RAPID WAVEFORM MODELING TECHNIQUES FOR SOFTWARE DEFINED RADIOS

Steven W. Cox
Steve.Cox@gd-decisionsystems.com
General Dynamics Decision Systems, 8220 E. Roosevelt, Scottsdale, AZ

ABSTRACT

A modeling process for rapidly developing waveforms on Software Defined Radios (SDRs) is presented. The Benefits of using this process are highlighted. The techniques presented in this paper are the result of research and product development for the General Dynamics Wireless Information Transfer System (WITS) which includes these military waveforms: AM, FM, Havequick, SINCGARS, SATCOM, and Link11. During WITS development, the Matlab/Simulink tool has been sucessfully applied to Waveform design to reduce product development time and guarantee specified performance. Both floating and fixed point modeling techniques will be discussed for DSP processors using CORBA as well as FPGAs.

By using this process of modeling, both Systems Engineers and Hardware/Software Implementors have visibility of the system design which leads to faster implementation with fewer design flaws.

1. INTRODUCTION

Software Defined Radios (SDRs) are mainly composed of two distinct entities: The hardware platform and the software application. The applications are known as "waveforms" and can be implemented in either real-time software, FPGA logic, or a combination of both. Regardless of implementation, the radio system is defined by the waveform running on the platform.

2. ACCURATE SYSTEM MODEL: THE FOUNDATION OF A WAVEFORM DESIGN

Because waveforms are real-time applications, it is critical that the signal processing paths be modeled accurately. Waveform control (mode switching, parameter refresh) and data processing (transec algorithms) can be included in the system model. Waveform control components are optional for waveform modeling because the model rarely benefits from a highly accurate model in these areas. The cost (simulation time, development time) of building these non-signal processing models does not justify their benefits. A typical SDR waveform model will consist of three components: 1.) The Transmiter, 2.) The RF Channel, and 3.) The Receiver as shown in Figure 1. The Transmitter and Receiver components shown include the RF interface and modem functions but may optionally include cyrptography, source coding, audio/video processing, and applications such as a browser, sensor, or display. To reduce simulation time, the model is designed to run at the baseband level or low-pass equivalent [1]. Models for direct IF up-conversion/downconverson can be used to verify the signal processing effects on baseband signals but the expense of longer simulation times is incured.

Each component of the waveform model contains subsystems that are termed "Objects" which model the exact algorithms performed by software Corba Objects, FPGA Cores, or hardware. This approach provides the benefit of being able to compare test vectors generated with the waveform model against software, FPGA, and Hardware implementations. An example of some basic model objects is shown in Table 1.

Most of the objects modeled are created from standard Matlab/Simulink Library blocks. In some cases such as an FPGA implementation or complex software algorithm, it is necessary to use a Matlab S-Function. "An S-Function is a computer language description of a dynamic system" [2]. The S-Function is created using C-code inside a Matlab wrapper which is compiled into a *.dll file. This *.dll is then called from a standard S-Function block in Simulink. Custom inputs, outputs, and parameter menus can be added to the S-Function. Some generic FPGA S-Functions are available as Off-the-Shelf Blocksets such as the Xilinx System Generator and Altera DSP Builder. These products are add-ins to the Matlab/Simulink Library and are discused in more detail in the Fixed Point Modeling section of this paper.

3. MODEL PARTITIONING

In addition to decomposing model subsystems into objects, the partitioning of objects to run on the hardware platform should also be captured in the waveform model. Partitions can be classified into several levels. At the top level is the hardware interface, for example, RF hardware up/down converters, modem board, security board, audio

board, serial I/O board. Below the hardware interface, partitioning can occur among various processors that occupy each board including FPGA processors. The advantage of this partitioning is that the detailed subsystem requirements can be allocated so that implementation is more focused on a per processor basis. The total requirements including processor utilization and latency for a given processor can therefore be assesed by using good partitioning. Furthermore, all I/O for each processor can be captured to use for hardware/software integration and validation.

