This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2017.2651054, IEEE Transactions on Emerging Topics in Computing

Hardware-Software Codesign of Wireless Transceivers on Zynq Heterogeneous Systems

Benjamin Drozdenko, *Member, IEEE,* Matthew Zimmermann, Tuan Dao, *Member, IEEE,* Kaushik Chowdhury, *Senior Member, IEEE,* and Miriam Leeser, *Senior Member, IEEE*
Department of Electrical and Computer Engineering, Northeastern University, Boston, MA, 02115

Abstract—Recently, wireless technology has seen many new devices, protocols, and applications. As standards adapt to keep pace with hardware availability and user needs, the trend points towards systems that achieve high data rates with low energy consumption. Moreover, there is an emerging vision of a transceiver architecture that can adapt to multiple protocols, existing and evolving. This architecture maps computation to underlying heterogeneous computing elements, composed of processors and field programmable gate array (FPGA) fabric. Here, we introduce a method for modeling a generic orthogonal frequency division multiplexing (OFDM) wireless transceiver on the Zynq system-on-chip by decomposing the standard specifications into a set of functional blocks used in multiple protocols. Implementing the 802.11a physical (PHY) layer as an example, our approach creates Simulink model variants for both transmitter and receiver, each with a different boundary between hardware and software components. We use these models to generate hardware description language (HDL) code and bitstream for the programmable logic and C code with an executable for the advanced RISC machine (ARM) processor. We validate, profile, and analyze the models using metrics including frame time, resource utilization, and energy consumption. Our results demonstrate how to select a co-design configuration considering execution time and energy, and show how our platform can be reused for multiple-input multiple-output (MIMO) and protocol coexistence.

Index Terms—Heterogeneous (hybrid) systems, Reconfigurable hardware, Data Communications Devices, Receivers, Transmitters, Signal processing systems, Wireless communication, Wireless systems, Hardware/software interfaces, Hardware/software codesign

1 Introduction

In recent years, the field of wireless technology has seen a tremendous surge in the diversity of devices, protocols, and applications. Wireless devices are more prevalent and used for a wide variety of purposes. The Internet of Things (IoT) comprises such devices as phones, watches, thermostats, cars, electric meters, sensors, clothing, televisions, and medical devices. Estimates show that there were 10 billion things in 2013, there will be 50 billion by the year 2020, and IoT is expected to be a $14.4 trillion business over the course of 2013-2023 [1].

As the need grows for the use of wireless networks for more diverse, data heavy applications, wireless protocols must be adapted to meet the various needs of these applications, including higher data rates and lower energy consumption. As the number of users increases on commonly-accessed mobile bandwidths, congestion becomes another issue, and more versatile methods must be put in place to handle the contention inherent in multiple access. Modern-day wireless communications standards are constantly evolving to meet the needs of an increasing number of devices. The latest standard for mobile phone technology is known as long-term evolution (LTE), and many research projects are in place to prototype and test the 5th generation (5G) technology.

These evolving protocols have brought a need for a flexible, reconfigurable, programmable design framework for future wireless systems. Such a system requires a radio frequency (RF) front end as well as elements to process the signals, often under strict timing or energy constraints. To handle the most time-sensitive tasks, heterogeneous computing architectures have been introduced, which combine a software (SW) processor component with reconfigurable hardware (HW). The premise of a software-defined radio (SDR) has opened up the field to allow for reconfiguration of a transceiver device for adaptation to evolving standards and protocols. From SDR, the idea of a cognitive radio (CR) has evolved to personalize SDRs [2]. Recent research has explored such problems as spectrum scarcity by sensing shared bandwidths and switching center frequency as needed [3]. Various SDR and CR testbeds have been introduced to prototype a real-time, online transceiver system, but each testbed has its limitations. Not all testbeds combine SW and reconfigurable HW [4], [5]. Some testbeds do not allow the developer to modify the HW component, or require the developer to use predefined SW routines [6], [7]. Few testbeds allow the developer a high-level interface for designing both SW and HW components, and fewer release their designs for use by the general public [8], [9], [10].

In this research, we target a Xilinx Zynq based platform coupled with an Analog Devices RF front end board. The Zynq chip includes both an embedded ARM processor for software implementations as well as Field Programmable Gate Array (FPGA) fabric. This platform is low cost, flexible and easy to upgrade. In addition, this platform allows us to experiment with which components are best suited for processor SW or reconfigurable HW. This choice may vary depending on which protocols and layers are supported, the target hardware platform, and characteristics of the environment such as congestion.

We explore the problem in HW and SW using commercially available tools, including MathWorks Simulink and Xilinx Vivado. We demonstrate our approach using IEEE

2168-6750 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2017.2651054, IEEE Transactions on Emerging Topics in Computing

802.11a transmitter (Tx) and receiver (Rx) Simulink models and ensure correctness by comparing against Annex G of the 802.11a specification [11]. These models require modification, such as different data types, to best target execution on HW or SW. We generate hardware description language (HDL) code and intellectual property (IP) core blocks for the components targeted for execution in HW, and also generate C code to be compiled into an executable that runs on the advanced RISC machine (ARM) processor from these high level models. We present information on timing, resource utilization, and energy consumption for each of the different HW/SW co-designs. This approach allows a designer to experiment with which implementations are best suited for the needs of different wireless protocols, depending on the usage scenario.

Our approach advances the state of the art in the following ways:

Common Modifiable Hardware/Software Platform: We target easily available, off-the-shelf commercial HW components including the Xilinx Zynq and Analog Devices RF transceiver chips and front ends, as well as software tools from MathWorks and Xilinx that are widely used in industry and academia for wireless transception and research. Our platform HW and SW can easily be replicated by other researchers and used for real-time implementations of SDR and CR, exploration of design tradeoffs both at the HW/SW co-design level, and at the algorithm selection level. Unlike SDR alternatives like WARP [6] and Sora [12], our platform allows the user to modify the HW blocks at a high level and requires no proprietary HW or SW. We plan to share our designs with other researchers.

Exploration of HW-SW Design Tradeoffs: Our approach provides a mechanism for prototyping widely-used wireless protocols using HW and SW variants on FPGA and ARM processor respectively. By mapping wireless behaviors to processing elements, we can determine whether any particular behavior is better suited for implementation on HW or SW, given information such as proximity to the RF front end, time and power metrics, and use of FPGA resources. Previous mappings of protocols to hardware testbeds have not included the embedded ARM processor as a target. Each component in the 802.11a PHY-layer processing chain has a HW and SW implementation, allowing designs to be analyzed for speed and energy consumption. The FPGA fabric can support real time processing as long as the path delay meets defined timing constraints. Built-in SW profiling tools can monitor execution time on SW and path delay on HW to ensure real-time operation. In addition, Vivado reports provide power consumption information for the FPGA, allowing choices to minimize energy usage. Future research will explore methods for choosing the fastest possible implementation or minimizing the energy used during active periods based on user constraints.

A Platform for Next Generation Wireless Research: Using a heterogeneous system that consists of a processor and reconfigurable HW, we can modify the functionality of the transmitter and receiver to adapt to evolving protocols. The RF front end can be programmed to dynamically change bandwidths by modifying such parameters as center frequency (f_c) and sampling frequency (f_s). The platform enables research on a number of signal processing and communications techniques, including choices for preamble detection, modulation, and encoding schemes that optimize such metrics as packet error rate, bit error rate, and link latency. We can support spectrum coexistence, such as LTE and Wireless Firewall (Wi-Fi) on the same channel, TV whitespace reuse, or co-operation with Radar. We can support spatial diversity, using multiple antennas (MIMO) and transmitting identical sequences using alternate encoding or modulation techniques, to overcome the effects of fading and interference. Subcarrier selection can be supported with a system that can dynamically assign symbols to specific subcarriers that have been identified to have maximum channel efficiency, rather than mapping modulated symbols to a fixed set of subcarriers. Future standards such as 5G LTE will be explored using this testbed.

The rest of the paper is organized as follows. Sec. 2 discusses similar research that has been undertaken in the fields of SDR, CR, and heterogeneous architectures. Sec. 3 explains key algorithms in the IEEE 802.11a standard. Sec. 4 describes our targeted HW and SW. Sec. 5 introduces the HW-SW co-design variants that collectively comprise our transceiver system model, including timing considerations and user workflow. Sec. 6 describes HW-SW interfacing and illustrates metrics measured from the experiments such as time, utilization, and energy. In Sec. 7, we propose topics for future research that can be prototyped using our flexible SDR testbed. In Sec. 8, we summarize our findings.

