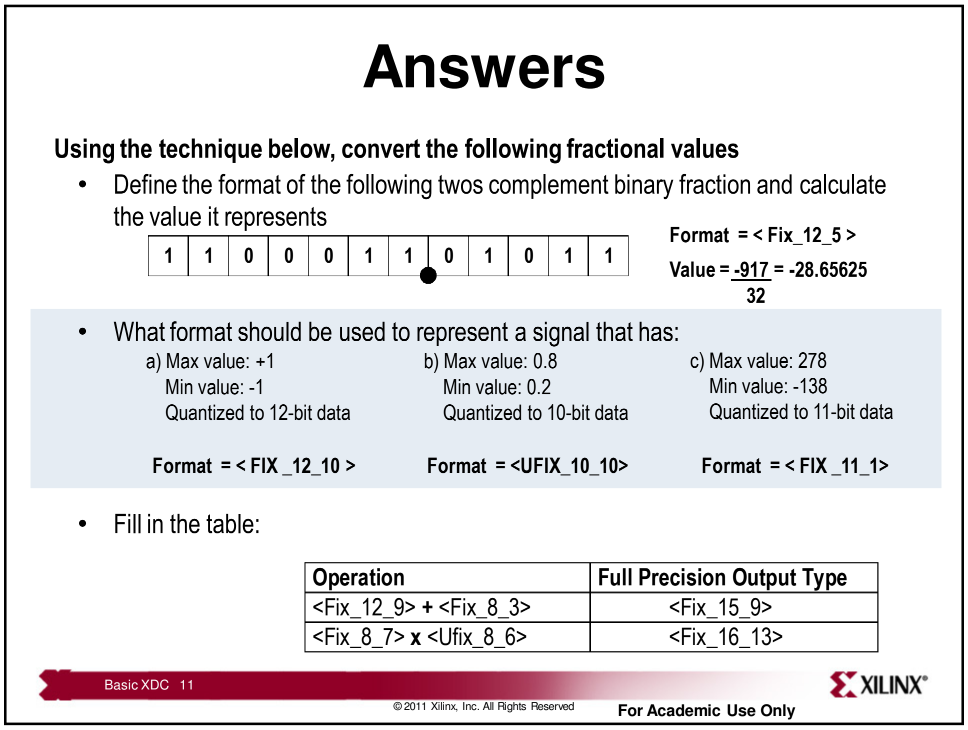
b) Max value: 0.8
Min value: 0.2
Quantized to 10-bit data

c) Max value: 278
Min value: -138
Quantized to 11-bit data

Format = < _ _ >   **Format** = < _ _ >   **Format** = < _ _ >

- Fill in the table:

Operation	Full Precision Output Type
<Fix_12_9> **+** <Fix_8_3>	
<Fix_8_7> **x** <Ufix_8_6>	



Answers

Using the technique below, convert the following fractional values

- Define the format of the following twos complement binary fraction and calculate the value it represents

1	1	0	0	0	1	1 ●	0	1	0	1	1

Format = < Fix_12_5 >

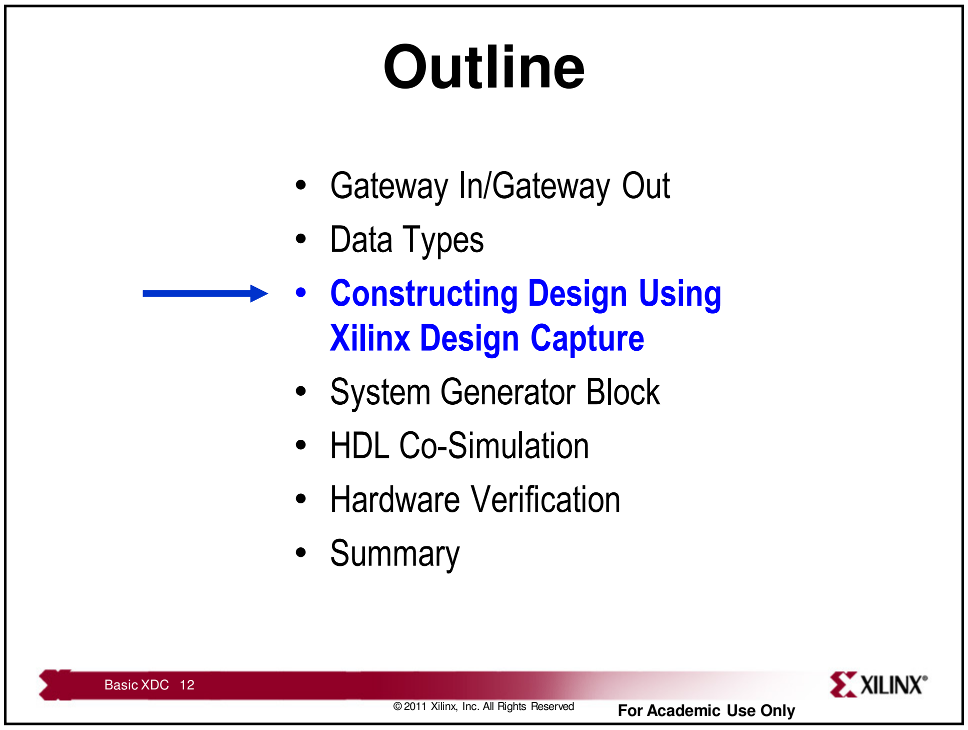
Value = $\frac{-917}{32}$ = -28.65625

- What format should be used to represent a signal that has:

a)	b)	c)
Max value: +1 Min value: -1 Quantized to 12-bit data	Max value: 0.8 Min value: 0.2 Quantized to 10-bit data	Max value: 278 Min value: -138 Quantized to 11-bit data
Format = < FIX _12_10 >	**Format = <UFIX_10_10>**	**Format = < FIX _11_1>**

- Fill in the table:

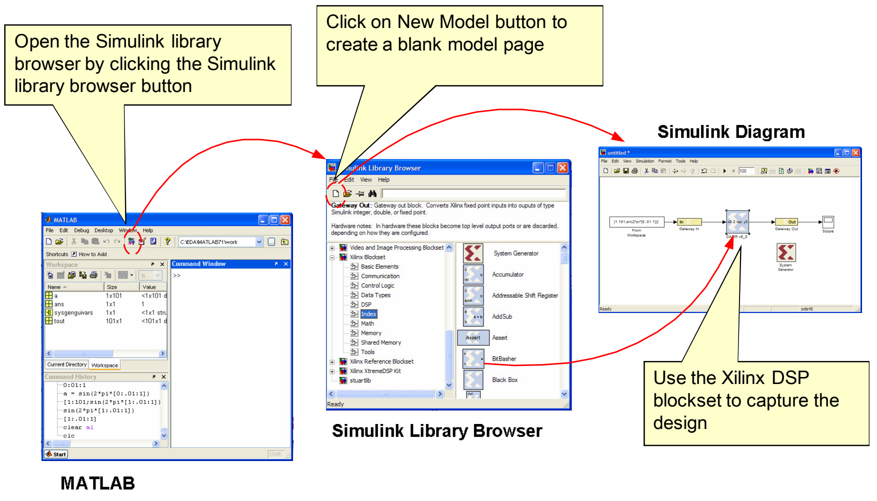
Operation	**Full Precision Output Type**
<Fix_12_9> **+** <Fix_8_3>	<Fix_15_9>
<Fix_8_7> **x** <Ufix_8_6>	<Fix_16_13>



Outline

- Gateway In/Gateway Out
- Data Types
- **Constructing Design Using Xilinx Design Capture**
- System Generator Block
- HDL Co-Simulation
- Hardware Verification
- Summary

Creating a System Generator Design

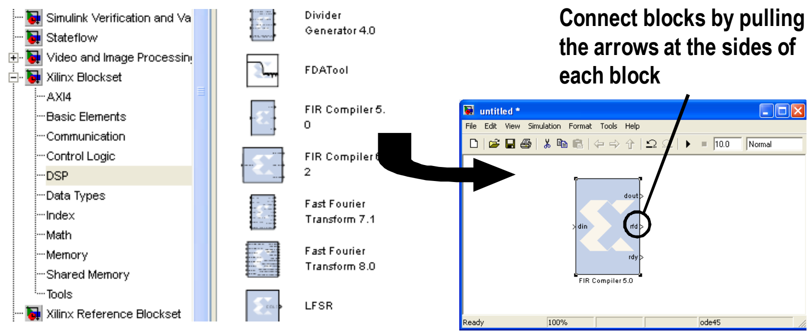

Open the Simulink library browser by clicking the Simulink library browser button

Click on New Model button to create a blank model page

Use the Xilinx DSP blockset to capture the design

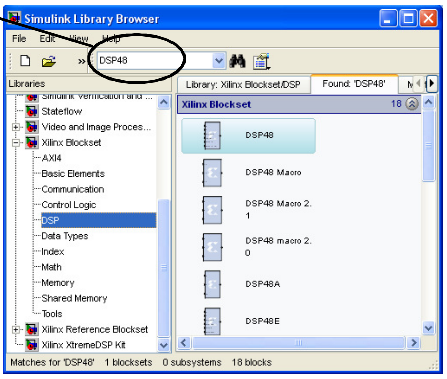
Creating a System Generator Design

- Build the design by dragging and dropping blocks from the Xilinx blockset onto your new sheet
- Design entry is similar to a schematic editor


Connect blocks by pulling the arrows at the sides of each block

Finding Blocks

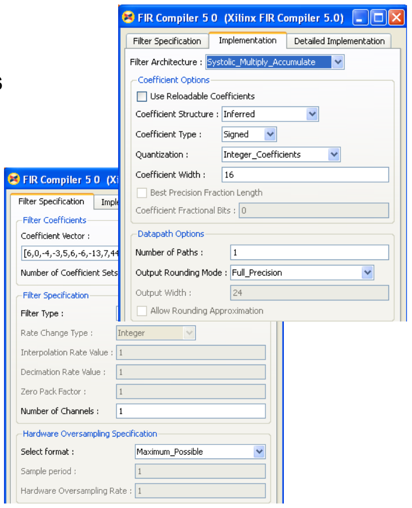
- Use the Find feature to search ALL Simulink libraries
- The Xilinx blockset has eleven major sections
  - AXI4: FFT, VDMA
  - Basic elements: counters, delays
  - Communication: error correction blocks
  - Control Logic: MCode, black box
  - DSP: FDATool, FFT, FIR
  - Data Types: convert, slice
  - Index: all Xilinx blocks (a quick way to view all blocks)
  - Math: multiply, accumulate, inverter
  - Memory: dual port RAM, single port RAM
  - Shared memory: FIFO
  - Tools: ModelSim, resource estimator



Matches for 'DSP48'   1 blocksets   0 subsystems   18 blocks

Configuring Your Blocks

- Double-click or go to Block Parameters to view and change the configurable parameters of a block using multi-tabbed GUI
- Number of tabs and type of configurable parameters under each tab is block dependent
- Some common parameters are:
  - Precision: User defined or full precision
  - Arithmetic Type: Unsigned or twos complement
  - Number of Bits: total and fraction
  - Overflow and quantization: Saturate or wrap overflow, truncate or round quantization
  - Latency: Specify the delay through the block
- **Note**: While all parameters can be simulated, not all are realizable