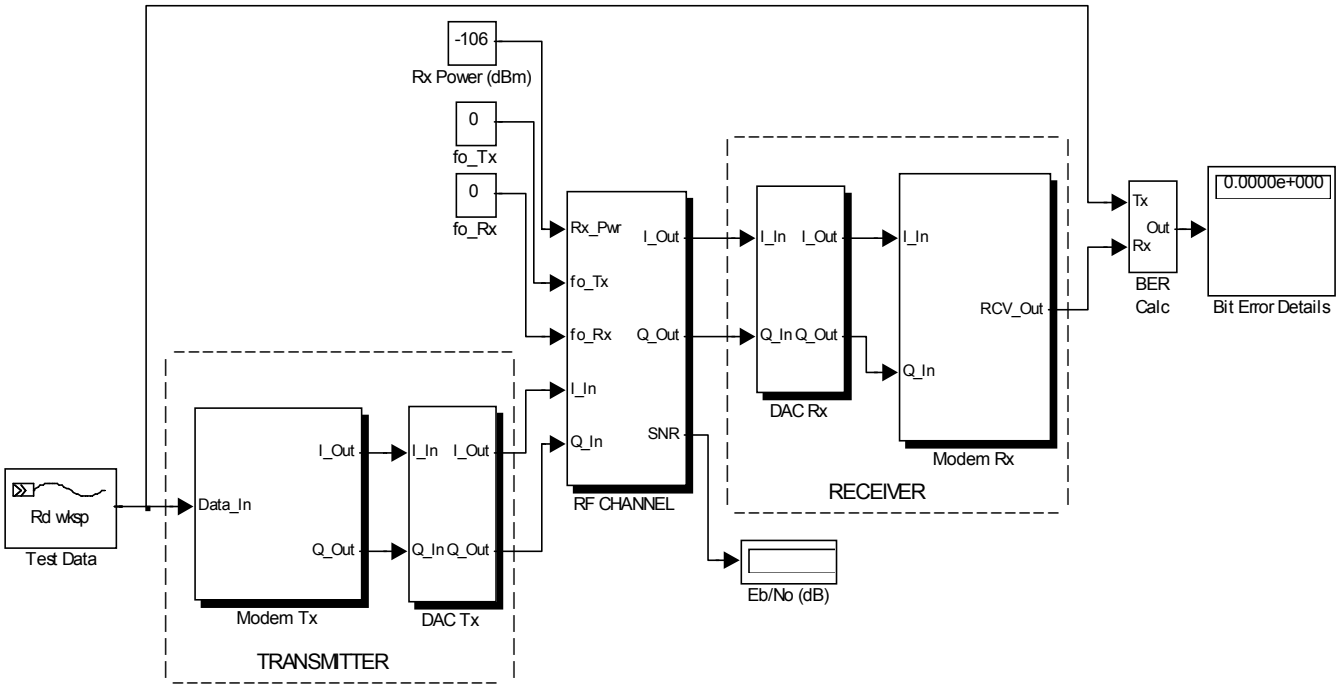

Figure 1. Waveform Model Components.

	Transmitter		RF Channel		Receiver	
O	**Modem Tx**	**DAC Tx**	**RF Tx**	**RF Rx**	**DAC Rx**	**Modem Rx**
B	Type converter	Amplitude Ramping	Tx Frequency Offset	Rx Frequency Offset	I/Q Decimation	FIR Pre-Detection Filters
J	Repeat 4X	I/Q Interpolation	Interferer/ Jammer	AWGN Combiner	Signal Conditioning	Demodulator/ Carrier Tracking
E	FIR Pulse Shaping Filter	Digital I/Q Modulation		I/Q Mismatch	FIR Filtering	IIR Filtering
C	Modulator			DC Offset		Bit Syncronizer/ Symbol Tracking
T	Interleaver			Rx Analog Filters		Correlator
S						De-Interleaver
	FEC Encoder					FEC Decoder

Table 1. Example Objects Within Model Components.

4. MODEL STRUCTURE

A proven method has been used to model waveforms using Matlab/Simulink. The method consists of three main parts: 1.)Matlab *.m file script, 2.)Simulink *.mdl file, and 3.)Filter and data files *.mat files. Running the *.m file initializes the model system parameters, loads all filter coeficients, loads all test data, and plots all filter responses. Next, the Simulink model is run which uses the parameters and coefficients stored in the Matlab Workspace. Simulation data can be output to real time FFT analyzers, scopes, stored as *.wav files, or stored in the workspace for further analysis and refined plotting.

5. FIXED POINT MODELS

Many SDR applications require high speed signal processing and control timing. The FPGA is frequently used for demanding fixed-point processing. To model the fixed-point subsystems several off-the-self tools are available for Matlab/Simulink.

