THE IMPLEMENTATION OF OFDM WAVEFORMS ON AN SDR DEVELOPMENT PLATFORM SUPPORTING A MASSIVELY PARALLEL PROCESSOR

Brian A. Dalio (Coherent Logix, Inc., Austin, TX, USA, dalio@coherentlogix.com);
Kevin A. Shelby (Coherent Logix, Inc., Austin, TX, USA, shelby@coherentlogix.com)

ABSTRACT

In this paper, we present the agile development, implementation, and verification of an OFDM waveform PHY layer on a Software Defined Radio (SDR) development platform. The SDR platform is supported by a tailored and highly productive development system emphasizing the value of a structured development process that takes the developer through modeling, trade-off analysis, implementation, and verification. We comment on the OFDM models, the design process used to create them, their characterization and verification, and their ultimate performance characteristics. Central to the SDR platform is a massively parallel processor which is used for all signal processing, control, and data management aspects of the OFDM waveform. We present details of this computational fabric and its advantages in the implementation of OFDM waveforms and SDR applications.

1. INTRODUCTION

Traditional SDR platforms use a General Purpose Process (GPP), a Digital Signal Processor (DSP), and various Field Programmable Gate Arrays (FPGAs) to accomplish the waveform processing related to the Physical and Link Layers. This architecture (generally) meets SDR functional and computational performance requirements, but is physically large, costly, and not very power efficient. The FPGA devices are generally in very large packages and their static and dynamic power dissipation is relatively high. (Lower power and smaller FPGA devices are available, but these devices generally do not have the capacity or performance required for SDR applications.) The DSP can also be performance limiting, providing a relatively limited number of multiply-accumulate and other functional units. These characteristics have proved to be a barrier to low-power, small form factor SDR systems.

Another difficulty of the traditional SDR architecture is that the parts of the system require different design, development, implementation, and verification processes. While applications for embedded GPPs generally can be developed in standard software programming languages, DSPs often require low-level programming in non-standard or manufacturer-specific languages or language extensions. Further, the FPGA portion of the design generally is not represented in a software language at all but has to be coded in a Hardware Description Language (HDL), such as VHDL or Verilog.

Given the different representations that must be used for the portions of the system depending on whether it happens to be located in the GPP, DSP, or FPGA, it can be quite costly (and error-prone) to move functionality from one resource to another as part of a re-partitioning as system design requirements and/or performance constraints change over time.

We present an alternative SDR design process and platform that minimizes the above difficulties by allowing for a completely software based baseband processing solution. (See [1] for an approach that attempts to maintain the traditional architecture.) This solution, described in section 2, comprises a hardware platform based on a massively parallel processing fabric, a unified toolset and design methodology, and a set of reusable, reconfigurable components. To provide a concrete example during the ongoing development of this SDR design process and platform, an implementation of the 802.11g ([2], [3]) PHY layer was created, described in section 3. A summary of current results, other waveform implementations in progress, and ongoing methodology, tool, and architecture development is given in section 4.

2. THE HYPERX™ SDR ENVIRONMENT

2.1. HyperX Processor Architecture

The HyperX processor architecture is a scalable unit-cell-based hardware fabric—the HyperX fabric—which consists of an array of HyperSlice™ units. Each HyperSlice contains one to four Data Memory and Routing (DMR) units and one Processing Element (PE) unit. The complete hardware architecture is formed by replicating the HyperSlice core unit-cell to create a massively parallel processing system. Surrounding the core is a programmable I/O structure, the HyperIO™.

The PE instruction set architecture is designed for computation intensive communications and image/video processing. The PE core contains a 24-bit multiplier, 16-bit ALU, 40-bit accumulator, 40-bit barrel shifter, and floating-point, reciprocal, and reciprocal square-root unit. The PE can natively process 8-bit (using a SIMD capability), 16-bit fixed point, and 32-bit floating point data. The PE is based on the Harvard architecture

model using flat, separate instruction and data memories. Operands can be fetched from a DMR's memory, instruction fields, and registers. The register set includes sixteen 16-bit general purpose registers, a 40-bit accumulator, three 16-bit pointer registers, and three 16-bit index registers. The directly addressable data memory space is 32 KB with 8 KB available from each adjacent DMR. The PE operates on a variable width instruction word and is built on a variable stage, fully registered pipeline for maximum parallelism and hardware efficiency. Zero overhead looping and conditional moves are supported to minimize pipeline breaks.

