CASE-STUDY OF A XILINX SYSTEM GENERATOR DESIGN FLOW FOR RAPID DEVELOPMENT OF SDR WAVEFORMS

David Haessig (BAE SYSTEMS, CNIR, Wayne, NJ, USA, david.haessig@baesystems.com);
Jim Hwang, Sean Gallagher, Manuel Uhm (Xilinx, Inc., San Jose, CA,
jim.hwang@xilinx.com, sean.gallagher@xilinx.com, manuel.uhm@xilinx.com);

ABSTRACT

This paper describes a case study examining two distinct design processes for implementation of FPGA-based software defined radio subsystems. We compare a traditional RTL design approach with a model-based design flow involving automatic code generation using System Generator. Both design processes were applied to the development of a common SATCOM waveform: Mil-Std-188-165a. Results indicate a 10:1 improvement in development efficiency using System Generator, based on quantitative comparison of the time consumed in developing system simulations, algorithm documentation, code design and debugging, hardware implementation, and algorithm verification. The time savings associated with performance analysis using hardware co-simulation is also assessed.

1. INTRODUCTION

FPGAs have become widely used in the design of physical layer and baseband processing for software-defined radios. However, prevailing programming models, derived from design flows used to design ASICs, have traditionally limited accessibility of FPGA-based signal processing subsystems to designers with a background in chip design. This situation has been changing over the last five years with the emergence of a new class of FPGA programming flows based on high level modeling in MATLAB and Simulink. The Xilinx *System Generator for DSP*^{TM}, the first Simulink-based tool for FPGA design introduced in 2000, and subsequent tools share the common purpose of bringing FPGA technology to a wider audience, while increasing designer productivity, especially in the area of modem development for software-defined radios (SDR). For the radio designer who lacks expertise in traditional ASIC hardware design flows, these tools provide a means for architecting, synthesizing, and validating high performance systems employing FPGAs. For the more traditional FPGA designer, these tools can provide significant productivity improvements over traditional methods, e.g., by enabling better exploration of the architecture space and creation of more realistic and robust test harnesses. Whilst productivity improvements have been reported anecdotally and in internal corporate documents, there have been few published results that attempt to quantify productivity improvements derived from new design flows [2].

In this paper we present a case study comparing a traditional VHDL-based design flow to a System Generator based FPGA design process, using a subset of the military SATCOM waveform Mil-Std-188-165a as test application. We compare the two design flows in terms of waveform specification and documentation, design and performance, and debugging effort (in both the simulation environment and hardware). In addition, we identify ways in which traditional FPGA tools (e.g., logic synthesis) continue to play fundamental roles. The notion of SCA Rapid Development is described, and we indicate areas in which FPGA tools must improve or expand in order to fully service the SDR community, especially regarding the hardware/software interface and the control plane (e.g., SCA compliance).

2. FPGA DESIGN FLOWS

Traditional FPGA design flows have historically mirrored processes originally developed for building application-specific integrated circuits (ASICs). An untimed system model is usually created in an imperative language like C or MATLAB. This design phase represents the primary opportunity for algorithm exploration, and typically provides test vectors for validating the implementation. The initial implementation is typically described in a hardware description language (HDL) like VHDL or Verilog at a register transfer level (RTL) that allows a behavioral (as opposed to purely structural) logic description. The RTL description typically involves instantiation of reusable components called intellectual property (IP) blocks (e.g., Viterbi decoder, FFT), often provided by the FPGA vendor, to ensure efficient implementation of complex functions on a particular device fabric. Functional conformance testing to the original system model is done using HDL simulation by creating an HDL test harness that imports test vectors provided by the system model. This loose coupling between system model and implementation makes debugging difficult and time-consuming. For example, test vectors provide only an input/output relation, so it is often necessary to rework the system model (often written by an entirely different design team) in order to extract internal state or signals for debugging.

In contrast, design flows based on System Generator and similar tools derive a hardware realization directly from the system model via automatic code generation [1][3]. Sometimes referred to as *model-based design* [4], such high level design approaches aim to increased productivity (from higher levels of abstraction) and reliability (from automatic code generation and more robust test methodologies). System Generator provides a more attractive programming model than HDLs for a signal processing engineer, as well as a greater ability to explore architecture and debug complex algorithms realized in hardware. Embedded in Simulink [5], System Generator provides abstractions and tool interfaces expressly designed to facilitate hardware design [7]. System Generator provides libraries of Simulink blocks (the Xilinx Block Set) with bit and cycle true simulation for a wide range of functions ranging from communication algorithms (e.g., Viterbi decoder, interleaver), DSP algorithms (e.g., FFT, FIR filter), down to lower level building blocks for memories, arithmetic structures, and logic. Since the system model is captured in Simulink, the debugging capabilities of that tool can be brought to bear, including data visualization and test harness creation.