The most generic, is the Matlab Fixed-Point Blockset which is a collection of bit-true primitive blocks and an overflow monitor. Various rounding and truncation options are found in each block and the ability to overide using double precision is available locally as well as globally. The advantage of using this blockset is that the impact of bit-width and scaling of each fixed-point operation can be determined through simulation. Also, the ability to override data types with double precision is provided so that a comparision between fixed and floating point implementations can be analyzed. One dis-advantage of this tool is the absence of cycle-true simulation which accounts for the real-time clock interaction between blocks. Another dis-advantage is that there is not a one-to-one mapping between the model blocks and actual FPGA implementations. This increases development risk since an experienced FPGA designer is required to interpret the Matlab model and design equivalent algorithms in FPGA native code.

More specific blocksets for FPGA modeling are available through the FPGA manufacturers. These include the DSP Builder from Altera and the System Generator from Xilinx. The concept behind these tools is that each block in the blockset is a C-code model of a highly optimized VHDL core. VHDL code synthesis is made possible by a compiler block which translates each block's parameter settings and the connections between blocks into a VHDL directory. The VHDL code produced can be highly optimized if good FPGA design practices are observed during model contruction. In addition to being bit-true and cycle-true these tools offer the added advantage of code systhesis. Many improvements in the FPGA cores are made possible by advances in FPGA technology [3], and use of blockset compilers allows the design to evolve and track available technology.

When designing for fixed-point implementations, the dynamic range of each processing function is critical to overall system performance. In fixed-point modeling, the dynamic range is controlled by the number of bits and the use of scaling (binary point) within each primitive block. In many cases, the fixed-point model is designed to closely approximate floating point processing within a given dynamic range. The more constained the dynamic range and signal properties are, the more efficient the design since fewer bits can be used to process a given signal over a restricted range. The effects of improper dynamic range management can be seen in the form of dc offset, quantization noise, and overflow. The use of rounding vs. truncation can mitigate the dc offset problem at the expense of more processor resources. Using more bits per function can reduce the quantization niose floor but also increases processing requirements.

Test signals such as sinusoids, unit impulse, dc offset, and modulated I/Q data have proven useful in analyzing dynamic range requirements and performance of the model. A common flaw in fixed-point design is the disregard to real-world signal responses. The presence of noise, interferers, and transients can have various unwanted effect in fixed-point functions. The advantage of working in the Matlab/Simulink environment is that many real-world signals can be modeled using the standard library. Also, laboratory test signals can easily be imported to the Matlab workspace from test equipment such as a digital oscilloscope or network analyzer and run through the fixed-point models.

6. REAL-TIME MODEL ANALYSIS

In addition to signal processing performance, other real-time aspects of the system model should be evaluated to ensure feasible implementation and temporal performance. These include: Latency, Throughput, Processor Utilization, and Group Delay. The analysis of these performance measures is more important for software processors vs. FPGA processors. One reason is that FPGA processors perform sample by sample processing which is strictly tied to a sample clock. Another reason is that FPGA processors are much faster at performing functions. Conversely, software processors need to perform packet (block) based processing in order to avoid high overhead for each object in the processing chain. Because the packets are not usually synchronized to a common reference clock, flow control becomes an issue as well as synchronization with time critical events such as correlation and frequency hopping control.

Processor Utilization for an FPGA is total number of logic cells that the design will occupy. This can be easily computed by the FPGA software implementation tools before running on an actual FPGA.

Software processor utilization is dependent on data packet size, processor speed, and function complexity. To get accurate estimates of utilization, characterization of the exact function running on the processor is required. The feedback loop between model and characterization causes the waveform design flow to be less iterative. For many functions, software processor utilization for a single function can be computed as

$$U_{SW} = \left(\frac{\text{function processing time}}{\text{sample}} \right) \text{(fs)}$$

Where fs is the sample rate in samples/second and the function processing time per sample is a normalized rate of performance. The cumulative percent utilization can be computed as

$$U_{SW\_Total} = \sum_{N=1}^{M} U_{SW_N}$$

Where M is the total number of processing functions in the signal processing chain.

