A Reconfigurable Random Access MAC Implementation for Software Defined Radio Platforms

Uchénna Kevin Anyanwu

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

Allen B. MacKenzie, Chair

Luiz A. Da Silva

Carl B. Dietrich

June 12, 2012

Blacksburg, Virginia

Keywords: Software Defined Radio, Wireless Networks, Random Access MAC

A Reconfigurable Random Access MAC Implementation

for Software Defined Radio Platforms

Uchénna Kevin Anyanwu

(ABSTRACT)

Wireless communications technology ranging from satellite communications to sensor networks has benefited from the development of flexible, SDR platforms. SDR is used for military applications in radio devices to reconfigure waveforms, frequency, and modulation schemes in both software and hardware to improve communication performance in harsh environments. In the commercial sector, SDRs are present in cellular infrastructure, where base stations can reconfigure operating parameters to meet specific cellular coverage goals. In response to these enhancements, industry leaders in cellular (such as Lucent, Nortel, and Motorola) have embraced the cost advantages of implementing SDRs in their cellular technology. In the future, there will be a need for more capable SDR platforms on inexpensive hardware that are able to balance work loads between several computational processing elements while minimizing power cost to accomplish multiple goals.

This thesis will present the development of a random access MAC protocol for the IRIS platform. An assessment of different SDR hardware and software platforms is conducted. From this assessment, we present several SDR technology requirements for networking research and discuss the impact of these requirements on future SDR platforms. As a consequence of these requirements, we choose the USRP family of SDR hardware and the IRIS software platform to develop our two random access MAC implementations: Aloha with Explicit ACK and Aloha with Implicit ACK. A point-to-point link was tested with our protocol and then this link was extended to a 3-hop (4 nodes) network. To improve our protocols' efficiency, we implemented carrier sensing on the FPGA of the USRP E100, an embedded SDR hardware platform. We also present simulations using OMNeT++ software to accompany our experimental data, and moreover, show how our protocol scales as more nodes are added to the network.

Acknowledgments

I would like to thank my advisers, Dr. Allen B. MacKenzie and Dr. Luiz DaSilva, for supporting me, giving me the opportunity to thrive, and believing in me throughout my graduate years at Virginia Tech. Without them, none of this would be possible.

I must also thank the folks at The Telecommunications Research Centre at Trinity College in Dublin, Ireland, for their support. A special thanks to Paolo di Francesco at Trinity College for his effort, encouragement, and being a "backbone" for this thesis, and specifically, taking photos of the lab equipments.

The research leading to these results has received funding from the European Unions Seventh Framework Programme (FP7/2007-2013) under grant agreement n. 258301 (CREW project). It was also partially funded by the Science Foundation Ireland under Grant No. 10/CE/I1853. This research was supported by a Bradley Fellowship from Virginia Tech's Bradley Department of Electrical and Computer Engineering and made possible by an endowment from the Harry Lynde Bradley Foundation.

Dedication

This is dedicated to my madre and hermanas for their moral support and encouragement. I would not be where I am today without them. May God bless their hearts and allow them to achieve anything they set their minds on. And, to my late padre, Dr. Kane C. Anyanwu, who taught me that I must question everything, work smart, set high goals, and enjoy life. May his soul rest in peace.

Contents

List of Abbreviations

ADC	Analog-to-Digital Converter
ARQ	Automatic Repeat Request
ASIC	Application Specific Integrated Chip
CCA	Clear Channel Assessment
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
CS	Carrier Sense
CSMA	Carrier Sense Multiple Access
DAC	Digital-to-Analog Converter
DDC	Digital Down Conversion
DSP	Digital Signal Processor

FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
GPP	General Purpose Processor
IF	Intermediate Frequency
IRIS	Implementing Radio in Software
KUAR	Kansas University Agile Radio
MAC	Medium Access Control
NIC	Network Interface Controller
OFDM	Orthogonal Frequency Division Multiplexing
OS	Operating System
OSSIE	Open Source SCA Implementation, Embedded
PHY	Physical Layer
RF	Radio Frequency
SCA	Software Communications Architecture
SDR	Software Defined Radio
SORA	Software Radio

TDMA	Time Division Multiple Access
TI	Texas Instruments
UHD	Universal Hardware Driver
USB	Universal Serial Bus
USRP	Universal Software Radio Peripheral
WARP	Wireless Open-Access Research Platform

List of Figures

List of Tables

Chapter 1

Introduction

The last two decades have seen an incredible proliferation of wireless devices around the world [1]. As a result of this technological proliferation, service providers developed different wireless standards, and, as a consequence, customers have to buy different devices to connect to different wireless standards.

The architecture of these wireless devices was implemented in such a way that MAC to PHY of the network protocol stack were implemented in hardware, meaning that once the communication algorithms were programmed on the chip, there was nothing that could be done to reprogam those algorithms. This traditional way of implementing wireless devices is inflexible, restricts upgradability and reusability. Soon, researchers took the hardware-oriented radio design to a different direction by porting communication blocks from hardware to software for more flexibility; this new architecture was called software-defined radio (SDR). An ideal SDR implementation would convert an RF signal

to the digital domain as soon as possible; in other words, the antenna would be directly connected to the ADC and then the ADC is connected to the DSP. However, this cannot be realized since high frequencies used by the telecommunications industry cannot be processed by cheap-of-the-shelf DSPs.

A SDR is a radio that accommodates a significant range of RF bands and air interface modes through software [2]. This means the SDR hardware is capable of performing different functions depending on the current state of the software. Since the software is run on a general purpose or embedded processor, controlled by an operating system with a scheduler, SDR functions will experience delays. The interface to route samples from the radio front-end to the processor will also include a significant delay in processing of the waveforms, rendering the SDR system incapable of meeting customer requirements. In result of these shortcomings, a significant amount of research has been invested in designing and implementing SDR packages that allows developers to implement efficient communication radios, and, if implemented properly, delays in the software can be reduced. Examples of SDR package are GNU Radio [3], and Implementing Radio in Software (IRIS) [4]. These SDR packages provide basic communication blocks to create a simple transceiver in software, using available modulator and demodulator components, like BPSK. Recently, medium access protocol (MAC) implementation on inexpensive SDRs has been a huge interest amongst SDR researchers. MAC protocols, such as 802.11a/b and Zigbee, have been implemented in general purpose processors (GPP) since it is easier to program in software, but yields slow performance. However, commercial

implementations such as the Atheros [5] on hardware outperforms ones implemented in pure software. Taking the hardware approach to MAC SDR is an arduous task, so judiciously implementing only time-critical functionalities such as back-off or carrier-sense for MAC protocols in FPGA, which is usually the closest programmable component to the antenna, for random access protocols, implemented in software, can yield enormous benefits.

1.1 Motivation

So far, SDR research has been largely focused on physical-layer communications, ignoring higher layer issues. In fact, MAC performance is an important design consideration in high-speed network applications. Traditionally, MAC functions are implemented in SDR software, where there are non-negligible delays [6], which results in longer than usual Rx-Tx turnaround time and carrier sense blind-spot [7], which is the time it takes to transmit a packet after the channel has been sensed idle. Both Rx-Tx turnaround time and carrier sense blind spot contribute to high retransmissions and low end-to-end throughput. For these reasons, implementing time-critical MAC functions closer to the RF front-end will decrease processing delays, yielding better performance for MAC protocols. One of the primary goals of this work is to implement a set of random-access MAC on an embedded SDR platform and test the performance of carrier sensing and non-carrier sensing MAC protocols in a multi-hop network.

1.2 Related Works

In the scope of SDR research, there have been several research endeavors to enable MAC protocols on SDR platforms. The research efforts have pushed SDR development closer to realizing easily programmable, scalable, yet powerful SDR platforms for research needs.

It is widely known that MAC implementations require strict timing, and, in order to meet those timing constraints, SDR researchers must implement time-critical functions closer to the RF front-end. Nychis, Hottelier, Yang, Seshan, and Steenkiste, at Carnegie Mellon University, pioneered enabling MAC protocols on GNU radio's SDR platform, exposing bus delays inherent in SDR architecture (i.e., user-to-operating system and operating system-to-FPGA delays) and then proposed the split functionality approach to improve network performance for a CSMA and TDMA-like protocols [6]. One of their objectives was to provide a basic toolbox for SDR researchers to easily implement MAC protocols. Some of the functionalities that were implemented are carrier sense, back-off, packet recognition, fine-grained radio control. Split-functionality improved network performance of their CSMA and Bluetooth-like TDMA implementations. Future work will include implementing various MAC implementations utilizing their split-functionality.

Murphy and Sabharwal, at The University of Texas at Austin, employed a hardware-centric approach to decrease delays on MAC protocol implementations[8]. They implemented a set of fundamental MAC functions and modulations schemes in FPGA that can achieve low latency and high network throughput comparable to ASIC-based 802.11 im-

plementations, and also can be used to build a variety of wireless systems. The Physical layer implementation developed in this platform is OFDM. The OFDM can be used for 802.11a/b/g/n implementations; however, it can also be used for advanced MIMO applications, where several antennas are used to receive RF frequencies. MAC implementations are executed in the on-board PowerPC that is embedded in the FPGA. Since the PowerPC is embedded in the FPGA, and not an external device, MAC implementations in the embedded processors will run faster in the FPGA than if the embedded processor was connected to the FPGA externally. MAC protocols that have been developed and tested: ALOHA, basic Carrier Sense Multiple Access (CSMA), and Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). Additionally, a mesh network of WARP nodes is a more recent accomplishment.

Jow, Schurgers, and Palmer , at the University of San Diego, developed a flexible SDR platform for testing 802.11-like protocols, called CalRadio [9]. The platform consists of two parts: the main board and the RF front-end. The main board has a ARM7 core, which runs MAC implementation, and a 5471 DSP core, which runs the master controller of the MAC/DSP. The advantage of the CalRadio design is that it is a software MAC, which means the time of development is low. The programming language used to create new MAC implementation in the DSP is ANSI C. Several advantages of using the CalRadio platform are: new MAC formats, inter-frame spacing control in DSP, programmable CCA thresholds, and more. Several MAC implementations, such as CSMA/CA, have been successfully tested on this platform setting. The need to improve the performance of

CaliRadio in outdoor conditions is an on-going research problem.

1.3 Thesis Organization

This thesis is organized in the following chapters:

- Chapter 2 surveys a variety of SDR architectures for networking and MAC protocol development. It enumerates the advantages and disadvantages of several SDR architectures and proposes requirements for high-performance SDR architectures for networking research.
- Chapter 3 presents the IRIS platform and proposes the Aloha protocol for the IRIS SDR platform. In this chapter, we describe the Aloha and Implicit ACK experimental test-bed and discuss its performance results including retransmissions and end-to-end throughput.
- Chapter 4 proposes an FPGA-based CSMA module to improve network performance. Enhancements to the signal processing protocol, called the VITA protocol, in the FPGA are discussed. This chapter also presents several experimental setups and provides an analysis of the performance of a CSMA multi-hop network.
- Chapter 5 presents the conclusion and directions for future work.

Chapter 2

Architectural Assessment for Embedded SDR Platforms

2.1 Motivation

Recently, there has been a need for inexpensive SDR solutions, resulting in three unique software radio architectures for networking research. These include GPP-based, Hardware-based, and Heterogeneous platforms, all of which have their advantages and disadvantages. The driving forces behind the proliferation of SDR architectures is the instantiated need for easily programmable, low-powered, inexpensive, and scalable SDR platforms. Additionally, the need to explore more sophisticated SDR applications, beyond traditional point-to-point applications, is becoming more important as SDR technology

become more ubiquitous. An area that has gained interest is MAC layer protocol design on SDRs. Although these implementations have been restricted to software, an alternative solution is to design time critical parts of the protocols closer to the front-end hardware. In this section, we will discuss a variety of software and hardware implementations, and suggest requirements and limitations that govern SDR solutions.

2.2 Competitive SDR Software Architectures

There are three widely used SDR software frameworks to support SDR research: GNU radio, IRIS, and OSSIE. Each software architecture has its own unique underlying architecture, yet provides sufficient capabilities for SDR researchers to develop sophisticated software radios. These SDR softwares allow the user to either port already designed signal processing blocks, such a filter, signal scalar or any DSP function, or design their own, and then connect them according to the user specification. The benefits of using such frameworks are as follows: (1) development speed; (2) portability ; (3) standard library reuse . In the following sections, we will describe the uniqueness, advantages and disadvantages, of each of the SDR platforms that are available to SDR researchers.

