FPGA Prototyping Using Agilent SystemVue

Application Note

Introduction

This application note outlines a design flow for Field Programmable Gate Array (FPGA) prototyping, using the Agilent SystemVue software, as well as third-party applications that integrate well with SystemVue. The SystemVue FPGA flow can be used to quickly validate communications digital signal processing (DSP) algorithms and accelerate physical layer (PHY) performance measurements, such as bit-error-rate (BER). Although specific applications and hardware platforms are named in this case study, the flow generally applies to a variety of platforms and vendors.

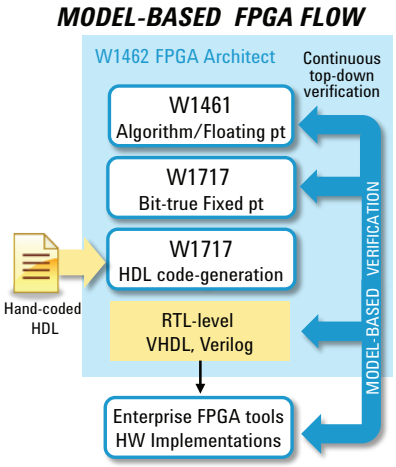

Figure 1. Overview of the FPGA design flow in SystemVue

Figure 1 illustrates the general SystemVue FPGA design flow and features. It also highlights the associated tools that can be used in each stage of the flow.

A more detailed, self-paced tutorial for VHDL/Verilog hardware design in SystemVue can be found in the "Tutorials" section of the SystemVue release 2012.06 or later documentation. Supported customers may also download the HDL tutorial online at: **http://edocs.soco.agilent.com/display/sv201206/Getting+Started+with+Hardware+Design**

Also in this application note, a military communication receiver and BER measurement application example is used to demonstrate a model-based design approach to a software-defined radio (SDR) flow that moves from system-level architecture to hardware verification. A conceptual diagram of this flow is shown Figure 2.

Introduction

The design process diagrammed in Figure 2 consists of the following steps:

1. System design and validation in floating point
2. System design and validation in fixed point
3. HDL code generation
4. HDL validation using co-simulation with Aldec Riviera-PRO
5. Generation of the FPGA programming file
6. Loading the .bit file into the FPGA
7. Generation of test signals for FPGA receiver test
8. Testing of the FPGA

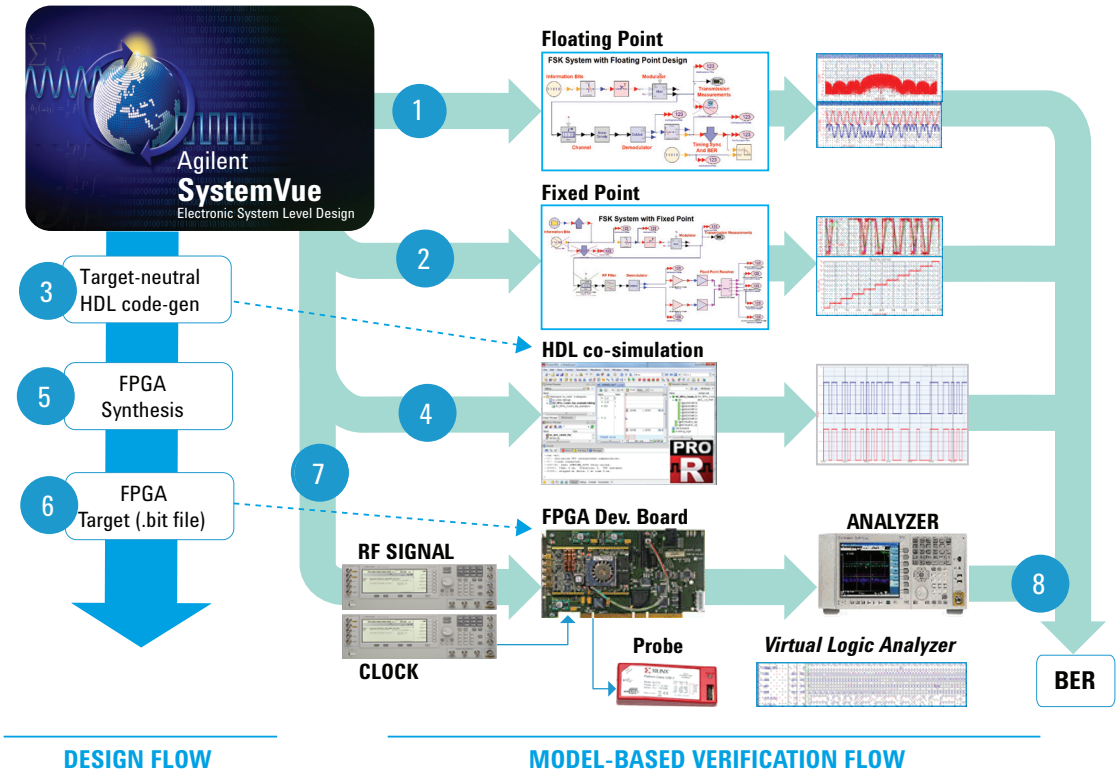

Figure 2. Conceptual diagram of the existing SDR FPGA flow

This process uses standard SystemVue libraries in both floating and fixed point for design and verification, along with FPGA implementation tools from Xilinx and Aldec. With this flow, FPGA-based communications and radar systems can be designed, verified and implemented.

About the design: A real-time FSK BER tester

To illustrate the FPGA prototyping flow in more detail, a BER tester for a Frequency Shift-Keying (FSK) receiver architecture will be taken from a system-level concept down to an FPGA implementation on a common, external development board. The BER tester recovers data bits within an FSK system and compares them against a known input test vector. It outputs both the recovered bit stream, as well as a running tally of any errors. In addition to the testing and I/O functions, some timing synchronization algorithms are also included. Supported SystemVue customers can download these files for SystemVue 2012.06 or later at: **http://edocs.soco.agilent.com/display/eesofkcsysvue/FPGA+flow.**

Step 1

System Design and Validation in Floating Point

The first step in the design process is to create a working model that can be used to validate the receiver algorithms, first under ideal conditions, and then under a variety of stressful conditions (e.g., impairments and noise). This initial architectural and algorithmic modeling is done using floating-point models in any of several formats: built-in graphical blocks, C++ or math language (.m).

The floating-point design for the FSK system, including transmitter, communications channel and receiver, is shown in Figure 3. Here, the transmitter portion of the system begins with generation of the data payload. The resulting bit stream is then mapped with FSK. Transmitter performance can be characterized at this point, including waveform and spectrum measurements. (The Agilent 89600 VSA software can also be used for additional analysis capability on simulated results, like these.) The bit stream then continues into a simple channel model consisting of delay and additive white Gaussian noise (AWGN). A user-defined channel model also can be included to verify a wider variety of test cases.

The receiver design consists of FSK demodulation, de-mapping into baseband bits, timing synchronization, and BER measurement. A correlation is performed to detect the delay between recovered data and original data, which then calibrates the composite delay in the synchronization model.

Simulation is used to validate the overall FSK system. Using internal SystemVue graphs or external 89600 VSA software, the transmitter output is first verified to ensure the time waveform and frequency spectrum are acceptable. For the receiver, we can then observe that the recovered data payload bits are aligned properly with the original bits and BER is zero. All received bits have been completely recovered.

If the floating-point BER result is not zero, the FSK receiver algorithms must be debugged before proceeding to fixed-point or hardware implementation. Since the floating-point design has been validated, we can proceed to Step 2.

