A CASE STUDY COMPARING TRADITION TO MODEL-BASED RAPID DEVELOPMENT OF SDR WAVEFORMS – PART II

David Haessig, Robert Regis (BAE Systems, NES, Wayne NJ, USA, david.haessig, robert.regis@baesystems.com); Mark Hermeling (Zeligsoft, mark@zeligsoft.com)

A CASE STUDY COMPARING TRADITION TO MODEL-BASED RAPID DEVELOPMENT OF SDR WAVEFORMS – PART II

David Haessig, Robert Regis (BAE Systems, NES, Wayne NJ, USA, david.haessig, robert.regis@baesystems.com); Mark Hermeling (Zeligsoft, mark@zeligsoft.com)

ABSTRACT

In [1], the authors compare traditional RTL design flow to that of model-based design, applying both to a common problem – implementation of the physical layer of SATCOM waveform MIL-STD-188-165a. They report a 10:1 improvement in productivity in the areas of algorithm simulation testing, code generation, and waveform integration. That study carried the comparison through to the point of hardware-in-the-loop testing, each design implemented autonomously in a single FPGA node, looping back the transmit signal to the receiver. In the current paper we describe an effort to complete the waveform implementation thru interoperability with another node, a COTS modem. Finally, a paradigm for creating SCA-compliant FPGA-hosted components via auto-code generation is proposed, and the impact and implications that this can have on code reuse is discussed.

1. INTRODUCTION

Field-Programmable-Gate Arrays (FPGAs) are increasingly used in the implementation of software-defined radios (SDRs), particularly for the implementation of the baseband processes in wideband modems. This is due to the fact that they are inherently reconfigurable at runtime, a feature clearly required of SDRs and not possible with Application Specific Integrated Circuits (ASICs). It is also something that allows them to be more amenable, to some degree, to the Software Communications Architecture (SCA) standard required of JTRS Radios.

To encourage their use as an alternative to the ASIC, FPGA developers have produced a suite of development tools and programming methods that closely mirror existing ASICs design flows. This has had the effect of discouraging entry into the pool of FPGA developers, somewhat limiting this pool to those already having a background in chip design. In the last six years, however, this situation has been changing due to the emergence of a new class of FPGA programming flows based on high level modeling in Matlab, Simulink, C, and UML. Those that function within the Simulink environment include the Xilinx System Generator for DSP™, the Altera DSP Builder™, and Synplicity's Synplify DSP™. These are in essence opening a new path to RTL (Register Transfer Level) design by adding an additional layer of abstraction to the layer already provided by VHDL (or Verilog). This is making it easier for a larger audience to become FPGA developers, and increasing developer productivity, especially in the area of modem development [1].

It is interesting to note the type of developers who, from this author's viewpoint, are using the tool. One group that is more readily accepting the tool are the algorithm developers, those who may lack expertise in the traditional ASIC hardware design flow, but have a good knowledge of DSP, Matlab and Simulink. For this group these tools provide a pathway to architecting, synthesizing, and validating high performance algorithms on real hardware. The other group is the traditional FPGA designer who is well versed in the more deeply entrenched methods of RTL development. For them these tools are often accepted more slowly, particularly because these individuals can already produce the FPGA-hosted functionality they need by using methods they already know quite well and also because they see shortcomings in the tools, either real or perceived due to their inexperience. Nevertheless, this relatively new FPGA development flow based on model-based design is proving to provide significant productivity improvements over traditional methods [2,3,4,5], not only through the avoidance of hand-coding, but by enabling better exploration of the architecture space, by combining into a single model the models used for algorithm development and the model used for code generation, and by creating a better environment for joint development of the algorithm, the code, and the test vectors involved in producing embedded DSP on FPGAs.

Recently [1,4] BAE Systems and Xilinx took advantage of a unique opportunity to compare the new and traditional RTL design methodologies discussed above. Two development efforts were run in parallel, both seeking to implement a subset of the SATCOM waveform, MIL-STD-188-165a, on a pair of software-defined radio platforms. In both cases the individuals working these efforts were experts in the design methodology they employed. In addition, both had access to an identical set of IP cores, therefore keeping the comparison fair. The effort for each design was measured in man-hours, each developer logging their time as well as noting features of the design flow that were particularly easy or troublesome. The results of the study were quite remarkable – the design developed with model-based design produced in less than 1/10th the total hours of

the traditional approach! This is a result that is not only quite amazing, but often also received understandably with caution and skepticism. Can one conclude from these results that model-based RTL development will consistently achieve similar savings when developing the physical layer of software defined radios? I think it safe to say that this example yielded a productivity improvement result that is on the high side of the typical or average result. Nevertheless, it shows that in cases such as these, model-based design can very significantly improve the process of RTL development. Additional studies are needed to hone in on the typical improvement achievable in modem development.