2.2.1 GNU Radio

GNU Radio [3] is an open-source SDR software that is popular among novice and expert SDR developers. This open-source software has an extensive support-base and many de-

velopers world-wide to contribute its advancement. GNU Radio facilitates development of simple to complex blocks, such a modulator, in Python; however, the underlying signal processing primitives are written in C++ for performance reasons. GNU radio is designed to be completely component based, which means that a each signal processing primitive is a component and can be connected with other components to develop a full SDR application. In addition, it relinquishes the developer from all portability, multithreading, and interfacing issues.

GNU Radio is designed to be multithreaded, with one thread per signal processing block, to take advantage of multicore processors. The multithreaded nature, combined with the low overhead processing requirements, allow flow-graphs to be composed of a greater number of fundamental processing blocks (adders, multipliers, filters, correlaters) than platforms with greater overhead costs (e.g. an SCA implementation with connecting middleware) in which having many small digital signal processing components would be detrimental to processing throughput. The increased granularity of GNU Radio's flow-graphs is an advantage when constructing and testing modulation or medium access schemes (among other things) as it allows easy reconfiguration of the flow-graph as well as the use of the many pre-made DSP blocks that come with GNU Radio. While GNU Radio is portable among operating systems and architectures, there is no built-in interface to make use of distributed/heterogeneous processing elements (embedded DSP or FPGA) [10].

2.2.2 OSSIE

The Open Source SCA Implementation Embedded (OSSIE) [11] architecture provides a platform to ensure portability and configurability of software and hardware. The primary goal of this SDR platform is to educate the graduate engineer or researcher, who may have C++, on structured development of SDR applications. The Software Communications Architecture (SCA), developed by the Department of Defense, is at the core of OSSIE, OSSIE provides a toolbox for radio prototyping, development and testing of SDR components and waveforms [12].

OSSIE enables SDR developers to create a component, which is a signal processing block that can interface with or be deployed on different devices such as a sound card, GPP, networks, or an external host device. An application utilizing a feature of remote waveform processing allows the SDR developer to deploy several compute intensive signal processing blocks on a remote host, leaving only less intensive blocks on the host computer, to balance the workload-this attribute is unique to SCA. These components can also be connected with other components in a multiprocessor application using Common Object Resource Brother Architecture (CORBA) middlware to develop a full waveform that can be used in a communication system. A clear advantage of this component-based architecture enables flexibiliy, portability, and manageability[12][13].

2.2.3 IRIS

Implementing radio in software (IRIS) is a SDR package developed by researchers at Trinity College, Ireland[4]. Its primary goal is to enable SDR developers to build highly reconfigurable, platform independent, network nodes. In the latest release of IRIS,IRIS 2.0, there are two types of engines in the architecture: Process Network (PN) Engine and Stack Engine, together used to develop a full network stack.

The PN engine is similar to the physical layer of a radio, containing signal processing blocks such as modulation schemes, filters, encoders, etc. Data flow is unidirectional since the PN engine is supposed to mimic the flow of a radio. The Stack engine is similar to the network stack, containing the parity checks, deframers, and a medium access protocol. The data flow in the stack engine is bi-directional, since MAC protocols might require access to the same data, multiple times, before it is passed on to the PN engine.

2.3 SDR Hardware Architectures

We classify SDRs into three major types of architectures: GPP-based, Hardware-based, and Heterogeneous. These architectures support front-end signal processing such as RF processing, analog-digital conversion, interpolation, and decimation. Due to the performance issues in analog-digital conversion, the performance of the RF front-end and the DSP back-end processing must be robust. In the following section, we will provide a meaningful background of each SDR hardware architecture, and discuss their advantages

and disadvantages.

2.3.1 GPP-based Architectures

GPP-based architectures include the Universal Software Radio Peripheral (USRP) by Ettus Research, the Software Radio (SORA) by Microsoft. In GPP-based architectures, transmitter and receiver chains, as well as the MAC protocol, are implemented in the GPP. The front-end processing does only the conversion from RF to IF and finally to baseband, where samples are sent to the host which has either a DSP or GPP for further processing. A primary advantage of GPP-based architecture is code resusibility and quick development time on a familiar programming language, such as C++ or C, increasing productivity and curbing an otherwise steep learning curve. However, this architecture inherits long processing delays both in the transmit and receive chain, which is called Rx/Tx delay. Processing, queuing, and bus transfer can easily add up to delay up to hundreds of microseconds [7]. Unfortunately, this huge delay is unacceptable to carrier sensing MAC protocols which have to meet time constraints on the order of tens of microseconds.

2.3.2 Hardware-based Architectures

Hardware-based architectures include the Wireless Open-Access Research Platform (WARP) by Rice University, and the Kansas University Agile Radio (KUAR) platform. In this architecture most if not all of the signal processing is done in the FPGA, and the higher layer

networking, routing, and application protocol implementations on an external GPP or soft core processor in the FPGA.

Hardware-based radio attempts to reduce processing delays that are inherent in GPP-based architectures by doing most of the processing close to the front-end. Since signal processing tasks, such as filtering and mixing, can be performed more quickly in FPGA than on GPPs, the processing times in the receive and transmit chain can be significantly decreased. Another advantage is the increase in performance of FPGAs and DSPs for commonly used algorithms for signal processing. In [14] a 1024-point Cooley-Tookey FFT algorithm performed better on FPGA and DSP than on GPP, even though the GPP operated on a high clock frequency. FPGAs represent a good trade off between flexibility, performance and costs, and DSPs presents a high flexibility, low system fabrication cost and medium power consumption [14]. WARP developers have already implemented 802.11 MAC/PHY protocol successfully. Since all of the processing is done on FPGA, the WARP platform requires special programming knowledge that the researcher may or may not have. The advantage of this research endeavor is that the WARP board can reach good MAC performance.

A disadvantage with hardware-based architecture is the development time. FPGA and low-level DSP code development is tedious and takes more time than high-level GPP code development, and is platform specific, which ultimately resists code portability. This drawback has prompted researchers and companies (i.e., Xilinx) to develop APIs that will decrease development times on FPGA for SDR applications.

2.3.3 Heterogeneous Architectures

After deploying several GPP-based SDR platforms, Ettus Research took a different approach in their development of SDR platforms and developed the USRP E100, a heterogeneous architecture for SDRs. This architecture has a Xilinx Spartan 3 FPGA and an Texas Instrument embedded processor. The embedded processor is composed of a ARM GPP and a Texas Instrument C64X+DSP, both of which on the same die for fast data movement between processing cores for computational intensive applications. This platform is unique in that it achieves high bandwidth and low latency by bringing the GPP closer to the front-end and allowing the distribution of workload among various computational devices. Additionally, this platform has software tools that allow code development to be less painstaking than a pure hardware-based platform.

Although the concept of heterogeneous processing is not an old one, the application of heterogeneous processing to SDR platforms is still at its infancy and has yet to be embraced by the SDR community. A current issue is how to best balance processing tasks between multiple processors, and at the same time, minimize data passing latencies between those processors.

2.4 Requirements for Research-based SDR Platforms

In this section we will briefly describe several fundamental-technical and non-technical -requirements for networking research on software radio platforms:

- The hardware platform must be inexpensive. The relationship between cost of SDRs and performance is roughly proportional; the more expensive the hardware becomes, the more likely it is to have high performance signal processing capabilities. A key challenge for software radio manufactures is to develop an SDR hardware that is reasonably priced (i.e., around 1,000 dollars) and has a good performance. Since most research labs want buy many SDRs (25 or more) to develop a network of nodes, quickly escalating the cost to tens of thousands, which is within the budget of many SDR funded projects. A proponent of minimizing cost per performance is Ettus Research who develops the USRP SDR hardware platform. On the other hand, if the price per SDR hardware were ten times more (i.e., around 10,000 dollars), as it is for the WARP platform, then the total cost will be easily out of the budget for many research labs.

- The software should have the potential to be open-source. Researchers have benefited from sharing code with software developers all over the world through the use of open repositories. A good example of an open-source SDR software project is GNU Radio, which has hundreds of contributors, and many professionals and researchers from around the world. The reason for their success is that their code is maintained by an open community, formed by both researchers and professional in the software radio community.

- Support for flexible MAC layer development. The Medium Access Control (MAC) Layer, in software radios, is a huge interest for software radio researchers. Before the

development of software radios, MAC layer functions for wireless devices were implemented on ASICs, called Network Interface Controller (NICs), where some functions were fixed. Implementing MAC functions on a NIC allowed manufactures to achieve the needed networking performance. Software defined radios introduced the ability to reprogram the MAC Layer implementation dynamically using Field Programmable Gate Arrays (FPGA) or embedded processors. The ability to reprogram the MAC allows the researcher to implement and test various protocols, and moreover, develop multi-band, multi-protocol SDRs.

- Support for flexible baseband processing. SDR platforms can benefit from a flexible baseband processor, specifically one that has an embedded DSP and GPP on the same chip to carrier out intense signal processing algorithms. Development to balancing workloads between embedded GPPs and DSP is under way. [15].

- Support for portability, code resuability, and code integration are software concerns for SDR developers. SDR software platforms must be portable from one platform to the next, without huge amounts of redevelopment efforts; in other words, the SDR software must be cross-compiled so that it can benefit multiple groups of developers using many operating systems or embedded systems. Enabling software blocks for one SDR application to be ported to another SDR application is a powerful capability for SDR developers. This capability can only be achieved through generic interfaces in the form of pointers to generic buffers that are passed between SDR blocks. The benefit of this capability is that SDR developers spend little time think-

ing about the inner-workings of the SDR platforms and more time on developing their SDR systems.

2.5 Limitations of SDR Platforms

In this section, we describe the delays and limitations that are inherent in affordable off-the-shelf SDR equipment. We explain how these limitations hinder the performance of sensing-based MAC layer implementations.

SDR performance is based on several key delays - CCA performance, Rx-Tx switching time, and MAC processing delay. Speed of light propagation is not included in the delay since it is a non-negligible delay.

- In a carrier sense-based protocol, the transmitting node needs to sense the channel is idle as soon as possible before the channel is occupied by another transmitting node in the network. This carrier sense-based protocol depends on the performance of how fast samples are received, averaged and compared to a threshold value in software or hardware. In [16], carrier-sensing was done in software. One advantage to this is that it can be easily implemented using a high level programming language. The alternative, carrier-sensing in FPGA, can be faster because sensing is done closer to the front-end hardware[6]. The benefits of this implementation are less retransmissions and better channel utilization, both of which increases end-to-end throughput in a multihop network.

- Rx-Tx switching time is the time it takes for a radio node to switch from Rx mode to Tx mode[17]. This is a requirement for all wireless nodes, because transmission and reception cannot happen at the same time. 802.11 requires a wireless node to switch from Rx to Tx in 600ns. Because SDRs are implemented with cheaper devices and controlled by software, SDRs have higher Rx to Tx switching times, sometimes in the order of microseconds.

- MAC processing delay is the time it takes the packet at the front of the queue to gain access to the medium [18]. MAC implementations can experience a huge overhead in processing queued packets simply because MAC implementations are usually implemented in software, where latencies are usually high [6]. There are several strategies that have been used to overcome this issue : to have the MAC layer, processing packets at the begining of the queue at highest priority; implement some of the MAC layer functions (pre-buffering ACKs) as close to the front-end as possible; implementing efficient DATA and ACK techniques such as selective ACK or implicit ACK, in either software or hardware. Unfortunately, current SDRs implementations lack these simple strategies because the time invested in development is usually not proportional to the performance gains of the system.

2.6 Future challenges of SDR Platforms

Implementing MAC protocols on SDR platforms is a difficult task for the reasons explained earlier. Research is under way to improve MAC performance of these SDR platforms. Current SDRs lack efficient communication bandwidth between on-board hardware, FPGA resources, and MAC implementations that utilize the channel wisely. The current challenge is to design improved carrier-sensing implementations on embedded and traditional SDR platforms. For example, a strategy that has been employed to improve network performance of embedded SDR platforms for CSMA-based protocols is to implement energy detection closer to the front-end hardware and control when to transmit packets buffered in the FPGA. These techniques and others will decrease carrier sense blind spots and ultimately increase end-to-end throughput in a multi-hop scenario.

