FPGA BASED WAVEFORM DESIGN TECHNIQUES FOR SOFTWARE DEFINED RADIOS

Steven W. Cox (General Dynamics Decision Systems, 8220 E. Roosevelt, Scottsdale, AZ, steve.cox@gdds.com)

ABSTRACT

Design techniques are presented for FPGA based waveforms on Software Defined Radios (SDRs). The benefits of using these techniquies will be highlighted.

This paper assumes that FPGAs will be the target processor for all real time signal processing of a waveform. The role of software processors will also be analyzed with respect to the FPGA based waveform. The techniques presented in this paper are the result of internal research and development for General Dynamics wideband modem development.

In waveform development, the FPGA has been sucessfully exploited to cover areas where high-speed and low latency signal processing is demanded. FPGA waveform design flow has many trade-offs and the types and efficiency of different design flows will be presented. FGPA integration flow also has benefits and limitations that will be identified.

Quantization and optimization techniques for FPGA utilization will be shown for waveform development. Multi-Mode and Multi-waveform FPGA designs will also be discussed.

FPGA waveform interfacing methods will be discussed as a means for communication between various system processors.

Various partitioning methods of control/data processing vs. signal processing will also be shown.

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. FPGA BASED PLATFORM ARCHITECTURES

In order to implement FPGA Based Waveforms efficiently, the SDR platform architecture must be appropriately defined. Many SDR platforms contain a mix of FPGA, Digital Signal Processor (DSP), General Purpose Processor (GPP), and ASICs. These mixed architectures tend to offer the waveform designer a non-homogenous development environment. Hence, many different disciplines and interfacing challenges exist that impede the development of real-time waveforms.

To remedy this situation, the layout shown in Figure 1 may be used. Here, FPGAs are the central real-time processing machines connected together with high speed data buses. Sparse GPPs are used for control and non-real time processing. This philosophy is similar to that of the PC computer, which uses a GPP to interface with the user while it controls dedicated hardware such as modems, video display cards, audio cards, DVD players, and other real-time processing peripherals.

Another reason for selecting such architecture is that software programming is most easily done for non-real time tasks. Again, the PC computer is an excellent example of this point. It demonstrates a software controlled system rather than a software defined one.

Finally, there are some very good benefits from choosing an all FPGA based platform. These benefits include: low-latency, high speed, parallel processing, real time synchronization on a sample by sample basis, and real time telemetry monitoring.

Embedded Microcontrollers and on-chip GPPs are available inside FPGAs and can be used for control and non-real time processing in addition to external GPPs.

Military systems require separation of Black and Red Data. This is usually controlled by the NSA and therefore separate hardware devices are needed for the security section of the platform. Also, FPGA Based Waveforms are easily made Software Computer Architecture (SCA) compliant. A simple software interface object to FPGA is required to encapsulate the FPGA functions.