2 Related Work

For designing SDRs, there have been many HW devices, SW tools, and systems or solutions introduced.

2.1 SDR Hardware

Any SDR implementation that can support physical radio transmissions requires a radio frequency (RF) front end. For easy connection to computer hardware, many SDR projects use the Ettus Research Universal Software Radio Peripheral (USRP), an RF front end board commonly used in wireless research [15]. USRPs feature integrated circuit chips for RF transception by Analog Devices Inc. (ADI), including the wideband wireless AD9361 and AD9364 transceivers [16]. To attach to the FPGA Mezzanine Card (FMC) slot on Xilinx FPGA and Zynq System-on-Chip (SoC) boards, ADI introduced the AD-FMComms series.

2.2 SDR Software

Specialized SW is needed to effectively work with the SDR systems and perform the signal processing tasks needed to instantiate wireless communications. GNU Radio is one of the most widely used SDR programs, owing to the fact that it is open source, HW-independent, and modifiable [17]. Its graphical interface, GNU Radio Companion, allows the user to build block diagrams to represent complex encoding and decoding schemes. However, the GNU scheduler is built for operation on a CPU, and much additional effort would be required to make designs real-time, clocked, and compatible with custom HW components. MATLAB and Simulink are also widely used tools for modeling algorithms in digital signal processing systems. As the basis for communications

2168-6750 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2017.2651054, IEEE Transactions on Emerging Topics in Computing

TABLE 1: Comparison of features for different SDR systems and solutions

Features	Atomix [4]	CODIPHY [5]	WARP [6]	Sora-Ziria [12]	CoPR [8]	Airblue [9]	Iris [10]	CRASH [13]	NI LTE/Wi-Fi Testbed [14]	Our Method
System can be described at a high level	✓	✓	✓	✓	✓	✓				✓
Combines SW processor&Reconfigurable HW			✓	✓			✓	✓	✓	✓
Developer can program HW/FPGA					✓	✓	✓	✓		✓
Can use to study HW/SW codesign tradeoff								✓		✓
Easily modifiable for Next Gen protocols	✓	✓								✓
Can use to study protocol coexistence									✓	✓
Full design open and available to public								✓		✓

system design, MathWorks produces HW support packages for interfacing with commonly-used RF front-ends, including for Zynq-Based Radio and USRP-based Radio [18]. For example, in [19], a high-level cognitive radio framework is designed for bidirectional transception using the USRP N210, MATLAB, and IEEE 802.11b.

2.3 SDR Systems and Solutions

An overview of SDR systems and solutions is shown in Table 1. Several features are desirable in an SDR, including high level description, both CPU and FPGA available, and open design specifications that are available to the public. System designers also want to program the FPGA and study HW-SW codesign tradeoffs. Wireless engineers want to modify processing blocks for next generation protocols and study protocol coexistence on the same bandwidth.

Some research focuses on high-level SDR descriptions that automatically trace down to low-level implementations. Atomix is a modular SW framework for building applications on wireless infrastructure that builds an 802.11a transceiver using fixed-timing computations called *atoms* to utilize the cores of a multi-processor DSP [4]. Atomix is intended only for synthesis on a variety of DSPs, not for reconfigurable HW. CODIPHY uses Xilinx System Generator to generate synthesizable designs from MATLAB programs and automatically generate C and VHDL for an 802.11a/g Tx and Rx [5]. However, the testing behind CODIPHY is all emulated, not tested on live FPGAs or SoCs.

Some SDR projects are implemented in both HW and SW on a heterogeneous platform that comprises processor, FPGA, and often many custom-made components. WARP is a programmable platform for prototyping wireless networks that combines an RF transceiver, a Xilinx Virtex-4 FPGA board, and an open-source repository of reference designs and support materials [6] [20]. Wireless open-access research platform (WARP) has been used to build a full duplex 802.11 network with OFDM and a MAC protocol [21], and an algorithm for estimating time-of-arrival for OFDM-based transceivers [22]. However, WARP is a fixed HW device with much implemented in ASIC; for this reason, it is difficult to update to accommodate the latest spectrum bands and protocols. The Sora soft-radio stack combines a multi-core CPU and a radio control board, which consists of a Virtex-5 FPGA, PCIe-x8 interface, and DDR2 SDRAM [7]. The Sora platform uses the Ziria language to write high-level SDR descriptions, and is tested using an LTE-like PHY layer and testbed to ensure real-time operations [12]. Unlike WARP, Sora can accommodate various RF front ends. However, SORA does not allow for its Virtex FPGA to be programmed by designers, instead forcing them to use the provided tools, and its internal routines are hidden. CoPR, an automated framework for implementing partial reconfiguration-based adaptive HW systems on Xilinx FPGAs is prototyped for a multi-standard CR transmitter [8]. Airblue introduces an FPGA-based SDR platform for the PHY and MAC layers [9]. However, this platform does not include a SW processor and so cannot be used for studying HW-SW co-design issues.

Numerous recent works have proposed an SDR or CR platform that utilizes a Xilinx Zynq SoC, which combines the ARM-based Processing System (PS) and Programmable Logic (PL) FPGA fabric. In [23], SDR is modeled using GNU radio adaptations for Zynq and Zynq clustering. In [24], Zynq ZC702 boards are combined into a scalable cluster, and a Zedboard task mapper partitions data flows across the FPGAs and ARM cores. Iris uses XML description to link together components to form a full radio system, run them within a PS or PL engine, and test using OFDM for video transmission [10]. In a similar work, the Zynq SoC implements digital pre-distortion as required by 3G/4G base stations, using Vivado HLS to design the PL component [25]. However, these projects either do not make their source code publicly available or were tested using static wireless communications protocols that cannot be modified easily. CRASH utilizes the Zynq Z-7045 System-on-Chip (SoC), which combines both FPGA and ARM processor, and a custom-made PCB to interact with the USRP N210 to perform spectrum sensing [13]. In [26], a platform is proposed using the Zynq with partial reconfiguration and the ADI FMComms4 with tunable operating frequency to enable dynamic, low-power, high-performance CR. This latest work has not been fully implemented and tested. Most recently, the National Instruments real-time LTE/Wi-Fi testbed proposes a new platform for the study of 802.11 and LTE coexistence in 5G technologies [14]. However, this product does not explore HW-SW co-design issues.

3 IEEE 802.11A STANDARD COMPLIANCE

Like the Open Systems Interconnection (OSI) model, 802.11 describes a multi-layered communications approach, with the physical (PHY) layer representing the lowest-level signal processing operations, the media access control (MAC)

2168-6750 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2017.2651054, IEEE Transactions on Emerging Topics in Computing

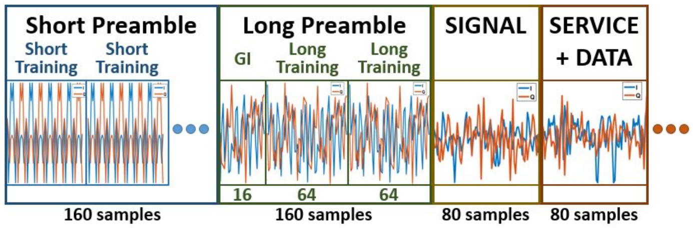


Fig. 1: 802.11a Preamble

layer covering types of messages and contention, and the application layer handling the top-level purpose. 802.11a is designed specifically for the 5-6 GHz RF bandwidth and expects data rates of 6-54 Mbit/s. To meet real-time constraints, an 802.11a transceiver must complete its PHY-layer transactions before the end of a fixed period or else suffer unacceptable data losses. These transactions include scrambling, convolutional encoding, block interleaving, phase shift keying (PSK) modulation, symbol-to-subcarrier mapping, and orthogonal frequency division multiplexing (OFDM) modulation. The 802.11a transceiver system consists of a transmitter (Tx) that perform these transactions and a receiver (Rx) that undoes them.

The 802.11a specification provides an example for encoding a frame for the OFDM PHY in Annex G [11]. The Physical Layer Convergence Procedure (PLCP) maps the PHY Sublayer Service Data Units (PSDU) into a framing format suitable for transferring data, called a PLCP Protocol Data Unit (PPDU). Our initial work properly generates and decodes the frame described in Annex G.

