Enabling Protocol Coexistence: Hardware-Software Codesign of Wireless Transceivers on Heterogeneous Computing Architectures

A Dissertation Presented
by

Benjamin Drozdenko

to

The Department of Electrical and Computer Engineering

in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy

in

Computer Engineering

Northeastern University
Boston, Massachusetts

April 2017

To Xavier & Kaushallya.

Contents

List of Figures

List of Tables

Acknowledgments

Here, I wish to thank those who have supported me during the process of the dissertation work. I would like to thank Professor Miriam Leeser for her guidance throughout the Ph.D. process and her support in my role as the MathWorks TA. I thank Professor Kaushik Chowdhury for his guidance and support in my role as a Graduate Research Assistant for multiple semesters and for his advice on pursuing a tenure-track assistant professor position. I thank Professor Stefano Basagni for his support as a member of my dissertation committee and for his company at the 2016 FGRE Summit in Ghent, Belgium. I thank Xilinx and Analog Devices for their hardware donations. I thank MathWorks for their research funding and donation of MATLAB licenses. I want to thank the Northeastern University Provost's office for awarding me with the Dissertation Completion Fellowship to conclude my research writing in the Spring 2017 semester.

I'd also like to express gratitude to the many research assistants in my laboratory who have helped me or provided me with emotional support during my stint as a full-time Ph.D. student. First, I'd like to thank Yousof Naderi for his support in structuring my presentations and writing proposal materials. I thank Ramanathan Subramanian for his help on the MathWorks USRP-based 802.11b project and publishing the IEEE Access and CrownCom papers. I thank the undergraduate students Taylor Skilling and Eric Doyle for providing support on this project. I thank Matt Zimmermann and Tuan Dao for their assistance on the Zynq-based 802.11a project and in publishing the IEEE TETC and FPL 2016 papers. I thank Rahman Doost-Mohammady and Majid Sabbagh for their collaboration on the Xilinx Zynq SoC, Vivado block diagrams, and ADI LibIIO workflow. I thank Carlos Bocanegra Guerra for contributing information to compare 802.11 and LTE protocols. I thank Fan Zhou, Meenupriya Swaminathan, and Takai Eddine Kennouche for our collaborations, especially during Infocom 2016. Most recently, I thank Suranga Handagala for his contribution of hardware-friendly matched filtering designs and his help on the LTE/Wi-Fi Coexistence project and in publishing the FPL 2017 paper. I thank Jieming Xu and Mehmet Gungor for their collaboration on Simulink and wireless topics. I thank Prof. Leeser's other students, especially Jonathon Pendlum, Mahsa Bayati, Xin Fang, Chao Liu, and Janki Bhimani, for attending my practice presentations and providing me feedback. I also thank Prof. Chowdhury's other students, including Ufuk Muncuk, William Tomlinson, Subhramoy Mohanti, Kübra Alemdar, and Kunal Sankhe for attending my presentations and for their company on group outings.

Most of all, I'd like to express appreciation my wife, Kaushallya Adhikari, and my son, Xavier Adhikari Drozdenko, for their moral support throughout my Ph.D. career. I could never have done it without you!

Abstract of the Dissertation

Enabling Protocol Coexistence: Hardware-Software Codesign of Wireless Transceivers on Heterogeneous Computing Architectures

by

Benjamin Drozdenko

Doctor of Philosophy in Computer Engineering

Northeastern University, April 2017

Dr. Miriam Leeser, Advisor

In an increasingly interconnected world, there has been an explosion in the number of wireless devices in the Internet of Things. This recent increase in wireless devices has been accompanied by a rising number of protocols for wireless communications, each focusing on different purposes such as execution time reduction, energy reduction, handling higher congestion levels, or operation at different bandwidths. This increase has also caused heavy congestion on particular bandwidths. Due to spectrum scarcity, the need has arisen for these devices to operate on the same bandwidths. However, existing wireless devices are inflexible and have no capabilities to coexist with devices using other protocols. Software-Defined Radios (SDR) have introduced new platforms for dynamically modifying wireless system designs, and heterogeneous computing has enabled implementation on different computing elements. Until now, researchers have focused on designing complete protocol-specific processing chains on static computing architectures. However, SDR has opened the door for flexibility in wireless transceivers, and heterogeneous computing systems can be used to meet the needs for lower execution time and power consumption.

This dissertation introduces new Field Programmable Gate Array (FPGA)-based design techniques to receive multiple protocols on the same computing platform. Our methods incorporate tunable parameters, such as FIR filter length and number of bits per fixed-point word, to explore design tradeoffs regarding clock cycle, resource utilization, power consumption, and detection accuracy. This research separates the physical (PHY) layer receive chains into a set of building blocks, including rate transition, pattern detection, and Orthogonal Frequency Division Multiplexing (OFDM) demodulation. This research introduces a practical resampling technique to accommodate several protocol rates while taking FPGA resource utilization into account. Having identified pattern detection as a time critical operation that needs to be performed with sufficient accuracy to avoid

triggering false alarms, this research incorporates a hardware-friendly matched filtering technique to reduce FPGA utilization while maintaining detection accuracy. This research introduces a technique to use a single FFT to handle different numbers of frequency subcarriers. The implementation of the LTE physical downlink shared channel (PDSCH) and 802.11a protocols are implemented to test our techniques. LTE is the standard for high-speed wireless communication for mobile phones and data terminals; Wi-Fi uses variants of the IEEE 802.11a standard. To ease the system development process, this research develops models using MathWorks Simulink, which supports auto-generation of Hardware Description Language (HDL) code for the non-critical sections and incorporation of hand-tuned HDL code as part of its black box interface. These building blocks can be used by the wireless system modeling community to meet the needs of modern evolving wireless standards. This FPGA-based design framework for protocol coexistence is tested to support the receiver chains of 802.11a and LTE downlink protocols. FPGA-specific component designs are provided to support each protocol, including rate transition, matched filtering, and OFDM demodulation. This framework also provides a workflow for tuning parameter settings, such as filter length and fixed-point word size. In the future, this framework will allow researchers to achieve high-performance transceiver implementations on FPGA fabric for multiple cutting edge protocols.

Chapter 1

Introduction

In recent years, the field of wireless technology has seen a tremendous surge in the diversity of devices, protocols, and applications. 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. The current standard for wireless laptop communications is known as Wireless Firewall (Wi-Fi), using the IEEE 802.11g standard. This protocol has evolved to support multiple antenna operation in 802.11n, low power consumption in 802.11ah, and higher bit rates in 802.11ax. The latest standard for mobile phone technology, devised by the 3rd generation partnership project (3GPP), is known as long-term evolution (LTE). The current variation of LTE technology in place is the 4th generation (4G), and many research projects are in place to prototype and test the 5th generation (5G) technology.

These trends in wireless communications and networking have been accompanied by new technologies in the fields of reconfigurable and heterogeneous computing. The field programmable gate array (FPGA) has emerged as the premier technology to design and prototype fast, cycle-accurate electronic systems that can be reprogrammed to meet the needs of evolving applications. For this reason, FPGAs are worthy candidates for wireless applications, in which both timing accuracy and modifiable functionality are requirements. More recently, the System-on-Chip (SoC) has emerged as a platform for experimenting with heterogeneous systems, which contain more than one computing element (CE). SoCs that combine a central processing unit (CPU) with an FPGA, such as the Xilinx Zynq, have been introduced which ease the transfer of data between CEs. These SoCs allow researchers to better explore on which CEs certain behaviors are best placed, a topic known as hardware-software co-design.

1.1 Research Challenges

The increase in the sheer number of devices has introduced several issues in wireless networking, including increased congestion, spectrum scarcity, larger data sizes, and increased demands for energy. Modern-day wireless communications standards are constantly evolving to meet the needs of an increasing number of devices. Each protocol introduces new challenges that need to be addressed, including ways to address contention in an overcrowded wireless spectrum. The advent of big data and the emergence of streaming applications have brought about a need for sending more data in a shorter amount of time. For this reason, modern protocols must support higher bit rates and lower error rates. Also, wireless mobile devices must be recharged infrequently; hence, devices must evolve to consume less energy.

The problem of spectrum scarcity has caused the FCC to open up new bandwidths for use by multiple protocols. Statistics have shown that there are large differences in utilization of spectrum bandwidths. While the bandwidths using LTE are frequently overutilized and congested, bandwidths reserved for other purposes may often be underused. For these reasons, spectrum is scarce and the unlicensed bands are an extremely valuable commodity. To help resolve this, the FCC has opened up previously-reserved licensed bandwidths for reuse, such as the TV whitespace and military RADAR bandwidths. The presence of multiple protocols on the same bandwidth is a new scenario that FPGAs can be instrumental in prototyping.

Although many people may think of 2.4 and 5.8 GHz bands as Wi-Fi bands, they're really open to any industrial, scientific, and medical (ISM) purposes. For this reason, mobile phone technology wants to use these ISM bands for control if presently unoccupied to boost coverage in their cellular networks. Companies have proposed LTE protocol variants to accommodate this, such as LTE-U and MulteFire by Qualcomm and License Assisted Access (LAA) by Ericsson. Cellphone carriers such as T-Mobile and Verizon have indicated interest in deploying this idea as early as possible. Modern smartphones can communicate using both LTE and Wi-Fi protocols, but they incorporate separate chips to handle each protocol. Each chip is designed to use a different fixed frequency bandwidth for transception. A novel idea in the case of one connection outage, would be to switch to the other protocol. However, there are many challenges of coexistence between Wi-Fi and LTE. First, there is the challenge of synchronization, the detection of a fixed pattern in either standard and arrangement of clocks for proper timing. Whereas Wi-Fi uses OFDM, the LTE downlink uses OFDMA (multiple access). At the MAC layer, Wi-Fi uses the distributed coordination function (DCF), while LTE allows for flexible resource allocation using either frequency division

duplexing (FDD) or time division duplexing (TDD). Wi-Fi manages contention using carrier sensing multiple access with collision avoidance (CSMA/CA), while LTE base stations, or eNodeB's handle this using subchannel allocation. Thus, given their differences, there are outstanding questions about how a device could handle multiple protocols.

Designing wireless transceivers to meet the modern needs of networking raise multiple issues in the realm of electronic system-level (ESL) design, including inflexible hardware (HW), insufficiently fast software (SW), the need for lower-level design knowledge, and generality. In the past, wireless transceivers have typically been made with static HW, in which the protocol specifications could not be modified. In recent years, 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 [13]. Recent research has explored such problems as spectrum scarcity by sensing shared bandwidths and switching center frequency as needed [14].

A SDR system requires a radio frequency (RF) front end and processing elements to encode, modulate, and recover signals. SDRs are typically made with lower-end processors, which alone are too slow to fully handle all the physical (PHY) layer processing blocks at the required sampling rates. For this reason, some SDRs have adopted heterogeneous computing architectures, which consist of multiple unlike computing elements. Of particular note is the system-on-chip (SoC), which consists of a field-programmable gate array (FPGA) and a reduced instruction set computing (RISC) processor. However, the design of FPGA logic is difficult; designers need experience with hardware description language (HDL) or similar knowledge to program FPGAs. Additionally, published FPGA designs do not often take into account the FPGA's interfaces with the processor and with the RF front end HW. A final issue concerning ESL design is that existing HW-SW codesign environments do not take into account the issues specifically related to wireless communications and transception.

Various testbeds that have been introduced to prototype real-time, online SDR transceiver systems fall short, failing to address all of these issues. Each testbed has limitations that prevent its widespread adoption and make it insufficient for the study of protocol coexistence in particular. Not all SDR testbeds combine a SW processor component with reconfigurable HW for the most time-sensitive tasks. Some SDR testbeds, such as WARP [15] and Sora [16], do not allow the developer to modify the HW component, or require the developer to use predefined SW routines. Very few SDR testbeds present the developer with a high-level interface for designing both SW and HW components. Even fewer testbeds release their designs for use by the general public.

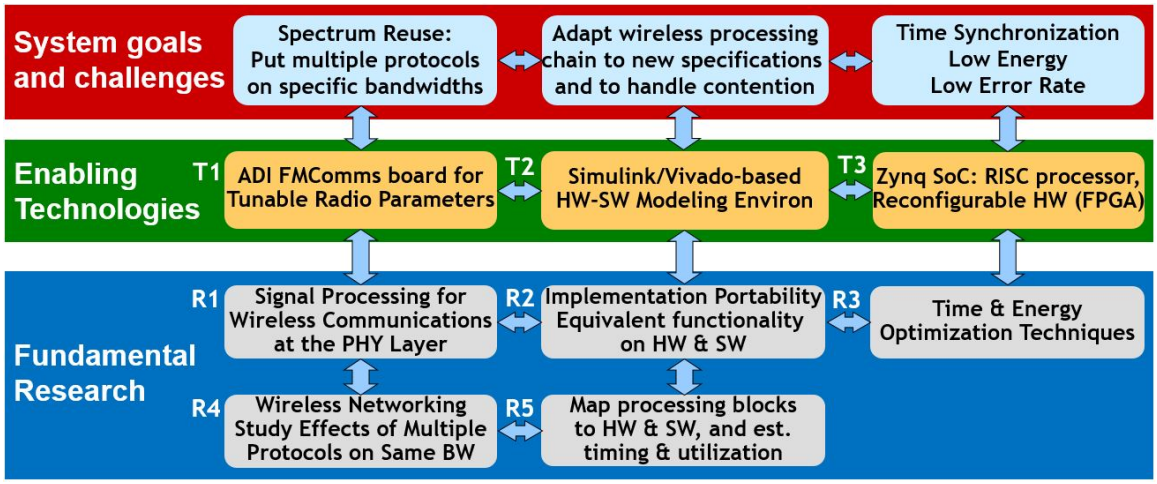


Figure 1.1: Proposal Challenges, Technologies, and Research Areas

1.2 Organization and Contributions

This dissertation introduces a new method for exploring wireless protocol coexistence on the same spectrum bandwidth, focusing specifically on the Wi-Fi and LTE protocols. It considers the main challenges, technologies, and research topics shown in Fig. 1.1. To be specific, I address topic R4 in wireless networking, to study the effects of multiple protocols on the same BW. To enable this research, I make use of enabling technologies T2 and T3, a Simulink & Vivado-based modeling environment for implementation on the Zynq SoC.

First, I propose an FPGA-based approach to model the receiver chains for both 802.11a and LTE protocols, starting with their lowest-layer physical (PHY) receiver chains for one antenna. I identify and propose methods for handling the major issues associated with dual-protocol support, including rate transition, joint pattern detection, and OFDM demodulation with different numbers of subcarriers.

This approach advances the state of the art by providing the following contributions:

- An FPGA-based design framework for protocol coexistence, supporting the receiver chains of multiple protocols, tested with 802.11a and LTE.
- FPGA-specific component designs to support each protocol, including rate transition, matched filtering, and OFDM demodulation.

- A workflow for tuning parameter settings, such as filter length and fixed-point word size, and gleaning relevant results, such as utilization.

The platform introduced for this study will provide a means for researching the issues regarding protocol coexistence at the PHY layer and can be adapted to use RF hardware.

The remainder of this dissertation is organized as follows. Chapter 2 discusses similar research that has been undertaken in SDR and provides a background of PHY layer wireless communications algorithms. This chapter discusses HW and SW used for general purpose SDR, high-level SDR design frameworks, and SDR platforms based on CPU, FPGA, and SoC. It provides detail about the PHY layer processing blocks that comprise the 802.11a standard, focusing on the preamble, coding, and modulation schemes used by this protocol. It describes and defineds terminology used by the LTE standard, focusing on the downlink (DL) and the DL shared channel (DL-SCH).

Chapter 3 describes the methods that were employed for designing PHY layer wireless system. To clarify the types of systems used for prototyping, the following terms are used:

- *Online*: Refers to a wireless system design that transmits and receives RF electromagnetic signals using a wireless transceiver chip.
- *Offline*: Refers to a wireless system design that does not use a wireless transceiver chip.
- *SW-based Design*: A system design in which almost all processing is performed on CPU.
- *HW-based Design*: A system design in which almost all processing is performed on FPGA.
- *HW/SW Codesign*: A system design in which the processing is partially performed on FPGA, and partially performed on CPU.

In Sec. 3.1, an online 802.11b PHY layer wireless system is prototyped using Ettus Research USRP N210s and a host PC. In Sec. 3.2, an offline 802.11a PHY layer wireless system is prototyped in a cycle-approximate manner using the Xilinx Zynq SoC heterogeneous system. In Sec. 3.3, an offline FPGA-centric approach to accommodate multiple protocols is prototyped using the Zynq SoC.

Chapter 4 describes the experiments performed and the results attained to assess the quality of the wireless systems and their effectiveness in utilization of resources and accuracy. In Chapter 5, I summarize my dissertation and provide some research topics that I plan to address in my future research.

Chapter 2

Background

For a deeper understanding of the challenges inherent in the creation of a cycle-accurate transceiver system and work already undertaken by the research community in this area, this chapter provides background information about commonly used platforms for study in wireless networking topics and details about the most commonly used wireless standards, 802.11 and LTE.

2.1 SDR Related Work

2.1.1 SDR Hardware

Any SDR implementation that can support physical radio transmissions requires a radio frequency (RF) front end. Analog Devices Inc. (ADI) manufactures integrated circuit chips for RF transception, including the wideband wireless AD9361 and AD9364 transceivers [17]. ADI also introduced the AD-FMComms series, which attach to the FPGA Mezzanine Card (FMC) slot on Xilinx FPGA and Zynq System-on-Chip (SoC) boards. 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 that uses ADI transceiver chips [18].

2.1.2 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 [19]. Its graphical interface, GNU Radio Companion, allows the user to build block diagrams to represent complex

Table 2.1: Comparison of features for different SDR systems and solutions

Features	ATOMIX	CODIPHY	WARP	Sora-Ziria	CoPR	Airblue	Iris	CRASH	NI LTE/WiFi Testbed	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								✓		✓

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. The RF Network-on-Chip (RFNoC) architecture provides an extension to targeting the FPGA on Ettus USRPs [20]. However, this environment requires that the user manually specify which algorithms go on HW and SW.

MATLAB and Simulink are also widely used tools for modeling algorithms in digital signal processing systems. As the basis for communications system design, MathWorks produces HW support packages for interfacing with commonly-used RF front-ends, including for Zynq-Based Radio and USRP-based Radio [21]. For example, in [22], a high-level cognitive radio framework is designed for bidirectional transception using the USRP N210, MATLAB, and IEEE 802.11b.

2.1.3 SDR High-Level Design Frameworks

An overview of SDR systems and solutions is shown in Table 2.1. 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 [23]. 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 [24]. However, the testing behind CODIPHY is all emulated, not tested on live FPGAs or SoCs.

2.1.4 SDR Platforms with CPU and FPGA

Some SDR projects are implemented in both HW and SW on a heterogeneous platform that comprises processor, FPGA, and often custom-built 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 [15] [25]. WARP has been used to build a full duplex 802.11 network with OFDM and a MAC protocol [26], and an algorithm for estimating time-of-arrival for OFDM-based transceivers [27]. 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 [16]. 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 [28]. 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 [29]. Airblue introduces an FPGA-based SDR platform for the PHY and MAC layers [30]. However, this platform does not include a SW processor and so cannot be used for studying HW-SW co-design issues.

2.1.5 SDR Platforms on Xilinx Zynq SoC

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 [31], SDR is modeled using GNU radio adaptations for Zynq and Zynq clustering. In [32], 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 [33]. 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 [34]. However, these

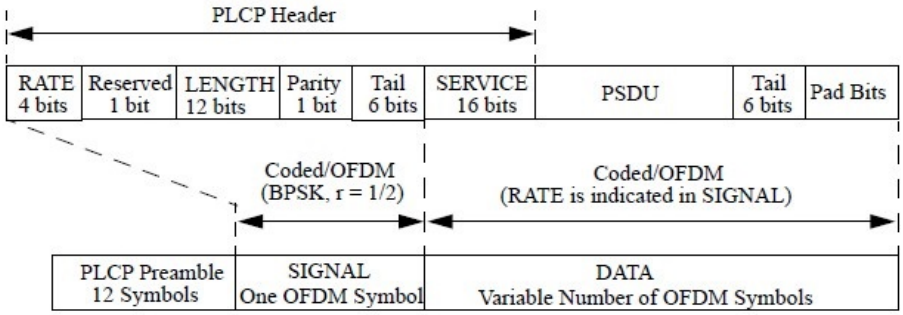


Figure 2.1: PPDU Frame Format [1]

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 [35]. In [36], a CR platform using the Zynq with partial reconfiguration and the ADI FMComms4 with tunable operating frequency is proposed to enable dynamic, low-power, high-performance cognitive radio with abstracted software control. However, this latest work has not been fully implemented and tested.

2.2 IEEE 802.11 Standard Specifications

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) 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 [1]. Our initial work produced a MATLAB program to properly generate and decode the transmitted frame as described in Annex G. The Physical Layer Convergence Procedure (PLCP)

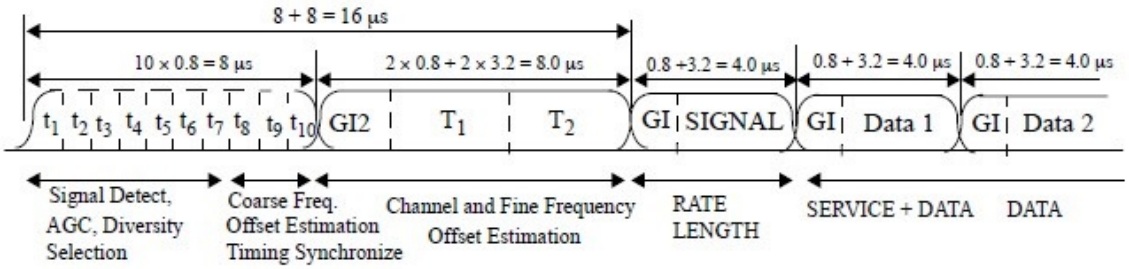


Figure 2.2: 802.11a Preamble [1]

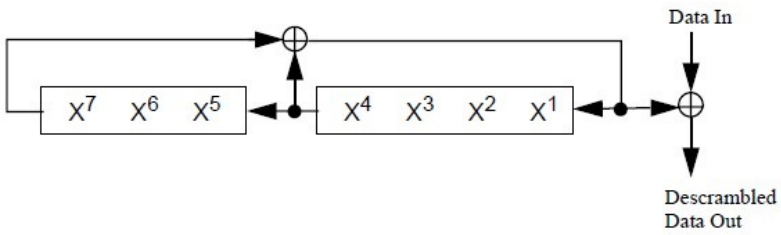


Figure 2.3: 802.11a Scrambler [1]

maps the PHY Sublayer Service Data Units (PSDU) into a framing format suitable for transferring data, called a PLCP Protocol Data Unit (PPDU), as shown in Fig. 2.1.

2.2.1 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. 2.2. 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. 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. In full, the total length of the preamble is 16 μs.

2.2.2 Scrambling

The scrambler performs a bitwise XOR with incoming data and a bit sequence randomly generated using a linear feedback shift register (LFSR), as shown in Fig. 2.3. The scrambler LFSR uses the

generator polynomial in equation 2.1.

$$S(x) = x^7 + x^4 + 1 \tag{2.1}$$

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`.

2.2.3 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. 802.11a supports rates 2/3 and 3/4 as well, and these rates can be achieved by puncturing the output, or by omitting some encoded bits to increase the coding rate [1].

2.2.4 Block Interleaving

Data interleaving is a two-step permutation performed on coded data. The first permutation maps adjacent bits to nonadjacent subcarriers using equation 2.2.

$$\begin{aligned} i = \frac{N_{CBPS}}{16}(k \mod 16) + \left\lfloor \frac{k}{16} \right\rfloor \\ k = 0, 1, ..., N_{CBPS} - 1 \end{aligned} \tag{2.2}$$

where k is the index before the first permutation, i is the index after the first permutation, and N_{CBPS} is the number of coded bits per symbol. The second permutation ensures adjacent coded bits are mapped alternately to avoid long runs of low reliability bits, and is defined by equation 2.3.

$$\begin{aligned} j = s \left\lfloor \frac{i}{s} \right\rfloor + (i + N_{CBPS} - \left\lfloor 16 \frac{i}{N_{CBPS}} \right\rfloor) \mod s \\ i = 0, 1, ..., N_{CBPS} - 1 \\ s = \max \frac{N_{BPSC}}{2}, 1 \end{aligned} \tag{2.3}$$

where j is the index after the second permutation and N_{BPSC} is the number of coded bits per subcarrier. These equations vary based on modulation and code rate. However, since I focus on

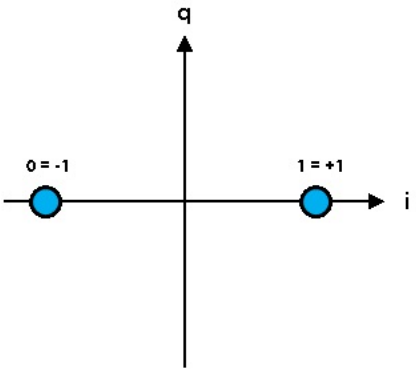


Figure 2.4: BPSK Constellation [1]

BPSK with 1/2 code rate, then $N_{CBPS} = 48$, $N_{BPSC} = 1$, and $s = 1$, which simplifies the interleaving to equation 2.4.

$$\begin{aligned} i &= 4(k \mod 16) + \left\lfloor \frac{k}{16} \right\rfloor \\ j &= i + (i + 48 - \left\lfloor 16\frac{i}{48} \right\rfloor) \mod 16 \end{aligned} \tag{2.4}$$

Using this equation, the one-step interleaving permutation can be calculated beforehand.

2.2.5 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. I built our implementation for BPSK, which exhibits the constellation shown in Fig. 2.4.

2.2.6 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. 2.5 illustrates the mapping of input symbols, pilots, and null samples to IFFT inputs.

2.2.7 OFDM Modulation with Cyclic Prefix Attachment

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 [1]. The cyclic prefix is added to reduce

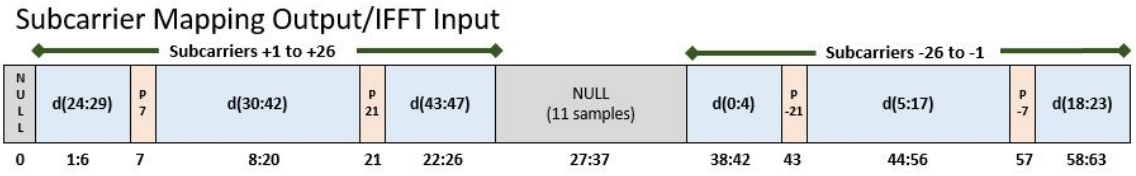


Figure 2.5: Symbol to Subcarrier Mapping Diagram

Figure 2.6: OFDM Modulated Data with Cyclic Prefix Attached

inter-symbol interference and lower the effects of multipath fading by creating a Guard Interval (GI), as shown in Fig. 2.6.

2.3 3GPP LTE Standard Specifications

Long Term Evolution (LTE) refers to the continuously changing standard for mobile and cellular communications. The first generation (1G) of cellular telephony was designed only for voice communication using analog signals. In North America, the Advanced Mobile Phone System (AMPS) used FDMA to separate channels and operated in the 800 MHz band. 1G started the trend of facilitating separate channels for the direction of communications, one *forward* or *downlink* (DL) from base station (BS) to mobile station (MS), and one *reverse* or *uplink* (UL) from MS to BS. UL communications operated on the 824-849 MHz band, and DL on the 869-894 MHz band. Each band was divided into 832 channels, with 416 available to each cell provider; of these 416, 395 were used for traffic and 21 for control. AMPS used the FSK modulation scheme for control and FM for voice channels [37].

The second generation (2G) was intended to provide higher quality mobile voice communications that would be less susceptible to noise. 2G was designed for digitized voice and evolved into three major systems, D-AMPS, GSM, and IS-95 CDMA. Digital AMPS (D-AMPS) and GSM were both combined TDMA-FDMA systems. D-AMPS was designed to be backward-compatible with AMPS and used the same bands as AMPS. It was defined by Interim Standard 54 (IS-54) first and IS-136 later. D-AMPS used TDMA to combine three 7.95-kbps digital voice channels. Digital data was modulated using a QPSK carrier and transmitted on a 25 MHz FDMA band. Global System

for Mobile Communication (GSM) was a European standard developed to act as a 2G standard to replace all incompatible 1G technologies in Europe. GSM used two 25-MHz bands for duplex communication, 890-915 and 935-960 MHz [37].

Third generation (3G) cellular telephony provides communication for both digital data and voice. he emphasis of the technology is high spectral efficiency, high peak data rates, low latency, and frequency flexibility. The LTE specifications were developed by the Third Generation Partnership Project (3GPP) [38]. GSM and UMTS are the predecessors of the LTE air interface and are referred to as second generation (2G) and third generation (3G) technologies, respectively. GSM was developed as a circuit switched network meaning that radio services are configured at the user's request and resources remain allocated until terminated by the network controller. This type of operation is well suited to supporting voice calls. Eventually, GSM was enhanced to support low data rate services with packet switching capability but data rates were limited by GSM's air interface, time division multiple access (TDMA). In TDMA, each user is assigned to a particular channel (frequency band) and time slot which serves to limit capacity as the channel spacing is only 200 kHz. UMTS uses code division multiple access (CDMA) as its air interface. In CDMA, active users transmit simultaneously over the allocated bandwidth, typically 5 MHz. Signals are separated from each other by the use of orthogonal variable spreading factor (OVSF) spreading codes. The advantage of OVSF codes is that resources can be allocated asymmetrically among the active users. UMTS supports both circuits switched services for voice calls and packet switched for data sessions. Due to its larger bandwidth and superior spectral efficiency, UMTS can support higher data rates than GSM [39] [2].

Unlike GSM and UMTS, LTE is a purely packet switched network, in which groups all transmitted data into suitably sized blocks, called packets. LTE uses orthogonal frequency division multiple access (OFDMA) in which the spectrum is divided into resource blocks (RB) that are composed of twelve 15 kHz subcarriers. By dividing the spectrum in this manner, channel equalization is simplified for mitigating frequency-selective fading. LTE supports higher order modulation schemes up to 64-QAM, and its bandwidth allocations can be as high as 20 MHz. In addition, LTE makes use of MIMO so that very high theoretical data rates can be achieved. For Release 8, the highest data rates possible are 75 Mbps in UL and 300 Mbps in DL [39] [2].