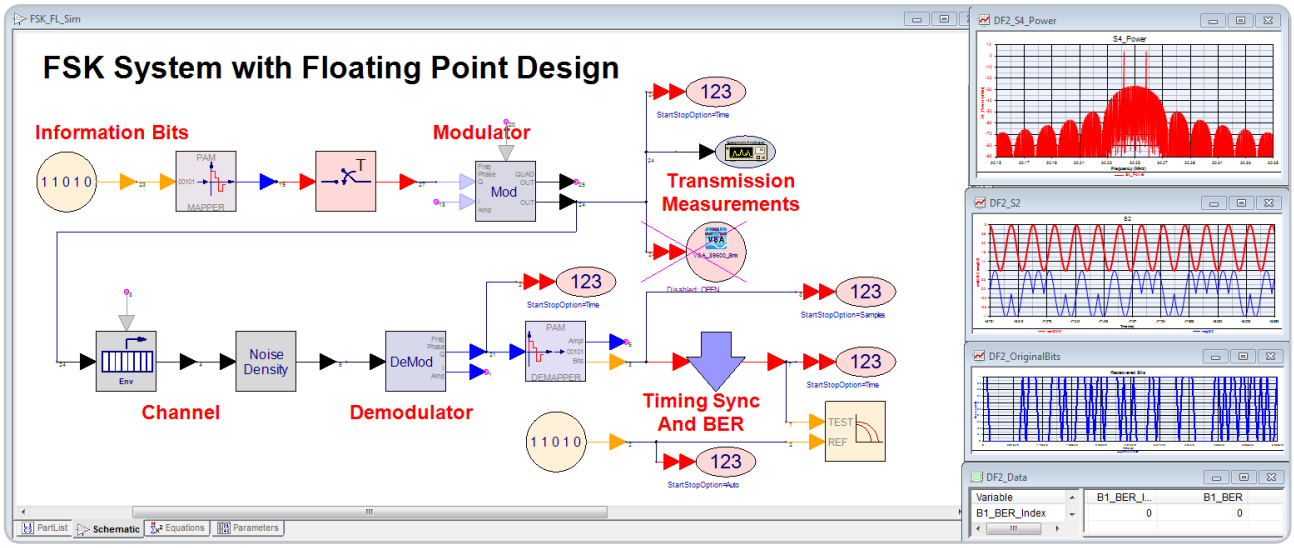

Figure 3. Connecting to test improves the fidelity of system architecture design. Conversely, leveraging the design platform for early R&D validation creates new value for test.

Step 2

System Design and Validation in Fixed Point

The next step on the path to a hardware implementation is to consider the effects of quantization and finite-precision arithmetic on the algorithms. Will they still work? How much precision is required for the specified system performance? Do the calculations introduce latencies into the design that ruin the performance? Are architectural changes needed to perform the calculations faster, or using fewer resources or battery power? Are some functions easier to implement in analog versus digital hardware?

In this case, SystemVue's fixed-point library may be used to consider these questions. The W1717 Hardware Design Kit is a SystemVue design personality that not only includes the fixed-point library and its diagnostic simulation support, but also VHDL/Verilog code-generation.

Because the fixed-point models have well-defined hardware behaviors, such as overflow, underflow and latency, they represent a different simulation datatype from the baseband floating point (blue pin) or timed-envelope (black pin) datatypes shown in Step 1. A datatype converter is necessary to connect the floating-point transmitter and channel sections to the fixed-point models in the receiver. A fixed-point design for the FSK receiver is shown in Figure 4.

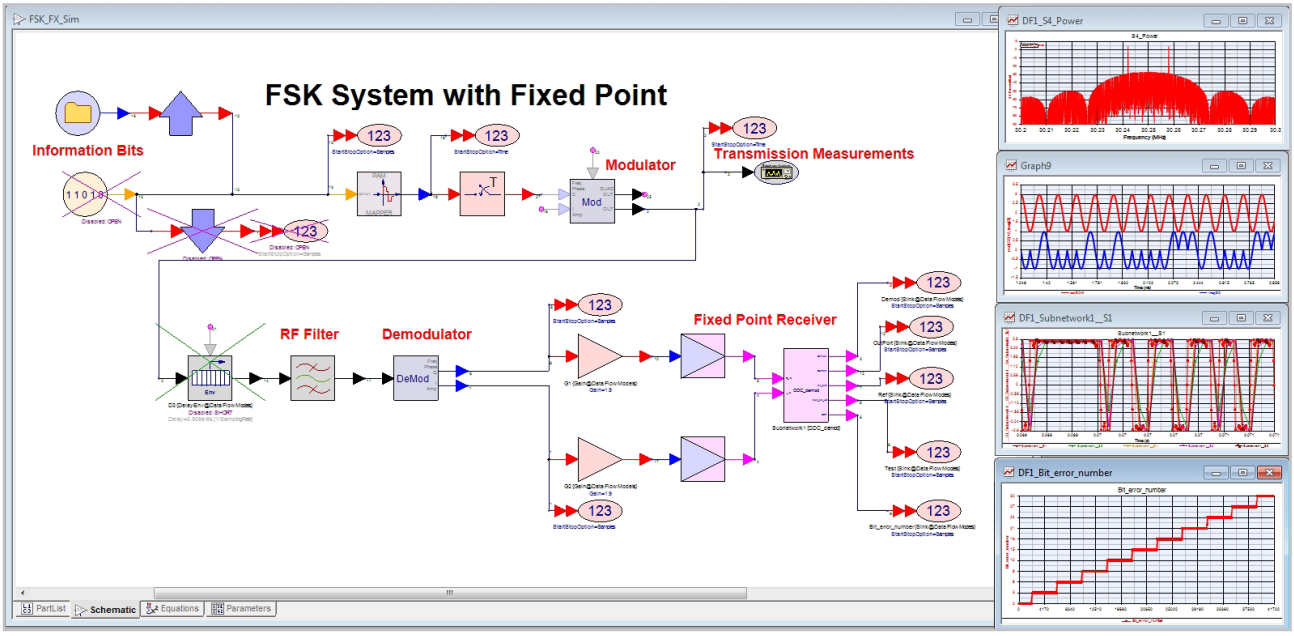

Figure 4. FSK system design in fixed point

Although the transmitter and channel models in Figure 4 are pictured as floating-point simulation blocks, a real radio signal can be substituted for these blocks, using test equipment to capture a live signal and bring it into the simulation at run-time, or by reading a stored waveform out of an external file.

Step 2

System Design and Validation in Fixed Point

The remainder of this application note focuses on the hardware implementation of the baseband receiver and BER tester. The transition from a simple floating-point block to a series of fixed-point blocks is shown in Figure 5.

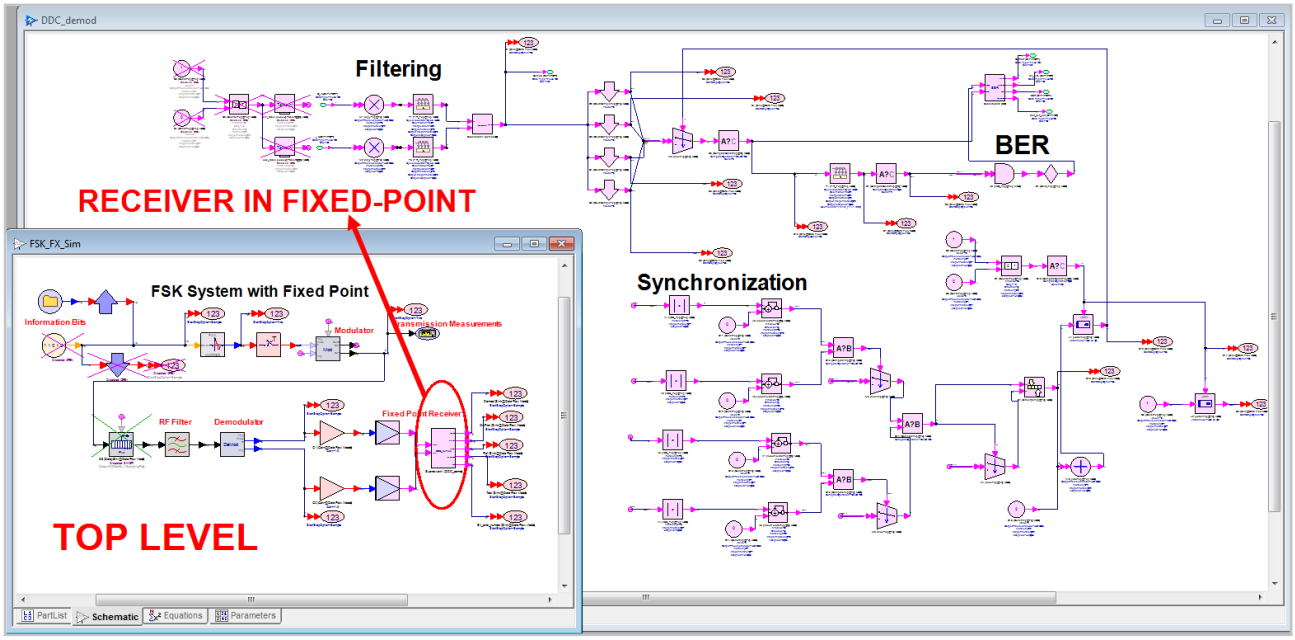

Figure 5. Detailed structure of the FSK receiver design in fixed point

Prior to entering the fixed-point domain, the complex-valued signal is split into separate real-valued I and Q paths, and then digitized. Although SystemVue offers several ADC models with various impairments, an ideal datatype converter is used to cast the floating-point values into fixed-point values. These values have a specific number of bits allocated to the integer and mantissa portions of the numeric stream, and a representation (e.g., signed/unsigned and 2's complement). At this point, the connections between fixed-point components (magenta colored pins) are single schematic wires, not buses. That is, a 16-bit value is one schematic wire, not 16 individual wires grouped into a bus having a value of "1" or "0." This flexibility makes it easy to change precision, or even sweep precision as a system variable, when determining the architectural implications of word length, without altering the schematic. Note that the input and output waveforms of these fixed-point models will be bit-accurate and cycle-accurate, without needing a detailed hardware implementation or commitment to a vendor.