PLCP Preamble: The PLCP Preamble consists of 10 repetitions of a short training sequence and 2 repetitions of a long training sequence, as shown in Fig. 1. The short sequence is used for AGC convergence, diversity selection, timing acquisition, and coarse frequency acquisition in the receiver. The long training sequence is used for channel estimation and fine frequency acquisition [11]. The PLCP preamble is the same for every PPDU frame. The short preamble consists of 12 OFDM subcarriers, while the long preamble consists of 53. The total length of the preamble is 16 μs.

Scrambling: The scrambler performs a bitwise XOR with incoming data and a bit sequence randomly generated using a linear feedback shift register (LFSR). The scrambler LFSR uses the generator polynomial in equation 1.

$$S(x) = x^7 + x^4 + 1 \tag{1}$$

This creates a bit sequence that repeats every 127 bits. Since an XOR is used, the same sequence is used to descramble data at the Rx. The SIGNAL field sent as the first transmitted symbol is not scrambled, but all data bits are. The scrambling sequence changes depending on what seed value is used. Our implementation uses the same seed value as the example in Annex G of the spec, which is `1011101`.

Convolutional Encoding: The convolutional encoder uses generator polynomials of $g_0 = 133$ and $g_1 = 171$. These correspond to a rate 1/2 code with maximum free distance for $K = 7$. The output sequence has a bit length of twice the input length for this 1/2 rate. Note that 802.11a also supports rates 2/3 and 3/4 [11].

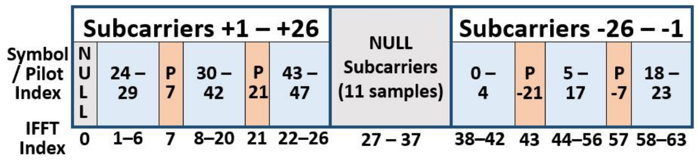


Fig. 2: Symbol to Subcarrier Mapping Diagram

Block Interleaving: Data interleaving is a two-step permutation performed on coded data. The first permutation maps adjacent bits to nonadjacent subcarriers, and the second permutation ensures adjacent coded bits are mapped alternately to avoid long runs of low reliability bits. Since we focus on BPSK with 1/2 code rate, this configuration simplifies the interleaving to equation 2.

$$\begin{aligned} i &= 4(k \mod 16) + \left\lfloor \frac{k}{16} \right\rfloor \\ j &= i + (i + 48 - \left\lfloor 16\frac{i}{48} \right\rfloor) \mod 16 \end{aligned} \tag{2}$$

where k is the index before the first permutation, i is the index after the first permutation, and j is the index after the second permutation. The number of coded bits per subcarrier, N_{CBPS}, is 1, and the number of coded bits per symbol, N_{CBPS}, is 48. Using this equation, the one-step interleaving permutation can be calculated beforehand.

PSK Modulation: PSK modulation is done to convert the input data to complex symbols. The specification defines that the OFDM subcarriers be modulated using BPSK, QPSK, 16-QAM, or 64-QAM. We built our implementation for BPSK, which has a constellation with points at -1 and 1.

Symbol-to-Subcarrier Mapping and Pilot Insertion: Next, the 48 complex symbols are mapped to subcarriers and combined with a set of 4 pilots whose polarity changes based on frame count. The 52 subcarriers are arranged in a specific order for the 64-point Inverse Fast Fourier Transform (IFFT), and nulls (zeros) are placed in the empty locations. Fig. 2 illustrates the mapping of input symbols, pilots, and null samples to IFFT inputs.

OFDM Modulation with Cyclic Prefix: The OFDM Modulator combines a 64-point IFFT and cyclic prefix attachment, in which the last 16 samples in time are prepended to the IFFT output [11]. The cyclic prefix is added to reduce inter-symbol interference and lower the effects of multipath fading by creating a Guard Interval (GI).

4 Target Hardware and Software

4.1 Hardware Setup

Our HW consists of an RF front end, the ADI FMComms3 board; a Xilinx Zynq evaluation board; and a host computer. We attach an Ethernet cable and a JTAG cable between the host PC and the Zynq board and connect the FMComms3 board to the FPGA Mezzanine Card (FMC) slot on the Xilinx FPGA board. The FMComms3 features the wideband wireless AD9361 transceiver chip. The AD9361 transceiver supports up to 2 transmit (Tx) and 2 receive (Rx) channels at bands from 70 MHz to 6.0 GHz [16]. The Xilinx Zynq SoC has an embedded ARM processor and FPGA fabric.

2168-6750 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2017.2651054, IEEE Transactions on Emerging Topics in Computing

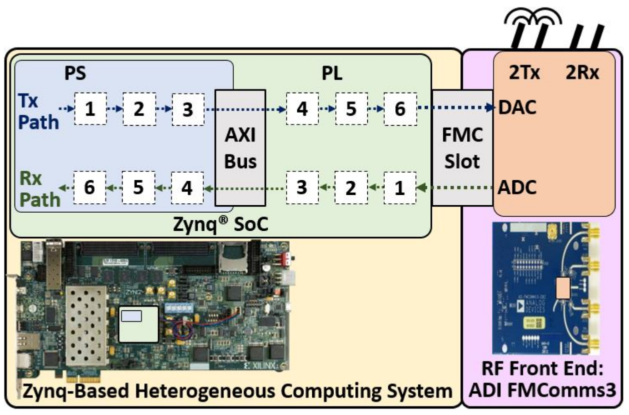

PS
PL
Tx Path
1
2
3
AXI Bus
4
5
6
DAC
FMC Slot
2Tx
2Rx
Rx Path
6
5
4
3
2
1
ADC
Zynq® SoC
Zynq-Based Heterogeneous Computing System
RF Front End: ADI FMComms3

Fig. 3: 802.11a Transceiver HW: Zynq SoC & FMComms3

TABLE 2: Zynq Board Comparison

	Zedboard	ZC706
Device	Z-7020	Z-7045
FPGA	Artix-7	Kintex-7
LUTs	53,200	218,600
Registers	106,400	437,200
DSP Slices	220	900
BRAM Blocks	140	545

Throughout this paper we use Xilinx terminology: Processing System (PS) for the ARM processor and Programmable Logic (PL) for the FPGA fabric. In our experiments, we target two different Zynq boards: the ZC706, which contains a Xilinx Z-7045 chip and the Zedboard, which contains a slightly less capable Z-7020. These boards are compared in Table 2. Internal to the Zynq processor, an Advanced eXtensible Interface (AXI) bus connects the PL and the PS. We use the ethernet cable to send start and stop signals from host PC to Zynq PS. The 802.11a transceiver system contains both a Tx path, from PS to PL to FMComms3, and a Rx path, from FMComms3 to PL to PS, as shown in Fig. 3.

4.2 Software Tools

To target the hardware described above, we use commercially available tools from MathWorks and Xilinx as illustrated in Fig. 4. We use MathWorks Simulink to create and simulate synchronous dataflow models. Specialized toolboxes from MathWorks, HDL Coder and Embedded Coder, allow us to target the PL and PS, respectively. Additional MathWorks hardware support packages allow us to interface with the Zynq SoC and the ADI FMComms3 [27].

We start by making a Simulink model to capture all the information about the Zynq transceiver system. In this model, we set radio parameters such as sampling frequency and number of samples per frame. We distinguish one subsystem in the model to target for execution on the PL, presuming that all the other model components are targeted to run on the PS. We run the HDL Workflow Advisor wizard to auto-generate an IP Core block for the Tx or Rx design under test (DUT). The wizard auto-generates a Vivado block diagram to combine the DUT with all the AXI interface components. The wizard creates a *generated model* to interact with the PL via AXI. Then, the wizard invokes Xilinx Vivado

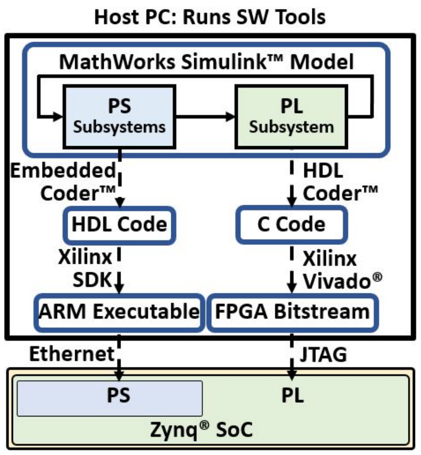


Fig. 4: High-Level SW Tool Workflow for Zynq PL & PS

from the command line to synthesize, implement, and make a bitstream [28]. We program the PL with this bitstream.