Chapter 3

A Multi-hop MAC Protocol for Software Defined Radios

3.1 Motivation

Carrier-sensing-based MACs have been popular in WLANs because no coordination is required for channel sharing. But achieving good carrier sensing performance requires that a node be able to switch quickly between sensing the channel (to detect that it is idle) and beginning transmission. Many software defined radios have been unable to achieve short enough switching time because the switching circuitry must instruct the board to setup the transmit path, transmit samples, deactivate the transmit path, and then do a similar process for the receive path when receiving. Since SDRs are developed with in-

expensive integrated circuits, switching times can be non-negligible in carrier sense performance. The major alternative to carrier sensing is to use some form of time division MAC scheme. These require substantially more coordination and have also been difficult to implement on SDRs because of their tight synchronization requirements that most inexpensive SDRs cannot adhere to without significant hardware changes. MAC layer researchers have depended on simulation results to prove not only scalability in multi-hop networks. In fact, it is attractive to couple simulation results with actual implementation as it proves that the MAC protocol works as expected, but also, proves that the MAC protocol can be deployed in a real-world setting. The goal of this chapter is to describe a research endeavor to understand the performance of inexpensive SDRs in a multi-hop setting.

3.2 Related Work

Supporting MAC development on SDR has been a challenge for wireless researchers. An attempt to use two different software frameworks, GNU Radio and Click [19] for PHY-MAC development on SDRs is described in [20]. First is the GNU Radio, which uses flow graphs to implement PHY layer transmit and receive paths. Three ways of combining Click and GNU Radio were presented: first, to develop both MAC and PHY layers with Click; second, to port GNU Radio blocks to Click and use these ported blocks to develop both MAC and PHY implementations; third, to use inter process communication to com-

bine Click and GNU Radio [3] to support MAC and PHY layers. The first two of these designs were easy to implement, but the third required in-depth experience with Click and GNU Radio. A TDMA MAC was developed using the second design approach, porting GNU radio blocks to Click. This protocol worked well, in a two-node setup; however, time slots needed to be long (e.g. at least a second). A possible solution to this issue is to improve clock synchronization. It follows that GNU Radio and Click can be used together for PHY-MAC development, but it is not an easy task as improvements in both the user-space and OS level are imperative for a high performance system.

Schmid, et al. reports that the interface between the SDR back-end software and the hardware front-end produces non-negligible delays in the transmit and receive chains, contributing to one of the many challenges MAC developers face [7]. Moreover, SDRs have huge latencies both in software and the bus between the FPGA and GPP. Since SDR baseband processing is done in GPPs, buffers that hold the baseband samples can contribute to additional latency in the transmit and receive paths. Buffer sizes must be adjusted to an optimal size, depending on the operating environment [21].[7] implemented 802.15.4 in GNU radio and tested end-to-end performance—throughput and round trip times (RTTs)—-were measured. The measured RTT for the their 802.15.4 implementation on the GNU Radiol was 25 ms, as compared to 8 ms for a radio chip implementation. Several solutions were proposed to mitigate latency issues: first, change the USB 2.0 interface to a faster interface such as Gigabit Ethernet or PCI-Express; second, use meta data to control when packets are sent; third, pre-compute samples ahead of time and store them

into a buffer in the front-end; and fourth, offload often used algorithms, such as filters, to the front-end hardware.

Another important research in the development of networking protocols on SDR platforms evaluates the performance of MAC protocols on inexpensive, off-the-shelf SDR platforms [6]. The platform available at the time was the USRP1. The USRP1 hardware had obvious limitations, including FPGA space, limited ADC/DAC bandwidth, and slow bus interface to the MAC in software. To address these limitations they implemented a split-functionality approach that enabled improved implementations of core MAC functions in hardware and software. The split-functionality approach suggested that time-critical MAC functions, like carrier-sensing, precision scheduling of packets, packet recognition, and partial back-off must be implemented close to the front-end hardware. These changes in the USRP1 hardware improved the performance of both TDMA and 802.11-like MAC protocols in GNR Radio.

3.3 MAC Protocols and Design Choices

SDR hardware and software limits the choice of MAC protocols that can be implemented. Although current multi-core GPPs are capable of running sophisticated state machines written in high-level programming languages, sending packets to the front-end hardware and on to the channel fast enough to meet timing constraints of tens of microseconds will require both software and hardware changes. In [22] and [16], popular MAC protocols

were described but only the least time-constrained and hardware intensive protocols were chosen for implementation due to hardware and software limitations. For these same reasons, we have chosen initially to implement and attempt to enhance the Aloha protocol because it imposes few timing constraints on the host processor. Our initial implementation platforms are the USRP1 and USRP N210 from Ettus Research [23], both of which are affordable SDR platforms. In this chapter we will discuss the implementation of explicit and implicit ACK techniques in IRIS 2.0 using the USRP1 and USPR N210. Then, we will describe our experimental setup and parameters for testing our implementations. Finally, we will evaluate the performance of both ACK strategies in a multi-hop scenario based on retransmissions, RTT, and end-to-end throughput.

3.4 Proposed Implementations

For the reasons discussed previously, we have implemented an Aloha-based MAC protocol with a stop-and-wait ARQ scheme. This protocol will resolve contention using random back-off, checks for a match in MAC layer address, checks each frame's CRC, and executes other basic MAC layer functions. Additionally, we have implemented a simple routing protocol that checks the packet's IP address of each packet. The following section describes the ACK strategies that were implemented in our Aloha protocol. In both designs, collisions are eminent as DATA or ACKs at the source can conflict with forwarded packets in the relay on the way to the destination node in a two hop network.

3.4.1 Explicit ACK Strategy

In our implementation of Aloha 3.1, the source node, node A, sends 500 byte packets

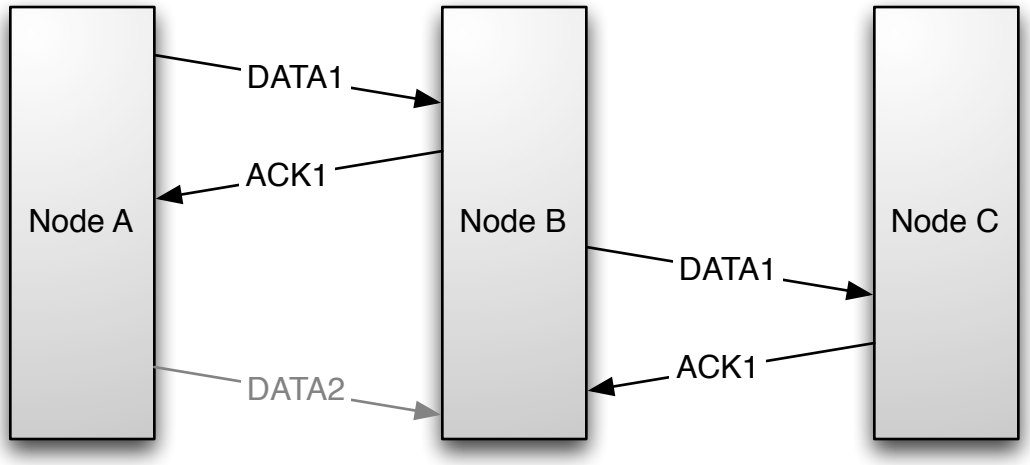

Figure 3.1: Depicts the flow of data packets from the source to the destination of the explicit ACK strategy in a two-hop network

to the relay node immediately upon arrival from the MAC layer. When the packet is received in the MAC layer of the relay node, it will check the received packet's MAC address and determine if it is the recipient. For duplicate packets at the relay node, it discards the packets and waits for another packet with the next sequence number. The relay node then forwards the packet to the destination node, and, if successfully received by the destination node, the destination node sends the packet up to the routing layer where it is checked once more and then finally sent to the file writer where the payload is written to a file. The results in the following section, show that adding a short delay before sending the next data packet at the source alleviates retransmissions and improves throughput by avoiding collisions that can occur from interfering nodes.

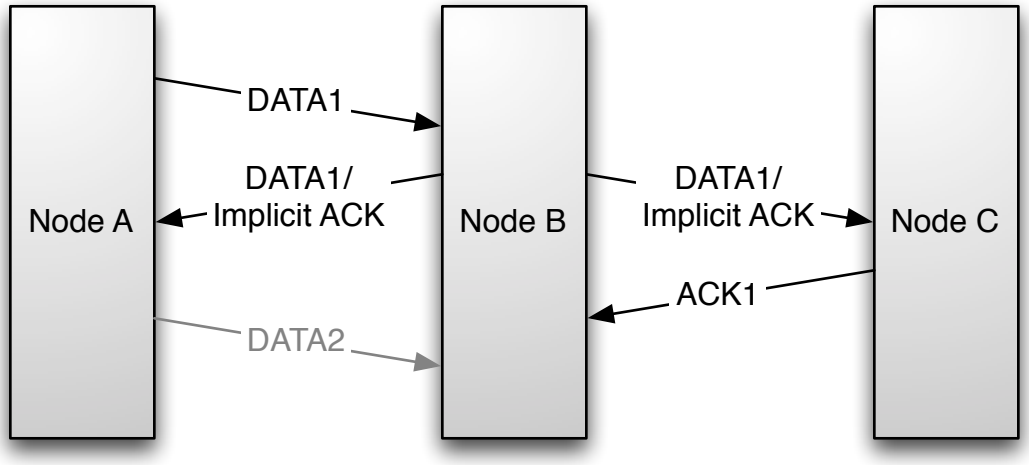

Figure 3.2: Depicts the flow of data packets from the source to the destination of the implicit ACK strategy in a two-hop network

3.4.2 Implicit ACK Strategy

The implicit ACK strategy 3.2 is similar to the explicit ACK strategy; however, there is one slight important difference: the packet forwarded by the relay node is treated as an ACK packet for the DATA packet sent by the source node. The destination node still sends the ACK to the relay node as it does not need to forward the DATA packet. We have implemented this enhancement to decrease collisions and increase end-to-end throughput.

3.5 Experimental Setup

Two types of experiments were conducted—Explicit ACK with delay and Implicit ACK. We conducted two sets of experiments: the first using three nodes on the USRP1, and the second with four nodes on the USRP N210. Configurations for the experiments are summarized in Table 3.1.

Packet size at MAC Layer	600 B
Center Frequency	5.00 GHz
Bandwidth	1 MHz
Modulation Scheme	OFDM
Active data subcarriers	192 of 256
Subcarrier Modulation	BPSK
ARQ timeout	50 ms

Table 3.1: PHY and MAC Parameters used in the experiment.

Laboratory setup for the three-hop(four node) network is depicted in Figure 3.3.

We have made sure each node can successfully transmit to its neighbors.

3.6 Results

In order to test the performance of the protocol, we sent 500 packets from source to destination. To understand the sequence of events that takes place over the air, we noted the time stamps of each transmission and reception from each node and produced the plot

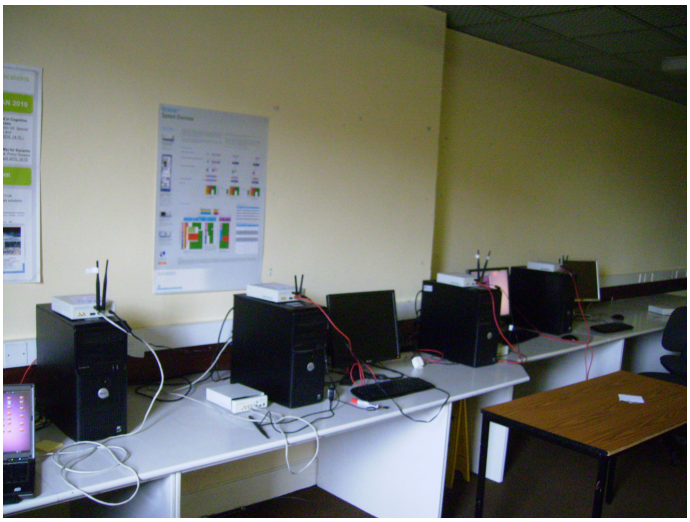

Figure 3.3: Laboratory setup for four N210 nodes.This photo was taken by Paolo di Francesco at Trinity College and used with permission.

shown in Figure 3.4. In the time plot, we have identified the collisions that occur in each link. The figure shows that the random latency associated with the USRP1 hardware and IRIS software contributes to the intra-flow interference from source to destination. According to the timeline , sometimes the DATA from Node A collides with the ACK from Node C because when the ACK is transmitted by Node C, Node A is ready to transmit as well. However, due to the randomness of the timing collisions do not happen all the time. To mitigate these collisions, we noticed that increasing the source node delays in the MAC layer decreased the chance of collisions at the relay, hence increasing end-to-end throughout. Each bar in the throughput chart shown in Figure 3.5 represents the average throughput over 50 trial runs for Explicit and Implicit ACK techniques.

