SOFTWARE DEFINED RADIO (SDR) BASED IMPLEMENTATION OF IEEE 802.11 WLAN BASEBAND PROTOCOLS

BY

Shravan Kumar Surineni

B. Tech, Jawaharlal Nehru Technological University, 2000

THESIS

Submitted to the University of New Hampshire

in Partial Fulfillment of

the Requirements for the Degree of

Master of Science

in

Electrical Engineering

December, 2004

This thesis has been examined and approved.

_____

Michael J. Carter, Ph.D.
Thesis Director,
Associate Professor,
Department of Electrical and Computer Engineering.

_____

Kondagunta U. Sivaprasad,
Professor,
Department of Electrical and Computer Engineering.

_____

William H. Lenharth, Ph.D.
Research Associate Professor,
Department of Electrical and Computer Engineering,
Director, UNH Research Computing Center.

_____

Kevin Karcz, M.S.
Technical Lead,
Wireless Consortium, UNH- InterOperability Lab.

_____

Date

DEDICATION

To my parents

SREEMAN NARAYANA and LEELAMANI SURINENI

for their support and encouragement

ACKNOWLEGEMENTS

I would like to formally thank Prof. Michael Carter for his expert guidance and support throughout the thesis. I would also like to thank Prof. Sivaprasad and Dr. William Lenharth for giving me an opportunity to pursue graduate studies at UNH and for providing me an opportunity to work at the InterOperability Lab and also for serving on my thesis committee. I would like to express my deepest appreciation to Kevin Karcz for supporting me throughout my graduate career.

I would also like to express my appreciation for the support of the staff of InterOperability Lab, especially Wireless Consortium. Finally, I want to express special thanks to all my friends for their support and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Software Defined Radio Prototyping of IEEE 802.11

Wireless LAN Protocols

By

Shravan K. Surineni

Department of Electrical and Computer Engineering

and

UNH InterOperability Laboratory

The IEEE 802.11 family of wireless LAN protocols defines multiple physical layer implementations of which direct sequence spread spectrum (DSSS, 802.11b) and orthogonal frequency division multiplexing (OFDM, 802.11a/g) are currently the most popular. Market pressures are forcing the convergence of multiple wireless protocols into the same access device, and shortened product design cycles dictate rapid prototyping of new or enhanced protocols. The computationally intensive signal processing algorithms and high data rates associated with these protocols necessitate dedicated hardware implementation of some portions of the signal processing chain, yet allocating separate hardware resources for each of these standards would make the "universal access device" bulky and inefficient. Re-using the same software-

reconfigurable hardware to handle different processing algorithms would enable an efficient, flexible alternative to current prototyping and implementation methods.

In this thesis, the feasibility of using Software Defined Radio architectures as a prototyping tool for wireless LAN baseband signal processor implementations is explored. Signal processing architectures and algorithms for DSSS and OFDM protocols were developed in the Simulink and Matlab environments, and were then translated to VHDL hardware descriptions. A reference design for a OFDM transmitter was synthesized for implementation on a Xilinx Virtex II FPGA, and functional and timing simulations verified the design correctness.

Chapter I

Introduction

1.1 Motivation

There has been rapid growth in the field of Wireless Local Area Networks (Wireless LAN or WLAN) over the last decade. Wireless LANs have transformed from bulky, costly, niche technologies into high performance systems operating at near-wire speeds. The Institute of Electrical and Electronics Engineers (IEEE) is driving the standardization of the wireless LAN technologies and the IEEE 802.11 standard [1] is currently the most popular wireless LAN protocol. Different wireless LAN standards (IEEE 802.11b, IEEE 802.11a and IEEE 802.11g, Bluetooth, HiperLAN etc.) exist that provide users with different levels of connectivity in terms of speed, area of coverage, and ease of use. The convergence of these existing and future technologies would produce a seamless, ubiquitous wireless network with voice, video and multimedia and broadband data services traveling across multiple wireless interfaces providing anytime, anywhere communications to its users. Such technology would enable users to always be connected to a network through a single device which has the ability to run different wireless LAN standards. This poses lot of challenges at the different layers of the network, right from the wireless interface (radio) to the application level. The device would have to monitor the different RF signals on different wireless interfaces and switch to standards appropriately. Implementing these different Wireless LAN technologies on a

single device that is limited in computational power and size would be challenging and exiting. In this thesis, we have focused on the issues in the implementation of the physical layer of the different Wireless LAN protocols, with specific attention to the defining architecture and implementation of the baseband processing of the IEEE 802.11 physical layer.

The IEEE 802.11 defines multiple standards at the baseband, and spread spectrum and orthogonal frequency division multiplexing (OFDM) based standards are currently popular. Several complex and highly sophisticated signal processing algorithms have been developed for these standards to give better bit error rates and improve the quality of these standards. Implementing these multiple standards on the physical layer would require running these different algorithms for different standards on the same device. These computationally intensive signal processing algorithms require lots of hardware resources and allocating separate hardware resources for each of these standards would make the device bulky and inefficient. Reusing the same hardware to handle different algorithms, thus enabling a reconfigurable and flexible software radio system would be an efficient alternative. The feasibility of a Software Defined Radio (SDR) based implementation for two of the IEEE 802.11 physical layer technologies, Spread Spectrum and OFDM is studied in this thesis.

1.2 Contribution

The main contributions of this thesis include MATLAB-based reference implementations of two different physical layers for the Wireless LANs along with

architectures for SDR based prototyping. Analysis of algorithms for generation, detection and estimation of these modulations is carried out for performance evaluation, and architectures for implementing the same in software and hardware are proposed. Computationally efficient algorithms are studied and enhanced to enable simple architectures for SDR-based Wireless LAN implementation.

There has been research in the field of implementation issues for the different physical layers for the Wireless LAN protocols, but little work has been done on the aspect of reconfigurability of these systems. Digital Signal Processing (DSP) would seem to be an obvious choice for a reconfigurable solution for WLAN protocols. The WLAN supports high data rates (in the range of 1-54 Mbps) and higher data rates would be supported by the next generation WLAN standards. To achieve these high rates, significant processing needs to be done in dedicated hardware. The most powerful current DSPs can only offer decoding rates of up to a few Mbps, and are not scalable with the evolving standards. As a result, we have chosen an FPGA implementation over a DSP implementation.

Kevin Karcz at University of New Hampshire's InterOperability Lab has designed a sytem level implementation of OFDM system. The developed system operates on a packet level and the receiver assumes the samples of the entire packet are available before demodulation. This system has been modified to suit a real-time operation working on a symbol by symbol basis. New architecture is proposed and system level design is carried out, to enable hardware prototyping. The proposed reference architecture for SDR based implementation uses a parallel and pipelined approach to achieve the desired rates, using significant hardware resources. Such a hardware intensive

approach is not suited for the uniprocessor architecture of a DSP. The proposed architecture enables implementation of the design on a Xilinx Virtex-II FPGA. The proposed architecture is capable of supporting up to 36 Mbps data rates.

The SDR-based prototyping of the WLAN baseband processor is broken into three parts:

1. Understanding the communication algorithms used in the WLAN baseband processor and implementing these algorithms in software
2. Designing a flexible and simple hardware architecture based on the initial software implementation above.
3. Implementing these algorithms in hardware.

1.3 Overview

The rest of the thesis is organized as follows: In Chapter 2, we will briefly describe the evolution of different WLAN standards and build a better understanding of the different physical layers used in these protocols. Spread spectrum and OFDM modulation are described in detail and the main building blocks of these modulation schemes are discussed. In Chapter 3, we discuss the choice of hardware for prototyping the WLAN baseband. The advantages and disadvantages of the two hardware approaches, the DSP-based implementation and FPGA-based implementation are discussed in detail. Chapter 4 discusses the different prototyping and simulation methodologies available for rapid prototyping of communication systems and explains the data flow of the design process. Chapter 5 discusses the high-level design of an OFDM baseband processor and the modifications done to the original implementation done by Kevin, to enable a SDR based prototyping of the OFDM receiver. The implementation is done in MATLAB [2] . Some

well-known architectures and signal processing algorithms are presented and trade-offs for different architectures are examined. Chapter 6 discusses the low-level design using block diagrams and the implementation issues. The issues in implementing communication and DSP algorithms in hardware and using Xilinx Intellectual Property (IP) cores are described. Finally, we conclude with some directions for future work in Chapter 7.

Chapter II

Background

2.1 Introduction

The Institute of Electrical and Electronic Engineers (IEEE) 802.11 standards are the most widely deployed Wireless Local Area Network (WLAN) standards compared to the other proprietary protocols. The IEEE 802.11 standard, completed in 1999 and reaffirmed in 2003 [1], specifies an operating air interface in the unlicensed 2.4 GHz Industrial, Scientific, and Medical (ISM) band. The base standard includes both Frequency Hopping (FHSS) and Direct Sequence (DSSS) spread spectrum methods. The DSSS mode is based on a 11-chip Barker spreading code. BPSK and QPSK modulations are used. The IEEE 802.11a standard, also released in 1999 defines an operating air interface in the 5.1-5.8 GHz Unlicensed National Information Infrastructure (UNII) bands. It uses Orthogonal Frequency Division Multiplexing (OFDM) to achieve data rates up to 54 Mbps. The IEEE 802.11g standard, released in June 2003, uses the same OFDM modulation as 802.11a, but enables operation in the 2.4 GHz ISM band. Table 2-1 gives a summary of these different standards and the technologies used. In the following sections we will describe these technologies, along with the basic concepts of spread spectrum and OFDM.

	IEEE 802.11b	IEEE 802.11a	IEEE 802.11g
RF Freq.	2.4 GHz ISM band	5.15-5.825GHz UNII-2 bands	2.4 GHz ISM band
Spreading	Direct Sequence	OFDM	OFDM, Direct Sequence
Bandwidth	22 MHz	20 MHz	20 MHz
Modulation	QPSK	BPSK, QPSK, 16QAM, 64QAM	BPSK, QPSK, 16QAM, 64QAM
Max. Power	100 mW	800 mW	100 mW
Throughput	1-11 Mbps	6 – 54 Mbps	1 – 54 Mbps
MAC	CSMA/CA	CSMA/CA	CSMA/CA
Sensitivity	-80 dBm	-80 dBm	-80 dBm

Table 2-1 WLAN Standards Comparison

2.2 Spread Spectrum

Spread spectrum is a form of digital communication technique, which has been used in military communications for years. The core principle behind spread spectrum is the use of noise-like carrier waves, and as the name implies, bandwidths much wider than required for simple point-to-point communication at the same data rate. There are fundamentally two different types of spread spectrum: frequency hopping spread spectrum, and direct sequence spread spectrum.

2.2.1 FHSS – Frequency Hopping Spread Spectrum

In Frequency Hopping Spread Spectrum a single carrier switches frequency to reduce the likelihood that it will interfere with, or be interfered by, other carriers. The FHSS physical layer was part of the base standard and is not used in the higher rate extensions of the base standard.

2.2.2 DSSS – Direct Sequence Spread Spectrum

The energy in a single carrier is spread over a wider spectrum by multiplying data bits with a special 11-bit pattern called a "spreading sequence". A spreading sequence typically consists of a combination of +1/-1's, which is chosen such that it satisfies some unique auto/cross correlation properties. The length of the spreading sequence is fixed and is known as the "spreading gain". In the time domain, each bit is spread into a number of chips equal to the spreading gain and correspondingly, the frequency spectrum of the signal also expands by the same factor. This expansion in the frequency domain makes Direct Sequence Spread Spectrum signals resistant to narrow band interfering noise. The IEEE 802.11 specification uses an 11-bit spreading sequence called a "Barker Sequence". The IEEE 802.11b-1999 version uses an 8-bit sequence. Two spread sequences are defined, a mandatory Complementary Code Keying (CCK) and an optional Packet Binary Convolutional Coding (PBCC).

2.2.2.1 CCK – Complementary Code Keying

CCK is used to increase IEEE 802.11b's peak data rate from 2 Mbps to 11 Mbps. It does this by increasing the symbol rate from 1 Mbps to 1.375 Mbps, then taking the

data in 8-bit blocks (8*1.375 = 11). Six of the 8 bits are used to choose 1 of 64 complementary codes, which are 8 chips long and clocked out at 11 MHz. The other two bits are combined with the code in the QPSK modulator.

2.2.2.2 PBCC – Packet Binary Convolutional Coding

The optional PBCC is also defined in the IEEE 802.11b/g specifications. PBCC uses Forward Error Correction to improve the link performance when noise is the limitation. The data is encoded using a convolutional encoder, which utilizes a 6-stage memory. The encoder produces two bits for each input data bit and these output states are mapped to QPSK states. A code word controls how a chosen word alternates over time.

2.2.2.3 OFDM – Orthogonal Frequency Division Multiplexing

Orthogonal Frequency Division Multiplexing was adopted as the modulation technique for the IEEE 802.11a high rate wireless LAN and for the IEEE 802.11g high-speed extensions in the 2.4 GHz band. It uses a multi-carrier approach and is considered a robust method to overcome the hostile effects of a wireless channel such as multi-path propagation.

The basic principle of multi-carrier modulation is to de-multiplex the original high rate data stream into a number of parallel lower data rate streams and then modulate a different carrier with each of these lower data rate streams with a different frequency. The resultant signals are transmitted together in the same band. The separation between the different carrier frequencies imparts the “orthogonal” nature to this frequency division-multiplexing scheme. The frequencies are chosen such that the spectral null of

one modulated carrier is exactly at the frequency of an adjacent carrier. Even though this would make the spectra of the modulated carriers overlap, it ensures that the signals at the different frequencies are recovered at the receiver without significant mutual interference. To achieve this end, it would require a large number of oscillators, each locked to precise frequencies, for the orthogonality of the carriers to hold. This is expensive and cumbersome to implement in practical systems.

However, the advances in Digital Signal Processing, particularly the Fast Fourier Transform (FFT), make implementation of OFDM viable. Using IFFT and FFT, data can be digitally modulated/demodulated onto a large number of carriers. Another important feature of an OFDM system is the length of the FFT transform. The set of sub-carriers generated during one transform is called an OFDM symbol. The duration of the OFDM symbol, or the length of the FFT, is related to one of the main reasons for using OFDM, which is the reduction of multi-path or echo effects. The longer the length of the FFT, the longer the echoes that can be handled. For indoor wireless networks, the transmitter and the receiver may be separated by up to 100 m, and the delay spreads are up to 300 *ns*, and the channel bandwidth is in the range of 20 MHz. When using a 20 MHz channel and with a OFDM symbol duration of 3.2 *us,* the number of carriers required would be 64, and a 64 point FFT would be sufficient.