Finally, we generate an executable for the PS using MathWorks tools. By setting the generated Simulink model to run in External mode, Simulink uses Embedded Coder to generate C code for all processing blocks in the Simulink model [29]. Then, Simulink invokes Xilinx SDK to package and compile the executable for the PS [28]. When we press the play button in the Simulink model, it sends a signal via Ethernet to launch the executable on the PS.

5 Transceiver HW-SW Co-design

5.1 Design Variants

Fig. 5 and Fig. 6 show the processing chains for transmitter and receiver respectively. We have functionally equivalent software (PS) and FPGA hardware (PL) versions of each of the blocks in these figures. For each processing chain, we have explored HW-SW codesign by creating a number of Tx and Rx variants that each implement a different number of functional blocks in HW and SW. Data movement as well as processing is an important consideration when deciding which processing block to put in HW or SW. Each design variant moves data once between PL and PS. The reason that there is only one HW/SW divide point is that transferring data between computing elements adds additional overhead that we want to minimize. In future experiments, we may incorporate more divide points.

For the transmitter, each design variant adds one new block in FPGA hardware. V1 has the complete processing in software on the ARM processor. Subsequent designs assign one or two additional components to the PL, starting with the component closest to the RF board. Seven different versions were explored, where V7 has the entire Tx chain implemented on the PL. These versions are identified in Fig. 5. Specifically, V1 is a SW-only design that implements all functional blocks on the ARM processor. V2 moves the preamble insertion onto the PL. The preamble insertion block is placed at the end of the transmit processing chain, just before the RF front end. We apply OFDM modulation to the preamble beforehand and store the processed data in a lookup table. V3 adds the IFFT and cyclic prefix attachment components to HW. V4 adds the BPSK Modulation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2017.2651054, IEEE Transactions on Emerging Topics in Computing

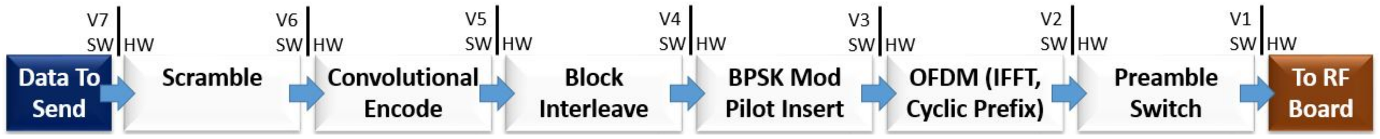


Fig. 5: 802.11a Transmitter Chain HW/SW Codesign Variants

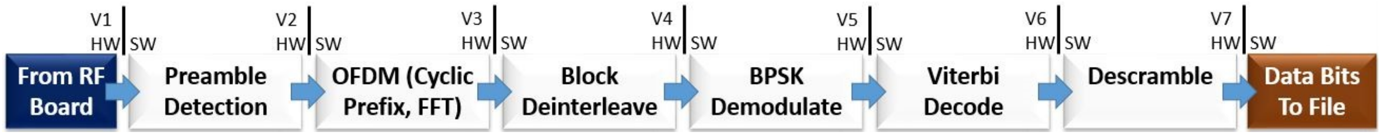


Fig. 6: 802.11a Receiver Chain HW/SW Codesign Variants

and Symbol-to-Subcarrier mapping components to HW. V5 adds the Block Interleaving component to HW. V6 adds the Convolutional Encoder component to HW. V7 is a HW-only design that only does file I/O on the PS and performs all processing on the PL fabric.

A similar approach is taken for the receiver model, for which we also model seven variants, as shown in Fig. 6. The PS-Only design is designated V1, and PL implementation versions increase incrementally from there. V2 adds the Preamble Detection component to reconfigurable HW. The preamble detection method uses a matched filter block to efficiently correlate two frames of fixed-point input samples with the expected long preamble sequence. Each subsequent version from V3 to V7 adds an additional component of the frame recovery subsystem to the PL. V3 adds OFDM modulation to the PL, including cyclic prefix removal and FFT. The FFT block adds a latency of 159 samples before the output is valid. V4 adds BPSK demodulation and subcarrier-to-symbol mapping to the PL. It uses a delay line to gather the 64 valid samples and a selector to reorder them and remove the pilot and empty guard subcarriers. V5 adds block de-interleaving to the PL using a selector. V6 adds Viterbi decoding, which introduces a delay of two frames. V7 adds the Descrambler component to the PL. We evaluate all these different versions with respect to timing, resource utilization and energy efficiency.

5.2 Timing Considerations

In order to maintain a real-time transceiver system and prevent unacceptable data losses, we need to closely monitor execution times. For simplicity and to meet SW tool requirements, we instantiate a *fixed step time* for both PL and PS. Since operations on the PL fabric can operate many times faster than on the PS, we design for the PL to process one sample at a time and the PS to process one frame at a time. The time per frame, t_f, is therefore the product of the number of samples per frame, n_{spf}, and the time per sample, t_s, as shown in equation 3.

$$t_f = t_s n_{spf} \tag{3}$$

By setting the appropriate step times in the model variants, we can ensure that data is transferred between PL and PS at the desired rate. However, we must be careful to ensure that all the processing operations in the PL subsystem complete within the sample time, t_s. If they do not, then we run the risk of *underflow*, where new received RF bits are lost at the FMComms3 Analog to Digital Converter (ADC) ports, or zero bits must be sent at the Digital to Analog Converter (DAC) ports. In addition, we must also be careful to ensure that all the processing operations on the PS complete within t_f. If they take longer, then the PS would lose data sent from the PL or fail to send data to the PL in time, causing issues with data integrity. A summary of the relevant timing variables is listed in Table 3.

TABLE 3: Timing Details

	Variable	Rx Value	Tx Value
PL Step Time/Sample Time	t_{ps}	12.5 μs	1 μs
# Samples per Frame	n_{spf}	80	80-403
PS Step Time/Frame Time	t_{pf}	1 ms	80-403 μs

According to the IEEE 802.11a specifications for the lowest bit rate, each OFDM symbol represents 24 data bits [11]. If we assign only one OFDM symbol to a frame, each frame has 24 data bits. After convolutional coding, this becomes 48 coded bits. After BPSK modulation and mapping, this becomes 64 symbols. Finally, after OFDM modulation, this becomes 80 time samples per frame. For the Rx, we decide to have the PS process one frame at a time every 1 ms, and to have the PL process one sample every 12.5 μs. In contrast, the Tx models are designed to use a fixed sample time of 1 μs and increase the number of samples per frame, n_{spf}, for each design variant. Thus, the Tx model variants have an increasing PS frame time of between 80 and 403 μs.

5.3 Common System Components

For all the design variants, there were several system components that are used consistently. These components are necessary for implementing a wireless behavior in HW that has equivalent functionality to the SW version. Before sending data between the PS and the PL, multiple inputs (e.g. data, validIn, and reset signals) must be packed on one end and unpacked at the other end. This packing consists of concatenating inputs into 32-bit unsigned integer for the Advanced eXtensible Interface (AXI) interconnect. A system reset signal is packed into the AXI input, and pulsed once at the start. We use sample and frame counters to handle logic specific to a sample or frame number.

