RAPID SDR WAVEFORM DEVELOPMENT IN FPGAS USING DSP BUILDER

Steven W. Cox
General Dynamics C4 Systems
8201 E. McDowell Road, MDR3125
Scottsdale, Arizona 85257
(480) 441-1736
steve.cox@gdds.com

Joel A. Seely
Altera Corporation
101 Innovation Dr
San Jose, CA 95134
(408) 544-8122
jseely@altera.com

ABSTRACT

Software defined radio technology is achieving rapidly growing acceptance as a military communications platform because of its security advantages and its ability to be reconfigured to meet specific mission parameters. Programmable logic offers the performance, flexibility, and cost-effectiveness required for SDR systems. One of the challenges in implementing SDR designs is the long and growing list of waveforms that must be implemented in programmable logic as well as the need to be able to rapidly develop new waveforms.

Traditional waveform implementations in FPGA require separate design efforts at both the system and component level. This is repetitious, time-consuming and costly. What is needed is a development flow that allows the designer to more efficiently implement these waveforms at the component level. This paper will describe an optimal FPGA design flow for waveform implementations using the DSP Builder tool. It will outline the benefits to be derived from using FPGA building blocks and a system-level tool in waveform development for Software Defined Radios (SDRs).

1. INTRODUCTION

The use of FPGAs in software defined radios is becoming ubiquitous as more systems require interoperability across multiple standards. Part of this trend is due to the heavy reliance on configurable hardware, which is used for increasing levels of processing when implementing the waveforms [1]. To quickly develop these waveforms, some of which are extremely complex, tools with a higher level of abstraction are required. Altera's DSP Builder tool and associated IP blocks provides an integrated environment that can be used to develop hardware implementations of waveforms. This method streamlines the FPGA design flow using: Mathwork's Simulink capabilities, fixed point blockset with Altera FPGA objects, and interfaces to third party tools to generate a synthesizable FPGA HDL. This tool allows multi-disciplined users to work at higher levels of abstraction in a common workspace. All aspects of waveform development: design, simulation and verification can be addressed at the Simulink level prior to hardware implementation.

2. TRADITIONAL FPGA WAVEFORM DESIGN METHOD

SDR waveform design has typically been extremely inefficient. In the past, system-level specifications and simulations were "thrown over the wall" to the hardware designers who then started coding in their favorite Hardware Definition Language (HDL). There were, of course, some challenges with this approach. First, the system designer had no insight into the implementation details of the FPGA and, therefore, could not best optimize the system design without lengthy communications with the engineers implementing the design. Secondly, the designer needed to be an expert in HDL—not the sort of expertise an engineer was likely to pick up overnight. Third, this approach involves manual code generation, which is time-consuming and tedious, as well as likely to require extensive debugging—all of which increases development time and cost. This approach also contains some inherent tendencies towards inefficiency, since the system must be created twice, first on the system-level tool and then on the implementation tool—once again increasing the time and cost of system development. Figure 1 provides an example flowchart for this traditional waveform development flow.