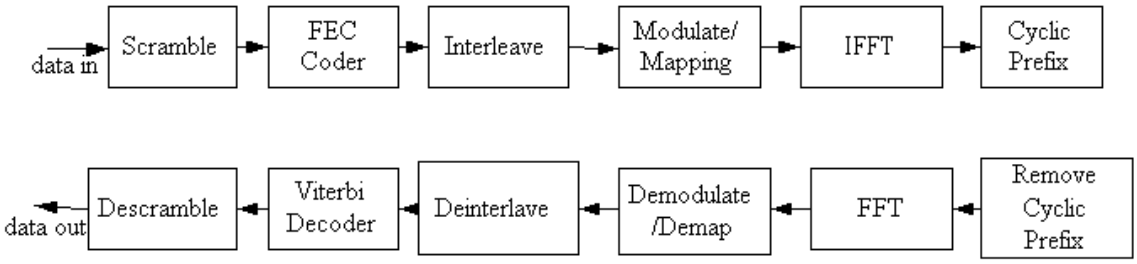

Figure 2-1 Components of OFDM system

Figure 2.1 shows the various components of an OFDM transmitter and receiver. The data entering into the transmit chain is scrambled and convolutionally encoded. The encoded data is sent to the interleaver. The interleaver consists of a block interleaver and a matrix interleaver. The interleaver decorrelates the data and spreads adjacent data over many subcarriers. The data is then passed to a modulation mapper, which is commonly Phase Shift Keyed (PSK) or Quadrature Amplitude Modulation (QAM) depending on the type of communication system. The IEEE 802.11a standard specifies usage of Binary PSK, Quaternary PSK, 16 QAM and 64 QAM. While the wireless standards use lower order modulations, wired OFDM technologies such as the digital subscriber loop (xDSL) use higher order modulations up to 256 QAM and 1024 QAM. The modulator thus concurrently and separately modulates a set of tones in the OFDM spectrum. The output of the modulator is then converted from serial to parallel form and the complex frequency domain data is thus transformed to time domain using Inverse Fast Fourier Transform (IFFT). The time varying data is then cyclically extended with a cyclic prefix to reduce inter-symbol interference between successive OFDM symbols. The cyclic extension of the OFDM symbol, or the guard interval, also helps in reducing the effects of multipath

as long as the guard interval is greater than the maximum channel delay spread albeit at the cost of a part of signal bandwidth.

Taking the cyclic prefix into account, the signal model for OFDM transmission over a multipath channel becomes very simple. The transmitted symbols $X_{l,k}$ at time slot l and sub-carrier k are solely affected by a factor $H_{l,k}$ which is the channel transfer function (Fourier transform of channel impulse response) at that sub carrier frequency and by additional white Gaussian noise. Equation 2.1 below, shows the influence of the channel and noise on the received signal.

$$Y_{l,k} = H_{l,k} * X_{l,k} + N_{l,k} \tag{2.1}$$

Where * indicated convolution operation.

An equalizer block follows the removal of the cyclic prefix and the FFT, and removes the influence of the channel. The FFT brings the data back to the frequency domain and the equalizer retrieves the input data symbols using an estimated channel inverse model. To help in estimation of the channel transfer function, special training symbols are embedded in the OFDM symbol.

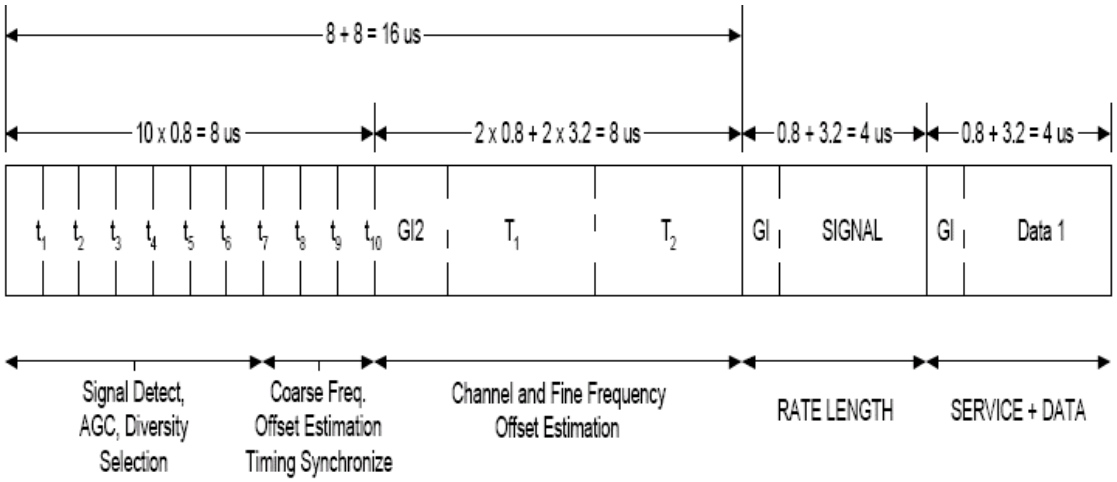

Figure 2-2 OFDM packet structure

The channel transfer function is obtained by interpolation between the coefficient values estimated from the training symbols. However, this simple equalization scheme might not work for those frequencies where the channel transfer function is relatively small because dividing by a small value amplifies the noise, and thus make detection of the input data difficult. Since the data was channel-coded at the transmitter, one can still retrieve the symbols that experienced a bad channel with the help of those that were received successfully. Following the equalizer block is the Viterbi decoder for decoding the convolutionally coded data. The descrambler following the decoder, outputs the original bit stream sent for transmission.

2.3 Summary

The two popular WLAN systems use two different protocols in the physical layer, namely Spread Spectrum and OFDM. The baseband processing of these two systems is vastly different with one using a single carrier modulation and the other using multi-

carrier modulation. In particular, the OFDM physical layer supports a variety of data rates by using convolutional coding and BPSK, QPSK and QAM modulations. The error-correction code used in OFDM systems has rate ½, K=7, and polynomials {133, 171}. The rate of the convolutional code determines the amount of redundancy added to the transmitted code word. A lower code rate is thus stronger, but makes inefficient use of bandwidth. The higher order modulations required for higher rates also demand high SNR at the receiver to be able to demodulate the signal. Thus the longer constraint length and higher order modulation increases the complexity of OFDM systems. Thus the OFDM systems require signal processing architectures that support data rates in the range of 6-54 Mbits/s. Before we describe the architecture used for simulating and prototyping the OFDM system, we shall review some of the important aspects of simulation and prototyping of communication systems and WLAN systems in particular.

Chapter III

Prototype Hardware Selection

3.1 Introduction

The continuously increasing number of competing standards in the wireless arena reduces the life cycle of products and makes it necessary to prototype these different standards while they are still at draft stage, so that the time required to get the products out would be significantly less by the time the draft proposals are standardized. The demand for design flexibility and the availability of fast digital signal processors (DSP) and reconfigurable logic (FPGA, PLD) makes digital design an ideal choice for prototyping different wireless standards. A flexible platform with a single user interface, which would enable one to prototype these different WLAN protocols, is desired [18]. The radio-frequency section, analog-to- digital interface, baseband functional blocks, and a platform that offers flexibility and allows reconfiguration in implementing these various blocks is discussed in this chapter.

3.2 Software Defined Radio (SDR) Architecture

An ideal Software Defined Radio (SDR) is entirely implemented digitally, so that it can be completely reconfigurable via software. This section explains the generic SDR

architecture that would permit implementation of different WLAN protocols on a single design.

3.2.1 Radio- Frequency Front End

Two different methods exist for converting a radio frequency signal to a baseband signal. In the classic super-heterodyne architecture, the signal is first converted to an intermediate frequency (IF), is amplified and filtered, and then converted to baseband as shown in Figure 3.1. In the direct-conversion technique, also called Zero IF (ZIF) technique, the entire frequency range of interest is directly converted to baseband, without involving any intermediate stages. The direct conversion method would be the ideal radio interface for a software radio implementation. Although the choice of IF does influence the rest of SDR architecture, it will not be discussed further as front-end design is not part of the project.

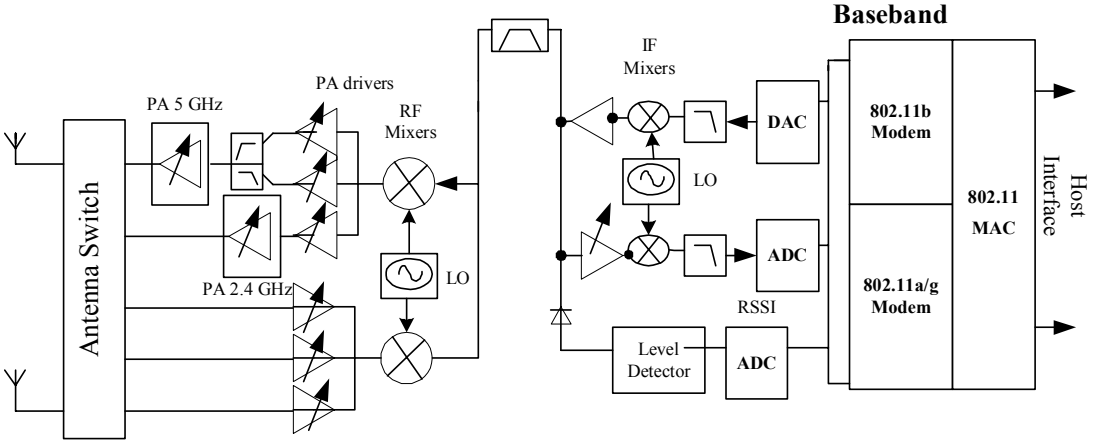

Figure 3-1 Generalized WLAN interface

3.2.2 Analog/Digital Converters

A/D and D/A converters are critical blocks as they are the interface between the analog and digital domains. They are largely responsible for modem performance and are subject to many constraints. Signal SNR is linked to converter resolution by the equation

$$SNR_{A/D} = 1.76 + 6.02b + 10 \log (2BW/F_{sampling}) \quad (3.1)$$

Where b is the DAC resolution in bits, $F_{sampling}$ is the sampling frequency and BW the bandwidth of interest. $F_{sampling}$ is bound by the Nyquist theorem as twice the bandwidth of the signal to be sampled. The band of interest is 22 MHz for 802.11b systems and 20 MHz for 802.11a systems requires a sampling frequency of 44 MHz and 40 MHz respectively. The signal quality goals are defined by the requirements set by the standards. The goal of the thesis is to implement the baseband section of the 802.11a/b/g protocols and hence the ADC/DAC selection is not discussed further.

3.2.3 Baseband Signal Processing

IEEE 802.11b uses BPSK and QPSK modulation, and IEEE 802.11a/g uses BPSK, QPSK, QAM16 and QAM64 modulations. IEEE 802.11a/g uses OFDM as a multiplexing technique. OFDM is a multi-carrier modulation where a high rate data stream is divided into a number of low rate data streams and each carrier carries one of these low rate data streams. FFT-based processing is used to convert the signals from time domain to frequency domain and vice versa in OFDM modulation. The implementation of the baseband or digital signal processing section is important, as there are several options available. The possible options are:

1) Multi- mode ASICs where device functionality can be switched according to the mode of operation,
2) DSP-based implementation
3) Programmable logic-based implementation (using FPGA or PLD).

The long design cycle of ASICs makes them unsuitable for prototyping. Both DSP and programmable logic based designs have advantages over other techniques. The next section discusses the advantages of both FPGA-based and DSP-based implementations and how the FPGA-based implementation is ideal for Software Defined Radio based prototyping of WLAN baseband processing.

3.3 Comparison of DSP and FPGA systems

Comparison of the signal processing complexity of IEEE 802.11 baseband signals in DSP and FPGA systems is discussed in this section.

3.3.1 Introduction to DSP and FPGA systems

The DSP is more efficient specialized microprocessor, typically programmed in C or in assembly code for performance. It is well suited to complex, math-intensive tasks and signal processing. Its performance is limited by its clock rate and the number of operations it can do in a clock cycle. The performance of DSP chips is generally given in terms of Millions of Instructions Per Second (MIPS). Programmable logic (FPGAs), though not as flexible as DSP, allows performance of the most computationally demanding operations in dedicated hardware sub-modules running in parallel. The

number of available gates on the FPGA limits its performance, and the performance of the FPGAs is generally given in terms of the number of available system gates and the maximum clock speeds. FPGAs are uniquely suited for repetitive DSP tasks, such as multiply and accumulate operations (MAC), because they can perform these repetitive tasks in parallel. The Virtex II from Xilinx represents the state of the art in FPGA with up to 6 Million system gates and typical clock frequencies up to 300 MHz. The TMS 6000 series from Texas Instruments represents the state of the art in DSP chips with up to 2400 MIPS and clock speeds up to 300 MHz. However, with FPGAs, achievable clock speeds are often far less than the stated maximum clock speed, whereas with DSPs the MIPS rate is more often achieved. Efficient architecture is the key in utilizing the full clock speed of an FPGA.

3.3.2 DSP and FPGA Architectures

The Architecture of DSP and FPGA systems is explained in this section.

3.3.2.1 Architecture of DSP Processor

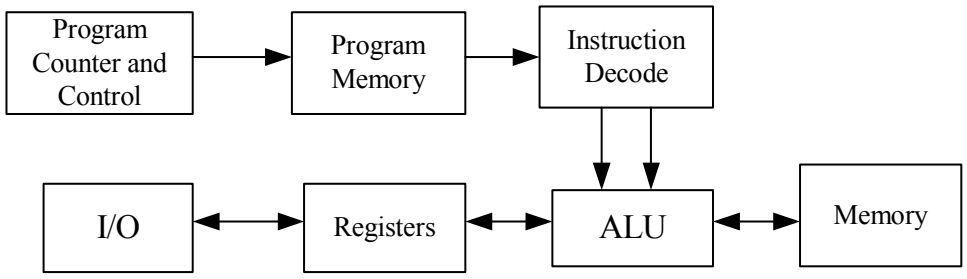

Figure 3-2 Architecture of a DSP processor

In a DSP processor, the ALU is tailored towards DSP functions. The main DSP function is the "multiply and accumulate" operation and the important factor is memory bandwidth. The program, which is stored in the program memory, determines the execution sequence and the program memory itself might be an overhead. The DSP structure is generally fixed with a fixed size data width, and some of the algorithms might not need the full data width. The ALU itself has instructions to support lots of different algorithms, and the algorithms needed in 802.11 signal processing might not need all the instructions supported. Flexibility of a DSP processor lies in the fact that it executes instructions sequentially. But it would be disadvantageous when executing loops and for doing data input and output operations.

Sample Rate	**Operations per Sample Period**	
	Two Multiplier ALU at 200 MHz	**Four Multiplier ALU at 400 MHz**
8 kHz	50000	200000
1.2288 MHz	325	1302
20 MHz	20	80
44 MHz	9	36
100 MHz	4	16

Table 3.1 Comparison of ALU performances

The algorithm performance in a DSP processor (which in turn determines the maximum sample rate) is determined by the speed (maximum clock rate) supported by ALU, the number of parallel operations supported by the ALU and the number of operations required by the algorithm on a per symbol basis.