2168-6750 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2017.2651054, IEEE Transactions on Emerging Topics in Computing

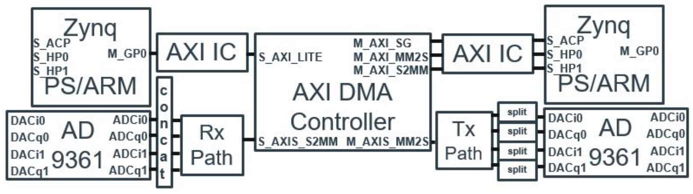


Fig. 7: AXI Connections between Zynq PS and AD9361 Chip

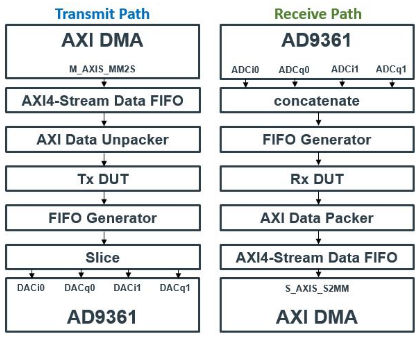


Fig. 8: AXI-Stream Transmit and Receive Path Detail

6 Experimental Results

As described in Section 4 our target hardware consists of a board containing a Xilinx Zynq chip, an interface to an ADI FMComms RF front end, and a host PC. The FMComms board makes use of an AD9361 chip. Internally, communications on the Zynq chip uses an AXI interconnect, which is used to transfer 32-bit words in a time-synchronous manner between PL and PS. There are two AXI interfaces which we use: AXI-lite is a memory mapped protocol and AXI-Stream is intended for high-speed streaming data. We use AXI-lite for the Rx and AXI-stream for Tx. To support the AXI-stream interface, the Vivado block diagram must contain both AXI Interconnect and AXI Direct Memory Access (DMA) Controller IP Cores as shown in Fig. 7. To retrieve RF data bits from the FMComms3 ADC ports, the in-phase and quadrature (I&Q) bits are concatenated for both channels, processed through the Rx path, and sent to the DMAC AXI slave interface. To transmit data bits on the FMComms3 DAC ports, the bits travel from the DMAC AXI master interface, through the Tx path, and are split into I&Q components for each channel. Detailed diagrams of the Tx path and Rx path are shown in Fig. 8.

6.1 Timing Results

For the 802.11a Tx, the execution timing results on the PS are shown in Fig. 9. The maximum PS frame time decreases as more components are moved onto the PL. Moving the IFFT to PL in V3 results in the largest drop in frame time. Also, the ZC706 frame time of 55 μs is significantly lower than the Zedboard frame time. While the maximum frame time on the ZC706 does not decrease between V3 and V6, we have seen that the V7, the PL-only version, exhibits the lowest maximum and average frame time on both the Zedboard and the ZC706. To meet the 802.11a specifications, we would need to meet a 4 μs maximum frame time. Thus, further optimization is needed to reduce the PS frame time.

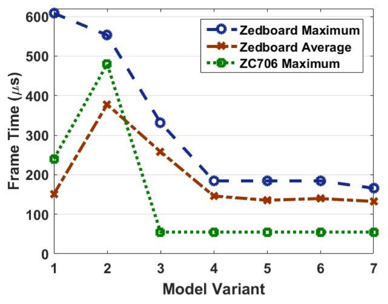


Fig. 9: Transmitter Frame Times on Zedboard & ZC706

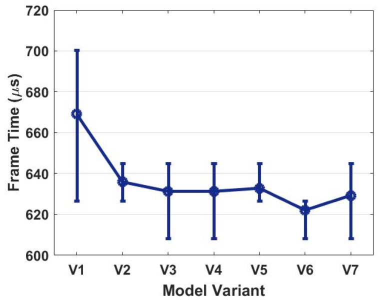


Fig. 10: Receiver Frame Times on ZC706

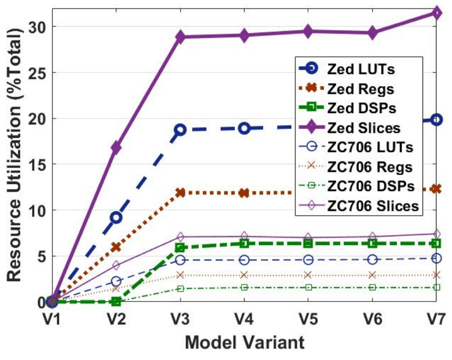
TABLE 4: Data Path Delay on ZC706 PL

	Tx (ns)	Rx (ns)
V1	n/a	n/a
V2	11.11	313.70
V3	16.17	317.43
V4	18.33	311.14
V5	15.84	313.12
V6	16.52	307.73
V7	16.04	318.89

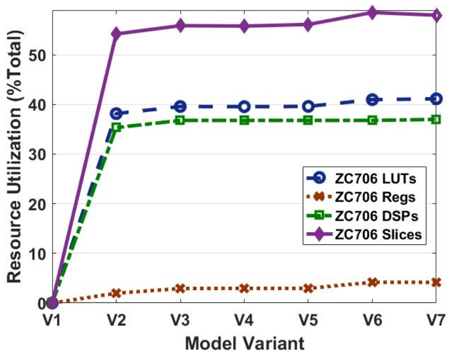
For the Rx, the execution timing results on the PS are shown in Fig. 10. Similar to the Tx, the Rx maximum PS frame time decreases as more components are moved onto the PL. Moving the preamble detection to PL in V2 results in the largest drop in frame time, but there are also significant drops when the FFT is moved in V3 and the Viterbi Decoder is moved in V6. Notably, moving the Descrambler component to PL in V7 does not show a decrease in frame time, suggesting that it may be better placed in SW.

For an idea of how long the same operations take to process on the PL, we look for the maximum data path delay of the Tx and Rx, which are shown in Table 4. Since the FPGA implementation is inherently parallel, at under 320 ns, the Rx PL delay is faster than any SW implementation. This data path delay indicates that the Rx path on the PL can definitely keep up with the Tx, whose sample time is currently set to 1 μs. Our real challenge is meeting the 802.11a specification, for which the Rx PL sample time would need to be 50 ns. However, by implementing preamble detection in a different way and reducing the size of the matched filter, we could reduce the number of data dependencies, thereby decreasing the path delay significantly.

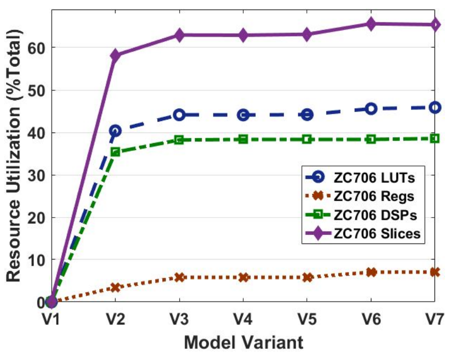
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2017.2651054, IEEE Transactions on Emerging Topics in Computing



(a)



(b)



(c)

Fig. 11: Resource Utilization for (a) Tx on Zedboard, (b) Rx on ZC706, (c) Combined Tx-Rx on ZC706

6.2 Resource Utilization

Our Tx design can be accommodated on either the ZC706 or the Zedboard, although only the ZC706 has sufficient resources for the Rx implementation. The Tx resource utilization results are shown in Fig. 11a. These results show increasing lookup table (LUT), register, and digital signal processor (DSP) usage as more components are put onto the PL. The number of registers decreases slightly from V2 to V3 due to the different data types involved. The slice registers hold state information that reduces because V2 must transfer data in 32-bit sample form, while V3 holds data in single-bit form. V2 must hold each sample in complex, 16-bit fixed-point format before initiating IFFT processing, and 64 data samples make up a frame. In all model versions, even the PL-only variant, the FPGA is at less than 5% utilization on the ZC706 and 20% on the Zedboard, meaning that it retains many LUTs and registers for use by prospective component variations (e.g. QPSK), higher OSI layers and other designs.

The Rx resource utilization on the ZC706 is shown in Fig. 11b. Like the Tx, these Rx utilization results show increasing lookup table (LUT), register, and digital signal processor (DSP) usage as more components are put onto PL. The largest increase comes from the initial placement of preamble detection on the PL in V2. Note that the Rx uses a significant portion of the FPGA resources, with as much as 60% of the total slices, the main grouping of logic resources. Still, we see that there remain many LUTs and registers for use by higher OSI layers or other designs.

A combined Tx and Rx design could be implemented on the ZC706 or more powerful boards. Such a combined design would be appropriate for a modern bidirectional transceiver, since even a designated Tx must have an Rx component to receive ACKs. The combined Tx and Rx resource utilization is shown in Fig. 11c.

6.3 Power Efficiency

In addition to meeting timing and resource requirements, we are also interested in generating power efficient designs. Since the Zynq PS is based around an embedded ARM processor designed for low power, it is more power efficient than some alternative processors such as TI6670 DSP used in Atomix, which consumes 5-8 W [4]. The Zynq platform always provides power to the ARM processor; thus, using FPGA fabric adds to the overall power consumption. The Xilinx Zynq SoC we use has 7-series FPGAs, which consume less power than the Virtex-4 FPGA in WARP or the Virtex-5 in Sora [30]. The FPGA power consumption is related to the SoC chip area and resource utilization; hence, each version of our Tx and Rx designs that puts another block onto FPGA fabric increases overall power consumption. Xilinx Vivado offers synthesis options for speed or area optimization that we plan to explore in future work. The power results were derived by running the Vivado Power Report with fixed environmental settings (e.g. output load 5 pF, ambient temperature 25 °C). The Tx and Rx power consumption on the Zedboard and ZC706, respectively, are shown in Table 5. Our results show that the FPGA fabric is more power efficient than the ARM processor because each power increase is only a fraction of the ARM power in V1.