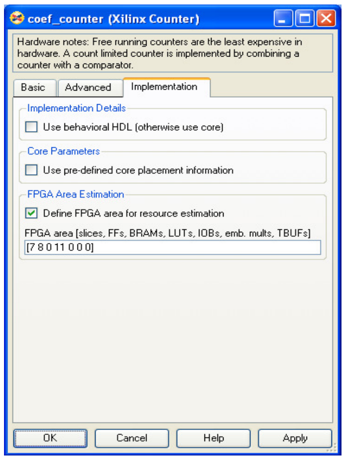


Filter Architecture : Systolic_Multiply_Accumulate
Use Reloadable Coefficients
Coefficient Structure : Inferred
Coefficient Type : Signed
Quantization : Integer_Coefficients
Coefficient Width : 16
Best Precision Fraction Length
Coefficient Fractional Bits : 0
Number of Paths : 1
Output Rounding Mode : Full_Precision
Output Width : 24
Allow Rounding Approximation

Coefficient Vector : [6,0,-4,-3,5,6,-6,-13,7,44
Number of Coefficient Sets
Filter Type :
Rate Change Type : Integer
Interpolation Rate Value : 1
Decimation Rate Value : 1
Zero Pack Factor : 1
Number of Channels : 1
Select format : Maximum_Possible
Sample period : 1
Hardware Oversampling Rate : 1

Configuring Your Blocks

- Some common parameters are:
  - Provide Reset and Enable Ports
  - Sampling Period: Can be inherited with a “-1” or must be an integer value
  - Use Behavioral HDL
  - Use Placement Information for Core
  - FPGA Area/Use Area Above For Estimation
    - Slices
    - FFs
    - LUTs
    - IOBs
    - Embedded Mults
    - TBUFs
- **Note**: While all parameters can be simulated, not all are realizable



Hardware notes: Free running counters are the least expensive in hardware. A count limited counter is implemented by combining a counter with a comparator.

Implementation Details
☐ Use behavioral HDL (otherwise use core)

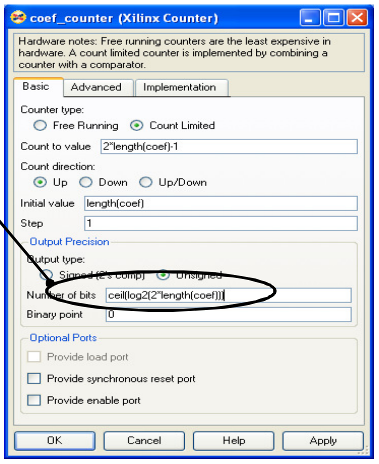
Core Parameters
☐ Use pre-defined core placement information

FPGA Area Estimation
☑ Define FPGA area for resource estimation

FPGA area [slices, FFs, BRAMs, LUTs, IOBs, emb. mults, TBUFs]: [7 8 0 11 0 0 0]

Values Can Be MATLAB Equations

- You can also enter equations in the block parameters, which can aid in calculation and in your understanding of the model parameters
- The equations are calculated at the beginning of a simulation
- Useful MATLAB™ operators
  - + add
  - - subtract
  - * multiply
  - / divide
  - ^ power
  - pi π(3.1415926535897….)
  - exp(x) exponential (ex)



Hardware notes: Free running counters are the least expensive in hardware. A count limited counter is implemented by combining a counter with a comparator.

Counter type: Free Running [✓ Count Limited]

Count to value: 2*length(coef)-1

Count direction: [✓ Up] Down Up/Down

Initial value: length(coef)

Step: 1

Output Precision

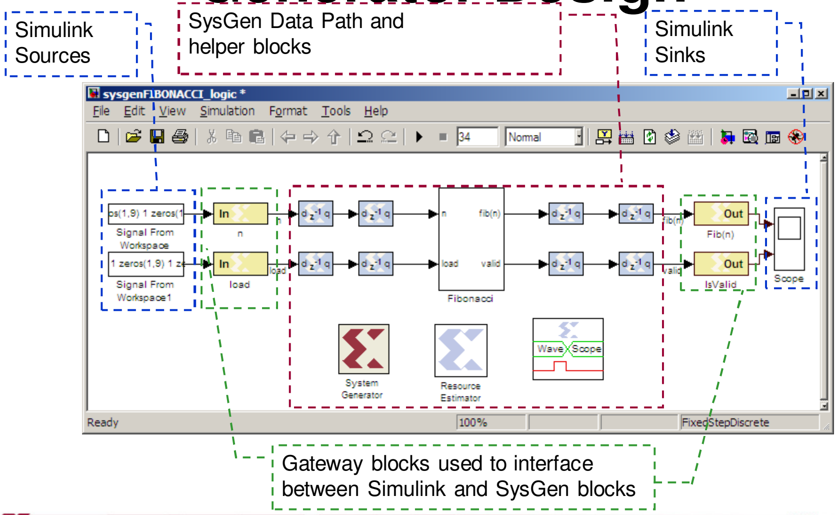
Output type: Signed (2's comp) [✓ Unsigned]

Number of bits: ceil(log2(2*length(coef)))

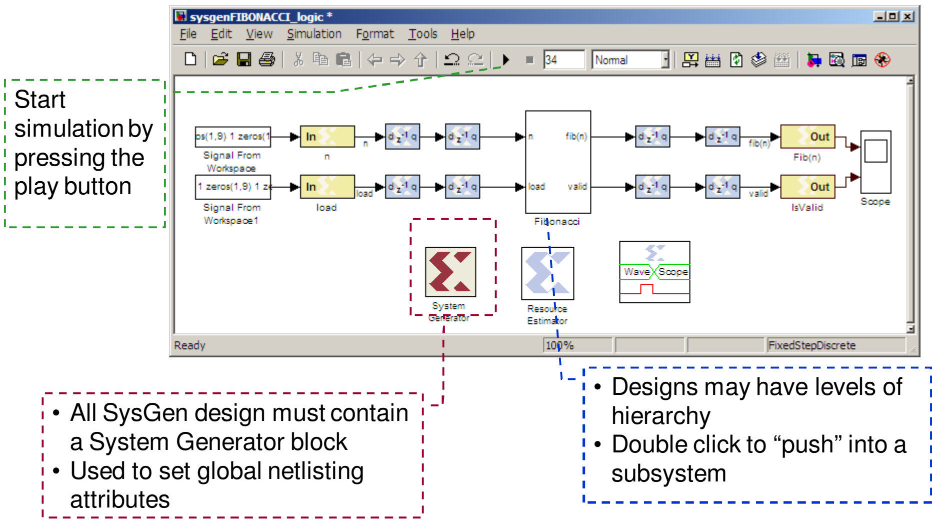
Binary point: 0

Optional Ports
☐ Provide load port
☐ Provide synchronous reset port
☐ Provide enable port