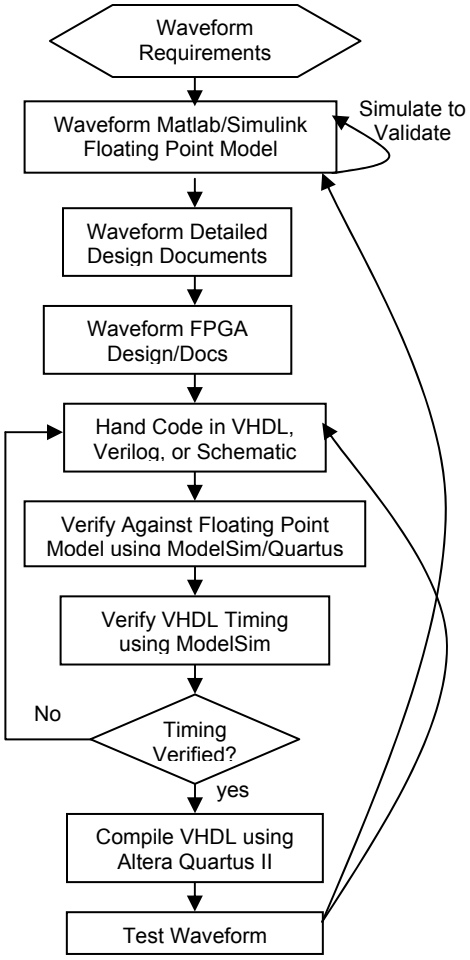

Figure 1: Traditional Waveform Design Flow

3. DSP BUILDER: SIMPLIFYING WAVEFORM DESIGN

As FPGAs increase in complexity, it is necessary to have system-level tools that can aid the designer in simplifying the design methodology. Tools such as Altera's DSP Builder have been developed to address the issues found when performing complex system development such as waveform design. With this tool, a new design flow consists of 5 segments: defining architecture, implementing/designing modules, integration of modules, and translating the design to physical FPGA and verifying the part in the lab. See Figure 2:

4. ARCHITECTURE DEFINITION

The typical process for FPGA waveform implementation is to start with an existing model, and then "porting" it to an FPGA. Floating point Simulink models of standard waveforms such as FM, SSW, or MIL-STD 110A are examples of waveforms that are readily available These floating-point models can be used as a guideline and comparison tool, as well as for initial sizing and architecture mapping estimates for the FPGA implementation. A functional block-diagram of a typical waveform is shown in Figure 3.

5. IMPLEMENTATION/SIMULATION:

DSP Builder combines the algorithm development, simulation, and verification capabilities of The MathWorks MATLAB and Simulink system-level design tools with VHDL synthesis, simulation, and Altera development tools. The waveform design entry uses Altera DSP Builder blocksets and Simulink toolbox blocksets in the Simulink schematic capture environment.