TABLE 5: Power Usage, Tx on Zedboard, Rx on ZC706

	Tx (W)	Tx (ΔV1)	Rx (W)	Rx (ΔV1)
V1	1.530	0	1.566	0
V2	1.819	0.289	2.343	0.777
V3	1.840	0.310	2.354	0.788
V4	1.845	0.315	2.111	0.545
V5	1.844	0.314	2.106	0.540
V6	1.847	0.317	2.111	0.545
V7	1.842	0.312	2.115	0.549

The Tx total power increases from 1.530 to 1.842 Watts as more components are placed on the PL. However, this increase of 312 mW is small when compared to the Tx PS consumption, which alone is 1.53 W on the Zedboard. As expected, the power increases as more components are added to the PL, most notably AXI in V2 and IFFT in V3.

The Rx total power also increases as more blocks are put on the PL, most notably AXI and preamble detection in V2 and FFT in V3. However, we see a significant decrease when BPSK is placed on the PL in V4. The reason is the data type change from samples to coded bits. Whereas V3 transfers 64 32-bit fixed point samples from PL to PS, V4 only transfers 48 bits packed into 2 32-bit integers. Thus, the load on the AXI interconnect is reduced by a factor of 32. From V4 to V7, the Rx power increases only 4 mW, which is minor compared to the ZC706 PS consumption of 1.566 W.

2168-6750 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2017.2651054, IEEE Transactions on Emerging Topics in Computing

TABLE 6: Preamble Detection Matched Filter Variants

	Default	HDL Long	HDL Training
Data Path Delay (ns)	499.6	313.7	131.6
%LUTs	8.89	38.16	15.77
%Registers	4.34	1.96	1.26
%DSPs	99.22	35.33	14.67
Total Power (W)	2.65	2.34	2.09

TABLE 7: Viterbi Decoder Variants

	Delay-Based	BRAM-Based
Data Path Delay (ns)	307.73	314.45
%LUTs	40.97	40.34
%Registers	4.17	3.24
%DSPs	36.78	36.78
#BRAM Tiles	0	2
Total Power (W)	2.358	2.357
Viterbi Power (W)	0.011	0.005

6.4 Variants of Processing Blocks

Next, we focus on two components that consume many resources, preamble detection and Viterbi decoding, and explain the details and tradeoffs associated with the design of each.

6.4.1 Preamble Detection

Our preamble detection method uses a matched filter block to efficiently correlate two frames of fixed-point input samples with the expected long preamble sequence. When the complex magnitude exceeds a predefined normalized threshold, a flag is set to identify that the preamble was found. In addition, the index of maximum correlation is used by a selector block to choose which sample in the delayed frame is first OFDM demodulated.

Our modeling environment aided in the identification of preamble detection as a major source of path delay and resource utilization. Thus, we prototyped different versions of the preamble detection processing block for variant V2, which use different algorithms for the matched filter (MF) component. These variants are shown in Table 6.

The first MF variant was manually assembled from the default components, which are a delay line and an array of multipliers and adders for each received sample. Since this default version auto-generates HDL for each individual multiplier and adder, it is not HDL optimized and it is very inefficient. The Vivado synthesis process used over 99% of the DSPs for it, and it has a very long data path delay. Using the HDL-optimized MF with the full long preamble was therefore preferable. However, since the long preamble is composed of repetitions of a shorter training sequence, we found the best results using this training sequence for the MF coefficients instead. The HDL-optimized *training* MF showed a 2.38X reduction in data path delay over using the long preamble, as well as a 1.12X reduction in power and a smaller number of LUTs, registers, and DSPs utilized.

The modeling environment shows value for highlighting that preamble detection is a bottleneck. In addition, the resource utilization analysis identifies that the Zedboard can now be used for the Rx chain in addition to the ZC706. In the original *long* versions of the design, due to the large number of LUTs and DSPs needed, we were forced to use the ZC706. However, using the *training* version uses only a fraction of those LUTs and DSPs, meaning that the resources available on the Z-7020 SoC are sufficient for implementing all model variants, even the HW-only design.

6.4.2 Viterbi Decoder

The Viterbi decoder processing block reverses the effects of the convolutional encoder by calculating maximum likelihoods. Since the decoding is based on probabilities, it requires a delay of a few dozen samples before it can produce valid output data bits. Since it requires memory to hold intermediate state values, an implementation may use different resources to accomplish this, either by use of registers or block RAM (BRAM).

Since Viterbi decoding was also revealed to be a source of delay and resource usage, we prototyped different versions of the Viterbi Decoder (VD) processing block for model variant V6, which are shown in Table 7. By using the default delay-based version, we observed the lowest data path delay. However, using the version that holds state memory in BRAM uses fewer LUTs and registers, which slightly lowers the overall power consumption. By looking specifically at the power consumed by the VD block, we see a power reduction of 6 mW. Thus, swapping the VD variant illustrates a tradeoff between time and power that can be dynamically tuned for either objective. This brand of adaptability is most useful when there are few of one resource available, and switching the implementation of a processing block would be beneficial for utilizing less of the overused component (e.g. LUTs or registers) and utilizing more of an underused component (e.g. BRAM).

6.5 Next Generation Enhancements

Having demonstrated the capability of our modeling environment to prototype the 802.11a processing chain, we explore extending the design to research areas of interest to the next generation wireless community. In particular, we show how our modeling environment is suited for exploring such issues as protocol coexistence and multiple-input, multiple-output (MIMO) operation.

6.5.1 LTE / Wi-Fi Coexistence

The reusability inherent in our modeling environment allows us to explore LTE and Wi-Fi coexistence on the same channel. The IEEE 802.11a standard provides the functional basis for IEEE 802.11g, the protocol used by Wireless Firewall (Wi-Fi) devices. As an initial study into this topic, we note that OFDM is used by both protocols. However, to support OFDM for both protocols would require different IFFT sizes and cyclic prefix lengths, as well as flexible subcarrier allotments to form OFDMA ('MA' in the acronym implies the addition of *multiple access*) used in the downlink channel for LTE. Our modeling environment can easily modify processing blocks to prototype the different settings for each standard. For example, we can vary the IFFT sizes required by LTE OFDM and collect metrics to identify the impact of the different size, as shown in Table 8. The results

2168-6750 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2017.2651054, IEEE Transactions on Emerging Topics in Computing

TABLE 8: OFDM Block IFFT Size Variants

IFFT Size	**64**	**128**	**256**	**512**	**1024**
Data Path Delay (ns)	15.15	16.76	16.81	15.64	17.99
%LUTs	19.86	22.38	27.8	37.18	54.86
%Registers	12.33	14.47	19.09	27.65	44.11
%DSPs	6.36	7.73	9.09	10.45	11.82
Total Power (W)	1.842	1.841	1.849	1.854	1.872

TABLE 9: All-HW Model MIMO Variants

# Tx/Rx chains	1 Tx	2 Tx	1 Rx	2 Rx
Data Path Delay (ns)	10.63	11.37	25.07	25.27
% LUTs	1.89	4.06	7.15	13.87
% Registers	1.14	2.11	3.44	6.84
% DSPs	1.44	1.78	25.78	51.56
% Slices	3.18	6.48	12.24	22.04
Total Power (W)	1.930	1.938	2.128	2.321
PD Power (W)	n/a	n/a	0.183	0.359

show increases in data path delay, resource utilization, and power for rising IFFT sizes.

Considering the case of LTE, larger amounts of control flow exist here compared to 802.11. Presuming this control flow exhibits a large amount of divergence, it may be better placed on the PS. This would require more communication from PS to PL to administer functional changes, and introduces *multiple* HW-SW divide points. In this case, while the streaming data is best suited for AXI-streaming transfers, we may reserve AXI-lite channels for handling infrequent control messages from the PS to the PL.

