DESIGN AND TESTING OF SPACE TELEMETRY SCA WAVEFORM

Dale J. Mortensen[1] (ZIN Technologies, Inc., Brook Park, Ohio, USA; dale.mortensen@zin-tech.com);
Louis M. Handler (NASA Glenn Research Center, Cleveland, Ohio, USA;
Louis.M.Handler@nasa.gov); and
Todd M. Quinn[1] (ZIN Technologies; todd.quinn@zin-tech.com)

ABSTRACT

A Software Communications Architecture (SCA) Waveform for space telemetry is being developed at the NASA Glenn Research Center. The space telemetry waveform is implemented in a laboratory testbed consisting of general purpose processors, FPGAs, ADCs, and DACs. The radio hardware is integrated with an SCA Core Framework and other software development tools. The waveform design is described from both the bottom-up signal processing and top-down software component perspectives. Simulations and model-based design techniques used for signal processing subsystems are presented. Testing with legacy hardware-based modems verifies proper design implementation and dynamic waveform operations.

The waveform development is part of an effort by NASA to define an open architecture for space based reconfigurable transceivers. Use of the SCA as a reference has increased understanding of software defined radio architectures. However, since space requirements put a premium on size, mass, and power, the SCA may be impractical for today's space ready technology. Specific requirements for an SCA waveform and other lessons learned from this development are discussed.

1. INTRODUCTION

The Space Telecommunication Radio System (STRS) project team at the NASA Glenn Research Center is currently studying the Software Communications Architecture (SCA) to support the design effort of an open architecture for software defined radios in the space environment. In order to better understand the application of such an architecture to space-based radios, the STRS waveform development team is currently working on a prototype SCA waveform that mirrors the functional characteristics of current NASA space telemetry [1]. The waveform's basic characteristics are QPSK modulation, ½ rate Viterbi coding, and 1 Mbps user data throughput.

An SDR-3000 development platform, part of the testing and validation laboratory at NASA Glenn, was utilized for the waveform development. The platform consists of a number of PowerPC multipurpose processors, field programmable gate arrays (FPGAs), digital-to-analog converters (DACs), analog-to-digital converters (ADCs), a real-time operating system, the Harris SCA core framework, and a communication board support package. This platform was used to transmit and receive signals to other commercial satellite modems at 70 MHz intermediate frequency (IF) for testing and validation purposes.

Both a bottom-up and top-down design approach was implemented, as described in the next section. Testing and validation methods and results are described in section 3. To conclude, a discussion of implications for space-based radio applications is in section 4. Lessons learned are included throughout these sections.

2. DESIGN & IMPLEMENTATION

Knowing where to begin the development was a significant challenge, even with a basic understanding of the SCA. Developing this SCA waveform required experience in several different areas, such as middleware, object oriented embedded programming, FPGA design, digital signal processing, not to mention space communications. On occasion industry software engineers were consulted to supplement the waveform team's experience and knowledge. Specifically, during the course of this effort, the team acquired knowledge in the following areas:

- Use of the software development and monitor tools accompanying the core framework.
- Use of the interface definition language (IDL) to define various interfaces for the components of a waveform.
- The SCA domain profile specification and how to deploy and configure various parts of the waveform.

[1] Work performed under contract NAS3-99154.