Latency mainly depends on packet size, buffer size, and sample rate. For objects that perform buffering only, the latency can be computed as

$$L_{\min} = \frac{\text{buffer length}}{\text{fs}}$$

For objects that perform a signal processing function latency can be computed as

$$L = (\text{packet size}) \left(\frac{\text{function process time}}{\text{sample}} \right) + L_{\min}$$

It is convenient to build models that use a frame size (Matlab term for packet) equal to the actual packet size. The benefit of this accuacy is that the software design can direcly follow the model. Simulink shows the frame size automatically above each interconnect line between subsystems. By having characterization data available for existing model objects, the waveform designer can predict latency and utilization at the system level and thus optimize the design latency through the model.

7. RF CHANNEL MODELS

It has been proven useful to use a common RF channel model among many waveform models for a given hardware platform. In many software radio applications, the system noise figure is a dominant specification for the RF channel model. In addition, the RF power level (P_R) in dBm is often specified for performance measurements such as receive sensitivity, interference levels, and detection thresholds. A typical RF channel model will input modulated complex I and Q signal pairs as

$$s(nT_s) = A\cos(2\pi f_m nT_s) + jA\sin(2\pi f_m nT_s)$$

where A is constrained to 1.0, n is the sample index, and Ts=1/fs is the baseband sampling interval. The linear noise power can be computed as

$$P_N = kT_R B$$

where k =1.38e-23 (Boltzmann's Constant) and T_R is the receiver system temperature. This temperature can be written as

$$T_R = 293 * (10\text{^}(\text{NF}/10) - 1) + 293$$

where NF is the receiver noise figure in dB. B is the noise bandwidth, which may be equivalent to fs if the signal is complex. By taking the square root of the linear noise power, we can obtain a noise gain that can be used to scale a zero mean random noise generator with variance = 1.0.

$$G_N = \sqrt{P_N}$$

This noise is then added to the signal s(nT) as shown in Figure 2.

To adjust the signal amplitude based on a given RF input power in dBm, a signal gain is computed as