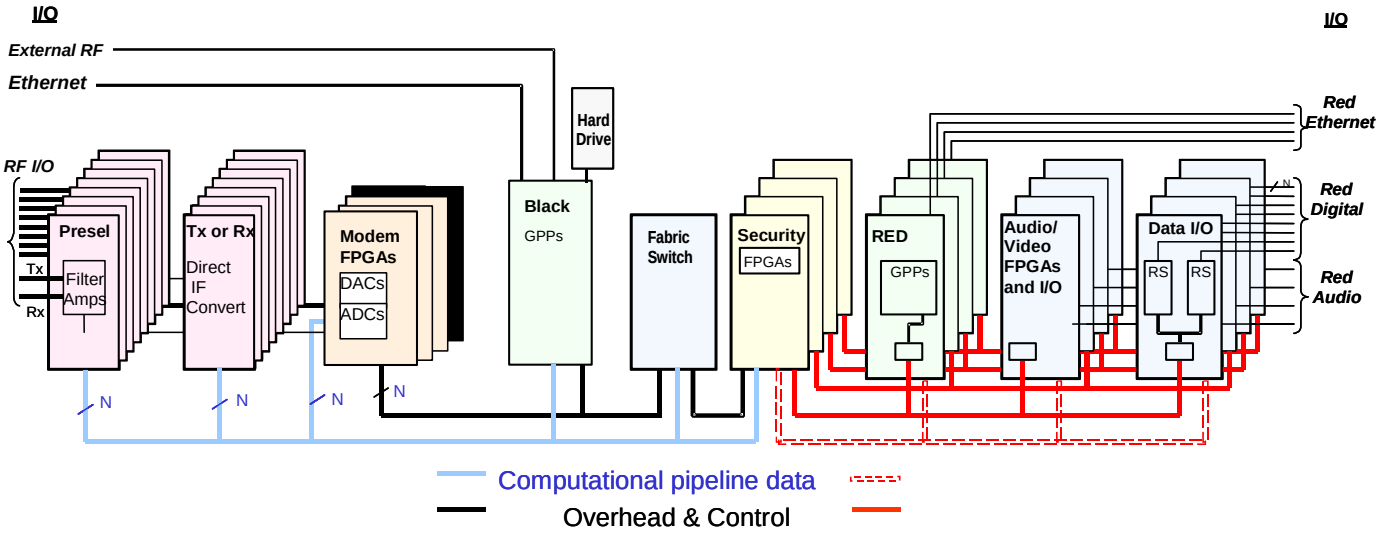

Figure 1. FPGA Based System Architecture.

	Modem Section (Black Side)			Security Section (Red Side)		Audio/Video Section (Red Side)		Networking (Red Side)
WAVEFORM FUNCTIONS	**Tx FPGA**	**Rx FPGA**	**GPP (Black)**	**Tx/Rx FPGA**	**GPP (Red)**	**Tx/Rx FPGA**	**GPP (Red)**	**GPP (Red)**
	Pulse Shaping	Matched Filtering	Transec/ Comsec Key Manage-ment	Comsec Encoder/ Decoder	Control Inter-face	Vocoder	Control Inter-face	Internet Protocol Stack
	Spreading	De-Spreading	Mode Switching control	Transec Variable Generator		Audio Filter	Circuit Routing	Ad-hoc Networking Protocol
	Baseband Modulation	Baseband Demodulation	Presets			Virtual Serial I/O		Wireless IP Protocol Stack
	IF Upconvert/ Filtering	IF Downconvert/ Filtering	Human Machine Interface			Rate Converter		
	Tx Hop Timing/ Synchronization	Rx Hop Timing/ Synchronization	Circuit Routing			Video Compression		
	OFDM Tx Timing	OFDM Rx Timing				Speech Recognition		
	FEC Encoder	FEC Decoder				Image Processing		
	Interleaving	Denterleaving				User Identification		
	Tx Hop Controller/ Waveform Clocks	Rx Hop Controller/ Waveform Clocks				Virtual CODEC		
	MIMO	Channelization				Audio AGC		
	Synthesizer	Beam forming						
		Rake Rx						

Table 1. Partitioning of FPGA Based Waveform Functions Across the Platform Architecture

3. FPGA WAVEFORM PARTITIONING AND INTERFACING

One of the first steps in the waveform design process is to partition the various waveform functions across the platform. Since we have chosen to use only FPGAs and GPPs, this partitioning is made easier. Table 1 contains a listing of commonly used waveform functions divided into each section of the FPGA based platform.

Some partitioning trade-offs exist between Modem FPGA and Audio/Video FPGA depending if security is needed. For non-secure waveform modes, audio filtering and conditioning may be done in the modem section closer to the modulation/demodulation functions, or it could be performed in the audio/video section closer to the I/O source/sink. Another area where partitioning trade-offs exist is between RF and Modem sections. Such functions as RF AGC and selectivity filtering can be allocated in either domain depending on the amount of digital processing.