2G and 3G cellular networks are designed to interact with a number of players, or *users*. First, there is a radio network controller (RNC) that allocates radio resources among the users. Next, there is a base station (BS) that transmits signals to and receives signals from the users. In UMTS, this BS was called a Node B. In LTE, the enhanced Node B (eNodeB or eNB) combines the RNC

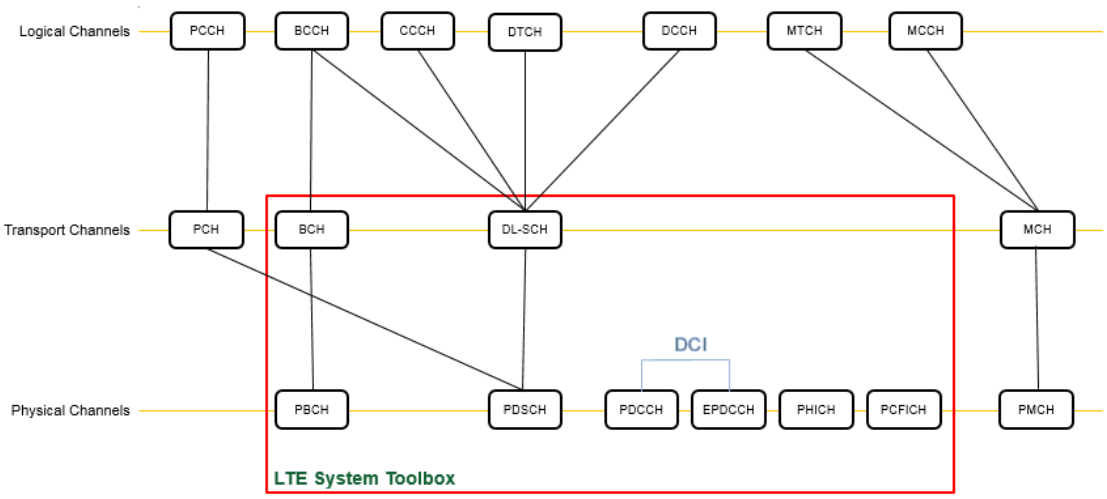


Figure 2.7: LTE Downlink Channels [2]

functionality with the Node B functionality. Finally, there are user devices; in GSM, the user device is called a mobile station (MS). In LTE and UMTS, it is called user equipment (UE) [39] [2].

The LTE radio access network is comprised of the following protocol entities.

- Packet Data Convergence Protocol (PDCP)
- Radio Link Control (RLC)
- Medium Access Control (MAC)
- Physical Layer (PHY)

Header compression, ciphering, segmentation and concatenation, and multiplexing are handled by the PDCP, RLC, and MAC protocols. The PHY layer includes encoding and decoding, modulation and demodulation, and antenna mapping. LTE PHY layer also includes OFDM modulation, including the time-frequency structure of the resource blocks, adaptive modulation and coding, hybrid-ARQ, and MIMO. System DL data follows the mapping between logical channels, transport channels, and physical channels indicated in Fig. 2.7 [2]. In comparison, system UL data follows the mapping indicated in Fig. 2.8 [2]. The portion outlined in red highlights the physical channels, transport channels, and control information on which I focus for the purposes of this research, using LTE System Toolbox [40] for modeling the functionality at a high level.

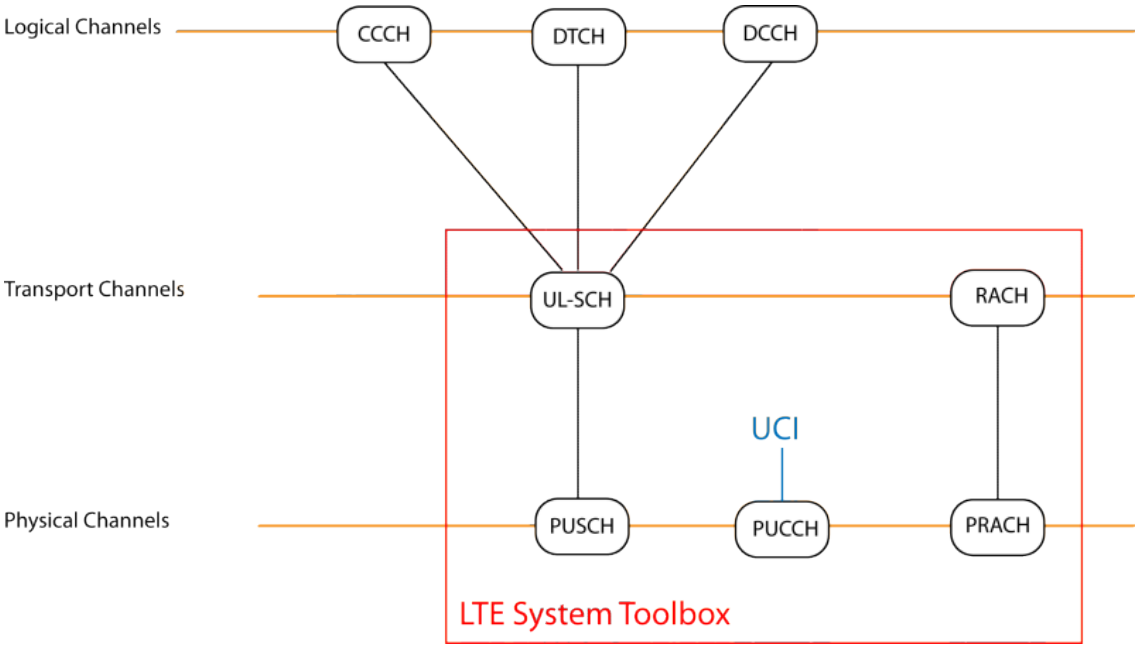


Figure 2.8: LTE Uplink Channels [2]

2.3.1 Resource Grids

LTE is intended to operate on any one of six bandwidths, from 1.4 to 20 MHz. Digitally modulated symbols must be mapped to an index in a matrix called a resource grid where the rows represent frequency subcarriers and the columns represent one OFDM symbol in time. A sample DL resource grids is shown in Fig. 2.9.

In LTE, each OFDM symbol can contain data from multiple physical channels and signals. The primary synchronization signal (PSS) and secondary synchronization signal (SSS) shown in dark blue are fixed sequences that can be used by the receiver for aligning the start of the signal, much like the 802.11a preamble. For all LTE DL signals, regardless of the channel bandwidth, the PSS and SSS always occupy the central 62 subcarriers in the inner 1.92 MHz band.

The user bit sequence to be transmitted is inserted into the downlink shared channel (DL-SCH), and then modulated and mapped into the physical downlink shared channel (PDSCH) shown in cyan. The number of LTE resource blocks, n_{RB}, each containing 12 subcarriers, can range from 6 to 100. Thus, the (I)FFT sizes needed for OFDM (de)modulation can range from 128 to 2048, as opposed to the 802.11a fixed FFT size of 64. Moreover, while the 802.11a CP length is fixed at 16 samples (0.8μs), the LTE CP length varies. On the first OFDM symbol in a time slot, the normal

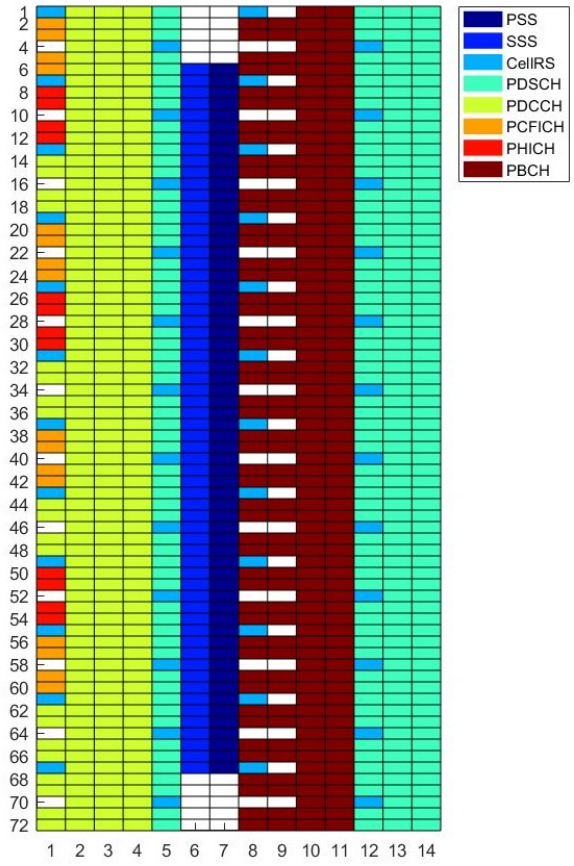


Figure 2.9: LTE Downlink Resource Grid Example

CP length ranges from 10 to 160 samples. In the remaining six OFDM symbols in a time slot, the normal CP length ranges from 9 to 144 samples. Note that LTE also specifies an extended CP option that is not modeled in this study [12].

2.3.2 Synchronization Signals

In LTE, there are two downlink synchronization signals which are used by the UE to obtain the cell identity and frame timing, the primary synchronization signal (PSS) and the secondary synchronization signal (SSS). The division into two signals is aimed to reduce the complexity of the cell search process. [3]. The physical cell identity, N_{ID}^{cell}, is defined by equation 2.5,

$$N_{ID}^{cell} = 3N_{ID}^{(1)} + N_{ID}^{(2)} \tag{2.5}$$

where $N_{ID}^{(1)}$ is the PHY layer cell identity group with a range of 0-167, and $N_{ID}^{(2)}$ is the identity within the group with a range of 0-2. Thus, there are 504 possible unique PHY cell identities.

2.3.2.1 Primary Synchronization Signal

The PSS is based on a frequency-domain Zadoff-Chu sequence, which are a construction of Frank-Zadoff sequences [41]. These codes are useful because they have a cyclic autocorrelation of one at zero lag, and zero at all nonzero lags. This sequence is ideal for pattern detection (a.k.a. synchronization) because the correlation between the ideal sequence and a received sequence is greatest at a lag of zero. When there is any lag between the two sequences, the correlation is zero [3].

The PSS is a sequence of complex symbols, 62 OFDM symbols long. The sequence, $d_u(n)$, used for the PSS is generated according to equation 2.6 [3].

$$\begin{aligned} d_u(n) &= e^{-\frac{j\pi un(n+1)}{63}}, \text{ for } n = 0, 1, ..., 30 \\ d_u(n) &= e^{-\frac{j\pi u(n+1)(n+2)}{63}}, \text{ for } n = 31, 32, ..., 61 \end{aligned} \tag{2.6}$$

where u is the Zadoff-Chu root sequence index, which depends upon the cell identity within the group $N_{ID}^{(2)}$. If $u = 0$, then $N_{ID}^{(2)} = 25$; if $u = 1$, then $N_{ID}^{(2)} = 29$; and if $u = 2$, then $N_{ID}^{(2)} = 34$. The PSS is mapped into the first 31 subcarriers on either side of the direct current (DC) subcarrier. The PSS uses six resource blocks with five reserved subcarriers on each side. Since the DC subcarrier contains no information, the PSS always maps to the middle 62 subcarriers within an OFDM symbol in a resource grid. $d(n)$ is mapped from lowest subcarrier to highest subcarrier. However, the PSS is mapped to different OFDM symbols depending on which frame type is used. Frame type 1 is frequency division duplex (FDD), and frame type 2 is time division duplex (TDD). For the purposes of this study, I focus on FDD frame type 1, since this configuration was seen to be more frequently used in the U.S. For this type, the PSS is mapped to the last OFDM symbol in slots 0 and 10, as shown in Fig. 2.10 [3].

2.3.2.2 Secondary Synchronization Signal

The SSS is based on maximum length sequences (m-sequences). An m-sequence is a pseudo-random binary sequence that is created by cycling through every possible state of a shift register. The synchronization signals are generated using three m-sequences, $\widetilde{s}$, $\widetilde{c}$, and $\widetilde{z}$, of length 31 [3].

Two binary sequences, $s_0^{(m0)}$ and $s_1^{(m1)}$, each of length 31, are different cyclic shifts of m-sequence $\widetilde{s}$ used to generate the SSS. The cyclic shift is determined from the indices m_0 and

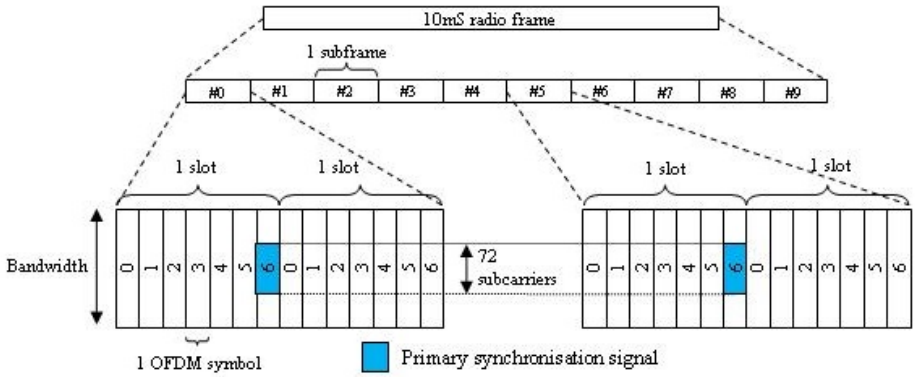


Figure 2.10: LTE PSS for FDD frame type [3]

m_1, which are derived from the cell-identity group, $N_{ID}^{(2)}$. These values can be read from table 6.11.2.1-1 in [12]. The two sequences are scrambled with a binary scrambling code ($c_0(n)$, $c_1(n)$), which depends on $N_{ID}^{(2)}$. The cyclic shift value of the first sequence transmitted in the radio frame determines the binary scrambling code, ($z_1^{(m0)}$, $z_1^{(m1)}$), that is used to scramble the second SSS sequence in each radio frame. The sequences for s_0 are given by equation 2.7 [3].

$$\begin{aligned} s_0^{m_0} &= \widetilde{s}((n + m_0) \mod 31) \\ s_0^{m_1} &= \widetilde{s}((n + N_{ID}^{(2)} + 3) \mod 31) \end{aligned} \tag{2.7}$$

The sequences for c_0 and c_1 are given by equation 2.8 [3].

$$\begin{aligned} c_0(n) &= \widetilde{c}((n + N_{ID}^{(2)}) \mod 31) \\ c_1(n) &= \widetilde{c}((n + N_{ID}^{(2)} + 3) \mod 31) \end{aligned} \tag{2.8}$$

The sequences for z_1 are given by equation 2.9 [3].

$$\begin{aligned} z_1^{m_0} &= \widetilde{z}((n + (m_0 \mod 8)) \mod 31) \\ z_1^{m_1} &= \widetilde{z}((n + (m_1 \mod 8)) \mod 31) \end{aligned} \tag{2.9}$$

To alternate the sequence transmitted in the first and second SSS transmission of each radio frame, the scrambled sequences are interleaved. This interleaving allows the receiver to determine the frame timing from observing only one of the two SSS sequences. If the first SSS signal observed is in subframe 0 or subframe 5, an LTE receiver can synchronize upon observation of the SSS signal in subframe 0 or subframe 5 of the subsequent frame [3].

The SSS is transmitted one OFDM symbol earlier than the PSS, but in the same subframe. The SSS is mapped to the same central 72 subcarriers as the PSS [3].

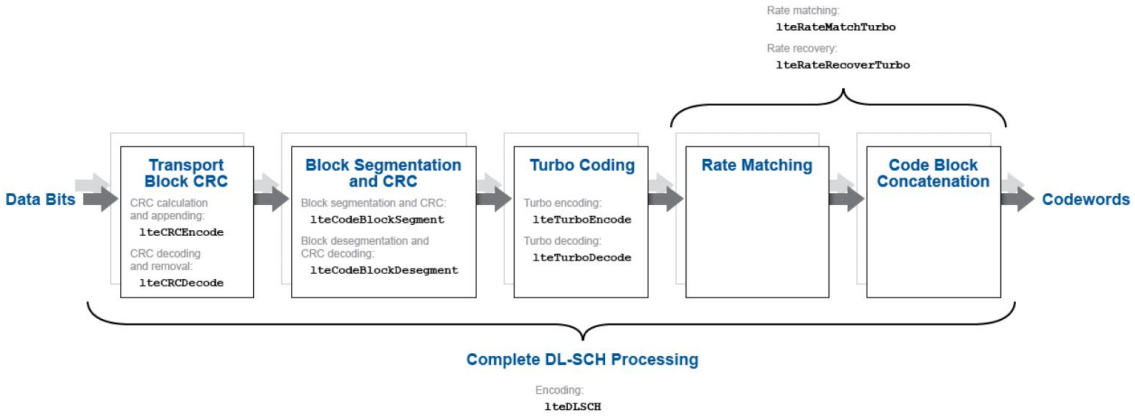


Figure 2.11: LTE DL-SCH Processing Blocks [4]

2.3.3 Physical Downlink Channels

The DL channels have the following purposes [42]:

- **PDSCH**: Physical Downlink Shared Channel. The main DL data channel that carries the data to be transmitted. 3GPP uses the term *shared* for the DL shared channel (DL-SCH) transport channel because it is shared by many logical channels such as BCCH, CCCH, DCCH, and DTCH.
- **PDCCH**: Physical Downlink Control Channel. The main DL control channel that carries scheduling information of different types, including the downlink control information (DCI) and information about DL data mapping.
- **PBCH**: Physical Broadcast Channel. The dedicated DL broadcast channel that carries system information for UEs requiring to access the network, including the BCH transport channel and system parameters such as the system BW, system frame number, and the number of transmit antennas used by the eNodeB.
- **PCFICH**: Physical Control Format Indicator Channel. The DL channel transmitted at the earliest OFDM symbols in each subframe that informs the UE about the format of the signal being received, including the number of control symbols present in the current subframe.
- **PHICH**: Physical HARQ Indicator Channel. The DL channel used for acknowledging UL packet reception to the UE.

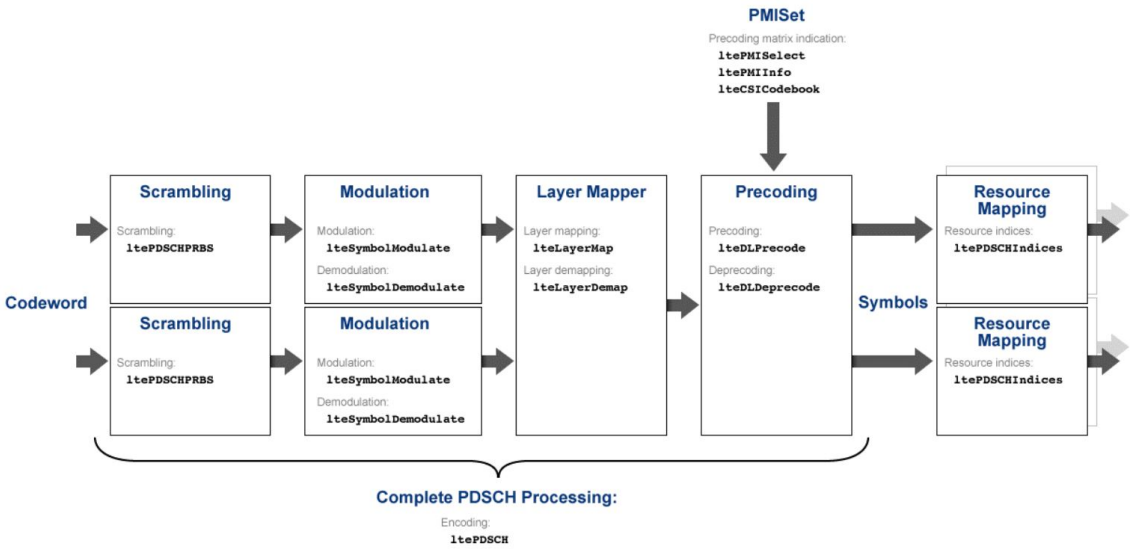


Figure 2.12: LTE PDSCH Processing Blocks [5]

- **PMCH**: Physical Multicast Channel. The DL channel that is used for multi-casting a packet to a set of UEs instead of a single UE.

Each PHY DL channel uses a different combination of encoding, modulation, and mapping techniques to produce the output symbols. Since the PDSCH carries the actual data to send, I shift focus to the processing blocks needed to produce these symbols. First, the data bits must be encoded to produce codewords, as shown in Fig. 2.11 [4].

Next, the DL-SCH codewords must be encoded to produce PDSCH symbols, as shown in Fig. 2.12 [5]. The last stage of PDSCH processing, resource mapping, places the PDSCH symbols into their appropriate indices in the resource grid, an example of which was illustrated in Fig. 2.9. After this processing, the last phase is OFDM modulation, which transforms the resource grid into an array of complex time samples to be transmitted by an RF front end.

2.3.4 Enhanced Inter-Cell Interference Control

Inter-cell interference refers to the scenario in which UEs in different neighboring cells both attempt to use the same resource, such as a frequency channel, at the same time. In LTE, Inter-Cell Interference Coordination (ICIC) refers to a set of techniques in which restrictions are applied to the radio resource management (RRM) block. The purpose of these restrictions are to improve channel conditions for the users that are most severely impacted by the interference. In such a way, LTE

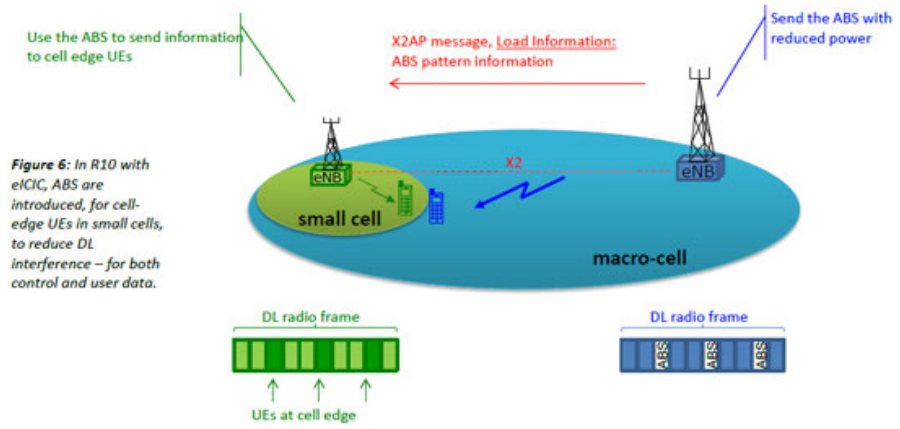
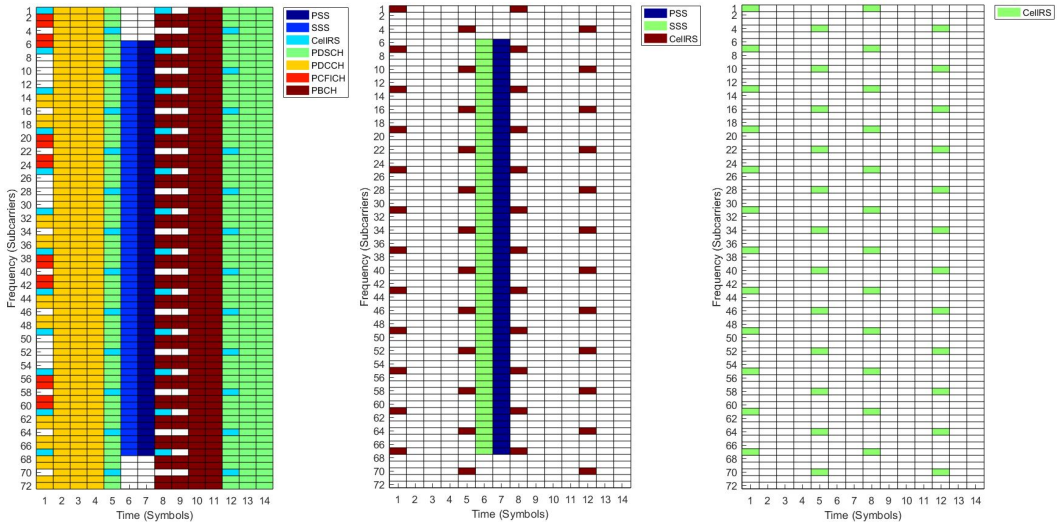


Figure 2.13: Almost Blank Subframes in eICIC [6]

systems can attain a high spectral efficiency. Resource management can be achieved through fixed, adaptive, or real-time coordination. Additional inter-cell signaling can assist this process. Inter-cell signaling generally refers to the communication interface among neighboring cells and the received measurement message reports from UEs [43].

In 3G release 8, LTE ICIC is introduced as an optional method for decreasing interference between neighboring macro base stations. In this version, the power of a portion of the frequency-domain subchannels is lowered. This portion can then be received close to the base station only. Since these subchannels do not interfere with the same subchannels used in neighboring cells, using this technique data can be sent faster on those subchannels to mobile devices close to the cell [44] [45].

Enhanced ICIC (eICIC) is part of the 3G release 10 LTE-Advanced heterogeneous network (HetNet) approach. In this approach, within the coverage area of macro cells are a number of smaller pico cells. In practice, these pico cells may be referred to as hotspots within larger areas such shopping centers or airports. The macro cells use higher power and can emit signals over longer ranges. In contrast, the pico cells use lower power and can emit signals over shorter distances. To reduce interference between a macro cell and several pico cells in its coverage area, eICIC coordinates the blanking of subframes in the time domain in the macro cell. In such a way, the macro cell subframes do not interfere and data transmissions can be much faster. This technique increases the overall system capacity when several pico cells are used in the coverage area of a single macro cell, since each pico cell can use these empty subframes without interfering with the other pico cells. However, this method decreases the capacity of the macro cell because it can't use all the subframes. For this reason, the number of subframes assigned for exclusive use in pico areas must be increased



(a) ABS Config 1 (b) ABS Config 2 (c) ABS Config 3

Figure 2.14: LTE Sample ABS Transmission Configurations

or decreased when traffic patterns change, and methods must be researched to effect these changes quickly [44]. An illustration of the eICIC process is shown in Fig. 2.13.

Thus, ICIC and eICIC differ in that ICIC is intended for mitigating interference between neighboring macro cells and eICIC is meant for coordinating between macro cells and pico cells in a HetNet configuration for increasing system capacity [44].

2.3.4.1 Almost Blank Subframe

The major change to eICIC in LTE R10 is the addition of time domain ICIC, which is realized via the Almost Blank Subframe (ABS). The ABS allows eICIC to better support heterogeneous network deployments, especially interference control for DL control channels. ABS includes only control channels and cell-specific reference signals, but no user data, and is transmitted with reduced power. Some possible ABS transmission configurations are described in Sec. A.3.4 of [46] and illustrated in Fig. 2.14. When eICIC is active, the macro-eNB transmits ABS subframes in a semi-static pattern. During these ABS subframes, UEs at the edge of the cell can receive DL information, both control and user data. The macro-eNB informs the eNB in the small cell about the ABS pattern [6].

2.4 LTE / Wi-Fi Coexistence Studies

Studies for coexistence between the LTE and Wi-Fi protocols are detailed in a number of wireless networking research papers. In [47], the negative impact of LTE transmissions on Wi-Fi performance are noted and a few methods for addressing them are mentioned. In [48], it is suggested that restricting LTE activity in the presence of many WLAN users may improve WLAN performance. In [49], a coexistence scheme is introduced to use the ABS subframes in the eICIC interference coordination mechanism to increase throughput in Wi-Fi networks. Various coexistence mechanisms are tested and evaluated in [50]. In [51], a cognitive coexistence scheme referred to as CU-LTE is proposed to enable spectrum sharing between U-LTE and Wi-Fi networks. However, none of these works perform tests with heterogeneous systems or online radio equipment.

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 [52]. However, this product uses separate chips for LTE and Wi-Fi, and so it cannot be used to explore the issues related with HW-SW co-design on the same wireless transceiver device.

Chapter 3

Methodology and Design for PHY Layer Wireless Systems

In this chapter, I describe the methods that I have employed for designing PHY layer wireless HW-SW systems. First, in Sec. 3.1, I introduce the methodology for designing a wireless system that is not real-time using Ettus Research USRP N210s and running MATLAB on a host PC. Then, in Sec. 3.2, I describe methods for designing a cycle-approximate PHY layer 802.11a wireless system on a heterogeneous architecture, targeting the Xilinx Zynq SoC and ADI FMComms3 RF board. Finally, in Sec. 3.3, I reveal techniques for detecting multiple protocols on FPGA, focusing upon the differences between 802.11a and LTE DL-SCH.

3.1 Online 802.11 CPU-based Design

SDRs that can be controlled by CPUs allow fine-grained control of their operation by executing the processing steps in user-accessible program code [14]. This technology forms the building block for applications needing high levels of reconfigurability, such as access points that support multiple wireless standards, or for systems like cognitive radios that incorporate situational intelligence to evolve with the radio frequency (RF) environment [53]. For example, in SDRs, the network designer can tune basic elements, such as modulation, spectrum spreading, scrambling, and encoding through software functions, instead of relying on static hardware, thereby allowing unprecedented access to all aspects of the radio operation. However, the expertise required to successfully navigate the mix of hardware design, software implementation, wireless standards requirements, and computational

timing limitations is significant, which requires specialized training and lengthens time to project completion.

A basic SDR system is composed of a computer connected to a RF front end capable of receiving and transmitting radio signals. An RF front end requires an antenna suited for specified RF bands of interest, a transceiver chip that is comprised of at least one local oscillator, analog-to-digital converter (ADC), and digital-to-analog converter (DAC), and an interface (e.g. Ethernet cable) that connects the front end to the computer. The computer may have a general purpose processor (GPP) to process the digital output, and computer programs to realize specialized tasks such as filtering, amplification, and modulation, which have traditionally been implemented in hardware. The design concept of the SDR is advantageous because it reduces the need for special purpose hardware and allows the developer to add new functionality to the radio by modifying the software. The flexibility inherent in the SDR allows for the potential to support many wireless standards, whereas a single hardware transceiver can only support a few or one standard. Hence, the SDR device can be seen as an increasingly affordable alternative.

Any modern wireless standard relies on accurate timing to complete the standards-specified tasks. In SDR, as the received and transmitted signals are represented as arrays of data samples collected by the front-end, software processing contributes to delays. Additionally, when multiple nodes operate in a shared channel, timing issues add to the challenge of ensuring synchronized behavior between multiple nodes. In the absence of hardware clocks, the SDR must devise a means of calculating how much time has elapsed, so that transmission and reception functions are performed at the appropriate intervals. The processing functions and their internal parameters must also be open for change, should a better algorithm be designed, or if no set thresholds may be possible, as is the case in highly challenging environments with variable noise floor. Finally, the software running on the SDR must be structured in a hierarchical manner, so that its functionality can be separated into layers that are compliant with the Open Systems Interconnection (OSI) model. Thus, the base drivers that interface with the RF front-end platform should be abstracted from the physical (PHY) layer functionality, which in turn should be abstracted from the medium access control (MAC) layer logic. In summary, there are many design challenges that must be overcome before a highly customizable SDR platform is made available for general purpose use.

This section details my approach to realize a SDR platform using commonly available tools. I believe that true and repeatable systems-level research is only possible when a commonly used processing environment is used in conjunction with affordable SDR hardware. This motivates my choices for basing my work on MATLAB software and Ettus USRP® N210 hardware [18]. My

approach introduces a novel methodology for an implementation starting at the USRP hardware driver (UHD) and building progressively up the protocol stack. To facilitate quick deployment, it includes an initialization script for the setting and tuning of the reconfigurable parameters at the physical layer based on the specific channel measurements at the chosen experimental site. Importantly, it complies with the processing definitions in the IEEE 802.11b specification, though hardware limitations increase the time to completion of the entire transmission/reception cycle compared to an off-the-shelf hardware-only Network Interface Card. The full details of this design are documented in [22], [54], and [55].

3.1.1 System Architecture Overview

The operational steps that architect my system are shown in Fig. 3.1. In a given SDR pair, I identify clearly the transmitting and receiving node by using the terms designated transmitter (DTx) and designated receiver (DRx). This terminology helps avoid ambiguity in describing a bi-directional transceiver link, where the transmitter must send out its DATA packet and then switch to a receiver role to get the acknowledgement (ACK). Thus, in the discussion ahead, the DTx alternates between its transmit and receive functions, and the DRx alternates between receive and transmit functions.