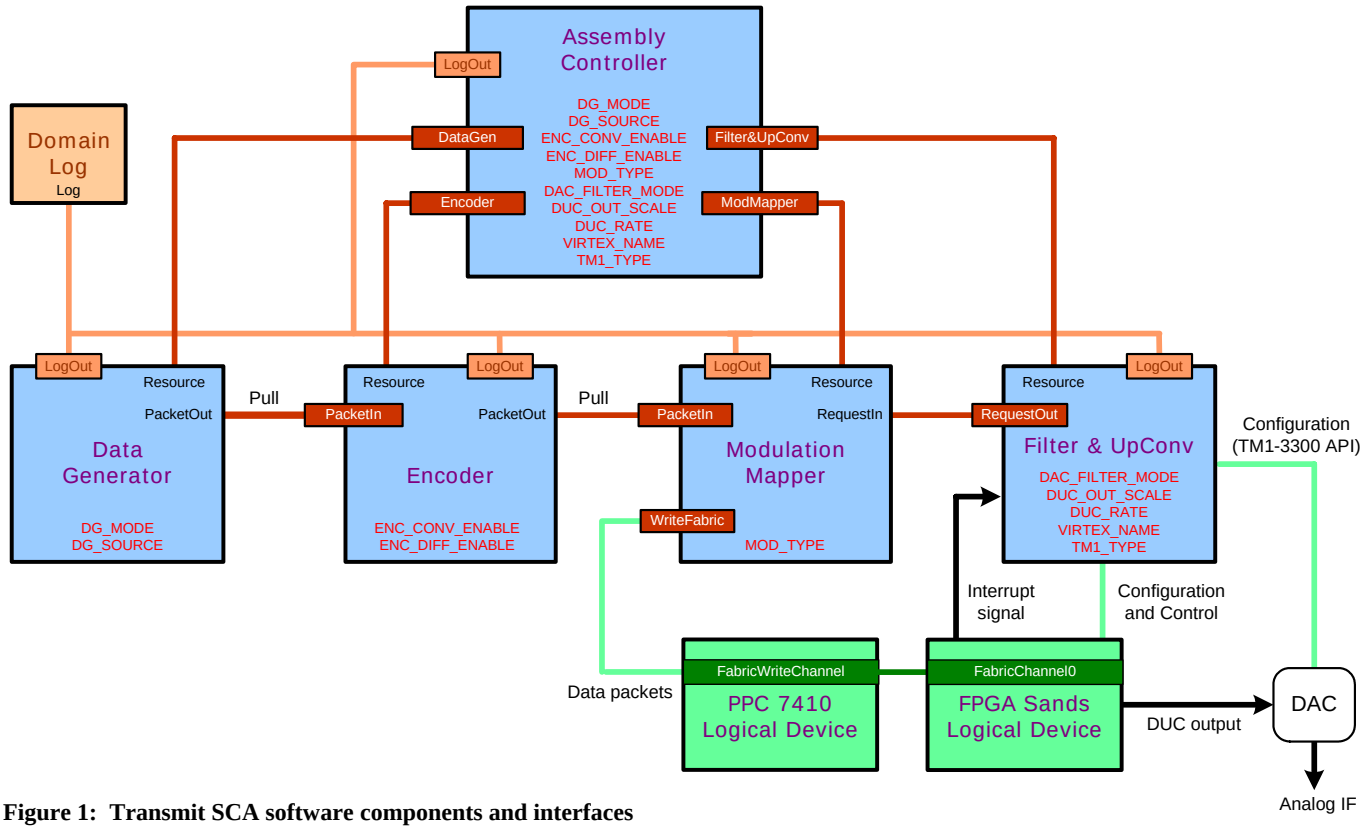

Figure 1: Transmit SCA software components and interfaces

- How the core framework uses CORBA and how CORBA applies to the waveform.
- The process path needed for developing the various components of the waveform and connecting them together.

The waveform development process followed can be summarized in the following steps [2]:

1. Identify the functionality that comprises the operation of the waveform.
2. Identify the interfaces between the components
3. Create and compile IDL for the waveform specific interfaces (e.g. PullPacket)
4. Write CORBA Servant code
5. Create XML
6. Test and debug

In parallel, the various waveform digital signal processing functions, such as the modulation mapping, were tested and debugged in a non-SCA waveform. Then these functions were integrated with the corresponding SCA waveform software component.

The initial development tasks focused on how the waveform is managed by the SCA core framework, and how the various sections of the SCA handle deployment and operation of the waveform. The SCA core framework provides a Domain Manager, Application Factory and Application entities for deployment and control of the waveform. The waveform developer only needs to concentrate on a set of basic application interfaces such as Port, PropertySet, Resource and others as described in the SCA specification. Waveform components are developed with these base application interfaces and interact with the SCA deployment and control mechanism through information provided in the Software Assembly Descriptor XML file, and other supporting XML files [3].

Identifying the functionality of the various components that would comprise the transmit portion of the space telemetry waveform produced the following four software components, (as shown in Figure 1):

1. Data Generator – produces internally generated data patterns, and provides an interface with external data sources.
2. Encoder – convolutionally at ½ rate and differentially encodes data.
3. Modulation Mapper – converts the binary data to modulation symbol samples.
4. Filter & UpConv – performs pulse shaped filtering and digital up conversion.