System Generator also extends Simulink through standard APIs to interface directly to HDL simulation tools (enabling import of HDL modules) and directly to hardware platforms (hardware co-simulation). In System Generator, hardware co-simulation entails automatic generation of an FPGA bitstream from Simulink, as well as its incorporation back into Simulink itself. This allows the user to exploit FPGAs to significantly accelerate simulation, while also providing the ability to validate a design working in hardware, all without necessarily having to invoke a traditional FPGA tool explicitly.

3. CASE STUDY RESULTS

3.1 Problem Description and Motivation

BAE Systems CNIR, a developer of advanced military communication radio systems, continually evaluates tools and design processes that offer potential for improved efficiency. A project to implement a SATCOM waveform on the BAE C4ISR radio offered an ideal opportunity to evaluate a new model-based design approach using System Generator. This design flow made extensive use of the system model including design capture, system simulation, and auto-generation of RTL VHDL, providing opportunity for significant time and cost savings. Two parallel development efforts were performed. The waveform was implemented to the same set of requirements using both a traditional and a model-based development flow. Each design was carried through to implementation on hardware. Effort for each design was measured in man-hours, and each developer noted activities that were particularly easy or troublesome. Upon completion, the developers compared notes and noted advantages or disadvantages to their

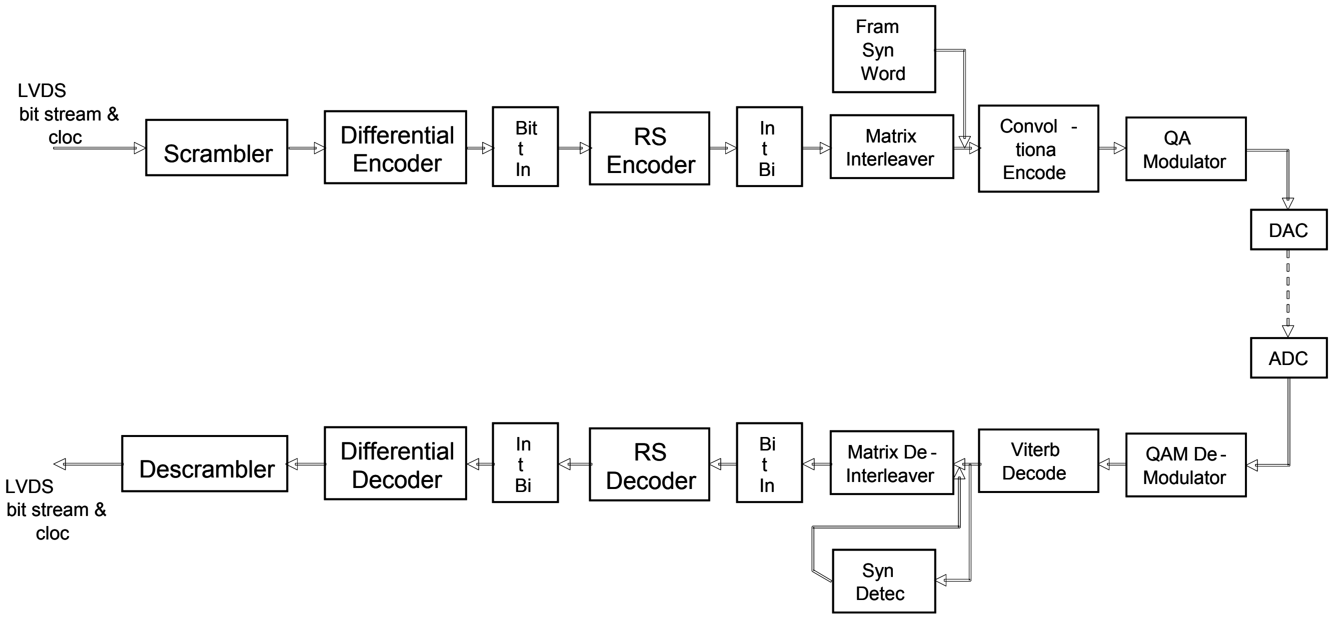

Figure 3.1 – Generic Block Diagram of the SATCOM Waveform

particular design flow, as described below.