Interfacing the different platform sections can be done in a variety of ways depending on the data speed and flexibility requirements of the platform. In some cases, it may be required to have each modem FPGA have full access to each RF section through the use of a fabric switch [1]. Dedicated or multiplexed direct hardwire connections can also be used to connect FPGAs with other hardware devices.

A key difference between DSP processor (serial) based platforms and FPGA (parallel) based platforms lies in the routing of data. For DSP processor platforms the data must be packetized and the exact time of each data sample is not known unless costly time-stamping is applied. This idea is contrary to the concept of FPGA based waveforms. Ideally, the data processed by the FPGA can be modeled sufficiently such that transfer between FPGAs is well defined and synchronized on a sample-by-sample basis.

One advantage of FPGA based processing is that co-location of speed intensive functions such as FFTs may be multiplexed or shared in the same FPGA. This dramatically saves resources compared with running several FFTs in different processors and having to communicate between them. Let us extend this philosophy a bit further by suggesting that other memoryless waveform functions such as certain modulators/demodulators can be multiplexed within a given FPGA. Functions with memory could also be multiplexed such as FIR filters but some additional resources would be needed to store coefficients and tap values.

4. MULTI-MODE AND MULTI-WAVEFORM FPGA DESIGN

Many current and future waveforms have multiple modes of operation. These modes include combinations of the following: plain text (PT), cipher text (CT), single-channel (SC), frequency-hopped (FH), Voice, Data, TDMA, OFDM, FEC, DS spread spectrum, and various modulation types.

Multi-Mode waveforms require that a single instance of the waveform application be able to switch between various modes upon user command within several hundred mili-seconds in most cases. To accomplish this within an FPGA(s), either serial or parallel implementations may be considered. The most straightforward approach is the parallel Multi-Mode Waveform Model shown in Figure 2. Here all of the functions needed for the various modes are designed and implemented at the same time in the FPGA. When a different mode is selected, certain functions are bypassed and others brought in-line. The advantage to this approach is that the design cycle is short. The disadvantage is that more FPGA resources are consumed than are needed for any given one mode. An alternative to the parallel approach is the serial multiplexed scheme. This method is depicted in Figure 3. Here a core library of waveform functions is designed and placed in a repository on the FPGA. A core controller is then used to multiplex the functions between different waveform application cores or modes. In effect, the core library can be thought of as a group of subroutines that get called whenever a waveform application needs the service. The core controller management complexity is a disadvantage of this scheme however; the savings in resources may outweigh this cost.

Multi-Waveform FPGA design is a challenging area. The driving requirements in this area are the number of simultaneous waveform applications per FPGA and the ability to “Swap” applications on the fly. Three approaches will be described for handling these requirements: 1.) Dynamic Reconfiguration, 2.) Static Partitioned Waveforms, and 3.) Shared Core Library

The first approach deals with Dynamic Reconfiguration, which is also known as dynamic partial reconfiguration. This off-the-shelf approach allows swapping applications in and out of the same FPGA without disturbing any applications that are already running. Details are specified in [2] for implementing Dynamic Reconfiguration however, not many FPGA developers are reported to have used this method and it may take more users for this to be a useful technique of multi-waveform design. Dynamic Reconfiguration does not affect nor can it be influenced by, the FPGA design flow since it is a manipulation of the compiled source code.

The second multi-waveform approach is that of Static Partitioned Waveforms. This is shown in Figure 4. Here, multiple waveform cores are designed into a single FPGA and run simultaneously, however, they cannot be swapped out unless each application has its own FPGA.

The third approach to multi-waveform design uses a shared core library that is also useful for multi-mode waveforms within a multi-waveform FPGA. This was introduced in Figure 3 and is elaborated on in Figure 5 for the multi-waveform scenarios.