As seen from the Table 3.1, the DSP processor has much flexibility for voice applications, which uses 8 kHz sampling rate, and for CDMA cellular phone communication chip rates, which uses 1.2288 MHz sample clock. The 20 MHz and 44 MHz sample rates are associated with 802.11a and 802.11b systems and the number of operations that can be done per sample is reduced to 36. High rate analog to digital converters typically use sample rates above 100 MHz, in applications such as digital down conversion, and the DSP processing power is greatly reduced at this rate. Some important features of the one of the state-of-art DSP processors, the TMS320C6414 from Texas Instruments (Ti) are as given below

- Highest performance fixed point DSP with 2, ns instruction time
- Up to 720 MHz clock rate
- Six ALUs, which support Single 32-bit, Dual 16-bit, or Quad 8-bit arithmetic per clock cycle.
- Two multipliers supporting four 16*16-Bit Multiplies per clock cycle.
- Viterbi decoder co-processor
- L1/L2 Memory architecture that supports 128 K-bit L1P program cache, 128 K-bit L1D Data Cache, 8 M-bit L2 Unified Mapped RAM/Cache
- Two external memory interfaces

3.3.2.2 Architecture of FPGA

There is currently a wide range of FPGA products being offered by many semiconductor vendors, including Xilinx, Altera and Atmel. The architectural approaches

of different manufacturers are different. However, a generalization can be made in that most FPGAs are organized as

- An array of logic elements, and a set of
- Programmable interconnections between the logic elements, the I/O pins, and other resources such as on-chip memory.

The logic elements: Each logic element typically consists of 1 or more n-input RAM based look-up tables where n is between 3 and 6, and several flip-flops.

Configuration: Configuration of the FPGA and interconnection of logic elements, the I/O pins and other resources inside the FPGA is accomplished by downloading a bit-stream into a static RAM configuration memory inside the FPGA. This bit stream defines the functionality of each of the logic elements and the internal routing between each of the logic elements. Different applications can be supported inside the FPGA by reconfiguring the FPGA with appropriate bit streams.

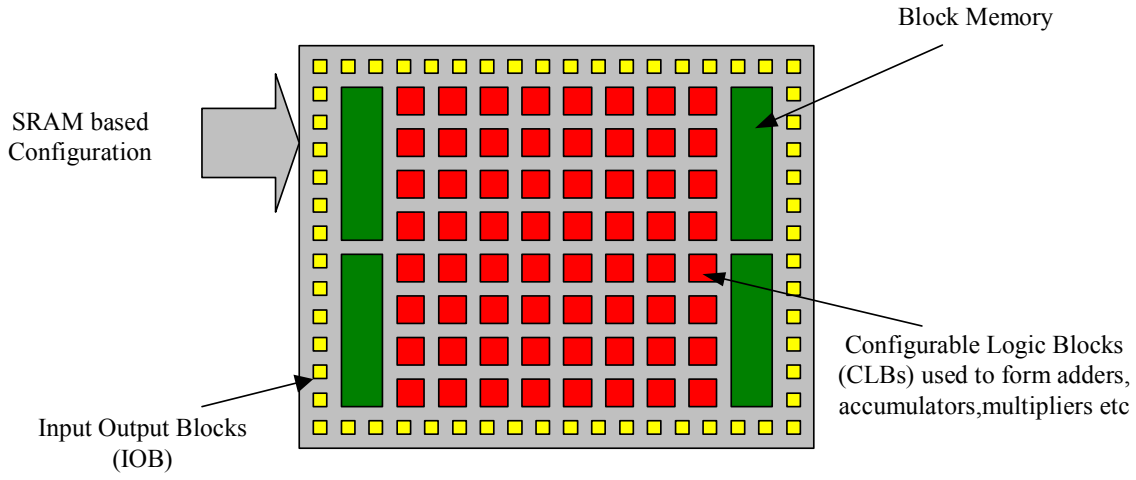

Figure 3-3 Structure of a FPGA

The Xilinx Virtex-II series: As a specific example, the Xilinx [2] Virtex-II family of FPGAs is considered. The basic logic element in this FPGA is a slice; it consists of

- Two 4-input look-up tables (LUTs)
- Two flip-flops
- Several multiplexers
- Additional silicon to support other applications

Four of these slices form a configurable logic block (CLBs). Some important features of the Xilinx Virtex-II FPGA are

- Largest FPGA with up to 8 million equivalent system gates
- Up to 3 Mbits of embedded BlockRAM memory
- Up to 168 18*18 multipliers
- Up to 12 DCMs (16 global clocks)
- Up to 1108 user I/O pins
- 64 to 16224 CLBs

3.3.3 Memory Access and Parallel Processing

The Analog to Digital (A/D) sample rates for WLAN protocols are on the order of tens of MHz, with 44 MHz of frequency used for 802.11b and 20 MHz for 802.11a and 802.11g. In a DSP system, the processor uses shared resources and the processor core takes interrupts from the other parts of the system. As the sample rates are in the range of 20-44 MHz, it becomes very hard for the DSP to transfer data without any loss. FPGAs have dedicated logic for transmitting and receiving data, which makes it easy to maintain

these high rates of I/O. With the advantages of minimal signal loading and with no overhead for instruction fetching or data fetching, FPGAs become a more suitable choice in high bandwidth and high data rate systems.

In DSP, large data sets can be used as it is optimized for use of external memory. FPGAs traditionally have limited amounts of internal memory, which makes smaller data sets ideal. The current FPGAs have overcome this obstacle with more than 3 Mbit on-chip memory. Current DSP chips have multiple arithmetic units for performing arithmetic operations. They include several MAC units and, up to a point parallelism can be used to increase the performance of MAC operations. The degree of concurrency is limited by the total number of MACs in the DSP chip and thus limits the full exploitation of potential concurrency in the DSP algorithm. At the cost of lower precision in calculation, the effective number of MAC units can be increased.

FPGA hardware allows allocating separate resources for fully exploiting the concurrency in the signal processing algorithms. A 32-tap FIR filter has to perform 32 multiplications followed by an addition for each output of the filter. An FPGA can be configured to perform all of these operations in parallel using dedicated multipliers and then add their product.

3.3.4 Reconfigurability and Reuse

DSP enables branching of the program by which switches can be easily implemented. In an FPGA, each configuration requires dedicated resources. The advantage of the FPGA lies in the fact that each configuration has dedicated resources, which makes parallel implementations possible

Re-use of processing units is supported by DSP, where a multiplier that is used in implementation of an FIR filter can be reused in a FFT routine. In FPGAs reuse of processing units is not permitted, as the resources are not shared between different routines. In the Virtex II FPGA, this obstacle is overcome by having a large number of multipliers on the FPGA.

The digital components found in a communication modem include DSP chips, RISC microprocessors, microcontrollers, RAMs and communication chipsets (such as PCI and Ethernet interfaces) that can be completely reprogrammed in software. FPGAs allow design of modems in which the majority of the system can be reconfigured remotely both in software and hardware. A DSP program has to take advantage of the architecture of the specific DSP processor, where only the program can be reconfigured. FPGAs with short turn around times allow custom architectures to be made for different applications.

3.3.5 Implementation of DSP Algorithms

Standard DSP algorithms are generally implemented in C, which can be directly used on a DSP processor, leading to lower implementation time. Standard C programs should be converted to a hardware description language (HDL) to implement them on FPGA. Although tools are available that convert a standard C program and map it to a hardware description language, it is a tougher migration path. The migration path is easy, however, when translating a high-level block diagram-based system directly to a Hardware Description Language (HDL) and implementing it on FPGA.

3.3.5.1 Digital Down Converter

Digital Down Conversion (DDC) is a fundamental part of the communication system. Digital radio receivers have fast A/D converters delivering vast amount of data, but in many cases the signal of interest is a small portion of that bandwidth. DDC allows the rest of the data to be discarded, thus allowing more intensive processing to be performed on the signal of interest.

3.3.5.2 Implementing MAC based FIR filter

The FIR filter is one of the basic building blocks common in nearly all DSP systems. The output sample stream is generated by convolving the input sample stream with N filter coefficients. In equation 3.2, N multiplications and N-1 additions are required to compute each value in the output stream Y*(n)*..

The operation of an FIR filter is described by equation 3.2,

$$Y(n) = \sum_{i=0}^{i=N-1} k(i)S(n-1) \tag{3.2}$$

The filter coefficients *k(i)* determine the frequency response of the system. In applications that require a large number of filter coefficients, the arithmetic load would be substantial. Figure 3.3 shows that the same function can be realized as a set of delay elements and multipliers with one delay and one multiplier for each tap of the filter followed by a summation function.

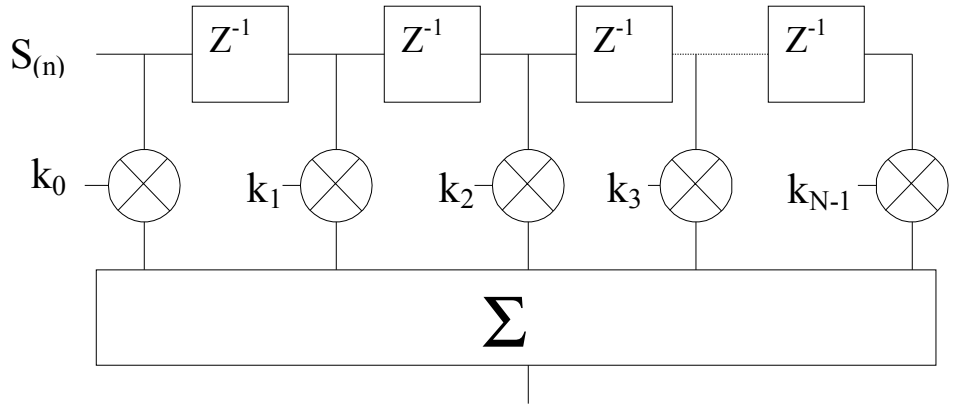

Figure 3-4 FIR filter in graphical form

The basic operation of the FIR filter equation is "multiply and accumulate" (MAC). It requires a multiplier that gives the product of the two operands presented at its input and an accumulator to accumulate the data samples. The main requirement is to provide operands (data samples and coefficients) to the multiplier input. Coefficients and data samples are stored in memory, and address generation units select the data and coefficient pairs to be presented to the multiplier. The memory should be able to supply two operands to the multiplier in a single clock cycle, which requires two separate memory banks or a dual-port memory. The MAC based filter operates at a rate of N times the sample rate. As the sample rate increases, the product of N and sample rate becomes very large. If a single MAC engine has to support an 8-tap filter at 100 MHz, it would require a MAC engine rate of 800 MHz and memory bandwidth of 1.6 GHz. A solution would be to split the number of taps (N) between a collection (called a "farm") of MAC engines. If the above filter were split into 8 different MAC engines, then each MAC engine rate would be 100 MHz. The 'DSP farm' is a term used to describe an array

of DSP processors on a PCB, each of which perform a part of a bigger function. Although the MAC rate is divided, the memory bandwidth requirement remains the same. In each clock cycle the filter is performing 8 multiplications, which requires 16 words. If the memory words are also split among blocks of memory using distributed RAM, it would relax the requirements on memory bandwidth.

As sample rates and the number of taps on the filter increase, more calculation units and more memory accesses would be required. This leads to more individual memories of smaller capacity. By continuously dividing the number of taps between more and more MAC engines, a fully parallel filter structure can be achieved.

The Virtex-II FPGA has dedicated 18-bit * 18-bit multiplier blocks that can be used in the MAC-based FIR filter. Higher order multipliers can be implemented using the logic slices. There is 3 Mb of dual-port RAM in 18 kbit block select RAM resources and 1.5 Mb of distributed RAM which can be used as storage for the N filter coefficients and N data samples. The configurable nature of the FPGAs would permit a designer to exploit the high degree of potential parallelism in many DSP algorithms to achieve even higher performances.

3.3.6 Availability of Libraries and IP Cores

DSP manufacturers provide extensive libraries for many of the building blocks and signal processing algorithms. A growing number of DSP algorithms are now available in the form of *Intellectual Property* (IP) also known as *cores.* These IP modules are being made available for FPGAs as predefined modules (whose parameters can be tuned as necessary). A variety of standard DSP algorithms for filters, correlators,

transforms, memories and math functions can be used to support a wide variety of applications including communication and signal processing. These cores make it much easier for DSP system designers to use FPGA technology and reduce the implementation time.

3.3.7. Architectures for Implementing DSP Algorithms in FPGA

The DSP inspired architecture is not the optimal architecture to implement the FIR filter (or other DSP algorithms) in an FPGA. General-purpose DSP chips are designed with architectures to perform reasonably well over a wide range of applications. Novel signal processing algorithms cannot be optimally supported with these general-purpose DSP chips. FPGA hardware is configured with internal SRAM memory through which it is easy to configure the FPGA to support a wide range of applications. This allows designing custom architectures for particular applications.

Optimization of DSP computational hardware and optimal implementation of various application-specific DSP algorithms in FPGA have been studied over the past decade. One approach to implement FIR filters is distributed arithmetic. A generic model of distributed arithmetic feature is shown in Figure 3.4. The calculations in distributed arithmetic are implemented using table look-ups, additions and subtractions. Filter throughput is no longer dependent on filter length when distributed arithmetic is used. For given sample precision the processing rate remains constant and is independent of filter length. Distributed arithmetic involves the computation of many partial products in parallel, and the summation of all these partial products, to compute the final product.

However, storage must be provided to store all of the multiplication tables used to accomplish partial product multiplication.

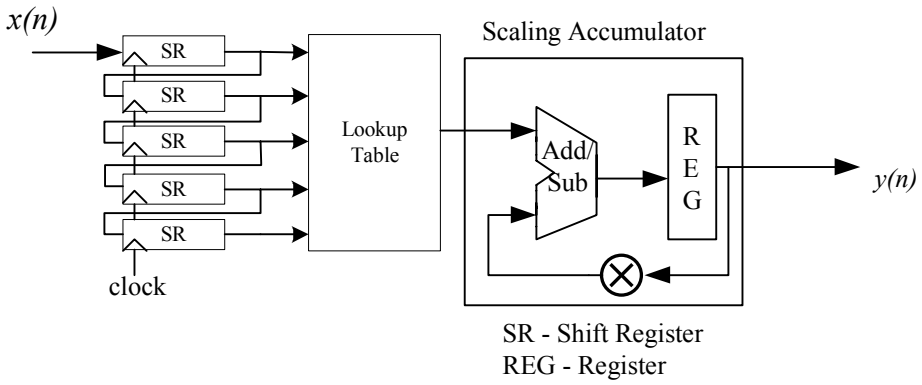

Figure 3-5 Distributed arithmetic filter mechanism

A shift register can be used to shift the multiple-bit input value into the FPGA, one bit at a time. With an input variable of N bits length, N cycles are needed to complete a single inner product calculation. Speed is improved by splitting the input word into sub-words, and shifting each of these sub-words into FPGA in parallel.