This section defines real-time operation over the course of an entire DATA-ACK packet exchange using equation (3.1) below:

$$t_{receive} \leq t_{radio} \tag{3.1}$$

where t_{radio} is the frame time stipulated by the USRP radio's analog-to-digital converter (ADC) and $t_{receive}$ is the average time to recover any given frame, which includes the time to retrieve a frame from the receive buffer, process the retrieved frame to decode it into the corresponding bits, and other memory and conditional operations.

This design sets forth the timing deadline given by equation 3.1. Such an operation will guarantee a stable, basic bi-directional link that shows no sign of any undesirable system behavior, such as buffer underflow or buffer overflow.

In the initialization step, the system is preset with recommended parameters and lets the user to modify a number of parameters for the entire transceiver chain. The user then in a simulation-only environment, initiates a parameter exploration stage, where all the *nodes* are virtual and are contained within the same computer. The DTx and DRx codes are executed with the user-supplied parameters as constants, and the code cycles through possible variations in the settings of processing

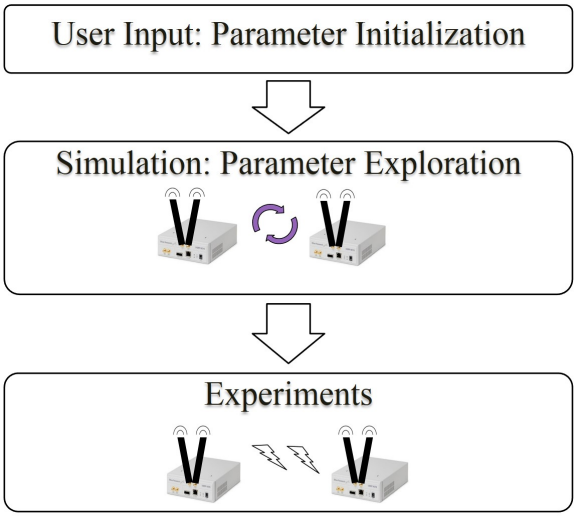


Figure 3.1: System Architecture

blocks as well as entire algorithms, each time identifying the performance that results from these settings.

From this data set, the user is presented with a feasible set of parameter settings. These parameter settings result in less than 5% packet loss at the receiver. This represents the *best case* scenario, for it should be noted that further channel outages will be introduced by the actual wireless channel. Once the user selects one of the possible feasible configurations returned by the search, the code is ready for driving the USRPs for over-the-air experiments.

I adopt the IEEE 802.11b PHY and MAC layer packet structure specifications in my implementation [56] [57]. This approach collects all the bits in the packet in multiples of 8 octets, which forms one USRP frame. This makes it easy for us to work with the MATLAB system objects (specialized objects required for streaming, henceforth referred to as objects) and with PHY and MAC header fields in the DATA/ACK packet that happen to have sizes that are multiples of 8 octets. Multiple USRP frames will compose the standard-compliant 802.11b packet.

I use differential binary phase shift keying (DBPSK), as the differential component enables us to recover a binary sequence from the phase angles of the received signal at any phase offset, without compensating for phase. In addition, DBPSK requires only coarse frequency offset compensation, without any closed-loop techniques. If residual frequency offset is much less than DBPSK symbol rate, then the bit error rate (BER) approaches theoretical values [22].

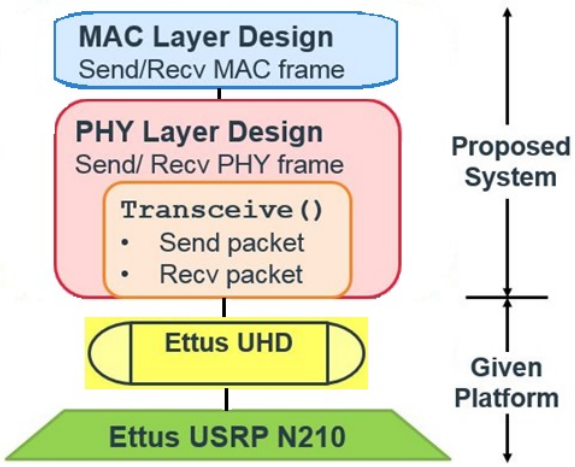


Figure 3.2: System Methodology

3.1.2 State-action based System Design

This approach involves first designing a number of (i) state diagrams to reflect the logical and time-dependent operational steps of my system, and (ii) block diagrams to reflect the sequential order of operations. Furthermore, I structure the MATLAB code in a way that enables slot-time synchronized operations. For the implementation, I use MATLAB Coder to generate the MEX functions for the USRP objects on an Ubuntu 64-bit platform that serves as the host computer for the USRPs.

Since the underlying code in a MEX function is written in C, it is generally faster than the interpreted MATLAB. The speed-up in performance can vary depending on the application. I preferred the MEX interface because it can enforce a consistent processing time per frame. The interpreted MATLAB, unlike the MEX, lacks this ability because it exhibits significant deviation from the desired timing. In addition, time-sensitive operations such as frequency offset compensation, show speed improvement using MEX.

This system design builds upon an already-defined platform, the USRP, produced by a well-known platform supplier, Ettus Research [18]. The communication between the USRP and host computer is established in MATLAB using the Communications System Toolbox (CST) USRP Radio support package, which acts as a wrapper for the Ettus USRP Hardware Driver (UHD) drivers. Identifying the manner in which the RF samples are transported between the USRP and a calling function defines the manner in which I must build the physical (PHY) layer, as illustrated in Fig. 3.2.

The UHD transfer of a frame of samples to a transmit buffer is performed as soon as it is requested while the UHD retrieval of a frame from a receive buffer has to wait until the next rising

edge of a clock cycle before trying to retrieve again. The most common undesirable behaviors that can occur are *underflow* and *overflow*. *Underflow* occurs when the radio requests for a frame of data from the receive buffer, but the host is not yet ready to provide it. *Overflow* occurs when the receive buffer becomes full and buffered data must be overwritten.

3.1.2.1 Slot-time synchronized operations

Any IEEE 802.11-based wireless transceiver implementation must have the ability to perform operations based on some slot-based timing. Performing such slot-time synchronized operations will let us realize time-sensitive functions, for example, make a node wait for a back-off (BO) duration before sending a DATA packet. Interpreted MATLAB or any other software that runs on the host computer may have trouble performing such operations in this manner, even by actively waiting. For this reason, I rely on the USRP for timing. In this design, the frame time is the minimum time the system takes to make a decision and hence, I equate it to the slot time. In this regard, the transceive function performs two actions: it gets a frame from, and puts a frame into the USRP buffers at fixed time intervals [22]. Using the value for USRP interpolation/decimation defined in Sec. 3.1.3.3, I can calculate the slot time. Then, I write the while loop in the main program so that it calls the transceive function once per loop, running helper functions to prepare data to transmit or process received data based on the active state, as shown in the program code in Listing 3.1.

```
while ~endOfTransmission
if (state==Tx)
data2Tx = processData2Tx();
end
dataRxd = transceive(data2Tx);
if (state==Rx)
processRxdData(dataRxd);
end
end
```

Listing 3.1: Main program that calls transceive function

I define a slot time as the smallest unit of time in which the SDR can make a decision. A data frame is sent or received every slot time and further, the functions I define for processing the received data frame or preparing a new data frame to transmit are intended to complete in less than a slot time to ensure timing accuracy. In practice, I recognize that the processing time for certain frames may exceed the radio time, t_{radio}, but the recovery time, $t_{receive}$, converges to the radio time.

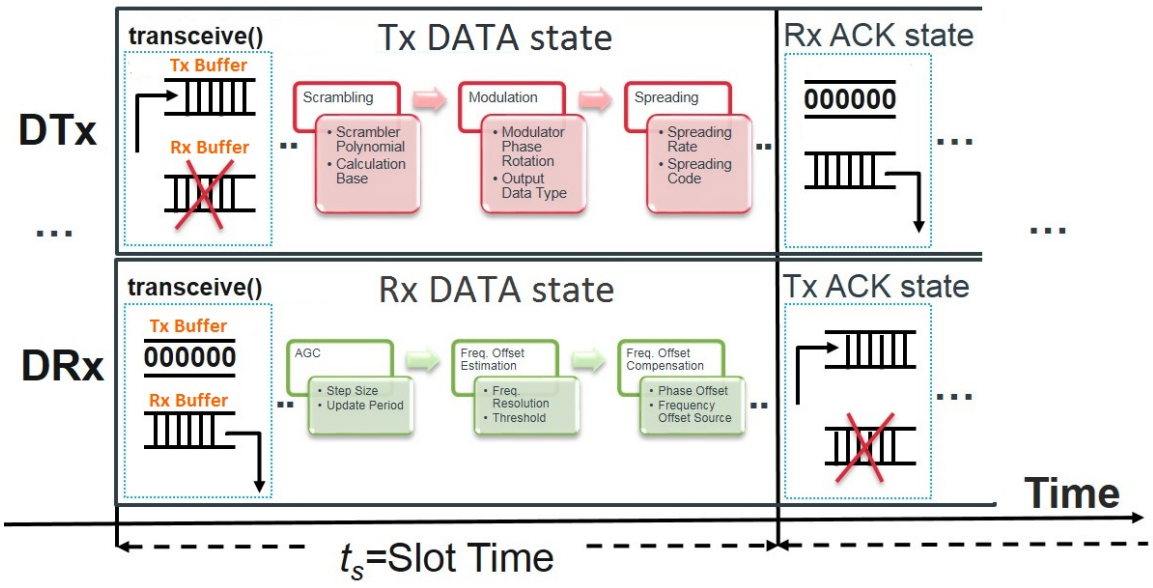


Figure 3.3: Transceive Function Behavior as Defined by Operational State

At the heart of the transceiver model is the *transceive* function, as shown in Listing 3.2. By design, *transceive* is called at a constant time interval that I define as a slot time. At each slot time, *transceive* sends and receives a fixed number of samples, which I refer to as a *USRP frame*. When a node (either DTx or DRx) enters a transmit state (refer to Fig. 3.3), it transmits the samples in the transmit buffer and ignores all samples in the receive buffer. On the other hand, when a node enters a receive state, it retrieves samples from the receive buffer for processing and puts zeroes in the transmit buffer. This way, I make sure that the samples in the transmit and receive buffer are current and relevant.

```
function dr = transceive(ft, d2s)
persistent hrx htx;
% Initialize received data variables
dr = complex(zeros(nspf,1));
ns = 0;
% Initialize system objects once
if isempty(hrx)
hrx = ...; htx = ...;
end
% Flag to release system objects
if ft
release(hrx); release(htx);
else
step(htx,d2s);
while (ns == 0)
[dr,ns] = step(hrx);
end
end
```

Listing 3.2: Transceive function code

The step method of the transmitter object operates in a blocking way as it returns only after the radio accepts the frame to be transmitted. On the other hand, the step method of the receiver object returns right away, hence it is non-blocking.

The step call of receiver object will return 0 as length of the received frame if there is not enough data in the radio. Once the radio collects enough data, the next step call returns a non-zero length value and the valid data. Since I know the sample rate of the data and the number of samples in a frame, I can calculate how long it takes to get one frame of data from the radio. The while loop blocks the transceive function until a frame of data is received. Therefore, I can use the call duration of this function as the clock source.

3.1.2.2 Designated Transmitter State Machine

In implementing the carrier sense multiple access with collision avoidance (CSMA/CA)-based protocol in the link layer, I identify 4 main states for the DTx, as shown in Fig. 3.4. Table 3.1 identifies the blocks in each substate and is described in detail in Sec. 3.1.2.4.

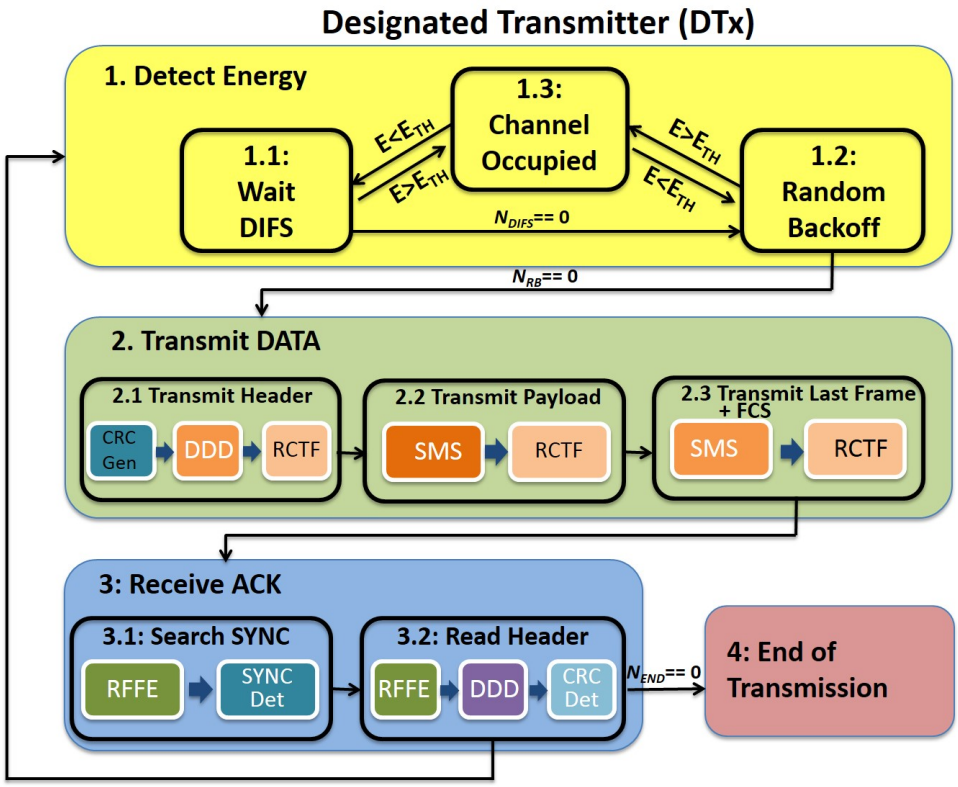


Figure 3.4: States for the Designated Transmitter (DTx)

Table 3.1: Common Combinations of Operations for a Substate

Block	Block Components
SMSRC	Scrambling, Modulation, Spreading, and Raised Cosine Transmit Filter (RCTF)
RFFE	Radio Frequency Front End: includes Automatic Gain Control (AGC), Frequency Offset Estimation and Compensation, and Raised Cosine Receive Filter (RCRF)
PD	Preamble/SYNC Detection: Find SYNC in Rx'd USRP frames
DDD	Despreading, Demodulation, and Descrambling

Detect Energy

At START, a new USRP frame arrives, and gets stored in a receive buffer. The DTx begins to continually sense energy in the channel and decides to transition either into a back-off state or to a

transmit state depending on whether the channel is busy or not. A random amount of time is chosen uniformly from a progressively increasing time interval. Only when the channel is free does the DTx decrement the back-off time; otherwise, it stalls. Only when the back-off time counts down to zero does the DTx attempt to transmit.

Transmit DATA

Upon entering this state, the DTx prepares the DATA packet and then, by calling the transceive function continually, places it in the transmit buffer of the USRP which then gets transmitted over the air. After transmitting the DATA packet, two possibilities exist. The transmission is successful with the reception of an ACK, or the transmission is not successful due to packet collision with another DTx.

Receive ACK

As soon as the DATA packet is transmitted, the DTx moves into the Receive ACK state, searching and decoding the Physical Layer Convergence Procedure (PLCP) header in the received ACK. If that is successful, the frame control and the address fields are read-out from the subsequent MAC header and checked for accuracy. The DTx then progresses to transmit a new frame and repeats the above mentioned sequence of steps until the last frame is successfully transmitted. On the other hand, if no ACK is received, the packet is considered lost and the DTx backs-off for an increased random back-off time and re-attempts transmission.

End Of Transmission

When there are no more DATA packets left to be transmitted, the DTx reaches the end of transmission (EOT) state.

3.1.2.3 Designated Receiver State Machine

Similarly, I identify 3 main states for the DRx as shown in Fig. 3.5. Unlike the DTx, the DRx does not perform energy detection.

Receive DATA

When the DRx successfully detects the Preamble and the Start Frame Delimiter (SFD), it decodes the PHY and MAC header and then progresses to extract the payload. When extracting the last set of

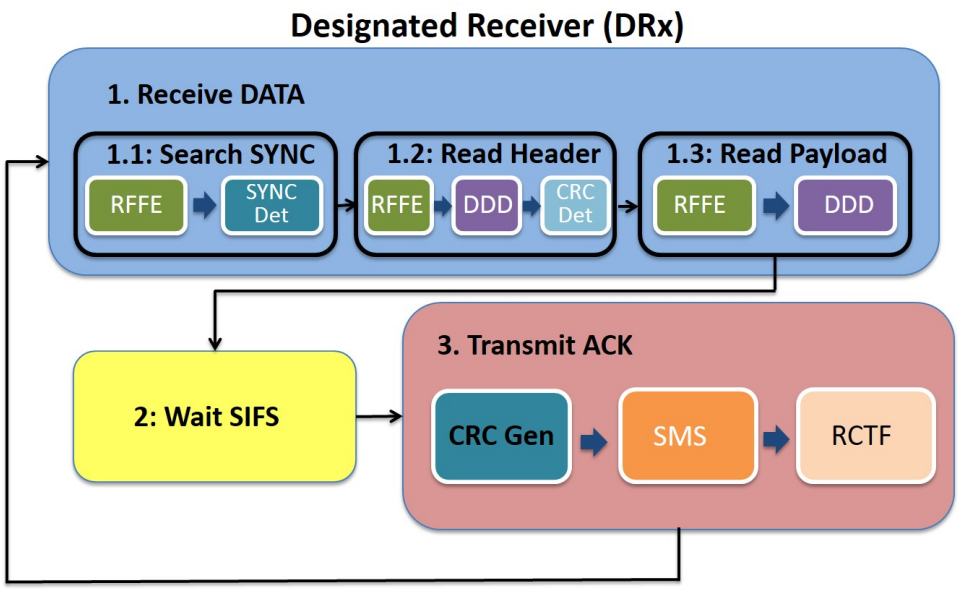


Figure 3.5: States for the Designated Receiver (DRx)

payload bits, Frame Check Sequence (FCS) is obtained and checked.

Wait SIFS

The DRx waits for a fixed interval of time, referred to as Short Inter-frame Space (SIFS), before sending an ACK packet post reception of the DATA packet.

Transmit ACK

The DRx sends out an ACK addressed to the DTx when it successfully retrieves all the payload bits.

3.1.2.4 System Blocks

Within each of the substates in the FSM diagrams (Figs. 3.4 and 3.5), there are sequential operations that need to be performed. In order to simplify the logic of which operations must be performed in each state, I define a number of *blocks* to comprise the most common operations, as shown in table 3.1. Identifying the grouping of blocks with the related substates helps better organize and restructure the implemented code.

In each substate of DTx state 2 (Tx) and DRx state 2 (Tx ACK), SMSRC is performed prior to each *transceive* (send and receive operation). In DTx substate 3.1 and DRx substate 1.1,

RFFE and PD are performed after each transceive. In DTx substate 3.2 and DRx substates 1.2, RFFE and DDD are performed after each transceive.

3.1.3 PHY Layer Algorithms

3.1.3.1 RF Front End Algorithms

The components in the RFFE block recover a signal prior to preamble detection. These include the automatic gain control (AGC), frequency offset estimation and compensation, and raised cosine filtering. The ordering of these components is an important consideration, and through exhaustive simulations, I found the preceding order to be ideal. The AGC algorithm counters attenuation by raising the envelope of the received signal to the desired level. I chose to use the MATLAB `comm.AGC` object [58]. To accurately estimate the frequency offset between the receiver and the transmitter, I chose to use `comm.PSKCoarseFrequencyEstimator` object, which uses an FFT-based-based method, based on equation (3.2), that finds the frequency that maximizes the FFT of the squared signal:

$$f_{offset} = \arg\max_{f} \mathcal{F}\{x^2\} \tag{3.2}$$

where x is the signal, $\mathcal{F}$ denotes the Fast Fourier Transform (FFT), and f_{offset} is the frequency offset.

Speeding up the RFFE block

From the initial experiments, I know that a frequency resolution (on the order of 1-10 Hz) is necessary in order to do preamble detection accurately. Setting such a low frequency resolution takes too long to execute with a sample rate of 200 kHz, or 200,000 samples per sec. For this reason, I decided to decimate the signal by a factor of 22 (the RCF factor times the spreading rate) before FOC (which is, in essence, an FFT). After decimation, I experimented with raising the FOC's frequency resolution by an order of magnitude higher, (10-100 Hz), and determined that it is accurate up to 100 Hz and meets the timing guidelines.

I employ a FIR Decimator step, as shown in Listing 3.3, that enables us achieve an order of magnitude reduction in RFFE block execution time. In essence, I am able to get enough frequency estimation accuracy with reduced sample rate (hence the use of decimation) and 100 Hz frequency resolution, which requires much less processing power than full frame higher resolution estimates.

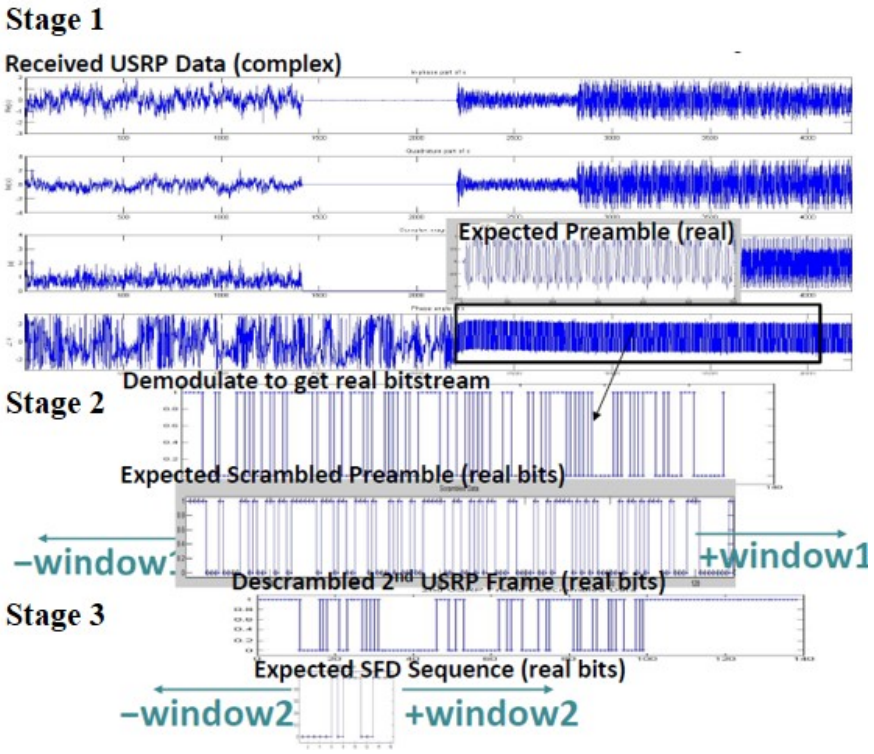


Figure 3.6: The Two Stages of Preamble Detection: Coarse and SFD

```
(1) dsp.FIRDecimator('DecimationFactor',22);
(2) comm.PSKCoarseFrequencyEstimator('Algorithm','FFT-based', ...
'FrequencyResolution',cef,'ModulationOrder',2,'SampleRate',(2e5/22));
```

Listing 3.3: RFFE Decimation Method

3.1.3.2 Preamble Detection Algorithms

The IEEE 802.11b preamble is a sequence of all one bits that undergoes scrambling. Since the scrambling phase is not known, and the received signal is correlated to the zero phase scrambled sequence, the maximum correlation position may not be the synchronization position. Therefore, the standard provides SFD, to fine tune the synchronization time.

Preamble detection (PD) is performed in two stages. In the first stage, I perform a cross-correlation of the received complex data after raised cosine filtering with the expected preamble (real) to get an estimate of where the preamble starts, to give the so called synchronization delay. Finally,

in the second stage, I look for the Start Frame Delimiter (SFD) immediately after the preamble in the descrambled bit stream. If it is not in the expected place, I perform a cross-correlation on a window of descrambled frame samples to the left and right to further fine-tune the synchronization delay.

Preamble Detection Speed Improvement

Detecting the Preamble fast and with high accuracy is critical to the speed at which the nodes can reliably exchange DATA/ACK packets.

- I experimented with several MathWorks utilities to compute cross-correlation faster (e.g. `dsp.CrossCorrelation` object, `xcorr` function).
- In one implementation, I exploit the property of the cross-correlation of two real signals in the frequency domain to compute the same (i.e. the point-wise product of the Fourier transform of the two signals), followed by an inverse Fourier transform results in the Cross-correlation of the two signals. Since one of the signals is the expected preamble, its Fourier transform can be pre-computed and loaded into the workspace during run-time.
- I determined the version of `dsp. Crosscorrelator ('method','fastest')` compiled using MEX to be the fastest among all the candidate methods for computing cross-correlation with increasing signal lengths, as shown in Fig. 3.7. It is important to note that although I operate with signal lengths on the order of 10^3, preamble detection is a frequent operation, so savings in time add up quickly.
- I declare a packet detection only if the second stage finds a perfect match for the SFD. This approach minimizes false packet detections greatly.

3.1.3.3 Parameter Selection

The initialization step described in Sec. 3.1.1 lets us carefully choose a number of design parameters (see table 3.2).

Constant Parameters for USRP & IEEE 802.11b Frame

I recognize parameters that cannot change during packet transmission/reception and have to be fixed. The number of octets in the payload per IEEE 802.11b packet should be maximized to decrease the header overhead. In that case, a large frame size is preferred as it reduces the percentage of overhead

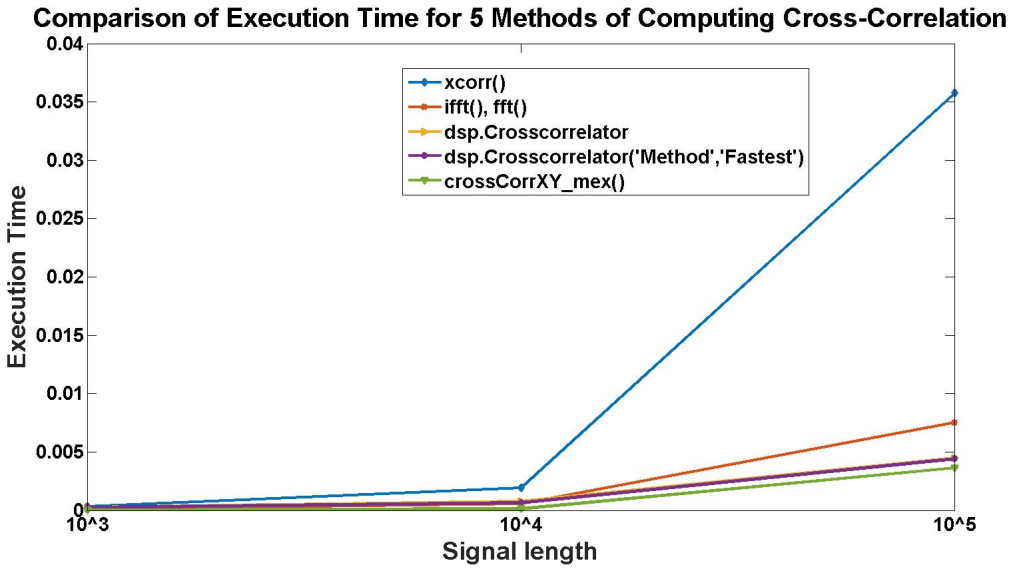


Figure 3.7: Comparison of Execution Time for 5 Methods of Computing Cross-Correlation

Table 3.2: Important Parameters

Param	**Block**	**Description**	**Val/Range**	**Tunable?**
R_i, R_d	USRP	USRP Interpolation/Decimation Factor	500	No
L_f	USRP	USRP Frame Length	64 bits	No
L_p	Frame	#Octets per IEEE 802.11b Frame Payload	2012 octets	No
K	RFFE	AGC Max Power Gain	30-60	Yes
N	RFFE	AGC Adaptation Step Size	0.01-0.5	Yes
Δf	RFFE	Frequency Resolution	1-100 Hz	Yes

processing. On the other hand, the frame size should be minimized to make quick decisions with a small number of samples or bits, unlike a large frame size which increases the frame time, thereby reducing the resolution of time ticks for the system. I chose frame length of 1408 as a well balanced compromise between these two requirements. For this reason, the frame length is left fixed.

The USRP N210 analog-to-digital converter (ADC) operates at a fixed rate of 100 MHz. The USRP interpolation-decimation rates control the rate of transmitting and receiving frames. For

example, setting interpolation rate, R_i, and decimation rate, R_d, to 500 ensures that the ADC and DAC convert a sample every 5 μs, as shown in equation (3.3).

$$t_{sample} = R_i/100Msamples/sec = 500/10^8 = 5 \times 10^{-6} sec/sample \quad (3.3)$$

For example, setting frame length, L_f, to 1408 samples means that a frame is retrieved by the transceive function every 7.04 ms, as shown in equation (3.4).

$$t_{radio} = L_f \times (R_i/100Msamples/sec) = 1408 \times (500/10^8) = 7.04 \times 10^{-3} sec/frame \quad (3.4)$$

Even though the system may take more than 7.04 ms to process a frame every once in a while, the buffers in the USRP receiver prevents the system to overrun (or lose samples) and the system, on average, stays real-time.

Tunable Parameters for RFFE Block

Tunable parameters can change during transception. For example, the AGC adaptation step size controls the convergence speed of a received signal's envelope to the desired level. In other words, it governs the speed of convergence. Finally, the frequency offset estimation component's frequency resolution setting is an important design consideration as it is inversely proportional to the FFT length. A lower frequency resolution gives more accurate offset estimates, but with increased computational time.

3.1.3.4 Same-Frequency Channel Operation

In a multi-node setting, it is advantageous to operate the transmit and receive links, at the DTx and DRx, in the same band of frequencies. Thus, I set both DTx and DRx to operate at the same center frequency. Unlike different-frequency channel operation, this eliminates the need for repeated switching of transmit and receive center frequencies when transitioning among the energy detection, transmit, and receive states. In addition, it makes for an easier implementation of medium access and contention resolution.

From the initial experiments, I learned that the receive-only port, *RF2*, of the USRP leaks about 7 dBm into the transmit & receive port, *RF1*. The effect of this leakage causes the DTx to detect the preamble in its own DATA packet while it is waiting for an ACK. I added logic to ensure that the DTx rejects its own DATA packet as soon as it reads the MAC header and does not find the expected ACK frame control sequence.

3.2 Offline 802.11 CPU/FPGA Co-design

As some conclusive lessons from the online CPU-based study, I realized that the CPU-based platform is not sufficiently fast to handle the bit rates required by 802.11. For this reason, it was necessary to explore alternate computing platforms that are able to execute wireless processing blocks in the same amount of time that the wireless transceiver chip transmits and receives one complex fixed-point sample. Furthermore, to meet the needs of modern wireless protocols, I saw the need to shift the protocol of test from 802.11b to 802.11a. Modern Wi-Fi protocols are based on 802.11a (eg. 802.11g/n/ac), which uses Orthogonal Frequency Division Multiplexing (OFDM) as a major method for increasing bit rate. Later in this dissertation, I identify same major similarities between 802.11a and LTE, which is also OFDM-based.

For this I target a Xilinx Zynq based platform coupled with an Analog Devices FMCOMMS RF front end. The Zynq chip includes both an embedded ARM processor for software implementations as well as 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.

