REAL-TIME, SOFTWARE-BASED CHARACTERIZATION OF RECEIVER DYNAMIC CHANNEL PERFORMANCE ON AN SDR DEVELOPMENT PLATFORM

Brian A. Dalio (Coherent Logix, Inc., Austin, TX, USA, dalio@coherentlogix.com);
Ivan Aguayo (Coherent Logix, Inc., Austin, TX, USA, aguayo@coherentlogix.com)

ABSTRACT

We present the design and implementation on a hardware development platform of a real-time, software-based analysis and characterization environment for evaluating the performance of software defined radio (SDR) receivers in the presence of dynamic channel conditions as well as additive white Gaussian noise (AWGN).

Accumulating the data to accurately determine the BER / PER performance of a receiver during the design exploration phase can require a farm of simulation servers and years of CPU time. This new characterization and analysis environment supports the characterization of a software model of the receiver in real-time. It sends test parameters (e.g., SNR, message size) to the development platform where separate long-period uniform random number generators (URNGs) are used to ensure uniqueness of test messages and channel conditions. In the development system, test messages are automatically generated, transmitted through a multipath, dynamic fading channel emulator, and presented to the receiver for decoding. The decoded results are compared to the original message and relevant error statistics collected. All operations are under the control of an interactive host system program. Since the system runs in real-time, results can be accumulated as fast as the waveform can operate. The environment has been implemented on a software-based integrated radio waveform development system and used to characterize the performance of multiple receiver configurations. We present details of an example characterization, system performance, and overall resource usage.

1. INTRODUCTION

Fast, efficient, and flexible receiver performance characterization was required to support SDR waveform development work carried out by our team. To this end, we developed a 100% software based channel emulator, as described in [1]. While this channel emulator had the necessary features and performance to meet our needs, its configuration and management were more complex than should be imposed on (non-expert) users. Further, there was no particular support to carry out automatic characterization. We therefore undertook the design and implementation of a characterization environment to encapsulate and automate the use of the channel emulator.

Alternative approaches such as simulation in a modeling environment (e.g., The MathWorks' MATLAB / Simulink [2]) or using dedicated hardware (e.g., Spirent's SR5500 Wireless Channel Emulator [3], Elektrobit's EB Propsim F8 RF Channel Emulator [4]) were evaluated. The purely software approach provided great flexibility and visibility of process and results, but was orders of magnitude too slow for use beyond the conceptual level. The dedicated hardware solutions were capable of real-time use, but at the cost of modeling flexibility.

Our approach was to develop our own characterization environment making use of our previous channel emulation work and building on the success of our Radio Waveform Development System [5]. This system is built on the hxISDE / hxHADS development environment and massively parallel HyperX processor architecture described in [6].

We first present the overall system design for our characterization environment, including details of the components, their functions, and resource use. Details of both the simplified, initial version of the environment and the follow-on complete version are given. The environment's GUI is described along with the user's control over test scenario parameters. We then present results from an example use of the environment. Finally, a summary of the work and a description of our on-going efforts are given.

2. SYSTEM DESIGN AND IMPLEMENTATION

In the following subsections we present the overall design of the characterization environment, its design goals and their rationales, the partitioning of the environment between the host PC environment and the hxHADS development environment, the individual components of the environment, and the structure, processing, and GUI of the host PC software tool.

2.1. Overall System Design

Our design goals (and rationales) for this environment include the following:

- 100% implemented in software: By implementing the characterization environment completely in software, we support rapid development, easy reconfigurability, and full visibility of all operations.
- Usable from early in the development process: The earlier in the design process that performance can be characterized, the more time that will be available to

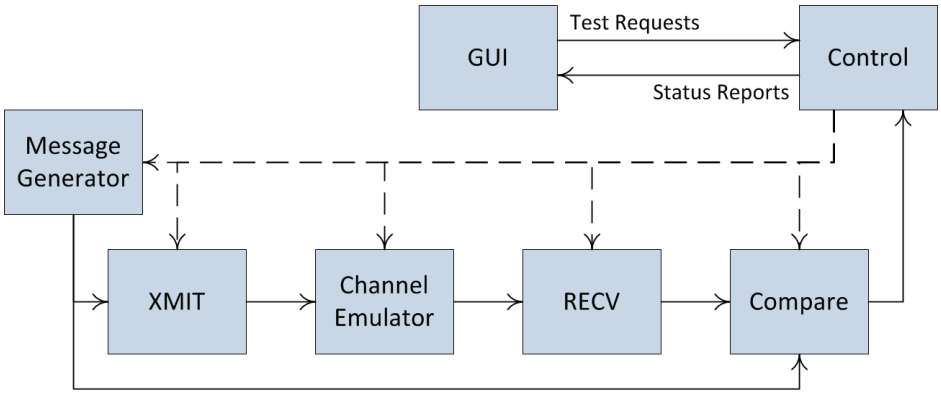

Figure 1. System Design

explore design alternatives. Ideally, performance should be characterized throughout the design refinement process from the abstract functional level through to the final implementation level.