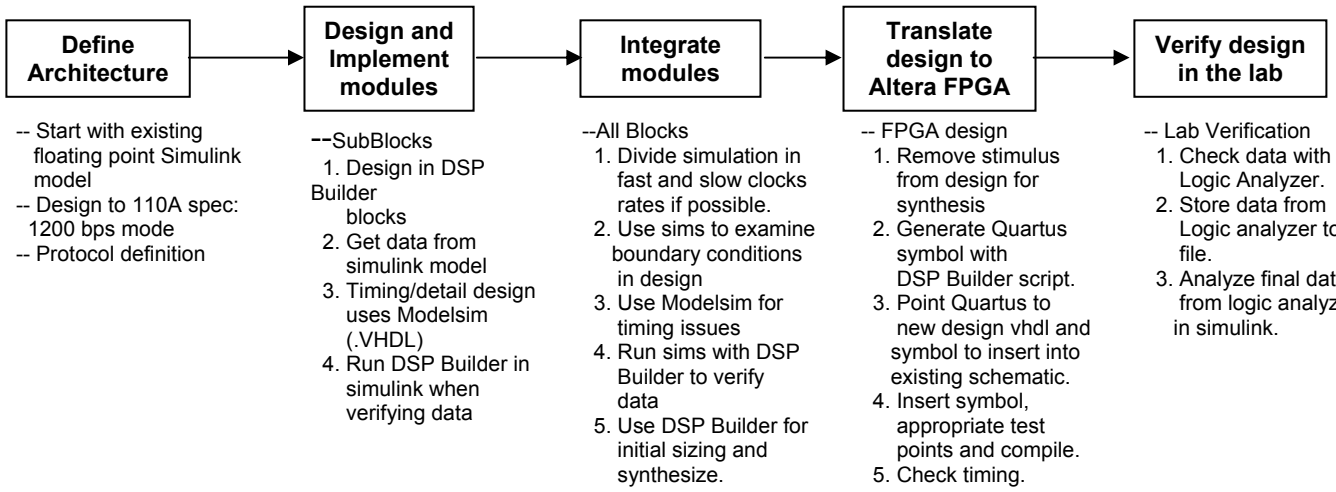

Figure 2: Altera DSP Builder Design Flow

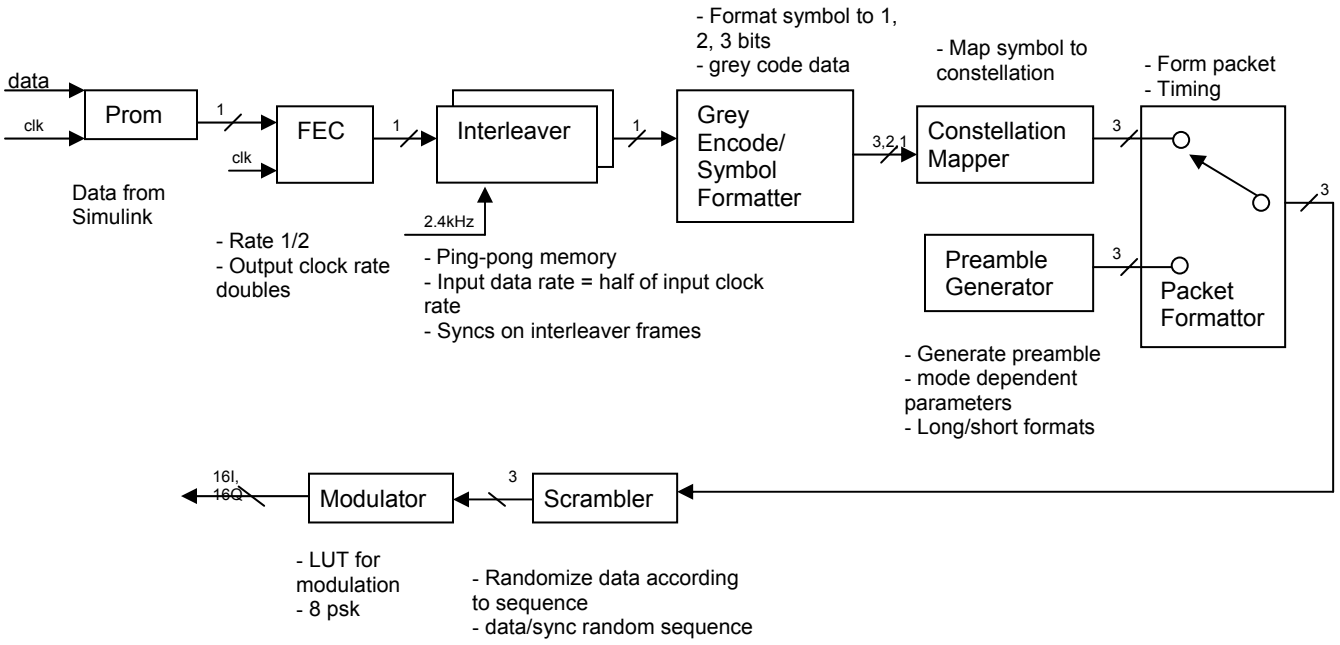

Figure 3: Transmit 110A Top Level Block Diagram

More complex functions such as FIR filters, NCO, FFT, and others, can be integrated into DSP Builder using MegaCore functions that plug into the tool. An example of the FIR Megacore configuration page is shown in Figure 4:. These wizard driven IP blocks are valuable because you can alter parameters and watch the simulated output before finally adding the HDL to the system.