I explore the problem of codesign in HW and SW using commercially available tools, including MathWorks Simulink and Xilinx Vivado. I demonstrate our approach using IEEE 802.11a transmitter (Tx) and receiver (Rx) Simulink models and ensure correctness by comparing against Annex G of the 802.11a specification [1]. These models require modification, such as different data types, to best target execution on HW or SW. I generate HDL code and 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 ARM processor from these high level models. I 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. The full details of this design are documented in [59] and [60].

This approach advances the state of the art in several ways. The most significant contributions are as follows.

Common Modifiable Hardware/Software Platform: I 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. I 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, I 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, I 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. I can support spectrum coexistence, such as LTE and WiFi on the same channel, TV whitespace reuse, or co-operation with RADAR. I 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.

Table 3.3: Zynq Board Comparison
https://www.sharelatex.com/project/58d3cd83f6efb8957450479d

	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

3.2.1 Target Hardware and Software

3.2.1.1 Hardware Setup

Our HW consists of an RF front end, the ADI FMComms3 board; a Xilinx Zynq evaluation board; and a host computer. I 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 [17]. The Xilinx Zynq SoC has an embedded ARM processor and FPGA fabric. Throughout this paper I use Xilinx terminology: Processing System (PS) for the ARM processor and Programmable Logic (PL) for the FPGA fabric. In our experiments, I 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 3.3. Internal to the Zynq processor, an Advanced eXtensible Interface (AXI) bus connects the PL and the PS. I 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. 3.8.

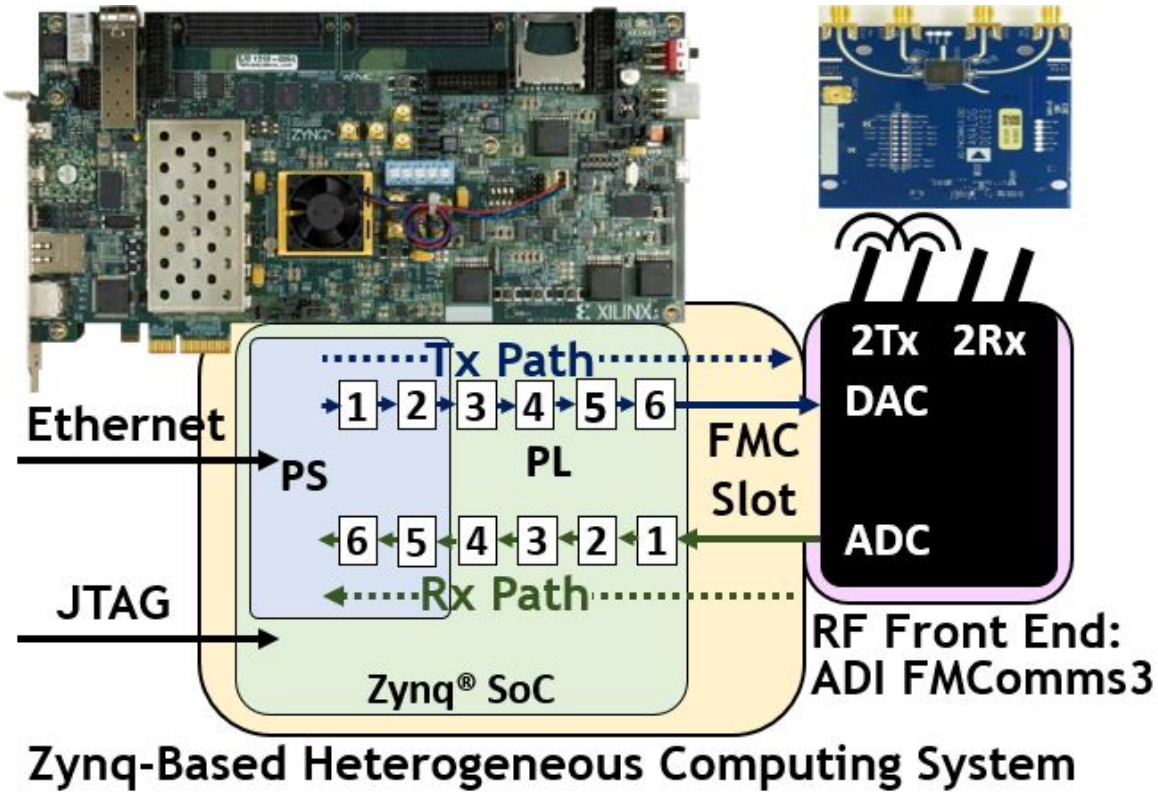


Figure 3.8: 802.11a Transceiver HW: Zynq SoC & FMComms3

3.2.1.2 Software Tools

To target the hardware described above, I use commercially available tools from MathWorks and Xilinx as illustrated in Fig. 3.9. I 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 [61].

I start by making a Simulink model to capture all the information about the Zynq transceiver system. In this model, I set radio parameters such as sampling frequency and number of samples per frame. I 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. I 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

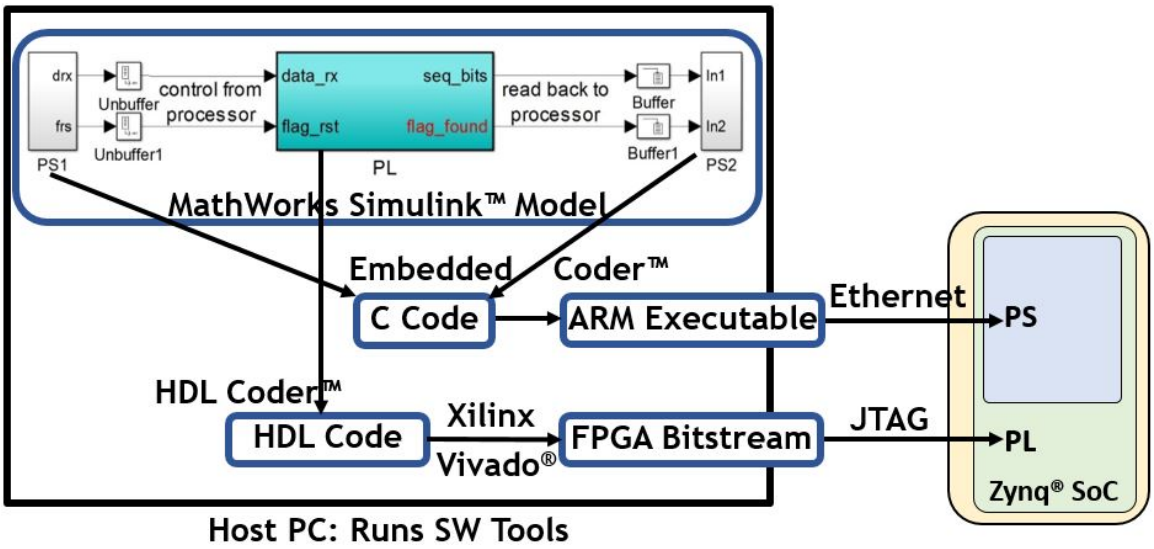


Figure 3.9: High-Level SW Tool Workflow for Zynq PL & PS

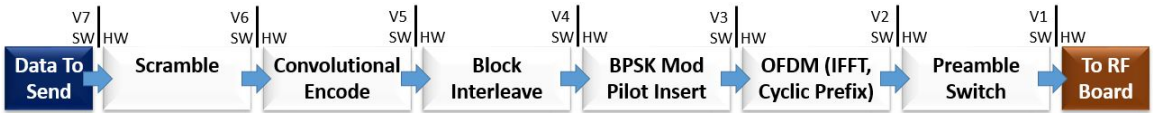


Figure 3.10: 802.11a Transmitter Chain HW/SW Codesign Variants

from the command line to synthesize, implement, and make a bitstream [62]. I program the PL with this bitstream.

Finally, I 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 [63]. Then, Simulink invokes Xilinx SDK to package and compile the executable for the PS [62]. When I press the play button in the Simulink model, it sends a signal via Ethernet to launch the executable on the PS.

3.2.2 Transceiver HW-SW Co-design

3.2.2.1 Design Variants

Figures 3.10 and 3.11 show the processing chains for transmitter and receiver respectively. I have functionally equivalent software (PS) and FPGA hardware (PL) versions of each of the blocks in these figures. For each processing chain, I have explored HW-SW codesign by creating a number

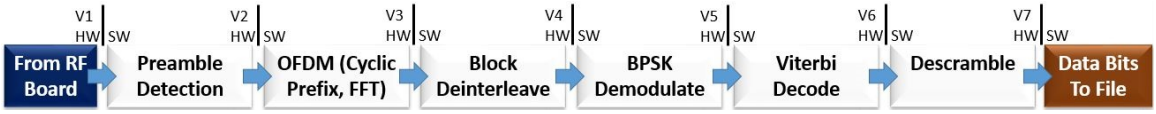


Figure 3.11: 802.11a Receiver Chain HW/SW Codesign Variants

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 I want to minimize. In future experiments, I 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. 3.10. 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. I 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 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 I also model seven variants, as shown in Fig. 3.11. 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

Table 3.4: 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

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. I evaluate all these different versions with respect to timing, resource utilization and energy efficiency.

3.2.2.2 Timing Considerations

In order to maintain a real-time transceiver system and prevent unacceptable data losses, I need to closely monitor execution times. For simplicity and to meet SW tool requirements, I 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, I 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 3.5.

$$t_f = t_s n_{spf} \tag{3.5}$$

By setting the appropriate step times in the model variants, I can ensure that data is transferred between PL and PS at the desired rate. However, I 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 I 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, I 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 3.4.

According to the IEEE 802.11a specifications for the lowest bit rate, each OFDM symbol represents 24 data bits [1]. If I assign only one OFDM symbol to a frame, each frame has 24 data bits.

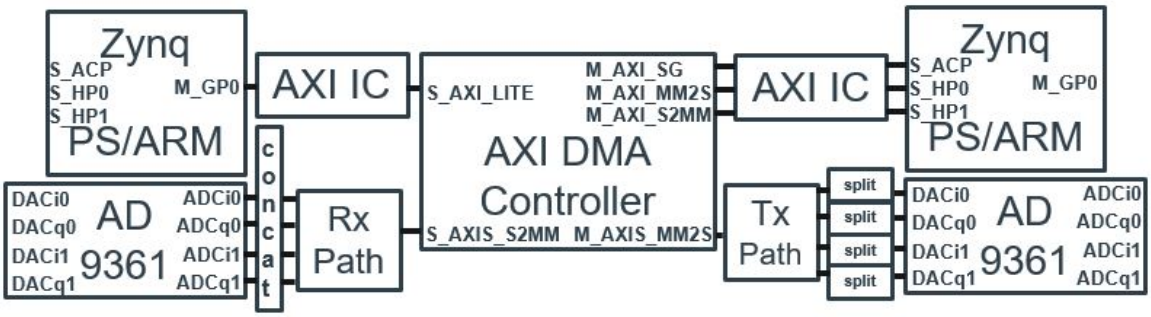


Figure 3.12: AXI Connections between Zynq PS and AD9361 Chip

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, I 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.

3.2.2.3 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. I use sample and frame counters to handle logic specific to a sample or frame number.

3.2.3 HW-SW AXI Interfacing

As described in Section 3.2.1 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 I use: AXI-lite is a memory mapped protocol and AXI-Stream is intended for high-speed streaming data. I use AXI-lite for the Rx and AXI-stream for Tx. To support the

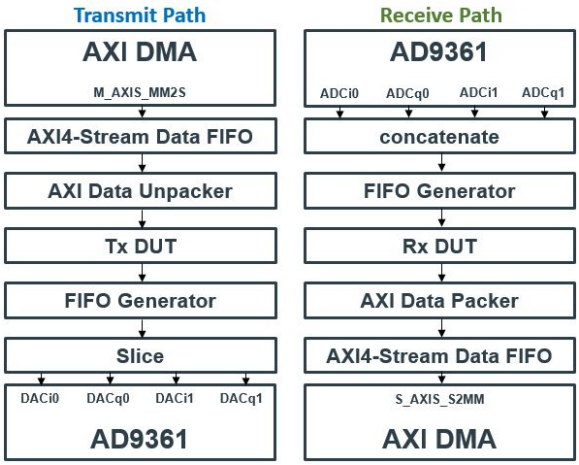


Figure 3.13: AXI-Stream Transmit and Receive Path Detail

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. 3.12. 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 di

3.2.4 Considerations for Reusability and Optimization

3.2.4.1 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. 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.

3.2.4.2 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.

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 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.

3.3 Offline LTE/802.11 FPGA-based Design

One of the main lessons learned from the offline 802.11 HW/SW codesign study was that performing processing blocks on the FPGA fabric caused a huge improvement in processing time, moving closer to the timing requirements specified by the 802.11a standard. However, another lesson learned was that the resources on FPGA are limited, and that the wireless systems designer must be conscious of both the utilization of FPGA resources and the increasing data path delay, which controls the achievable clock cycle. Thus, in the next phase of this research, we must further explore the key factors that would allow us to support multiple protocols on the same device. Since the codesign study identified the transmitter chain as having low resource utilization and delay, we can presume that it

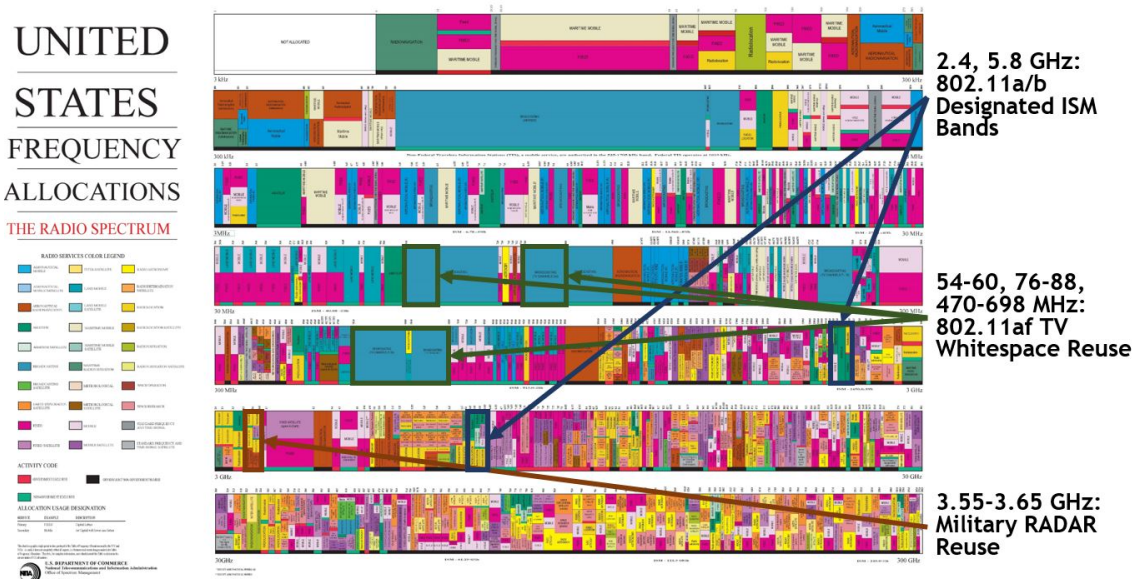


Figure 3.14: US Spectrum Bandwidths for Wireless [7]

would be fairly simple to place two parallel Tx chains on the same FPGA fabric. Therefore, we focus this next research phase on the receiver chain with the goal of identifying how to efficiently detect the presence of one of multiple protocols that may be operating on the same frequency bandwidth.

As mentioned in the introduction, modern wireless communications standards are constantly evolving to meet the needs of an increasing number of devices. The fact that these protocols are consistently evolving makes wireless transceivers a great application for reconfigurable hardware. Each protocol introduces new challenges, including ways to address contention in an overcrowded wireless spectrum. The problem of spectrum scarcity has caused the FCC to open up new bandwidths for use by multiple protocols. Some key bandwidths in the US frequency spectrum are shown in Fig. 3.14 [7]. As illustrated in the figure, spectrum is scarce and the unlicensed bands are an extremely valuable commodity. To help resolve this, the FCC has opened up previously-reserved licensed bandwidths for reuse, such as the TV whitespace and military RADAR bandwidths. The presence of multiple protocols on the same bandwidth is a new scenario that FPGAs can be instrumental in prototyping.

There are large differences in utilization of spectrum bandwidths. While the bandwidths using LTE are frequently overutilized and congested, bandwidths reserved for other purposes may often be underused. Although many people may think of 2.4 and 5.8 GHz bands as Wi-Fi bands, they're really open to any industrial, scientific, and medical (ISM) purposes. For this reason, mobile

phone technology wants to use these ISM bands for control if presently unoccupied to boost coverage in their cellular networks. Companies have proposed LTE protocol variants to accommodate this, such as LTE-U and MulteFire by Qualcomm and License Assisted Access (LAA) by Ericsson. Cellphone carriers such as T-Mobile and Verizon have indicated interest in deploying this idea as early as possible.

This section presents an FPGA-based approach to model the receiver chains for both 802.11a and LTE protocols, starting with their lowest-layer physical (PHY) receiver chains for one antenna. I identify and propose methods for handling the major issues associated with dual-protocol support, including rate transition, joint pattern detection, and OFDM demodulation with different numbers of subcarriers.

3.3.1 Challenges of Protocol Coexistence

Modern smartphones can communicate using both LTE and Wi-Fi protocols, but they incorporate separate chips to handle each protocol. Each chip is designed to use a different fixed frequency bandwidth for transception. A novel idea in the case of one connection outage, would be to switch to the other protocol. However, there are many challenges of coexistence between Wi-Fi and LTE. First, there is the challenge of synchronization, the detection of a fixed pattern in either standard and arrangement of clocks for proper timing. Whereas Wi-Fi uses OFDM, the LTE downlink uses OFDMA (multiple access). At the MAC layer, Wi-Fi uses the distributed coordination function (DCF), while LTE allows for flexible resource allocation using either frequency division duplexing (FDD) or time division duplexing (TDD). Thus, given their differences, there are outstanding questions about how a device could handle multiple protocols. Wi-Fi manages contention using carrier sensing multiple access with collision avoidance (CSMA/CA), while LTE base stations, or eNodeB's handle this using subchannel allocation.

A major benefit of the flexible SDR testbed described in Sec. 3.2 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 3.5. The functional blocks of our 802.11a implementation, especially those concerning scrambling and block interleaving, can be reused 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 inherent in our modeling environment allows us to explore LTE and Wi-Fi coexistence on the same channel, and also allows the same SDR hardware to switch between access

Table 3.5: Wireless Standard Block Comparison

		802.11a	802.11b	802.11g	802.11p	802.11n	802.11ac	802.11ad	LTE	LTE-A
Processing Blocks	**Scrambling**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**
	FEC 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)		
	Dig 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)
	DSSS		(2)	(2)						
	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 Detect	**(1)**	(2)	(2)	(2)	(2)	(2)	(2)	(2)	(2)

(1) Implemented & Reusable, (2) Not Yet Implemented, but Reusable

standards by retaining common processing blocks and downloading additional blocks as needed.

As part of this research, I recognize the need to identify how easily processing blocks can adapt to support multiple standards. 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, I note that OFDM is used by both the 802.11g and LTE 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, where MA stands for multiple access, used in the LTE DL channel. Our modeling environment can easily modify processing blocks to prototype the different settings for each standard.

In LTE, many additional considerations must be taken into account in order to model the signals properly. An overview of the major considerations is shown in Table 3.6 [12]. Whereas

Table 3.6: Comparison of 802.11a and LTE Parameters [12]

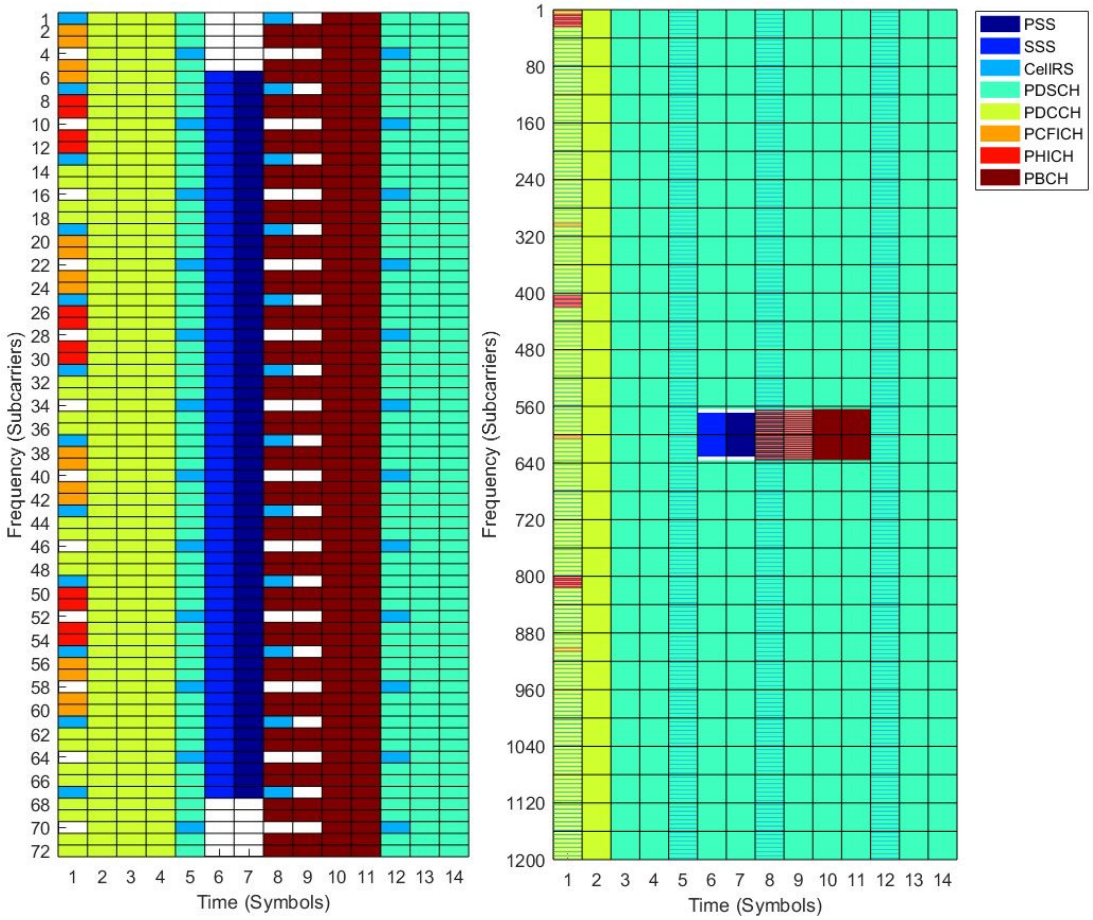
	802.11a	LTE					
f_{BW}	20	1.4	3	5	10	15	20
n_{RB}	n/a	6	15	25	50	75	100
n_{FFT}	64	128	256	512	1024	1536	2048
f_{smp}	20	1.92	3.84	7.68	15.36	23.04	30.72
n_{CP0}	16	10	20	40	80	120	160
n_{CP1+}	16	9	18	36	72	108	144
RMC#	n/a	4	5	6	7	8	9

802.11a operates on a 20 MHz carrier bandwidth, f_{BW}, LTE is intended to operate on any one of six bandwidths, from 1.4 to 20 MHz. In addition, the mapping of information to frequency subcarriers is not as simple as in 802.11a. Instead, digitally modulated symbols must be mapped to an index in a resource grid where the rows represent frequency subcarriers and the columns represent one OFDM symbol in time. Two sample DL resource grids were shown in Fig. 3.15. Reference Measurement Channel (RMC) 4 is a 1.92 MHz example resource grid illustrated in Fig. 3.15a, and RMC 9 is a 20 MHz resource grid shown in Fig. 3.15b.

In LTE, each OFDM symbol can contain data from multiple physical channels and signals. The primary synchronization signal (PSS) and secondary synchronization signal (SSS) are fixed sequences that can be used by the receiver for aligning the start of the signal, much like the 802.11a preamble. For all LTE DL signals, regardless of the channel bandwidth, the PSS and SSS always occupy the central 62 subcarriers in the inner 1.92 MHz band.

The user bit sequence to be transmitted is inserted into the downlink shared channel (DL-SCH), and then modulated and mapped into the physical downlink shared channel (PDSCH) shown in cyan. The number of LTE resource blocks, n_{RB}, each containing 12 subcarriers, can range from 6 to 100. Thus, the (I)FFT sizes needed for OFDM (de)modulation can range from 128 to 2048, as opposed to the 802.11a fixed FFT size of 64. Moreover, while the 802.11a CP length is fixed at 16 samples (0.8μs), the LTE CP length varies. On the first OFDM symbol in a time slot, the normal CP length ranges from 10 to 160 samples. In the remaining six OFDM symbols in a time slot, the normal CP length ranges from 9 to 144 samples. Note that LTE also specifies an extended CP option that is not modeled in this study [12].

Other differences include that the LTE DL-SCH receiver does not support BPSK digital



(a) RMC4: 1.92 MHz BW

(b) RMC9: 20 MHz BW

Figure 3.15: LTE Downlink RMC4 and RMC9 Resource Grids

modulation, and must incorporate Turbo decoding instead of Viterbi. Also, the channels and mapping indices for uplink (UL) transmissions from the user equipment (UE) to the eNodeB are different than for downlink (DL) transmissions from eNodeB to UE.

3.3.2 Multiple Protocol Support for PHY Layer Rx

In this section, I propose an overall ordering of processing blocks for receiving multiple protocols, as shown in Fig. 3.16. I also contribute high-level designs for realizing the most important components. I identify a technique for transitioning between sampling rates that makes efficient use of FPGA resources. I incorporate a hardware-friendly matched filtering mechanism to detect fixed patterns in both protocols. I mention a technique for performing OFDM demodulation with different numbers

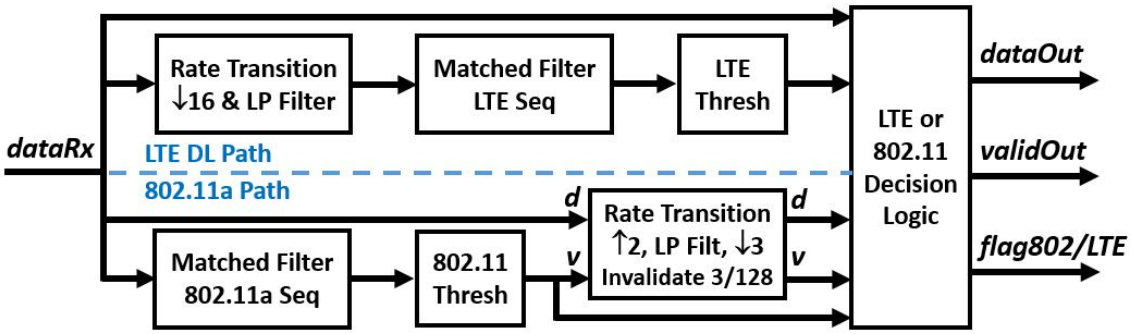


Figure 3.16: Dual Rate Transition and Pattern Detection FPGA Design

Table 3.7: Choice of Sampling Rates

$f_{s,rf}$ (Msps)	30.72	40	60	61.44
$r_{interp,802}$	125	1	1	125
$r_{decim,802}$	192	2	3	384
$r_{interp,lte}$	1	96	64	1
$r_{decim,lte}$	1	125	125	2
$r_{decim,ltepss}$	16	2000	2000	32
Achievable Clock Cycle (ns)	32.55	25	16.67	16.28

of subcarriers using a single FFT. In addition, I present a high-level design workflow that makes it easier to tune parameters and glean relevant results.

3.3.3 Sampling Rate Transition

Foremost, a proper device sampling rate must be chosen for the RF front end device, such as the ADI FMComms3 [64], since this parameter cannot be changed easily during the device's operation. To support both 802.11a and all LTE bandwidths, the device must accommodate both the 20 Msps and 30.72 Msps sampling rates, and so the device sampling rate, $f_{s,r}$, must be greater than or equal to the maximum of the two rates. $f_{s,r} >= 30.72Msps$. This raw received signal needs to be resampled to both the 802.11a and LTE rates using the interpolation and decimation factors shown in Table 3.7. However, a tradeoff exists for this variable because increasing the device rate, $f_{s,r}$, decreases the maximum acceptable achievable clock cycle, shown in the last row.

Elementary sampling theory tells us that in order to prevent aliasing, a lowpass (LP) filter is needed to remove spectral copies. After the interpolation by a factor of r_{interp} and before the deci-

mation by a factor of r_{decim}, I must filter out frequencies above the inverse of $\max(r_{interp}, r_{decim})$ times the Nyquist frequency. I use a finite impulse response (FIR) filter to achieve this, since these are easiest and most efficient to implement on FPGA fabric. Thus, I upsample, lowpass FIR filter, and downsample, as shown in equation 3.6.

$$\begin{aligned} y_{interp}[n] &= y_{origRate}[n/r_{interp}] \\ y_{LP}[n] &= \sum_{i=0}^{n_{rfc}} b_{LP}[i] \cdot y_{interp}[n] \\ y_{newRate}[n] &= y_{LP}[r_{decim}n] \end{aligned} \tag{3.6}$$

where b_{LP} are the LP filter numerator coefficients and n_{rfc} is the FIR filter length. I recognize that n_{rfc} is an important parameter in that it allows us to model the tradeoff between accuracy and efficiency. The number of resampling filter coefficients, n_{rfc}, can be increased to get more numerically accurate samples at the cost of utilizing more FPGA resources. However, for very large resampling factors as the $r_{decim,802} = 192$, such a lowpass filter would be infeasible because it would remove most frequency domain information. Furthermore, the use of a polyphase filter to perform the resampling in 3 separate phases was found to be inefficient on FPGA, as the reliance upon valid-in and valid-out signals cause long sample delays and inaccurate results.

Thus, I employ a technique that uses the close approximation of r_{interp}=2 and r_{decim}=3, and manually invalidate 3 samples. Doing so ensures that there are only 125 valid samples in every 192 samples input at the faster rate. However, such a technique is not expected to produce accurate results for long periods of time. Thus, I ensure that the x[3n/2] rate transition is enabled only when the start of a received sequence is found. To accomplish this I connect the valid input signal to the output of the 802.11a pattern detection mechanism. Next, I must ensure that the pattern detection is performed accurately.

The LTE path must also undergo a LP filter and downsampling before detection of the synchronization signals, since as shown in Fig. 3.15 the inner 1.92 MHz band must be extracted to compare with the expected fixed PSS pattern. Thus, I insert another rate transition block with r_{interp}=1 and r_{decim}=16 to remove this inner band for the subsequent pattern detection component.

3.3.4 Pattern Detection

To detect the start of a transmission, the multi-protocol receiver must recognized fixed patterns to synchronize its timing properly and begin OFDM demodulation at the precise starting sample. For 802.11a signals this pattern is comprised of short and long training symbols, and for LTE DL signals,

this pattern is comprised of the PSS and SSS. The most straightforward mechanism for detecting this pattern on reconfigurable hardware is the matched filter (MF), an FIR filter whose coefficients are the reversed complex conjugate form of the expected fixed pattern, as shown in equation 3.7.

$$\begin{aligned} b_{MF}[n] &= y^*_{fixPatn}[n_{patn-1}, n_{patn-2}, ..., 1, 0] \\ y_{MF}[n] &= \sum_{i=0}^{n_{mfc}-1} b_{MF}[i] \cdot y_{rawRxInput}[n] \\ i_{PatnEnd} &= \arg\max_i(y_{MF}[i]) \end{aligned} \tag{3.7}$$

where n_{patn} is the length of the fixed pattern (e.g. 802.11a preamble or LTE synchronization signals), b_{MF} are the MF numerator coefficients and n_{mfc} is the MF length. As another practical consideration, since finding the absolute maximum of the MF output would take a very long time, I instead find the index at which the preamble ends, $i_{PatnEnd}$, by locating when the value of the MF output first exceeds some threshold, $y_{MF}[i_{PatnEnd}] > v_{MFthresh}$.