The DMR provides data memory and routers for fast routing services to the processing resources. The fabric is created by joining the DMRs of adjacent HyperSlices together to form the on-chip network. This network operates independently of and transparently to the processing resource. It provides a programmable and adaptable communications fabric supporting arbitrary network topologies (H-tree, in-place or dilated Butterfly, Multi-D mesh, etc.). The DMR provides nearest neighbor, regional, and global communication across the device and from device to device through four different transfer modes: memory to memory, memory to register, register to memory, and register to register. Each of the transfer modes may physically use the DMR resources differently depending on locality of data and software requirements. All DMRs can operate independently of each other.

The HyperIO controls the data I/O to and from the core. Each HyperIO connects to a periphery DMR and associated I/O pads. The HyperIO extends the HyperX fabric with the added capability of seamlessly connecting the on-chip communications to other devices while preserving the same programming model across the device boundary. In addition, the HyperIO enables interfacing to memory, processor buses, analog-digital converters, sensors, displays, etc.

Different implementations of the HyperX architecture may vary in the size of the fabric and the amount and kind of HyperIO connections available. The current project was implemented on the hx2100 version of the HyperX technology, which has a 10x10 grid of PEs (100 total) and an 11x11 grid of DMRs (121 total).

2.2. *FAST*™ Design Process

The *FAST* Design Process has been developed to enable the rapid implementation, verification, optimization, and mapping of systems onto the HyperX hardware. This four step iterative approach guides the user through the necessary stages to facilitate efficient design implementation.

In the *Functional Analysis* step, the user captures the logical functionality of the system, defines the required representation precision and dynamic range, and determines the required number of operations for each function of the system. In this stage, high-level models of the various functions can be developed using a variety of tools (e.g., MathWork's MATLAB and/or Simulink, GNU Octave). A comprehensive verification infrastructure is developed (including test cases matching the functional models) that can be used throughout the process.

In the *Architectural Constraint Derivation* step, the user specifies the system performance constraints in the form of input/output data rate(s) and/or latency requirements for each of the units of logical functionality defined in the *Functional Analysis* step.

In the first part of the *System Design* step, the user analyzes the functional and architectural constraints and optimizes the design to meet functional and architectural requirements on the target hardware. This step can be iterative, based on the overall system performance requirements. In the second part of the *System Design* step, an ANSI-C version of the application is developed using standard coding techniques employing the representative natural hierarchy previously derived. Industry standard Message Passing Interface (MPI) [4] calls are used to express the parallelism of the system. From this ANSI-C / MPI representation, the final step, *Transformation to Hardware*, is fairly well automated.

In the *Transformation to Hardware* step, the user does automated mapping and optimization of the design to the physical resources available in the target hardware using the HyperX ISDE suite. The transformation is based on the parallel functionality; tasks can then be optimally assigned to different processing resources. This is a four-dimensional constraint-driven resource allocation process (two spatial dimensions of the HyperX fabric, time-of-execution dimension, and performance dimension). This step can be iterative, based on the overall system performance requirements.

2.3. HyperX ISDE™

The HyperX ISDE supports integrated application software development and verification in standard ANSI-C, without requiring detailed knowledge of the internals of the HyperX technology. The source code developed by the user is platform independent until compiled for the target HyperX device.

The programming model presents a system view and isolates the developers from hardware details by automating task allocation, memory allocation, and communication connections. (Manual optimizations and user tweaks are supported.) The programming model is transparently scalable across any size HyperX fabric or across multiple interconnected HyperX devices. After an application is compiled, it is linked to only those specific resources that are required to minimize power consumption.