Creating a System Generator Design



System Generator Design



Start simulation by pressing the play button

- All SysGen design must contain a System Generator block
- Used to set global netlisting attributes

- Designs may have levels of hierarchy
- Double click to "push" into a subsystem

Outline

- Gateway In/Gateway Out
- Data Types
- Constructing Design Using Xilinx Design Capture
- **System Generator Block**
- HDL Co-Simulation
- Hardware Verification
- Summary

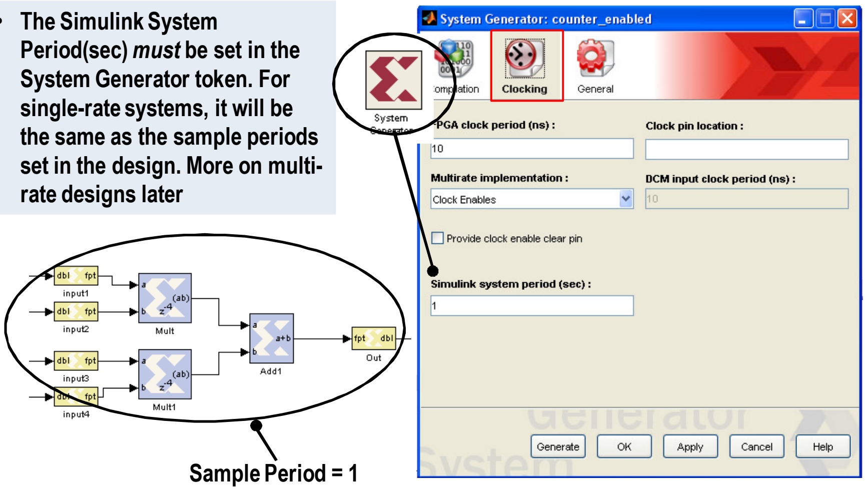
Sample Period

- Every System Generator signal must be "sampled"; transitions occur at equidistant discrete points in time, called *sample times*
- Each block in a Simulink design has a "sample period," and it corresponds to how often the function of that block is calculated and the results outputted
- The sample period of a block *directly* relates to how that block will be clocked in the actual hardware
- This sample period must be set explicitly for:
  - Gateway In
  - Blocks without inputs (**Note**: constants are idiosyncratic)
- The sample period can be "derived" from the input sample times for other blocks
- Remember Nyquist's theorem ($F_s \geq 2f_{max}$) when setting sample periods

System Generator Token

Setting the Global Sample Period

- The Simulink System Period(sec) *must* be set in the System Generator token. For single-rate systems, it will be the same as the sample periods set in the design. More on multi-rate designs later

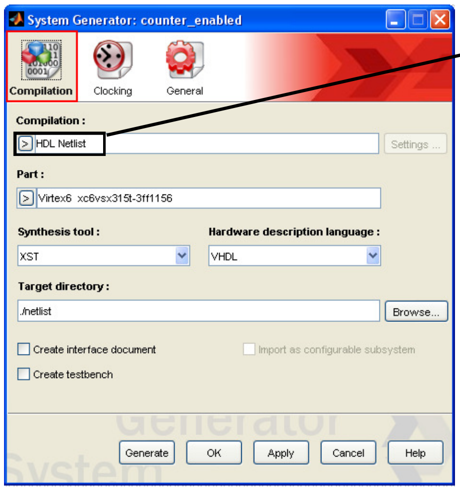


FPGA clock period (ns) : 10
Clock pin location :
Multirate implementation : Clock Enables
DCM input clock period (ns) : 10
Provide clock enable clear pin
Simulink system period (sec) : 1

Sample Period = 1

System Generator Token

Selecting a compilation target



Compilation : HDL Netlist
Part : Virtex6 xc6vsx315t-3ff1156
Synthesis tool : XST
Hardware description language : VHDL
Target directory : ./netlist
Create interface document
Import as configurable subsystem
Create testbench

Speed up simulation
- Various varieties of hardware co-simulation

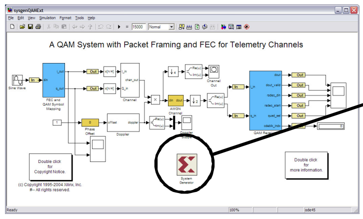
Generate Hardware
- HDL Netlist, NGC Netlist, Bitstream

Analyze Performance
- Timing Analysis

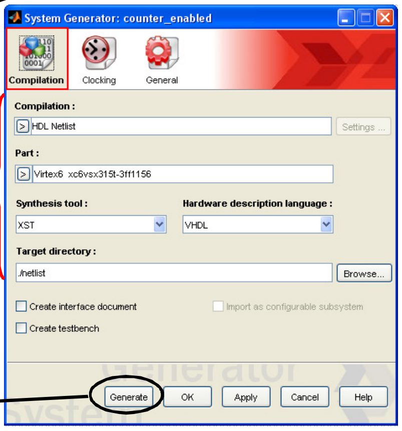
Connect in a larger design
- Export as pcore to EDK
- Connect in ISE Foundation