3.3.7.1 Fast Fourier Transform (FFT)

The data bits in OFDM are mapped to complex signal vectors using BPSK, QPSK or QAM modulation, and these vectors modulate sub-carriers using 64-point complex Fast Fourier Transform (FFT). The FFT is computationally the most demanding block of the transmitter and receiver design in the OFDM radio. State of the art DSP chips can perform the 64-point FFT in 60 μs. Additional time is required to perform bit-reversal operation if required. Using standard radix-4 Cooley-Tukey algorithm and using parallelism, the same 64-point FFT can be performed in a Virtex-II FPGA in less than 4 μs. This includes the bit reversal operation. The IEEE 802.11a standard defines the

FFT/IFFT period to be 3.2us, which is very difficult with the DSP technology available (at the time the project was started). Xilinx provides an FFT core, which is optimized specifically for the Virtex-II FPGA and is guaranteed to run speeds up to 100 MHz. The Xilinx FFT core provides a result vector every 192 clock cycles, which is equivalent to 3.2 μs at a 60 MHz clock speed . The Xilinx FFT core also eliminates the need to design the 64-point complex FFT. In order to reach the real-time speed requirement of 60 MHz for the IEEE 802.11a baseband, it is absolutely necessary to use a highly pipelined architecture as much as possible. This implies that all the arithmetic operations are to be performed in parallel, which is quite difficult using a DSP .

The I/O bandwidth is another limiting factor in a DSP chip. In an FPGA, with the exception of some dedicated pins, most of the pins can be used for I/O, which provides the extremely high I/O bandwidth required for FPGA-based signal processing designs. The large number of I/O pins and the multiple banks of internal memory can be used to employ a high degree of parallelism in an FPGA design. Separate memory banks are used for input/output and working memory in commercially available FFT cores. This permits concurrent I/O operation and FFT computation.

3.4. Summary

In this thesis we chose the FPGA as the target hardware for the baseband prototype based on the speed, size and reconfigurability requirements of the baseband design. A benchmark for the performance of a hardware solution is the number of operations per second it can support. To support the 54 Mbit/s rate specified by the IEEE802.11a

standard, the baseband design requires at least 15 GOPS (giga operations per second), which is almost 10 times faster than the fastest Texas Instruments DSP on the market (C64X) which advertises up to 1.6 GOPS. On the other hand, current FPGAs can easily support up to 20 GOPS due to the parallel processing nature of an FPGA. This makes FPGAs the logical choice for prototyping the IEEE 802.11a baseband design. The Xilinx Virtex II FPGA [3] was chosen as the target hardware as it is one of the most advanced FPGAs on the market supporting large designs of around 6 million gates. Using the large Virtex II FPGAs minimizes the partitioning of a design, thereby reducing the complexity and greatly increasing the overall speed. The Xilinx Virtex II FPGA is also well-supported by the Nallatech Extreme DSP system [3] which was chosen as the prototype platform in the design flow. The Extreme DSP platform enables real-time functional verification of a baseband design with system operating frequencies that exceed 100 MHz. The Extreme DSP prototyping platform supports the Virtex II FPGA and also has two ADC channels and two DAC channels. Extreme DSP also supports verification with the HP 16500C Logic Analyzer, which allows the capture of data from the baseband design running on Extreme DSP to compare to simulation results.

Chapter IV

Simulation and Prototyping Methodologies

4.1. Simulation and Prototyping

The purpose of simulation and prototyping is to develop and refine new ideas for communication. The simulation environment offers the designer a flexible and powerful environment on the desktop computer. In simulation, communication system parameters like signal to noise ratio (SNR), modulation types and other modeling parameters can be clearly specified and easily changed. The designer has more freedom in exploring the design space as the simulation environment allows design of algorithms without the constraints of real-time execution. In contrast, the prototyping environment connects the design to the real world. Test data is presented to the system from an uncontrolled environment using hardware interfaces such as analog to digital converters which present data of fixed width. The designer is restricted with limited hardware resources and the timing and power consumption requirements.

These two are rarely used together in the development process. Instead the design is developed as a two-step process. As a first step, new algorithms are developed based on simulation results. The description of the algorithm is used to develop a prototype in the second step. Simulation results cannot be directly used in the prototyping process, and data gathered from testing the prototype cannot be directly used in simulation.

The computational models assumed by simulators and by prototyping environments explain the difference between the two approaches. In a simulation environment the host computer's CPU provides all resources for performing signal processing. Most of the simulation languages and tools isolate the host-based simulation environment from the DSP-based execution environment. Real-time data acquired by the prototype hardware cannot be easily passed back to the host for analysis; likewise, simulated data generated on the host cannot be processed on the prototype.

In this thesis we have reviewed different approaches to the rapid prototyping of communication protocols and propose a data flow optimized for SDR-based prototyping of Wireless LAN protocols.

4.2 Simulation and Prototyping Environments

Simulation and prototyping of WLAN systems involves development and integration of several computationally intensive algorithms to enable different features required by these systems. The designer is faced with two important problems. First, simulation of communication systems involves block diagrams and mathematical equations while prototyping hardware is programmed in C, assembly or HDL. Second, simulations often run on a host computer, while prototypes run on hardware and the powerful features of simulation cannot be combined with the real-time constraints of the prototype hardware.

Each algorithm used in the simulation has to be tested independently before being integrated into a communication system. Interconnection of different algorithm blocks must be tested to ensure proper operation with neighboring blocks. The resulting block

diagram must be translated into a program suitable to execute on the prototype hardware. Mathematical equations are used in algorithm creation while block diagrams and simulation is preferred for system design. A digital signal processor is typically used in prototyping communication systems which require assembly language or C programming language to generate an executable routine. Both simulation and prototyping are inherent in communication system design, with initial design entry done on the host and final testing of the design is done on the prototyping hardware.

In a block-based system level design, each block is represented by an equation, which specifies the algorithm implemented by that block. The proper operation of individual blocks and the entire system is verified by simulation. The system is then translated into C or HDL and compiled to run on the DSP or FPGA.

The languages and design tools available today are largely incompatible with each other and are usually unable to execute both on the host and the prototype. Matlab, Simulink, C and hardware description languages interoperate poorly, and run either on a host or on a prototype. Matlab is the preferred programming language for algorithm designers, and is well suited to describe equations. Though add-on software packages exist that transform Matlab code to C code to run on a DSP, they are not very efficient. Though Simulink is the preferred choice for block diagram entry, it only runs on the host platform. C code written for a DSP typically uses DSP-only libraries, preventing it from executing on the host. Integrating C code or VHDL code with Matlab or Simulink is a very difficult task, and requires knowledge of the C-MEX interface or the Simulink S function interface.

The monolingual nature of today's languages and tools limits the rapid prototyping of complex communication designs. Though there is some flexibility offered by Matlab and Simulink to use both of them in designing a communication system and generating equivalent C code or HDL code, the compilers have limited capability to understand the design and the design is far from optimal. Another problem is the isolation of the host-based environment from the prototype-based environment. This makes it difficult for prototype hardware to pass the acquired data to the host for further processing. Likewise, simulated data generated on the host cannot be passed to the prototype.

4.3. Prototype Design Flow

Each of the languages reviewed is uniquely suited for different parts of communication system design. Simulink is suited for top-level design while Matlab is suited for algorithms. HDL and C are suited very well for hardware implementations. However, the strength of these languages cannot be combined by writing different parts of design using the most appropriate language.

There are various approaches for translating the floating-point algorithms to fixed point models. The Matlab add-on package Real-Time Workshop gives the flexibility of using the Matlab and Simulink environments, and can generate equivalent C code to run on a DSP. Another package, System Generator, gives the flexibility to use the Simulink environment to describe components of a prototype and generates a equivalent HDL code from these designs. Though both packages provide libraries of functions to use in individual environments, limited functionality is available from these libraries.

Another approach is to use fixed-point C code models which offer the advantage of fast simulation speed, but have the disadvantage of not being directly synthesizable to hardware. Using fixed-point C code in a prototype design flow requires another design flow step and more resources to translate the C code into a synthesizable HDL The EDA (Electronic Design Automation) industry is currently focusing on standardizing a fixed-point subset of C code called System C which can be easily translated to a hardware description language in an effort to close the gap between system-level floating-point models and hardware. In comparison to standard HDLs, System C does not offer any significant advantages in terms of ease-of-use. Thus, fixed point C is not well suited to be used in the rapid prototyping design flow due to the additional step required to translate it to hardware description language.

Simulink offers a graphical entry tool called System Generator. In this thesis we explored using Simulink for rapid prototyping of the Wireless LAN protocols. The Xilinx block set within the System Generator provides a system-level, fixed-point functional simulation environment as well as the means to create VHDL code that can be synthesized to a Xilinx Virtex FPGA. This VHDL code can be simulated on any of the various HDL simulation tools on the market. Since Xilinx Block-set is a component of Simulink, mixed-mode simulations with both floating-point and fixed point blocks are also possible. The disadvantage of System Generator is that the VHDL code generated within this tool is not easily readable and is not intended to be modified. This is a problem when transferring the tool-generated VHDL code to a development group that does not use System Generator. Another problem is that most of the Xilinx Cores used in the design are modeled as “black boxes”. Inside these “black boxes” the behavior of the

actual Xilinx cores are modeled for simulation purpose only and are not expanded until the Xilinx "place and route" stage in the design flow. The System Generator simulations are bit and cycle-true simulations of the actual hardware implementation, so it is very important to accurately emulate the timing and latency of the core "black boxes". If the core behavior is not modeled correctly in System Generator, the resultant VHDL implementation will have timing errors thus causing unnecessary iterations through the design flow. Another problem was that this package was in its early stages of development and was not very efficient in using the available prototype resources. The developed executables and the VHDL code have very little ability to communicate with the original simulation.

Another common approach for doing fixed-point modeling is to directly hand-code the HDL based on the floating-point algorithms and fixed-point algorithms developed in simulation. This approach has the advantage of creating optimized HDL that can be directly synthesized to hardware; however, it requires good knowledge of the system level design using Matlab and Simulink and the chosen HDL language, as well as the resources to code and simulate the HDL. Since creating HDL is a part of any prototype design flow which targets either an FPGA (Field Programmable Gate Array) or ASIC (Application Specific Integrated Circuit), the issue is to create HDL from the floating point design.

4.4. Summary

Today's traditional design methodology suffers from segregation. Each of the tools explained above have strengths, but these tools cannot be combined to avoid the

weaknesses inherent in each tool. An idea captured as a rough sketch of a block diagram, with equations noted beside each block, cannot be entered into a computer; the block diagram tool is incompatible with the equation tool. When debugging the system, particularly on hardware, the best analysis and debug tools available on the host must be abandoned for less usable hardware-specific tools designed for the hardware.

This thesis addresses these problem areas by unifying different classes of tools, exploiting both block diagram design and equation entry to serve as a prototype for the final implementation. An easy path is described to migrate from equations coded in Matlab, which cannot be executed on a hardware platform, to synthesizable code written in VHDL that can be directly mapped into an FPGA. The system level design was carried out using floating-point algorithms in Matlab, and fixed-point simulations are done in Matlab where required. Each of the blocks is then coded in VHDL and the functionality of each block is verified with its Matlab counterpart. This development environment, where both simulation and prototyping environments are used enabled the implementation of the final design.

Chapter V

HRDSSS System Design

The High Rate Direct Sequence Spread Spectrum (HRDSSS) modem function and design is explained in this section. The system is designed as a direct sequence spread spectrum phase shift keying modulator/demodulator.

5.1. Transmitter Design

The transmitter is designed as a direct sequence spread spectrum phase shift keying modulator. The system uses Differential Binary Phase Shift Keying (DBPSK) for 1 Mbit/s, Differential Quaternary Phase Shift Keying (DQPSK) for 2 Mbit/s and Complementary Code Keying (CCK) for 5.5 and 11 Mbit/s data rates. The different data rates and the associated symbol rates are shown below in Table 5.1.

Data Modulation	Data Rate (Mbit/s)	Symbol Rate (MSPS)
DBPSK	1	1
DQPSK	2	2
CCK	5.5	1.375
CCK	11	1.375

Table 5.1 Data rates and symbol rates for HRDSSS system

5.1.1. Spread Spectrum Modulator

The 11 chip Barker sequence {+1, -1, +1, +1, -1, +1, +1, +1, -1, -1, -1} is used in IEEE 802.11 as the spreading sequence. The preamble is always transmitted as a DBPSK waveform and the header can be configured to be either DBPSK or DQPSK. The data

packet modulation depends on the choice of data rate. The preamble is used for synchronization while the header contains information necessary to demodulate and decode the message portion of the packet.

For the 1 Mbit/s data rate, and for the header at all rates, the data encoder implements DBPSK coding by differentially encoding the serial data from the scrambler and driving the I and Q output channels together. For the 2 Mbit/s data rate, the data encoder implements DQPSK coding. This coding uses the differential coding of *dibits* (bit pairs). The serial data is formed into *dibits* in the differential encoder as described above. One of the bits from the differential encoder goes to the I channel and the other to the Q Channel. The I and Q channels are then both multiplied with the 11-bit Barker word at the spreading chip rate. This forms QPSK modulation at the symbol rate with BPSK modulation at the chip rate.

5.1.1.1. CCK Modulation

The spreading code length is 8 and is based on complementary codes. The chip rate is 11 Mchip/s and the symbol duration is exactly 8 complex chips long. The following formula is used to derive the CCK code words that are used for spreading at both 5.5 and 11 Mbit/s.

$$c = \{e^{j(\phi 1+\phi 2+\phi 3+\phi 4)}, e^{j(\phi 1+\phi 3+\phi 4)}, e^{j(\phi 1+\phi 2+\phi 4)}, -e^{j(\phi 1+\phi 4)}, e^{j(\phi 1+\phi 2+\phi 3)}, e^{j(\phi 1+\phi 3)}, -e^{j(\phi 1+\phi 2)}, e^{j\phi 1}\}$$

(LSB to MSB), where c is the code word. The terms: $\phi 1, \phi 2, \phi 3, \phi 4$ are defined below for 5.5 Mbit/s and 11 Mbit/s [1], [14].

This formula creates 8 complex chips (LSB to MSB) that are transmitted LSB first. The coding is a form of the generalized Hadamard transform coding where $\phi 1$ is

added to all code chips, $\phi 2$ is added to all odd code chips, $\phi 3$ is added to all odd pairs of code chips and $\phi 4$ is added to all odd quads of code chips.

For the 5.5 Mbit/s and 11 Mbit/s rates, the data is partitioned into nibbles (4 bits) or octets (8 bits), and one of the 16 possible CCK spread sequences is selected for 5.5 Mbit/s and one of the 256 possible CCK spread sequences is selected for 11 Mbit/s.