I recognize that n_{mfc} is another parameter to adjust to target either accuracy or efficiency. The full 802.11a preamble amounts to 320 samples, and the full LTE combined SSS/PSS sequence amounts to 274 samples. However, the designer must take into account the practical limits of the FPGA resources, and implementing FIR filters with this many coefficients cannot be done on moderately-sized FPGAs, such as the Kintex-7 variety on the Zynq SoC. Thus, while I would like the matched filter results to be as accurate as possible, I choose to use a reasonable $n_{mfc} = 64$, the length of one 802.11a long training symbol or one-half PSS without CP, to save FPGA resources for other components.

Our rate transition and MF designs use fixed-point numeric data types to represent complex signal values. Recognizing that the raw input Rx data is always fractional, $||y_{rawRxInput}|| < 1$, I also realize that the square of the MF output can only be above 1 when the fixed pattern is detected. Thus, I design our fixed point data types to always be signed and have one integer bit. The number of bits per word, n_{bpw}, can be increased to get more numerically accurate multiplies and accumulates since every fixed-point multiplication requires twice the number of fraction bits to produce an accurate result. However, implementing FIR filters with very high n_{bpw} cannot be done on moderately-sized FPGAs, so I identify another tradeoff. The parameter n_{bpw} can be increased to boost accuracy or lowered to save resources.

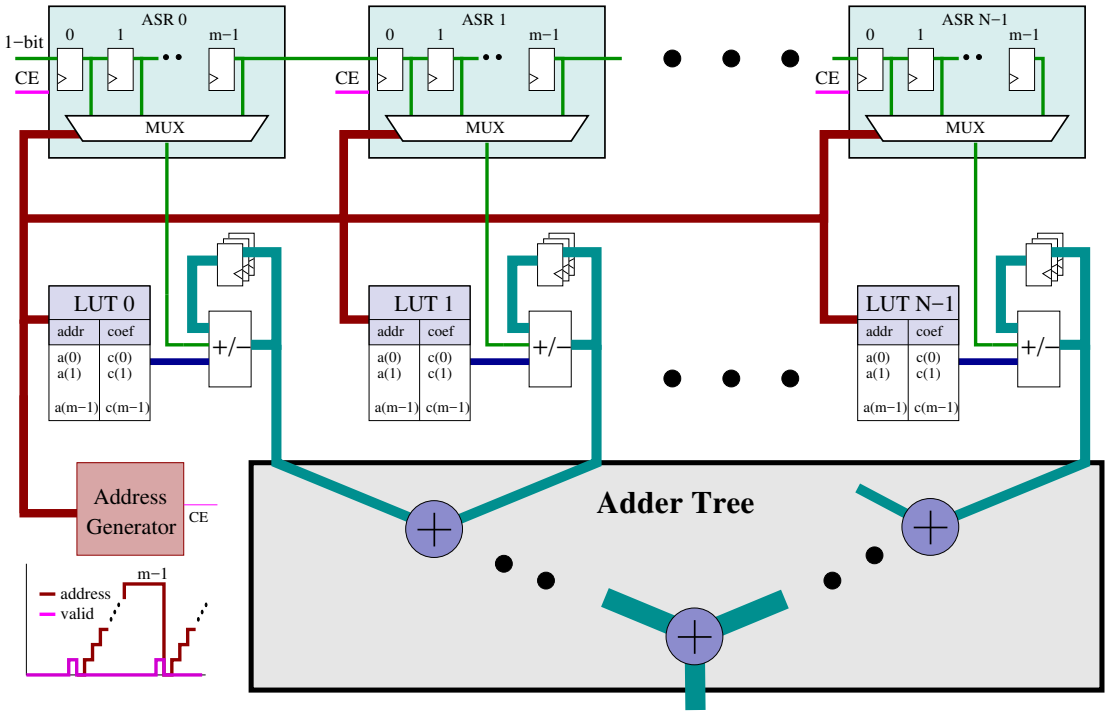


Figure 3.17: Hardware-friendly Modified Matched Filter Building Block

3.3.5 Hardware-Friendly Matched Filtering

Recognizing that the presence of multiple matched filters is very burdensome on the FPGA resources, I next explore an alternate means of finding the starting index at which to demodulate received signals. This alternative method I refer to as a hardware-friendly matched filter (HFMF); I base our high-level designs around the description given in [65], which is illustrated in Fig. 3.17. To save time and resources, the HFMF uses only 1 bit of the input signal and requires no actual multiplications. Instead, the multiply-accumulate (MAC) functionality is handled using bit shift registers and bitwise additions.

The HFMF design relies upon oversampling to achieve its functionality, since it requires a delay of at least 18 samples to complete the filtering operation for each sample, but this can be reasonably reduced to 10 to meet FPGA fabric clock requirements. In our implementation, since the 802.11 HFMF must operate on every raw input sample as shown in Fig. 3.16, I incorporate an upsampling factor of 20 on the input to ensure each HFMF output is calculated in time. In contrast, the LTE HFMF is enabled by a valid input signal that is only true once every 16 samples, so only an

upsampling factor of 2 is needed to calculate the result.

A HFMF that inputs complex data must consist of 4 real HFMFs. The real HFMF consists of multiple addressable shift registers (ASRs), each of which is responsible for carrying out m MAC operations each time a valid input sample is received. Although I still call it "MAC", it must be stressed that multiplication is simplified to addition or subtraction as the input is only 1-bit in size. Such a realization requires a minimum clock frequency equal to mf_{in} where f_{in} is the frequency of the input signal. The addresses for coefficients stored in LUTs and data stored in ASRs are generated by a counter that runs from 0 to $m-1$; the clock-enabled (CE) valid-in signal resets the counter to zero and also enables data shifting through the ASR. The results of all N MAC operations are combined in a binary adder tree to produce the final filtered output.

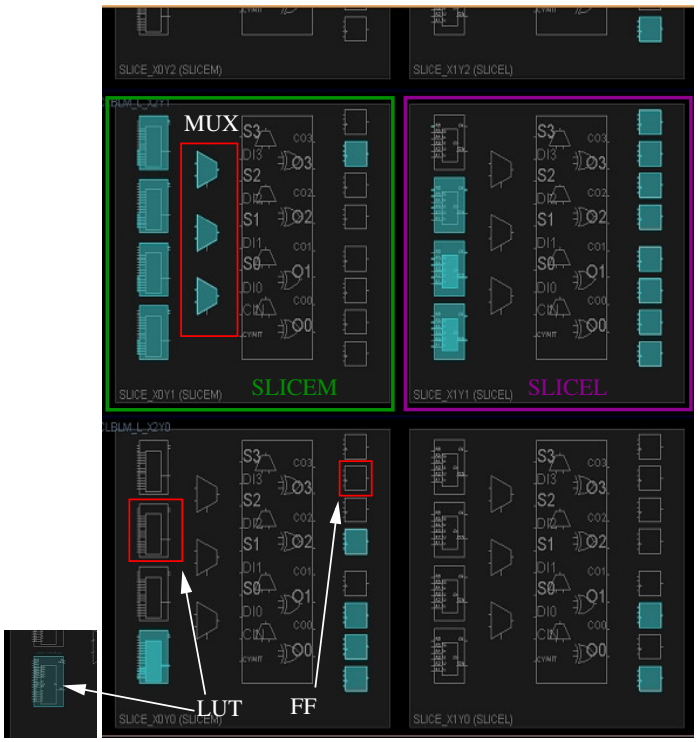


Figure 3.18: 8-bit ASR created using (a) black box VHDL (b) Simulink blocks

As a planned improvement to the design, Simulink Black Box Interface can be used to incorporate a custom HDL design into a Simulink block. Using black box VHDL for ASR has the advantage of mapping the code directly to RAM based shift registers available on the FPGA fabric. These registers can either be 16 or 32 bit wide, and can provide compact and power efficient

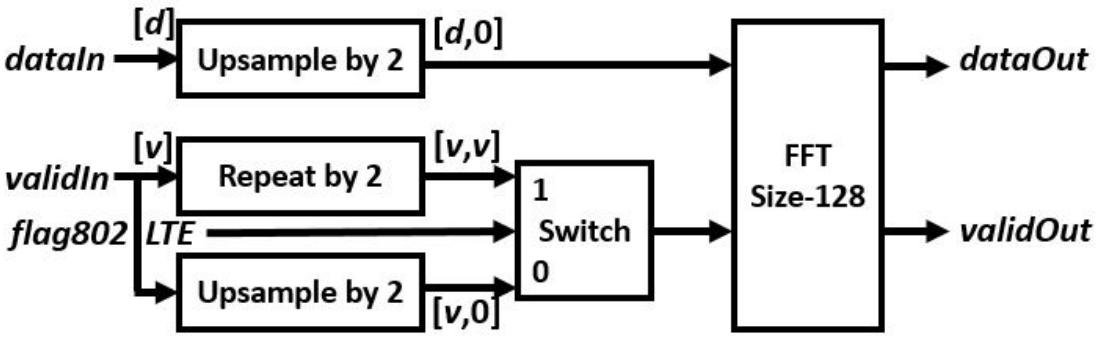


Figure 3.19: OFDM Demodulation for Multiple Numbers of Subcarriers

implementations. Fig. 3.18 (a) shows the FPGA footprint of an 8-bit ASR created using this approach. This ASR only occupies the space of a LUT within a single SLICEM, as opposed to the Simulink-based mapping, which uses 8 LUTs and 16 flip-flops, occupied over 5 slices. Such a high degree of miniaturization is possible when LUTs are configured to operate as RAM based shift registers which can be invoked either by VHDL inference or component instantiation using SRL16E primitives [66].

3.3.6 Joint OFDM Demodulation

As initially shown in Table 3.6, the number of subcarriers in LTE DL can vary between 72 and 1200, requiring FFT sizes from 128 to 2,048. In contrast, 802.11a has a fixed number of subcarriers and so its FFT size is fixed at 64. To handle different numbers of subcarriers, the system designer can incorporate multiple FFT components into his design. Alternately, I propose using a single FFT to support multiple divisions. In our method, I insert only the higher-sized FFT block and inject $2^{N_{LTEFFT}/64-1}$ complex zero samples after every other valid sample of the 64-sample sequence to be demodulated. For example, when $N_{LTEFFT} = 2$, I can oversample by 2 as shown in Fig 3.19.

In this figure, each *Upsample* block oversamples by 2 placing zeros in the intermediate points, and each *Repeat* block oversamples by 2 repeating each input sample. When the 802.11a preamble has been detected, the valid input signal is repeated and the FFT-128 functionally interprets interspersed zeros at the data input. Otherwise, an LTE signal is inferred and only valid input samples are used as data input. This technique is similar to *zero padding*, in which zeros are appended to the end of the sequence to be transformed and only every other sample of the output should be considered valid. In contrast, here I insert a zero after every valid sample, which copies the output in the frequency domain, meaning that the first 64 samples are valid FFT-64 output and the following

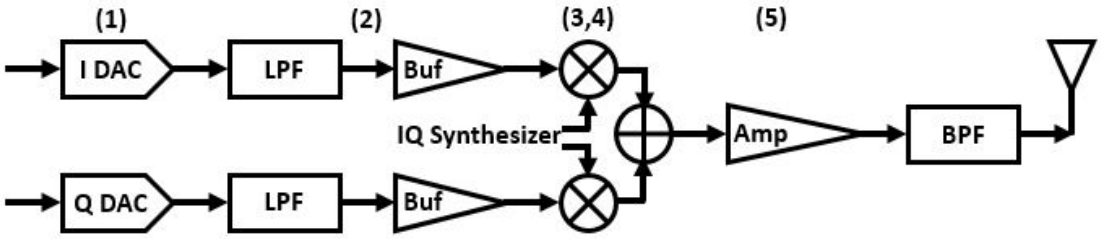


Figure 3.20: Zero-IF Transmitter Chain Defects [8]

64 samples are the same sequence.

3.4 Toward an Online LTE/802.11 Design

In order to transform the design described in Sec. 3.3 to accommodate online operation with live signals, a number of additional processing blocks would be needed within the receiver chain to counter the effects of distortion, offsets, and noise in a communication system. I refer to these anomalies as *defects* or *artifacts* of the transceiver, and in this section I develop models to counter their effects.

The transmit chain that typically exists on a zero-IF transceiver chip like the AD9361 is shown in Fig. 3.20. On the transmit path, there exist the following defects [8]:

1. **Sinx/x Distortion**: A result of the Sample and Hold process that is particularly troublesome at low ADC sampling rates.

2. **DC Offsets**: Positions at which modulator inputs cause carrier leakage in the final RF output, which cause the received constellation to be improperly centered and errors in synchronization.

3. **Phase Noise**: An artifact especially common in SDRs, caused by frequency drift in local oscillators (LOs), which are used to upconvert and downconvert signals between the RF frequency domain and the baseband frequency domain.

4. **IQ Imbalance**: Occurs because the quadrature LO signals are not completely orthogonal; the sine and cosine components of the LO signals are not exactly 90 degrees out of phase, and/or the amplitude of the in-phase (I) modulator is larger than that of the quadrature (Q) modulator, or vice-versa.

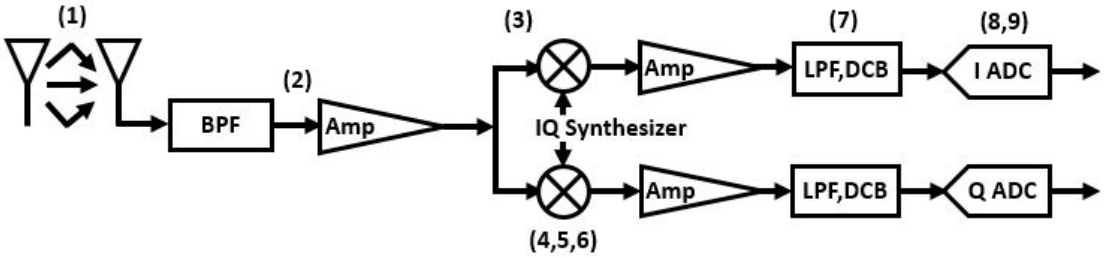


Figure 3.21: Zero-IF Receiver Chain Defects [8]

5. **Nonlinear Distortion**: Caused by the RF power amplifier to meet demanding adjacent channel power ratio (ACPR) and emission mask requirements; resolved by introducing digital pre-distortion (DPD) into the transmit signal.

Similarly, the receive chain that typically exists on a zero-IF transceiver chip like the AD9361 is shown in Fig. 3.21. On the receiver end, I must account for the following issues [8]:

1. **Multipath Channel**: Caused by the transmitted RF signal radiating in many directions off many physical structures and finally arriving at the Rx antenna via multiple paths and time delays.

2. **Thermal Gaussian Noise**: Originating in the resistive part of the antenna and made worse by the Rx noise figure, introduces additive white gaussian noise (AWGN) to the ADC inputs.

3. **Nonlinear Distortion**: Same defect as observed in the transmitter.

4. **Phase Noise**: Same defect as observed in the transmitter.

5. **Frequency Offset**: an *unavoidable problem* that must be dealt with in any communication link, often overlooked by introductory communications textbooks, which arises from the fact that the LO signals created by the phase lock loops (PLLs) in the Tx and Rx feature non-identical frequencies.

6. **IQ Imbalance**: Same defect as observed in the transmitter.

7. **DC Offsets**: Same defect as observed in the transmitter.

8. **Timing Offset**: Caused by the fact that the receiver does not know the perfect sampling instant, and that there is no way to align the clocks driving the Tx's DAC and the Rx's ADC.

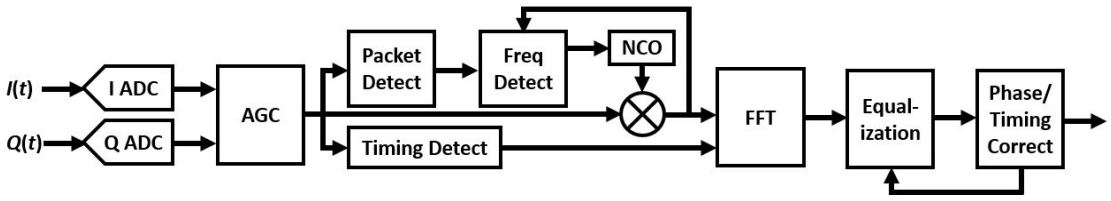


Figure 3.22: Synchronization and Signal Conditioning Blocks [8]

9. **Timing Drift**: Produced by the same non-identical frequencies that cause frequency offset, this causes the timing offset to change over time.

To counter these defects, the OFDM receiver must incorporate the following synchronization and signal conditioning processing blocks, as shown in Fig. 3.22.

- **AGC Compensation**: Ensures via a control loop that the received signal at the ADC output is correctly scaled.
- **Packet Detection**: Relies on AGC at normal levels to search for a unique *pattern* or *signature* in the expected data sequence.
- **Frequency Offset Estimation and Correction**: Detects the frequency offsets in the LO signals and corrects for them to ensure that OFDM demodulation can function correctly, separated into coarse and fine phases.
- **Timing Detection and Correction**: Also known as *Timing Acquisition*, this determines the exact time instances when the FFT block is filled with the proper data samples to be transformed.
- **Equalization**: Accounts for multipath effects by comparing the FFT of a received sequence to the ideal version originally sent by the transmitter.
- **Phase Drift Correction**: Deals with the unresolved frequency offset and phase noise by using the pilot tones embedded into OFDM symbols.

3.4.1 Automatic Gain Control

Controlling the gain of the raw received signal is already partially handled by most RF front end chips. In the past, AGCs have always been custom built for each chip to suit the device and protocol

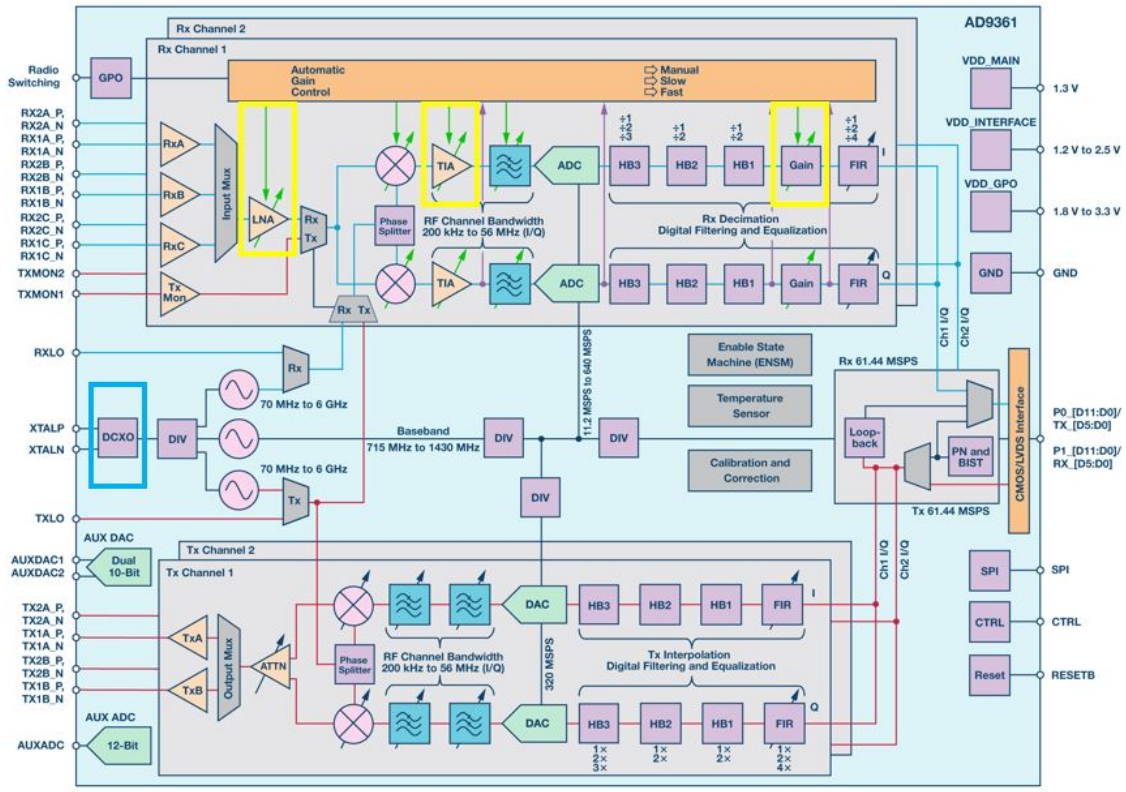


Figure 3.23: ADI AD9361 Chip Gain and Mixer Parameter Settings [9]

specifications. In SDRs, AGCs incorporate tunable parameters; for example, the AD9361 offers slow, fast, or manual AGC settings to automatically control the gain. The system designers may use these automatic settings or incorporate their own functionality for controlling the tunable gain parameters on the AD9361, as shown in Fig. 3.23. The AD9361's analog low noise amplifier (LNA), analog transimpedance amplifier (TIA), and digital gain may be tuned in this manner. The system designer may also tune the digitally controlled oscillator (DCXO) to counter the effects of frequency offset, described in the following section.

However, it is at the discretion of the designer to choose whether to incorporate a general AGC algorithm on FPGA fabric, which manually adjusts the strength of the input signal based entirely upon the raw received sample values. We must recognize that an additional AGC component may raise the signal amplitude more closely to the desired envelope, but would cost additional latency and resources that cannot be afforded. If an AGC component is desired, we can incorporate an additional control system on FPGA to employ the logarithmic loop algorithm, as shown in Fig. 3.24 [10]. In this method, the output signal, x_{AGC}, is an exponential function of the loop gain, x_{Gain}, as shown in

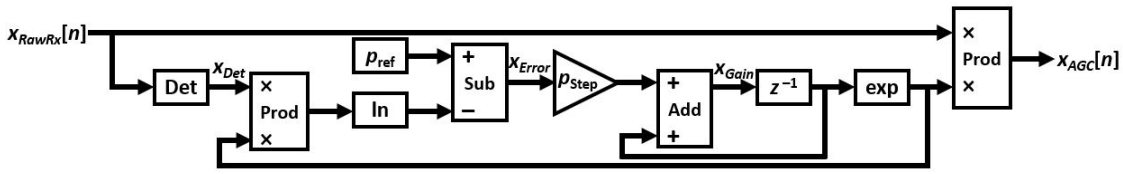


Figure 3.24: AGC Logarithmic Loop Method [10]

equation 3.8.

$$
\begin{aligned}
x_{Det}[n] &= \frac{e^{2x_{Gain}[n-1]}}{p_{Update}} \sum_{k=np_{Update}}^{(n+1)p_{Update}-1} |x_{RawRx}[k]|^2 \\
x_{Error}[n] &= p_{Ref} - \ln(x_{Det}[n]) \\
x_{Gain}[n] &= x_{Gain}[n-1] + p_{Step} \cdot x_{Error}[n-1] \\
x_{AGC}[n] &= x_{RawRx} \cdot e^{x_{Gain}[n-1]}
\end{aligned}
\tag{3.8}
$$

where $x_{Detector}[n]$ is the detector output, x_{Gain} is the loop gain, p_{Update} is the update period, $x_{Error}[n]$ is the error signal, x_{RawRx} is the raw (unprocessed) received signal, p_{Ref} is the reference value, and p_{Step} is the adaptation step size. Here, the *Det* subsystem represents a square-law detector, in which the output, $x_{Det}[n]$, is the normalized sum of the squared values of the input signal, $x_{RawRx}[n]$.

In a practical scenario, we can expect the fixed-point data types to reflect the output of the RF front end. In the case of the FMComms3, these are signed 16-bit fixed-point values with 15 fractional bits and 1 sign bit. Thus, they only allow values between -1 and 1, and $p_{Ref} = 1$ makes an ideal setting. The update period specifies how many samples of the raw received input should be used for generating the detector output. For raw Rx signals that exhibit little change in their envelope amplitude, increasing this parameter can help prevent variations in output signal level, but at the expense of more memory, resources, and clock time. A reasonable setting for FPGA implementation would be $p_{Update} = 10$. Most importantly, the adaptation step size parameter can be tuned to ensure the control system is critically damped. Increasing step size permits the AGC to respond more quickly for steady-state input, but also increases variation in output signal level, like an underdamped system. Decreasing step size prevents AGC from responding too quickly to abrupt changes in signal level, like in an overdamped system. In past tests, we have measured the efficiency of the AGC parameter combinations using attack time, decay time, and gain pumping as metrics. *Attack time* is the time duration in which the AGC responds to an increase in input amplitude. *Decay time* is the

time duration in which AGC responds to a decrease in input amplitude. *Gain pumping* is the variation in the gain value during steady-state operation. To minimize these three metrics, an initial value of $p_{Step} = 0.01$ can be used. As always, the user must be cognizant of the costs associated with these parameters, and choose values that balance the trade-off between accuracy and latency/resources.

3.4.2 Packet Detection and Timing Acquisition

The matched filter discussed in Sec. 3.3.4 is only a first step in detecting a fixed sequence and aligning the FFT input. A more sophisticated technique for online operation would recognize the presence of multiple fixed sequences using multiple stages. The first stage, which we refer to as *Packet Detection*, recognizes a repeated training sequence using autocorrelation estimates as shown in equation 3.9.

$$\begin{aligned} R_{est}[n] &= \frac{1}{N_{avg}} \sum_{k=0}^{N_{avg}-1} x_{AGC}[n-(N-1)+k] \cdot x_{AGC}[n-(N_{avg}-1)+k-N_{per}]^* \\ \sigma_{est}[n] &= \frac{1}{N_{avg}} \sum_{k=0}^{N_{avg}-1} x_{AGC}[n-(N-1)+k] \cdot x_{AGC}[n-(N_{avg}-1)+k]^* \\ r_{comp}[n] &= \frac{R_{est}[n]}{\sigma_{est}[n]} \end{aligned} \tag{3.9}$$

where $x_{AGC}[n]$ is the raw received input signal after AGC, $R_{est}[n]$ is the autocorrelation estimate, $\sigma_{est}[n]$ is the variance estimate, N_{avg} is the length of the averaging operation, N_{per} is the period after which a training symbol repeats. The comparison ratio is compared to a threshold, $r_{comp}[n] > v_{PDthresh}$, and the packet detection flag is asserted when it exceeds this threshold. Similarly, the flag can be cleared when it falls below a lower threshold. In 802.11a, since the short preamble appears first and its training symbols repeat every 16 samples, we set $N_{per} = 16$. The size of the averaging window is set to twice that value, $N_{avg} = 32$, to achieve better accuracy for autocorrelation estimates [8].

The second stage, which we refer to as *Timing Acquisition*, performs a cross-correlation with the second fixed sequence to obtain an accurate time instant at which the received OFDM symbol is most properly aligned with the FFT shift register. In this sense, timing acquisition is equivalent to the packet detection procedure described in equation 3.7. To work around the issue of complex multiplications being very hardware-intensive, we can either use the HFMF described in

Sec. 3.3.5 or employ a quasi-correlation method as shown in equation 3.10.

$$X_{Quasi}[n] = \frac{1}{N_{avg}} \sum_{k=0}^{N_{avg}-1} x_{AGC}[n+k-N_{per}] \cdot L[k]^* \\ L[k] = \text{sign}(\Re\{y_{fixPatn}[k]\}) + j \cdot \text{sign}(\Im\{y_{fixPatn}[k]\}) \tag{3.10}$$

where $X_{Quasi}[n]$ is the quasi-cross-correlation estimate and $L[k]$ is the quantized signum equivalent of the fixed sequence, $y_{fixPatn}[k]$. For 802.11a, since the long preamble appears next and its training symbols repeat every 64 samples, we set $N_{per} = N_{avg} = 64$.

3.4.3 Frequency Offset Detection and Correction

The same autocorrelation approach is used for both the fine and coarse frequency offset estimators. We can formulate the autocorrelation function using equation 3.11.

$$R_{est}[n] = \frac{1}{N_{avg}} \sum_{k=0}^{N_{avg}-1} x[n-(N-1)+k] \cdot x[n-(N_{avg}-1)+k-N_{per}]^* \\ \theta_{est}[n] = \text{atan2}(\Im\{R_{est}[n]\}, \Re\{R_{est}[n]\}) \\ f_{est}[n] = \theta_{est}[n] \frac{f_s}{2\pi \cdot N_{per}} \tag{3.11}$$

where N_{avg} is the length of the averaging operation, N_{per} is the period after which a training symbol repeats, and f_s is the sampling rate of the input data. For 802.11a, $f_s = 20$ MHz. For detecting the coarse frequency offset, we use the long preamble, so $N_{per} = N_{avg} = 64$. For detecting the fine frequency offset, we use the short preamble, so $N_{per} = 16$ and $N_{avg} = 32$. The first equation is referred to as a *sliding average*, and the `atan2()` function can be implemented efficiently using a Coordinate Rotation Digital Computer (CORDIC) algorithm [8].

Once the frequency offset is estimated, we can correct for it using a numerically controlled oscillator (NCO), which can generate a complex sinusoid at the negative of the offset frequency, $-f_{est}[n]$, as shown in equation 3.12.

$$x_{FOC}[n] = x_{AGC}[n] \cdot e^{-\frac{2j\pi n f_{est}[n]}{f_s}} \tag{3.12}$$

where $x_{AGC}[n]$ is the received signal after AGC, and f_s is the sampling frequency. In implementation, we would correct for the coarse offset first and the fine offset second. Thus, we must allow for two separate NCOs to generate complex sinusoids at each computed offset estimate.

While this method is efficient on use of resources, in practice we have seen that this method cannot regularly determine the correct frequency offset. Alternate frequency-domain methods were

shown to compute $f_{est}[n]$ more precisely, but these methods require very low frequency resolution and so require very high FFT sizes, which are not feasible in FPGA-based designs. Other methods for frequency offset estimation shown in use by MathWorks examples have shown reasonable accuracy without overwhelming resources [67]. Ultimately, further study is needed to determine the best method for existing RF and FPGA equipment.