About the FSK receiver algorithms

The timing synchronization algorithm is a key ingredient to making a good receiver in the real world. The design example highlighted in the application note provides just such an algorithm and may be adapted to other applications. The detailed fixed-point receiver design is shown in Figure 5. It consists of three subsections: filtering, synchronization and BER testing.

Step 2

System Design and Validation in Fixed Point

First, the system symbol clock must be recovered so that the sample points occur with the highest accuracy. Once the I and Q receiver inputs are oversampled, the best sample position is decided by the "best sample position decision" (BSPD) algorithm. After finding the best sample position, the received symbol clock is aligned with the original transmit symbol clock. Next, the receiver begins deciding whether the received bit values are 0 or 1. The initial set of recovered bit values are then compared to the beginning of the original bit sequence to determine the absolute time offset between the two bit streams. Note that in the presence of bit errors, this correlation will not be exact.

In this BER tester, the original bit stream was saved to a file and is read into the simulation as a reference. When the time offset has been corrected, the received bit sequence and the original sequence are now time-aligned, and the receiver can begin to accumulate the number of bit errors for the BER calculation.

As in Step 1, the BER of the fixed-point architecture is validated under ideal conditions and then under a range of non-ideal conditions. If the results are not what were expected, the user can use the SystemVue platform to debug the fixed-point algorithm, or return to Step 1 to try other floating-point algorithms. Although hardware-like behaviors have been accounted for, the generic fixed-point design could still have a variety of hardware implementations, using a variety of FPGA, ASIC and embedded processor vendors.

Once the fixed-point design has been validated, the design process can progress to Step 3.

Step 3

HDL Code Generation

The third step is to implement the fixed-point design as a register-transfer-level (RTL) hardware design, in a behavioral modeling language such as VHDL or Verilog. Conveniently, the models in SystemVue's W1717 fixed-point library directly support HDL code-generation. The procedure is as follows:

- Go to Workspace tree, and click on "HDL Code Generator1" (lower left of figure 6)
- Once the "HDL Code Generator Option s" window is opened, as shown in the center of figure 6, verify that this receiver model is the subnetwork you want to generate ("DDC_demod").
- Look at the "Target Configuration" to verify that it is "HDL Only".
- Then, click Generate to generate the VHDL code.
- Go to the Workspace directory named "FPGA" in Windows, and locate the HDL code files: **\FPGA\DDC_Demod_HDL\HDL*.vhd** as shown in the right hand of figure 6.

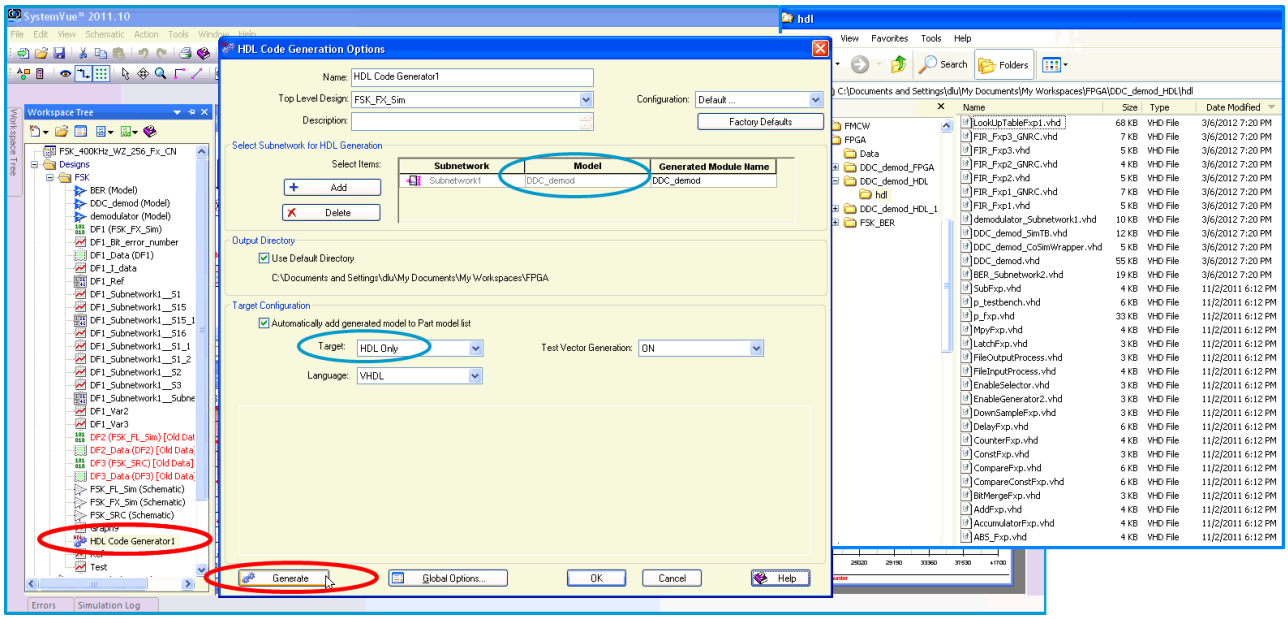

Figure 6. HDL code generation

Step 3

HDL Code Generation

Note that if you are planning to perform the HDL co-simulation using Aldec Riviera-PRO in step 4, make sure the executable path to Riviera-PRO is set up correctly on the "Code Generation" tab of the SystemVue Global Options, as shown in Figure 7.

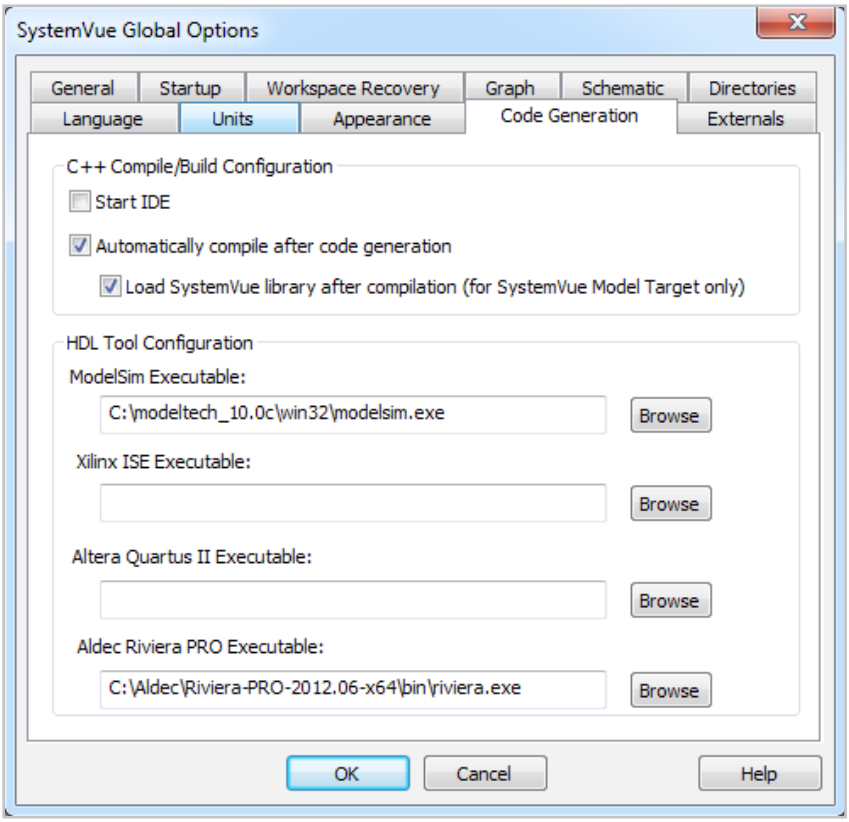

Figure 7. Configuring third-party software locations for coordinated use with SystemVue

Design teams doing high-performance communications and radar processing may already have their own library of HDL models which have been hand-optimized, validated, and are now trusted for re-use. It is not necessary to re-code these models. They can simply be imported into SystemVue and manipulated at a system level, alongside other blocks. If desired, these models can be set up to be "code re-generated," and become part of the same model-based design process.