System Generator Token

Generating HDL Code


A QAM System with Packet Framing and FEC for Telemetry Channels

Once complete, double-click the System Generator token



Compilation : HDL Netlist

Part : Virtex6 xc6vsx315t-3ff1156

Synthesis tool : XST

Hardware description language : VHDL

Target directory : ./netlist

☐ Create interface document

☐ Import as configurable subsystem

☐ Create testbench

- Specify the implementation Parameters
  - **HDL Netlist** as the compilation mode
  - Select the target part
  - Set HDL language
  - Set the **FPGA Clock Period** (in Clocking tab)
  - Check **Create Testbench**
- **Generate** the HDL

System Generator

Output Files

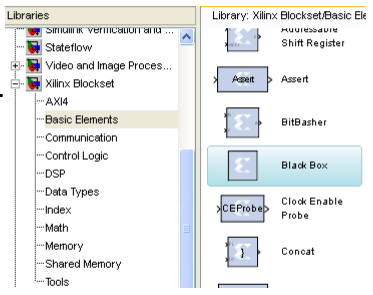
- Design files
  - VHD or V (HDL design files)
  - EDN or NGC (core implementation file and netlist file)
  - XCF (Xilinx constraints file for timing constraints)
- Project files
  - ISE (Project Navigator project file)
  - SGP (System Generator project file for Project Navigator)
  - TCL (scripts for Synplify and Leonardo project creation)
- Simulation files
  - DO (simulation scripts for MTI)
  - DAT (data files containing the test vectors from System Generator)
  - _tb.VHD or _tb.V (simulation testbench)

Outline

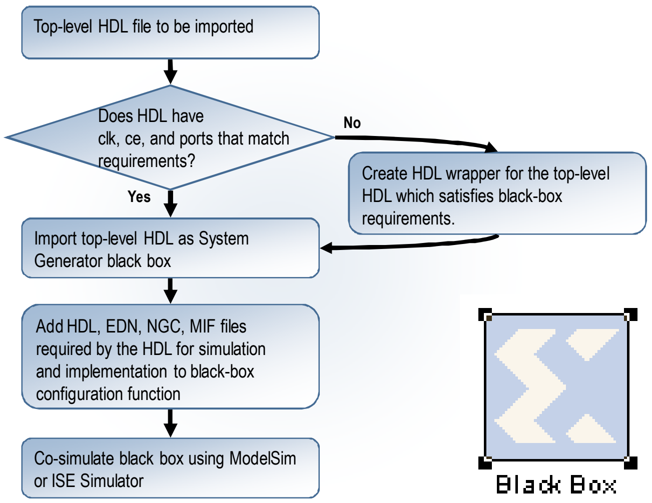
- Gateway In/Gateway Out
- Data Types
- Constructing Design Using Xilinx Design Capture
- System Generator Block
- **HDL Co-Simulation**
- Hardware Verification
- Summary

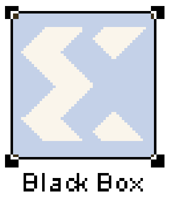
Black Box

- Allows a way to import HDL models into System Generator
- Allows co-simulation of black box HDL with Simulink by using either the ModelSim or the ISE® Simulator tool
- Integrates the imported HDL and implementation files (EDN, NGC) with the netlist generated from System Generator

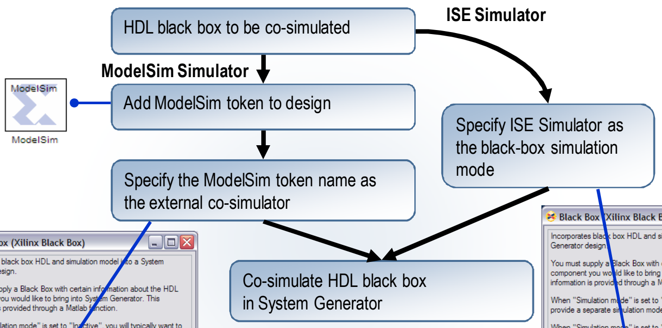


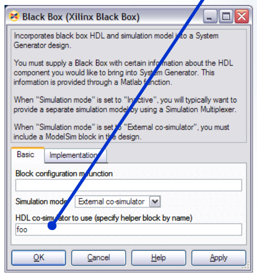
HDL Import Flow


Top-level HDL file to be imported
Does HDL have clk, ce, and ports that match requirements?
No
Create HDL wrapper for the top-level HDL which satisfies black-box requirements.
Yes
Import top-level HDL as System Generator black box
Add HDL, EDN, NGC, MIF files required by the HDL for simulation and implementation to black-box configuration function
Co-simulate black box using ModelSim or ISE Simulator



HDL Co-Simulation Flow


HDL black box to be co-simulated
ISE Simulator
ModelSim Simulator
ModelSim
Add ModelSim token to design
Specify the ModelSim token name as the external co-simulator
Specify ISE Simulator as the black-box simulation mode
Co-simulate HDL black box in System Generator



Black Box (Xilinx Black Box)

Incorporates black box HDL and simulation model into a System Generator design.

You must supply a Black Box with certain information about the HDL component you would like to bring into System Generator. This information is provided through a Matlab function.

When "Simulation mode" is set to "Inactive", you will typically want to provide a separate simulation model by using a Simulation Multiplexer.

When "Simulation mode" is set to "External co-simulator", you must include a ModelSim block in the design.

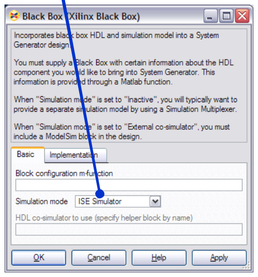
Basic | Implementation

Block configuration m-function:

Simulation mode: External co-simulator

HDL co-simulator to use (specify helper block by name): foo

OK | Cancel | Help | Apply



Black Box (Xilinx Black Box)

Incorporates black box HDL and simulation model into a System Generator design.

You must supply a Black Box with certain information about the HDL component you would like to bring into System Generator. This information is provided through a Matlab function.

When "Simulation mode" is set to "Inactive", you will typically want to provide a separate simulation model by using a Simulation Multiplexer.