The interactive graphic development environment provides several views of the application as it is developed, compiled, and debugged. The Software Centric System view shows a task-by-task breakdown as it was extracted from the user's ANSI-C / MPI source. It can be used for critical path analysis and for tradeoff analysis of latency, power, and resource use. The Communications Centric System view shows the task flow on the hardware PE / DMR fabric, showing how the software is mapped to the hardware and how the communications network synthesized

Table 1. OFDM Waveform Parameters

Parameter		*WiFi (802.11a/g)*	*Fixed WiMAX (802.16d)*		*Mobile WiMAX (802.16e)*		*LTE (UMTS)*		*Units*
DL	*DL Access*	OFDM	OFDM		S-OFDM		OFDM		
UL	*UL Access*	OFDM	OFDMA		S-OFDMA		SC-FDMA		
B	*Bandwidth*	20	DL	1.75/3/ 3.5/5.5/7	DL	1.25/ 2.5/5/10/20	1.4/3/5/10/ 15/20		*MHz*
			UL	1.25/ 3.5/7/14/28	UL	1.25/5/10/20			
N_{FFT}	*FFT Dimension*	64	DL	256	128/512/1024/2048		128/256/ 512/1024/ 1536/2048		*pts*
			UL	2048					
$\Delta f = B/ N_{FFT}$	*Subcarrier Spacing*	312.5	11.16		10.94		15.00		*kHz*
$T_{FFT} = 1/\Delta f$	*FFT Duration*	3.2	89.6		91.4		66.67		μs
$T_G = \%T_{FFT}$	*Guard Interval*	2^{-2}	2^{-3}		$2^{-[2:5]}$		$2^{-[2, 3.83, 3.678]}$		$\%T_{FFT}$
$T_S = T_{FFT} + T_G$	*Symbol Duration*	4	100.8		114.25, 102.83, 97.11, 94.26		83.27, 71.36, 71.88		μs
M	*Modulation*	BPSK, QPSK, 16/64QAM	BPSK, QPSK, 16/64QAM		BPSK, QPSK, 16/64QAM		QPSK, 16QAM, 64QAM (DL)		
c	*Coding*	CC	RS-CC, BTC, CTC		CC, BTC, CTC, LDPC		CC, Turbo		
r	*Data Rates*	6-54	2-134.4		2-134.4		DL	100	*Mb/s*
							UL	50	

from the user's application. The Energy Centric System view helps the user to understand the power consumption profile of the application as it is mapped to the hardware. By visualizing how the application consumes power across the two spatial dimensions of the hardware and through time, tradeoffs among resource use, application performance, and power consumption may be made. The Hardware Centric System view includes a debugging environment common to both software (via a cycle-accurate simulator representing the entire PE / DMR fabric) and hardware (through Ethernet or USB connection to hardware). The debugger presents the same capabilities (single step, breakpoints, data display/manipulation, source code display/tracing, etc.) for both simulation or hardware.

2.4. HyperX Application Development System (HADS)

The HADS board provides hardware and software developers a platform for designing applications targeted for the HyperX processor. In a microATX form-factor, the current generation of the HADS board (HADS1) has these resources:

- hx2100 HyperX processor with 384 MB DDR2 memory, 8 MB flash memory
- Freescale i.MX31 (ARM 11 core running Linux RedBoot O/S) with 128 MB SDRAM memory, 32 MB flash memory (on board), 1-4 GB SD flash memory
- Ethernet, USB, UART connectivity
- User access to all hx2100 HyperIO pins through daughter card connectors
- FPGA (Cyclone II 2C35) and CPLD (MAX EPM570) resources for interfacing

The HADS board interfaces seamlessly with the HyperX ISDE toolset described in section 2.3 above. Multiple HADS1 boards may be interconnected via System-to-System boards and used to develop high-performance multiprocessor systems.

2.5. Reusable SDR Components

A key part of any high productivity development process or environment is the ability to create and deploy reusable and reconfigurable components. This is especially true in the development of SDR applications as waveforms in general and families of waveforms in particular often share very similar if not identical requirements. Component reuse, however, has not been easy to achieve in the traditional architecture SDR (e..g., [5], [6]).