To deploy these components within the SCA core framework, an additional component called the Assembly Controller is needed. The SCA specification requires that

all external configuration, control and query requests are relayed by the core framework and processed by the Assembly Controller. For example, as shown in Figure 1, the Data Generator is a resource component which has properties DG_MODE and DG_SOURCE. These parameters can be set by a user via an external interface that communicates through the core framework domain manager. The domain manager passes the information along to the Assembly Controller. The Assembly Controller has port connections to the various components to relay the property values to the proper destinations.

The Assembly Controller, the Data Generator, the Encoder and the Modulation Mapper are components that are to be deployed on general purpose PowerPC processors. The SCA specification requires that these components communicate using CORBA. To minimize communication delays among distributed objects in the CORBA environment, the four components were collocated on the same processor. Connections between the components are achieved by specifying SCA ports on each component.

A PullPacket interface is defined in IDL to encapsulate the transfer of a data packet using CORBA. The PullPacket interface is used by the Modulation Mapper to transfer packetized data from the Encoder. The Encoder, in turn transfers data from the Data Generator with the same type of interface. In IDL, a PullPacketInterface is defined with a function called pullPacket. An IDL compiler for C++ is used to create the code to support the PullPacketInterface within the CORBA communication environment. The waveform components that support the PullPacketInterface must implement a pullPacket function.

The pullPacket function in the Data Generator creates a packet of data based upon the current DG_MODE setting. The data packet is passed back to the Encoder which adds its encoding and then passes the data packet back to the Modulation Mapper to prepare it for further processing. In a similar fashion, the interface between the Filter & UpConv requests a data packet by using a different CORBA interface called RequestOut.

Up to this point the waveform components fit nicely within the SCA core framework because they are to be deployed on general purpose processors (GPP). The filter and up converter functions however are deployed and executed inside an FPGA for performance reasons. This currently requires a SCA component, shown as the Filter & UpConv block of Figure 1. This represents the control part of the filter and up converter function, and resides on a GPP with a direct connection to the FPGA. The hardware platform on which the waveform is deployed has a board support package with various SCA logical devices which allow specialized hardware to operate within the core framework. The development platform uses flexFabric (platform specific RapidIO switched fabric) to quickly move data between various processors. The control portion of the Filter & UpConv can receive parameter control information from the Assembly Controller and configure the FPGA appropriately. Also, digital signal data packets from the Modulation Mapper can be directly sent over a flexFabric communication channel via a SCA port connection.

The WriteFabric interface between the Modulation Mapper and the filter & up converter functions on the FPGA uses a special mechanism to take advantage of the flexFabric interface to send data to the FPGA without CORBA. This is important since the Modulation Mapper and the FPGA are on different physical processor boards and the CORBA communication delays via Ethernet would be too long for the waveform to function as it's currently designed at a data rate of 1 Mbps.

A special association is needed to use the flexFabric to send data from the GPP to the FPGA. There is an indirect connection made to a proxy allowing the WriteFabric port on the Modulation Mapper to obtain a handle from the core

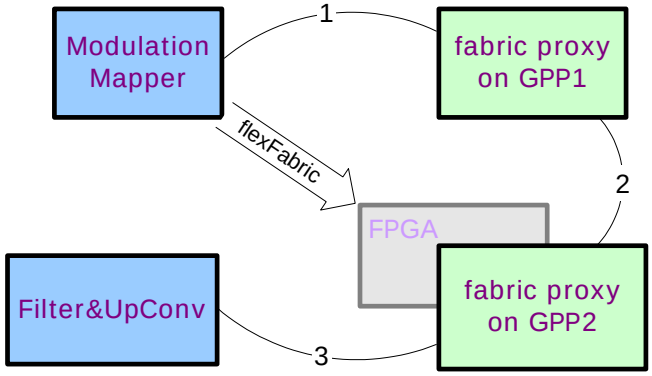

Figure 2: XML connections to FPGA

framework, as in item 1 below. This handle is used to access the flexFabric to write data to the FPGA. The implementation requires three XML connections in the software assembly descriptor (SAD) file, as Figure 2 illustrates:

1. From the GPP1 module (Modulation Mapper) to fabric proxy on the same device.
2. From fabric proxy on GPP1 to fabric proxy on GPP2, the device with a direct connection to the FPGA.
3. From GPP2 module (Filter & UpConv) to fabric proxy on the same device. This connection is a placeholder to complete the connection, but the handle in the Filter & UpConv is not usable by the waveform.