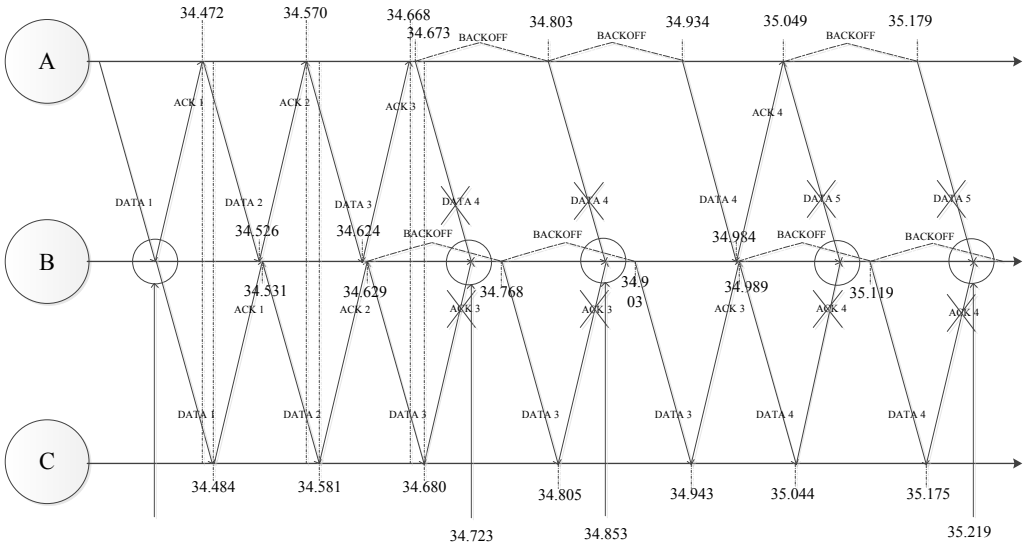

Figure 3.4: Timeline for three nodes for at least 1 ms.

3.6.1 Explicit and Implicit ACK Implementation Results

The optimal source node delay to be added in the MAC layer for the best throughput is 10 ms for Explicit ACK and 20 ms for the Implicit ACK. As we initially thought, the Implicit ACK provided the best performing MAC in terms of retransmission and throughput in USRP1. Similar experiments were conducted on the USRP N210 using four nodes (three hops). The USRP N210 results will be explored in the next section as we compare those results to simulation results. We developed a simulation model, with transmit and receive delays from IRIS, in OMNeT++ simulator to extend the number of nodes in our multi-hop network. The next section compares the simulation results to the implementations results to understand the scalability of our random access protocol for multi-hop scenarios.

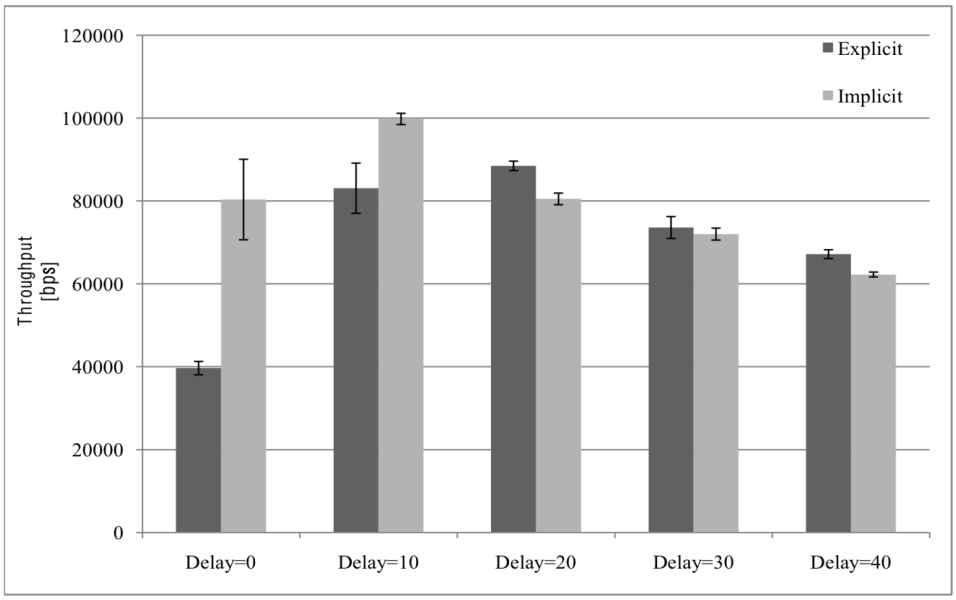

Figure 3.5: Throughput for Explicit and Implicit ACK for two-hop network using USPR1s

3.6.2 Comparing Simulation and Implementation Results

Testing our MAC implementation for many-hops is challenging because we are limited by the number of USRPs at our disposal. Fortunately, network simulators, like OMNeT++, provide a means to test a large number of nodes in a network and quickly evaluate the performance of the network under simple channel models [24]. In OMNeT++, each layer of a network stack is called a module. To replicate our MAC implementation in the OMNeT++ simulator, we measured the time delay between sending a DATA packet from the MAC layer to the time the MAC layer receives the ACK packet (i.e. RTT). We used this information to model each node in OMNeT++. The host latency was included as a delay in the MAC module, and the USB/Ethernet and front-end hardware was included in the PHY module. Since jitter [6] in USB/Ethernet and front-end hardware, respectively,

are caused by random attributes of physical devices, we modeled the USB/Ethernet and front-end hardware delay with a random Gaussian distribution, with mean and standard deviation calculated from RTT acquired from several over-the-air tests between two nodes. The output distribution provided a range of values that were similar to ones seen during over-the-air tests. Also, we tested two hops and verified that our simulation throughput matched with our experimental throughputs. Then, we extended the network to three, four, and five hops. In our simulation-based tests we noticed that extending our network beyond three hops will decrease end-to-end throughput. Figures 3.6 and **??** show a summary of our tests, comparing performance of simulation and experimentation on USPR N210. Similar to the USRP1 tests, adding 10 ms delay improved implicit ACK and explicit ACK for three hop network of USRP N210s; however, delays of more than 10 ms degraded performance. We also achieved higher throughput, but this improvement is a result of faster hardware. Comparing implementation to simulation, demonstrates that it is possible to accurately model the delays that impact MAC performance in a network simulator, such as OMNeT++.

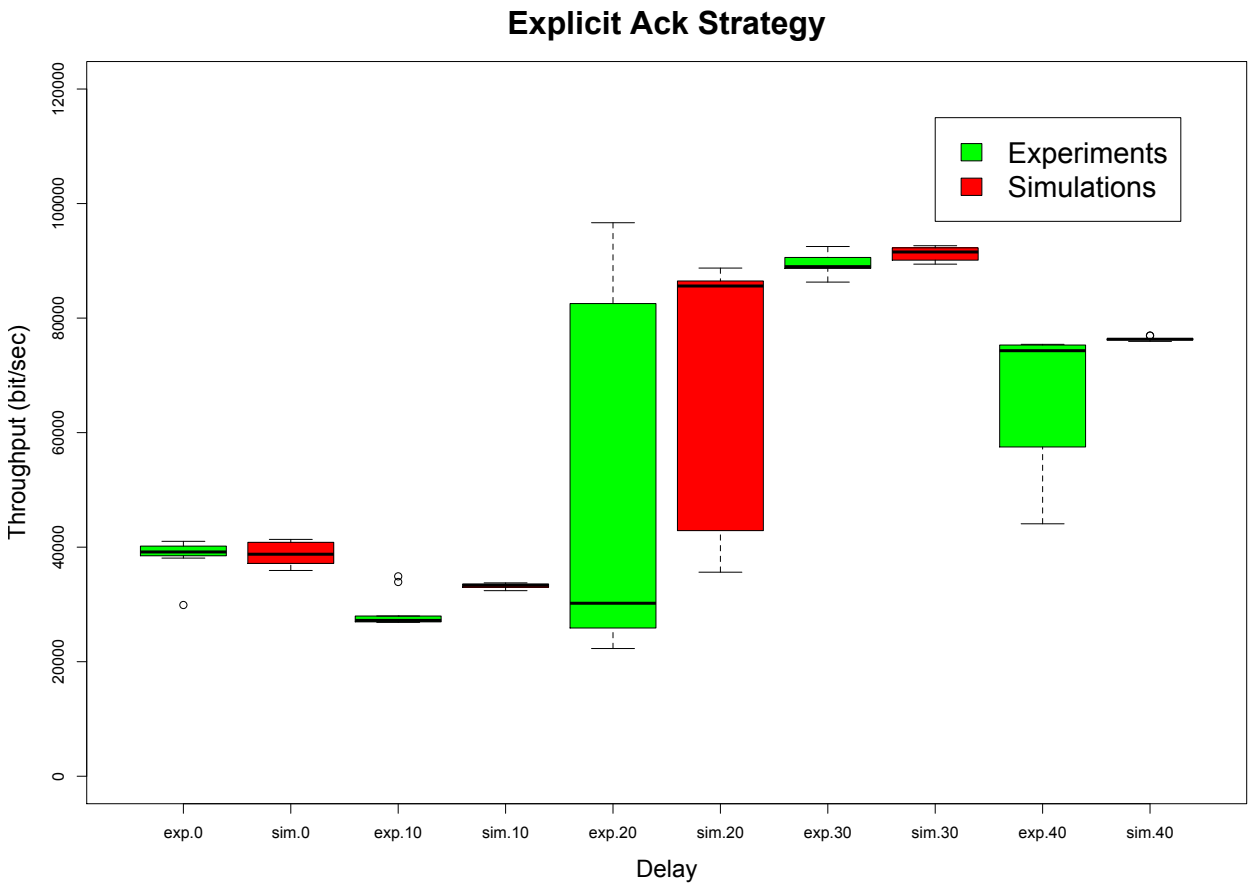

Figure 3.6: Throughput for Explicit and Implicit ACK for two-hop network using USPR N210s

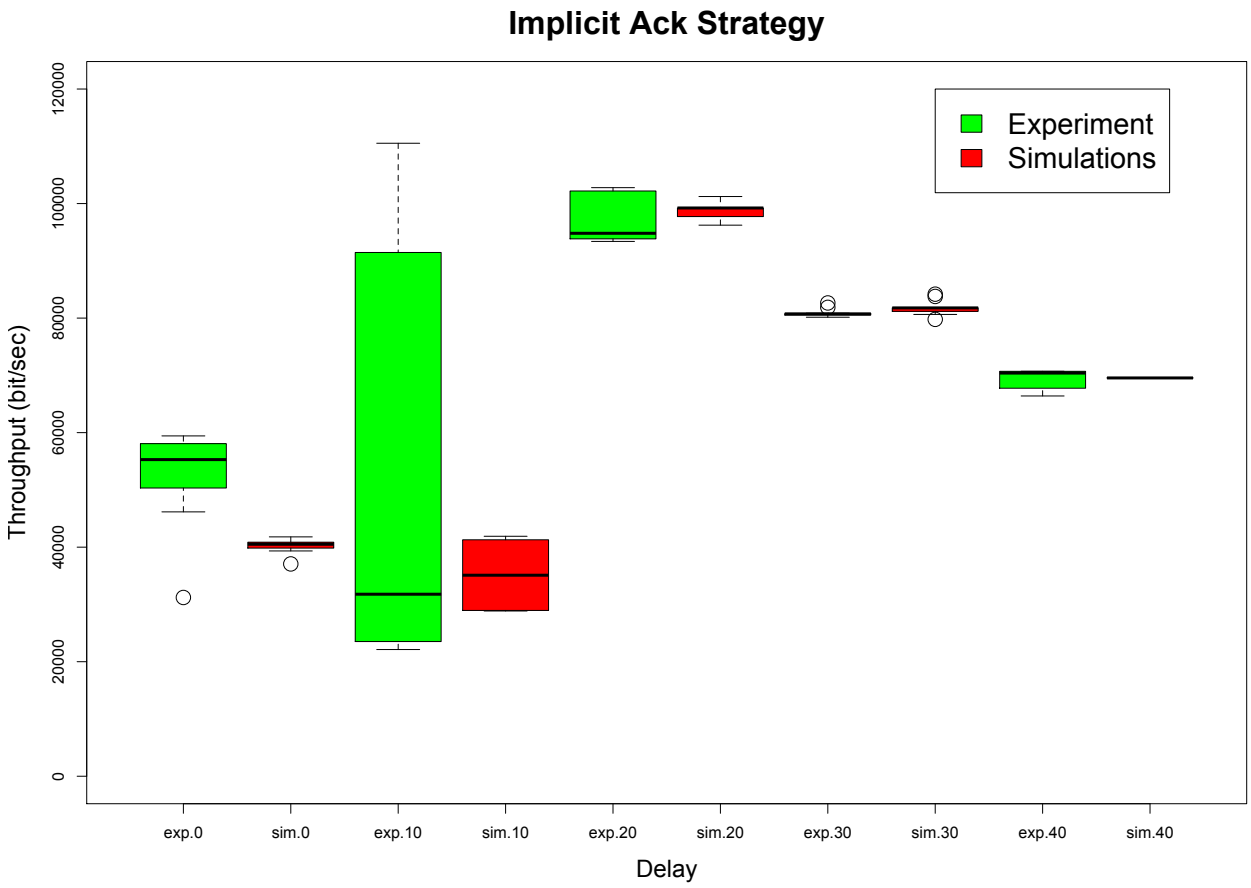

Figure 3.7: Throughput for Explicit and Implicit ACK for two-hop network using USRP N210s