The HyperX SDR Platform supports reusable and reconfigurable components via its completely software based design and development environment. Once a function has been written, it may be reused as simply as one would call a subroutine in a software program. Since an application for the HyperX SDR

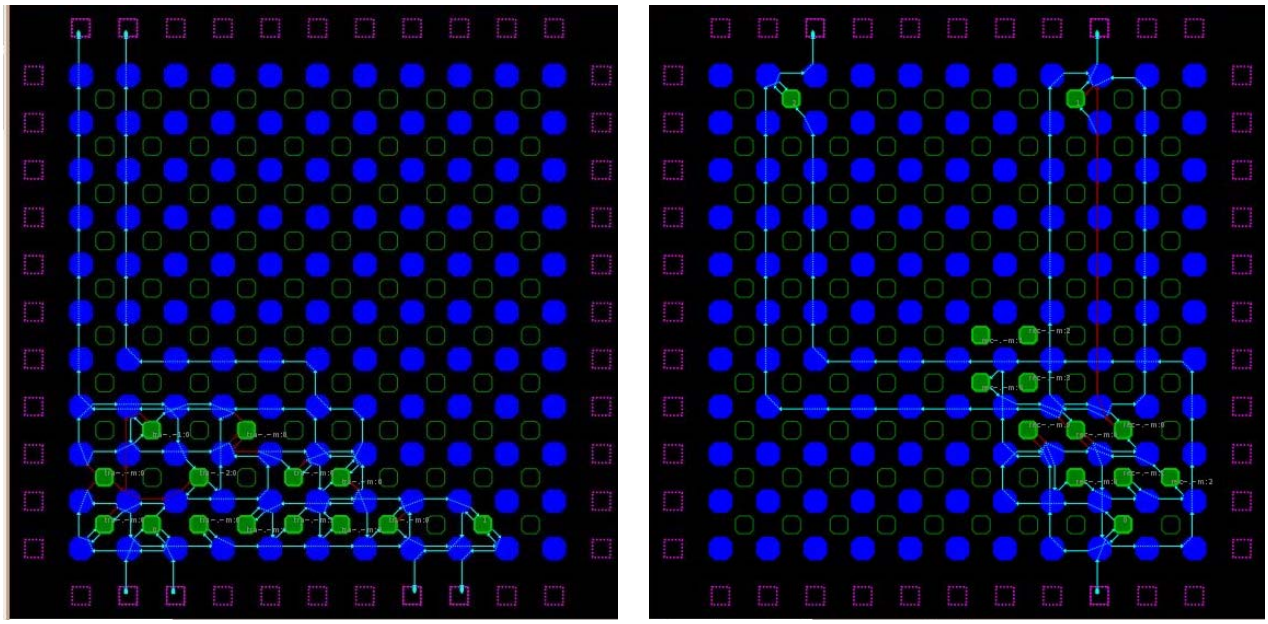

Figure 1. 802.11g PHY Layer Implementation, Tx left, Rx right

Platform is written in ANSI-C / MPI, it *is* a software application and reusing functions *is* as simple as calling a subroutine (or sending a message via MPI). Variations of functions may be provided that accomplish the required operation using different implementation techniques that emphasize different points in the tradeoff space of computational load, resource use, power consumption, and throughput performance.

In the current project, an example of this kind of reuse is given by the FFT module. The generic FFT may be configured for size (2^n for n=5..11), data representation (floating point or fixed point), aspect ratio on chip (rectangle or square, depending on the required floor plan), and number of PEs to use (four or eight). Test cases can be generated along with expected golden output for verification.

Another reusable component is the system-level Additive White Gaussian Noise (AWGN) module. Characterization of a receiver in the presence of AWGN can take a long time when each test must be done in a simulation environment (e.g., MATLAB). Characterization can be accomplished much more efficiently when a loopback version of a transceiver can be combined with a programmable AWGN channel model in hardware.