5.2. Spread Spectrum Receiver Design

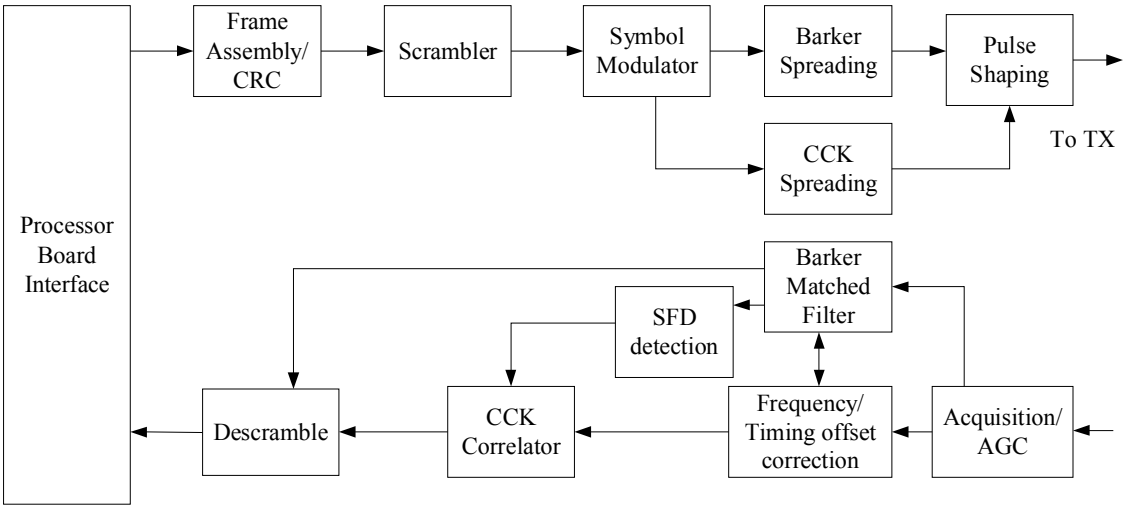

Figure 5-1 Block diagram of DSSS modem

5.2.1. Acquisition and ADC

The acquisition process comprises Energy Detect (ED) followed by a Barker sequence correlation. When energy above a threshold is present at the input of the receiver, the receiver declares “energy active”. This threshold is determined as -76 dBm based on the 802.11b receiver sensitivity specifications. The acquisition signals can also be used in a Carrier Sense (CS) mechanism. After energy detection, the correlator (Barker Matched Filter in figure 5-1) accumulates the Barker correlation over a symbol integration length for each of the 44 possible chip*4 (chip times 4) offsets. The highest

correlation peak is used to establish the correct symbol timing. Peak energy detection is used to determine the AGC scaling factor to use prior to the channel matched filter.

$$\text{Peak detected energy for M samples} = \sum_{i=0}^{M-1} I^2 + Q^2 \qquad (5.1)$$

Successive correlation peaks observed at 11*over (11 times) sample ratio, give the timing offset estimation and is used to correct symbol timing errors. Once signal acquisition is declared and symbol timing is achieved, the demodulator starts demodulating the data. The frame detection flag is set by comparing the demodulated header data to the Signal Frame Delimiter (SFD) for either long preamble mode or the short preamble mode.

5.2.2. Barker Matched Filter

There are two types of correlators in the spread spectrum receiver. The Barker matched filter performs correlation for the Barker sequence used in the preamble, header and the 1 Mbit/s and 2 Mbit/s data modes. The correlator is used in de-spreading to accumulate the chips. The Barker matched filter correlates the received chip sequence with the known Barker sequence. This operation can efficiently be done using an FIR filter structure. The input sample rate of the receiver is 4 times the Barker chip rate (44 MHz). The Barker matched filter calculates the correlation values across samples exactly 1 chip apart. Since the sample rate is four times the chip rate, it is necessary to also calculate the correlations across sample offset by 1/4 chip multiples. This results in 44 complex correlation values that are spaced 1/4 chip apart that enable timing synchronization to within ¼ chip.

5.2.3. Frequency and Phase tracking

The frequency and phase tracking block tracks the frequency and phase error and corrects it. The frequency offset estimation is modeled as a maximum likelihood (ML) phase estimator [7]. The signal is demodulated from the sign of the correlation peaks tracked by the Barker matched filter. The phase estimator calculates the phase error between the received signal samples and the expected signal samples. The phase error slope averaged over the received symbols gives the necessary frequency offset information.

5.2.3.1. Data Demodulation

The rest of the data demodulator and decoder use implementations developed by Richard Lynch at the University of New Hampshire's InterOperability Laboratory. A description of the spread spectrum demodulator is provided below.

To demodulate the data at 1 Mbit/s and 2 Mbit/s, the Barker matched filter method described in section 5.2.2 is used. The received signal samples are correlated with the known Barker sequence and the data bits are determined based on the highest correlation peak. For the 5.5 Mbit/s and the 11 Mbit/s modes, the data is demodulated using a CCK correlator. Once the header is demodulated, the phase of the last bit of the header is captured and used as a phase reference for the high rate data portion of the packet. The data is demodulated and decoded per symbol. In 5.5 Mbit/s mode, a bank of correlators compare the received data to the 16 possible CCK chip sequences. The biggest correlation of the above comparison demodulates 2 bits and 2 more bits are demodulated by the QPSK demodulation. For 11 Mbit/s mode, the input samples are processed in a bank of 256 correlators, and the biggest correlation peak is used to

demodulate 6 bits, while the last two bits are demodulated by the QPSK differential demodulation of the output.

Chapter VI

OFDM System Design

6.1. Introduction to OFDM system

6.1.1. OFDM Signal Representation

In an OFDM system, data is carried on multiple sub-carriers. The modulation of sub-carriers is done directly in the frequency domain using complex multiplication, the resulting data are transformed into the time-domain using the IFFT at the transmitter and transformed back to frequency-domain using the FFT at the receiver. The number of points of the IFFT/FFT used in a system depends on the number of sub-carriers used. In 802.11 Wireless LAN systems using the OFDM physical layer, the number of sub-carriers used is 64, which translates to using a 64-point IFFT/FFT.

The discrete-time representation of the signal using N sub-carriers, is given in equation 6.1,

$$x(n) = \frac{1}{\sqrt{N}} \sum_{k=-N/2}^{N/2-1} X(k) e^{j2\pi \frac{k}{N} n} \tag{6.1}$$

where *X(k)* is the complex modulation vector and $n \in [-N/2, N/2]$. At the receiver side, the data is recovered by performing an FFT on the received signal, i.e.

$$X(k) = \frac{1}{\sqrt{N}} \sum_{n=-N/2}^{N/2-1} x(n) e^{-j2\pi \frac{n}{N} k} \tag{6.2}$$

where $k \in [-N/2, N/2]$.

6.1.2. Cyclic Prefix

The cyclic prefix is a crucial feature of OFDM used to combat the inter-symbol-interference (ISI) and inter-channel-interference (ICI) introduced by the multi-path channel through which the signal is propagated. Shifting the time by T_{GUARD} creates a cyclic prefix, which means replicating part of the OFDM time-domain waveform from the back to the front. The duration of the guard period T_{GUARD} is chosen to be longer than the worst-case delay spread of the target multi-path environment. Three kinds of guard periods, for short training sequence (= 0 μs), for long training sequence(= T_{GI2}) and for data OFDM symbols (= T_{GI}) are defined in the IEEE 802.11 Wireless LAN standard. Once the condition that the worst case delay spread is less than T_{GUARD} is met, there is no ISI since the previous symbol will only have effect over samples within [0, τ_{max}] where τ_{max} indicates the worst case delay spread.

6.2. OFDM Frame Structure

The transmitted baseband symbol is composed of contributions of several OFDM symbols:

$r_{PACKET} = r_{PREAMBLE}(t) + r_{SIGNAL}(t-t_{SIGNAL}) + r_{DATA}(t-t_{DATA})$, where t_{SIGNAL} is equal to 16 μs and t_{DATA} is equal to 20 μs .Figure 5.1 shows the OFDM frame format. The frame contains a PLCP Preamble which is used for modem training, SIGNAL field which conveys the parameters (modulation, number of octets) of the following data packet and a DATA portion which contains a variable number of OFDM symbols.

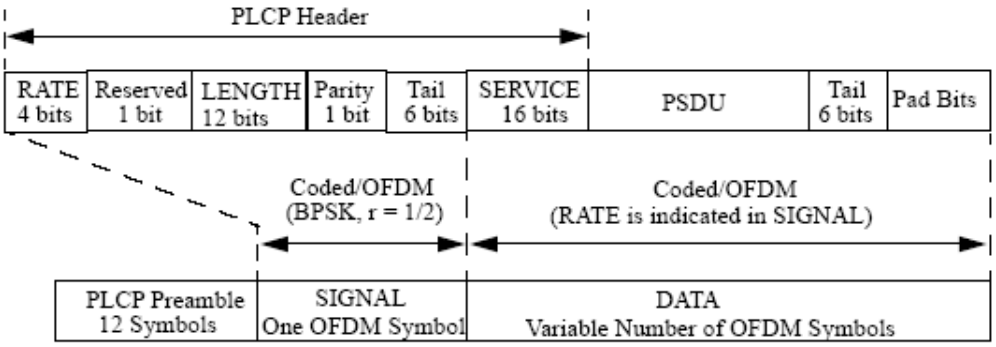

Figure 6-1 OFDM frame format

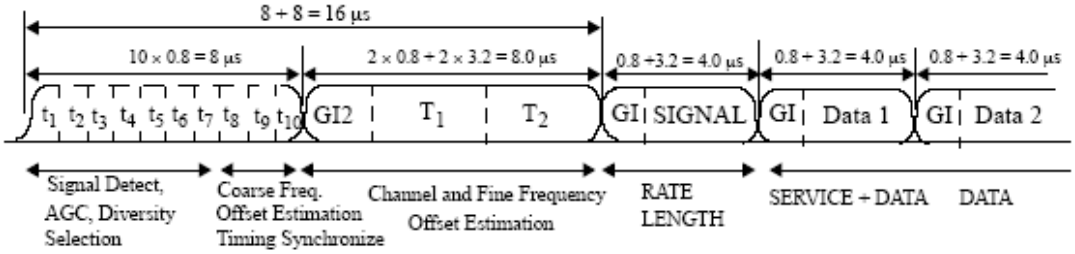

Figure 6-2 OFDM training sequence structure

Figure 5.2 shows the OFDM training sequence structure (PLCP preamble), where t_1 to t_{10} denote short training symbols and T_1 and T_2 are long training symbols. The PLCP preamble is followed by SIGNAL and DATA. GI indicates the Guard Interval for the OFDM data symbols. The first 10 symbols are used for AGC, frame detection, and coarse frequency offset estimation. Each of the symbols is 16 samples long, or equivalently 0.8 μs. The long training symbols are used for fine frequency offset estimation and also for channel coefficient estimation.

6.3. OFDM Transmitter Description

The encoding of a data packet into an OFDM signals is as follows:

1. Generate the short training sequence and long training sequence
2. Generate the SIGNAL field bits, coding and interleaving SIGNAL field bits, and map them into frequency domain, insert pilots and transform into time domain.
3. Prepend the SERVICE field, and add pad bits to the octet stream and form the DATA.
4. Scramble and encode the DATA using convolutional encoding and puncture to get higher rates, and map them into complex BPSK, QPSK, QAM16 or QAM64 symbols followed by pilot insertion.
5. Transform from frequency domain to time domain and add a cyclic prefix and concatenate the OFDM symbols into a single time-domain signal.

6.4. OFDM Receiver Design

The OFDM system design and simulation, based on the 802.11a OFDM physical layer specifications were developed by Kevin Karcz at University of New Hampshire's InterOperability Lab. The developed system has the ability to generate signals compliant with the 802.11 OFDM PHY specifications and also demodulate OFDM packets. However, the design was based on a block processing approach that assumes all the samples of the packet are available before demodulation commences. The system has been enhanced in this thesis work and new algorithms are used for better synchronization and frequency offset estimation and correction. The enhancements also enabled it to operate on-the-fly and mimic real hardware. The following sections detail the summary of modifications to the signal processing involved in the decoding of an OFDM signal

and the architectures developed in this thesis for enabling an SDR based implementation of the OFDM baseband and the algorithms used.

6.4.1. Synchronization

Synchronization in OFDM has been the topic of much research in recent years and numerous algorithms have been developed in this area. Synchronization in OFDM contains the following stages:

1. Frame detection
2. AGC adjustment
3. Coarse frequency offset estimation and correction
4. Symbol timing error estimation and correction
5. Fine frequency estimation and correction

6.4.1.1. Frame Detection

The preamble structure explained in section 5.4 is used for frame start synchronization. The periodicity of the short symbols, combined with energy detection, is used for frame detection and for frame start synchronization. The energy of the received signal is calculated and the signal is correlated with the known short symbols. The problem with correlation approach is that multi-path components are combined before decision is made. A new algorithm is used instead where two correlations are employed. In the second correlation step, the received signal is also correlated with itself with a delay of one short symbol. The autocorrelation of these known symbols creates peaks. Detection can be declared if the peaks exceed a threshold. The self-correlation of the

signal creates peaks at the length of the short symbols. The last peak is used as the position from where start of the next symbol is determined

6.4.1.2. AGC

AGC is run in parallel with the frame detection. The first 6 symbols of the 10 short symbols are used for AGC. A simple AGC is used where measurements over a window of samples is taken and the delay-multiply values are compared to the power measured. After the signal detection, these auto-correlations are averaged to get average signal power and the gain is determined to scale the input power to 1. This gain is used to scale all the samples afterwards within the same packet frame.

6.4.1.3. Frequency Offset Estimation

Due to the RF carrier frequency difference of the transmitter and receiver, each signal sample at time t contains an unknown phase factor $e^{j2\pi\Delta f_c t}$,where Δf_c is the unknown carrier frequency offset. This unknown phase factor must be estimated and compensated for each sample before the FFT operation at the receiver; otherwise, the orthogonality between sub-carriers is lost. The two long symbols following the ten short symbols in the OFDM training structure are used for frequency offset estimation. The two long symbols are essentially the same sequence repeated twice. The samples corresponding to the long symbols are correlated to estimate the frequency offset. To estimate $\rho = \Delta f_c t$, the frequency offset, the correlation sum J is calculated. Where,

$$J = \sum_{l=0}^{N-1} y(l)y^*(l+N) = e^{-j2\pi\rho} \sum_{l=0}^{N-1} \left| y(l) \right|^2 \tag{6.3}$$

Where, * denotes the complex conjugate operation.