3.4.4 Channel Estimation and Equalization

After a time-based signal has been properly synchronized, we can next account for the multipath channel and phase noise defects. To do so, we must make use of the long training symbols again. Undoing the effects of the channel is handled by the *Equalizer* subsystem, which multiplies the orientation of each received subcarrier by the reciprocal of the corresponding channel response subcarrier, as shown in equation 3.13 [8].

$$\begin{aligned} S_{Rx}[m, m+1, ..., m+n_{FFT}-1] &= \mathcal{F}\{x_{FOC}[m-n_{FFT}+1, ..., m]\} \\ C_{Eq}[m, ..., m+n_{FFT}-1] = \frac{1}{CResponse[m]} &= \frac{S_{Ideal}[m, ..., m+n_{FFT}-1]}{S_{Rx}[m, ..., m+n_{FFT}-1]} \\ C_{Eq}[0, i_{GuardSubcarriers}] &= 0 \end{aligned} \tag{3.13}$$

where $x[n]$ is the received input signal after frequency offset compensation, $C_{Eq}[m]$ are the Equalizer coefficients, $C_{Response}[m]$ is the channel response, $S_{Ideal}[m]$ are the ideal tones from the specification, and $S_{Rx}[m]$ are the received symbol tones at the timing acquisition. The Equalizer, recognizing that its current sample being processed is the last sample of the pattern with which the received signal was correlated, takes an FFT with n_{FFT} points. Since we know the actual FFT of the long symbol beforehand, known as the ideal tones or $S_{Ideal}[m]$, this method can be used to find the *channel response*, $CResponse[m]$. To correct for this channel estimate, we invert the channel response and then zero out the DC subcarrier and other guard subcarriers, in which there are neither data nor pilot symbols, to get the equalizer coefficients, C_{Eq}. Then, we multiply the FFT output for each subsequent synchronized OFDM symbol by these coefficients to get the equalized received input signal, as shown in equation 3.14 [8].

$$\begin{aligned} X_{FOC}[m, ..., m+n_{FFT}-1] &= \mathcal{F}\{x_{FOC}[m, ..., m+n_{FFT}-1]\} \\ X_{Eq}[m, ..., m+n_{FFT}-1] &= C_{Eq} \cdot X_{Eq}[m, ..., m+n_{FFT}-1]] \end{aligned} \tag{3.14}$$

In 802.11a, we can incorporate recognizing that the long training symbol repeats twice, we replace x_{FOC} with the average of the last 64 samples and the 64 samples before that, as shown in

equation 3.15.

$$x_{LongAvg}[m, ..., m+63] = \frac{1}{2}(x_{FOC}[m-63, ..., m] + x_{FOC}[m-127, ..., m-64]) \quad (3.15)$$

In addition, we know the 802.11a guard subcarriers to be fixed at FFT output indices $i_{Guard} = [28, ..., 38]$. The difficulty in FPGA implementation stems from the fact that the equalizer coefficients must be stored in BRAM. Thus, additional delays and counters are required to synchronize the storage of the equalizer coefficients, and their extraction from BRAM to coincide with the FFT output. This is complicated by the fact that LTE DL channels use different FFT sizes, n_{FFT}, and have different indices for their guard intervals, i_{Guard}.

The LTE specifications define different methods for channel estimation that incorporate pilot symbols within reference signals such as CellRS, for which 802.11a has no equivalent. These methods are described in Annex F of [68] and [69]. In these methods, the pilot symbols in CellRS in OFDM symbols 0 and 4 of each time slot are used instead of the SSS and PSS. This method certainly makes more sense for BWs above 1.92 MHz, in which the SSS/PSS only cover a fraction of the subcarriers. The algorithms for performing channel estimation in this manner are documented using MATLAB code in [70], and will be the basis for future FPGA-based designs. However, it is important to note that these methods would require storing most of the LTE resource grid for each subframe into BRAM memory, so that the entire subframe could be equalized at once. This storage of the entire subframe in BRAM is also beneficial for decoding OFDM symbols that appear before the SSS/PSS such as PDCCH and for correcting phase and timing errors, which are described in the next section.

3.4.5 Phase and Timing Error and Drift Correction

After a time-based signal has been properly OFDM demodulated, we can next correct for phase error (a.k.a. IQ imbalance) and drift by making use of the pilot symbols interspersed in specific frequency subcarriers. Using *maximum ratio combining* (MRC), we can optimize the signal-to-noise ratio (SNR) by lessening the impact of pilots that were suppressed by selective fading, as shown in equation 3.16 [8].

$$\begin{aligned} c_{MRC}[k] &= \frac{|X_{Eq}[i_{Pilots}[k]]|}{\sum_{k=0}^{n_{Pilots}-1} |X_{Eq}[i_{Pilots}[k]]|} \\ X_{AvgPhase} &= \sum_{k=0}^{n_{Pilots}-1} c_{MRC}[k] \cdot \angle(X_{Eq}[i_{Pilots}[k]) \\ \Theta_{PhaseError} &= \angle(X_{AvgPilot}) \end{aligned} \quad (3.16)$$

where n_{Pilots} is the number of pilot symbols in an OFDM symbol, $i_{Pilots}[k]$ are the indices of the frequency subcarriers in which the pilot symbols reside, $c_{MRC}[k]$ are the coefficients to be combined with each pilot angle, $X_{AvgPhase}$ is the averaged pilot phase, and $\Theta_{PhaseError}$ is the all-tone phase error. In 802.11a, the pilots are present in each OFDM symbol, $n_{Pilots} = 4$, and $i_{Pilots} = [43, 57, 7, 21]$. In LTE DL, the pilots are in the CellRS, which are only present in 2 OFDM symbols (0 and 4) out of 7 in of each time slot. The number and indices of pilots change based on BW; n_{Pilots} ranges from 12 to 200. In RMC4, for example, $n_{Pilots} = 12$ and the pilots are spaced 6 subcarriers apart $i_{Pilots} = [1, 7, 13, ..., 67]$.

In MRC, the coefficients are set in accordance with their signal strength as indicated by the channel estimate. As a simplification, we could assume that the phase change is small over the course of an OFDM symbol, meaning all tones including the pilots are rotated away from their nominal values by a constant phase error. This method is an equal gain combining average, and is shown in equation 3.17 [8].

$$X_{AvgPhase} = \frac{1}{n_{Pilots}} \sum_{k=0}^{n_{Pilots}-1} \angle(X_{Eq}[i_{Pilots}[k]) \tag{3.17}$$

Timing drift occurs due to the difference in reference clocks at Tx and Rx. Timing acquisition is only about to A positive timing offset, caused The timing error and phase drift are corrected at the same time by finding the slope of the pilot phase errors, as shown in equation 3.18 [8].

$$X_{AvgSlope}[i_{OFDMsymbol}] = \sum_{k=0}^{n_{Pilots}-1} \frac{c_{MRC}[k] \cdot \angle(X_{Eq}[i_{Pilots}[k])}{d_{Subcarrier}[k]} \tag{3.18}$$

where $d_{Subcarrier}[k]$ represents the distance (measured in $\pm$subcarriers) of each pilot symbol from the center frequency, and $X_{AvgSlope}$ is the change in phase per subcarrier. In 802.11a, the distances of the pilot symbols are $d_{Subcarrier} = -21, -7, 7, 21$. In LTE, the number of pilot

To prevent rapid variations in the phase drift estimates, the $X_{AvgSlope}$ parameter is used to create an average slope windowing filter, as shown in equation 3.19.

$$\begin{aligned} X_{AvgSlopeFilt}[i_{OFDMsym}] &= \frac{1}{n_{OFDMsym}} \sum_{k=0}^{n_{OFDMsym}-1} X_{AvgSlope}[i_{OFDMsym} - k] \\ c_{Drift}[m, ..., m + N_{FFT} - 1] &= X_{AvgSlopeFilt}[i_{OFDMsym}] \cdot [-\frac{N_{FFT}}{2}, ..., \frac{N_{FFT}}{2} - 1] \end{aligned} \tag{3.19}$$

where $i_{OFDMsym}$ is the index of this OFDM symbol, $n_{OFDMsym}$ is the number of OFDM symbols for which the average slope is held in memory, $X_{AvgSlopeFilt}[i_{OFDMsym}]$ is the filtered average pilot slope for this OFDM symbol, and c_{Drift} is the vector of phase drift coefficients. To effect a smoother

transition between pilot slopes without using many FPGA resources, we can set $n_{OFDMsym} = 8$. Then, $X_{AvgSlopeFilt}$ is computed from the current and last 7 computed average slope values, and only requires storage of the last 7 slopes in BRAM. We use this average slope to create a vector of coefficients to apply a phase shift correction to each of the N_{FFT} subcarriers.

To remove the phase error and timing offset, we must modify the phases of all information-bearing subcarriers to correct for these estimates, as shown in equation 3.20.

$$X_{PhaseCorr} = X_{Eq} \cdot e^{-j\Theta_{PhaseError} \cdot c_{Drift}} \tag{3.20}$$

Note that the equalizer coefficients mentioned in Sec. 3.4.4 must also be updated to reflect the new calculated phase error and drift, as shown in equation 3.21.

$$C_{Eq} = C_{Eq} \cdot e^{-j\Theta_{PhaseError} \cdot c_{Drift}} \tag{3.21}$$

This feedback loop is also reflected in Fig. 3.22.

Chapter 4

Experimental Results

In this chapter, I describe the experiments that I have performed and the results that I have obtained to show the efficacy of my designs for PHY layer wireless HW-SW systems. First, in Sec. 4.1, I show the results of a wireless system that is not real-time using Ettus Research USRP N210s and running MATLAB on a host PC. Then, in Sec. 4.2, I show the results of a cycle-approximate PHY layer 802.11a wireless system on a heterogeneous architecture, targeting the Xilinx Zynq SoC and ADI FMComms3 RF board. Finally, in Sec. 4.3, I show the results of design techniques for detecting multiple protocols on FPGA, focusing upon the differences between 802.11a and LTE DL-SCH.

4.1 Online 802.11 CPU-based Results

4.1.1 Online CPU-based Experimental Setup

In initial experiments, the USRP N210 is used as an RFFE for performing online transmission and reception. To easily facilitate the system design, we program the system components in MATLAB. The USRP N210 platform [18] is used because it allows us to define the parameters listed in Section 3.1.3.3, connect to a PC host using a gigabit Ethernet cable, and to program it using MATLAB [21]. I use the Ubuntu OS, with send and receive buffer sizes for queues set ensuring that there is enough kernel memory set aside for the network Rx/Tx buffers. I also set the maximum real-time priority for the `usrp` group to give high thread scheduling priority. This change is made by adding a line to the file `\etc\security\limits.conf` that sets the `rtprio` property for the `@usrp` group to 50. The overall setup is shown in Fig. 4.1.

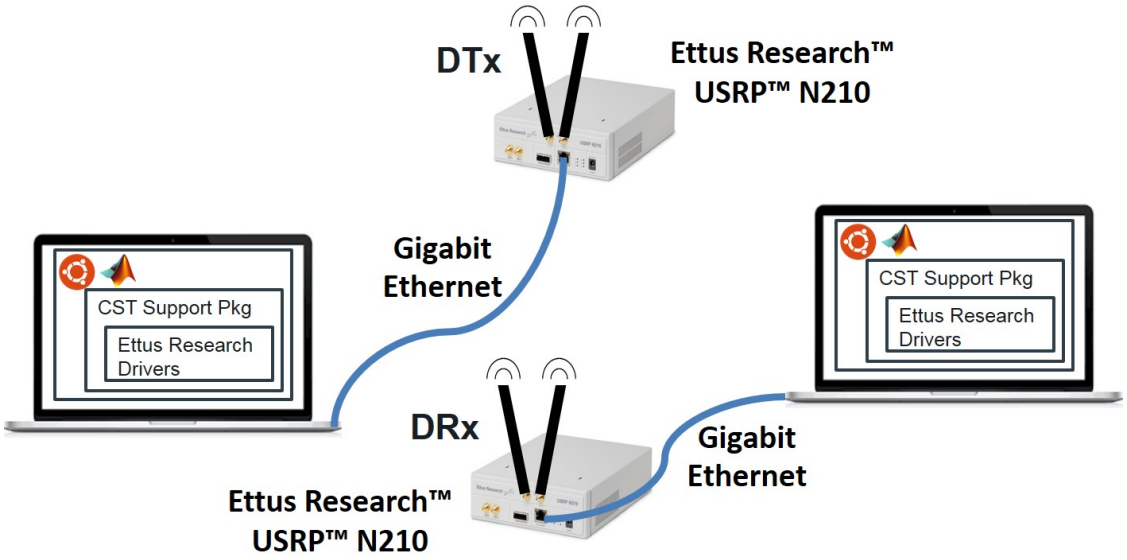


Figure 4.1: Transceiver Hardware Setup

I use the Communications System Toolbox objects for the large part of our design [71]. I used the `comm.AGC` object and the PSK coarse frequency offset estimator that allows us to work with FFT-based options. These objects facilitate easy generation of C code using MATLAB Coder. Here, the `comm.SDRuTransmitter` object puts a frame on the USRP transmit buffer, and `comm.SDRuReceiver` gets a frame from the USRP receive buffer. However, this approach has some disadvantages, such as a requirement for fixed frame length and single-threaded `step` methods.

A number of steps must be taken to make the MATLAB code ready for C code generation using MATLAB Coder. All variables that do not change over the course of the program execution are given a static size and type (including real or complex). All objects are declared as persistent variables as they cannot be passed into MEX functions. The first call to each function tests whether the persistent variable is empty, and initializes each object if true. The transceive, RFFE function code are designed in this same manner.

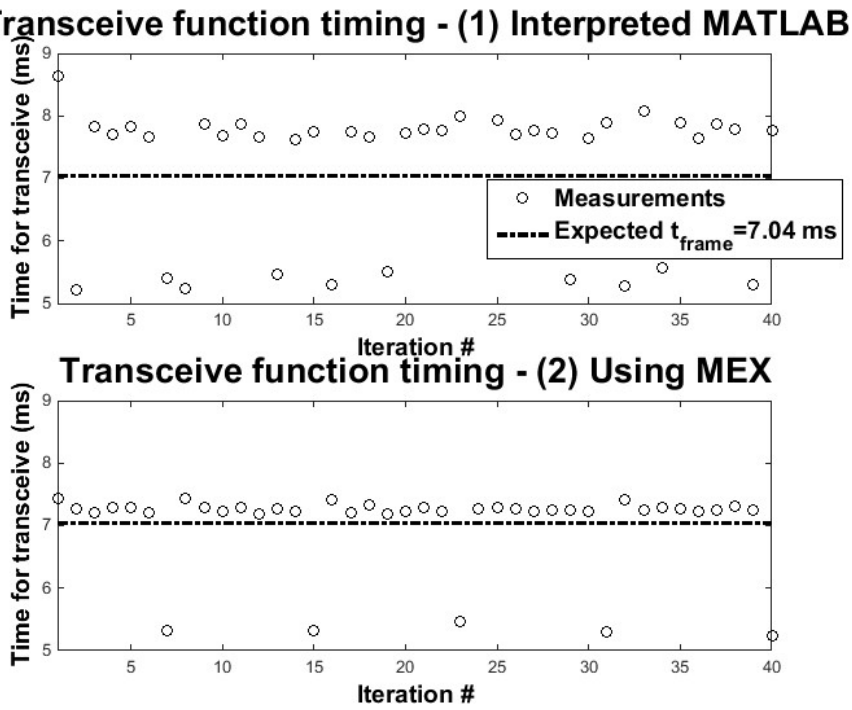


Figure 4.2: Transceive function timing for interpreted MATLAB vs. MEX

4.1.2 Calibration Results

4.1.2.1 Transceive Function Timing

The transceive function is at the core of our system design, since its ability to simultaneously receive and transmit a USRP frame at a near-constant time interval is key to our goal of slot-time synchronized operations. To compare its accuracy, I ran 2,000 time trials to see how long the transceive function takes from start to finish, and how this time difference changes over the course of a longer data bitstream. The timing using a transceive function in interpreted MATLAB and using C code compiled into a MEX are compared in Fig. 4.2. The timing exhibits some deviation. The function initially overshoots the expected time per USRP frame; on every subsequent iteration it then undershoots to make up for the time difference. Note that less undershooting is needed to compensate for initial overshoots, because the overshoot amounts have reduced significantly. The reason for this is that the MATLAB executable has more control over its timing since it does not incur any of the delays associated with interpreted MATLAB code.

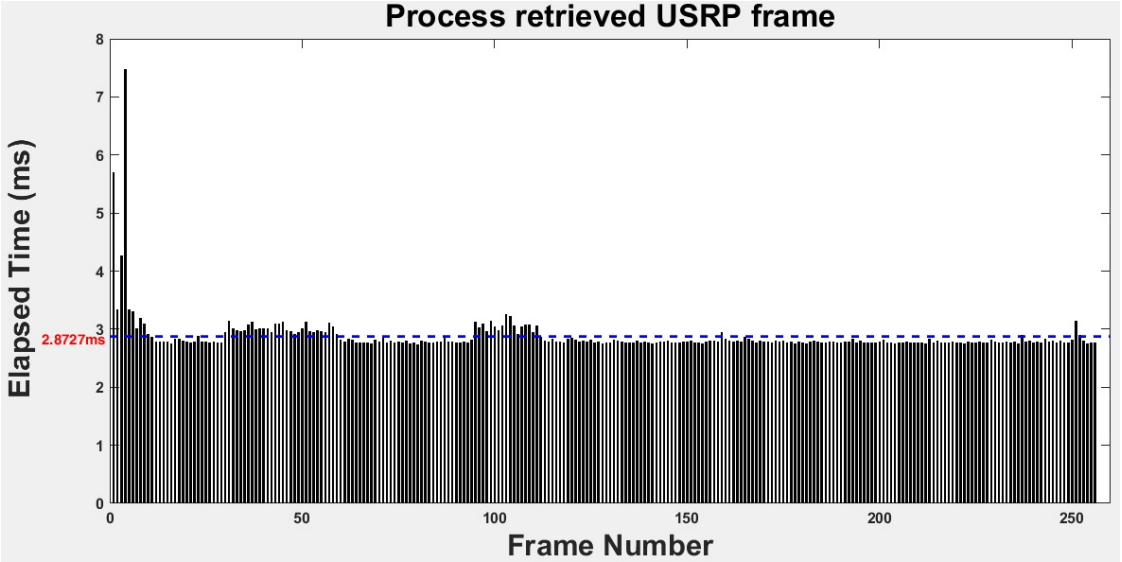


Figure 4.3: Process Time per USRP frame at DRx

4.1.3 Timing DATA Packet Reception at DRx

At the DRx, after preamble detection, the elapsed time to process each retrieved USRP frame corresponding to an entire DATA packet is shown in Fig. 4.3. The dotted line represents the average of all the frame processing times towards a DATA packet reception. The DTx sends out a DATA packet that is made up of 258 USRP frames. After recovering the header bits, the DRx retrieves the payload, which is 250.5 USRP frames (2004 octets). Since the Preamble is 128 bits long, it corresponds to 2 USRP frames. Hence, I account for the reception of (258 - 2) = 256 USRP frames in the DATA packet.

The time to process any given frame usually falls below the desired frame time, t_{radio}, and is fairly constant at 2.87 ms. The first set of frames have a higher processing time because they consist of the MAC header information that must be resolved (e.g. frame control, MAC address).

4.1.4 RFFE Block Timing

The timing of the RFFE block for various values of the frequency resolution parameter in interpreted MATLAB and C code compiled into MEX is shown in Fig. 4.4. The addition of a FIR decimation step in the RFFE block reduces the sampling rate of the input for the subsequent coarse frequency offset estimation (CFOE). This reduction helps in increasing the frequency resolution, currently set at 100 Hz, which is the key parameter in controlling the execution time of CFOE. Further, I benefit

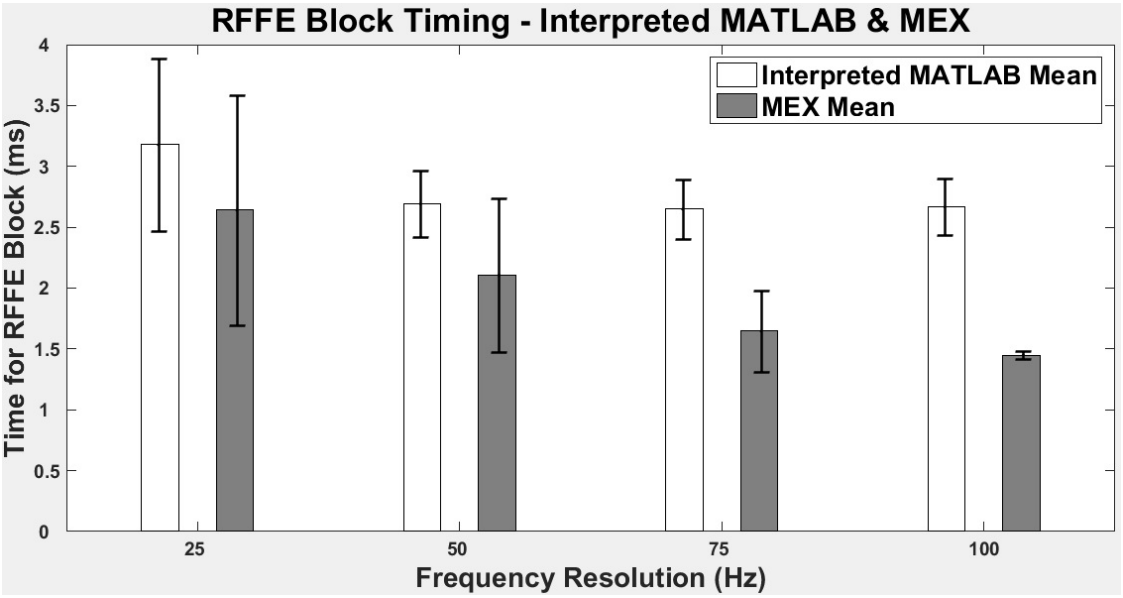


Figure 4.4: RFFE block timing using interpreted MATLAB and MEX

from the improved accuracy of CFOE in that it corrects the signal so well that the later preamble detection block produces the correct synchronization delay to detect the start of DATA/ACK packet. The results clearly establish that average execution time for the RFFE block decreases with increase in frequency resolution. The reason for this is that CFOE uses progressively smaller FFT lengths. As before, the average execution time using MEX is generally smaller than using interpreted MATLAB. Also, the standard deviation for MEX results is always significantly less. Hence, MEX is a better option for the purpose of enforcing consistent RFFE execution times, which is required for slot-time synchronized operations.

4.2 Offline 802.11 CPU/FPGA Results

Next, having seen that the CPU-based approach cannot meet the timing requirements needed for a standard-compliant 802.11 system, I performed experiments in HW/SW codesign using the Zynq SoC. In these experiments, the system was designed in Simulink and both C and HDL code were auto-generated to target the CPU and FPGA components. I collected metrics on timing, resource utilization, and power to gauge the effectiveness of the design.

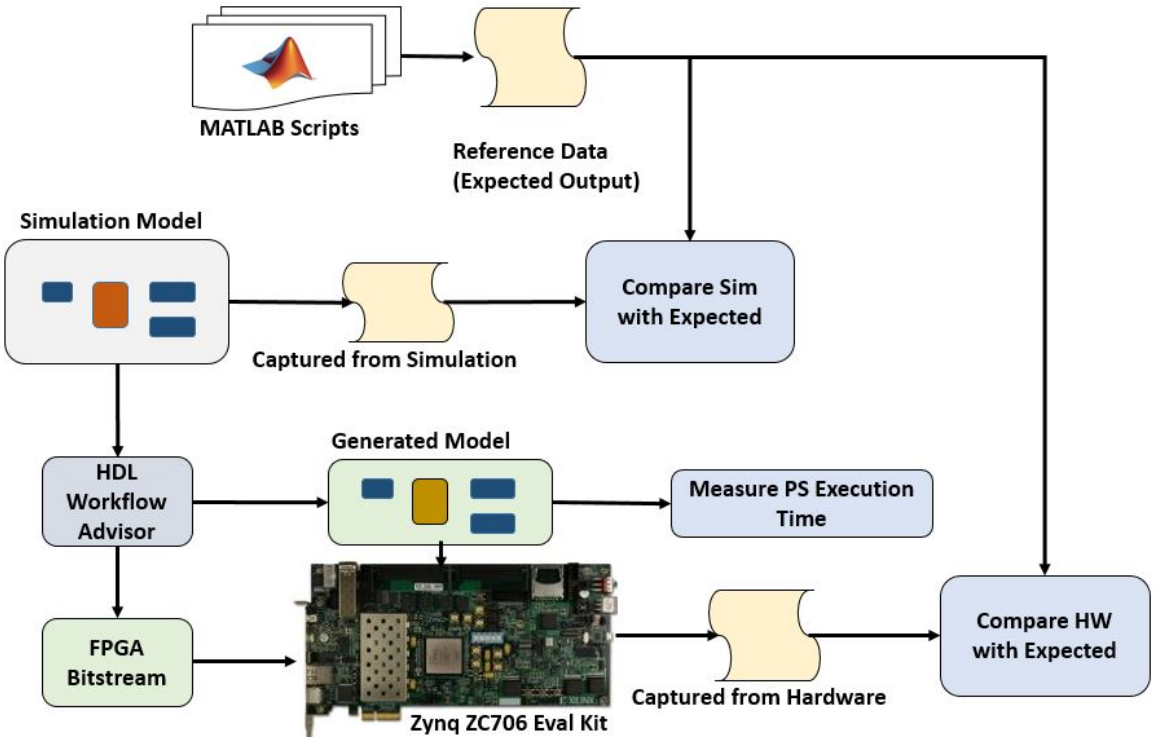


Figure 4.5: Unit Test Workflow Diagram

4.2.1 HW/SW Codesign Testing Workflow

Fig. 4.5 illustrates the unit test workflow. I start by creating MATLAB programs to generate and process data using 802.11a modulation and encoding schemes. These programs were used to create reference data that correspond to the expected output of the Tx and Rx. In Simulink, I develop or modify each model to create PL implementations of a given component, and then I compare the Simulink output to the reference data. Since the Simulink output often has a time offset for the initial frames, other scripts were developed to capture only the relevant data and compare it to the expected output.

4.2.2 Timing Results

For the 802.11a Tx, the execution timing results on the PS are shown in Fig. 4.6. 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

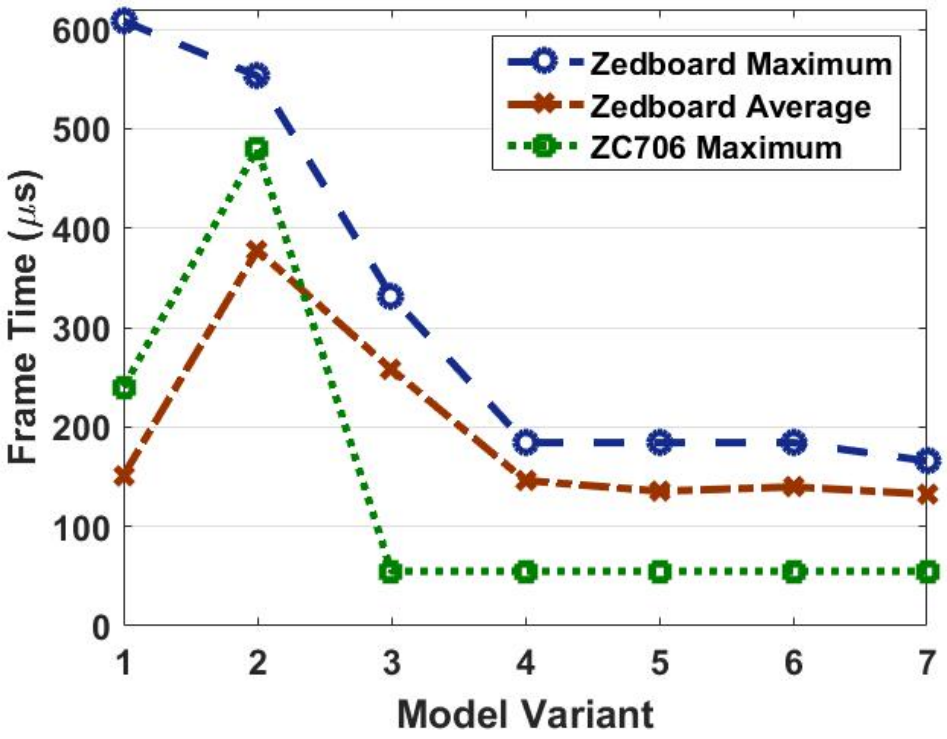


Figure 4.6: Transmitter Frame Times on Zedboard & ZC706

between V3 and V6, I 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, I would need to meet a 4 μs maximum frame time. Thus, further optimization is needed to reduce the PS frame time.

For the Rx, the execution timing results on the PS are shown in Fig. 4.7. 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, I look for the maximum data path delay of the Tx and Rx, which are shown in Table 4.1. 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

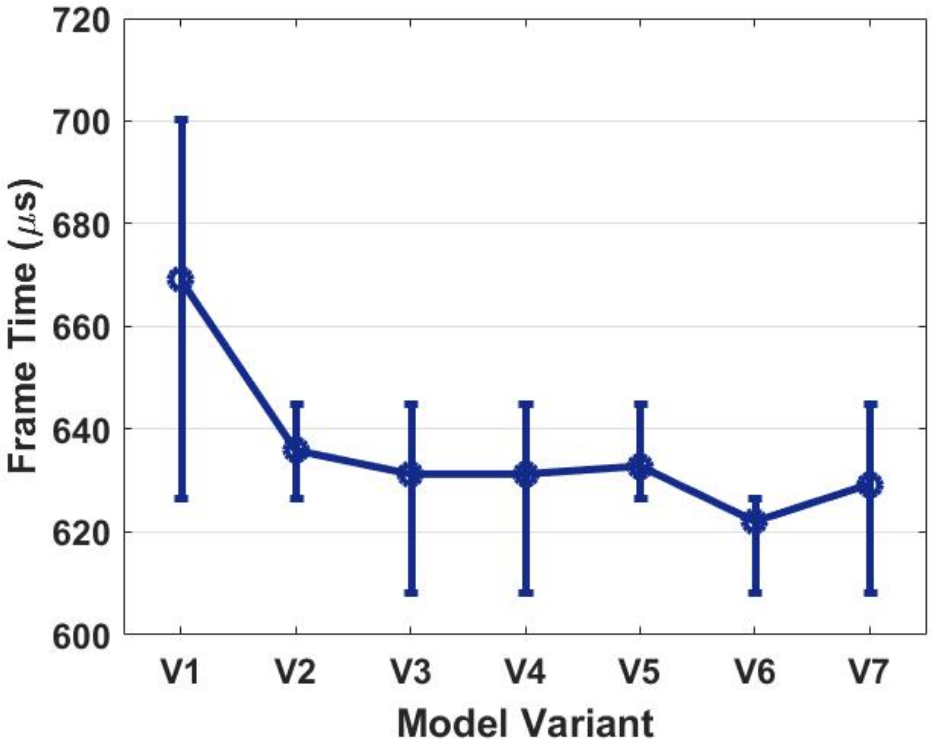


Figure 4.7: Receiver Frame Times on ZC706

Table 4.1: 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

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, I could reduce the number of data dependencies, thereby decreasing the path delay significantly.