To this end, we are working to continue the MIL-STD-188-165a case study, carrying it through the implementation of a complete receiver and transmitter (not simply a loop back within a single FPGA), a transceiver capable of transmitting and receiving from another node, and therefore including all needed receiver tracking functions -- carrier, phase, gain and bit tracking. To prove this, a subsequent goal is interoperation of the transmitter and receiver with a COTS MIL-188-165a modem, a modem known to adhere to the standard, the Radyne DMD2050. Thus we are connecting the Radyne with a LyrTech FPGA platform hosting the BAE developed implementation of that same waveform. Section 2 describes this ongoing work and the experience that one particular author had in delving into System Generator to accomplish it.

A secondary goal of the effort is the production of a test bench for evaluation of new methods for creation of SCA-compliant components hosted on FPGAs. An approach is proposed in Section 3 which addresses a key impediment to the hosting of SCA components on FPGAs – inefficiency.

2. IMPLEMENTATION RESULTS

An objective this case study (Part II) is to capture the efforts required to design, code, and integrate on hardware a MIL-188 waveform capable of interoperating with another modem, comparing the model-based design flow being applied currently with the traditional design flow applied to this same problem done one year earlier. One author, who is an expert in traditional RTL design flow, developed the original design using those methods with which he is very familiar. He also did the work described below to produce a very similar design using model-based design. Thus we have the unique opportunity have the same individual perform both designs, thereby removing the variability associated with the capabilities of two different individuals. This also afforded us the opportunity to assess SysGen from the viewpoint of a new user having extensive RTL development experience but little model-based or SysGen experience. The following sections describe the startup process, the waveform design, and the noted benefits/issues that occurred.

2.1. Getting Started – Conversion from RTL to Model-based Developer

Starting with a digital hardware design background and VHDL firmware experience meant that three new software tools had to be learned. Learning the Matlab language and its companion graphic modeling tool Simulink, both from MathWorks, was the first step. With sufficient software experience the new language is easy to learn and there are hundreds of books that teach it or include examples. The user doesn't need a strong understanding of Matlab before moving on to learning Simulink, and it is very convenient to use them together. A class in Simulink just scratches the surface of what the tool can do, but for the purposes of Sysgen, much of complexity and details can be skipped. Mastery of Matlab and Simulink really requires practice and individual concentration on the specialized toolbox and blocksets applicable to your discipline. The number of functions, parameters, and examples available could take a life time to learn. Nevertheless, the circuit design engineer can get started faster if he can concentrate on the hardware aspects of Sysgen while teamed with a system engineer with more of a mathematical background and presumably years of experience in signal processing analysis and modeling. The three day Xilinx DSP Design Flow class was intense and essential to getting started. There are a lot of new concepts and features that would be hard to pickup on your own. Topics included familiarization with the library of building blocks, number representation, clock enables, simulation, and synthesis. Even with the training it took a bit of practice and thinking how to best apply this tool to the design at hand.

2.2. The Waveform

The current design goal is to implement a subset of the Department of Defense standard MIL-STD-188-165A for satellite communication, including BPSK, QPSK, and OQPSK waveforms. The approach has been to start off with the simplest BPSK waveform and expand the design in stages until a full implementation and compliance is met. The design accommodates numerous selectable data rates up to 10 Mbps and includes various coding options including differential encoding, convolutional FEC encoding, and Reed Solomon encoding.

The Lyrtech VHS ADC V4 board with a DAC add-on contains a Xilinx Virtex-4 FPGA, a fixed 104MHz oscillator, ADC and DAC channels clocked at the 104 Msps rate. Lyrtech supplies a board support package that specifically handles Sysgen designs providing a wrapper with all necessary interfaces to control via software and the cPCI bus and access to various digital I/O in addition to the ADC and DAC chips. This arrangement allows not only

remote configuration of the FPGA but also Hardware in the Loop simulations in conjunction with Simulink.