So that we can estimate

$$\rho = \frac{1}{2\pi}\arg\left[\frac{J^*}{|J|}\right] \tag{6.4}$$

A coarse frequency offset estimation using short symbols is also performed. Correlating the adjacent short symbols we have,

$$K = \sum_{l=0}^{N/4-1} y(l)y^*(l+N/4) = e^{-j2\pi\rho/4}\sum_{l=0}^{N/4-1}\left|y(l)\right|^2 \tag{6.5}$$

so that

$$\frac{\rho}{4} = \frac{1}{2\pi}\arg\left[\frac{K^*}{|K|}\right] \tag{6.6}$$

since $\rho/4$ is less than 1 even at 100 ppm, there would be no ambiguity in determining the value of ρ. Since the metric K involves only 1/4th of the total samples in the symbol, the result is less accurate. In the proposed algorithm, the short preamble is used to obtain a rough estimation by accumulation and averaging over 16 samples and the long preamble is used to get the fine estimation by integrating across 64 samples. The combination of both frequency offset methods increases the robustness of the frequency estimation algorithm [7], [8]. Combining both coarse and fine estimations into the frequency offset estimation where we would get,

$$\rho = \left\lfloor \frac{4}{2\pi}\arg\left[\frac{K^*}{|K|}\right]\right\rfloor + \frac{1}{2\pi}\arg\left[\frac{J^*}{|J|}\right] \tag{6.7}$$

Where $\lfloor\ \rfloor$ indicates truncation towards zero. Research done by Jian Li, and Guoqing Liu [missing citation] has shown that using only the short OFDM symbols results in a 2.7 dB accuracy with respect to using only the long OFDM symbols. It has also been shown that

using both long and short OFDM symbols yields a performance gain of 1.9 dB compared to using only short symbols.

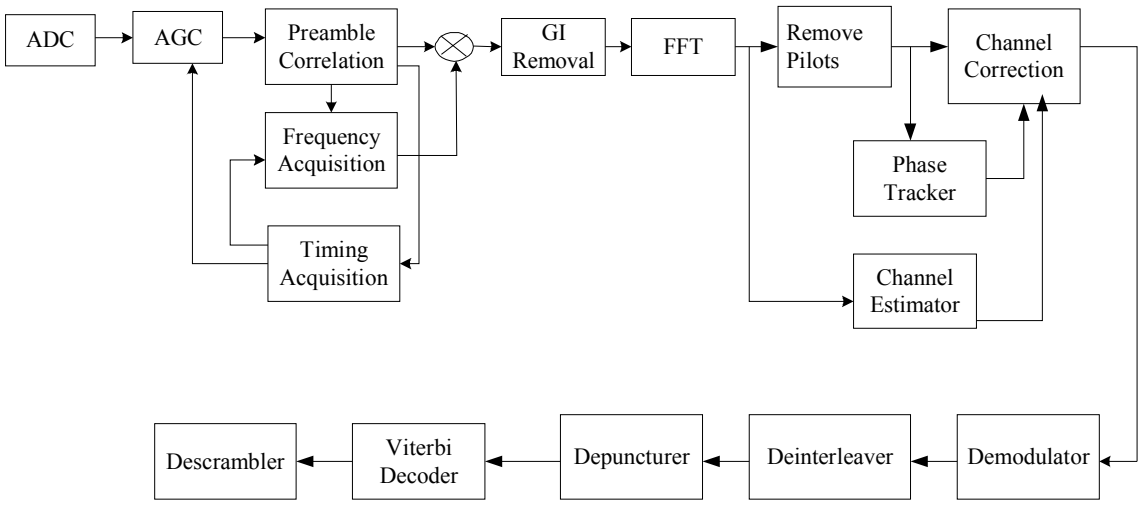

Figure 6-3 Architecture of OFDM synchronizer

6.4.1.4. Symbol Timing Estimation

Due to the difference in the sampling clocks between the transmitter and receiver, each signal sample is offset from its correct sampling time by a small amount which linearly increases with the index of the sample. The long preamble is followed by the SIGNAL symbol which contains information required to decode the data portion of the packet. Correct timing on the last symbol of the long training sequence is required, so the channel coefficient estimate from the header can be directly applied to demodulation of the PLCP header without any phase correction. Once the initial timing is acquired, the timing offset estimation is done in parallel with the frequency offset estimation. Due to the excellent autocorrelation properties of the preamble, the two correlation methods used in the detection process result in periodic strong peaks that enable the detection of the symbol boundary precisely. The timing offset estimation is verified by observing another

detection which would be 16 samples later. By using these periodic peaks, false alarm probability is reduced. By over-sampling the receive signal by a factor of two, better timing synchronization is achieved.

6.4.1.5. Channel Estimation

Channel estimation uses the two long symbols from the preamble, which are also used in frequency offset estimation. Once the frame start is detected, frequency offset is estimated, and the signal samples are offset compensated. They are transformed into the frequency domain by FFT. Each received sub-carrier is represented in frequency domain as,

$$Y(k) = H(k)\, X(k) + N(k) \tag{6.8}$$

and the channel H can be estimated as,

$$\hat{H}(k) = \frac{Y(k)}{X(k)} \tag{6.9}$$

but this will introduce noise enhancement, especially when $|H|$ is small.

6.4.1.6. Residual Frequency Offset Correction

The frequency offset correction is not perfect and the residual error tends to accumulate over samples. This effect is minor for short packets, but for large packets the accumulated error would be enough to cause orthogonality loss among sub-carriers. Four pilot sub-carriers provided in the data symbol are used for estimating the phase factor due to the residual error and for correction of the frequency offset.

6.4.2. Demodulation and Decoding

The frequency and timing offset corrected signal is then sent to the FFT block. The demodulator demodulates the corrected signal into the four different modulations i.e. either BPSK, QPSK, 16-QAM and 64-QAM. The demodulated message is then depunctured with inserting dummy zeros in the bit stream according to the designated puncturing scheme. The depunctured data is decoded using a Viterbi decoder which uses hard decisions to decode the convolutionally encoded bits**.** The rest of the system uses the blocks from original system, and the design allowing symbol-by-symbol based decoding.

6.5. Summary

The OFDM system design is explained. The original OFDM system designed by Kevin is enhanced to operate on a symbol by symbol basis. The various algorithms used in detection, synchronization, timing and frequency offset estimations are described. Enhanced algorithms are proposed to make the design more efficient. The enhanced system simulates a real hardware level implementation and provides easy migration from system-level design to hardware prototyping.

Chapter VII

Low Level Design

In this chapter the hardware architecture for a Software Defined Radio (SDR) based OFDM prototype and the design flow in implementing the prototype is presented. The system design was carried out in MATLAB and the floating-point algorithms used in the system were directly hand-coded into a synthesizable HDL. Since creating HDL code is a part of any prototype design flow which targets either an FPGA or ASIC, the issues are defining the architecture and creating the HDL code.

7.1. Design Requirements

There are several issues that needed to be addressed in the HDL design to achieve the goal of a real-time implementation for the IEEE 802.11a OFDM baseband processor. The system clock needs to run at least 60 MHz to perform real time signal processing due to the Xilinx FFT core performance limitations. The WLAN baseband design involves many arithmetic operations, typical to any DSP design. These arithmetic operations can slow down the final implementation. All the arithmetic blocks (adders, subtractors, multipliers, dividers) modeled in the system are combinatorial blocks where each operation is performed in 1 clock cycle. This results in a much slower implementation in FPGAs due to the number of operations that must be performed per clock cycle. A highly pipelined architecture in which the blocks have registered outputs as well as registers

between intermediate operations leads to a much faster implementation at the cost of a few clock cycles of latency depending on the structure of the operation. In order to meet the clock speed requirement of 60 MHz for the real-time OFDM baseband implementation, it is absolutely necessary to use a highly pipelined architecture as much as possible. To achieve this goal, most of the HDL blocks designed in the system were replaced with Xilinx core library math blocks, which are pipelined and optimized for the Virtex FPGA.

7.2. Design Flow

The design flow chosen to implement the OFDM prototype is shown in Figure 6.1. The design flow begins with floating-point modeling and simulation in MATLAB. The system level of implementation is described in chapter 5. The first version of the system was done as a block-based approach where all the processing is done on a packet after it is fully received. This was later changed to symbol level processing, where each symbol is processed "on the fly" in real-time. One of the important aspects of the design was to study the effects of fixed-point implementation of the FFT in VHDL. Fixed-point code to generate a 64-point FFT is developed and aimed at the next implementation in VHDL. Though both implementations gave similar results, the VHDL implementation of the FFT was replaced by the Xilinx FFT core, as it uses a highly pipelined architecture and meets the timing requirements imposed on the design. Fixed-point modeling and simulation of the hardware was done using the Xilinx block-set of the Simulink package. The advantages and disadvantages of using the Xilinx block set are explained in the simulation and prototyping methodologies chapter (Chapter 4) of this thesis. The next

step in the design was to develop synthesizable VHDL code to implement the design in the Xilinx Virtex II FPGA. Individual blocks of the transmitter were hand coded in VHDL and the entire system was synthesized. It was also verified that the design meets the timing requirements during functional simulation. The next step in the design flow would be to place and route the design on the target hardware using vendor specific tools and accomplish the working prototype of the design. Due to the non-availability of the Nalltech Extreme DSP platform, this functionality was not verified.

7.3. Prototype Implementation

7.3.1 Design Overview

The baseband transmitter design is a straightforward implementation of the OFDM system described in Chapter 5. However, a pipelined architecture is required to achieve the requirements of the hardware.

After the data is scrambled using the scrambler, the convolutional encoder operating at rate 1/2 generates 2 data bits for each input bit. Higher rates are achieved by using either rate 1/2, 2/3 or 3/4 puncturing patterns and removing bits from the encoded data stream. The data is then interleaved using a rectangular block interleaver, and then a matrix interleaver, based on the rate and number of data bits per transmitted OFDM symbol. The symbol mapper then maps the encoded bits to a BPSK, QPSK, 16-QAM, or 64-QAM constellation depending on the rate. The output of the symbol mapper is 48 complex values which comprise the data sample of a OFDM symbol. Pilot sub-carriers are added and the symbol is zero-padded before transforming the frequency domain data

using the 64-point inverse FFT. A cyclic prefix is added to the generated time domain signal and a time domain OFDM symbol is generated for transmission..

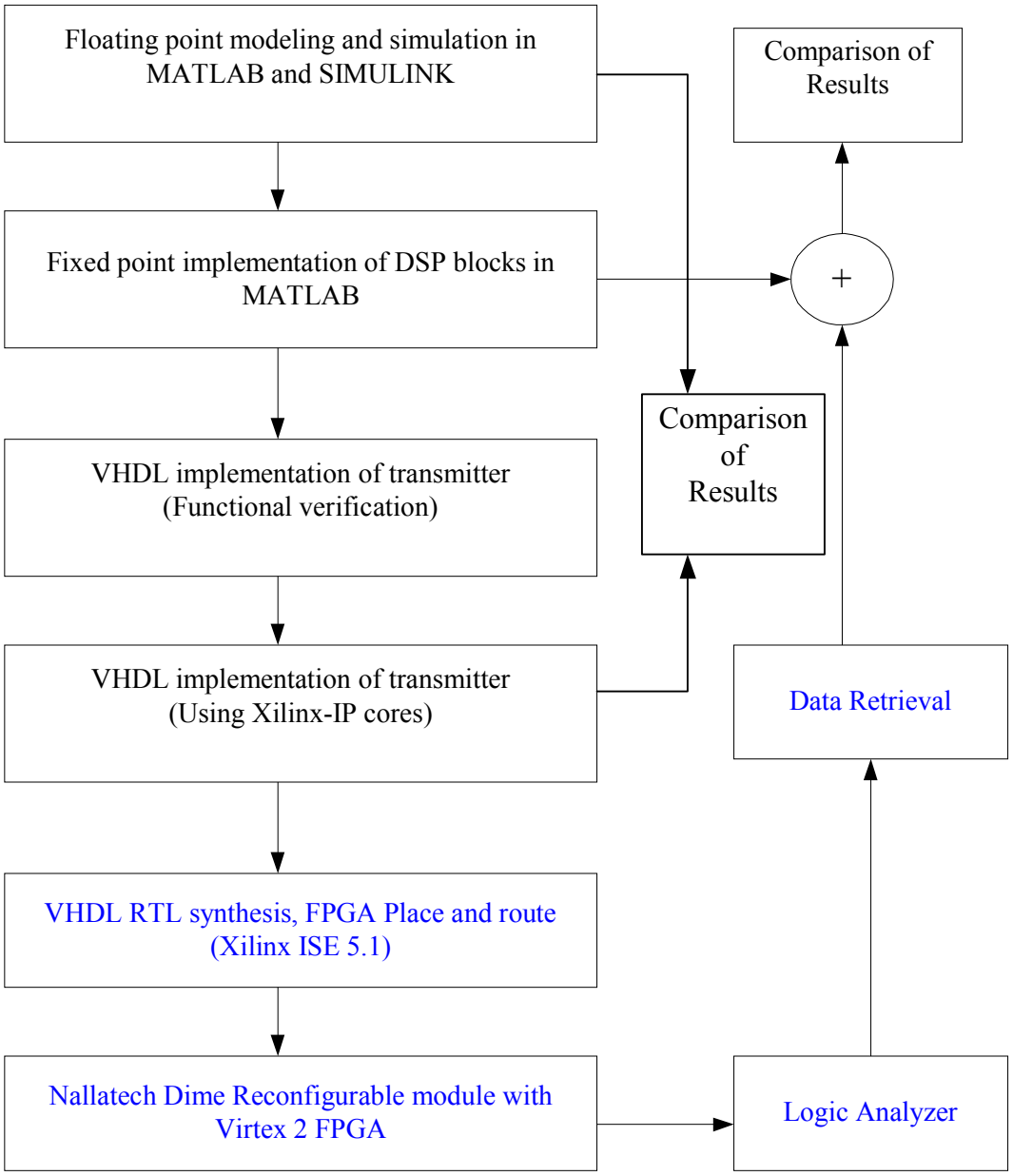

Figure 7-1 OFDM Prototype Design Flow

.

An important step is algorithm simplification. The trade-off between performance and computational complexity is crucial. Minor changes in performance often lead to a large reduction in hardware. It was found that using ROM-based look-up tables for preambles, encoder, interleaver and modulation mapping cuts the complexity of computing at the expense of increasing the memory requirement. The hardware size is dominated by memory and the FFT processor. A 16-bit fixed point format is used for the system in modulation mapping and FFT, as the Xilinx FFT core requires 16-bit fixed point data format. Using smaller bit widths would improve the performance of the receiver architecture, as it reduces the computational complexity in frame synchronization, correlation and offset estimation and correction.