3.7 Limitations and Possible Solutions

Our experiments exposed a limitations in SDR MAC development. Performance (throughput) degrades after three hops using Implicit ACK. Many-hop implementations are hindered by collisions within the transmission range of each node, so the more relay nodes we add, the poorer the performance. To reduce the collisions, we must employ a sensing-based MAC protocol for each node. With carrier sense each node senses the medium before it transmits, giving the node the opportunity to avoid causing collisions on the channel. Given these advancements to mitigate poor performance in SDRs, a reasonable solution is to utilize an already proposed approach: the split-functionality approach where time-critical function are in hardware while non-time critical MAC functions are in software to improve Rx-Tx transmissions, improving the network performance for inexpensive SDRs. In the next chapter we will implement carrier sense in an embedded platform, with performance constraints than the USRP N210, utilizing the split-functionality approach.

Chapter 4

A Reconfigurable Random Access MAC for SDR Platforms

4.1 Introduction

Designing MAC protocols is challenging due to the large Rx-Tx turnaround time, buffering mechanisms, and latencies associated with inexpensive SDRs. For example, the Rx-Tx turnaround time affects how fast a frame is transmitted after detecting the channel is idle. For these reasons, most SDR developers have focused on physical layer issues and neglected implementation of MAC layer protocols for emerging SDR platforms.

To overcome these issues, [7] quantified the delays in both the transmit and the receive chain of the USRP SDR and suggested hardware and software changes to minimize these

delays. [6] expanded on the previous work and proposed the split-functionality approach where time-critical MAC functions are judiciously implemented either in software or hardware. For example, back-off functionality and packet detection were implemented in hardware, close to the front-end, to reduce the Rx-Tx turnaround time for SDRs with high latency.

This chapters extends our previous work [22] from a non-embedded platform, the USRP N210, to an embedded platform, the USRP E100. More specifically, this chapter describes the following contributions:

1. We develop carrier sense block, which is SDR software platform agnostic, in FPGA
2. Utilizing the split functionality approach, where some MAC functions are in FPGA and others are in the SDR software.
3. We demonstrate the superior performance of our solution over multi-hops through series of experiments which showed that the FPGA-based carrier sense implementation increases throughput and decreases retransmission for random access protocols.

This chapter is structured as follows. First, we discuss the random access MAC protocols that will be implemented on the USRP E100. Second, we discuss the reason why we chose the USRP E100, describing its hardware advantages, disadvantages, and specifications. Then we discuss the performance of four protocols in a two-hop network of USRP E100s: Aloha with Explicit ACK, CSMA with Explicit ACK, Aloha with Implicit ACK,

and CSMA with Implicit ACK. Lastly, we conclude and mention future directions for this research.

4.2 Random Access MAC Protocols for SDR Platforms

In our previous work, we implemented two random access MAC protocols, Aloha with Explicit ACK and Aloha with Implicit ACK. Figure 4.1 depicts the DATA and ACK ex-

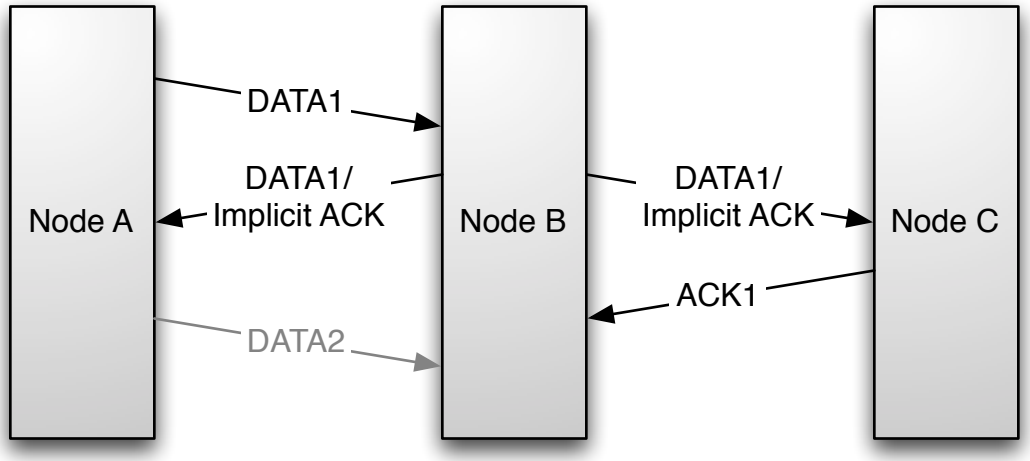

Figure 4.1: DATA and ACK exchange between nodes using Implicit ACK

change between nodes using Implicit ACK technique in a two-hop network.

Since the Aloha protocol has a theoretical channel capacity (efficiency) of 18 percent, which limits how much packets we can send in a mult-hop network, we have decided to implement CSMA, which can be at least 90 percent efficient, improving the potential number of packets that can be sent in our multi-hop network. However, we can expect performance degradations to occur as the number of hops increase.

Puschman et al., proposed that implementing carrier sensing in software is a reasonable choice because of its greater flexibility than the hardware version [25]. However, we believe that implementing carrier sense in flexible hardware, such as an FPGA, close to the front-end, will have a greater impact on performance than a software-based solution. The following section describes the implementation of the carrier sense design in FPGA.

4.3 SDR Hardware choice and specifications

The USRP E100 has a 720 MHz TI OMAP embedded ARM cortex A8 processor which runs a full embedded distribution of Linux (Angstrom). This enables development and deployment of software radio systems without the need for an external host computer. In addition to the ARM processor, the E100 contains a Xilinx Spartan 3A 1800 DSP FPGA and a TI C64+DSP processor. In this work we have not utilized the DSP.

An important feature of USRP E100 is that the embedded processor is interfaced to the Xilinx FPGA with a direct memory interface. This allows fast access to baseband samples streamed from the FPGA and increased opportunities to control the data stream on the FPGA.

It is clear from Figure 4.2 that the radio front-end is connected directly to the FPGA and that all data for transmission by or received from the radio front-end must pass through the FPGA.

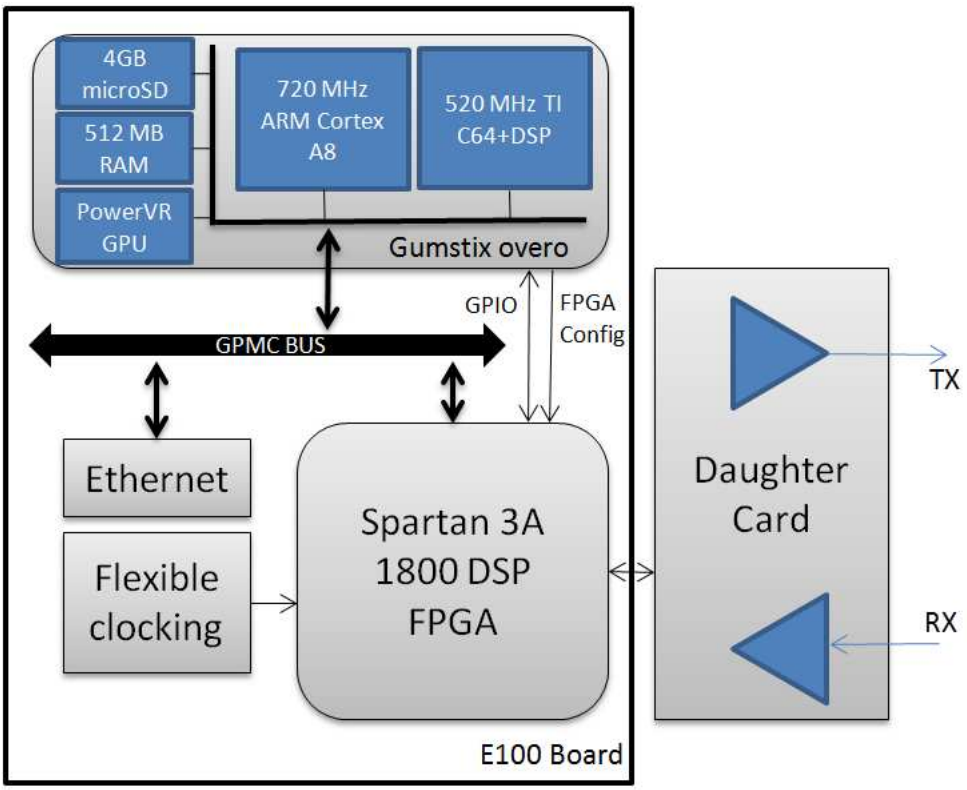

Figure 4.2: A Block diagram of the E100 USRP.

The FPGA is therefore ideally placed to implement latency-sensitive processes like sensing if a channel is occupied and controlling transmissions to avoid collisions. To implement this system it is necessary to integrate customized FPGA blocks into the existing FPGA software design. Currently the FPGA on the E100 contains the Rx and Tx data chains for the radio front-end as well as communication/debugging modules to aid in software radio design, see Figure 4.3. The USRP E100's SDR hardware is composed of two main hardware devices: an FPGA and RF front-end. The FPGA provides sample processing of I and Q samples in the IF domain, and the RF front-end enables up-conversion and down-conversion of transmitted and received RF signals.

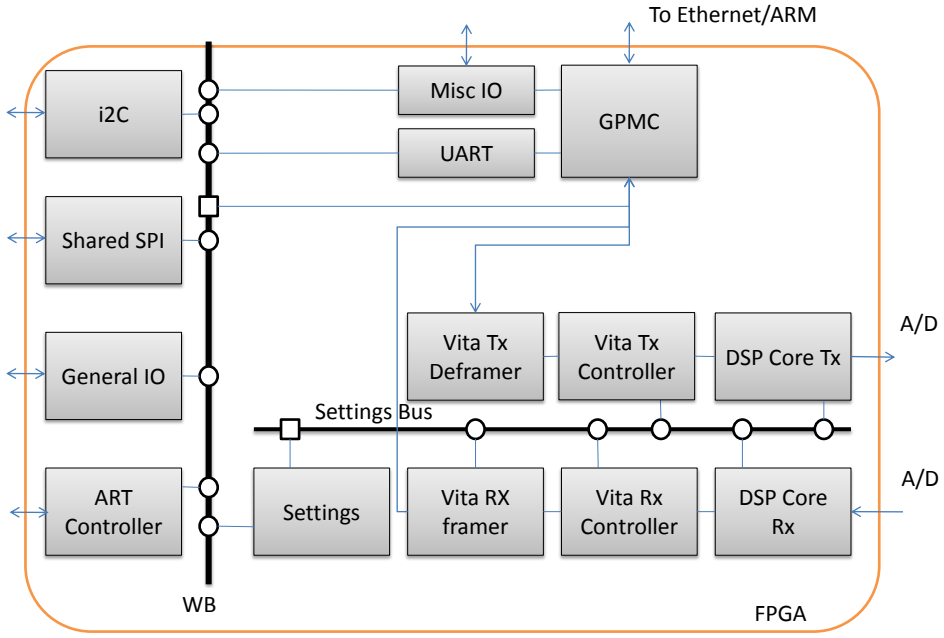

Figure 4.3: An overview of the hardware currently on the USRP E100 FPGA.