Due to other priorities, this project began only one month prior to this writing. Consequently, our initial interoperability goal has been restricted to that of the transmiter, as it is naturally less complex than the receive side. In the transmitter, shown in Figure 2.1, the circuit design generates the programmable data rate clock and registers the data input into a FIFO. Buffering is necessary because the data rates are programmed with a resolution that is not achievable by the simple integer division of the sampling clock. The data is clocked out of the FIFO, scrambled, differentially encoded, and optionally FEC encoded by a one half rate convolutional encoder. The data rate is either unchanged or double the initial rate at this point. Reed Solomon block coding and interleaving can be added. This presents a complication because the check symbols added to the data by Reed Solomon results in an increase in data rate that is not an integer but an awkward ratio of numbers. If the information data block size were 239 bytes and the number of check bytes added were 16, then for every 239 byte in 255 bytes would shift out. The encoded data rate in, this fictitious example would grow by 255/239 which is approximately 1.067. A fractional ratio guarantees that you won't get a conveniently commensurate relationship between the data rate and the sampling rate clock. Interpolation will be used extensively to not only increase the sampling rate but also to alter it to an integer submultiple of the unrelated sampling clock.

To align the signal samples running at four times the data rate to the incommensurate rate of the DAC clock we need to perform a rate change through the a programmable interpolator. Interpolators can multiply the sample rate of a bandlimited signal by an integer or even by a rational ratio, e.g. P/Q, if needed to produce a new sampled signal running at the desired rate. In this case we need the interpolator to take pulse shaped data signal sampled at four times the data rate and bring it up to an integer multiple of the half the sampling rate, i.e. 52 MHz. The interpolator used in this design multiplies the data by a factor of 1040/Q and is then followed by a fixed times two interpolator. This cascade of filters and interpolators results in a programmable data rate of 4 * 1040/Q * 2 which results in the exact DAC clock sampling rate of 104 MHz. The result of these figures is that for any non-zero positive integer value of Q the data rate can be programmed with a resolution of 12.5Kbps.

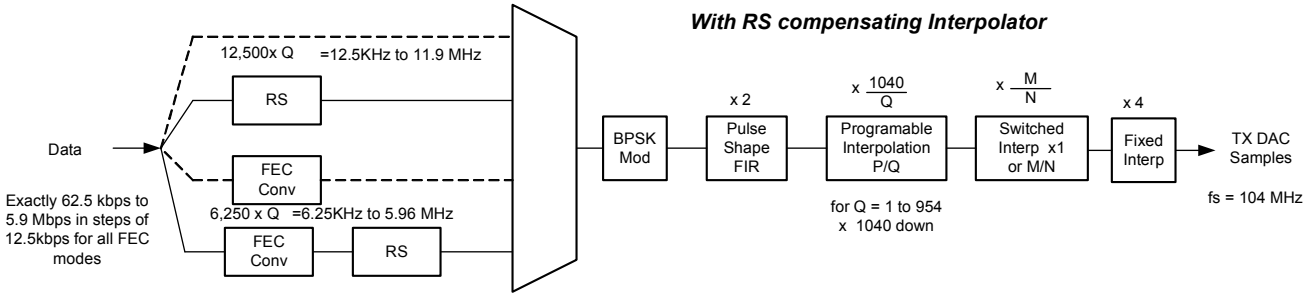

Figure 2.1 – Transmit Diagram

2.3 Benefits/Issues when Designing with SysGen

The first step in recording and communicating the design is to form a series of block diagrams in a tool like Visio for instance. This is the natural way the human brain works most efficiently. In addition, it is why board level circuit schematics continue to be drawn and why a programmer or VHDL designers often resort to drawing a sketch to clarify a point. So there are strong reasons to desire an effective design graphically based design tool for firmware and good reason to assume that better productivity can result. Schematic based design tools for firmware exist but have not dominated the industry for a number of reasons. Text based design using VHDL and Verilog hardware design languages still dominate despite the fact that they can be unnatural and tedious. Despite these obvious disadvantages, they are widely used because of the perceived advantages in the areas of standardization, reuse, portability, and wide spread knowledge base. Unfortunately even with the latest refinements available from competing vendors, these tools can't provide the level of productivity desired. Wider spread availability and diversification of IP blocks has provided the last significant boost in productivity with this methodology.