7.3.2. Prototype Architecture

The hardware architecture for the OFDM prototype is shown in Figure 7-2. As shown in the figure a baseband controller generates all the control signals required to enable/disable the different blocks in the transmitter. A bit counter and symbol counter are used to aid the generation of control signals. The scrambler was modeled with shift registers and a general rate 1/2 convolutional encoder is used. A lookup table-based puncturing block was developed which uses a dual-port RAM. The dual-port RAMs are extensively used as lookup tables, and also to store data in intermediate stages. Frame generation begins with enabling the transmit controller with the data to be transmitted and specifying the rate. The preamble generation block stores the 10 short symbols and the 2 long symbols in the time domain, and is enabled once the TXSTART is initiated. The PLCP header generator block creates the SIGNAL symbol that carries information required to decode the data portion of the packet. The Signal Field encoder block uses

part of the hardware resources used by the Data Encoder block. Encoding of the PLCP header and data are done in parallel. The two interleavers (general block interleaver and matrix interleaver) are combined into one block, and the interleaver indices for various block sizes are pre- computed and are stored in the dual-port RAM. The Pilot generator block also runs in parallel with the Signal Encoder and Data Encoder and generates the pilot signals to be inserted into the OFDM symbols. The Arrange Data For FFT block arranges the data into the correct format for the FFT.

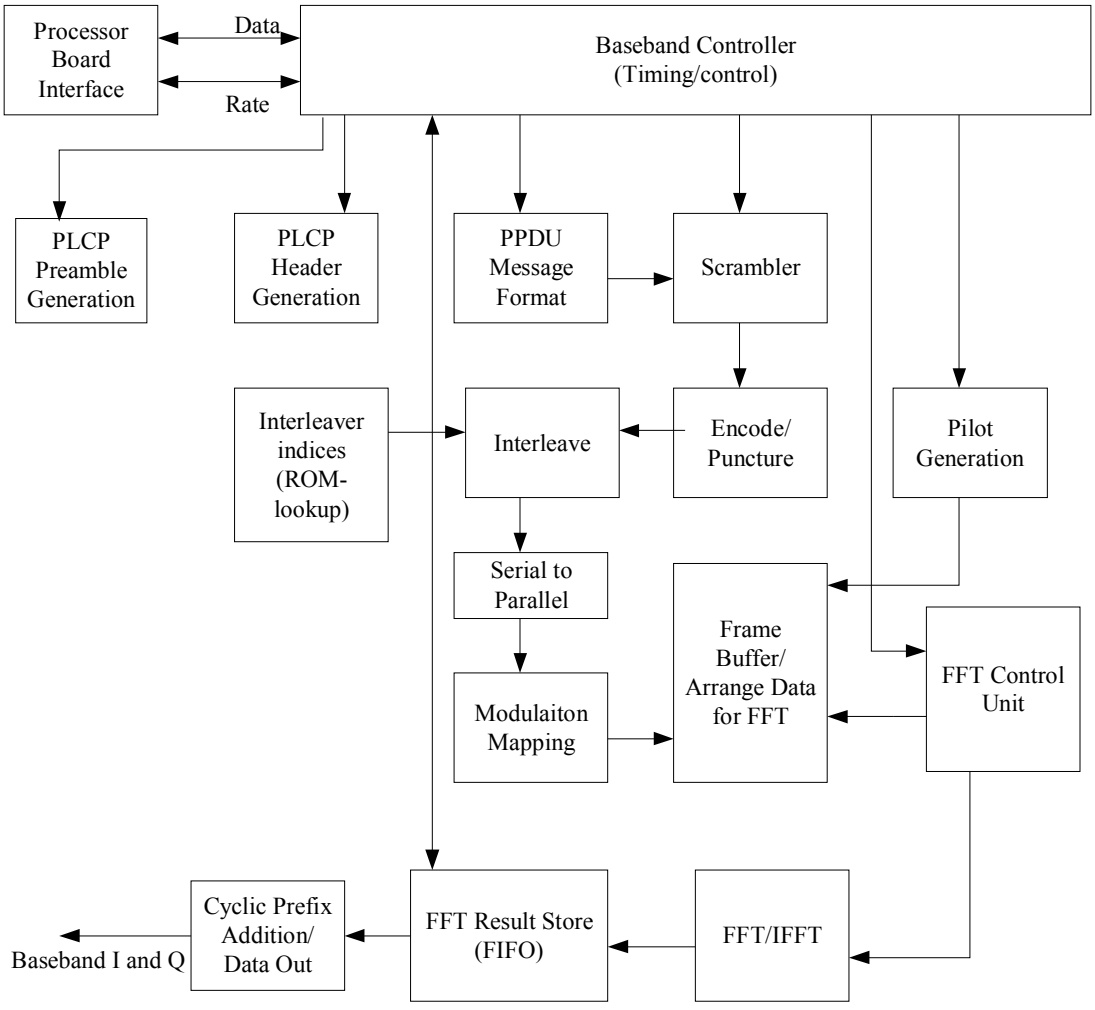

Figure 7-2 OFDM Transmitter Functional Block Diagram

The significant processing stage in the OFDM transmitter is the FFT block. This block is used both on the transmit and receive sides by inverting the FWD/REV signal in the FFT core. The FFT control block and the baseband controller block together generate the writing of data samples into the FFT data input memory and reading data out of the data output memory. The output of the FFT block is written to a FIFO, which contains the time domain OFDM signal to be transmitted. Two dual-port RAMs are used at each stage, one for real data and the other for imaginary data. The FIFOs are implemented with dual-port RAMs. The FFT controller also generates addressing for the output FIFOs. All the signal processing is done on the positive edge of the clock and registered outputs are used at each stage. The FFT core developed for the transmitter employs a Cooley-Tukey radix-2 decimation-in-time (DIT) FFT to compute the DFT of a complex sequence. In general, this algorithm requires the calculation of columns or ranks of radix-2 butterflies. These radix-2 butterflies are sometimes are referred to as dragonflies. For N=64 there are 4 dragonfly ranks, with each rank comprising 8 dragonflies. The input data for the core is a vector of 64 complex samples. The real and imaginary components of each sample are represented as 16-bit 2's complement numbers. The data is stored in on-chip dual-port block RAM. The complex output samples are defined with 16 bits of precision for each of the real and imaginary components. The FFT processor required 256 clock cycles to generate a block of output. The FFT processor in the final prototype was replaced with a Xilinx FFT core, which provides a netlist which is optimized specifically for the Virtex II FPGA and is guaranteed to run at 60 MHz. The Xilinx 64-point transform engine employs a Cooley-Tukey radix-4 decimation-in-frequency (DIF) FFT to compute the DFT of a complex sequence and provides output every 192 clock cycles,

which equates to 3.2 μs at 60 MHz. In terms of optimizing the baseband design for speed, pipelining using core blocks can contribute to a 3 to 4 times improvement in speed. Using a parallel processing architecture and the optimized Xilinx core library blocks directly contributed to reaching the goal of prototyping a real-time implementation of the IEEE802.11a OFDM baseband transmitter.

All the blocks of the transmitter side can be used in the receiver. The receiver would require additional Synchronization and Frequency offset correction blocks in addition to the transmitter blocks. Multiple transmitter chains in conjunction with a spatial processor can be used to model a MIMO OFDM baseband processor.

7.4. Testing and Verification

The functionality of each block of the OFDM transmitter system was individually tested and the outputs were compared with the outputs from the system level design. The FFT processor developed in this thesis, and the Xilinx FFT cores were tested and the outputs are found to agree with the developed fixed-point FFT code. However, further study is required to evaluate the necessary data word-widths and reduction in computational complexity associated with the FFT process. The functional testing of system as a whole was also carried out and the generated output signals are verified against the system level implementations. In the Virtex xc2v2000 FPGA, the system consumed less than 20 % resources, which leaves a lot of room for the receiver design and implementing other Wireless LAN protocols. These different protocols can run in parallel in conjunction with multiple RF front-ends thus enabling a true Software Defined Radio. A lot of time was spent optimizing the design. The required 60MHz clock speed

was achieved and after further optimization the design achieved 160 MHz clock rates in functional verification. Full working prototype implementation on the FPGA (on the hardware platform) was not completed due to non-availability of required hardware.

Chapter VIII

Conclusions and Future work

8.1. Conclusions

This thesis presents two important concepts which enable software-defined radio-based rapid prototyping of Wireless LAN systems. The selection of an FPGA hardware platform for prototyping provides the ability to accommodate future enhancements to the Wireless LAN protocols. Selecting a flexible platform enables designers to rapidly and smoothly transition from the simulation of a new communications system to a working prototype of the system while using existing designs. The virtue of simulating the system in Matlab and Simulink using the floating-point and fixed-point algorithms and migrating to VHDL provides the engineer with the ability to develop new algorithms in a language best suited for the algorithm, then rapidly integrate these algorithms into the prototype design. In addition, appropriate language design encourages modularity by encapsulating new algorithms in blocks, which can then be easily re-used in a different system. Appropriate language design enables flexible implementation and design clarity. The equations underlying algorithms written Matlab can be simply expressed and well documented with Matlab's rich set of mathematical operators. Simulink clearly captures the overall structure of a design in a simple block diagram. Finally VHDL coding of these blocks enables implementation of these algorithms on a prototype.

This thesis also develops novel algorithms for detection and synchronization, capable of robust operation in the unknown noise environments encountered in wireless communication systems. Using double correlation detection and synchronization offers performance gain compared to simple correlation detection and reducing the false alarm probability at the cost of increase in hardware. Simulations of these algorithms demonstrated a performance gain which can be traded with implementation complexity.

8.2. Future Work

The design implemented and the algorithms proposed in this thesis can be extended in a number of directions. A promising area for simulation and prototyping is the extension of these concepts and system level implementations to support FPGAs. The ability to efficiently generate synthesizable VHDL code from Simulink's Xilinx Block-set and the System generator would significantly enhance the power of the appropriate design language. Finally, extension of the OFDM baseband implementation to support multiple-input and multiple-output systems will enable the implementation of future Wireless LAN standards based on the evolving IEEE802.11n standard.

LIST OF REFERENCES

[1] IEEE Std 802.11_R2003, "IEEE Wireless LAN Edition – A compilation based on IEEE Std 802.11™-1999 (Reaffirmed 2003) and its amendments",2003.

[2] <http://www.mathworks.com>

[3] <http://www.xilinx.com>

[4] Richard Van Nee, Ramjee Prasad, OFDM for Wireless Multimedia Communications (Artech House Publishers – Boston, 2000)

[5] John G. Proakis "Digital Communications", McGraw Hill, New York, third edition 1995.

[6] J. Cyrus Sy, "Implementation of an OFDM transceiver using an SDR platform" Wireless Design Magazine, July 2002

[7] P.H. Moose, "A Technique for Orthogonal Frequency Division Multiplexing Frequency Offset Correction," *IEEE Transactions on Communications,* vol. 42, no. 10, October 1994

[8] T. M. Schmidl and D. Cox, "Robust frequency and timing synchronization for OFDM," *IEEE Transactions on Communications*, vol. 45, pp. 1613–1621, December 1997.

[9] Anthony Jamin et.all, "Software Radio Implementability of Wireless LAN's" , The 12th Thyrrhenian International Workshop on Digital Communications 2001.

[10] Yun Chiu et.all, "OFDM receiver design" *Project Report,* Berkeley University 2000

[10] H. Minn, M. Zeng, and V. K. Bhargava, "On timing offset estimation for OFDM systems," *IEEE Communications Letters*, 2000.

[11] < http://www.andraka.com/ >

[12] Yalamanchili, Sudhakar, VHDL Starter's Guide (Prentice Hall, 1st edition September 1997)

[13] Chang, K. C. (Kou-Chuan), Digital systems design with VHDL and synthesis: an integrated approach (Los Alamitos, Calif. : IEEE Computer Society, c1999).

[14] < www.agilent.com >

[15] Mark Webster, Carl Andren "Code word description for the Harris-Lucent Updated Compromise Proposal for TGb", IEEE 802.11-98/246-r2.

[16] S. Rajagopal, Baseband Architecture Design for Future Wireless Base-Station Receivers, *Masters Thesis*, Rice University, (May 2000).

< http://cmc.rice.edu/docs/docs/Raj2000May2BasebandAr.pdf >

[17] K. Chadha, A Reconfigurable Decoder Architecture for Wireless LAN and Cellular Systems, *Masters Thesis*, Rice University, (April 2001).

< http://cmc.rice.edu/docs/docs/Cha2001Apr2AReconfigu.pdf>

[18] B. Jones, Rapid Prototyping of Wireless Communications Systems, *Masters Thesis*, Houston, TX, Rice University (May 2002).