When "Simulation mode" is set to "External co-simulator", you must include a ModelSim block in the design.

Basic | Implementation

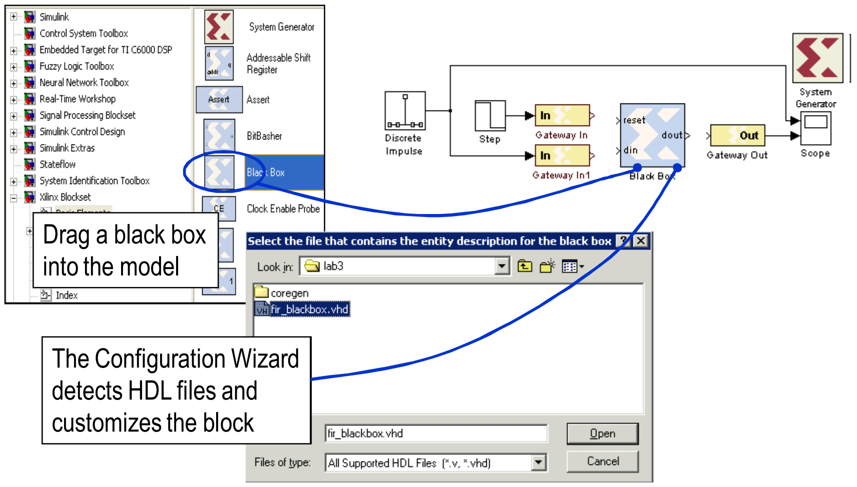
Block configuration m-function:

Simulation mode: ISE Simulator

HDL co-simulator to use (specify helper block by name):

OK | Cancel | Help | Apply

HDL Co-Simulation (Step 1)



Select the file that contains the entity description for the black box

File name: fir_blackbox.vhd

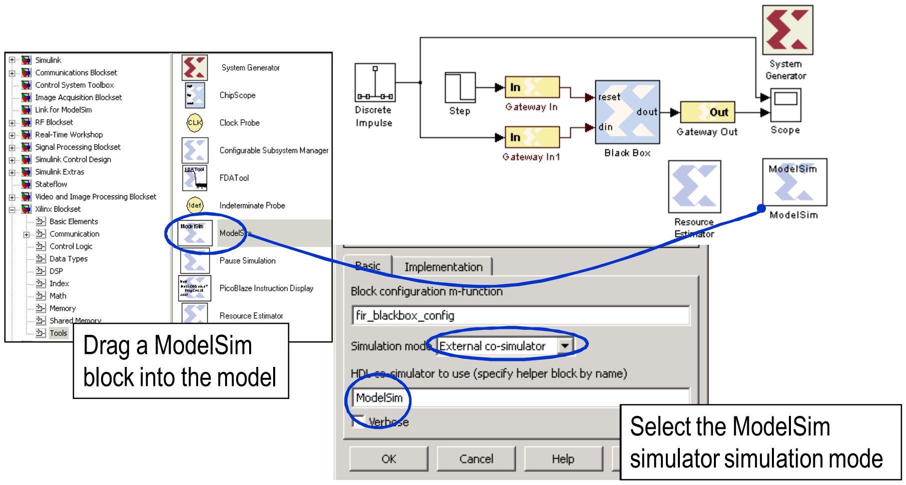
Files of type: All Supported HDL Files (*.v, *.vhd)

Drag a black box into the model

The Configuration Wizard detects HDL files and customizes the block

HDL Co-Simulation (Step 2)

ModelSim Simulator



Block configuration m-function: fir_blackbox_config

Simulation mode: External co-simulator

HDL co-simulator to use (specify helper block by name): ModelSim

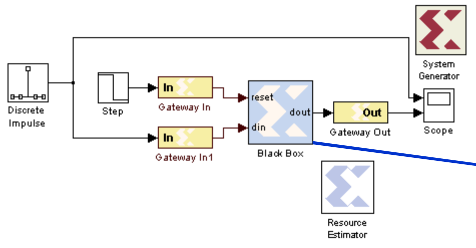
Verbose

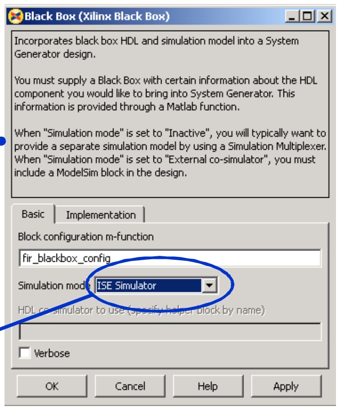
Drag a ModelSim block into the model

Select the ModelSim simulator simulation mode

HDL Co-Simulation (Step 2)

ISE Simulator





Incorporates black box HDL and simulation model into a System Generator design.

You must supply a Black Box with certain information about the HDL component you would like to bring into System Generator. This information is provided through a Matlab function.

When "Simulation mode" is set to "Inactive", you will typically want to provide a separate simulation model by using a Simulation Multiplexer. When "Simulation mode" is set to "External co-simulator", you must include a ModelSim block in the design.