Sysgen brings a lot more to the table than the schematic capture tools ever attempted while providing all the natural desirability of a graphical tool. Matlab, Simulink, Sysgen, and ISE essentially form an integrated design environment (IDE) optimized for the development of signal processing models and implementation. You can calculate filter coefficients easily with numerous design functions in Matlab. You can then paste the values or the formulas directly into the parameter dialogs of the Sysgen blocks. You can then simulate a combination of synthesizable models in parallel with conventional Simulink models and

compare them on a waveform display modeled after an oscilloscope. You can capture the results into the Matlab workspace and immediately plot them or their spectrum. The flow from simulation to design to evaluation and verification can be very fluid and represents a breakthrough in a fully integrated development environment.

The long sought holy grail of closing the gap between system engineering analysis and hardware design engineering may finally have arrived. If the communication system signal processing system engineer with strengths in mathematics and modeling can successfully communicate and share a common understanding and design with the hardware design engineer, a lot less information and mistakes will be made. Typically a system engineer's design is simulated and reported, and his job is done. The hardware engineer picks it up and inevitably starts his design from a partial understanding of what was accomplished with a set of tools that operate in the implementation domain with absolutely no semblance to the original model. The model is necessarily discarded and the final hardware design can not be verified against the original model that the system engineer worked so hard to perfect. With Sysgen the diagram is the model, which is simulated and analyzed with Simulink and Matlab, and it is also the firmware design that is generated and also simulated to verify successful translation. Design errors in the model are found and corrected earlier in the design process and the use of this same model through the implementation can be verified to be identical. The discovery of defects during the hardware design, or worse, after the product is completed because of faulty verification can be very costly. Many have been seeking the ultimate tool that accepts a design drawing and by pushing the magical button, the hardware automatically appears. We may never perfect this process but we may be close enough to make significant improvements in productivity and life cycle cost reduction.

Sysgen does a number of things behind the scenes that can save time. One has to know how it handles different clocking rates. Good design practice dictates that you design entirely synchronously and minimize the number of clock domains. Sysgen assumes that you are using one clock for the entire design, or subdesign. Within this clock domain it generates the appropriate clock enables at any integer submultiple of the clock invisibly. If you have an interpolating filter that produces four outputs for every input sample the clock rate divided by one and divided by four clock enables are automatically generated. Also if you insert this filter block into the design and run the input rate at the full clock rate it will generate an error. The filter needs the output clock to run at four times the input rate and that wouldn't be possible if the input rate is running at the full rate. If you are not entirely familiar with the internal design of a block, this can surprise you. Sometimes it is not that easy to determine why the error is happening but it usually can be avoided if the author of the block reveals the clocking requirements in help text. We encountered these errors quite often while learning to use the tool and applying FIR filter components that were predesigned and provided by the library.

Although Sysgen excels at signal processing signal flows, there was skepticism initially about how well it would handle other functions like control timing, state machines and the like. One could certainly continue using a VHDL top level design and inserting signal processing subsystems designed with Sysgen as needed. When something a bit more complex is needed like a BER detector and counter, the choice depends on which tool the individual designer feels most comfortable using. There was also the question of whether a state machine which is clearly a control mechanism can effectively be written and debugged in Sysgen. Mcode blocks seem to have solved this problem, allowing you to use a subset of Matlab code which is just as descriptive and well documented as the VHDL version.

The interested reader is referred to the presentation material for the quantitative comparison of the two design flows when applied through interoperability.

3. AN APPROACH FOR FPGA-HOSTED SCA COMPONENTS

The US DoD is sponsoring the development of the Joint Tactical Radio System (JTRS), a family of SDRs requiring the use of a software module interface standard called the Software Communications Architecture, or SCA [6,7]. The goal of the SCA is to enable software portability, thereby promoting software reuse across platforms, reducing costs, shortening development schedules, and quickening the incorporation of new processing devices as they become available.

The SCA as originally designed applies to General Purpose Processors (GPPs) which have sufficient resources to support a CORBA transport interface between components. Several different approaches have been suggested to extend the SCA to Special Hardware Processors (SHPs). We will look at the simplest approach that will enable us to integrate the functionality generated by SysGen into an SCA compliant application. The focus of this discussion shall be an architecture involving a single GPP connected to one or more FPGAs.

An often heard complaint leveled toward the SCA is that of code or processor inefficiency [8]. Inefficiency impacts power consumption, size, and cost, none of which can be spared particularly in handheld devices. This is especially true with FPGAs. In many applications there are often little to no resources available for a CORBA-type of interface. Another issue directly related to inefficiency is the complexity of the interface. FPGA interfaces have

traditionally been quite simple. For example, in an older family of BAE radios the interface is just clock and data, with a single bit value clocked in at a steady clock rate. There is probably no interface that is simpler or uses fewer FPGA resources. As data rate requirements have increased, high performance serial transport links are being incorporated such as RapidIO and PCI Express. Allowing the FPGA designer to select the data interface type appropriate for the design ensures processor efficiency, as an appropriately sized interface can be used. Code portability and reuse can be achieved by standardizing on the interface design as described below.