Step 4

HDL Validation using Co-Simulation with Aldec Riviera-PRO

The fourth step in the FPGA rapid prototyping design process is to validate the generated VHDL or Verilog models to ensure they have the same behaviors and input/output test vectors as the previous floating- and fixed-point models. Subtle differences in truncation, overflow, latency, and timing can cause unexpected system-level results.

Fortunately, the generated HDL models can be easily verified directly inside SystemVue using co-simulation with a supported HDL simulator. During the process of code-generation, SystemVue automatically loads the new HDL model on the schematic as another model choice, under the list of polymorphic models (Figure 8). SystemVue's support for polymorphism makes scripted validations and regression suites quite easy, but this topic is beyond the scope of this application note. For more information on polymorphism, please refer to the SystemVue tutorial video: **www.youtube.com/watch?v=LEEibGvIDvc.**

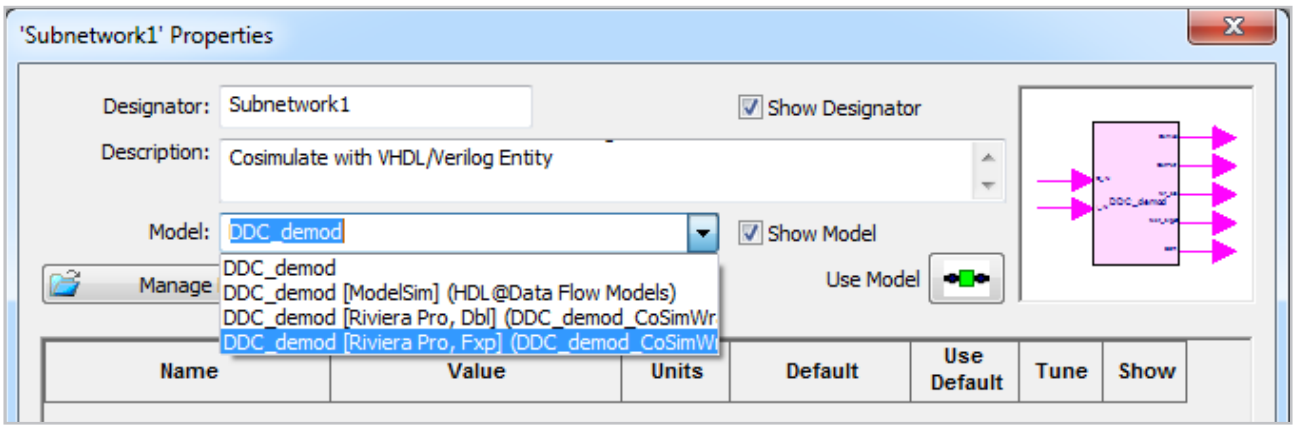

Figure 8. SystemVue's model polymorphism makes HDL co-simulation easy and enables a model-based design flow

SystemVue 2012 supports two HDL simulators: ModelSim from Mentor Graphics and Riviera-PRO from Aldec. Throughout the rest of this document, Riviera-PRO will be used to illustrate the flow.

As shown in Figure 8, there are two co-simulation models using Riviera-PRO:

- **DDC_Demod [Riviera Pro, Dbl]**, which is a co-simulation block that provides a floating-point interface. Here, the fixed-point conversion is done automatically inside the block.
- **DDC_Demod [Riviera Pro, Fxp]**, which is a co-simulation block with fixed-point interface.

Switch the model to DDC_Demod [Riviera Pro, Fxp] and then configure the Inputs and Outputs pages for word lengths and formats, as shown in Figure 9a and 9b.

Step 4

HDL Validation using Co-Simulation with Aldec Riviera-PRO

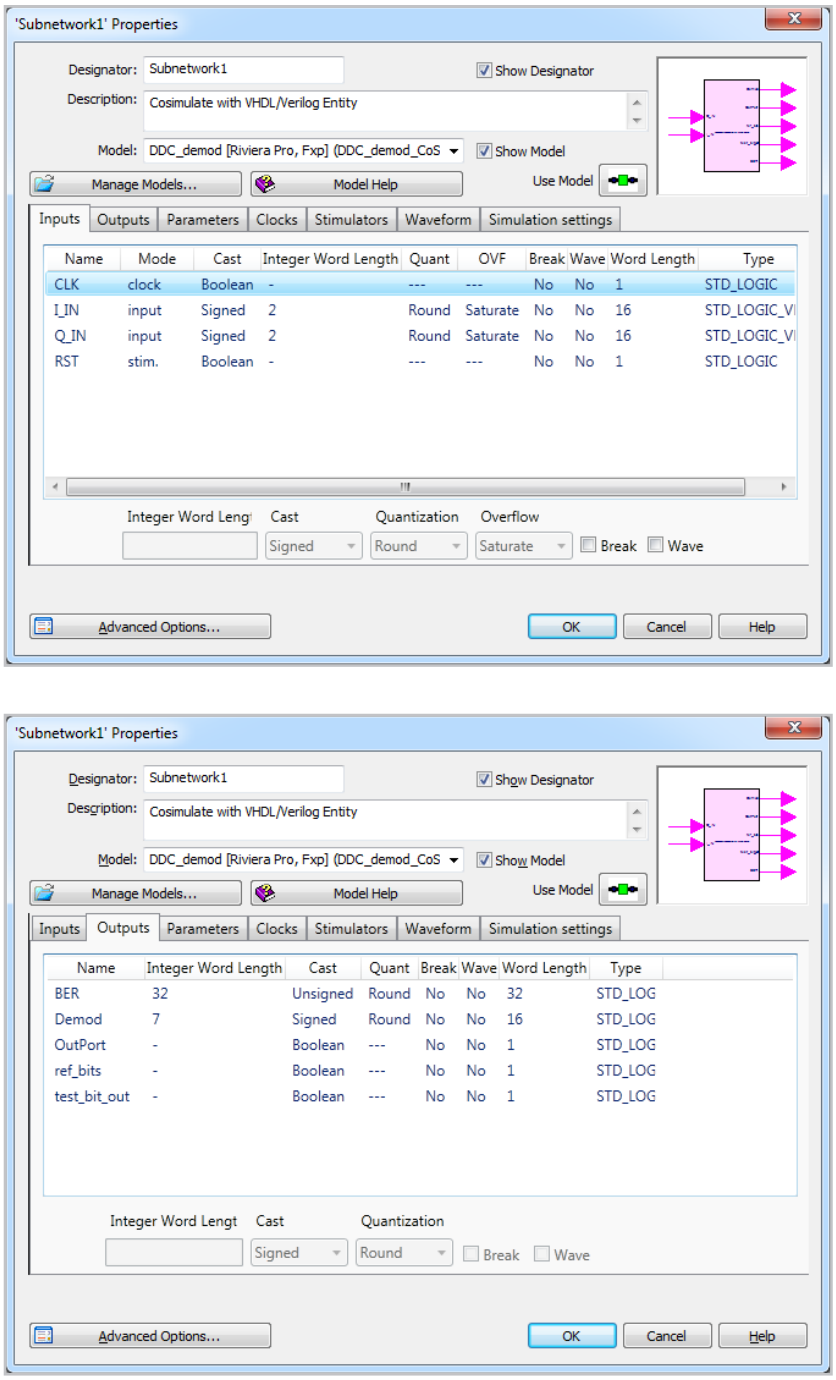

Figure 9. Inputs and Outputs configuration tabs for SystemVue co-simulation with Aldec Riviera-PRO

Step 4

HDL Validation using Co-Simulation with Aldec Riviera-PRO

Before running the simulation, make sure that the Simulation Settings are set to run in the automated "batch mode" as shown in Figure 10. Refer to SystemVue documentation for further details about other modes.