The FPGA uses the VITA radio transport (VRT) protocol to communicate with the ARM processor. VITA was developed to provide interoperability between diverse SDR components by defining a transport protocol to convey digitized signal data and receiver settings [26]. The protocol provides an IF data frame that contains a header followed by the data payload. These frames provide status information such as RF and IF frequencies, bandwidth, internal delays, sampling rates, A/D overflow, and user-defined parameters. Additionally, VITA provides buffers for in-coming and out-going frames from the SDR. The up-conversion and down-conversion of the baseband signals are carried out in the DSP CORE Tx and DSP CORE Rx blocks respectively in Figure 4.3. The Tx controller chooses when to send the data in the Tx buffer based on the rules received in the VITA

frame, e.g. time to send. The controller is also responsible for flagging underflow errors in the Tx chain and notifying the host when a burst is sent. Likewise the Rx controller decides when to send received data to the host, timestamps data received, and flags overflow errors in the stream. The framing and de-framing modules implement the framing policy of the VITA protocol.

To receive data before it is sent to the host and to control the Tx streams access to the channel—both of which are required by an FPGA carrier sense implementation—our carrier sense hardware needed to be tightly coupled with the VITA modules in the FPGA. Therefore it was necessary to tap into the inphase and quadrature (I/Q) samples at the VITA control block in the receive path, see Figure 4.4.

Moreover, the rate at which these samples are produced at this point in the chain is equal to the rate requested by the user, i.e down-conversion has already taken place, which is useful for debugging receiver issues. The I/Q samples are sent to the carrier sense block, which in turn produces a signal indicating whether a carrier is present or not, which is used by the modified VITA Tx Controller block in deciding when to transmit.

4.4 Carrier-sense Implementation

Our implementation is unique because it is SDR agnostic and allows for reconfigurability in the performance of the carrier-sensing in flexible hardware. The first section presents the implementation of the carrier sense module in the Xilinx FPGA. We describe the state

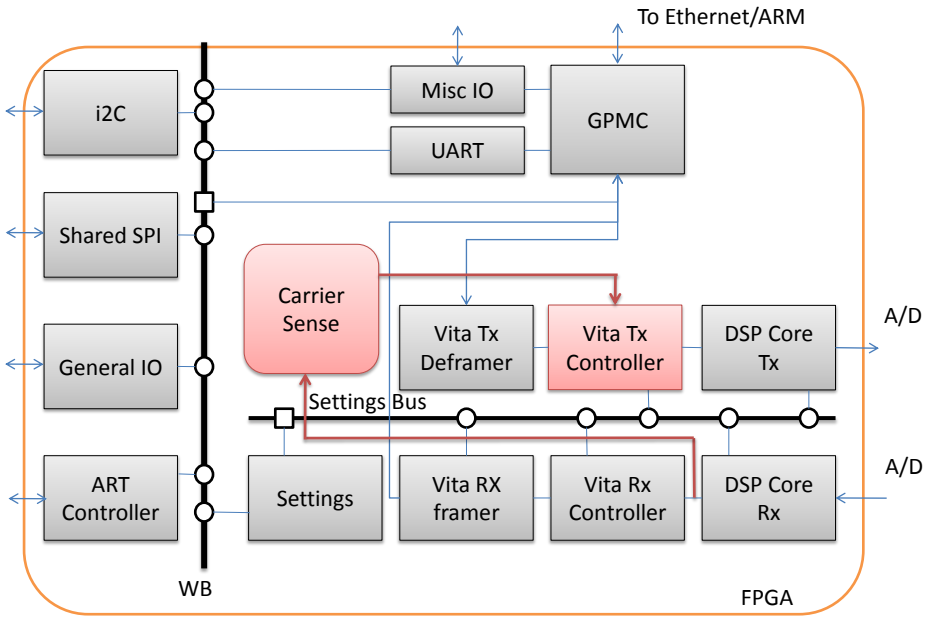

Figure 4.4: The integration of the carrier sense block into the FPGA architecture on the E100 FPGA.

machine and data paths that control the transmit chain of the VITA signal processing chain. In section 4.4.2 we describe the architectural changes that were made to IRIS to enable it to know when a packet has left the buffer.

4.4.1 FPGA Hardware Implementation

A block diagram of the carrier sense module as implemented on the USRP E100 FPGA is shown in Figure 4.5.

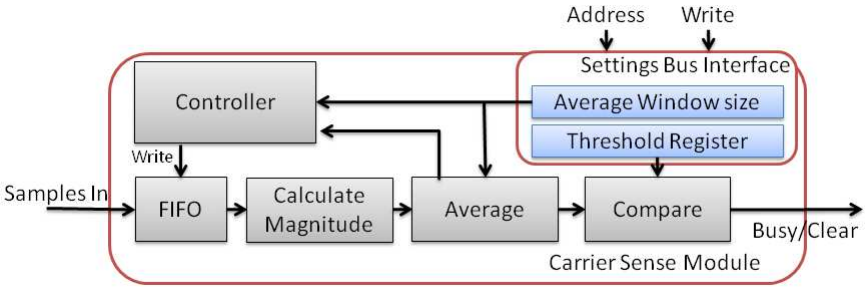

Figure 4.5: Block Diagram of the carrier sense module on USRP E100

Carrier Sense module

Data is collected from the receive chain and placed in a FIFO buffer. The magnitude of each I/Q value is then calculated and a sliding window average is taken over 64 baseband samples. The number of baseband samples can be increased or decreased by powers of two; however, increasing the number of baseband samples that are averaged increases carrier sense delay, though increasing the accuracy of the signal power estimate. Finally the average of the sample magnitude is compared with a programmable threshold value and the Busy/Clear signal is set accordingly.

The data path is controlled by a state machine with three states (Figure 4.6). At system start-up, the FPGA initializes the carrier sense (CS) controller to the Initial state. In the CS controller, the state machine waits for the strobe signal, which is activated when there is a new, valid sample at the output of the ADC, to be asserted. The CS controller takes this sample and writes it into a FIFO buffer and then transitions to the Compute state.

The CS controller remains in the Compute state until it receives N baseband samples (in

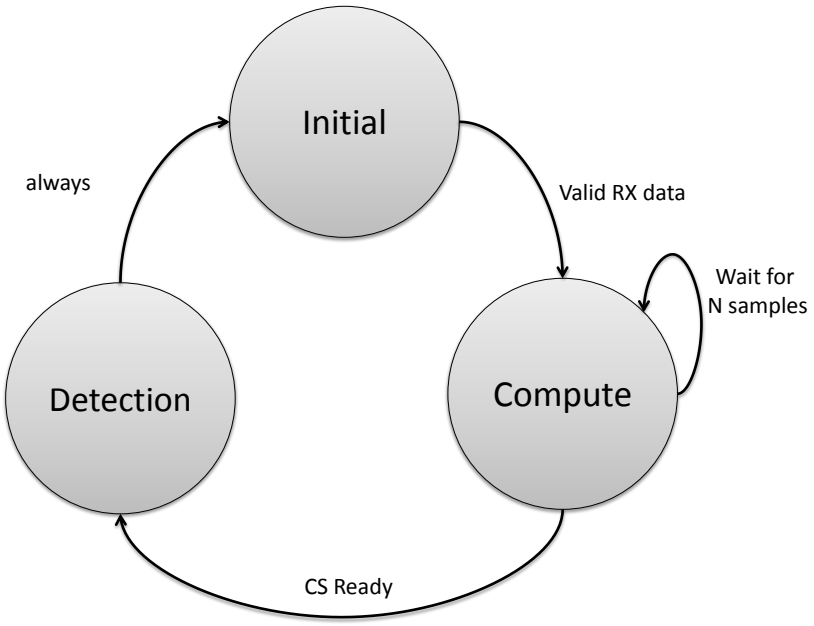

Figure 4.6: The finite state machine of the carrier sense controller module.

this implementation N=64). The average magnitude of these N samples is then calculated. When the average is obtained, the sample ready signal is asserted, instructing the controller to transition to the DETECTION state.

In the DETECTION state, the CS controller compares the current average calculated against a programmable threshold value. Since the USRP E100 has a 14-bit AD converter, we can expect the theoretical dynamic range, determined by 6.021*n + 1.763 dB [27], where n is the number of total bits of the ADC, to be 86.003 dB. The noise floor (no detectable packets) of the ADC requires a maximum 4-bit levels, which is 25.8 dB. We determined that a detectable frame requires 7-bit levels; in other words, the maximum magnitude of the signal during reception of a detectable frame required 8-bit levels, but the least measured

magnitude of a frame required 7-bit levels, so we chose 7-bit level (43.9 dB) to guarantee that the frame will be detected. If the current average is above the threshold, then the Busy/Clear signal is asserted, which instructs the Tx path in the FPGA to stop its transmission until the next carrier sense phase. If the current average is below the threshold, then the state machine transitions back to the INITIAL state, where the carrier sense phase is repeated.

To allow the carrier sense module to pause/resume transmissions based on the state of the channel a small change needed to be made to the VITA Tx control state machine, see Figure 4.7. The Idle state is the default state for when the Tx chain is not in use. The

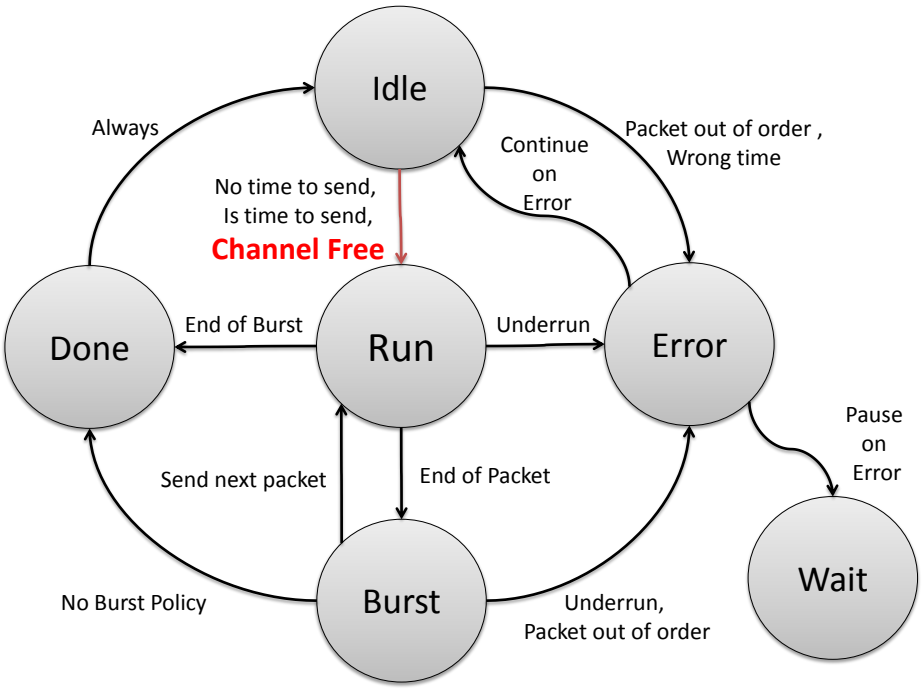

Figure 4.7: Vita TX Controller State Machine.

original state machine waits in the Idle state until it receives data to transmit. If the data

is valid and has no time stamp or it is valid and now is the time to send the frame, the state machine transitions to the Run state. In the Run state the frame is transmitted on the channel. At the end of the frame transmission the state machine either transitions to the Done state if the end of frame is also the end of the burst or the Burst state if more data is available to transmit.

The Burst state runs some error checks and then transitions back to the Run state if the UHD setting allows burst mode; otherwise, the state machine transitions to the Done state. The Done state alerts the UHD that a frame has been sent and always transitions back to the Idle state. All states may transition to the Error state if an error is detected. Once in the Error state the state machine can either pause in the Wait state for user intervention or transition back to Idle after informing the UHD of the error. This choice is based on a UHD setting.

The changes we have made to this state machine are minimal. Now, instead of transitioning from the Idle state to the Run state when data is valid and when the timing conditions are right, we also check that the channel is free. If all of these conditions are true then the state machine can transition to the Run state. There was no need for other changes to the state machine—in the Burst state for example—as we assume that all nodes broadcasting at the carrier frequency are carrier sense enabled and as such will not broadcast on the channel until the current broadcast is complete. If the carrier sense module was being used to avoid a primary user of the channel the Run and Burst states would also need to check that the Channel is free between frames.

Since we implemented the carrier sense code in FPGA, the carrier sense mechanism works independently of our MAC implementation in software. So, when frames are sent from the MAC layer in Iris, the MAC has no control over what is done in the FPGA. There are several advantages associated with implementing carrier sense in FPGA. Firstly, the FPGA-based carrier sense module can be used with any SDR software framework as it does not require any specialized software modules.