Another problematic notion for SCA compliant FPGAs is that of runtime component deployment and connectivity. The use of *multiple* SCA components in a single FPGA connected at runtime does not fit with the FPGA design flow. It implies that the synthesis, mapping, and the place & route operations be performed at the radio node, which is impractical as the radio nodes will often not have the processing resources needed to accomplish those tasks which can take hours even on a powerful workstation.

Runtime deployment can be anticipated though. Keep in mind that every possible deployment involves a mapping of components to processors. Each of these deployments needs to be tested before the SCA waveform is released. Hence an FPGA loadable bit-stream can be created a-priori for each of these deployments. Unanticipated designs would be built when defined and uploaded through the network. Another option to consider is the use of partially reconfigurable FPGAs with a single component per reconfigurable area and CORBA interfaces between them. A single reconfigurable FPGA is comparable to multiple non-reconfigurable FPGAs, hence this a-priori placing and routing will still be sufficient.

SCA Enabling RTL logic involves three major steps:

1 Communicating data to and from the RTL logic
2 Providing the RTL logic with values for it's configurable properties
3 Somehow implementing the CORBA CF::Resource interface for the RTL logic

One of the proposed standards for SCA-enabling FPGAs is the Component Portability Standard (CPS, also known as CP289) [9,10]. It provides guidelines as to how to implement the three steps mentioned above. CPS discusses the use of the Open Cores Protocol (OCP) as a basis for interface definition since it is bus, technology, and language independent. The proposed approach, embracing that idea, is as follows:

- Allow the selection of the FPGA interface type appropriate to the design, thereby having minimal if any impact on efficiency.
- Define the interface layout using OCP as a guide and apply that to all SCA components to be hosted on that FPGA, thereby enabling reuse
- Use the CPS idea of a global proxy to encapsulate the RTL logic as a *single* SCA component, thereby having no impact to the SCA with regard to the FPGA-hosted component,

As a tutorial example of this approach, consider how it is being applied in the design of the MIL-STD-188 waveform. In that application, Dual Port RAM was selected as the interface to carry data into and out of the transmitter and receiver. As shown in Figure 3.1, a single SCA component is contained in the FPGA, with a global proxy creating the SCA interface and including access to the data and address busses connected to the FPGA hosted RAM. The entire RTL logic, including the Dual Port RAM, is being constructed in SysGen. Note that this approach can be extended to allow for a high speed link, for example, PCI Express, by placing the associated core in the data flow path. The content of RAM, i.e. the memory map, is defined to include 4 segments: Configuration Parameters, Control and Status Words, and Data. The Configuration Parameters are read by the proxy to understand the interface and device operation. Control & Status Words are defined to contain the OCP suggested control flags (initialize, start, run, stop, release, configure, test), and the status flags (init done, release done, config done, test passed), with port control handled as suggested in Figure 3.2. A single proxy component can interface with any FPGA module written to this standard. A new waveform definition with a different FEC module, for example, built to this same standard, permits the proxy to be reused. In addition, the source code in the form of the SysGen model is also reused, although modified in accordance with the new FEC scheme. Note that this approach, applied here with RAM, can just as easily be applied with other common interface types, such as a FIFO, PCI Express, or others. Another potential advantage of precise definition of the interface content is the potential for auto-generating the GPP code implementing the proxy.

4. CONCLUSIONS

RTL development using the model-based design flow is contrasted to the traditional design flow in this continuing case study involving MIL-STD-188-165a. This paper describes the work to complete the waveform development through interoperability with a COTS modem, logging the time consumed by the effort and comparing that to a prior, traditional-flow effort. A method for design of FPGA-hosted SCA components is provided. This method has a significant advantage over those that place CORBA-like constructs into the FPGA, having minimal impact on

resource efficiency and allowing reuse by creation of interface standards.