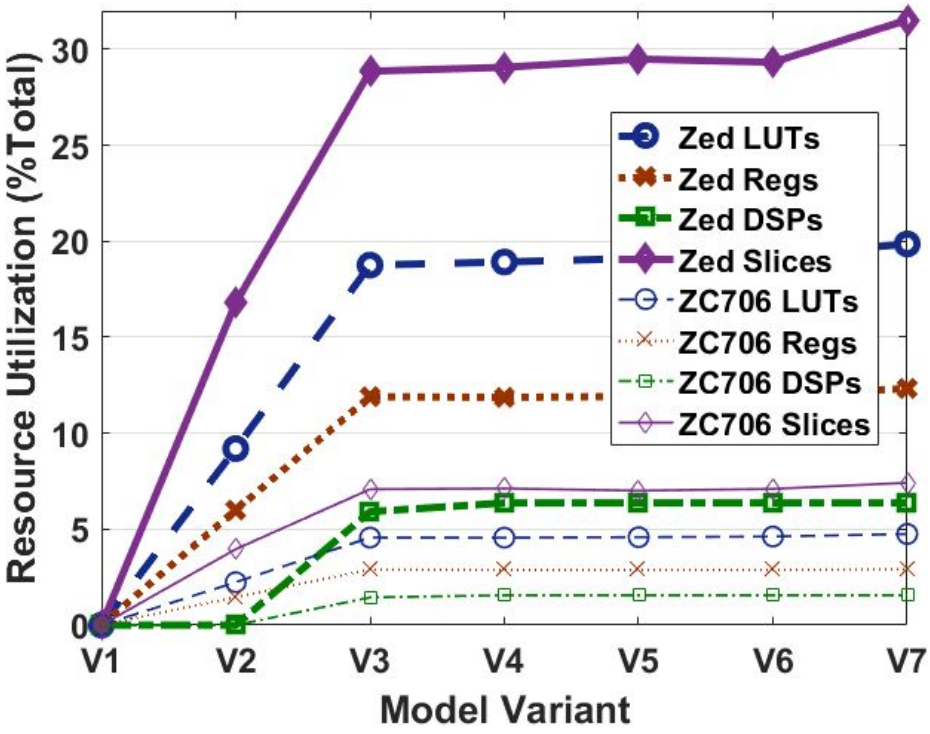


Figure 4.8: Resource Utilization for Tx on Zedboard

4.2.3 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. 4.8.

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. Thus, I decide to use the Zedboard for the Tx because it is a smaller chip that uses less energy while fully containing all functional logic.

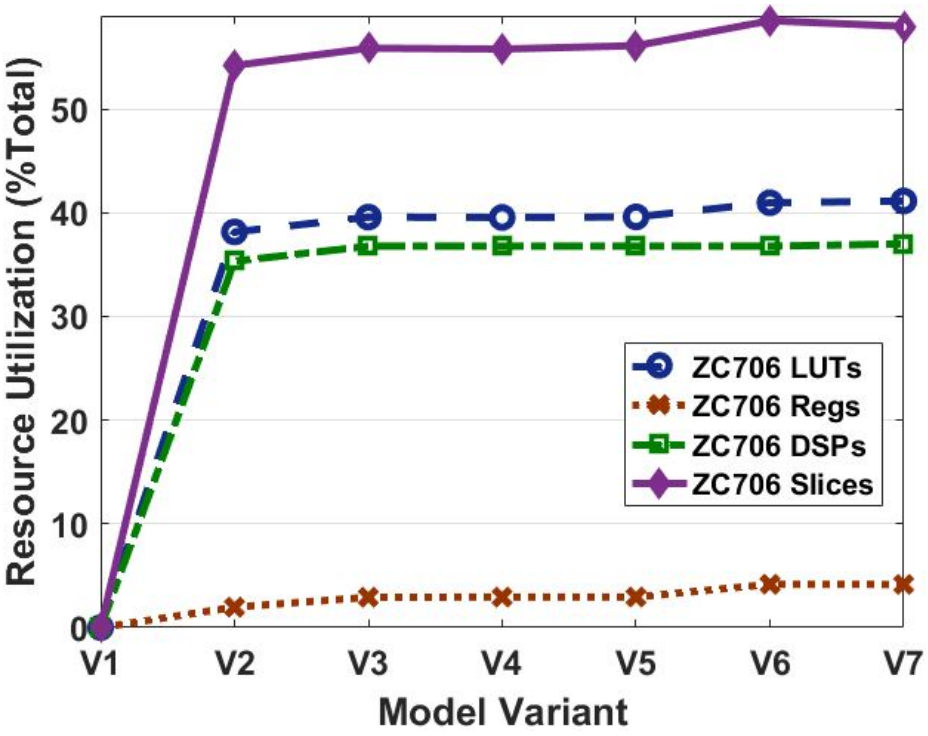


Figure 4.9: Resource Utilization for Rx on ZC706

The Rx resource utilization on the ZC706 is shown in Fig. 4.9. 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, I 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. 4.10.

4.2.4 Power Efficiency

In addition to meeting timing and resource requirements, I 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 naturally more power efficient than alternative processors such as the x86 available on

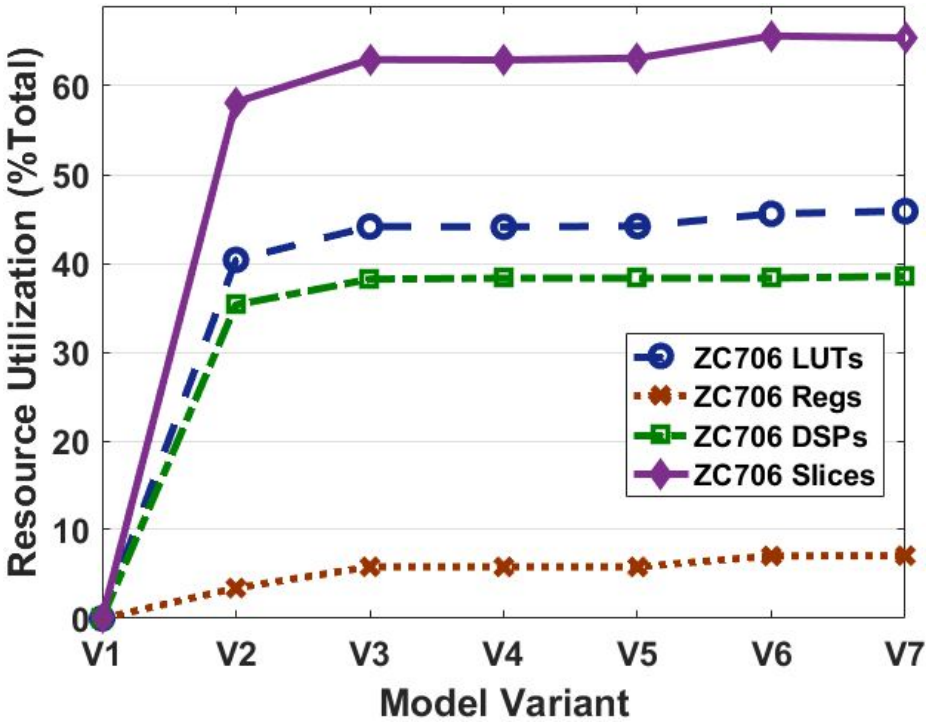


Figure 4.10: Resource Utilization for Combined Tx-Rx on ZC706

the host processor. The Zynq platform always provides power to the ARM processor, therefore using FPGA fabric adds to the overall power consumption even though the FPGA fabric is more power efficient. The amount of 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 I 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). The Tx and Rx power consumption on the Zedboard and ZC706, respectively, are shown in Table 4.2.

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, I see a significant decrease when BPSK is

Table 4.2: Power Usage, Tx on Zedboard, Rx on ZC706

	Tx (W)	Rx (W)
V1	1.530	1.566
V2	1.819	2.343
V3	1.840	2.354
V4	1.845	2.111
V5	1.844	2.106
V6	1.847	2.111
V7	1.842	2.115

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.

4.2.5 Variants of Processing Blocks

Next, I 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.

4.2.5.1 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, I 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 4.3.

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

Table 4.3: 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

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, I 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, I 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.

4.2.5.2 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, I prototyped different versions of the Viterbi Decoder (VD) processing block for model variant V6, which are shown in Table 4.4. By using the default delay-based version, I observed the lowest

Table 4.4: 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

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, I 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).

4.2.6 Next Generation Enhancements

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

4.2.6.1 OFDM Block IFFT Size Variants

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, I note that OFDM is used by both protocols. However, to support OFDM for both protocols would require

Table 4.5: 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

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, I can vary the IFFT sizes required by LTE OFDM and collect metrics to identify the impact of the different size, as shown in Table 4.5. The results 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, I may reserve AXI-lite channels for handling infrequent control messages from the PS to the PL.

4.2.6.2 MIMO Spatial Diversity

The IEEE 802.11n standard describes the extension of 802.11g for MIMO. Since the ADI FM-Comms3 supports MIMO with 2 transmit channels and 2 receive channels, our platform allows for further exploration of spatial diversity. By using multiple antennas, I 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, I 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,

Table 4.6: 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

surpassing the number of available slices. However, using the reduced preamble detection method described in 4.2.5.1, I can accommodate multiple receive chains on the FPGA. I modified model V7 for both the transmitter and the receiver, and capture the results in Table 4.6.

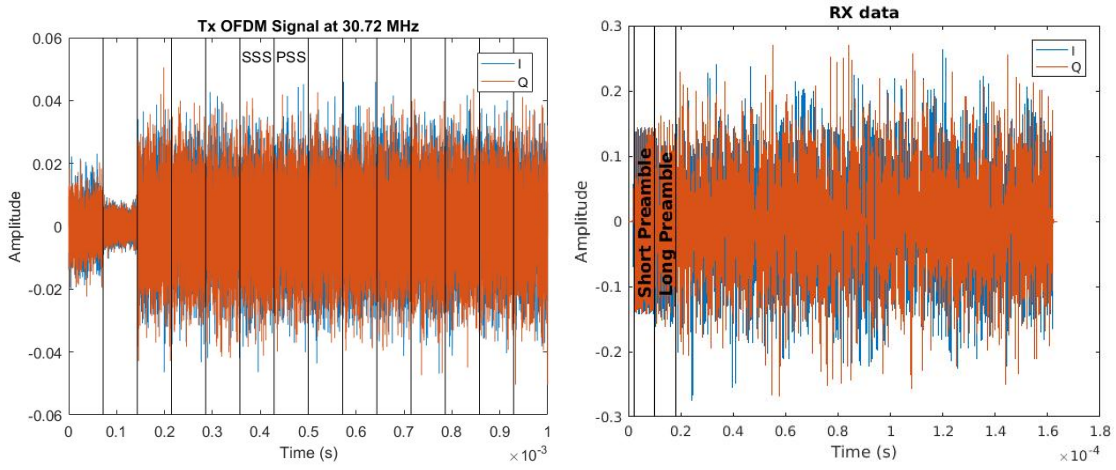
The results show that multiple transmit and receive chains can be implemented on FPGA fabric with only minor changes to data path delay. Duplicating the preamble detection (PD) block for the receive chains doubles the number of DSPs and slices used, as well as the power for that processing block. However, the total power only increases by a fraction. By using multiple antennas and transmitting identical sequences, I 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.

4.3 Offline LTE/Wi-Fi FPGA-based Results

Having shown both the value of using FPGAs for cycle-approximate wireless system design and the considerations needed for conserving resources, we next show the effectiveness of our FPGA-based designs for detecting both the Wi-Fi and LTE protocols on the same bandwidth.

4.3.1 LTE/Wi-Fi Experimental Setup

I used MATLAB R2016b to generate the 802.11a and LTE signals in Fig. 4.11. The encoding and modulation of the signals was performed step-by-step using system objects in Communications



(a) LTE Downlink Signal

(b) 802.11a Resampled Signal

Figure 4.11: Test receive signals: LTE Downlink and 802.11a

System Toolbox and functions in LTE System Toolbox. The 802.11a signal was generated at 20 MHz and then resampled at 30.72 MHz. The LTE signal is the first subframe of RMC4. I used Simulink models to capture the rate transition, matched filtering, and OFDM demodulation functionality. I target the Xilinx ZC706 evaluation kit, which features the Zynq Z-7045 System-on-Chip. I generate HDL code and build a Vivado 2015.4 block diagram and bitstream using the MathWorks HDL Workflow Advisor IP Core generation workflow, as described in Sec. 3.2 [59].

I performed three sets of experiments to gauge the efficiency of our designs. In the first set, I implemented the rate transition and traditional matched filter algorithms and varied the n_{rfc} and n_{bpw} parameters to see their effect on resource utilization and detection accuracy. In the second set, I implemented the rate transition and hardware-friendly matched filter algorithms to show the effect of using the HFMF to detect only the 802.11a preamble, the LTE PSS, or both. In the third set, I implemented the rate transition, HFMF, and OFDM demodulation processing blocks to analyze the effects of using either 2 FFTs of size 128 and 64, or a single FFT of size 128.

4.3.2 Traditional Matched Filter Results

Our first models prototyped the traditional matched filter for both 802.11a and LTE protocols, varying the filter length and fixed-point word size parameters. The FPGA clock cycle, resource utilization, and power results are shown in Table 4.7. The clock cycle time is the maximum data path delay reported by Vivado. The power results were found by running the Xilinx Vivado Power Report with

Table 4.7: Traditional Matched Filter and Rate Transition FPGA Resource Utilization Results

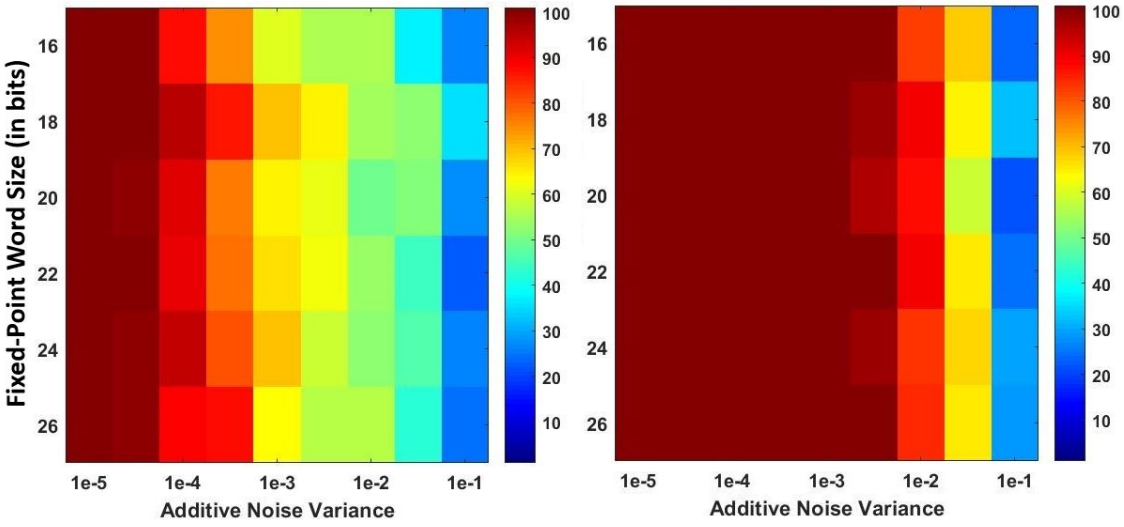
n_{rfc}	24					
n_{bpw}	16	18	20	22	24	26
Clock Cycle (ns)	74.9	83.2	81.1	85.8	85.6	94.6
LUTs	6.2%	7.0%	18.2%	27.5%	24.9%	69.5%
Registers	2.8%	2.5%	8.1%	8.9%	7.5%	12.7%
DSPs	66.0%	66.0%	65.1%	65.1%	100%	99.1%
Slices	11.4%	12.6%	27.6%	37.5%	35.5%	84.6%
Power (mW)	26	43	46	48	63	56
n_{rfc}	30					
n_{bpw}	16	18	20	22	24	26
Clock Cycle (ns)	76.3	81.7	80.8	85.9	85.9	94.3
LUTs	6.4%	7.3%	18.5%	27.8%	26.3%	71.6%
Registers	3.0%	2.8%	8.3%	9.1%	8.1%	13.5%
DSPs	68.0%	68.0%	67.1%	67.1%	100%	99.1%
Slices	11.8%	13.1%	28.2%	38.4%	38.6%	86.6%
Power (mW)	25	45	48	50	68	56

fixed environmental settings (e.g. output load 5pF, ambient temperature 25°C).

Foremost, the results show a reliance upon the Xilinx XtremeDSP48 slices (DSPs). DSPs are ideal for the traditional matched filter because they require such a large number of multiply and accumulate operations. The large increase in DSP usage moving from 24 to 26 bits demonstrates this reliance and why the fixed-point word size should be carefully chosen. To keep DSP usage reasonable, the word size must be 22 bits or fewer. Next, raising the FIR filter length from 24 to 30 causes only a slight increase in resources and power, but not to clock cycle.

Next, I introduce additive white Gaussian noise (AWGN) of variances 1E-5 to 1E-1 to the received 802.11a signals and measured how often the preamble was found and sequence decoded correctly. I ran 100 trials varying the same parameters and plotted the results in Fig. 4.12.

In each image plot, the darkest red represents most accurate and the darkest blue represents least accurate. Comparing the two images, it is evident that increasing the filter length to 30 shows a major improvement in accuracy, especially for noise variances between 1e-4 and 1e-2. For this reason, I use an FIR filter length of 30 in all subsequent experiments.



(a) Filter Length 24

(b) Filter Length 30

Figure 4.12: Receiver Accuracy as a Percent of 100 Trials

Table 4.8: Rate Transition and HFMF Resource Utilization

Model Variant	802.11	LTE	Both
Clock Cycle (ns)	82.9	86.0	13.4
LUTs	5.3%	5.6%	2.6%
Registers	2.6%	2.9%	2.5%
DSPs	39.6%	40.0%	11.1%
Slices	10.0%	10.9%	6.2%
Power (mW)	20	275	63

4.3.3 Hardware-Friendly Matched Filter Results

To test the design with rate transition and HFMF, I created three model variants. The first model incorporates an HFMF to detect only the 802.11a preamble. The second model incorporates an HFMF for the LTE PSS only. The third model incorporates 2 HFMFs to detect either fixed sequence. The hardware-friendly matched filter utilization results are shown in Table 4.8. Immediately, the benefits of using an HFMF for either or both protocols is evident. By replacing the 802.11a matched filter with the HFMF, I see an immediate decrease in utilization of all resources and power. In contrast, introducing the LTE HFMF causes a decrease in utilization but an increase in power estimates, the

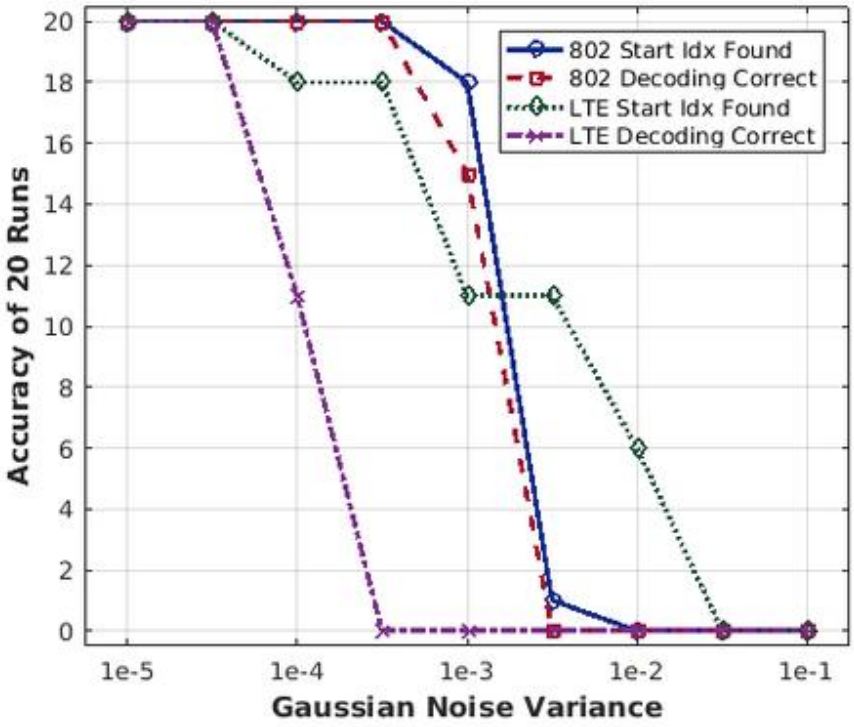


Figure 4.13: Rate Transition and HFMF Accuracy out of 20 Trials

root of which are from the traditional 802.11a MF. By replacing both MFs with HFMFs, I see the lowest resource utilization, a reasonable power usage, and a clock cycle below the 32.5 ns required for this sampling rate.

Next, I ran 20 trials varying the AWGN variance parameter, and gathered the accuracy results for the HFMF shown in Fig. 4.13. The HFMF shows approximately the same level of accuracy for finding the start of and decoding 802.11a signals up to a noise variance of 1e-3, but fails to operate afterwards. Thus, the HFMF design is preferable for such situations where the signal's SNR is high. The LTE signal shows less accuracy than the 802.11a signal, which shows that the LTE HFMF is more susceptible to noise.

4.3.4 OFDM Demodulation Results

The utilization results that include rate transition, HFMF, and OFDM demodulation are shown in Table 4.9. Using a single size-128 FFT shows improvement over the dual-FFT design in multiple ways. There is a 0.5% decrease in clock cycle and a 12.5% decrease in power. Also, the design

Table 4.9: OFDM Demodulation Resource Utilization

Model Variant	2 FFTs (64&128)	1 FFT-128
Clock Cycle (ns)	12.02	11.96
LUTs	4.9%	3.9%
Registers	4.2%	3.4%
DSPs	12.8%	12.1%
Slices	9.8%	7.6%
Power (mW)	16	14
Min Latency (smp)	173	307

requires 20% fewer LUTs, 19% fewer registers, and 23% fewer slices. I performed accuracy tests for 20 trials and saw similar results to those shown in Fig. 4.13. Thus, I have shown that using a single FFT for OFDM demodulation of protocols with different numbers of frequency subcarriers is possible and can be more efficient. I expect that the real value of this feature will be made clear with future versions that support multiple LTE bandwidths as was shown in Table 3.6. The downsides of using this single FFT approach are increased latency and additional high-level complexity. Foremost, the FFT-128 has a latency of 307 samples while the FFT-64 is 173 samples. Thus, in the single-FFT approach, the output isn't valid until after 134 multiples of the design clock cycle. Also, additional indexing is required in post-processing to put the output samples in the correct order.

Chapter 5

Conclusion

This dissertation research has demonstrated a method for modeling next generation protocol coexistence, and the implementation of a joint Wi-Fi/LTE processing chain as a proof of concept.

In my initial work prototyping 802.11b, I show the necessity of state-action diagrams and slot-time synchronized operations in order to realize an IEEE 802.11b standard-compliant PHY layer. In addition, the system allows the user reconfigure the parameter values as needed. I conclude that building our design around the concept of frame-time synchronized operations results in a system that adheres to our desired frame time and is able to reconfigure parameter values as needed. Using MEX is essential for realizing timing with little deviation from this frame time. Using the MATLAB Coder to automatically generate MEX functions, is beneficial in improving the speed consistency of our system blocks, most notably RFFE, which can vary its frequency resolution parameter.

In the HW-SW codesign studies, I have introduced and explored a method for exploring HW-SW co-designs for 802.11a wireless transmission and reception systems. I 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. However, I note that the entire transmitter PHY chain can easily fit on the Zedboard PL, and the entire receiver chain can fit on the ZC706 PL. 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. I 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.

The LTE/Wi-Fi coexistence research has demonstrated that it is possible to develop PHY layer transceivers that can accommodate multiple wireless protocols. I have tested our FPGA-based designs to prove that devices can effectively determine the start of a signal following either 802.11a

or LTE PHY layer specifications. I have identified several key parameters, the resampling lowpass filter length and the number of fixed-point bits per sample, for either improving accuracy or reducing FPGA resource utilization and power. I have explored the use of alternate non-matched filter based approaches to further reduce utilization and clock cycle. Finally, I have introduced a technique for reusing the same FFT block for OFDM demodulation of both protocols and demonstrated how it can be more efficient.

By analyzing the commonalities and differences between Wi-Fi and LTE processing blocks, I introduce a practical example in which the wireless networking community will be particularly interested. In performing this research, I revealed a new path and procedure for studying protocol coexistence that can be utilized and expanded upon by the wireless networking and HW-SW systems design research communities.

5.1 Future Work

In my future role as a tenure-track assistant professor, I plan to address many of the issues addressed in this research. I plan to complete the LTE DL-SCH and 802.11a PHY-layer receiver designs on FPGA fabric. In addition, I plan to add additional functionality to enable online operation, including the corrections of RF artifacts such as frequency offset and I-Q imbalance. In future research, I plan to prototype and test my designs with live, online signals. I plan to perform tests with online radio transmissions and measure bit error rate (BER) for the different co-designs. These PHY layer implementation will also be used as a basis for future work in higher layers (e.g. MAC). I plan to expand upon our investigation to prototype MIMO configurations such as spatial multiplexing and beamforming. The system will be adapted to other modern wireless standards such as 802.11ac for beamforming, 802.11af for UHF band reuse, and next generation 5G LTE technologies. For future implementations, I plan to target the Xilinx Ultrascale+ MPSoC, which will allow for larger FPGA-based designs and multiple processors.

For design space exploration, more algorithm refinement is needed to improve the performance of the bottleneck behaviors, most especially the matched filer, Viterbi decoder, and FFT/IFFT. The use of fixed-point data types instead of double-precision floating point is expected to significantly improve the performance of such behaviors as the matched filter, but more detailed analysis would need to be performed to determine how much this affects accuracy. Specification refinement could be improved to more fully exploit parallelism in these target behaviors. In addition, further architecture refinement is needed to explore the use of the FPGA PE as an alternative to custom HW

Table 5.1: Theoretical Time Parameters

Var	Description	Est Time	Est Freq
t_C	FPGA Logic Fabric Clock	5-10 ns	100-200 MHz
c_{ST}	Constant # Logic Stages	8	
t_D	FPGA Data Path Delay	40 ns	25 MHz
t_S	Sample Time	50 ns	20 MHz
t_F	Frame Time	4 μs	250 kHz
c_U	Constant Upsampling Factor	1000	
t_P	Processor Update Time	4 ms	250 Hz

for components that rarely need to be modified. For this exploration, I would need to synthesize HDL from SpecC behaviors for real-time operation on a FPGA or SoC.

One broad area of my future research explores HW-SW co-design in more detail by establishing a means for estimating important metrics directly from a model. A major problem is that designers don't know how long a Tx or Rx chain takes, and how much space or energy it requires. As the first part of my research, I derive metrics for timing, utilization, and energy from the processing blocks that make up a chain. This would allow for high-level prototyping of a system to make decisions about number and location of HW-SW divide points early in the design process, before synthesis and implementation. As an example of how timing metrics can be derived, I address the timing-related parameters shown in Table 5.1. Based on the location of the HW-SW divide points, I can ascertain which processing blocks and requisite data packing blocks must reside on each computing element. The presence of processing blocks on the FPGA adds to the data path delay, t_D. The data path delay is a multiple of the FPGA logic fabric clock, $t_D = c_{ST} \times t_C$, where c_{ST} refers to the number of intermediate stages in the FPGA portion of the transceive chain. MathWorks HDL Coder provides rudimentary facilities for estimating these delays. To confirm my estimates, I call upon Xilinx Vivado to calculate them from a synthesized or implemented design.

Similarly, I address the utilization-related parameters shown in Table 5.2. The resource utilization of the HW component for a codesign includes its number of logic slices, LUTs, 1-bit registers (a.k.a. flip-flops), DSPs, and BRAM blocks. A single logic slice on a Zynq contains 4 LUTs and 8 registers. One difficulty with estimation is that a single slice may contain LUTs or

Table 5.2: Theoretical Utilization Parameters

Var	Description	Zedboard	ZC706
u_S	Number of Logic Slices	13,300	54,650
u_L	Number of LUTs	53,200	218,600
u_R	Number of Registers	106,400	437,200
u_D	Number of DSPs	220	900
u_B	Number of BRAM blocks	140	545
p_H	Power of HW component		
p_S	Power of SW component	1.530 W	1.566 W

registers used by different processing blocks. Another complication arises from the fact that a single processing block (e.g. matched filter) may be implemented using different resources. Running the Vivado synthesis and implementation process with minor deviations may cause the block to use either mostly DSPs or mostly LUTs. Finally, the power of the HW component may be derived as a linear combination of its resources utilized, as shown in Equation 5.1.

$$p_H = u_S * c_S + u_L * c_L + u_R * c_R + u_D * c_D + u_B * c_B \tag{5.1}$$

The power of the HW component is usually small compared to the power of the SW component. According to the Vivado power report, the ARM processor uses a constant 1.53 W on the Zedboard and a constant 1.566 W on the ZC706. However, if more components were placed onto HW, then the ARM processor could spend more time in idle mode, in which case it would use only a fraction of its active power. As a possible extension, I can prototype this idle operation on the ARM and use a power meter to test the overall consumption.

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.

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

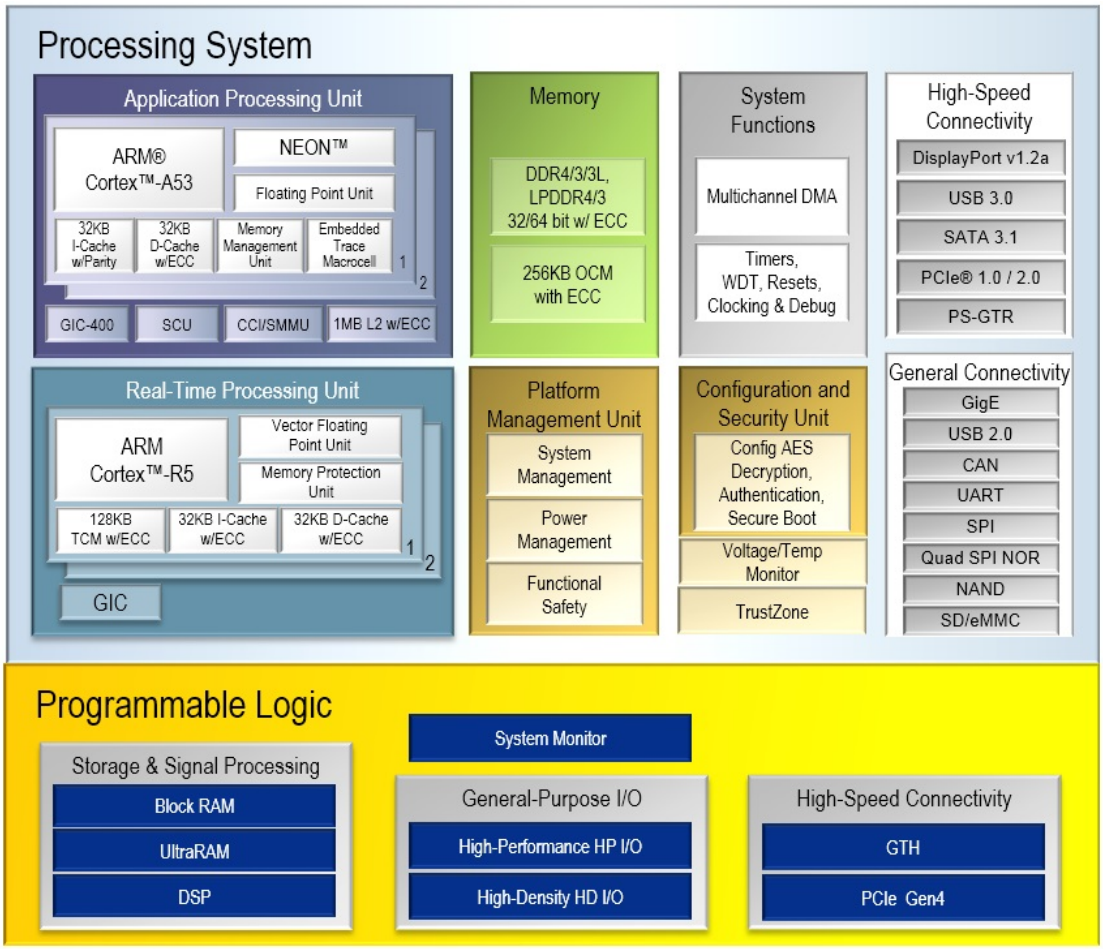


Figure 5.1: Zynq Ultrascale+ Architecture [11]

preamble detection with the 802.11a 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 different algorithms can be explored. While some SDR alternatives such as GNURadio and RFNoC are capable of supporting exploration of different algorithms on either HW or SW, our modeling environment allows manual decision of what blocks go on HW and SW. As part of my proposed research, I further explore optimized implementations of preamble detection.