The bottom-up design approach focused on developing the waveform functions independent of the SCA, yet

cognizant of the waveform's top-level module boundaries and interfaces. For example, GPP code was written for the data generator function that was independent of the encoding and mapping routines instead of being highly integrated. Likewise the FPGA code was written with SCA control delays in mind, in terms of buffering data to deal with relatively lengthy CORBA calls.

A model-based design approach was employed with the FPGA circuit development. Simulations of the digital up converter allowed parameters to be set properly for the given waveform specifications before testing on the hardware. VHDL code was auto-generated from the working simulations, and then brought into the FPGA synthesis CAD tool. The platform provided FPGA wrapper VHDL code was integrated with the application code. Figure 3 shows a functional block diagram of the platform FPGA wrapper with the transmit waveform functions. The block labeled "DUC" contains the auto-generated code from the simulation model.

3. TESTING & VALIDATION

Testing was focused to learn whether SCA waveforms can be used for space applications. Although the SCA start and stop methods were designed for normal use, testing and debugging was accomplished more efficiently using the runTest method. This allows a variety of tests to be invoked without changing the waveform.

The SCA components in the waveform inherit from the SCA CF::Resource interface which inherits the runTest method from the TestableObject interface. The runTest method was implemented in the SCA components to test passing data between components. The data in the XML preferences was used to control what data was sent for those components tested. A large value for the property NTIMES was entered to repeat the test for the corresponding number of packets where each packet was 4096 bytes long. Timing