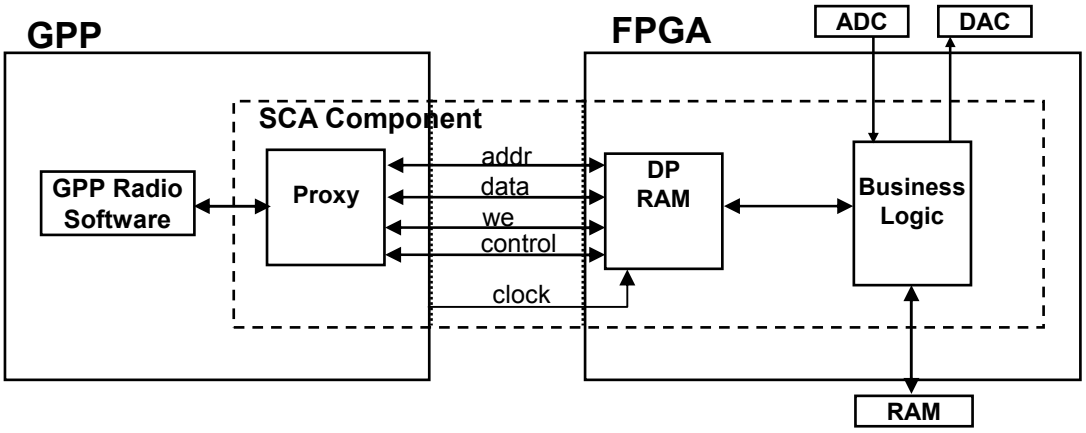

Figure 3.1 – The MIL-188 example shown using a dual-port ram interface, one of many possible interface types; note a single SCA component per FPGA

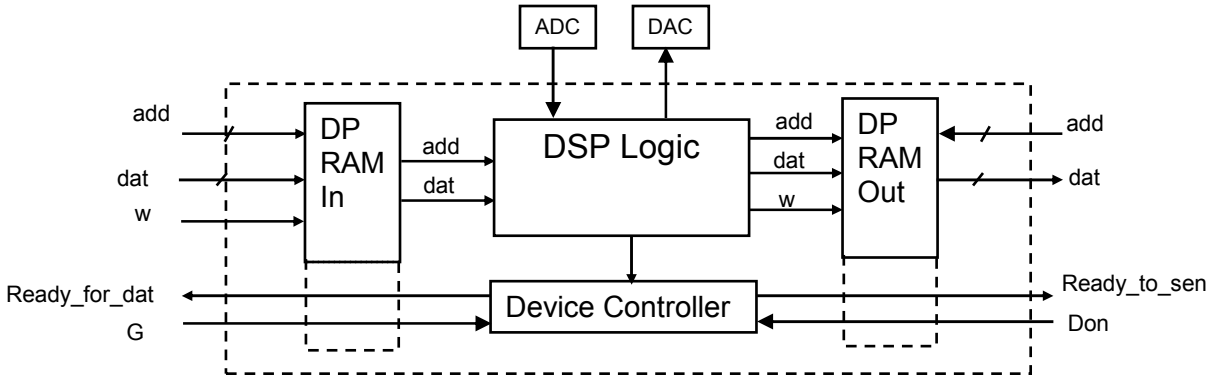

Figure 3.2 – MIL-188 FPGA block diagram with dual-port ram interfaces containing data and OCP defined control, status, and configuration parameters, memory mapped within RAM

5. REFERENCES

[1] Haessig, D., et.al., "A Case-Study of the Xilinx System Generator Design Flow for Rapid Development of SDR Waveforms", SDR Forum Technical Conference, Orange County, CA, Nov. 2005.

[2] Mathworks User Story, "Nissan Develops Emission Reduction System for Mass-Production Vehicles Using MathWorks Tools", http://www.mathworks.com/company/user_stories/userstory7614.html?by=industry

[3] Mathworks User Story, "RealTek Gains 50% of Market Share with a New Audio Chip Designed with MathWorks Tools", http://www.mathworks.com/company/user_stories/userstory4685.html

[4] M. McHenry and D. Raun, "BAE Systems Proves the Advantages of Model-Based Design", http://www.mathworks.com/company/newsletters/digest/2006/sept/bae.html?_cid=MLD0906naad3TA1

[5] 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, 2004.

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

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

[8] G. Bishoff, "SDR might be turning a corner", MTR Bulletin, V4, N40, August 2006.

[9] JTRS Program Office, "Extension for component portability for Specialized Hardware Processors (SHP) to the JTRS SCA Specification (a.k.a. CP 289)", v3.1, 17Mar05.

[10] J. Kulp, M. Bicer, "Integrating Specialized Hardware to JTRS/SCA Software Defined Radios", Milcom 2005.