6.5.2 MIMO Spatial Diversity

The IEEE 802.11n standard describes the extension of 802.11g for MIMO. Since the ADI FMComms3 supports MIMO with 2 transmit channels and 2 receive channels, our platform allows for further exploration of spatial diversity. By using multiple antennas, we can experiment with transmitting and receiving identical sequences, which can be used at the receiver to overcome fading and interference.

To prototype spatial diversity as a basis for future experiments, we must first recognize that some elements of the receive chain are ill-suited for replication. Simply attempting to copy the original preamble detection component multiple times easily overwhelms the FPGA resources, surpassing the number of available slices. However, using the reduced preamble detection method described in 6.4.1, we can accommodate multiple receive chains on the FPGA. We modified model V7 for both the transmitter and the receiver, and capture the results in Table 9.

The results show that multiple transmit and receive chains can be implemented on FPGA fabric with only minor changes to data path delay. Duplicating the preamble detection (PD) block for the receive chains doubles the number of DSPs and slices used, as well as the power for that processing block. However, the total power only increases by a fraction. By using multiple antennas and transmitting identical sequences, we can next experiment with using alternate encoding or modulation techniques for each channel and enable further evolution towards the MIMO functionality described in 802.11n and 802.11ac.

7 Discussion

7.1 Reusability for Wireless Studies

A major benefit of our flexible SDR testbed is the ability to reuse components for alternate 802.11 and mobile standards. A comparison of the protocol settings in several modern 802.11 and LTE-based cellular standards is given in Table 10. The functional blocks of our 802.11a implementation, especially those concerning scrambling and block interleaving, can be re-used in a number of different standards. However, some modifications would need to be made to support different convolutional encoding rates besides 1/2 and digital modulation schemes besides BPSK. This reusability allows us to explore LTE and Wi-Fi coexistence on the same channel, TV whitespace reuse, or co-operation with RADAR, and also allows the same SDR hardware to switch between access standards by downloading only the additional functional blocks and retaining the common ones.

In addition, the use of reconfigurable HW allows us to explore methods for optimal subcarrier selection. Rather than mapping modulated symbols to a fixed set of subcarriers, the wireless transceiver system can dynamically assign symbols to specific subcarriers that have been identified to have maximum channel efficiency.

7.2 Optimization Considerations

Developers familiar with Simulink may expect the slow execution times associated with running Simulink models on a host PC in Normal mode. However, this expectation is not reality in our modeling environment. Since our generated models run in External mode, C code is generated and compiled to an executable and the executable is run on the ARM processor. The Simulink model, running on the host PC, only uses the start and stop buttons to send a signal to the executable running on the ARM to begin or end. Optimization techniques for the Zynq ARM processor are not necessarily ideal for an FPGA implementation, and vice-versa. For this reason, Simulink libraries include alternate versions of blocks for either destination. For example, the FFT algorithm is handled by the *FFT* block in SW, or by the *FFT HDL Optimized* block in HW. Both blocks show improvements in new releases. In R2016a, the latter block has reduced latency for vector inputs.

Although our initial foray into HW-SW codesign uses only the built-in Simulink blocks, we plan to experiment with alternate custom implementations for the processing blocks that show the longest data path delays. MathWorks tools allow developers to create their own Simulink blocks, via either a Level-2 S-function for incorporating custom C code or a *Black Box Interface* for incorporating custom HDL code. The latter option would allow us to incorporate IP cores from Xilinx for LTE downlink.

While some algorithms can be optimized to work well for a specific protocol, these may also prohibit flexibility with other protocols. As an example, consider the Schmidl-Cox algorithm for preamble detection with the 802.11a

2168-6750 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2017.2651054, IEEE Transactions on Emerging Topics in Computing

TABLE 10: Wireless Standard Block Comparison: (1) Implemented & Reusable, (2) Not Yet Implemented, but Reusable

	802.11a	**802.11b**	**802.11g**	**802.11p**	**802.11n**	**802.11ac**	**802.11ad**	**LTE**	**LTE-A**
Scrambling	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**
Convolutional Coding									
1/2 Rate	**(1)**		**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**
2/3 Rate	(2)		(2)	(2)	(2)	(2)		(2)	(2)
3/4 Rate	(2)		(2)	(2)	(2)	(2)	(2)		
Digital Modulation									
BPSK	**(1)**	(D)	(D)	**(1)**	**(1)**	**(1)**	($\pi/2$)		
QPSK	(2)	(D)	(D)	(2)	(2)	(2)	($\pi/2$)	(2)	(2)
16-QAM	(2)			(2)	(2)	(2)	($\pi/2$)	(2)	(2)
64-QAM	(2)			(2)	(2)	(2)		(2)	(2)
Direct Seq Spread Spectrum		(2)	(2)						
Block Interleaving	**(1)**		**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**
OFDM	**(1)**		**(1)**	(2)	(2)	(2)	(2)	(DL)	(DL)
IFFT Size	**64**	n/a	**64**	**64**	64,128	64-512	512	128-2048	128-2048
Cyclic Prefix (μs)	**0.8**	n/a	**0.8**	1.6	0.8,0.4	0.8,0.4		4.69-33.33	4.69-33.33
Preamble Detection	**(1)**	(2)	(2)	(2)	(2)	(2)	(2)	(2)	(2)

preamble. This algorithm has been shown to be optimal for preambles that consist of a repeating training sequence, but not others. In contrast, our incorporation of a simple matched filter for this purpose could be used to detect any sort of preamble for various protocols with only minor model modification. The benefits of our modeling environment are that all of these aforementioned topics can be explored, which very few SDR alternatives are capable of doing. In future work, we plan to explore these optimization methods further.

8 Conclusion

We have introduced and explored a method for exploring HW-SW co-designs for 802.11a wireless transmission and reception systems. We have shown that for direct feedthrough algorithms, moving more components to execution in HW results in faster execution speed, but adds the risk of overwhelming FPGA resources. Moreover, while energy consumption increases as more components are placed on the programmable logic, the amount is negligible when compared to the embedded ARM energy consumption. We show that many of the components developed for this base design can be reused for prototyping MIMO, other variants of 802.11 such as Wi-Fi, and LTE protocols.

In the future, we plan to perform tests with online radio transmissions and measure bit error rate (BER) for the different co-designs. This 802.11a PHY layer implementation will also be used as a basis for future work in higher layers (e.g. MAC). We plan to expand upon our investigation of MIMO to prototype spatial multiplexing and beamforming. In addition to fully exploring the design space, the system will be adapted to other modern wireless standards such as 802.11ac for beamforming, 802.11af for UHF band reuse, LTE, and 5G.

Acknowledgments

The authors would like to thank Xilinx, Inc. for their donation of Zynq evaluation boards and software, Analog Devices, Inc. for their donation of FMComms boards, and MathWorks, Inc. for their donation of MATLAB software licenses and generous research funding. We would also like to thank Carlos Bocanegra Guerra for his contributions.

References

[1] J. M. Bradley. (2013) The Internet of Everything: Creating Better Experiences in Unimaginable Ways. Cisco Systems. [Online]. Available: http://blogs.cisco.com/digital/the-internet-of-everything-creating-better-experiences-in-unimaginable-ways#more-131793

[2] J. Mitola III and G. Q. Maguire Jr., "Cognitive Radio: Making Software Radios More Personal," *IEEE Personal Commun.*, vol. 6, no. 4, pp. 13–18, 1999.

[3] I. F. Akyildiz, W.-Y. Lee, M. C. Vuran, and S. Mohanty, "NeXt Generation/Dynamic Spectrum Access/Cognitive Radio Wireless Networks: A Survey," *Computer Networks*, vol. 50, no. 13, pp. 2127–2159, 2006.

[4] M. Bansal, A. Schulman, and S. Katti, "Atomix: A Framework for Deploying Signal Processing Applications on Wireless Infrastructure," in *12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 15, Oakland, CA, USA, May 4-6*, 2015, pp. 173–188.

[5] A. Dutta, D. Saha, D. Grunwald, and D. Sicker, "CODIPHY: Composing On-demand Intelligent Physical Layers," in *Proceedings of the second workshop on Software radio implementation forum*. ACM, 2013, pp. 1–8.