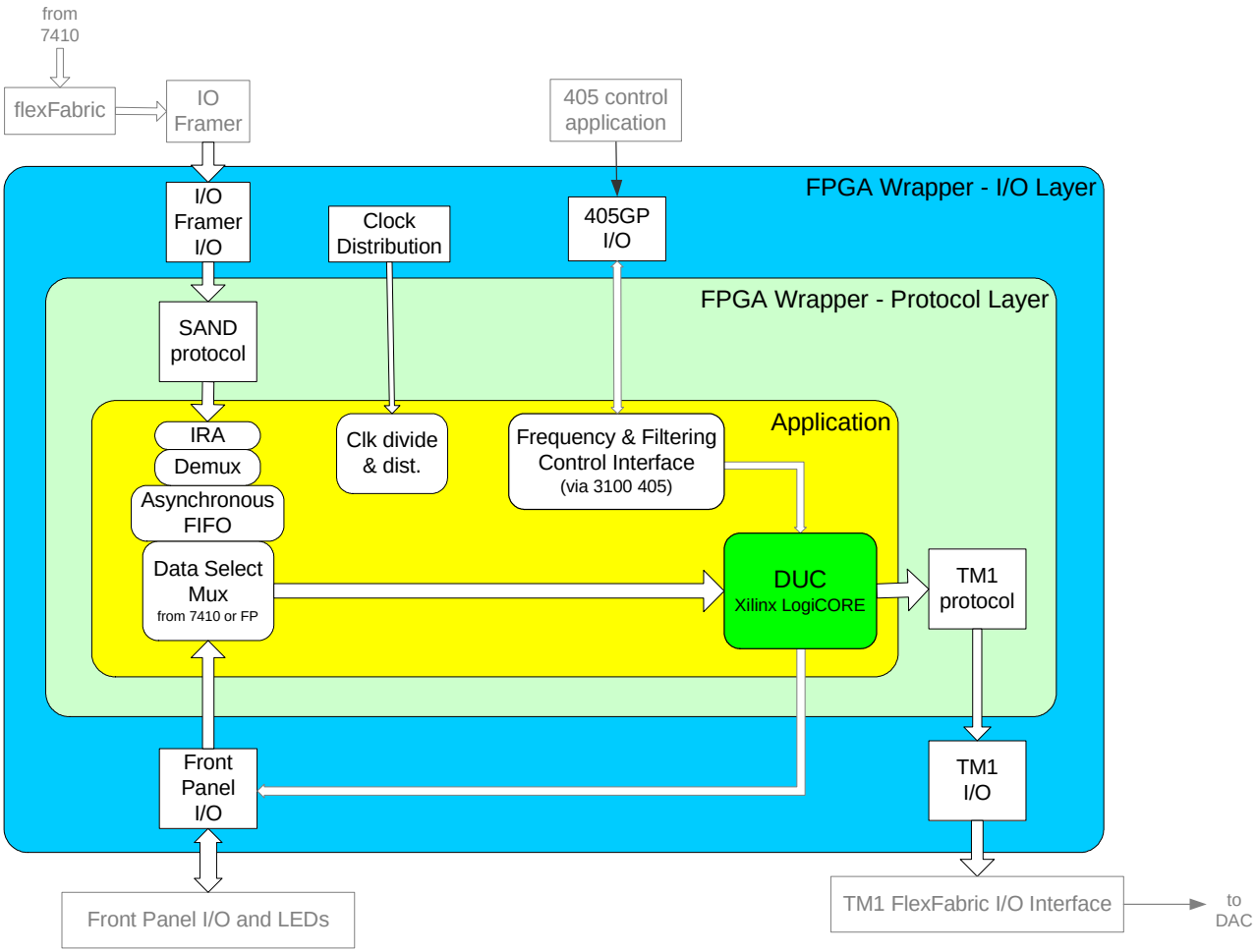

Figure 3: FPGA functions and wrapper integration

was kept and the lapse time was computed for the pertinent tests. Thus, debugging of different portions of the waveform's functionality was possible by changing property values with the user interface.

Referring to Figure 1, a test of the Modulation Mapper pulling a packet from the Encoder which pulls the packet from the DataGenerator and sends the packet over the flexFabric to the FPGA was performed. It took 2.05 milliseconds to send each packet. The results for the test of the Filter & UpConv requesting a packet from the Modulation Mapper which pulls the packet from the Encoder which pulls the packet from the DataGenerator and sends the packet over the flexFabric to the FPGA was 3.09 milliseconds per packet. The difference of about 1 millisecond is the time to send a request from the Filter & UpConv to the Modulation Mapper. This relatively significant delay is due to using CORBA between different boards in the SDR. There will be more about the implications of this in the next section.

A challenge in waveform testing and debugging was the time it takes to make a simple change before it can be debugged. The process of making a code change, recompiling, rebooting the hardware, loading the core framework, and starting the user interface usually takes at least 15 minutes. This time delay makes the debug process cumbersome and inefficient by today's standards. Additional challenges encountered were timeout errors, insufficient error messages from the core framework, system hang ups, and limited documentation.

Several COTS legacy hardware modems were used in the validation testing of the waveform. Some of these modem specifications are proprietary, such as the synchronization technique and forward error correction details, so interoperability with this equipment became a challenge. A few of the original waveform specifications needed to be changed along the way as the testing revealed some of the differences with the legacy modems. In particular the addition of differential encoding became necessary to allow phase ambiguity resolution in the commercial receivers. The original waveform design was a unique word method of synchronization, but this was not possible given the proprietary nature of the COTS modems.

The transmit waveform has been successfully tested with legacy modem receivers using pseudo random bit sequence data and differential encoding. Additive white Gaussian noise was added at the 70 MHz IF yielding the BER performance plotted in Figure 4. Some degradation from theoretical for differentially encoded QPSK is observed [4]. This is due in part to the unmatched pulse-shaped filtering between the transmit waveform and the commercial receiver. The proprietary nature of the legacy modem receivers makes matching the filter difficult.

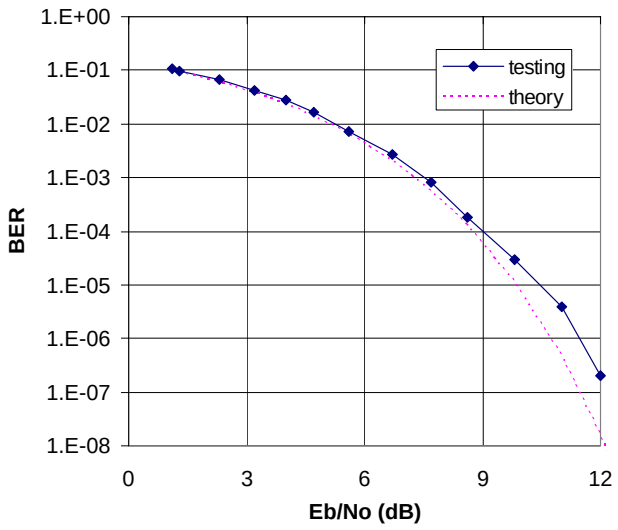

Figure 4: BER performance with legacy modem

4. IMPLICATIONS FOR SPACE BASED SDR

Certain aspects of the SCA are important when considering deployment, especially those that relate to size, weight, and power for a space-based radio. Development of this space telemetry waveform has brought forth issues regarding FPGAs, memory, and waveform file system complexity.