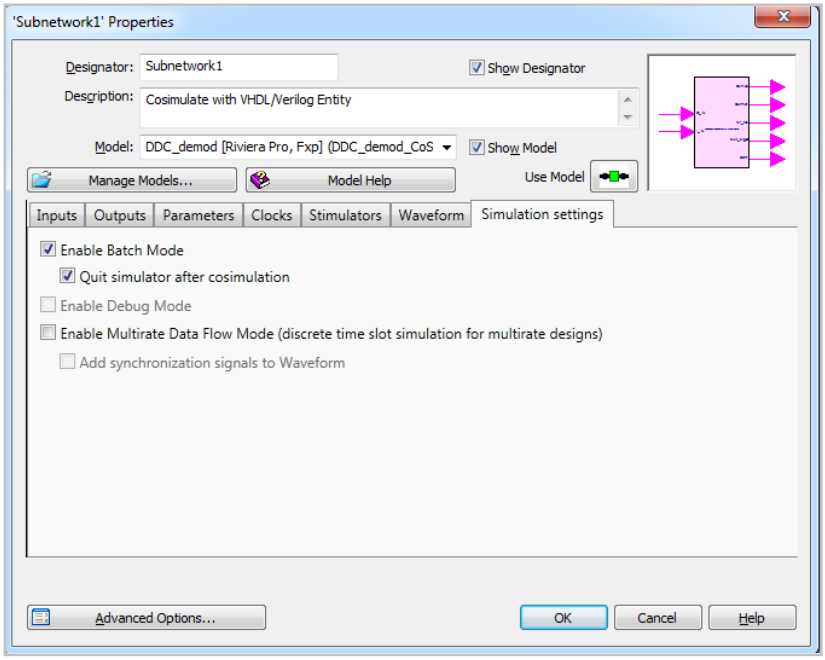

Figure 10. HDL co-simulation settings to enable "Batch Mode"

Run the simulation. A Microsoft® shell window will be invoked showing the detailed progress of the co-simulation process as shown in Figure 11.

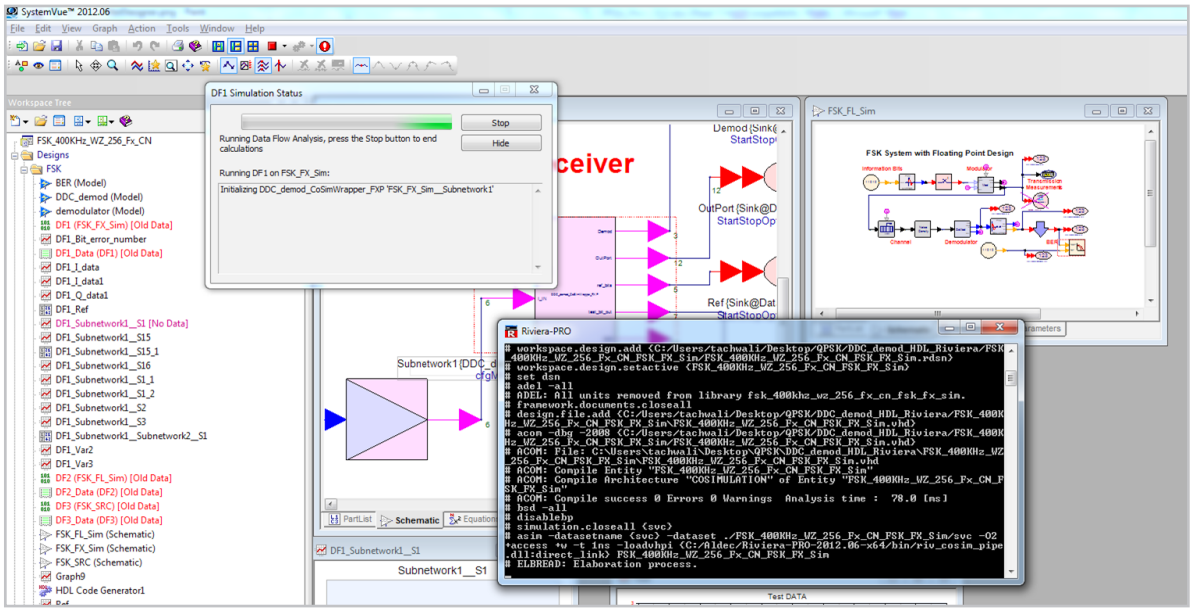

Figure 11. Riviera-PRO during the SystemVue co-simulation process

Step 4

HDL Validation using Co-Simulation with Aldec Riviera-PRO

Go to the generated dataset and plot the Ref and Test signals (reference bits and received bits) to verify the results, as shown in Figure 12. Note that the comparison must start after a certain period of the simulation to account for processing delays on the receiver side.

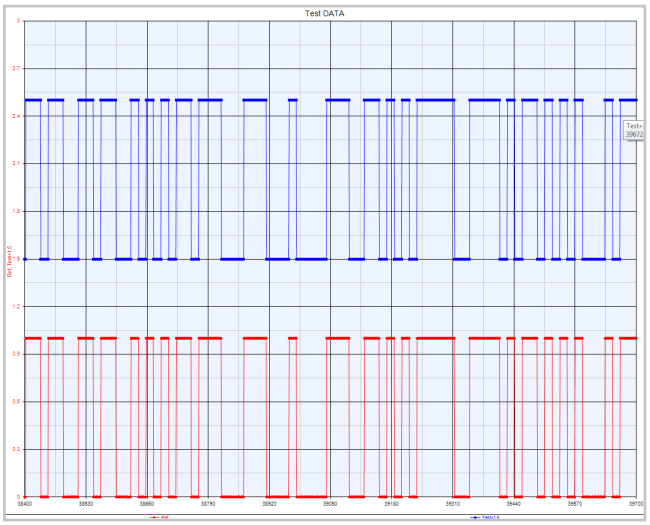

Figure 12. HDL co-simulation results comparing input and output bits for BER testing. The co-simulation results are returned to the SystemVue environment for post-processing and visualization.

Configurations for external applications

If you are running these HDL code generations and co-simulations for the first time, it may be necessary to visit the SystemVue "Global Options" screen (see figure 7). Here, users can configure the search paths to local software applications such as Aldec Riviera-PRO, the Xilinx and Altera synthesis tools, and other applications that integrate with SystemVue. In particular, if there is no declared Windows® path to Riviera-PRO, SystemVue will not be aware of that application, and HDL code generation will not automatically create and load models appropriate to that simulator. If no polymorphic co-simulation models are available after code-generation, check this settings tab.

HDL co-simulation using user-defined HDL libraries

When using SystemVue's built-in HDL code generation feature, the system automatically builds and loads the polymorphic co-simulation models. However, in the case of HDL models that originate outside SystemVue, HDL co-simulation requires a couple of extra steps, depending on the simulator. With Riviera-PRO, the following additional steps are needed:

- **In Riviera-PRO:**
  Compile the HDL files and generate an .XML library file for SystemVue.

- **In SystemVue:**
  Load the generated XML library from Aldec and instantiate the co-simulation models as parts on the schematic.

These manual steps are described in more detail in the SystemVue documentation (see section entitled "HDL Cosimulation with Riviera PRO"). After completion of the steps, the SystemVue schematic will have HDL co-simulation instances for Riviera-PRO or ModelSim that look like the screens in Figure 13a and 13b, each with parameters meaningful to that particular engine.

Step 4

HDL Validation using Co-Simulation with Aldec Riviera-PRO

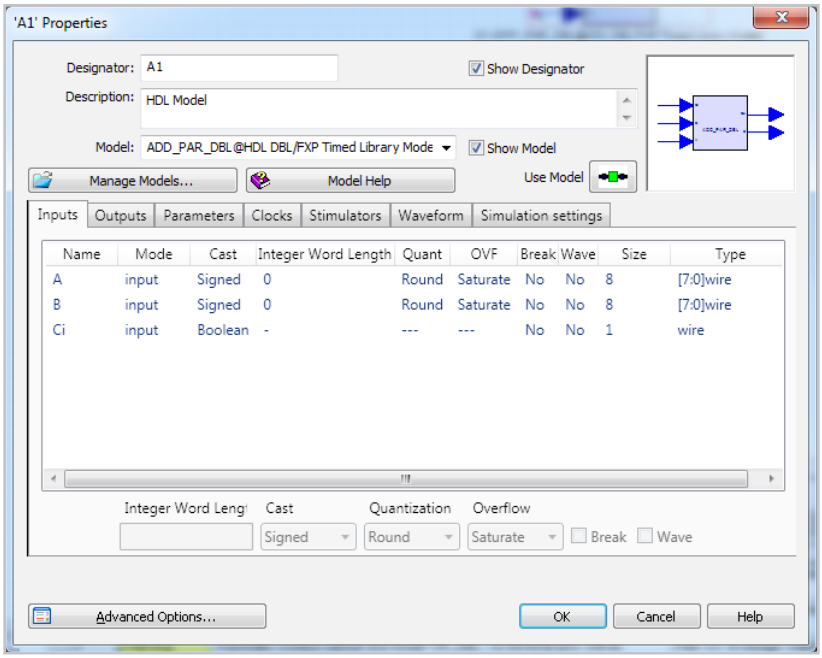

Figure 13a. Co-simulation block for HDL models imported by hand into Aldec Riviera-PRO, and then exported as models to SystemVue.