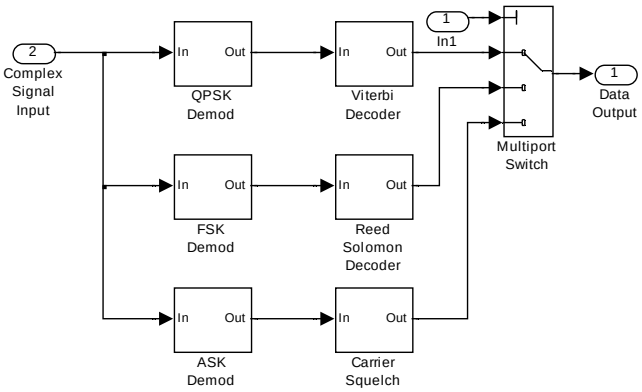

Figure 2. Parallel Multi-Mode Waveform Model.

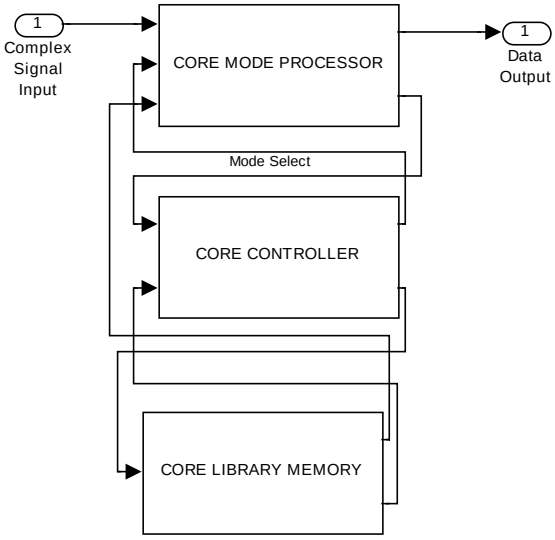

Figure 3. Serial Multiplexed Multi-Mode Waveform Model.

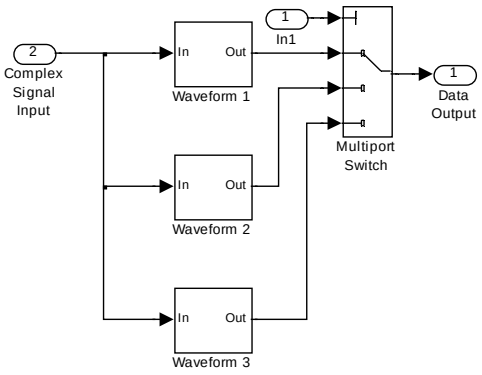

Figure 4. Parallel Multi-Waveform Model.

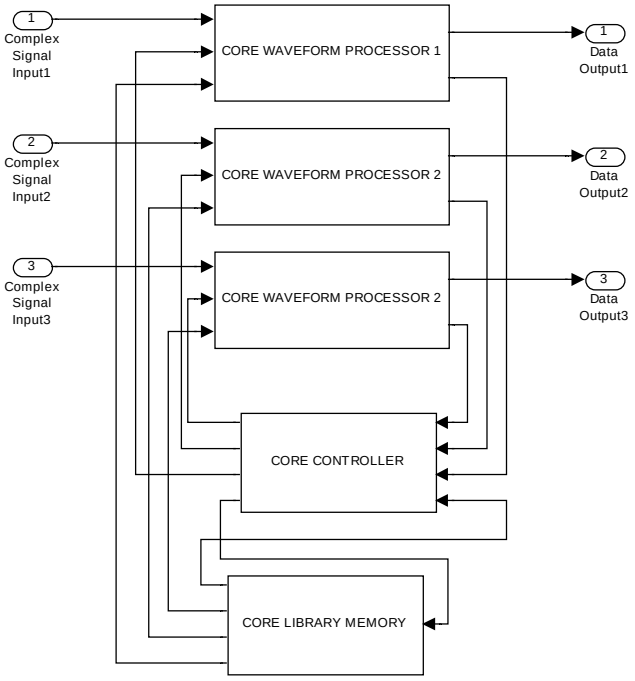

Figure 5. Serial Multiplexed Multi-Waveform Model.