- Fully user visible and programmable interface: By providing a programmable interface to the control / data structure of the characterization environment, we give the user the capability of constructing their own characterization strategies and protocols.
- Able to run faster than real-time: Statistically significant characterization requires a significant number of tests across a wide range of SNR points under multiple scenarios (channel conditions). Since the entire environment including the system under test is expressed in software, we are not limited by a physical time scale. The characterization environment can process tests as fast as the software can run. For the systems we have so far evaluated, this rate is far in excess of the waveforms' actual run rate. The ability to perform characterization faster than real-time shortens the amount of overall time required to collect a suitable number of test results.
- Integrated in the same development system as the system being characterized: Having a unified system for both the characterization environment and the system under test allows for a smoother synchronization and integration process and much easier automation of the characterization.

Given these goals, we developed the architecture shown in Figure 1 (solid lines indicate data flow while dashed lines indicate control flow). The GUI accepts characterization scenario definitions from the user, constructs packets used to send test requests to the control unit, logs and maintains statistics on the test results, and presents graphical feedback to the user. The Control unit accepts test requests from the GUI and generates the configuration information appropriate to carry out the test for the other units of the characterization environment. It also collects the result of the test (from the Compare unit) and returns status information to the GUI.

The Message Generator creates random content data messages of the requested size and appropriate format. These are passed to the transmit side of the link. The XMIT and RECV units implement the transmitter / receiver pair being characterized. In between is the Channel Emulator unit, performing the required channel impairment. The Compare unit compares the original message with the received / decoded message and tabulates discrepancies, which are reported to the Control unit.

As part of a multi-stage development strategy, we first implemented a version of the characterization environment with a Channel Emulator unit supporting (relatively) limited channel models. Once the overall data and control flow of the environment was completed, we followed this initial version with an enhanced version of the Channel Emulator unit supporting full multi-path, dynamic fading channel models.

The initial version of the characterization environment easily fit within a single hx3100 HyperX device, including not only the full characterization environment (exclusive of the GUI, which runs on the host PC) but also an example XMIT-RECV path. See Figure 2 for details. The GUI runs on the Host PC and communicates with a two-board hxHADS development system through an Ethernet connection. In the hxHADS system, the network connectivity is supported by the iMX31 processor on the GPP-IO (General Purpose Processor-I/O) board in slot S1. Only minimal processing occurs at this