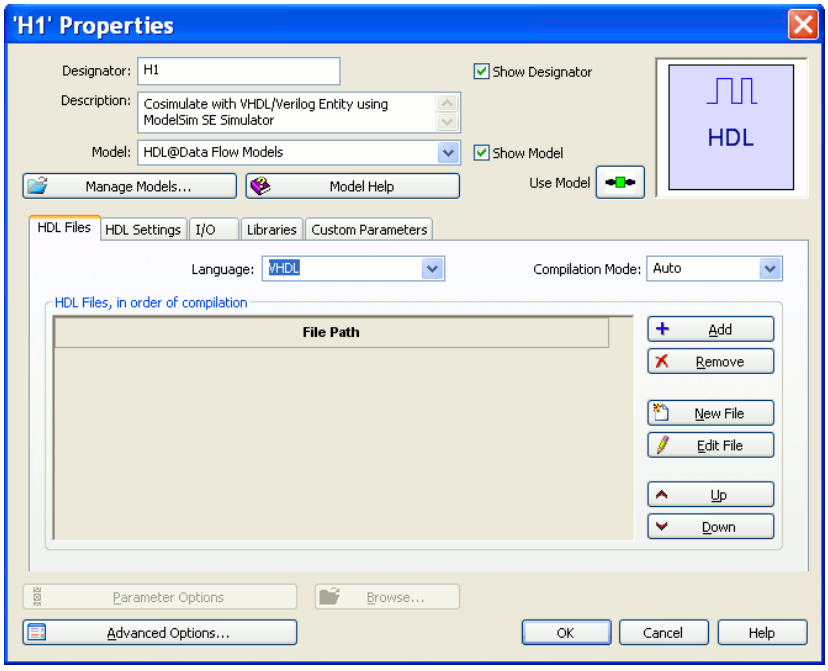

Figure 13b. Co-simulation block for models imported by hand into SystemVue, for co-simulation with Mentor ModelSim SE.

Step 5.

Generate the FPGA Programming File

The next step in the FPGA rapid prototyping process is to target the behavioral-level RTL to a specific FPGA hardware part within an FPGA family. This capability is not included with SystemVue, but low-cost tools for synthesis and place & route are available from FPGA vendors such as Xilinx and Altera. If the parameter choices for clock domains, I/O and other parameters are known, SystemVue's code generation screen makes it possible to continue code-generation all the way to the FPGA programming file, also known as a ".bit" file. Xilinx ISE for Virtex platforms and Altera Quartus II for Stratix platforms are both supported from the SystemVue user interface (UI).

Alternatively, many users already have experience with FPGA design tools, and are more comfortable running these tools standalone, to ensure greater control. In this case, SystemVue's generated HDL file tree and associated wrappers and testbenches have been imported into Xilinx ISE. The Xilinx tool is used to complete the synthesis to a particular hardware programming file (Figure 14).

The associated design steps are:

1. Open the "`Configxx.xise`" file in Xilinx ISE
2. Generate the program file (top.bit)
3. Load the program file into the FPGA

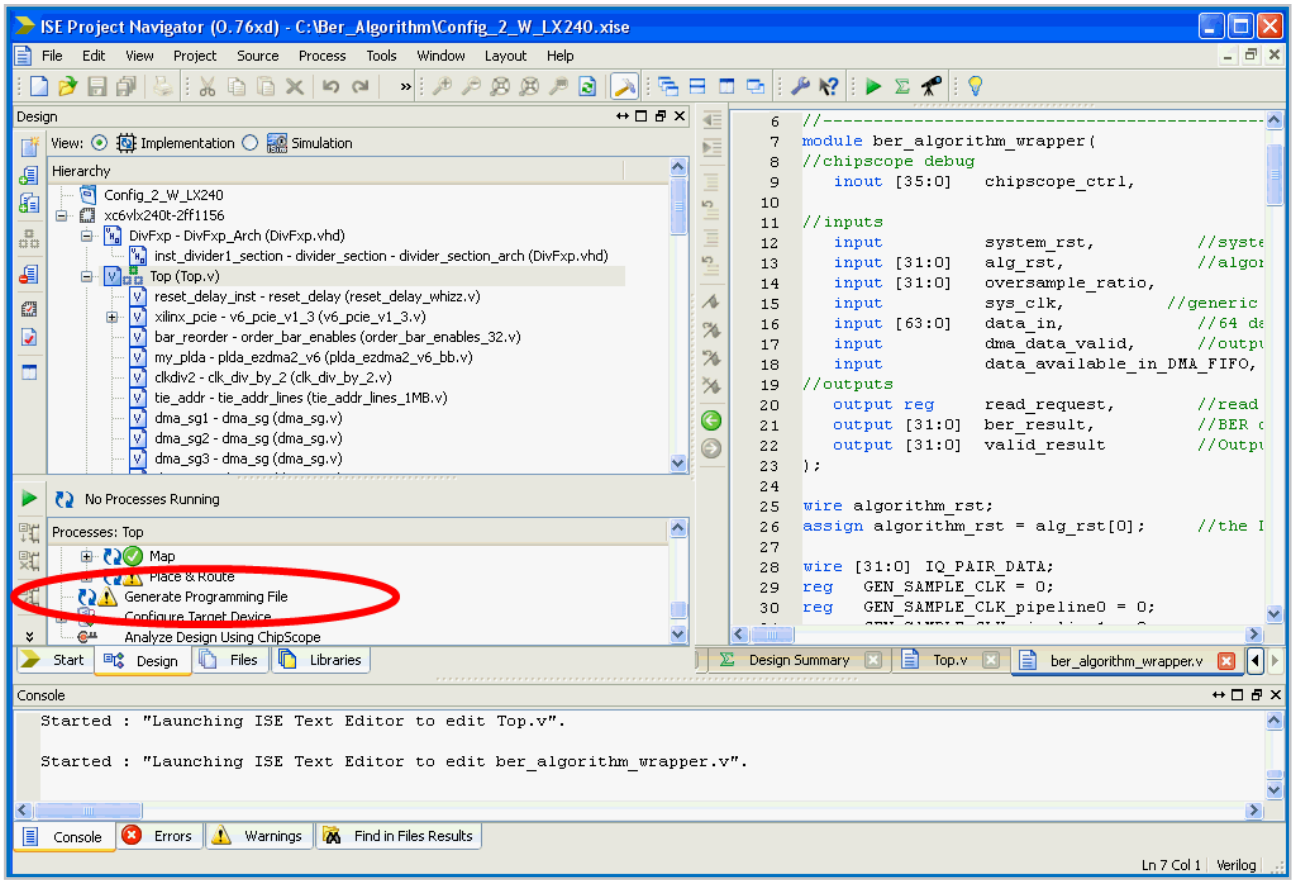

Figure 14. Using Xilinx ISE to finish generating the programming file

Step 6.

Load the .bit File into the FPGA

In Step 6, the FPGA .bit file is loaded onto the target, which is typically an FPGA development board using a USB, PCI Express® or LAN connection. For Xilinx Virtex boards, the iMPACT software within the Xilinx ISE suite can be used for this task.