A block diagram of the waveform is given in Figure 3-1. Elements that are implemented include: scrambling/descrambling, differential encoding/decoding, Reed Solomon encoding/decoding, interleaving/de-interleaving, convolutional encoding/Viterbi decoding, and RS frame synchronization. A subset of the standard requirements was implemented, limiting the uncoded data rates to 16, 64, 256, and 8096 kbps, rates achievable by varying the input clock, and requiring only a single set of modulation and coding parameters:

- QPSK
- Reed Solomon (219, 201)
- Viterbi ½ rate

To clarify the evaluation, it was decided to separate the effort into the following distinct categories, each covered separately in sections that follow.

- Signal Processing Chain
- Clock generation and issues
- Hardware interfaces

A single individual worked each effort, one handling the traditional RTL development, and another the model-based RTL development. Developer productivity can vary significantly between individuals, but we believe both were evenly matched in the abilities brought to bear on this problem. The individual using traditional methods was a highly experienced RTL designer, having over 15 years experience in the field, including experience in the implementation of communication links. The individual employing model-based methods was also an experienced RTL developer and an expert in using System Generator, but had no prior experience in the implementation of communication links.

3.2. Performance Comparison

3.2.1. Signal Processing

To facilitate accurate measurement of the hours spent in producing a working design, we have partitioned the waveform into functional blocks, with each developer recording or estimating the time spent on that function separately. Additionally, the development process was broken into sub-categories capturing the typical sequence of tasks involved in RTL development. As given in Table 3.1, these categories are:

-- Algorithm & Interface Specification /Documentation
-- Module Design
-- Modeling, Simulation & Design Verification
-- VHDL Coding
-- VHDL Behavioral Verification

Table 3.1 – Comparison of Development Man-Hours: Traditional versus Rapid Development Approach

	Development Time -- Signal Processing Chain (man-hours)						
Traditional Approach							
	Algorithm Interface Specification / Documentation	Module Design Definition	Modelling, Simulation & Design Verification	VHDL Coding	VHDL Code Behavioral Verification	Hardware Integration & Lab Testing	Notes
Reed Solomon RS Encode	40	40	0	40	60	20	Integrate purchased IP
Reed Solomon Decode	20	80	0	60	100	20	Integrate purchased IP
Scrambler / Descrambler	1	1	0	1	6	3	
Convolutional Encode	1	1	0	1	1	1	
Viterbi Decode	8	8	0	8	16	24	Integrate in-house IP, development not shown
Differential Encoder / Decoder	1	1	0	1	4	2	
Interleaver / Deinterleaver	40	16	0	16	36	60	
PSK modulator (2,4,8)	5	5	0	4	3	3	
RS frame Sync	4	6	0	4	6	4	
TOTALS:	**120**	**158**	**0**	**135**	**232**	**137**	**782**
Rapid Development Approach							
	Algorithm Interface Specification / Documentation	Module Design Definition	Modelling, Simulation & Design Verification	VHDL Coding	VHDL Code Behavioral Verification	Hardware Integration & Lab Testing	
Reed Solomon RS Encode	1	0.25	2	0	0	*	Integrate Xilinx Core
Reed Solomon Decode	1	0.5	3	0	0	*	Integrate Xilinx Core
Scrambler / Descrambler	0	0.25	3	0	0	*	
Convolutional Encode	0	0.25	1.5	0	0	*	
Viterbi Decode	0	0.5	2	0	0	*	Integrate Xilinx Core
Differential Encoder / Decoder	0	0.25	1	0	0	*	
Interleaver / Deinterleaver	0	0.5	2	0	0	*	Integrate Xilinx Core
PSK modulator (2,4,8)	1	0.5	4	0	0	*	
RS frame Sync	1	4	16	0	0	*	
TOTALS:	**4**	**7**	**34.5**	**0**	**0**		**45.5**

-- Hardware Integration & Lab Test

These columns require some explanation. The first column contains the time spent up front in architecting the overall design, defining, for example, memory types and placement between modules, clocking requirements, state machines for control, etc. In addition, this column includes time defining the interface requirements between modules. The second column, 'Module Design', contains time expended in architecting and designing the individual modules. For example consider the design of the interleaver, a module involving double buffering, addressing and/or reordering, etc., all decisions requiring time and thought to produce. The column 'Modeling, Simulation & Design Verification' covers the work to simulation test the module prior to VHDL coding to verify that the algorithm is designed and operating properly. The next, 'VHDL Coding', is self explanatory. 'VHDL Behavioral Verification' is the effort to test the VHDL by performing a behavioral simulation, verifying that the input-output behavior matches the results expected. Finally, 'Hardware Integration & Lab Testing' is that time spent verifying the operation of each module on the part itself, a step that is done to ensure that neither propagation delays nor timing errors are altering the behavioral operation.