Basic | Implementation

Block configuration m-function: fir_blackbox_config

Simulation mode: ISE Simulator

HDL co-simulator to use (specify helper block by name):

Verbose

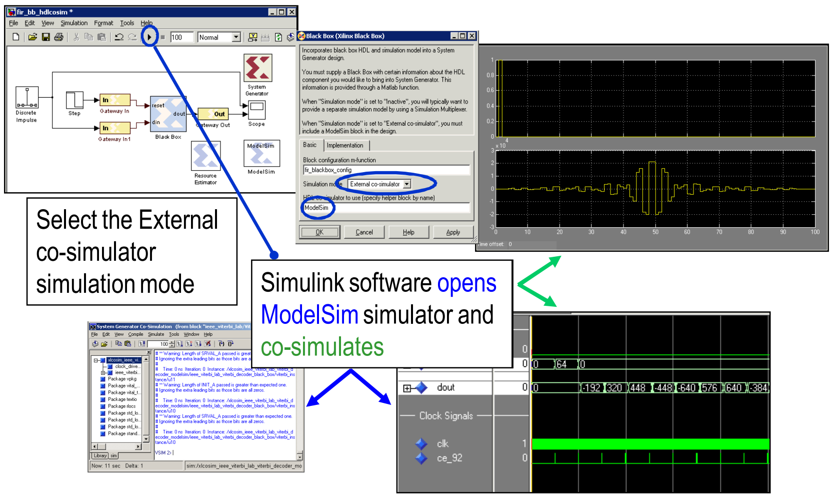
Select the ISE Simulator simulation mode

XILINX®

For Academic Use Only

HDL Co-Simulation (Step 3)

ModelSim Simulator



Block configuration m-function: fir_blackbox_config

Simulation mode: External co-simulator

HDL co-simulator to use (specify helper block by name): ModelSim

Select the External co-simulator simulation mode

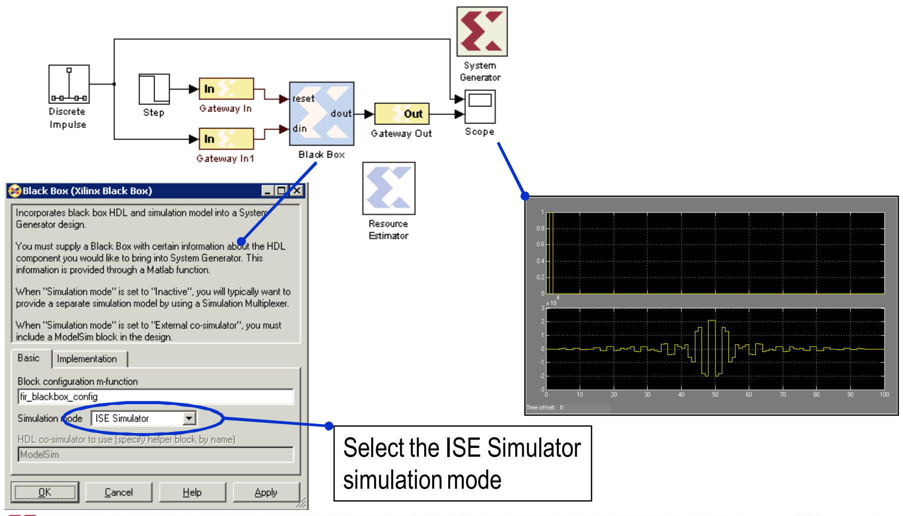
Simulink software opens ModelSim simulator and co-simulates

XILINX®

For Academic Use Only

HDL Co-Simulation (Step 3)

ISE Simulator



Incorporates black box HDL and simulation model into a System Generator design.

You must supply a Black Box with certain information about the HDL component you would like to bring into System Generator. This information is provided through a Matlab function.

When "Simulation mode" is set to "Inactive", you will typically want to provide a separate simulation model by using a Simulation Multiplexer.

When "Simulation mode" is set to "External co-simulator", you must include a ModelSim block in the design.

Block configuration m-function: fir_blackbox_config

Simulation mode: ISE Simulator

HDL co-simulator to use (specify helper block by name): ModelSim

Select the ISE Simulator simulation mode

Outline

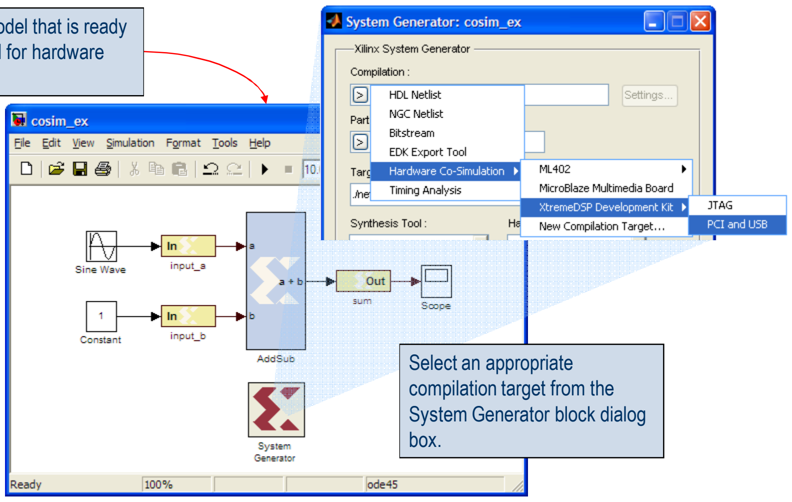
- Gateway In/Gateway Out
- Data Types
- Constructing Design Using Xilinx Design Capture
- System Generator Block
- HDL Co-Simulation
- **Hardware Verification**
- Summary

Why Hardware Co-simulation?

- Ease of use - users do not need to know:
  - VHDL/Verilog
  - How to run the Xilinx tools
  - Details about the FPGA architecture
  - 3rd party boards API
  - Device driver details
- Hardware verification
- And of course, speedup

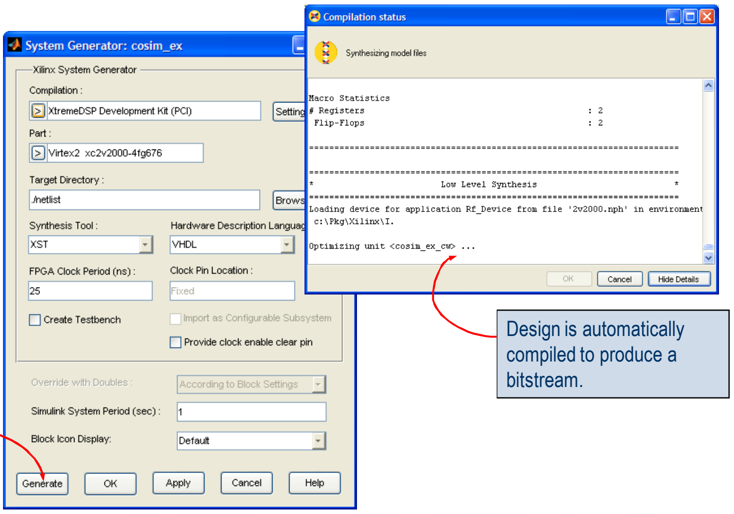
Choosing a Compilation Target

Start with a model that is ready to be compiled for hardware co-simulation.