$$G_S = \sqrt{2(10^{\frac{(P_R - 30)}{10}})}$$

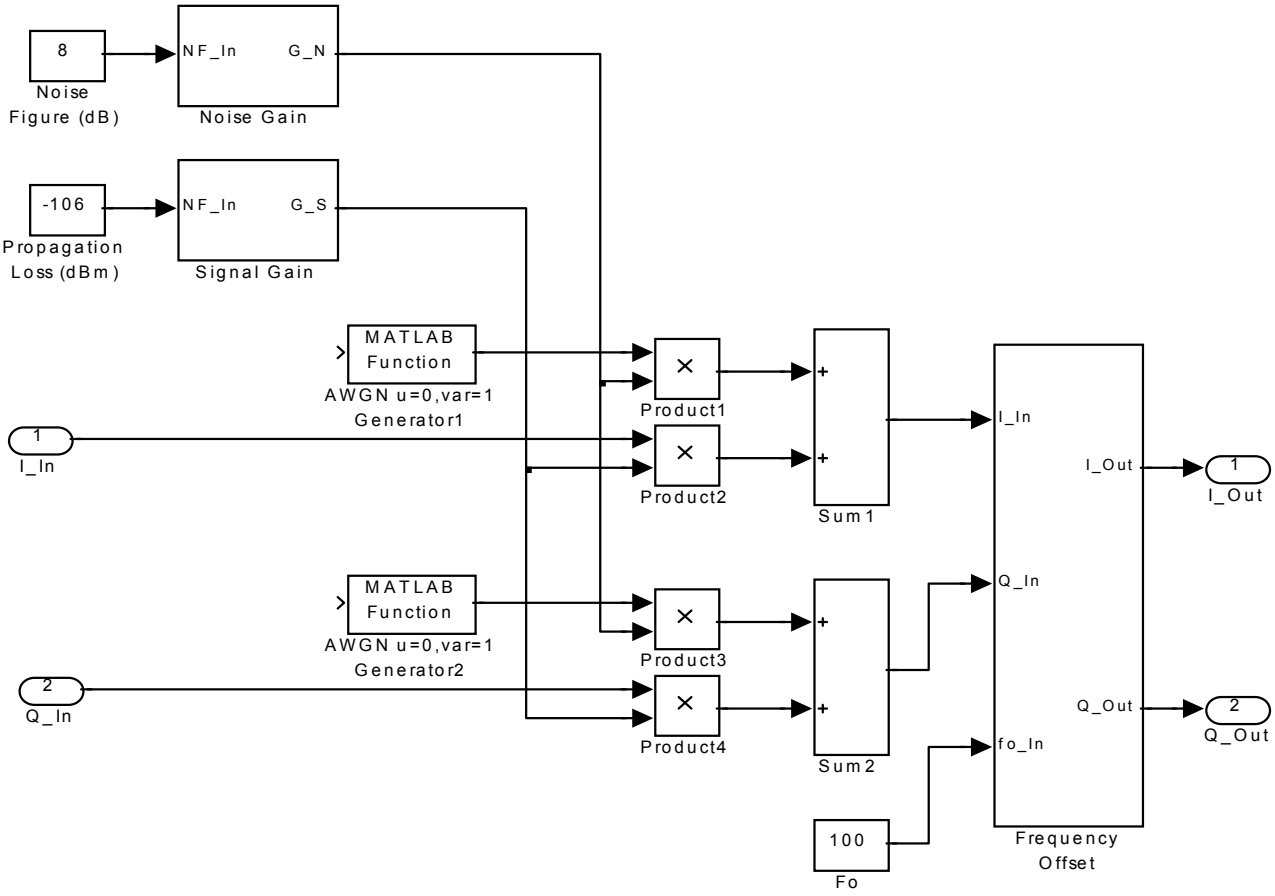

Figure 2. RF Channel Model Components

Note that the AWGN noise generators must be statistically independent. The advantage of this modeling technique is that a common RF Channel subsystem can be used to combine AWGN for any transmitter whose output is normalized to +1/-1 amplitude. Since it is convenient to impose normalization of the transmitter I/Q outputs for SDR systems, this technique achieves the goal of having an accurate system model.

This RF Channel model accounts for receiver hardware noise floor with respect to input power. Additional model components such as RF AGC, Analog filters, and amplifier non-linearity can be added to enhance the system model accuracy.

7. REDUCED PRODUCT CYCLE TIME

The reduction of waveform development cycle time is of key importance to many SDR programs. To demonstrate the main benefits of using this modeling process, the application of the model to the software/firmware development process is shown Figure 3. Here the waveform model is created during the system design phase and is used to drive both the FPGA and software development cycles. Because of the philosophy of an accurate system model, both the FPGA design and the software design have a one-to-one mapping with the system design. This allows the verification of units within the FPGA and software to be done using test vectors generated directly from the waveform model.

For FPGA development, the use of standard software tools alone can lead to a lengthy and costly development cycle. By the use of waveform modeling, the FPGA design can be simulated very quickly using system level tools and then a minimal set of test vectors used to compare the FPGA implementation with the waveform model.

In the software development cycle, long development times can result from the gap between system design and software implementation. To help bridge this gap, the waveform model can be used directly by the software engineer to produce a production C-code equivalent. It has been proven useful for the software engineer to work directly from the waveform model when creating the software code. By running simulations of the model, the software engineer can gain an understanding of how the waveform behaves and is able to generate software test vectors directly. This results in savings of manpower since the waveform system engineer is relieved from generating an unknown number of test vectors.

In addition to software and FPGA development cycle improvements, the waveform system design cycle is greatly improved using this modeling approach. System

design documentation and design reviews greatly benefit from the graphical output of the model. Model subsystems, filter characterization plots, time/frequency signal plots, and performance (SINAD/BER) are all model generated.

This approach is unique in that the creation and use of the Matlab/Simulink models is tailored around the design of waveforms. In essence, the process is waveform centric so that visibility of the waveform is kept at a system level where key decisions can be made that affect: 1.) Development cycle time, 2.) Performance, and 3.) Cost. Quantitatively, software development cycle time has been reduced by 30%.