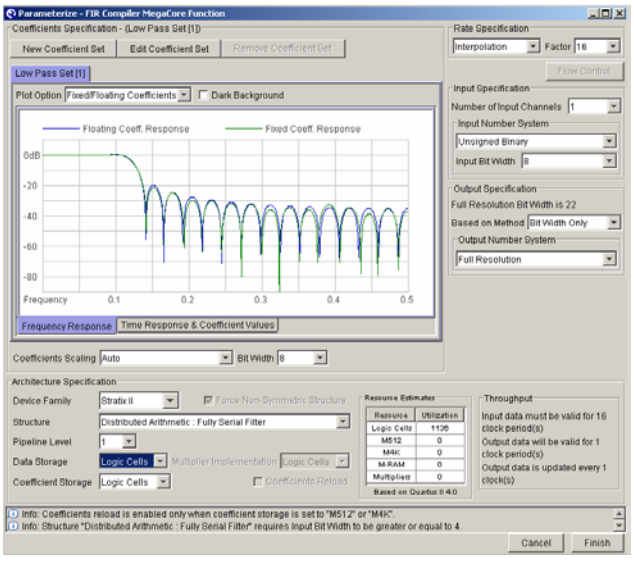

Figure 4: FIR Compiler Megacore Configuration Wizard

The DSP Builder tool provides Simulink graphics conversion to VHDL for third party VHDL simulators. Modelsim, an efficient and fast event driven simulator, can be used for timing analysis on the subblocks. The subblock timing design flow is shown in Figure 5: As the designer verifies the timing and control in Modelsim, he updates the design in the Simulink environment. With the push of a button, conversion of the Simulink graphics to

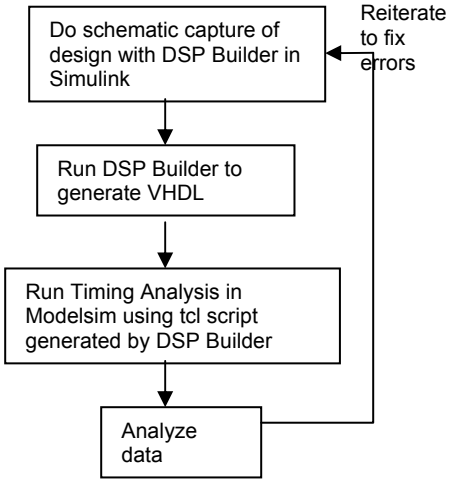

Figure 5: DSP Builder Timing Design Flow

VHDL, stimulus for the subblock design, scripts to load and compile the updated design into Modelsim are all created. This automation streamlines the iterative process of simulating a design to assure a properly designed block and removes all the onerous, but necessary task of organizing scripts for translation to other tools.

The next step is data verification. The initial "known good" floating point models are compared with the DSP Builder models and verified. The DSP Builder model and floating-point models are run separately in Simulink and their data sets of interest are stored in the Simulink environment workspace. Commands in Simulink manipulate the DSP Builder fixed-point data by sub-setting the data sets to compare to the floating-point model. To correct errors, the designer updates the models in Simulink and reruns the simulations to verify. The data verification flow is shown in Figure 6:.

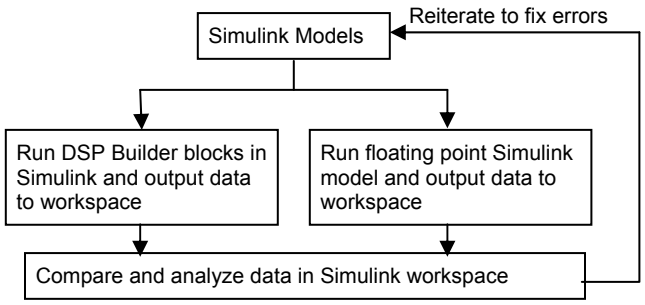

Figure 6: DSP Builder Data Verification Flow

6. INTEGRATION/SIMULATION:

DSP Builder models are built in C-Code, which simulates than HDL interpretive simulators. For this reason, data validation was accomplished in the Simulink environment. After design validation, an initial sizing and synthesis of the DSP Builder design is performed in Simulink. This provides an early heads up to determine if further work is needed on the design to compensate for sizing and timing constraints. This methodology provides a good technique to fix problems in the early stages of the design cycle prior to the time consuming and costly fixes at the end of the design cycle.