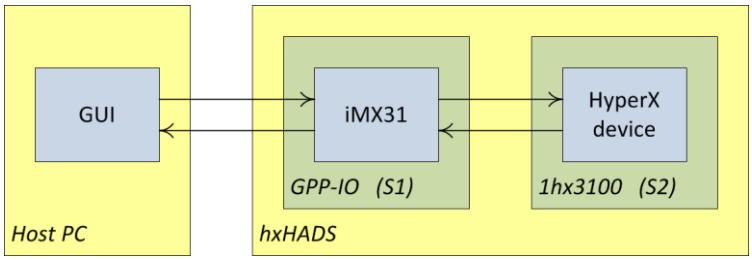

Figure 2. Characterization Environment, 1 Board

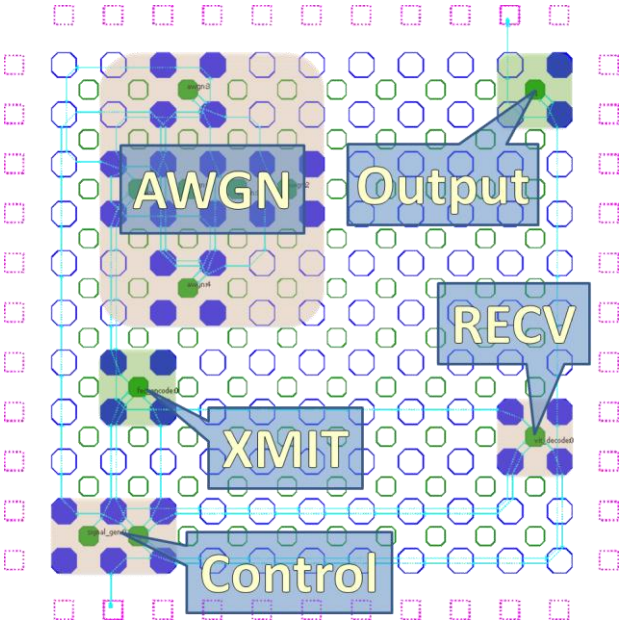

Figure 3. Initial Version Layout View

point; the primary purpose of this board is development system control and to route communications to and from the rest of the hxHADS system. The processing of all of the units of the characterization environment (aside from the GUI) is supported by the hx3100 HyperX device on the 1hx3100 board in slot S2.

For the follow-on version of the characterization environment, we used a four-board hxHADS system with three 1hx3100 boards. The resources available in this expanded system allow for significantly more complex XMIT and RECV units (each of which are allotted their own hx3100). The third hx3100 is used to support the full multi-path, dynamic fading Channel Emulator unit. See Figure 4 for details. As with the initial version, the GUI runs on the Host PC, network connectivity servers run on the iMX31 processor on the GPP-IO board in slot S1, and they communicate via Ethernet. The 1hx3100 board in slot S2 supports the XMIT unit, the one in slot S3 supports the Channel Emulator unit, and the one in slot S4 the RECV unit. The Message Generator unit shares slot S2 while the Compare unit shares slot S4. Appropriate parts of the Control unit are split among slots S2, S3, and S4.

More details about each of the units are provided in the following subsections.

2.2. On the hxHADS Development System

The initial version of the characterization environment (as diagrammed in Figure 2) fits entirely in a single hx3100 HyperX device. An annotated layout view of the hardware resource allocation of this version is shown in Figure 3.

The functionality of the Control, Message Generator, and Compare units (as shown in Figure 1) are folded into a pair of HyperX processing elements (PEs) in the lower left of the layout. Also included in this section is the Input Processing function which mediates the incoming communications from off-chip. Each incoming test series request to the Control unit from the GUI describes a set of tests to run. The request parameters include the size of data message to run (expressed as a number of 16-bit words), the SNR value and average signal power for the AWGN unit, and the number of messages to send. For a given test series, the same SNR and average signal power values are used for all messages. Also included in the test series request are some initialization and control information (e.g., whether to reset the statistics gathering, generator state vector) that aid in chaining together a set of individual test series.

The data content of each message in a test series is independently created by the Message Generator unit. This unit uses a long period ($\approx 2^{115}$ 8-bit bytes in the current implementation) Tausworthe URNG [7] to generate the message contents to ensure non-correlation of message contents across runs.