5. FPGA WAVEFORM DESIGN FLOW

New FPGA design flows [3] based on high-level, parallel, simulation environments offer code synthesis directly from a block diagram systems model. At the center of these new design flows are tools that allow access to hardware interfaces and abstract the FPGA fabric to a systems level. This gives the systems designer direct insight into the platform for which he is designing a particular function. This is made possible by the bit true and cycle true abilities of these tools. One problem with traditional design flows is that the system designer does not have insight into the implementation details of the fabric and therefore cannot best optimize the system design without lengthy interaction and written communication with implementation engineers who use low-level tools such as VHDL, Verilog, or schematic capture. An example of an FPGA based waveform done using the Xilinx System Generator in Matlab Simulink is shown in Figure 6 [4].

In addition, to direct code synthesis from the systems model, the model can also serve as a systems documentation vehicle. Some tools such as Matlab Simulink support document generation from the model itself and this can be coupled with other tools such as Requisite Pro and Rational Rose to form a complete system definition and code producing model. At this time there is not a definitive link between these tools. One feature that could be added would be a way to automatically verify system requirements with model-generated data. For example, the system requirement may call for a certain Bit Error Rate (BER) at a given input power to the system. When the model is run, data from the BER calculation block could be mapped to a Requisite Pro requirement to mark it verified at the system design level. Other requirements that may be mapped are latency, attack/release times, acquisition times, and distortion.

Another valuable feature of Matlab Simulink is the ability to do co-simulation with the target FPGA while running other Simulink components. This process involves compiling the FPGA code for the target and placing a co-simulation block in model. Then, data from the model can be routed in and out of the real-time operating block. The co-simulation block could also be part of an actual system with RF and ADC/DAC interfaces so as to allow very accurate simulation at a high level of abstraction.

6. FPGA WAVEFORM INTEGRATION FLOW

Once an FPGA based waveform has been designed, simulated, synthesized, and place/routed, the integration of the bit file with other FPGA bit files and GPP software must be done to realize the fully functioning waveform. Since most of the critical real-time functions were simulated and proven during the design flow, the risk and duration of waveform integration is greatly reduced.

One advantage of using a highly accurate model to synthesize the FPGA code is that test vectors can be taken from the system under integration to compare with the model. Also, test vectors from the model may be used in DSP signal generating equipment to provide a stimulus of known data. If problems are found during integration the model must be updated and the FPGA bit file must be regenerated. This is the only disadvantage to this approach compared with DSP processor based waveforms in which coded changes can be recompiled in a shorter time. However, the risk of code errors is greatly reduced by using the FPGA design flow with accurate system models.

7. FPGA WAVEFORM QUANTIZATION

During the FPGA waveform design flow, it is best to first create a floating point model using the FPGA Simulink blockset with the “Override with Doubles” option turned on. This can be done at the block primitive level or globally for a system or subsystem in Simulink to allow simulation of a system with 64 bit floating point precision.

Once confidence is gained with the floating point model, the quantization process should begin. Each block in the model must be analyzed for fixed-point properties such as Number of Bits, Binary Point Position, Arithmetic type (Unsigned or Signed), Quantization Behavior (Round or Truncate), and Overflow Behavior (Wrap or Saturate). In addition, any filter coeficients must also be analyzed for fixed piont behavior.

Fixed point filter design can be done using the Matlab Filter Design and Analysis Tool (fdatool) by entering the “Set Quantization Parameters” window and enabling “Turn quantization on”. In this window all of the filter arithmetic and filter coeficients can be quantized and a graph of both floating and fixed point responses are shown in the window above. By adjusting the fixed-point properties of the filter, one can visually approximate the floating point behavior for such responses as Magnitude, Phase, Group Delay, Impulse Response, Step Response, or Pole/Zero configuration.