To help with analyzing data during integration, tricks such as separating slow and fast clock dependencies is used. This helps alleviate extensive simulation times for signals that are artificially slow. In the 110A example, the slow clock dependences run from the beginning of the waveform chain to the input of the scrambler. The fast clock dependencies run from the scrambler to the output of the modulator. Due to the latency in the interleaver, the data at the output does not become valid until one frame has been written in. Only valid data at the output of the data formatter was captured to the workspace and the fast dependencies were run as a separate simulation with only the valid data output from the formatter. This integration/simulation methodology significantly reduces simulation time for the high-speed portion of the circuit and allows efficient design validation.

7. SYNTHESIS:

Input pins must be added to the design in place of the Simulink stimulis model before synthesis. In the Simulink model, input stimulus was stored in a PROM and DSP Builder automatically generated the PROM file for synthesis. The DSP Builder tool also generates a Quartus script to load the design into the tool to create a symbol. The user must anticipate the test points needed to observe the entire new design. These test points are included in the symbol block so they can be connected in the schematic. If additional test points are needed after the compile, the user must update the Simulink DSP Builder model. In order to embed the new design in an existing schematic, pointers to the new VHDL files and its symbol were included in the Quartus project. Then the project was compiled and timing results were analyzed. Embedding the new DSP Builder design in the existing Schematic Capture design is easy and straightforward.

8. VERIFICATION IN THE LAB

Once the design raw binary file is created it can be loaded onto a development board such as the DSP Development Board (see Figure 7:) or your own board. Each sub-block can then be checked with a logic analyzer to make sure the output from the hardware matches the simulation results. Finally a snapshot of the output of the waveform in a familiar constellation plot (see Figure 8:) can be taken.