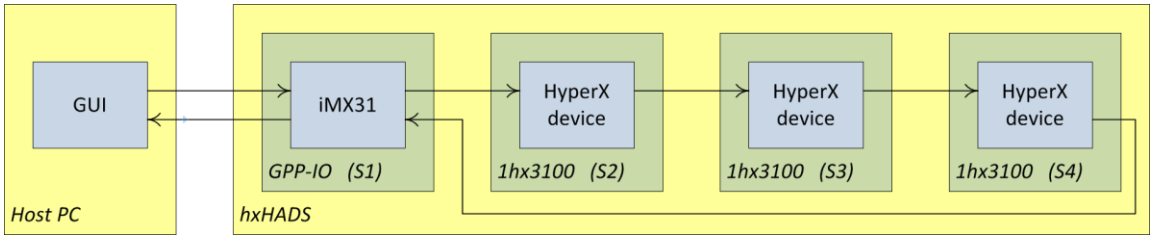

Figure 4. Characterization Environment, 3 Board

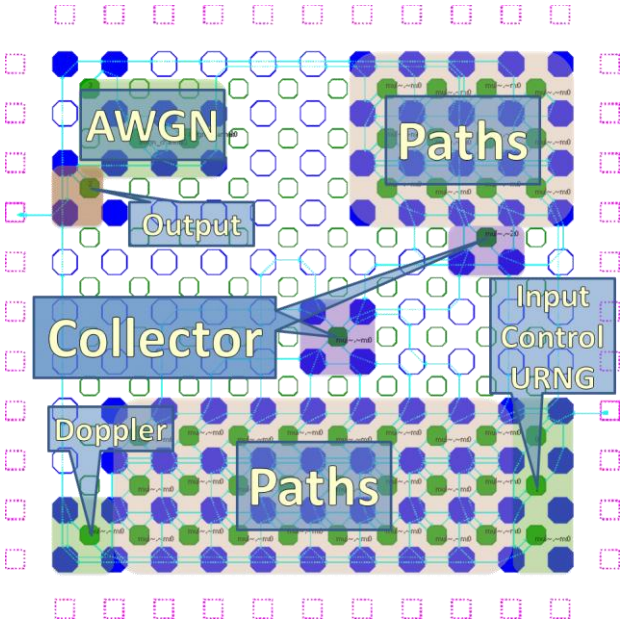

Figure 5. Full Channel Emulator Layout View

The message is then passed to the XMIT unit for generation of baseband I/Q values. (A copy of the message is passed to the Compare unit to match against the results obtained from the RECV unit, described below.) In the initial version of the characterization environment, a simple transmitter is used, consisting of an FEC encoder followed by modulation. This encoder is rate 1/2, K=7. The encoded bits are then punctured to result in a rate 3/4 code. BPSK modulation is then used to produce the final baseband I/Q signal. The entire XMIT unit is implemented in a single HyperX processing element.

The baseband I/Q signal is then passed through the Channel Emulator unit. In the initial version of the characterization environment, the emulator performs only additive white Gaussian noise (AWGN) impairment. The desired SNR and average signal power values (fixed for all tests of a given series) are passed to the AWGN unit from the Control unit at the beginning of the test series. The AWGN unit itself is described in more detail in [1]. In brief, a set of long-period Tausworthe generators are used to produce uniform random numbers which are converted to Gaussian distribution (using the Box-Muller transformation [8]). The final Gaussian values are scaled (according to the specified SNR and average signal power) and then used to impair the I/Q signal.

The AWGN unit in the initial characterization environment was designed and implemented to support high speed operation. Using a six HyperX processing element layout, it is capable of generating in excess of 30 M complex samples/second at the system typical clock frequency of 500 MHz.

After impairment, the I/Q signal is then passed to the RECV unit. This unit must match the encoding / modulation scheme implement by the XMIT unit described above. After BPSK demodulation, the quantized, 3-bit soft values are depunctured and then processed by a Viterbi decoder (constraint length 7) to retrieve the message's data contents. The entire RECV unit is implemented in a single HyperX processing element.

The received message is matched against the originally generated message by the Compare unit. The Compare unit counts all non-matching bits and passes these statistics to the Control unit for tabulation. From this information, the Control unit maintains statistics for the messages of the current test series. When the requested number of messages has been run, the Control unit transmits this information to the GUI running on the Host PC via the Output Processing function. The Control unit (with the included Message Generator and Compare units folded in) is implemented in a single HyperX processing element. The Input Processing function and the Output Processing function are each implemented in their own HyperX processing elements.

As shown in Figure 3, in total only 11 of the 100 HyperX processing elements available in an hx3100 are required to implement the hxHADS portion of the initial version of the characterization environment. This table shows the approximate size (measured in lines of executable ANSI-C code) of each unit:

Control	60
Message Generator	40
XMIT	44
Channel Emulator (AWGN)	125
RECV	146
Compare	10
Total	**425**

Once the initial version of the characterization environment was completed, we moved to implement the enhanced version using the full multipath, dynamic fading Channel Emulator unit. An annotated layout view of the hardware resource allocation of the enhanced Channel Emulator unit is shown in Figure 5. All other units of the characterization environment remained the same.

As shown in Figure 5, in total only 46 of the 100 HyperX processing elements available in an hx3100 are required to implement the 12-path dynamic fading Channel Emulator unit. This table shows the approximate size (measured in lines of executable ANSI-C code) of each unique unit (i.e., not including the per-path units):

Multipath Channel Cell	42
Control, I/O Management	28
Doppler Generator	93
URNG	76
Sequence Collector	30
Total	**269**

Each path unit includes the following sub-units:

Fading Generator	71
Polyphase Interpolator	39
Impulse Response	112
Total	**222**

Each of the paths is implemented by simply instantiating another copy of the identical code. The overall size of the Channel Emulator unit is therefore on the order of only about 500 lines of ANSI-C code. When added to the size of the rest of the characterization environment, we arrive at a total size of fewer than 1,000 lines of ANSI-C code.