3. OFDM WAVEFORM IMPLEMENTATION

3.1. OFDM

OFDM describes a multi-carrier signaling method whereby the available signal bandwidth is subdivided among multiple orthogonal subcarriers, the contents of which are modulated independently according to some form of QAM or PSK. SDR interests stem from widespread adoption of OFDM across a range of standard waveform applications, e.g. 802.11a/g, 802.16d/e, LTE DAB, DVB-T/H, coupled with the inherent scalability afforded a multicarrier signaling arrangement.

Starting with the Inverse/Forward Fast Fourier Transform pair at the center of the TX/RX baseband processing chains, respectively, OFDM can be interpreted as a bank of parallel single carrier modems tightly packed to maximize bandwidth efficiency. The choice of subcarrier spacing (Δf) is driven by the anticipated channel characteristics with an interest toward ensuring flat fading within the individual subcarrier bandwidth. This permits use of a single tap equalizer along with the associated channel estimation per subcarrier at the receiver.

Viewed in the time domain, narrow subcarrier spacing (relative to the occupied signal bandwidth) equates to slow signal variation/long dwell time per subcarrier with respect to the channel impulse response. Inserting a guard interval between OFDM symbols whose duration exceeds the RMS channel delay spread eliminates the potential for inter-symbol interference (ISI). Introducing known pilot subcarriers, inserted to aid the receiver in tracking carrier frequency offset, mitigates the effects of inter-carrier interference (ICI) thereby preserving orthogonality among the data subcarriers.

Examined from an SDR perspective, the I/FFT represents a fixed processing core, the operating rate of which depends on the relationship between a small number of system parameters as listed in Table 1*:* the signal bandwidth (B) divided among a prescribed number of subcarriers (N_{FFT}) determines the subcarrier spacing. The FFT window duration (T_{FFT}) is inversely proportional to the subcarrier spacing. Adding a guard interval (T_G) in turn determines the overall symbol period (T_S).

Various methods of scalability exist given this arrangement, the underlying differences between which point naturally to an SDR solution built on a configurable processing fabric. For example, LTE and WiMAX employ fixed subcarrier spacing (the

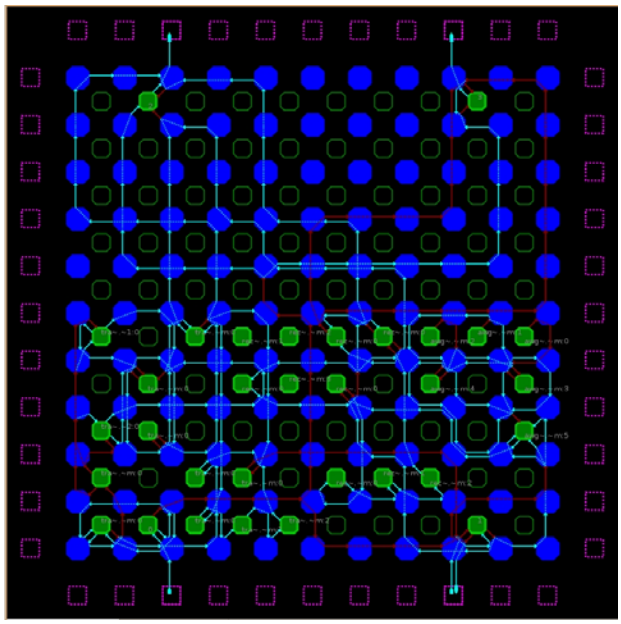

Figure 2. 802.11g Loopback with AWGN Channel Impairment

corresponding I/FFT window duration is also fixed). Additional data and pilot subcarriers are transmitted to increase the occupied signal bandwidth. Mobile WiMAX in addition includes provisions for a selectable guard interval to account for a range of channel delay spreads. DVB-T/H, on the other hand, employs fixed signal bandwidths subdivided among different numbers of subcarriers depending on channel conditions, i.e., coherence bandwidth and required delay spread tolerance. (The DVB-T/H standard includes provisions for different signal bandwidths, e.g. 5, 6, 7 or 8 MHz, depending on the regulatory domain. However, the choice of signal bandwidth is fixed per region independent of the number of subcarriers used.)