One feature that could be added to these tools is a Processing Gain Analyzer to estimate scaling and bit sizes. To do this manually involves tedious effort and is a disadvantage of the FPGA waveform approach. For example, a digital phase-locked loop contains feedback with a loop filter that can be very sensitive to dynamic range. When attempting to quantize a recursive circuit such as this, one must pay careful attention to the growth of registers and apply scaling after each multiply

operation. The trade-off here is that input dynamic range will dictate the maximum "Number of Bits" and "Binary Point Position". However, since there is feedback in the loop, the input to the loop filter can change its output which inturn mixes with loop input to infuence the loop filter input. Bounds must be estimated at each stage and at each arithmetic calculation point to control the register growth or quantization error.

Overall, the goal of Quantization is to approximate floating point operation with a minimum amount of FPGA resources while meeting system performance requirments.

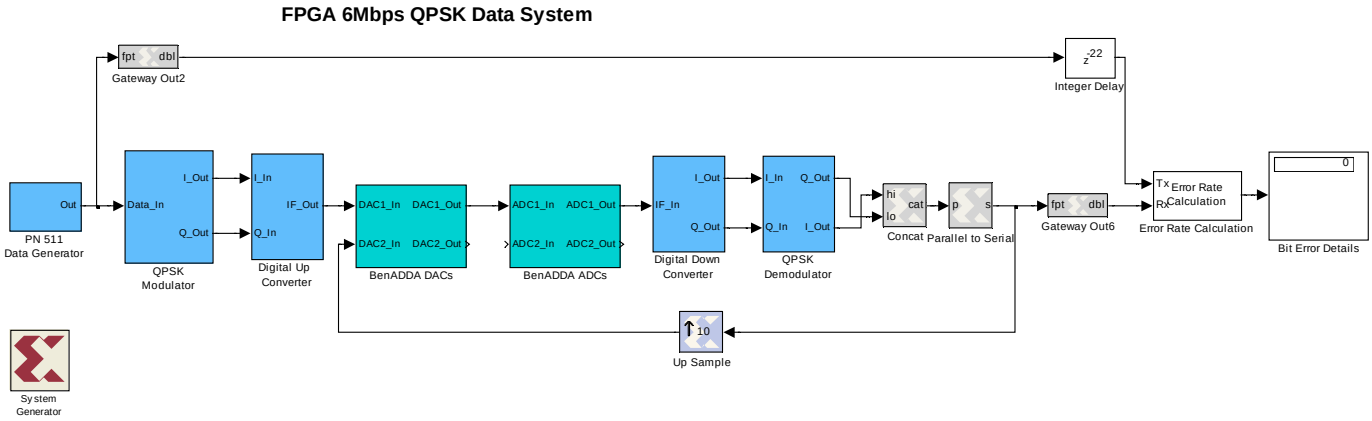

Figure 6. FPGA Based Waveform Model using Xilinx/Matlab Simulink Design Flow.

8. CONCLUSION

In this paper, concepts and techniques were given for FPGA based waveform design. An FPGA based system architecture and mapping of waveform functions was shown, suitable for future military JTRS type communications applications. Both Muti-Mode and Multi-Waveform applications were discussed in relation to FPGA based systems. An efficient design flow was overviewed along with a corresponding integration flow. Techniques for FPGA waveform quantization were given as well.

As DSP computing technology migrates from the serial, instruction set architecture, von Neumann machine to the parallel, system based design flow, FPGA based systems and waveforms will be an enabling technology for higher speed and more complex communication systems. The end result will be that waveforms can be developed faster with more features.

9. REFERENCES

[1] D. Bouvier, "RapidIO, The Embedded System Interconnect", RapidIO Trade Association, 2003.

[2] Xilinx Application Note XAPP290, *Two Flows for Partial Reconfiguration: Module Based or Small bit Manipulations*, New York, May 17, 2002.

[3] Xilinx System Generator for DSP v3.1 Reference Guide.

[4] C. H. Dick, f. j. harris, and M. Rice, "Maximum likelihood carrier phase syncronization in FPGA-Based software defined radios", Acoustics, Speech, and Signal Processing, 2001. Proceedings. (ICASSP '01). 2001 IEEE International Conference on , Volume: 2 , 7-11 May 2001 Page(s): 889 -892 vol.2.