Secondly, the carrier blind spot [7] is significantly decreased, since the latency from sensing to sending frames is greatly reduced as compared as to a full software implementation of CSMA. This decreases the chances of collisions with other nodes' transmissions.

4.4.2 IRIS SDR software enhancements

Spreading the MAC between the FPGA and software implementation has some drawbacks, as the software MAC does not know if a frame has been sent or if the frame is just waiting in the Tx buffer. The original ARQ MAC protocol (described in chapter 3) requires knowledge of when a frame is sent across the network. In the software-only version of the protocol it could be assumed that the frame would be sent immediately. The time spent forwarding of the frame through the FPGA logic is relatively fixed due to the deterministic nature of FPGA signal processing. It is necessary for the MAC to estimate the time in order to start the retransmission counter; the frame is then retransmitted if no ACK is received by the MAC before the timer is up.

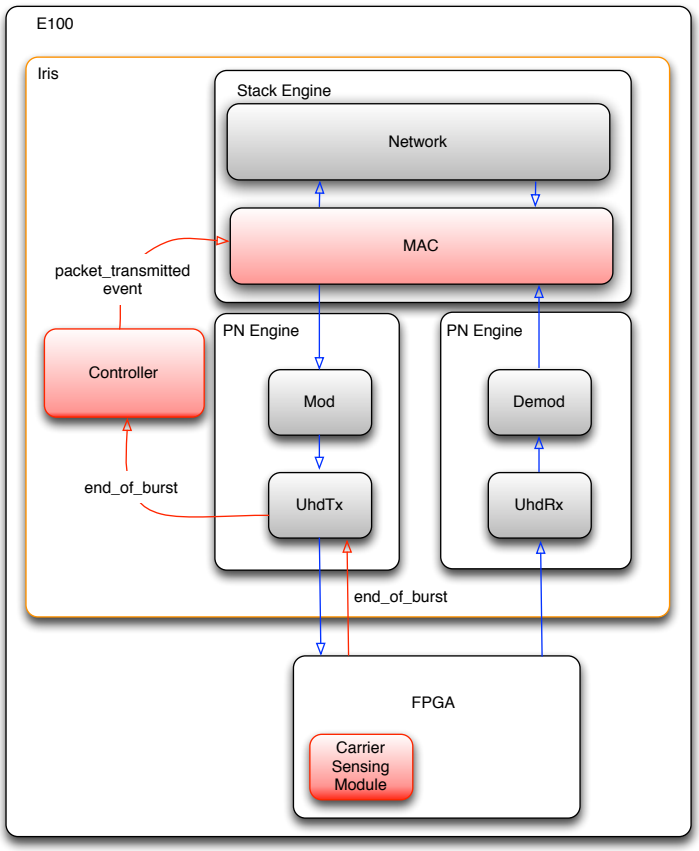

Figure 4.8: Iris node - software Implementation. The original system is shown in grey which added features to allow carrier sense feedback to the MAC highlighted in red

However, in the FPGA/software system the MAC unit can no longer assume that the frame is sent and so does not know when to start its transmission timeout timer. If the MAC starts the timer immediately, and the carrier sense unit stops the frame from being transmitted, the MAC unit might assume the frame has been lost and retransmit it. This would lead to unnecessary frames being sent immediately on the channel and a lower throughput in the system. Figure 4.8 shows a block diagram of the software implemented on the E100. Iris components used in the software only version protocol, which do not

interface to the FPGA, are shown in grey. The modifications necessary to allow feedback from the FPGA carrier sense module are highlighted in red. In order to give feedback to MAC functions implemented in software as to when a frame is sent we exploited the existing frame sent feedback chain of UHD. When a frame is sent the USRPs transmit state machine transitions to the Done state, see Figure 9, which in turn causes a frame sent (end of burst) signal to be forwarded to the UHD. We created an Iris controller block that eavesdrops on these UHD status messages and reports back to the MAC unit whenever a frame is transmitted over the air. The MAC was changed to utilize this information to start the retransmission timeout timer. Thus, the retransmission timeout timer is only initiated once the frame has been sent across the channel instead of when it leaves the MAC layer. In this way, the software and hardware work in unison to provide access to the channel.

4.5 Tests and Results

Our two-hop testbed network consisted of three USRP E100 nodes with the XCVR2450 daughterboard as an RF front-end. We programmed each node to transmit 100 frames from source node to destination node. Every node is within the interference range of every other. Tables 4.1 and 4.2 lists parameters from the experimental setup and Figure 4.9 shows the in-lab setup of our testbed.

Since the wireless channel and the SDR latencies introduce unpredictability to the system

Parameter	Value
Packet size	600 B
Center Frequency	5.09 GHz
Bandwidth	200 Khz
Modulation Scheme	QPSK
Active data subcarriers	192 of 256
FFT size	256
UHDRx buffer size	4196
ARQ timeout	200 ms

Table 4.1: Table of PHY and MAC parameters used

Component	Version
UHD/E100 FPGA	003.003.000
IRIS	2

Table 4.2: HW & SW version

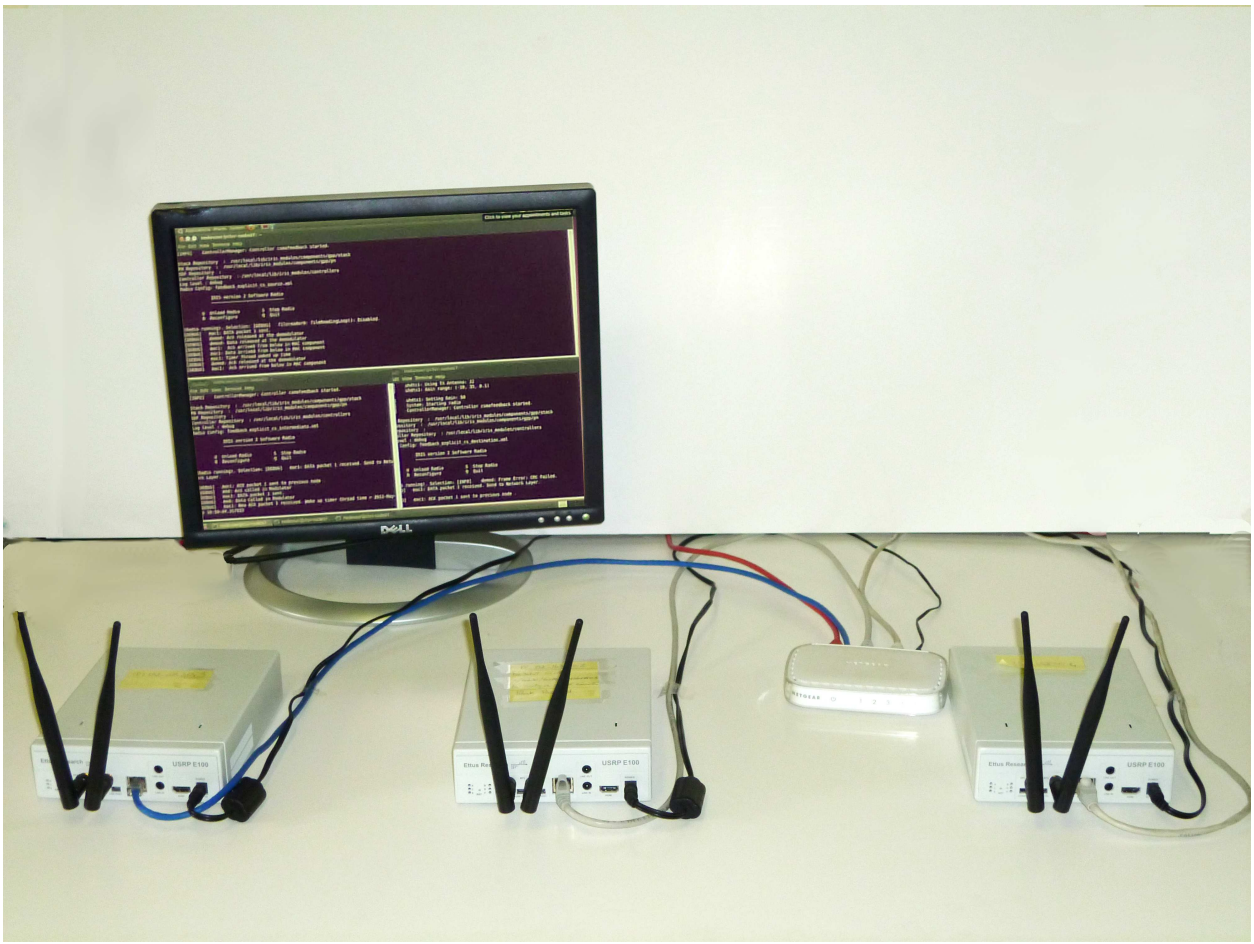

Figure 4.9: Multihop experiment setting in the CTVR wireless lab. This photo was taken by Paolo di Francesco at Trinity College and used with permission.

performance, we ran each experiment 50 times. To understand the system performance we looked at the total number of retransmissions and end-to-end throughput.

Figures 4.10 and 4.11 depict the DATA and ACK exchange of Aloha with explicit ACK strategy and Aloha with implicit ACK strategy. The Aloha with explicit ACK strategy shows a typical example where collisions are frequent in the channel between frames. The CSMA in FPGA in figure 4.11 shows more packets going through the channel and

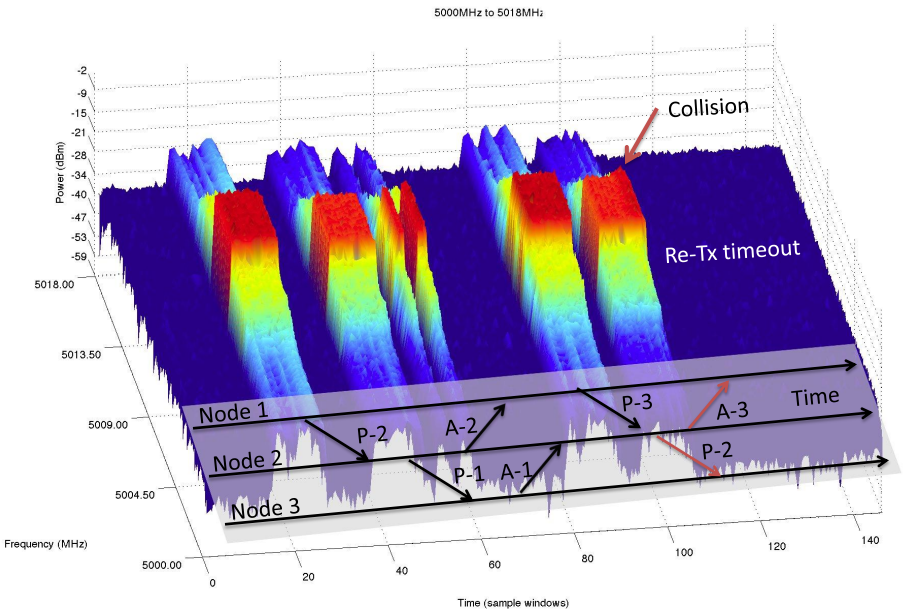

Figure 4.10: Packets exchange sequence in the *Aloha Explicit* without the Carrier Sensing in FPGA. Data captured using a Rohdes & Schwarz FSVR Spectrum Analyzer.

less gaps, which represents timeouts, among transmitting nodes.

There are a few observations from Figures 4.12 and 4.13 First, Aloha with implicit ACKs had less variation in retransmissions than Aloha with explicit ACKs. Second, carrier sense for both protocols decreases retransmissions; however, retransmissions can still occur due to the timeout expiration from the MAC due to long delays in the receive path.

Figure 4.12 shows that the carrier sensing throughput improves end-to-end throughput for Explicit and Implicit ACK strategies by avoiding collisions between packets exchanged by nodes in each other's interference range.