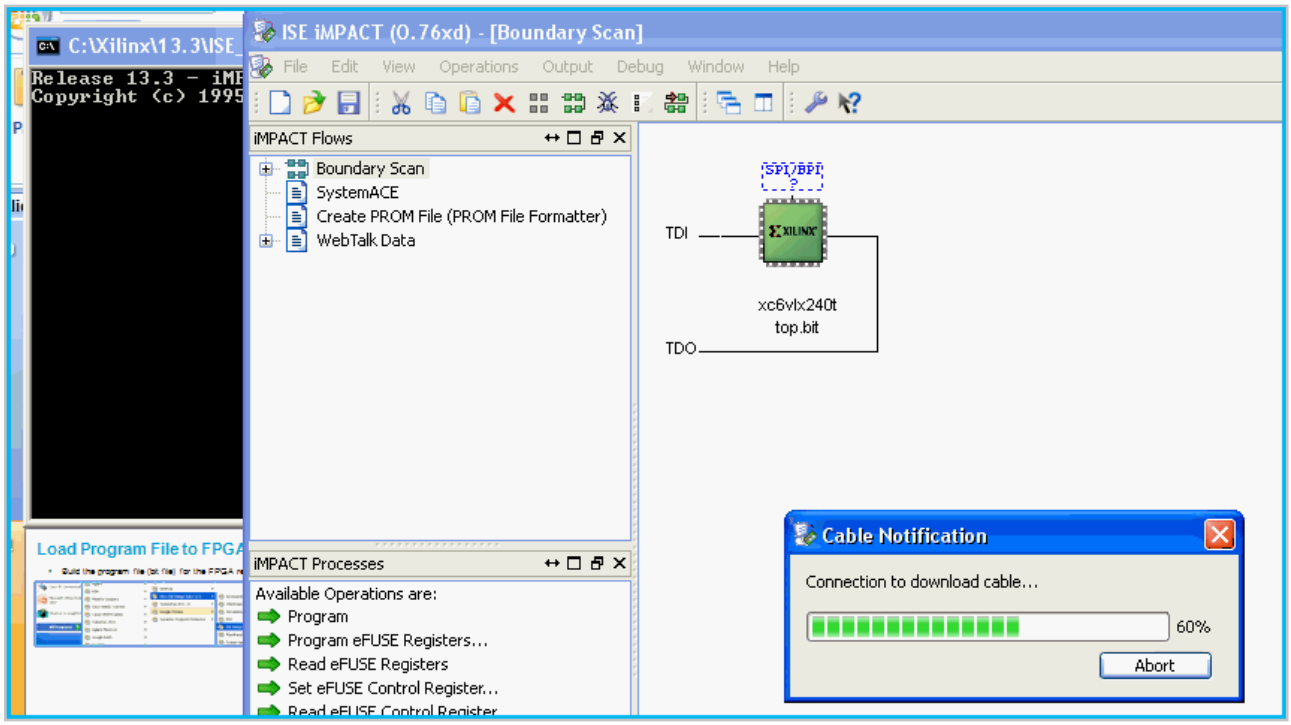

Figure 15. Using Xilinx iMPACT to load the .bit file into the FPGA

The processing sequence for loading the program into the FPGA board is as follows:

- Under Windows Start, run Xilinx ISE Design Suite 13.x/ISE Design Tools/ Tools/Impact.
- Double click on "Boundary Scan."
- On the opened iMPACT window, right click on the green box and load the program. The .bit file will be automatically loaded into the FPGA.

Step 7.

Generate Test Signals

Step 7 involves hardware verification of FPGA algorithms in the final step of the model-based design flow. Recall that as the design moved from floating to fixed point and to HDL, it was necessary to verify its integrity at each step. Now, with the final hardware implementation, verification is performed one more time. Depending on the rest of the system around it, the component can finally be tested in real-time at the true symbol rate of the signal.

Now that the FPGA has been programmed, it needs a hardware test vector to act as a stimulus and some kind of display to monitor the results. Additionally, for this application, a specific FSK signal must be generated. It is also necessary to provide a hardware clock signal of the same timebase.

For the purposes of this application note, two Agilent ESG or MXG signal generators are employed for hardware signal generation. However, a wide variety of signal sources may be suitable.

To create the test waveform, a SystemVue simulation is used. Returning to Step 1, the simulated transmitter signal can be automatically downloaded into an Agilent signal generator using SystemVue's "SignalDownloader" (Figure 16). This instrument connection is easy to configure and has been described in Agilent application note **http://cp.literature.agilent.com/litweb/pdf/5990-7757EN.pdf**. This particular signal is embedded with a known number of bit errors, in order to validate the final BER algorithm.

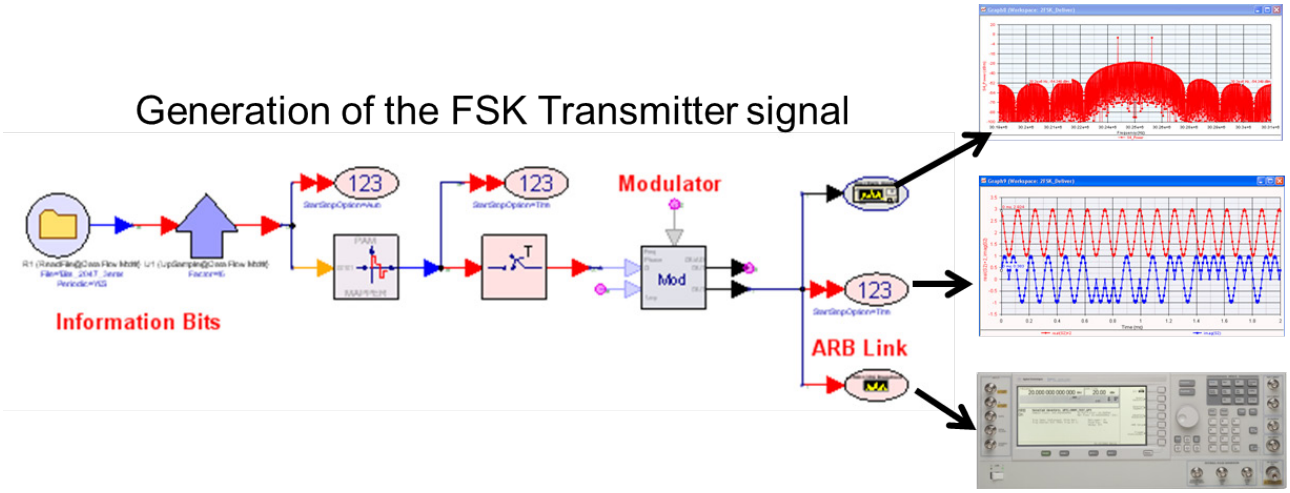

Figure 16. Simulating and downloading the FSK signal, in order to test the FPGA receiver

Step 8.

Testing the FSK Receiver using SystemVue

If you are following this application note using the SystemVue Workspace and wish to complete validation of the FSK BER tester using actual test equipment, here is a short list of resources you will find helpful.

Software and Hardware Requirements

Software needed to reproduce this work:

- SystemVue 2012.06 with the W1717 Hardware Design Kit, or a configuration that includes this feature
- The SystemVue .wsv workspace, downloadable by supported users from the Agilent EEsof KnowledgeCenter article (login required):

  http://edocs.soco.agilent.com/display/eesofkcsysvue/FPGA+flow
- Agilent 89600 VSA software, with option 105 (Agilent simulator connectivity) and option 300 (hardware connectivity)
- Agilent Connection Expert (Agilent I/O Libraries, version 15.0 or higher)
- Xilinx ISE, version 13.1 or higher

Helpful download links may be found at:

www.agilent.com/find/eesof-systemvue-download-apps

Hardware needed to reproduce this work:

- FPGA development board
- Sources: Agilent signal generators, such the ESG/MXG families, to provide:
  - 250-kHz FSK input signal into the FPGA
  - FPGA clock signal
- Analyzer
  Agilent Infiniium Oscilloscope and/or Agilent 16900 Logic Analyzer

Step 8.

Testing the FSK Receiver using SystemVue

At this point, the designed FPGA receiver has been programmed. As shown in Figure 17, the test signal in the FSK format has been generated, downloaded to an Agilent ESG E4438C signal generator and connected to the FPGA input port. The clock data generated by a ESG E4432D is connected to the FPGA clock input. A *Chipscope* performance cable is connected to the FPGA and also connected to SystemVue through a USB cable.