3.1.1.1 Interpretation of Results

The accumulated hours are shown at the right side of the table (encircled): 782 hours for the traditional versus 45.5 hours for model-based design. As stated, both designs were carried through to implementation on hardware. The traditional design was implemented on a BAE custom radio, consuming 137 hours for integration and test. The model-based design was loaded directly onto a WildCard3000, a PCMCIA card containing a Xilinx 2V3000, and consequently no time was required. To be fair, if one removes the 137 hours spent on hardware integration, we're left with 645 hours, which is still well over a 10:1 improvement – a remarkable result.

Some of the key differences between these two design methodologies are revealed by where the hours were spent. Note that with the tradition flow, significant time is spent in the first two columns, architecting the overall design, the interfaces, and in designing the individual modules. It should be noted that both designs had the benefit of the following cores:

-- Reed Solomon Encoder
-- Reed Solomon Decoder
-- Viterbi Decoder

Thus neither had an unfair advantage due to significant development associated with these functions. Nevertheless the traditional designer spent considerable time on the Reed Solomon functions. This occurred due to the effort required to construct the non-integer based clocks and clock enables required by this module. This did not occur with System Generator as it includes an algorithm used to generate clock enables automatically (without human intervention) based on the various clock domains present in the design.

Also noted is the small amount of time consumed by the model-based designer in these same categories. In this application the modules provided by System Generator met the needs of the user, and no time was required to study and understand the inner workings of the block, or to create a custom block. This was not the case, however, with the RS Frame Sync, which required some custom development (construction of the algorithm using foundational building blocks such as registers, multiplies, logic, etc.) and testing.

Another key observation is revealed in the third column. Notice that no time was spent by the traditional designer to model and simulation test the modules upstream of VHDL coding. Often this step is skipped because of time constraints and very tedious work required to model the algorithm faithfully, in a bit-true and cycle-true fashion. And when all this work is done, there is no direct connection to the final product, so it is perceived as a less-essential, less-productive step. Errors in the design must then be caught at the VHDL Behavioral Verification stage.

On the other hand, when following the model-based process, the designer is inclined (actually required) to build a complete, bit-true, cycle-true behavioral model, because that model represent his final product. The discovery and removal of errors and bugs at that step occurs as a consequence. Notice that this step consumed the greatest number of hours when using System Generator and model-based design. This is typically a process of "build a little, test a little". As modules are brought into the design, time is required to "get it to work" properly. Why is this advantageous? Debugging at this stage is faster and easier than at the VHDL stage. Errors are discovered upstream of compilation and within a graphical user environment. Debugging VHDL behavioral models requires synthesis of the VHDL (i.e. a compilation) and is less intuitive. Thus, considerable time is spent waiting for synthesis to complete and in locating and correcting bugs. This is evidenced in the "Behavioral Verification" column of the Traditional Development effort, where more time was spent than any other category, time avoided with model-based design flow which forces one to address errors prior to VHDL coding (column 3).

3.2.2. Clocking

System Generator determines a clock running at the highest rate and builds enabling logic to throttle any number of lower rates as needed [7]. Thus, the task of deriving multiple clocks is removed from the designer by default. On the other hand, with traditional design, the designer is responsible for generating all clocks, which in the present application was complicated by the multiple clocks needed to handle non-integer data rates across the Reed Solomon coder/decoder. Carrying clock enables through by hand is often difficult for a human, and well suited for machine generation.

3.2.3. Hardware Interfaces

Hardware interfacing is one of the more often difficult aspects of system integration. In this application all of the interfaces were the Low Voltage Differential Signaling (LVDS) type. At the time this paper was written, neither design was implemented through to the LVDS interfaces; however, it is anticipated that neither would exhibit an advantage, and that both would require the same amount of time to implement. One should note that with System Generator, some COTs vendors provide interface blocks, allowing the easy connection of a Simulink model with a System Generator segment targeted to run concurrently on their FPGA card, with no effort required in developing driver interfaces.