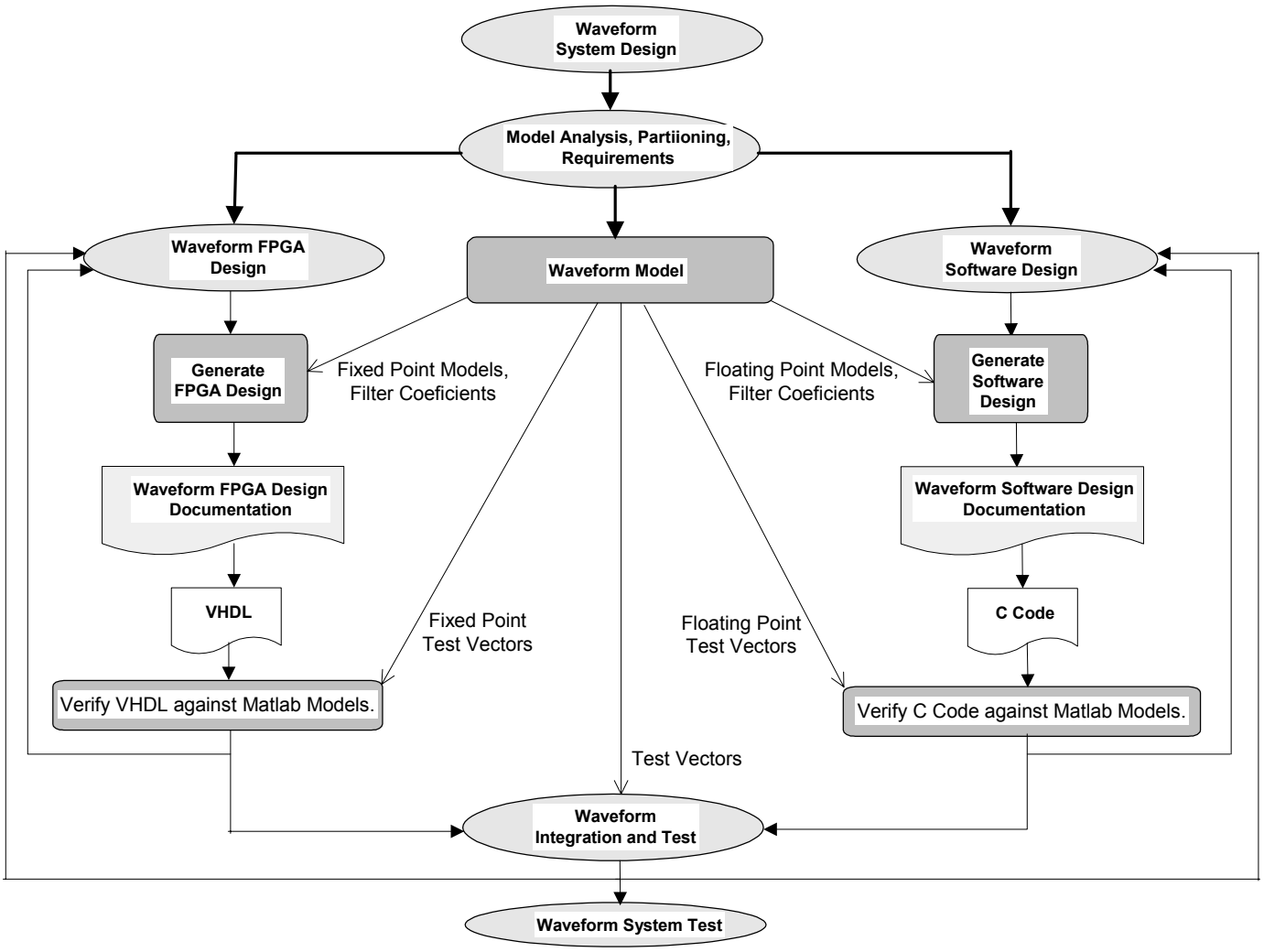

Figure 3. Rapid Waveform Modeling Applied to Software/FPGA Development Cycle.

8. CONCLUSION

In this paper, a process and techniques were given for rapid waveform modeling. The importance of an accurate system model was discussed and how that model fits into SDR development to reduce production cycles. The benefits of using this process were presented through the use of flow diagrams. Because of the graphical nature of the modeling tools used and the accuracy of the model, development teams are able to work efficiently to produce higher quality products for software defined radios.

9. REFERENCES

[1] M.C. Jeruchim, P. Balaban, and K.S. Shanmugan, *Simulation of Communication Systems*, Kluwer Academic/Plenum Publishers, New York, 2000.

[2] The Mathworks, Inc., *Writing S-Functions*, 1998.

[3] Dick, C.; Harris, F., *FPGA signal processing using sigma-delta modulation*, IEEE Signal Processing Magazine, Volume: 17 Issue: 1, Jan. 2000 Page(s): 20 -35