2.3. On the Host PC

The user interface to the characterization environment runs on a Host PC attached to the hxHADS system through an Ethernet connection. This GUI carries out a number of functions, including:

- Collection of test scenario parameters from the user.
- Transmission of test series requests to and collection of test results from the hxHADS system.
- Maintenance and presentation of test results to the user.

The test scenario parameters include the range and resolution of SNR values across which to characterize performance, the size (in 16-bit words) of the messages to transmit, and the average signal power of the waveform. Along with these basic test scenario parameters, the GUI also allows the user to adjust the two parameters that drive the operation of the adaptive test scheduling, namely the number of individual messages processed by each test series and the starting number of 'required failures'.

A key part of the characterization environment operation is its adaptation in the face of observed performance. In order to collect meaningful BER and PER statistics as rapidly as possible across the entire SNR range and resolution of interest, it is useful to schedule test series adaptively. In particular, those regions with high BER do not need as many individual tests run as those regions with low BER. In the current implementation, we schedule test series to achieve equal numbers of packet errors as opposed to equal numbers of tests run. The current number of packet errors to achieve is called the 'required fails'. When all SNR points of interest have at least that number of packet errors, the number of required fails is adjusted upwards (doubled in the current implementation).

For example, at the beginning of a characterization run, the number of required fails could be set to 1. Test series requests would be generated from the lowest to the highest SNR value in order until each SNR point had suffered at least one packet error. When that occurs, the number of required fails would be doubled to 2. Any SNR point that already had at least two packet errors would be skipped over in favor of those SNR points that were still at one packet error. When all SNR points have suffered at least two packet errors, the number of required fails would be doubled to 4, and the process continued. In practice, the result of this adaptation is that the characterization environment automatically apportions relatively more computational capacity to those SNR points with the lowest PER.

As mentioned above, the GUI also manages a data base of test results and displays the appropriate statistics to the user. See Figure 6 for an example run (discussed below). In this figure, the BER curve (red '+' marks) and the PER curve (blue '*' marks) are displayed. The overlaid green bars show the total number of tests conducted for each SNR point on a log scale. In the figure, the adaptive apportioning of computational effort towards those SNR points with relatively lower PER can be clearly seen. As expected, the number of tests required to achieve a given number of packet errors rises exponentially to the right (indicated by the linear rise on the log scale chart).