3.2.5 Hardware Co-simulation Benefits

The System Generator version of this waveform design, implemented on a Virtex-II Pro 20 (2VP020) consumed 27% of the part and ran easily at 100 MHz (far exceeding the 8 MHz clock rate required to achieve the specified data rates). Hardware co-simulation in free-running clock mode [6] increased the waveform simulation speed by roughly three orders of magnitude over a pure software simulation.

4. SCA COMPLIANCY

The Joint Tactical Radio System is a prime driver of software defined radio technologies [8]. For this system, the software layer controlling the system is known as the Software Communication Architecture (SCA) [9]. In this study a SATCOM waveform was built using System Generator and run on generic co-simulation hardware. It was not SCA compliant. Implementation within an SCA-compliant radio would require the addition of "wrapper code" adhering to the SCA standard, which represents a significant effort by a designer knowledgeable and skilled in the SCA. It is therefore of considerable interest to explore extensions to the design flow that generates SCA-compliant code automatically, using System Generator for development of embedded hardware and firmware and Real-Time Workshop (MathWorks) for generation of application C code.

Many waveform developers would prefer to remain insulated from the intricacies of the SCA, in part because . there are no clear design guidelines in this regard today. The current JTRS endorsed specification (SCA Version 2.2) does not specify all aspects of waveform implementation involving FPGAs and DSP processors. Version 3.0 was intended to address this issue, particularly the Specialized Hardware Supplement, but it is not clear at the time of writing whether or not Version 3.0 will be adopted by JTRS or industry. Other proposals, e.g., OCP-IP, have also been proposed.

Such uncertainty in the specification presents an opportunity for a toolset that can automatically generate VHDL component-level wrapper code for FPGAs or equivalent C code for DSP processors. Such tools could improve waveform developer productivity and significantly lower the cost of development, test and verification associated with an SCA-compliant radio. Furthermore, a tools-based approach would be far more scalable than an approach requiring each waveform developer to track the emerging specification and to write SCA-specific code for each waveform. A tool such as this would take the output from System Generator and automatically generate the code necessary to make the waveform SCA-compliant. The onus of ensuring SCA compliance would then fall on the tool supplier rather than the waveform developer.

Future extensions of the current work include exploration of SCA rapid development approaches. We envision a proposal to build an open-source tool as described above. A further outcome of this study would be design guidelines for development of SCA-compliant waveforms. We expect to report on this work at a future SDR Forum Workshop or Technical Conference.

5. CONCLUSIONS

We have presented a detailed case study performed to quantify benefits of adopting a System Generator design flow for FPGA-based signal processing systems. In the development of even a simplified MILSAT waveform, we observed a 10:1 productivity improvement over a traditional RTL / IP core design flow. As system complexity and device complexity continue to grow, the need for higher level tool flows for embedded hardware and software systems will become increasingly important. It is expected that industry case studies such as this should continue to provide considerable value in sharing knowledge of the "state-of-the-art".

6. REFERENCES

[1] C. Dick and J. Hwang, "FPGAs: A Platform-Based Approach to Software Defined Radios", Chapter in Software Defined Radio: Baseband Technologies for 3G Handsets and Basestations, John Wiley & Sons, Ltd, 2003.

[2] J. Frigo, M. Gokhale, T. Braun, J. Arrowood, "Comparison of High-Level FPGA Design Tools for a BPSK Signal Detection Application", Proceedings, SDR Forum Technical Conference, 2003.

[3] J. Hwang, B. Milne, N. Shirazi, J. Stroomer, "System Level Tools for FPGAs," Proceedings FPL 2001. Springer-Verlag 2001.

[4] The MathWorks, Inc., "About Model-Based Design", http://www.mathworks.com/applications/dsp_comm/description/mbd.html.

[5] The MathWorks, Inc., "Simulink Documentation", http://www.mathworks.com/access/helpdesk/help/toolbox/simulink/.

[6] Xilinx, Inc., "Hardware Co-Simulation Clocking", http://www.xilinx.com/products/software/sysgen/app_docs/user_guide_Chapter_4_Section_3.htm

[7] Xilinx, Inc., "System Generator User Guide", http://www.xilinx.com/products/software/sysgen/app_docs/user_guide.htm.

[8] JTRS Overview, http://jtrs.army.mil/sections/overview/fset_overview.html

[9] SCA Technical Overview, http://jtrs.army.mil/sections/technicalinformation/fset_technical_sca.html

7. ACKNOWLEDGEMENTS

Special thanks to Bob Regis, who developed the waveform using traditional RTL methods and provided many insights as to the relative merits of these techniques. Bob was assisted by Andrew Comba who developed a pure Simulink model of the waveform.