This development effort intentionally placed as many waveform functions as possible in the GPP [1]. The FPGA was used for remaining functions that would not meet data rate performance in the GPP. In actual space radio applications FPGAs are favored over GPPs because of performance and power efficiency. Optimization is key for limited resource space-based radios. The model-based design approach offers portability but is not yet optimal from a performance standpoint. A standard FPGA wrapper would help with reusability and portability of optimized code. Since there is no standard FPGA wrapper, there is a porting challenge for each new radio, having a different FPGA implementation. Unfortunately, the SCA does not currently address FPGA application interfaces adequately. Although there is on going work in this area, it is recommended that industry and the standards bodies increase their efforts.

The bottom-up waveform development approach produced a non-SCA waveform, which allows an interesting comparison with the SCA waveform in terms of resources. The non-SCA waveform is a combination of all the data flow functions, such as encoders and upconverters, running without any SCA infrastructure.

As an estimate of the effort involved and resources required, the SCA waveform consisted of 69 files: 22 ".h" files, 25 ".cpp" files, 18 XML files, and 4 IDL files, whereas the non-SCA waveform consisted of 15 ".c" files.

The previous SCA waveform file count does not include 44 additional files, 11 generated for each IDL file. The SCA and non-SCA waveforms each consist of about 11600 lines of code. Although these numbers of lines appear to be similar, the SCA waveform is much more complex. It contains implementations of the CF::Resource interface methods, and CORBA for data transfer, whereas the non-SCA waveform contains extra test programs required for bottom-up testing.

In terms of memory footprint, there are significant differences for the SCA and non-SCA waveforms. The SCA waveform consisted of 6.3 MB generated in 7 "*.out" files whereas the non-SCA waveform consisted of 0.5 MB generated in 2 ".out" files. The SCA waveform is almost 13 times as large as the non-SCA waveform, even before the core framework and CORBA are included in the SCA environment. On the test platform's GPPs, the core framework took over 35 MB of memory, which includes 6 MB for the XML parser alone. The XML files are used for dynamic deployment, which may not be necessary on a space-base radio due to the static nature of the mission requirements. In addition, the ACE/TAO ORB took about 12 MB. Although there are other much smaller ORBs available, the core framework and ORB would still consume a significant proportion of the required resources. Current reconfigurable space radios have only about 2 MB of memory to do everything, including the operating system. The processing power in terms of GPP speed and FPGA gates is also at a premium, so it would be difficult to fly such an SCA waveform on space transceivers in the near term. However, a viable "light weight" version of the SCA may enable the SCA to fly on future missions.

In the meantime, NASA is developing an open architecture radio infrastructure that parallels the SCA in many aspects but is small enough to fly on near-term missions. Tradeoffs with the flexibility the SCA offers and the constraints of the space-based radios are a major part of the architecture design. The SCA space telemetry waveform effort reported on in this paper has enabled the NASA architecture team to understand and assess the use of the SCA for space. Many subtle aspects were only discovered through this hands-on development. Future plans involve a port of the SCA space telemetry waveform to the new NASA software radio infrastructure as one of the first test cases.

5. ACKNOWLEDGEMENTS

The authors would like to especially thank the following NASA Glenn STRS waveform team members for their contributions to this effort: Daniel Oldham, Thomas Bizon, and Jeffrey Glass.

6. REFERENCES

[1] D.J. Mortensen, M. Kifle, C.S. Hall, T.M. Quinn, "SCA Waveform Development For Space Telemetry", SDR Forum Technical Conference, November 2004.

[2] A. Gonzalez, R. Hess, "JTRS SCA Developer's Guide", Raytheon for JTRS JPO, June 2002.

[3] JTRS-5000SCA, Appendix D, rev 2.2.

[4] B. Sklar, *Digital Communications* 2nd ed., Prentice Hall,UpperSaddle River, NJ, 2001.