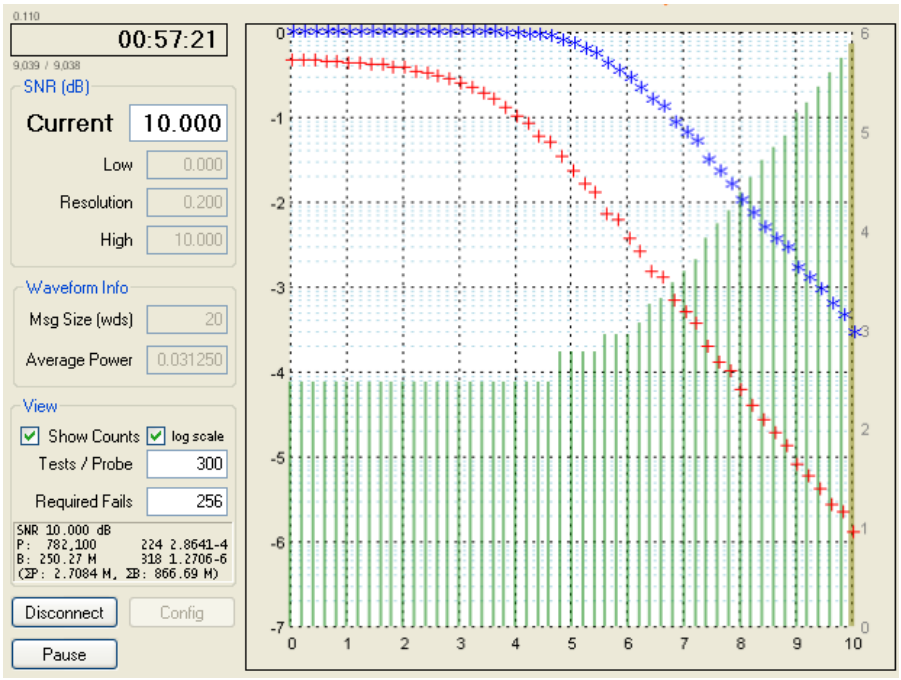

Figure 6. Characterization Environment GUI

3. EXAMPLE RESULTS

An example run of the characterization environment is shown in Figure 6. This run was made using the XMIT and RECV unit described above across an SNR range of 0 dB to 10 dB in 0.2 dB increments. The system clock rate was 500 MHz. After about one hour of wall-clock time, approximately 2.7 M test messages (about 870 M bits) had been run, for a rate of about 800 tests per second. As shown in the figure, for the 10 dB SNR point, 782,100 individual tests had been run (about 250 M bits) with 224 packet errors (318 bit errors) detected. All other SNR points had suffered at least 256 packet errors by this point. During this run, the time required to achieve at least the given number of packet errors was as follows:

Number of Packet Errors	HH:MM:SS
1	00:00:32
2	00:00:44
4	00:01:12
8	00:02:07
16	00:04:04
32	00:07:32
64	00:15:57
128	00:31:17
256	01:00:25

As expected, the amount of time to achieve each successive level of required fails is about twice that of the previous level. The slight irregularities are due to the quantization of the number of individual messages sent in each test series.

4. SUMMARY AND ONGOING WORK

In this paper we have presented the design and implementation of software implemented, real-time, adaptive characterization environment that is capable of automatically collecting Bit Error Rate and Packet Error Rate statistics. While the overall performance of the system has met its original goals, additional feature and performance enhancement opportunities have been identified. In particular, given the computational capacity of the HyperX device, we will explore the migration of the adaptive nature of the characterization environment from the Host PC GUI to inside the hxHADS system. This migration will permit a tighter feedback loop between the running of the test series and the adaptive sweeping of channel model parameters. An additional performance enhancement could be achieved by coordinating the use of multiple hxHADS systems in parallel. Since the test series are all independent, it is possible to use multiple hxHADS systems in parallel and combine their results. A linear speedup in results collection would result.

Aside from the initial Bit Error Rate and Packet Error Rate statistics that are currently collected, we have been asked by users to collect Burst Error Rate statistics. The code necessary to collect these statistics has been added to the Compare unit (at a cost of an additional 45 lines of ANSI-C code but no additional cost in HyperX processing elements). The characterization environment also has the capability of monitoring the cycle count cost of any of the units under test. Though at present we do not make use of this information, input from our users has indicated that this additional statistical information is of value. We are exploring the most useful way to make this information available to aid in performance bottleneck analysis, clock reduction analysis for power savings, etc.

Regarding the included Channel Emulator unit, we are at present extending its capabilities in a number of directions. While the current 12 path capacity is suitable for most applications, certain advanced channel models require significantly more (e.g., up to 24 paths). Our internal waveform development team has also expressed a need for MIMO (in particular 2x2 and 4x4) connections beyond the current SISO capability. A further need of that group is more complex impairment models themselves, beyond the current Rayleigh / Ricean capability. We are also exploring the addition of mobility modeling to support the analysis of sets of MANET nodes.

5. REFERENCES

[1] B.A. Dalio, I. Aguayo, K.A. Shelby, "The Design and Implementation of a Real-Time, Software-Based, Multi-Path Fading Channel Emulator for an SDR Development Platform", Proceedings of the 2010 European Reconfigurable Radio Technologies Workshop and Product Exposition, Jun. 2010

[2] The MathWorks, http://www.mathworks.com/

[3] Spirent Communications, http://www.spirent.com/

[4] Elektrobit, http://www.elektrobit.com/

[5] N. Patel, K.A. Shelby, and B.A. Dalio, "Radio Waveform Development System Providing an Integrated Approach to SDR Waveform Design and Implementation", Proceedings of the SDR '10 Technical Conference and Product Exposition, Dec. 2010.

[6] B.A. Dalio and K.A. Shelby, "The Implementation of OFDM Waveforms on an SDR Development Platform supporting a Massively Parallel Processor", Proceedings of the SDR '09 Technical Conference and Product Exposition, Dec. 2009.

[7] P. L'Ecuyer, "Maximally Equidistributed Combined Tausworthe Generators", Mathematics of Computation, v65 n213, Jan. 1996, p. 203-213.

[8] G.E.P. Box and Mervin E. Muller, "A Note on the Generation of Random Normal Deviates", The Annals of Mathematical Statistics, v29 n2 p. 610–611.