Channel Estimation and Tracking can be configured to accommodate a variety of pilot arrangements including those involving time varying subcarrier assignments. The data rate afforded a given signaling arrangement is derived from the choice of modulation and coding employed upstream in the processing chain, the configuration of which can be selected as needed to accommodate a broad range of system implementations.

Scalability can be extended to include the Radio Front End and Antenna configurations. MIMO techniques, e.g. space-time coding or spatial multiplexing, can be employed to extend the usable signaling range or to increase system throughput.

3.2. 802.11g Implementation

The current project has as its goal for the first phase of implementation a (reduced) 802.11g PHY layer on the HyperX SDR platform using the *FAST* design process. (See [7] for the implementation of similar blocks on a traditional DSP architecture.) A full OFDM transmitter is implemented and in this initial phase of the project the receiver processing is taken through post-equalizer symbol demodulation. This enables a constellation demonstration suitable for AWGN or static multipath characterization. Provisions for frequency offset due to finite crystal tolerances or Doppler shift are left for the next phase of development. All data rates of 802.11g (6, 9, 12, 12, 18, 24, 36, 48, 54 Mb/s), are supported in this implementation. The Physical layer Service Data Unit (PSDU) length may range from 1 to 4,095 octets. The target system clock frequency of the HyperX SDR platform is set to 225 MHz. (The completion of the receiver is in progress in the second phase of the project; more information about this is given in section 4 below.)

Following the steps of the *FAST* design process, the project began by creating a fully functional MATLAB model of the 802.11g transmitter and receiver PHY layers. For the full PHY layer (i.e., without the limitations in the receiver path mentioned above), about 900 lines of MATLAB code were required, along with about 560 lines of code to serve as test harness, debug plotting, etc.

From these models, bit-accurate C++ models were created and then used for the performance and resource usage estimates. The overall system design was developed from these estimates. The ANSI-C / MPI coding of the functionality was produced next and was verified to be bit-accurate to the higher level models.

This sequence of steps from the Architectural through Transformation to Hardware went through a series of quick iterations as tradeoffs were made to balance resource usage, required performance (e.g., data throughput), and computational cost. These tradeoffs were balanced against the 225 MHz target clock rate of the HyperX SDR platform.

The sizes of the final modules are now given. "LOC" stands for Lines of Code and includes not only all executable code but also extensive in-line comments and documentation (considerably inflating the size of the modules).

Transmitter Module	**LOC**
Transmitter (top level)	83
Frame converter	231
Header transmission	1,148
Scrambler	511
Convolutional Encoder	1,054
Interleaver	830
Modulator	511
IFFT	930
Header data concatenation	171
Total:	**5,469**
Receiver Module	**LOC**
Receiver (top level)	82
Preamble and CP deletion	116
Channel Estimation	1,115
FFT	910
Equalizer	251
Demodulator	1,075
Total:	**3,549**

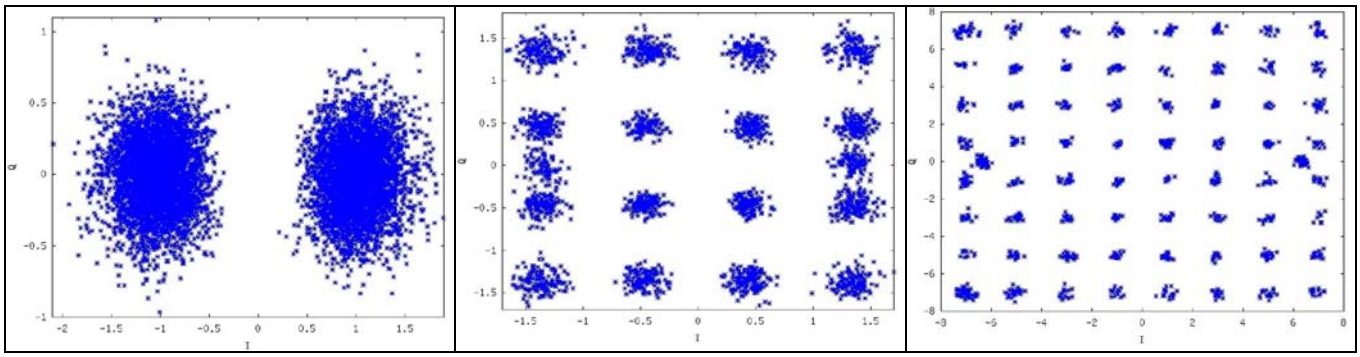

Figure 3. Sample AWGN Impaired Constellation Diagrams

The design also included about 800 LOC representing the interface between the PHY layer code and the HyperX Application Development System.