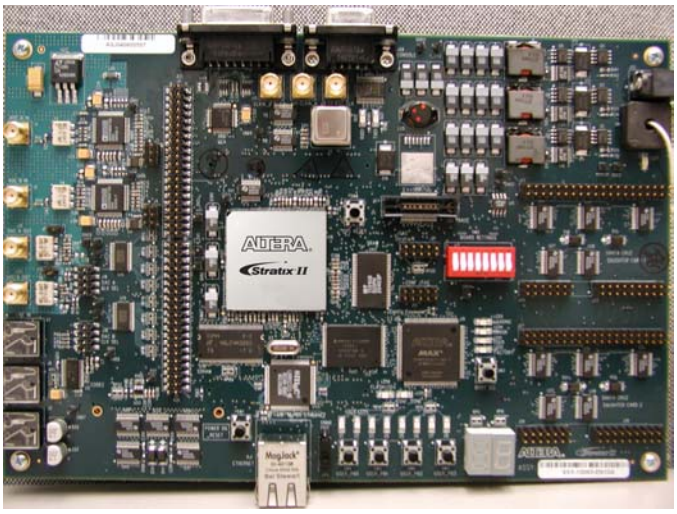

Figure 7: Example Stratix II DSP Development Board

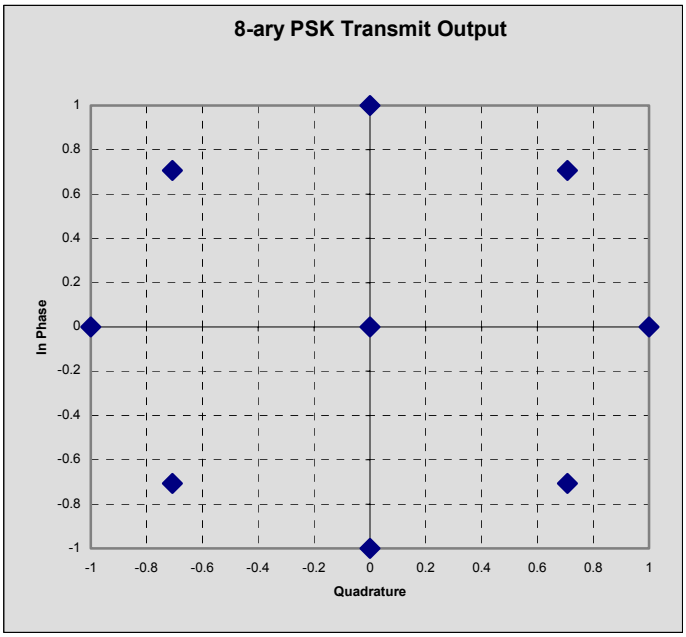

Figure 8: Constellation Plot For 110A Transmit Output

The value of this method is that the logic analyzer captured final data (I and Q) can be verified against the data in the floating point Simulink environment, which typically takes only a few days. The upfront simulation and verification of each of the sub-blocks makes the final checkout process run smoothly and quickly.

9. CONCLUSIONS:

Figure 9: shows the expected review points for the different stages of a waveform design and an estimate of the time to complete each of these by the traditional design methodology and using DSP Builder for the 110A test case.

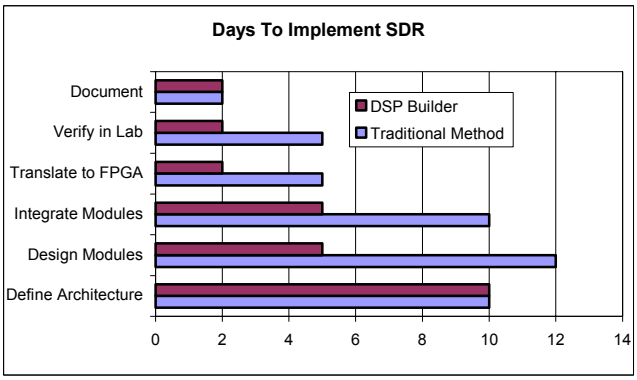

Figure 9: Comparison of Time to Complete Portions of 110A SDR Using Traditional Method and DSP Builder

The Simulink/DSP Builder flow allows the designer to allocate time in an appropriate manner for developing waveforms. With the traditional method, the FPGA designer spends an enormous amount of time in the lab (30 % and up) troubleshooting FPGA design/system issues. The DSP Builder/Simulink/Modelsim simulation flow allows the designer to rapidly identify problems and troubleshoot by adding appropriate taps and re-simulating the model. Taking simulations to an integration level significantly reduces risks, time and resources in the lab.

The following points should be considered for future tool development: 1. Plotting with the DSP Builder/Simulink environment doesn't always result in cycle accurate outputs. This makes it a necessity to validate timing with 3rd party tools (Mentor Graphics ModelSim) 2. All blocks should be parameterized when picking bus widths. 3. Signal Compiler window should import timing constraints to Quartus. 4. In order to make the test bench self checking, it must have some sort of assertion checking. Several of these enhancements are scheduled to be added in future releases of DSP Builder.

The DSP Builder tool allows the hardware to be abstracted to a higher level so the FPGA and system waveform developers can operate in a common environment or be one-in-the-same person. The tools have a very short learning curve if the user is familiar with Matlab/Simulink and hardware design. As the tool matures, some enhancements may be added to help speed up waveform development. Some suggested enhancements include expanding the parameterization for building buses, seamlessly importing timing constraints to the synthesis and place-and-route tool (Quartus II), and automatically creating assertions when generating test benches to make them self-checking.

The flow described has major advantages of streamlining validation, lab checkout and providing a common environment within which multi-disciplined personnel can work and communicate. With its existing and upcoming features, DSP Builder is a powerful tool for rapidly developing SDR Waveforms on FPGAs.

10. REFERENCES

[1] S. W. Cox, “FPGA Based Waveform Design Techniques for Software Defined Radios”, SDR Forum Technical Conference, HW-1-005, November, 2003.

[2] DSP Builder User Guide, Altera
http://www.altera.com/literature/ug/ug_dsp_builder.pdf