[6] WARP Project, Rice University. (2015) Wireless Open-Access Research Platform. [Online]. Available: http://warp.rice.edu/index.php

[7] K. Tan, J. Zhang, J. Fang, H. Liu, Y. Ye, S. Wang, Y. Zhang, H. Wu, W. Wang, and G. M. Voelker, "Sora: High Performance Software Radio Using General Purpose Multi-core Processors," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI, April 22-24, Boston, MA, USA*, 2009, pp. 75–90.

[8] K. Vipin and S. A. Fahmy, "Mapping Adaptive Hardware Systems with Partial Reconfiguration using CoPR for Zynq," in *2015 NASA/ESA Conference on Adaptive Hardware and Systems, AHS 2015, Montreal, QC, Canada, June 15-18, 2015*, 2015, pp. 1–8.

[9] M. C. Ng, K. E. Fleming, M. Vutukuru, S. Gross, Arvind, and H. Balakrishnan, "Airblue: A System for Cross-layer Wireless Protocol Development," in *Proceedings of the 2010 ACM/IEEE Symposium on Architecture for Networking and Communications Systems, ANCS 2010, San Diego, California, USA, October 25-26*, 2010, p. 4.

[10] J. van de Belt, P. D. Sutton, and L. Doyle, "Accelerating Software Radio: Iris on the Zynq SoC," in *21st IEEE/IFIP International Conference on VLSI and System-on-Chip, VLSI-SoC 2013, Istanbul, Turkey, October 7-9*, 2013, pp. 294–295.

2168-6750 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TETC.2017.2651054, IEEE Transactions on Emerging Topics in Computing

[11] *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: High-speed Physical Layer in the 5 GHz Band*, IEEE Std. 802.11a-1999.

[12] G. Stewart, M. Gowda, G. Mainland, B. Radunovic, D. Vytiniotis, and D. Patterson, "Ziria: Language for Rapid Prototyping of Wireless PHY," in *ACM SIGCOMM 2014 Conference, SIGCOMM'14, Chicago, IL, USA, August 17-22*, 2014, pp. 357–358.

[13] J. Pendlum, M. Leeser, and K. Chowdhury, "Reducing Processing Latency with a Heterogeneous FPGA-Processor Framework," in *22nd IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2014, Boston, MA, USA, May 11-13*, 2014, pp. 17–20.

[14] National Instruments, Inc. (2016) Real-time LTE/Wi-Fi Coexistence Testbed. [Online]. Available: http://www.ni.com/white-paper/53044/en/

[15] Ettus Research, Inc. (2016) USRP Networked Series. [Online]. Available: https://www.ettus.com/product/category/USRP-Networked-Series

[16] Analog Devices, Inc. (2015) Integrated Transceivers, Transmitters, and Receivers. [Online]. Available: http://www.analog.com/en/products/rf-microwave.html#integrated-transceivers-transmitters-receivers

[17] GNU Radio Project. (2015) GNURadio: The Free and Open Source Radio Ecosystem. [Online]. Available: http://www.gnuradio.org

[18] MathWorks, Inc. (2016) USRP Support from Communications System Toolbox. [Online]. Available: http://www.mathworks.com/hardware-support/usrp.html

[19] B. Drozdenko, R. Subramanian, K. Chowdhury, and M. Leeser, "Implementing a MATLAB-Based Self-configurable Software Defined Radio Transceiver," in *Cognitive Radio Oriented Wireless Networks - 10th International Conference, CROWNCOM 2015, Doha, Qatar, April 21-23, Revised Selected Papers*, 2015, pp. 164–175.

[20] P. Murphy, A. Sabharwal, and B. Aazhang, "Design of WARP: A Wireless Open-access Research Platform," in *Signal Processing Conference, 14th European*. IEEE, 2006, pp. 1–5.

[21] M. Duarte, A. Sabharwal, V. Aggarwal, R. Jana, K. Ramakrishnan, C. W. Rice, and N. Shankaranarayanan, "Design and Characterization of a Full-duplex Multiantenna System for WiFi Networks," *Vehicular Technology, IEEE Transactions on*, vol. 63, no. 3, pp. 1160–1177, 2014.

[22] A. Makki, A. Siddig, M. Saad, C. Bleakley, and J. Cavallaro, "High-resolution Time of Arrival Estimation for OFDM-based Transceivers," *Electronics Letters*, vol. 51, no. 3, pp. 294–296, 2015.

[23] R. Marlow, C. Dobson, and P. Athanas, "An Enhanced and Embedded GNU Radio Flow," in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*. IEEE, 2014, pp. 1–4.

[24] C. Dobson, K. Rooks, and P. M. Athanas, "A Power-efficient FPGA-based Self-adaptive Software Defined Radio," in *24th International Workshop on Power and Timing Modeling, Optimization and Simulation, PATMOS), Palma de Mallorca, Spain, September 29 - Oct. 1*, 2014, pp. 1–8.

[25] B. Özgül, J. Langer, J. Noguera, and K. A. Vissers, "Software-programmable Digital Pre-distortion on the Zynq SoC," in *21st IEEE/IFIP International Conference on VLSI and System-on-Chip, VLSI-SoC 2013, Istanbul, Turkey, October 7-9*, 2013, pp. 288–289.

[26] S. Shreejith, B. Banarjee, K. Vipin, and S. A. Fahmy, "Dynamic Cognitive Radios on the Xilinx Zynq Hybrid FPGA," in *Cognitive Radio Oriented Wireless Networks - 10th International Conference, CROWNCOM 2015, Doha, Qatar, April 21-23, Revised Selected Papers*, 2015, pp. 427–437.

[27] MathWorks, Inc. (2016) SDR Support from Communications System Toolbox. [Online]. Available: http://www.mathworks.com/hardware-support/zynq-sdr.html

[28] Xilinx, Inc. (2016) Vivado Design Suite - HLx Editions. [Online]. Available: http://www.xilinx.com/products/design-tools/vivado.html

[29] MathWorks, Inc. (2016) Embedded Coder Support Package for Xilinx Zynq-7000 Platform. [Online]. Available: www.mathworks.com/help/supportpkg/xilinxzynq7000ec

[30] Xilinx, Inc. (2016) Xilinx 7 Series FPGAs: Breakthrough Power and Performance, Dramatically Reduced Development Time. [Online]. Available: http://www.xilinx.com/publications/prod_mktg/7-Series-Product-Brief.pdf

Benjamin Drozdenko is a Ph.D. candidate and graduate research assistant at Northeastern University, co-advised by Prof. Chowdhury and Prof. Leeser. His dissertation research is about enabling protocol coexistence via HW-SW codesign of wireless transceivers on heterogeneous architectures. Ben previously worked for MathWorks, Inc. as a Signal Processing Content Specialist and Training Engineer, and for Raytheon as a Systems Engineer for ground-based Radar.

Matthew Zimmermann earned his M.S. in Electrical and Computer Engineering in 2016 from Northeastern University, where he completed his Master's project under Prof. Leeser, using Simulink to test designs on a Xilinx Zynq Evaluation Board. He also has been working at the MITRE Corporation since 2007 as a Hardware Engineer, where he develops FPGA-based Software Defined Radios. He earned his B.S. at Worcester Polytechnic Institute.

Tuan Dao is a combined BS/MS student in Computer Engineering at Northeastern University. He joined Prof. Leeser's Reconfigurable and GPU Computing Lab in 2016, where he works as an undergraduate research assistant. Earlier, he worked as an undergraduate research assistant at NUCAR, helping with the development of Hetero-Mark, a benchmark suite for heterogeneous computing systems based on OpenCL.

Kaushik Roy Chowdhury is an Associate Professor in the Electrical and Computer Engineering Department at Northeastern University, where he was an Assistant Professor from 2009-2015. He received his Ph.D. from Georgia Institute of Technology in August 2009. He is the winner of the NSF CAREER award in 2015. He is a Sr. Member of the IEEE. He serves as area editor for the journals Elsevier Ad Hoc Networks, Elsevier Computer Communications, and EAI Transactions on Wireless Spectrum.

Miriam Leeser is a Professor in Electrical and Computer Engineering at Northeastern where she leads the Reconfigurable and GPU Computing Laboratory. She received her Diploma and PhD degrees in computer science from Cambridge University in England. She is a senior member of the IEEE and of the ACM. She is an associate editor of ACM Transactions on Reconfigurable Technology and Systems, EURASIP Journal on Embedded Systems, and the International Journal on Reconfigurable Computing.

2168-6750 (c) 2016 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.