Select an appropriate compilation target from the System Generator block dialog box.

Design Compilation



System Generator: cosim_ex

Xilinx System Generator

Compilation : XtremeDSP Development Kit (PCI)

Part : Virtex2 xc2v2000-4fg676

Target Directory : ./netlist

Synthesis Tool : XST

Hardware Description Languag: VHDL

FPGA Clock Period (ns) : 25

Clock Pin Location : Fixed

☐ Create Testbench

☐ Import as Configurable Subsystem

☐ Provide clock enable clear pin

Override with Doubles : According to Block Settings

Simulink System Period (sec) : 1

Block Icon Display: Default

Compilation status

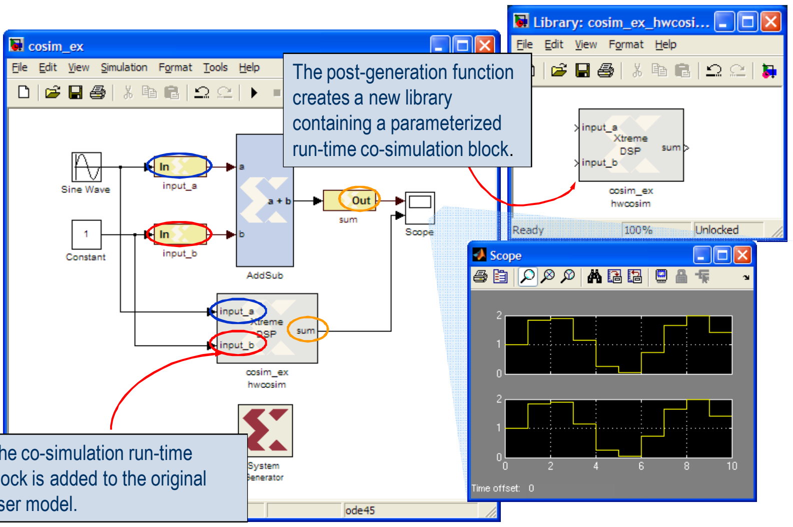
Synthesizing model files

```
Macro Statistics
# Registers                                            : 2
 Flip-Flops                                            : 2

=========================================================================


=========================================================================
*                        Low Level Synthesis                            *
=========================================================================
Loading device for application Rf_Device from file '2v2000.nph' in environment
 c:\Pkg\Xilinx\I.

Optimizing unit <cosim_ex_cw> ...
```

User presses the `Generate` button.

Design is automatically compiled to produce a bitstream.

Run-time Co-simulation Blocks



The post-generation function creates a new library containing a parameterized run-time co-simulation block.

The co-simulation run-time block is added to the original user model.

Choosing an Interface

- PCI/PCMCIA
  - Specialized co-simulation interfaces for a handful of boards (i.e., not a general solution)
  - Fastest co-simulation solution.
  - Ideally suited for high-bandwidth co-simulation applications
- JTAG (Parallel/USB)
  - Support for *any* board with a Xilinx FPGA, JTAG header, and clock source
  - Burst-transfer support
    - 1 Mbps down to the board
    - 0.5 Mbps back from the board
- Ethernet
  - Point-to-point
  - Network-based

Ethernet Hardware Co-simulation

- Two flavors:
  - Network-based
    - Remote access
    - 10/100/1000 Base-T
    - Ethernet-based configuration
  - Point-to-Point
    - Requires a direct connection between host PC and FPGA
    - 10/100/1000 Base-T
    - Ethernet configuration

Outline

- Gateway In/Gateway Out
- Data Types
- Constructing Design Using Xilinx Design Capture
- System Generator Block
- HDL Co-Simulation
- Hardware Verification
- **Summary**

Knowledge Check

Knowledge Check

- Describe the steps involved in hardware-in-the-loop verification

Answers

- Describe the steps involved in hardware-in-the-loop verification
  - Hardware-in-the-loop is a Simulink hardware accelerator which enables design verification in hardware. It is a Simulink-to-bitstream-to-Simulink push-button flow to simulate HDL-based and EDIF-based design
  - Three simple steps
    - Select the target board as a compilation option in the System Generator token
    - Compile (Generate) the design for the co-simulation
    - Copy a co-simulation run-time block into the user model

Knowledge Check

- List data types supported by the Xilinx System Generator blocks

Answers

- List data types supported by the Xilinx System Generator blocks
  - Twos complement signed (FIX)
  - Unsigned (UFIX)
  - Boolean
  - DSP48

Summary

- Every System Generator design interfaces with Simulink sources/sinks/blocks using Gateway In and Gateway Out blocks
- Gateway In block when realized in hardware provides input port at a top hierarchy level
- Gateway Out block when realized in hardware provides output port at a top level hierarchy level
- Hardware-in-the-loop enables verification of the design using hardware

Where Can I Learn More?

- Tool documentation
  - *Simulink Browser → Help → Simulink Help → Xilinx System Generator*
    - *A Tutorial Introduction*
    - *Using FPGA Hardware in the Loop*
    - *Hardware Design Using System Generator → System Level Modeling*
    - *Hardware Design Using System Generator → Automatic Code Generation*
- Examples
  - <Matlab_install>\toolbox\xilinx\sysgen\examples
- Support Website
  - DSP Website: http://www.xilinx.com/dsp