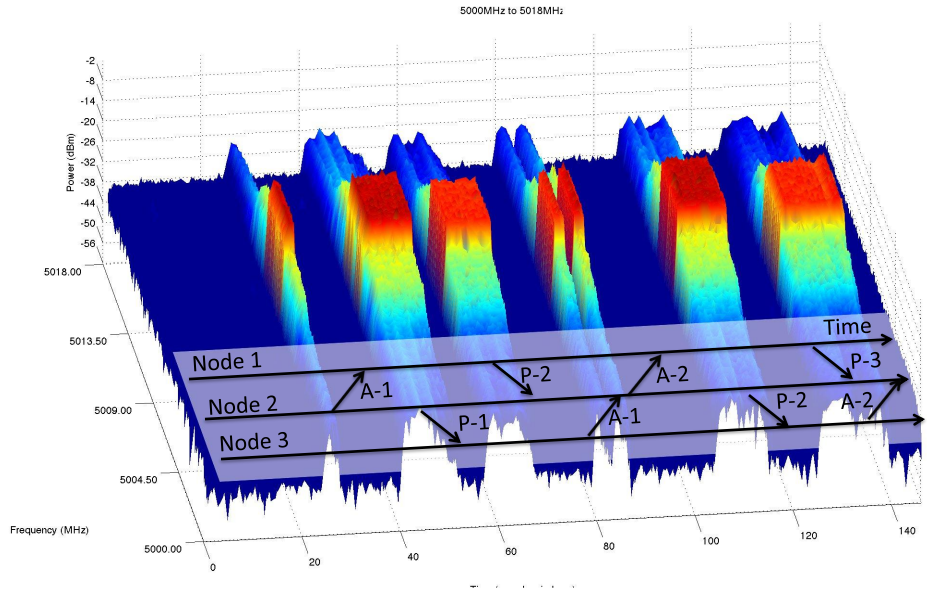

Figure 4.11: Packet exchange sequence for *Aloha Explicit* employing the Carrier Sensing module in FPGA.

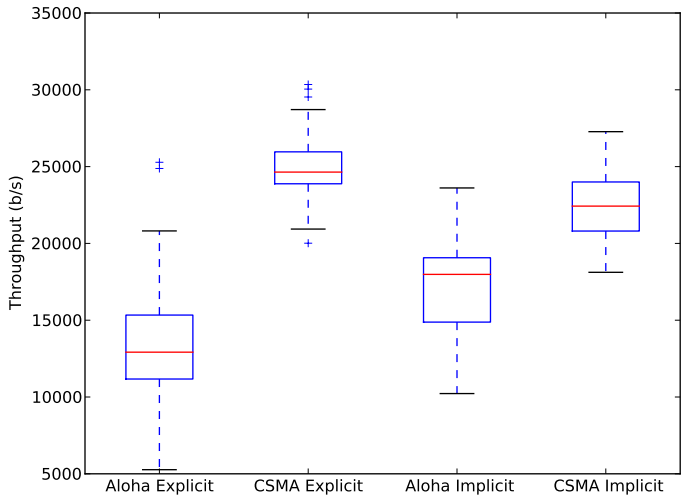

Figure 4.12: Experimental and Simulation Throughput

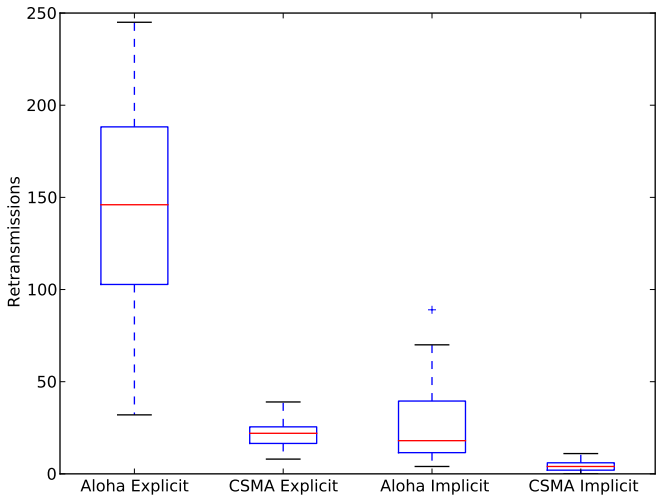

Figure 4.13: Experimental and Simulation Retransmissions. For each box, the whiskers below and above show the minimum and maximum value, respectively. The bottom and the top of the box represent the 1st and 3rd Quartile respectively. The horizontal line inside the box represents the median. Outliers are excluded from calculation of min/max and are shown as small crosses.

4.6 Conclusion

We have successfully implemented random access, carrier-sensing MAC protocols for inexpensive embedded and non-embedded SDR platforms. We utilized the split-functionality approach where we implement carrier-sense on FPGA and enable our original MAC protocol in software built with Iris to frame transmission status. We tested the performance of our MAC protocols on the USRP E100, and showed that our carrier-sensing MAC protocol performs well in a two-hop network. More specifically, we discovered the CSMA MAC protocol outperforms the carrier-sense Implicit ACK MAC protocol because we have decreased the carrier sense blind spot.

Chapter 5

Conclusion

5.1 Conclusion

This research involved two major focuses: assessing current SDR platforms—both GPP-based and hardware-based—to establish requirements for embedded SDR platforms for MAC development and the realizing of several random access MAC protocols for embedded and non-embedded SDR platforms. These research endeavors provide the means to explore advanced MAC implementations on embedded platforms and have advanced the open-source SDR community as a whole. Carrier sense implementation for the USRP E100 was presented in this work as an SDR agnostic FPGA block. Although we used IRIS SDR to develop the MAC protocol, the carrier sense block can be used to improve random access protocols in any SDR. We tested the performance of our two MAC protocols,

Explicit ACK and Implicit ACK, and then, we implemented carrier sensing in FPGA and tested it with our two protocols. Overall, our carrier sensing enabled protocols outperformed our non-carrier sensing protocols in terms of end-to-end throughput and retransmissions. We plan to publish our implementation in an open-source code repository for other SDR developers to utilize in their SDR experiments.

5.2 Contributions

The following are research contributions from the above endeavor:

- assessment of available SDR platforms,
- development and testing of Aloha enhancements (Explicit ACK with delay and Implicit ACK) , in a multi-hop environmnent,
- development, integration, and testing of a carrier sense block implemented in FPGA, and
- development and testing of simulation modules to cross-validate our experimental work.

5.3 Publications

Accepted 2011 Poster Presentation:

1. J. OSullivan, P. Di Francesco, U.K Anyanwu, L. DaSilva, and A. MacKenzie, "Multi-hop MAC implementations for affordable SDR hardware," in New Frontiers in Dynamic Spectrum Access Networks (DySPAN), 2011 IEEE Symposium on. IEEE, 2011, pp. 632–636.

Sumbitted 2012 DySPAN paper:

1. U.K Anyanwu, P. Di Francesco, J. OSullivan, McGettrick Séamas, L. DaSilva, and A. MacKenzie, "A Reconfigurable Random Access MAC Implementation for SDR Platforms," (DySPAN), 2012.

5.4 Future work

In this thesis, we have successfully implemented a carrier sense, random access MAC protocol for SDR platforms. In addition, we have promoted the split-functionality approach to SDR development.

There are several enhancements and areas of exploration that can build on this project :

- adding a random backoff module in FPGA,
- adding block acknowledgment logic to IRIS to improve MAC efficiency, and
- testing CSMA nodes in larger, more complex (and realistic) topologies.

Bibliography

[1] [Online]. Available: http://www.signallake.com/innovation/SWRprimer.pdf

[2] J. Mitola, "Cognitive radio: An integrated agent architecture for software defined radio," *Doctor of Technology Dissertation, Royal Institute of Technology, Sweden*, 2000.

[3] "Gnuradio source." [Online]. Available: http://gnuradio.org/redmine/projects/gnuradio/wiki/Download

[4] P. Sutton, J. Lotze, H. Lahlou, S. Fahmy, K. Nolan, B. Ozgul, T. Rondeau, J. Noguera, and L. Doyle, "Iris: an architecture for cognitive radio networking testbeds," *IEEE Communications Magazine*, vol. 48, no. 9, pp. 114–122, 2010.

[5] "Atheros." [Online]. Available: "http://www.qca.qualcomm.com/media/product/product_79_file1.pdf"

[6] G. Nychis, T. Hottelier, Z. Yang, S. Seshan, and P. Steenkiste, "Enabling mac protocol implementations on software-defined radios," in *Proceedings of the 6th USENIX sym-*

posium on Networked systems design and implementation. USENIX Association, 2009, pp. 91–105.

[7] T. Schmid, O. Sekkat, and M. B. Srivastava, "An Experimental Study of Network Performance Impact of Increased Latency in Software Defined Radios," in *Proc. of ACM workshop on Wireless network testbeds, experimental evaluation and characterization (WinTECH)*, 2007.

[8] P. Murphy, A. Sabharwal, and B. Aazhang, "Design of WARP: A wireless open-access research platform," in *European Signal Processing Conference*, 2006, pp. 1804–1824.

[9] A. Jow, C. Schurgers, and D. Palmer, "CalRadio: A portable, flexible 802.11 wireless research platform," in *Proceedings of the 1st international workshop on System evaluation for mobile platforms*. ACM, 2007, pp. 49–54.

[10] D. Tucker and G. Tagliarini, "Prototyping with GNU Radio and the USRP-where to begin," in *Southeastcon, 2009. SOUTHEASTCON'09. IEEE*. IEEE, 2009, pp. 50–54.

[11] M. Robert, S. Sayed, C. Aguayo, R. Menon, K. Channak, C. Vander Valk, C. Neely, T. Tsou, J. Mandeville, and J. Reed, "Ossie: Open source SCA for researchers," in *SDR Forum Techical Conference*, 2004.

[12] C. González, C. Dietrich, S. Sayed, H. Volos, J. Gaeddert, P. Robert, J. Reed, and F. Kragh, "Open-source sca-based core framework and rapid development tools enable software-defined radio education and research," *IEEE Communications Magazine*, vol. 47, no. 10, pp. 48–55, 2009.

[13] C. Aguayo Gonzalez, C. Dietrich, and J. Reed, "Understanding the software communications architecture," *IEEE Communications Magazine*, vol. 47, no. 9, pp. 50–57, 2009.

[14] F. Stefani, A. Moschitta, D. Macii, and D. Petri, "FFT benchmarking for digital signal processing technologies," University of Perugia, Tech. Rep., 2004.

[15] F. Bieberly, "Heterogeneous processing in software defined radio: Flexible implementation and optimal resource mapping," Master's thesis, Virginia Polytechnic Institute and State University, 2012.

[16] A. Puschmann, M. A. R. Kalil, and A. Mitschele-Thiel, "Implementation and Evaluation of a Practical SDR Testbed," in *Proc. of International Conference on Cognitive Radio and Advanced Spectrum Management (CogART)*, 2011.

[17] E. Magistretti, K. Kant, B. Radunovic, and R. Ramjee, "Wifi-nano: reclaiming wifi efficiency through 800 ns slots," in *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking*. ACM, 2011, pp. 37–48.

[18] G. Papadimitriou and A. Pomportsis, "Adaptive mac protocols for broadcast networks with bursty traffic," *IEEE Transactions on Communications*, vol. 51, no. 4, pp. 553–557, 2003.

[19] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000.

[20] R. Dhar, G. George, A. Malani, and P. Steenkiste, "Supporting integrated MAC and PHY software development for the USRP SDR," in *Proc. of the IEEE Worshop on Networking Technologies for Software Defined Radio Networks, 2006*. IEEE, 2006, pp. 68–77.

[21] H. Suganuma, M. Suzuki, and H. Morikawa, "A Buffer Size Tuning Mechanism for Software-Defined Radios," in *Proceedings of the 31ST Annual IEEE International Conference on Computer Communications (INFOCOM 2012)*. IEEE, 2012.

[22] J. O'Sullivan, P. di Francesco, U. Anyanwu, L. DaSilva, and A. MacKenzie, "Multihop MAC implementations for affordable SDR hardware," in *Proc. of the IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN)*. IEEE, 2011, pp. 632–636.

[23] "Ettus Research: Products," 2012. [Online]. Available: https://www.ettus.com/product

[24] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *Proceedings of the 1st International Conference on Simulation tools and techniques for communications, networks and systems & workshops*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008, p. 60.

[25] A. Puschmann, M. A. R. Kalil, and A. Mitschele-Thiel, "Implementation and Evaluation of a Practical SDR Testbed," in *Proc. of International Conference on Cognitive Radio and Advanced Spectrum Management (CogART)*, 2011.

[26] R. Normoyle and P. Mesibo, "The VITA radio transport as a framework for software definable radio architectures," in *SDR 08 Technical Conference and Product Exposition*, 2008.

[27] S. Younis, S. Bazarjani, and S. Ciccarelli, "Programmable dynamic range receiver with adjustable dynamic range analog to digital converter," Oct. 17 2000, US Patent 6,134,430.