Having prototyped on the Zynq SoC, I derive theoretical metrics on alternate SoC devices. For example, the Altera Arria 10® ARM-based SoC potentially offers performance improvement and

power reduction [72]. The Zynq UltraScale+ Multi-Processing SoC (MPSoC) architecture, illustrated in Fig. 5.1, which is designed for applications such as wireless, has both a Cortex-A application processor and a Cortex-R real-time processor that could improve the SDR's ability to adhere to specification times [11]. This MPSoC architecture is challenging to model because the presence of two processors introduces different types of HW-SW divide points. I investigate tradeoffs, using SWA to refer to the Cortex-A processor and SWR to refer to the Cortex-R processor, and labeling the data transfer points as HW-SWR, HW-SWA, and SWA-SWR.

As a separate topic of future work, I plan to develop a user interface that automates the generation of a transmit and receive chain. Given a user-specified list and ordering of processing blocks, the interface generates a model that incorporates all the specified blocks, handling all required data preparation and packing. In addition to implementing the processing block variants listed in Table 3.5 in both HW and SW, this automation would require adaptive packing of data for transfer using the AXI-Stream interconnect. Based upon the choice for the HW-SW divide points, different amounts of data must be sent between processing elements. The amount of data sent must still be packed into the size required by AXI-S, which is 32 bits. I automatically include the proper packing and unpacking routines in a transmit or receive chain, based on the number of bits required for a transfer.

Internally, communications on the Zynq chip uses an AXI interconnect, which can transfer 32-bit words in a time-synchronous manner between PL and PS. There are two AXI interfaces which I use: AXI-lite and AXI-stream. AXI-lite is a Memory Mapped (MM) protocol and AXI-stream is intended for high-speed streaming data. I use AXI-lite for the Receiver models and AXI-stream for the Transmitter models. To support the AXI-stream interface in the Transmitter models, the Vivado block diagram must contain the AXI Direct Memory Access (DMA) Controller, FIFO buffer, and various other IP cores as shown in Fig. 5.2. In a live, *online* scenario, to retrieve RF data bits from the FMComms3 Analog-to-Digital Converter (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 Digital-to-Analog Converter (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. For the purposes of data validation, I run our experiments in an *offline* mode, in which data intended for the RF front end is routed back to the Zynq PS for storage in a file. This method verifies that each model variant produces the same output and is functionally equal.

For each model variant, moving the HW-SW divide line changes the requirements for transferring data between PS and PL. The size and number of elements that must be transferred for

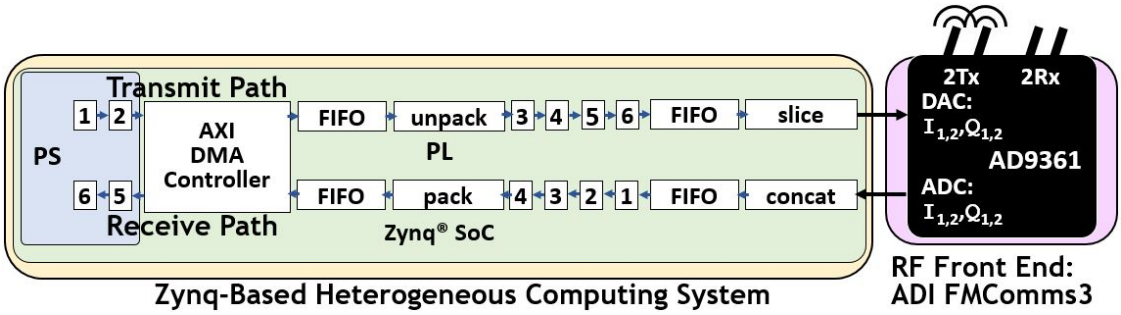


Figure 5.2: Modeling Hardware-Software Interface

Table 5.3: HW-SW Data Transfer for Tx

Variant	Data to Send	Data Type	Size of 1	#Elem
V1	Samples	Signed Fixed Point	16 bits	80
V2	Samples	Signed Fixed Point	16 bits	64
V3	Symbols	Signed Integer	1-8 bits	64
V4	Coded Bits	Boolean	1 bit	48
V5	Coded Bits	Boolean	1 bit	48
V6	Data Bits	Boolean	1 bit	24
V7	Data Bits	Boolean	1 bit	24

each model variant are listed in Table 5.3. Before sending any of the listed data types between PS and PL, I translate it to a 32-bit unsigned integer format for transfer using the AXI interconnect. I refer to this operation as *bundling*. In the case of transferring data bits or coded bits, the bits must be concatenated prior to transfer and separated after transfer. In the case of transferring complex samples, each sample must be split into its real and imaginary components and then concatenated prior to transfer. A summary of the data types, size, and number of elements required to perform the bundling operations for an 802.11a transceive chain using BPSK modulation and 1/2 coding rate are listed in Table 5.3. As part of this study, I extend this listed information to comprise the modulation schemes and FEC coding rates listed in Table 3.5. This enables the autogeneration of any potential transceive chain with every possible set of HW-SW divide points.

Bibliography

[1] *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.

[2] MathWorks, Inc. (2017) What is lte? [Online]. Available: http://www.mathworks.com/help/lte/gs/what-is-lte.html

[3] ——. (2017) Synchronization Signals (PSS and SSS). [Online]. Available: http://www.mathworks.com/help/lte/ug/synchronization-signals-pss-and-sss.html

[4] ——. (2017) Dl-sch processing functions. [Online]. Available: http://www.mathworks.com/help/lte/ref/dl-sch-processing-functions.html

[5] ——. (2017) Pdsch processing functions. [Online]. Available: http://www.mathworks.com/help/lte/ref/pdsch-processing-functions.html

[6] J. Wannstrom and K. Mallinson. (2017) Heterogeneous networks in lte. [Online]. Available: http://www.3gpp.org/technologies/keywords-acronyms/1576-hetnet

[7] National Telecommunications and Information Administration, US Department of Commerce. (2017) United states frequency allocation chart. [Online]. Available: https://www.ntia.doc.gov/page/2011/united-states-frequency-allocation-chart

[8] A. Schwarzinger, *Digital Signal Processing in Modern Communication Systems*, 1st ed. Lake Mary, Florida 32746: Andreas O. Schwarzinger, 2013.

[9] I. Analog Devices. (2015) Four Quick Steps to Production: Using Model-Based Design for Software-Defined Radio (Part 1). [Online]. Available: http://www.analog.com/en/analog-dialogue/articles/using-model-based-design-sdr-1.html

[10] MathWorks, Inc. (2017) comm.AGC System object. [Online]. Available: http://www.mathworks.com/help/comm/ref/comm.agc-class.html

[11] Xilinx, Inc. (2016) Zynq UltraScale+ All Programmable Heterogeneous MPSoC. [Online]. Available: http://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

[12] 3GPP, "Evolved Universal Terrestrial Radio Access (E-UTRA); Physical Channels and Modulation," 3rd Generation Partnership Project (3GPP), TS 36.211, Jan. 2017. [Online]. Available: http://www.3gpp.org

[13] 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.

[14] 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.

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

[16] 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.

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

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

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

[20] J. Malsbury and M. Ettus, "Simplifying fpga design with a novel network-on-chip architecture," in *Proceedings of the Second Workshop on Software Radio Implementation Forum*, ser. SRIF '13. New York, NY, USA: ACM, 2013, pp. 45–52. [Online]. Available: http://doi.acm.org/10.1145/2491246.2491251

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

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] B. Özgül, J. Langer, J. Noguera, and K. A. Vissers, "Software-programmable Digital Predistortion 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.

[35] 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.

[36] 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.

[37] B. A. Forouzan, *Data Communications and Networking*, 3rd ed. New York, NY, USA: McGraw-Hill, Inc., 2003.

[38] M. Nohrborg. (2017) Lte overview. [Online]. Available: http://www.3gpp.org/technologies/keywords-acronyms/98-lte

[39] P. S. Dahlman, E. and J. Skld, *4G LTE / LTE-Advanced for Mobile Broadband*. Kidlington, Oxford: Academic Press, 2011.

[40] MathWorks, Inc. (2017) Lte system toolbox. [Online]. Available: http://www.mathworks.com/help/lte/

[41] D. C. Chu, "Polyphase codes with good periodic correlation properties," *IEEE Trans. Inform. Theory*, vol. 18, 1972.

[42] I. Poole. (2017) Lte physical, logical and transport channels. [Online]. Available: http://www.radio-electronics.com/info/cellulartelecomms/lte-long-term-evolution/physical-logical-transport-channels.php

[43] C. Kosta, B. Hunt, A. U. Quddus, and R. Tafazolli, "On interference avoidance through inter-cell interference coordination (icic) based on ofdma mobile systems," *IEEE Communications Surveys Tutorials*, vol. 15, no. 3, pp. 973–995, Third 2013.

[44] M. Sauter. (2012) What's the Difference Between LTE ICIC and LTE-Advanced eICIC? [Online]. Available: http://mobilesociety.typepad.com/mobile_life/2012/03/whats-the-difference-between-lte-icic-and-lte-advanced-eicic.html

[45] M. M. Do and H. J. Son. (2014) Interference Coordination in LTE/LTE-A (1): Inter-Cell Interference Coordination (ICIC). [Online]. Available: http://www.netmanias.com/en/post/blog/6391/icic-interference-coordination-lte-lte-a/interference-coordination-in-lte-lte-a-1-inter-cell-interference-coordination-icic

[46] *3GPP TS 36.133. Annex A (normative): Test Cases*, 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA) Std. [Online]. Available: http://www.3gpp.org

[47] F. M. Abinader, E. P. L. Almeida, F. S. Chaves, A. M. Cavalcante, R. D. Vieira, R. C. D. Paiva, A. M. Sobrinho, S. Choudhury, E. Tuomaala, K. Doppler, and V. A. Sousa, "Enabling the coexistence of lte and wi-fi in unlicensed bands," *IEEE Communications Magazine*, vol. 52, no. 11, pp. 54–61, Nov 2014.

[48] T. Nihtilä, V. Tykhomyrov, O. Alanen, M. A. Uusitalo, A. Sorri, M. Moisio, S. Iraji, R. Ratasuk, and N. Mangalvedhe, "System performance of lte and ieee 802.11 coexisting on a shared frequency band," in *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, April 2013, pp. 1038–1043.

[49] E. Almeida, A. M. Cavalcante, R. C. D. Paiva, F. S. Chaves, F. M. Abinader, R. D. Vieira, S. Choudhury, E. Tuomaala, and K. Doppler, "Enabling lte/wifi coexistence by lte blank subframe allocation," in *2013 IEEE International Conference on Communications (ICC)*, June 2013, pp. 5083–5088.

[50] J. Jeon, H. Niu, Q. C. Li, A. Papathanassiou, and G. Wu, "Lte in the unlicensed spectrum: Evaluating coexistence mechanisms," in *2014 IEEE Globecom Workshops (GC Wkshps)*, Dec 2014, pp. 740–745.

[51] Z. Guan and T. Melodia, "U-LTE: Spectrally-Efficient and Fair Coexistence Between LTE and Wi-Fi in Unlicensed Bands," in *Proc. of IEEE Conference on Computer Communications (INFOCOM)*, San Francisco, CA, USA, April 2016.

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

[53] K. R. Chowdhury and T. Melodia, "Platforms and testbeds for experimental evaluation of cognitive ad hoc networks," *IEEE Communications Magazine*, vol. 48, no. 9, pp. 96–104, 2010. [Online]. Available: http://dx.doi.org/10.1109/MCOM.2010.5560593

[54] R. Subramanian, B. Drozdenko, E. Doyle, R. Ahmed, M. Leeser, and K. R. Chowdhury, "High-level system design of ieee 802.11b standard-compliant link layer for matlab-based sdr," *IEEE Access*, vol. 4, pp. 1494–1509, 2016.

[55] R. Subramanian, E. Doyle, B. Drozdenko, M. Leeser, and K. R. Chowdhury, "State-action based link layer design for ieee 802.11b compliant matlab-based sdr," in *2016 International Conference on Distributed Computing in Sensor Systems (DCOSS)*, May 2016, pp. 193–198.

[56] *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Higher-Speed Physical Layer Extension in the 2.4 GHz Band*, IEEE Std. 802.11b-1999, 1999.

[57] *Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std. 802.11, 1999.

[58] MathWorks, Inc. (2016) comm.AGC System object. [Online]. Available: http://www.mathworks.com/help/comm/ref/comm.agc-class.html

[59] B. Drozdenko, M. Zimmermann, T. Dao, K. Chowdhury, and M. Leeser, "Hardware-software codesign of wireless transceivers on zynq heterogeneous systems," *IEEE Transactions on Emerging Topics in Computing*, vol. PP, no. 99, pp. 1–1, 2017.

[60] ——, "Modeling Considerations for the Hardware-Software Co-design of Flexible Modern Wireless Transceivers," in *2016 26th International Conference on Field Programmable Logic and Applications (FPL)*, Aug 2016, pp. 1–4.

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

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

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

[64] Analog Devices, Inc. (2015) Introduction to boards based on the AD9361. [Online]. Available: https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms2-ebz/introduction

[65] C. Dick and F. Harris, "FPGA implementation of an OFDM PHY," in *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, vol. 1. IEEE, 2003, pp. 905–909.

[66] K. Chapman, "Saving costs with the srl16e," *Xilinx techXclusive*, 2000. [Online]. Available: https://www.xilinx.com/support/documentation/white_papers/wp271.pdf

[67] M. Luise and R. Reggiannini, "Carrier frequency recovery in all-digital modems for burst-mode transmissions," *IEEE Transactions on Communications*, vol. 43, no. 2/3/4, pp. 1169–1178, Feb 1995.

[68] *3GPP TS 36.141. Base Station (BS) conformance testing*, 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA) Std. [Online]. Available: http://www.3gpp.org

[69] *3GPP TS 36.104. Base Station (BS) radio transmission and reception*, 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA) Std. [Online]. Available: http://www.3gpp.org

[70] H. Zarrinkoub, *Understanding LTE with MATLAB: From Mathematical Modeling to Simulation and Prototyping*, 1st ed. 111 River Street, Hoboken, NJ 07030-5774: John Wiley and Sons, Inc., 2014.

[71] MathWorks, Inc. (2016) Communications System Toolbox. [Online]. Available: http://www.mathworks.com/help/comm/index.html

[72] Altera Corporation. (2016) Arria 10 fpga and soc. [Online]. Available: https://www.altera.com/products/fpga/arria-series/arria-10/overview.html

List of Acronyms

16QAM Order 16 Quadrature Amplitude Modulation. A type of QAM in which a set of four coded bits are translated to one of 16 possible complex symbols, each equally separated by 25% of the I and Q transmission amplitude range.

2G 2nd-Generation. A wireless telephone technology whose cellular telecom networks were commercially launched on the GSM standard.

3G 3rd-Generation. A mobile phone specification created by 3GPP, based on evolved GSM specifications.

3GPP 3rd Generation Partnership Project. A collaboration between groups of telecommunications associations, known as the Organizational Partners. The initial scope of 3GPP was to make a globally applicable 3G mobile phone system specification based on evolved GSM specifications.

4G 4th Generation. The variation of LTE technology currently in place, and the successor for 3G.

5G 5th Generation. The future generation of LTE technology, for which many research projects are experimenting and creating prototypes.

64QAM Order 64 Quadrature Amplitude Modulation A type of QAM in which a set of six coded bits are translated to one of 64 possible complex symbols, each equally separated by 12.5% of the I and Q transmission amplitude range.

ABS Almost Blank Subframe. In LTE eICIC, a subframe transmitted at lower power and only containing a fraction of the signals and channels in the resource grid.

ADC Analog to Digital Converter. A system that converts an analog signal into a digital signal, for our purposes implemented in a fixed ASIC.

ADI Analog Devices, Inc.. A company headquartered in Norwood, Massachusetts that makes wireless transceiver chips such as the AD9361 and RFFE boards such as the FMComms.

AGC Automatic Gain Control. A closed-loop feedback regulating circuit, the purpose of which is to provide a controlled signal amplitude at its output, despite variation of the amplitude in the input signal. The average or peak output signal level is used to dynamically adjust the input-to-output gain to a suitable value, enabling the circuit to work satisfactorily with a greater range of input signal levels.

AM Amplitude Modulation. A modulation technique used in electronic communication, most commonly for transmitting information via a radio carrier wave. In amplitude modulation, the amplitude (signal strength) of the carrier wave is varied in proportion to the waveform being transmitted.

AMBA Advanced Microcontroller Bus Architecture. An open-standard, on-chip interconnect specification for the connection and management of functional blocks in SoC designs.

APSoC All-Programmable System-on-Chip. A Xilinx, Inc. term used to describe the Zynq SoC, on which both the HW component (PL or FPGA) and SW component (PS or ARM CPU) are both reconfigurable devices.

ARM Advanced RISC Machine. A family of RISC architectures for computer processors, configured for various environments.

ASIC Application-Specific Integrated Circuit. An IC customized for a particular use, rather than intended for general-purpose use.

ASK Amplitude Shift Keying. A form of AM that represents digital data as variations in the amplitude of a carrier wave.

AXI Advanced eXtensible Interface. The third generation of AMBA interface defined in the AMBA 3 specification, is targeted at high performance, high clock frequency system designs and includes features that make it suitable for high speed sub-micrometer interconnect.

AXI-S Advanced eXtensible Interface - Streaming. A type of AXI interconnect in which data are not mapped to memory addresses, allowing for faster transmission speeds.

BPSK Binary Phase Shift Keying. The simplest form of PSK in which one coded bit is translated to one of two possible complex symbols, which are separated by a phase of 180°.

BRAM Block Random Access Memory. A dedicated two-port memory containing several kilobits of RAM, of which the PL on Xilinx FPGAs contains many blocks.

BTS Base Transceiver Station. A piece of equipment that facilitates wireless communication between UE and a network.

BW Bandwidth A subdivision of the electromagnetic spectrum reserved for wireless communications whose range is determined by some start frequency and end frequency. In 802.11, bandwidth is equal to the sampling frequency of the RFFE device.

C C Programming Language. The most frequently used language for programming CPUs, and the successor for the Basic Combined Programming Language (BCPL).

CDMA Code Division Multiple Access. A spread spectrum multiple access scheme that uses a predefined code, or spreading sequence, to spread the data bandwidth equally over the transmitted power.

CE Computing Element. A general term for an electronic system that can perform some user-defined processing instructions, such as a CPU or an FPGA.

CP Cyclic Prefix. In OFDM, this refers to a number of complex samples that are take from the end of a time-domain symbol and pre-pended to the beginning of the symbol to create a GI and reduce ISI.

CPU Central Processing Unit. In computer architecture, the computing element that is responsible for taking user commands in the form of user code and interacting with memory and I/O peripherals.

CRC Cyclic Redundancy Check. A type of FEC in which the redundancy value is produced using cyclic codes.

CSMA Carrier Sense Multiple Access. A MAC layer scheme for multiple access in which a node verifies the absence of other node traffic before starting a transmission.

CSMA/CA Carrier Sense Multiple Access with Collision Avoidance. A type of CSMA where if the transmission medium is sensed busy before transmission, then the transmission is deferred for a random interval.

CSMA/CD Carrier Sense Multiple Access with Collision Detection. A type of CSMA where the transmission is terminated as soon as a collision is detected.

DAC Digital to Analog Converter. A system that converts an analog signal into a digital signal, for our purposes implemented in a fixed ASIC.

DFT Discrete Fourier Transform. A mathematical technique in which a finite sequence of equally-spaced samples are transformed into an equivalent-length sequence of equally-spaced samples of the DTFT, a complex-valued function of frequency.

DL Downlink. In LTE communications, a subset of transmissions that originate at the eNodeB and terminate at the UE.

DL-SCH Downlink Shared Channel. In LTE, a PHY layer channel that contains the data bits to be transmitted.

DRX Designated Receiver. In the context of MAC layer experimentation, this refers to a wireless node whose main role is to receive a data packet from the DTX, but in doing so is required to have all the functionality of both a PHY layer TX chain and RX chain.

DSP Digital Signal Processor. A specialized IC on Xilinx PL that is specially designed for the implementation of fixed-point FIR filters; this is a major component of FPGA resources.

DSSS Direct Sequence Spread Spectrum. In wireless communications, a type of spectrum spreading to reduce signal interference used by the IEEE 802.11b standard for CDMA.

DTFT Discrete-Time Fourier Transform. A mathematical transform that inputs uniformly-spaced samples of complex numbers and produces a periodic function of a frequency variable.

DTX Designated Transmitter. In the context of MAC layer experimentation, this refers to a wireless node whose main role is to transmit a data packet to the DRX, but in doing so is required to have all the functionality of both a PHY layer TX chain and RX chain.

eICIC Enhanced Inter-Cell Interference Coordination. In LTE release 10, an enhanced ICIC method for heterogeneous networks.

eNodeB e-UTRA Node B The element in E-UTRA of LTE that is the evolution of the element Node B in UTRA of UMTS. It is the hardware that is connected to the mobile phone network that

communicates directly wirelessly with mobile handsets (UEs), like a base transceiver station (BTS) in GSM networks.

ESL Electronic Systems Level. An electronic design methodology focused on higher abstraction level concerns.

ETSI European Telecommunications Standards Institute. An independent, not-for-profit, standardization organization in the telecommunications industry (equipment makers and network operators) in Europe, headquartered in Sophia-Antipolis, France, with worldwide projection.

E-UTRA Evolved Universal Terrestrial Radio Access. The air interface of 3GPP's LTE upgrade path for mobile networks.

FDD Frequency Division Duplexing. A type of wireless communication system in which multiple frequency BWs are used for bidirectional communications, and TX and RX operate at different BWs.

FDM Frequency Division Multiplexing. A technique by which the total bandwidth available in a communication medium is divided into a series of non-overlapping frequency sub-bands, referred to in this dissertation as subcarriers, each of which is used to carry a separate signal.

FEC Forward Error Correction. A technique in digital communications systems in which additional redundancy bits are appended to a sequence of data bits as a means of checking the correctness of the transmission at the receiver.

FFT Fast Fourier Transform. An algorithm for performing a DFT on a computing element within a reasonable amount of time to convert a time-domain signal to a frequency-domain signal.

FIR Finite Impulse Response. In signal processing, a type of digital filter in which the impulse response has finite duration. Ideal for applications such as MF and rate transition.

FMC FPGA Mezzanine Card. A port on Xilinx development boards that allows for the fast transfer of data bits between a Xilinx FPGA or SoC and some peripheral I/O device, such as the ADI FMComms RFFE.

FMComms FMC Communications. A series of RFFE boards produced by ADI that easily connect to the FMC I/O port on Xilinx FPGA and SoC boards, thereby facilitating high-speed wireless sample processing.

FPGA Field Programmable Gate Array. An integrated circuit designed to be configured by a customer or a designer after manufacturing. The FPGA configuration is generally specified using an HDL, similar to that used for an ASIC.

GI Guard Interval. In OFDM-based wireless communications, this refers to the unused outer frequency subcarriers which are left blank to reduce ISI.

GNU GNU's Not Unix. An operating system and an extensive collection of computer software, composed wholly of free software, and responsible for the GNURadio SW modeling environment.

GSM Global System for Mobile Communications. A standard developed by ETSI to describe the protocols for 2G digital cellular networks used by mobile phones

HDL Hardware Description Language. A specialized computer language used to describe the structure and behavior of electronic or digital logic circuits, which can then be placed and routed to produce the set of masks used to create an IC or the bitstream that is used to program an FPGA. Examples of HDLs include VHDL and Verilog.

HFMF Hardware-Friendly Matched Filter. A method for decreasing the number of FPGA resources utilized for the MF algorithm.

HW Hardware. A type of CE that consists of fixed circuitry, which can either be static like an ASIC or reconfigurable like an FPGA.

IC Integrated Circuit. A set of electronic circuits on one small plate ("chip") of semiconductor material, normally silicon. This can be made much smaller than a discrete circuit made from independent electronic components.

ICIC Inter-Cell Interference Coordination. In LTE release 8, an optional technique for coordinating resource management between neighboring cells.

IEEE Institute of Electrical and Electronics Engineers. An organization responsible for coordinating research in electrical and computer engineering which, among other responsibilities, holds conferences, facilitates journal publications, and standardizes communications protocols.

IFFT Inverse Fast Fourier Transform. An algorithm for reversing the effects of the FFT. While equivalent in functionality to the FFT, the input and output are swapped, such that it takes in a frequency-domain signal and produces a time-domain signal.

I/O Input / Output. In computer architecture, the interfaces between the CPU and external peripherals.

ISI Inter-Symbol Interference. An artifact of early wireless communications systems in which the transmission power on one frequency BW would interfere with that of neighboring BWs; this is mitigated through the use of a GI.

ISM Industrial, Scientific, Medical. A designation of bandwidths within the US frequency spectrum that are available to be accessed by radios for any purpose. Used by IEEE standards such as 802.11 and 802.15.

ISO International Standards Organization. An organization responsible for standardizing networking information such as the OSI model.

JTAG Joint Task Action Group. An electronics industry association responsible developing a method of verifying designs and testing printed circuit boards after manufacture, and standardizes the protocol for programming Xilinx FPGAs.

LTE Long-Term Evolution. The international protocol for mobile wireless technology, standardized by 3GPP.

LUT Look-Up Table. A specialized IC on Xilinx PL that is specially designed for performing a 1-bit indexing operation; this is a major component of FPGA resources.

MAC Media Access Control. A sublayer of the datalink layer in the OSI networking model, responsible for providing channel access control mechanisms for multiple wireless nodes in a multiple access network.

MF Matched Filter. An algorithm used for pattern detection that uses an FIR filter with its coefficients set to the time-reversed complex conjugate form of an expected pattern, and effectively performs a cross-correlation with the expected pattern.

MIMO Multiple-Input, Multiple-Output. A wireless communications that can transmit and receive using multiple antennas at the same time, using techniques such as spatial diversity, spatial multiplexing, or beamforming.

MPSoC Multi-Processor System-on-Chip. Xilinx terminology for the next generation of APSoC, which consists of an FPGA and multiple CPUs, including an ARM Cortex-A application processor and an ARM Cortex-R real-time processor.

MRC Maximum Ratio Combining A method for phase and timing error and drift correction.

OFDM Orthogonal Frequency Division Multiplexing. A method of encoding digital data on multiple carrier frequencies. OFDM has developed into a popular scheme for wideband digital communication, used in applications such as digital television and audio broadcasting, DSL Internet access, wireless networks, powerline networks, and 4G mobile communications.

OFDMA Orthogonal Frequency Division Multiple Access. A multi-user version of the popular OFDM digital modulation scheme in which multiple access is achieved in OFDMA by assigning subsets of subcarriers to individual users.

OSI Open Systems Interconnect. A conceptual model that characterizes and standardizes the communication functions of a telecommunication or computing system without regard to their underlying internal structure and technology.

PDSCH Physical Downlink Shared Channel. A channel in LTE DL that holds the digitally modulated form of DL-SCH, which contains the user data to be transmitted.

PHY Physical layer. The lowest layer of the OSI networking model, responsible for defining the electrical and physical specifications of the data connection.

PL Programmable Logic. Xilinx terminology for the reconfigurable hardware component of an SoC, which contains DSPs, LUTs, and 1-bit registers.

PS Processing System. Xilinx terminology for the software component of an SoC, which contains a dual-core ARM CPU, memory, and I/O.

PSK Phase Shift Keying. A digital modulation scheme in which coded bits are translated to complex symbols for wireless transmission by rotating phase.

PSS Primary Synchronization Signal. In LTE, one of two synchronization signals which appears first in a subframe and is used in the cell search process.

QAM Quadrature Amplitude Modulation. A digital modulation scheme in the AM variety in which the constellation is arranged in a square grid with equal vertical and horizontal spacing.

QPSK Quadrature Phase Shift Keying. A type of PSK digital modulation in which two coded bits are translated to one of four possible complex symbols, each separated by a phase of 90 degrees.

RAM Random Access Memory. A type of computer data storage that allows data to be read from or written to in approximately the same amount of time regardless of the address location.

RB Resource Block A subdivision of the LTE resource grid containing 12 subcarriers.

RF Radio Frequency. A subset of the electromagnetic spectrum from 3 kHz to 300 GHz that is most frequently used for communications by wireless technology.

RFFE Radio Frequency Front End. A device that can communicate on RF bandwidths, which consists of both an RF wireless transceiver and an interface to a CE.

RISC Reduced Instruction Set Computing. A CPU design strategy based on the insight that a simplified instruction set provides higher performance when combined with a microprocessor architecture capable of executing those instructions using fewer microprocessor cycles per instruction.

RMC Reference Measurement Channel. In LTE, sample resource grids and waveforms that the standard provides for prototyping signals of different BWs and configurations.

RX Receiver. In a directional communication abstraction, this refers to the chain of processing blocks that converts complex fixed-point samples absorbed by an antenna on the RFFE to the set of data bits that are encoded within these samples.

SoC System-on-Chip. The general term for a heterogeneous computing system on a single fabricated chip that is comprised of more than one unlike CE, such as a CPU and an FPGA.

SSS Secondary Synchronization Signal. In LTE, one of two synchronization signals which appears first in a subframe and is used in the cell search process.

SW Software. In computer engineering, the portion of an electronic system that can be programmed with instructions written by a designer using a high-level programming language such as C. In the context of this dissertation, this usually refers to CPU instructions.

TDD Time Division Duplexing. A type of wireless communication system in which only a single frequency BW is used for bidirectional communications, and TX and RX operate at different timing intervals.

TX Transmitter. In a directional communication abstraction, this refers to the chain of processing blocks that converts data bits to complex fixed-point samples for electromagnetic radiation by an antenna on the RFFE.

UART Universal Asynchronous Receiver/Transmitter. A computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable. The electric signaling levels and methods (such as differential signaling, etc.) are handled by a driver circuit external to it.

UE User Equipment. In LTE, any device used directly by an end-user to communicate, such as a hand-held telephone, a laptop computer equipped with a mobile broadband adapter, or any other device. It connects to an eNodeB.

UL Uplink. In LTE communications, a subset of transmissions that originate at the UE and terminate at the eNodeB.

UMTS Universal Mobile Telecommunications System. A third generation mobile cellular system for networks based on the GSM standard, developed and maintained by the 3GPP.

UTRA UMTS Terrestrial Radio Access. An air interface used in UMTS.

UTRAN UMTS Terrestrial Radio Access Network. A collective term for the network and equipment used in UMTS that connects mobile handsets to the public telephone network or the Internet.

VHDL VHSIC Hardware Description Language. A HDL used in electronic design automation to describe digital and mixed-signal systems such as FPGAs and ICs.

VHSIC Very High Speed Integrated Circuit. A 1980s US government program whose mission was to research and develop very high speed ICs.

VLSI Very Large System Integration. A subset of ASICs containing as few as a hundred thousand to as many as one million tiny transistors.

Wi-Fi Wireless Fidelity. A technology for wireless communications between fixed-location computers that uses variants of the IEEE 802.11 standard.

WLAN Wide Local Area Network. A wireless network of devices that communicate using the Wi-Fi protocol, based on the IEEE 802.11 standard.