3.3. Results

Following the implementation of the transmitter and receiver PHY layers, the applications were place-and-routed in an hx2100 HyperX device. The resulting layouts are shown in Figure 1. The transmitter (left) and receiver (right) are each shown in a standalone verification configuration.

To demonstrate the complete functionality simultaneously as well as provide a means for characterizing the receiver performance, a combined "loopback" configuration connected via a hardware Additive White Gaussian Noise (AWGN) channel impairment module was implemented. This layout is show in Figure 2. The AWGN module was itself designed using the *FAST* process and implemented on the HyperX SDR platform. This module was about 640 LOC.

Characterization runs were made with 512 octet payloads across all data rates. Sample constellation diagrams for, left to right, 6 Mb/s (BPSK), 24 Mb/s (16-QAM), and 54 Mb/s (64-QAM) are shown in Figure 3. The measured Error Vector Magnitude (EVM) for 6 Mb/s at an SNR of 5 dB was 0.338. For 24 Mb/s at an SNR of 16 dB, the EVM was 0.098 and for 54 Mb/s at an SNR of 25 dB, the EVM was 0.014.

4. SUMMARY AND ONGOING WORK

In this paper we have presented the implementation of an 802.11g PHY Layer Transmitter and (reduced) Receiver completely in software as an application for the HyperX SDR Platform. The implementation was developed using the *FAST* design process. All 802.11g data rates were successfully implemented. The project continues—the completion of those parts of the receiver omitted in the initial phase are currently in progress.

Other waveforms have been or are being developed for the HyperX SDR Platform. These include strictly military waveforms such as the Soldier Radio Waveform (SRW) and the Wideband Networking Waveform (WNW) as well as civilian waveforms such as the present project, 802.11g, and 802.16e. Another example is GPS in both its civilian and military versions.

Each waveform implementation project has provided insight into how to improve, enhance, and extend. In particular, the next generation of the HyperX Application Development System (HADS2) is now undergoing board-level test. Replacing the single HyperX device capability of the HADS1 board with a 4-, 8-, and 16-slot backplane, HADS2 provides for up to 32 HyperX devices in a single system. Extended I/O and GPP boards are also being tested. In the toolset, automatic packing of concurrent tasks into a single PE and direct compilation of Simulink models are undergoing evaluation. We continue to improve the energy efficiency of the HyperX architecture itself by continuing the exploration of the tradeoff possibilities between software and hardware.

5. REFERENCES

[1] D. Haessig, J. Hwang, S. Gallagher, M. Uhm, "Case-Study of a Xilinx System Generator Design Flow for Rapid Development of SDR Waveforms", *Proceedings of the SDR 05 Technical Conference and Product Exposition*, 2005.

[2] IEEE, *IEEE Std 802.11-2007*, IEEE, New York, NY, June 2007.

[3] IEEE, *IEEE Std 802.11g-2003*, IEEE, New York, NY, June 2003.

[4] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*, MIT Press, Cambridge, MA, Sept. 1998.

[5] C. Epifanio and M. Uhm, "The Myths of Code Portability", *Proceedings of the SDR 07 Technical Conference and Product Exposition*, 2007.

[6] D. Rupe, "An FPGA Framework Supporting Software Programmable Reconfiguration and Rapid Development of SDR Applications", *Proceedings of the SDR 07 Technical Conference and Product Exposition*, 2007.

[7] V. Ramadurai, S. Jinturkar, S. Agarwal, M. Moudgill, J. Glossner, "Software Implementation of 802.11a Blocks on Sandblaster DSP", *Proceedings of the SDR 06 Technical Conference and Product Exposition*, 2006.