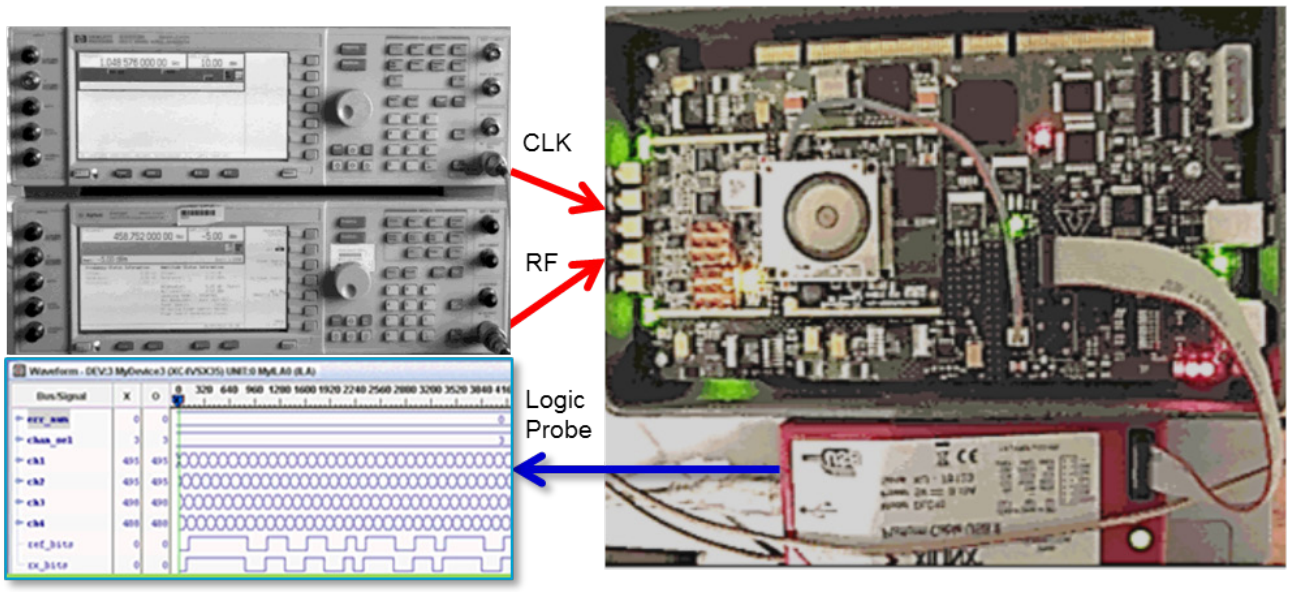

Figure 17. Testing the FSK FPGA hardware receiver, using a Xilinx probe to feed a

Step 8.

Testing the FSK Receiver using SystemVue

Now, the FPGA performance can be tested. First, verify the input FSK signal by connecting the signal generator's RF output to a RF signal analyzer or Infiniium oscilloscope. The FSK signal can be observed using the 89600 VSA software, which runs on either instrument. Figure 18 shows the instrument connections and the observed transmit waveform, constellation, spectrum, and EVM. Using *Chipscope*, the test and reference bits are aligned, and the expected number of errors are observed, as shown in Figure 19.

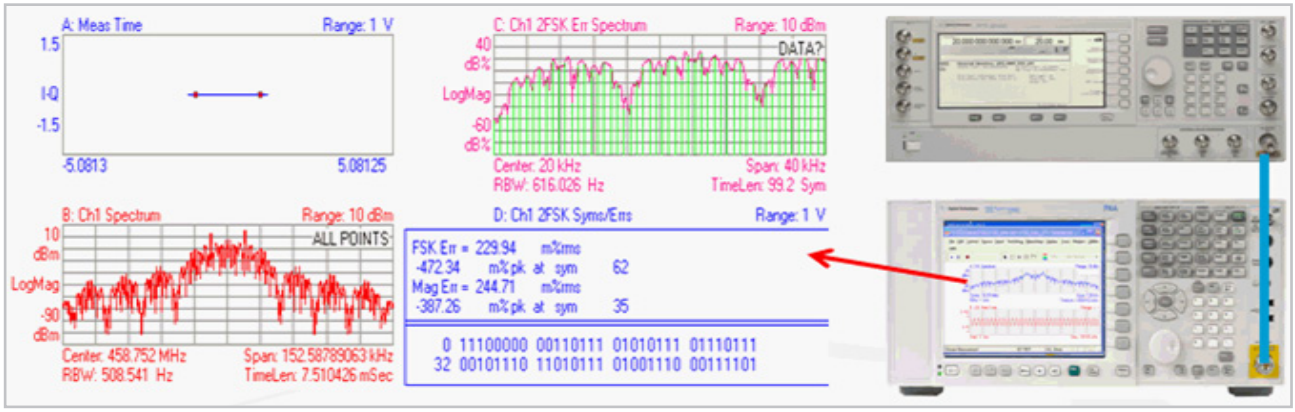

Figure 18. FSK signal constellation and spectrum

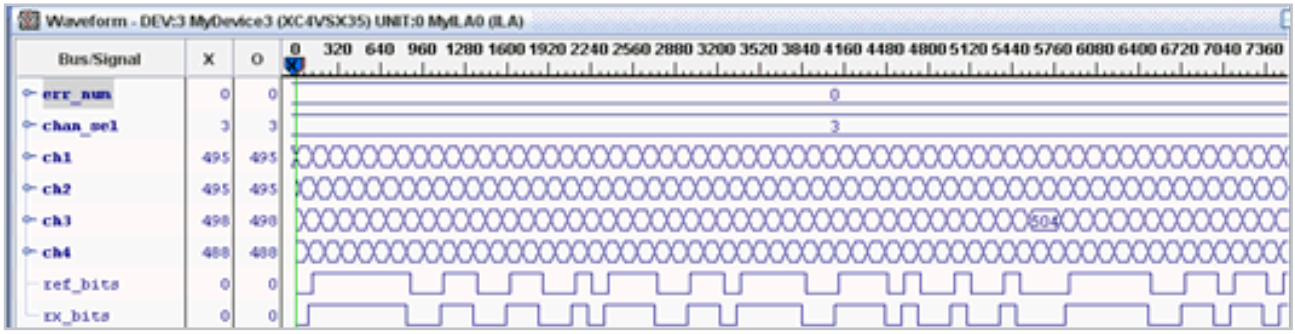

Figure 19. FSK receiver results, after FPGA processing. Above, `err_num` = number of accumulated bit-errors, while `ref_bits` and `rx_bits` are the received and recovered bitstreams, respectively.

Conclusion

As seen in this application note, SystemVue can be used to design and implement software-defined radio (SDR) communication systems using a model-based design flow at progressively detailed levels of implementation. Since SystemVue can also control test equipment, the design capability can also be applied to custom test personalities, such as the simple BER tester shown here.

Please visit the SystemVue resources listed below to see how SystemVue can also integrate RF design, math language and C++ modeling, and wireless standards libraries into a cross-domain design cockpit for communications system design.

For more information about SystemVue, please visit us on the web:

Product information
www.agilent.com/find/eesof-systemvue

Product Configurations
www.agilent.com/find/eesof-systemvue-configs

Request a 30-day Evaluation
www.agilent.com/find/eesof-systemvue-evaluation

Downloads
www.agilent.com/find/eesof-systemvue-latest-downloads

Helpful Videos
www.agilent.com/find/eesof-systemvue-videos

Technical Support Forum
www.agilent.com/find/eesof-systemvue-forum

www.agilent.com/find/eesof-systemvue

Agilent Email Updates

www.agilent.com/find/emailupdates
Get the latest information on the products and applications you select.

www.agilent.com

For more information on Agilent Technologies' products, applications or services, please contact your local Agilent office. The complete list is available at:

www.agilent.com/find/contactus

Americas

Canada	(877) 894 4414
Brazil	(11) 4197 3600
Mexico	01800 5064 800
United States	(800) 829 4444

Asia Pacific

Australia	1 800 629 485
China	800 810 0189
Hong Kong	800 938 693
India	1 800 112 929
Japan	0120 (421) 345
Korea	080 769 0800
Malaysia	1 800 888 848
Singapore	1 800 375 8100
Taiwan	0800 047 866
Other AP Countries	(65) 375 8100

Europe & Middle East

Belgium	32 (0) 2 404 93 40
Denmark	45 45 80 12 15
Finland	358 (0) 10 855 2100
France	0825 010 700*
	*0.125 €/minute
Germany	49 (0) 7031 464 6333
Ireland	1890 924 204
Israel	972-3-9288-504/544
Italy	39 02 92 60 8484
Netherlands	31 (0) 20 547 2111
Spain	34 (91) 631 3300
Sweden	0200-88 22 55
United Kingdom	44 (0) 118 927 6201

For other unlisted countries:
www.agilent.com/find/contactus

Revised: January 6, 2012

Published in USA, August 30, 2012
5991-1113EN

Agilent Technologies