< http://cmc.rice.edu/docs/docs/Jon2002May2RapidProto.pdf >

APPENDIX A

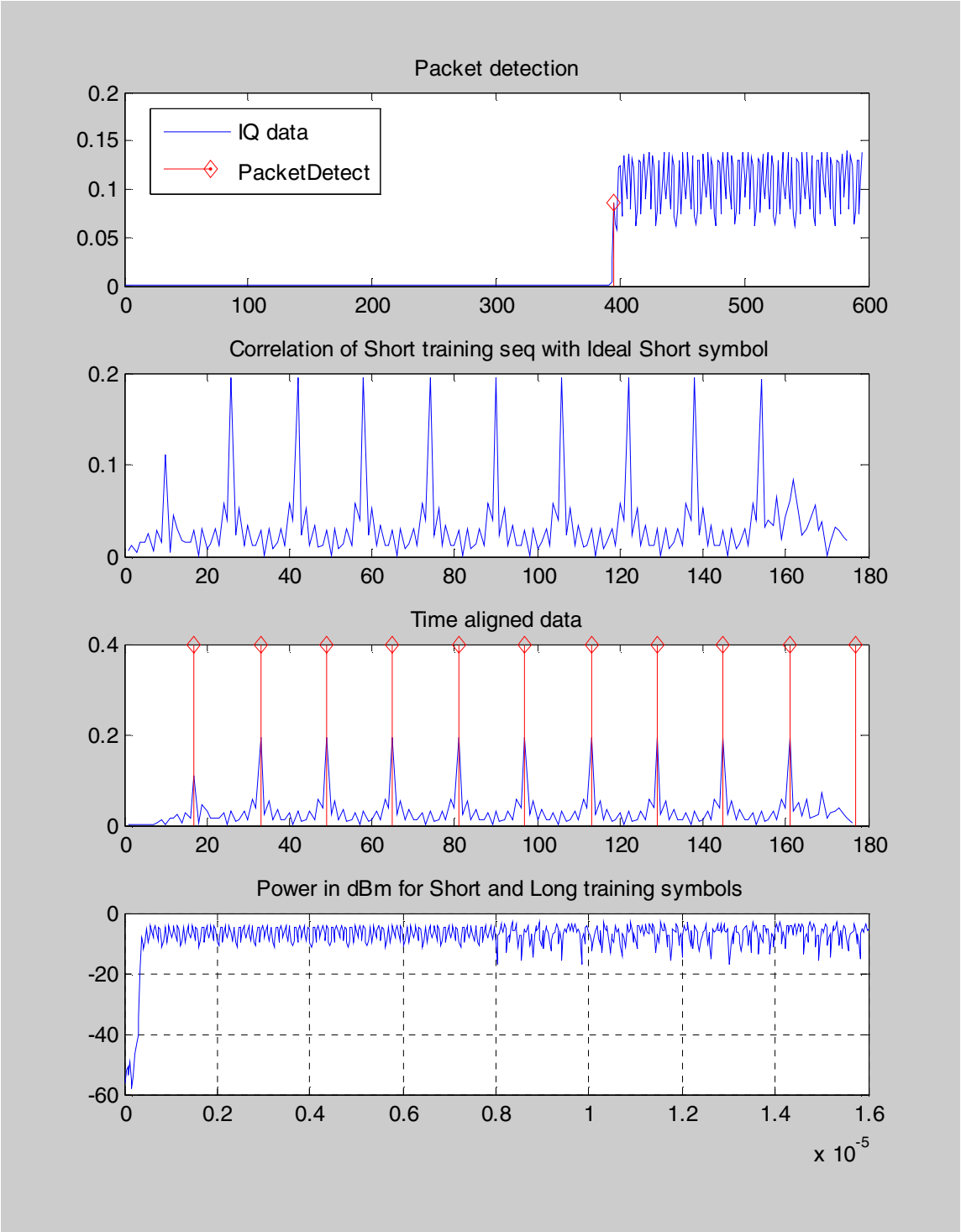

Figure A-1 Detection and Timing Acquisition

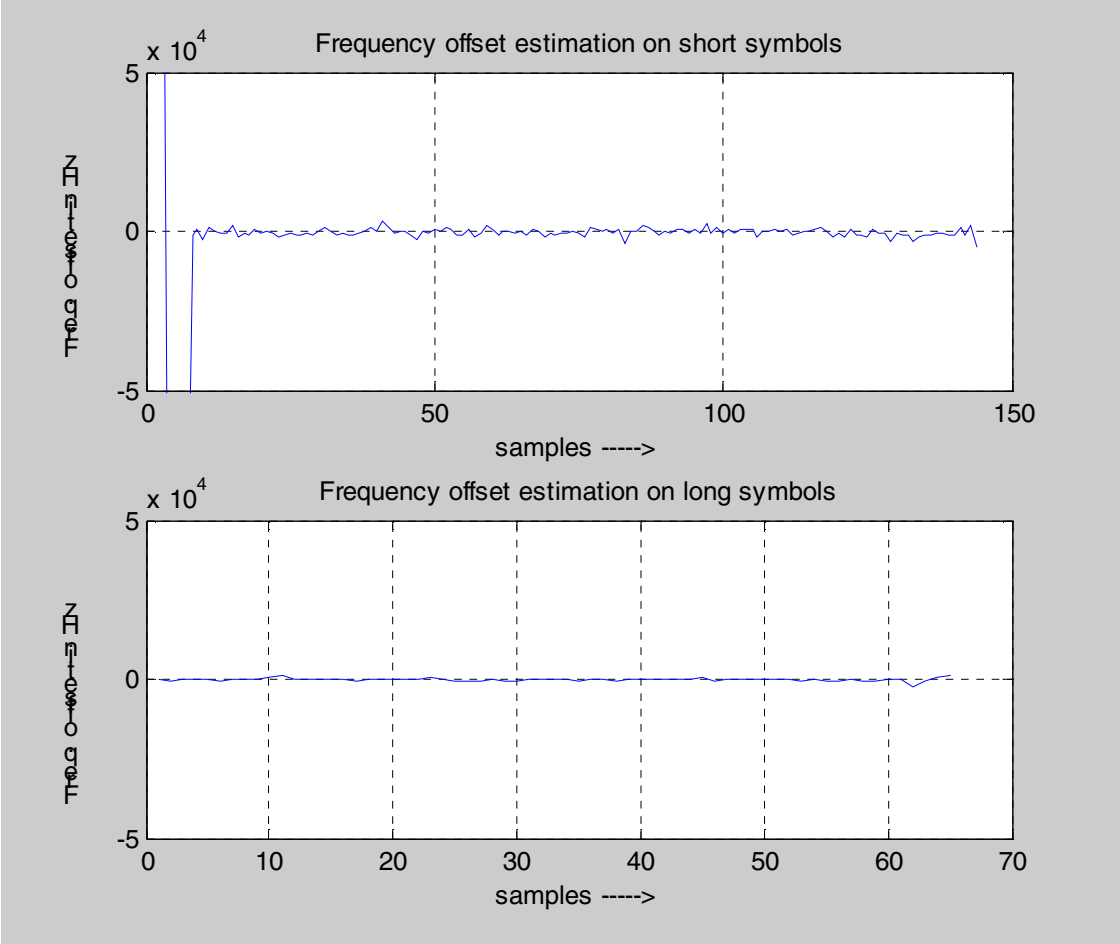

Figure A-2 Frequency offset estimation on Short and Long symbols

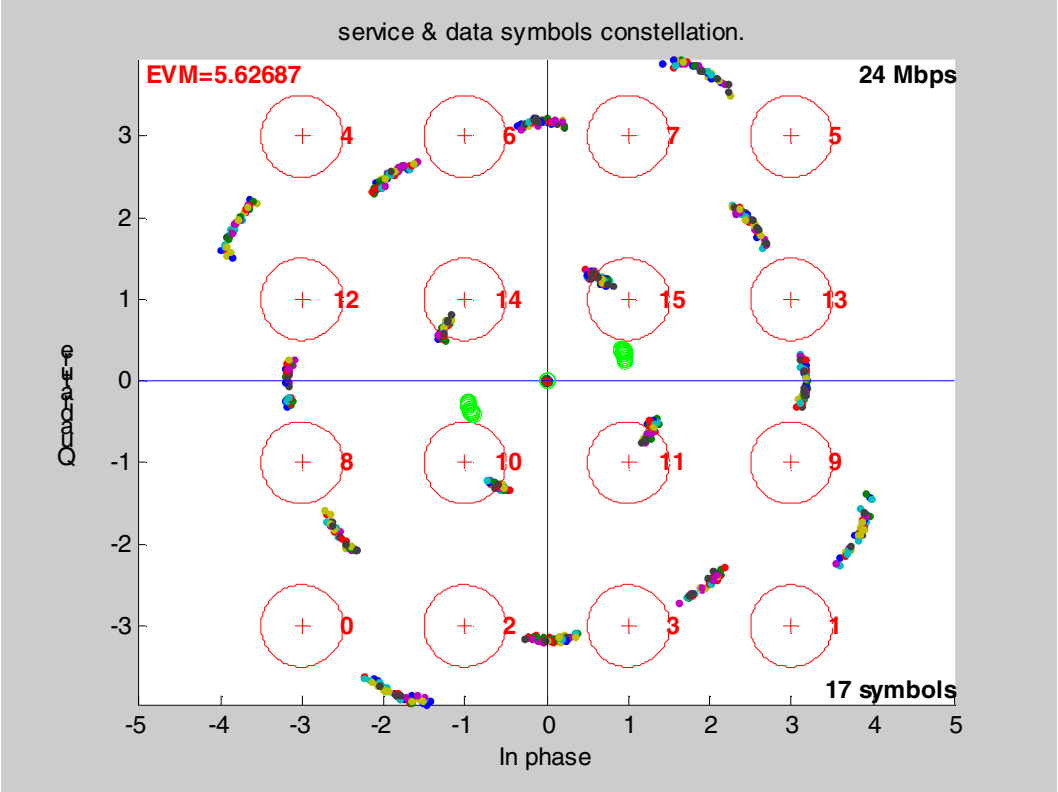

Figure A-3 Constellation before frequency offset correction

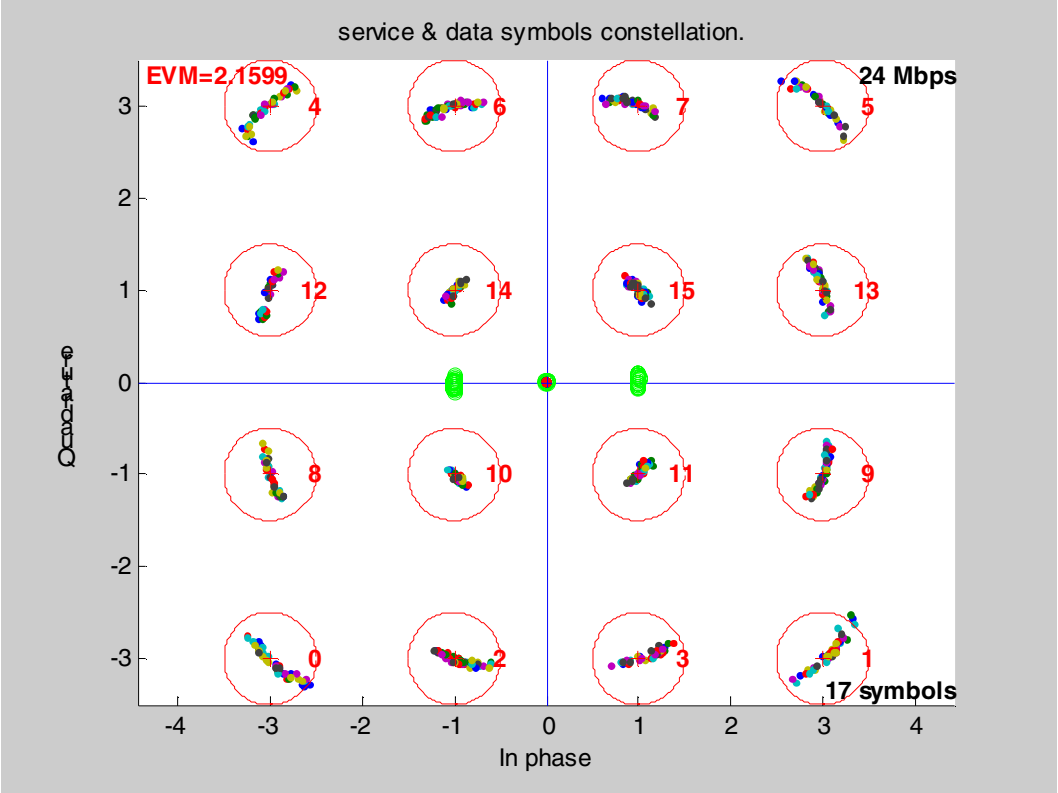

Figure A-4 Constellation after frequency offset correction

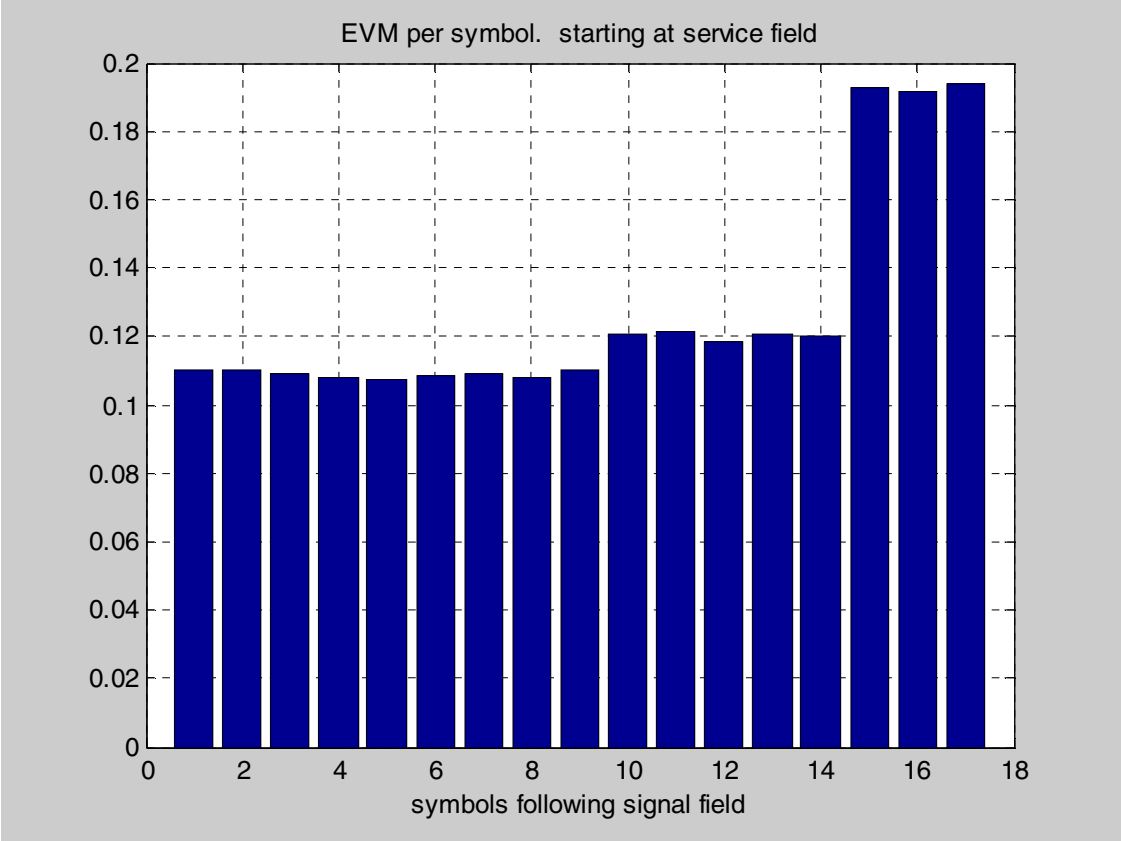

Figure A-5 EVM per symbol for decoded packet

Decoded Frame (0x)

80 00 00 00 00 30 b4 01 22 c8 00 40 96 46 84 03 00 40 96 46 84 03 30 00 00 40 96 46 84 03 ff 83 df 17 32 09 4e d1 e7 cd 8a 91 c6 d5 c4 c4 40 21 18 4e 55 86 f4 dc 8a 15 a7 ec 92 df 93 53 30 18 ca 34 bf a2 c7 59 67 8f ba 0d 6d d8 2d 7d 54 0a 57 97 70 39 d2 7a ea 24 33 85 ed 9a 1d e1 ff 07 be 2e 64 12 9d a3 cf 9b 15 23 8d ab 89 88 80 42 30 9c ab 0d e9 b9 14 2b 4f d9 25 bf 26 a6 60 31 94 69 7f 45 8e b2 cf 1f 74 1a db b0 5a fa a8 14 af 2e e0 73 a4 f5 d4 48 67 0b db 34 3b c3 bc 4e 89 aa

calculatedCRC32= 0x bc 4e 89 aa , receivedCRC32= 0x bc 4e 89 aa

Decoded Frame (hex)

FC: 80 00 (Beacon --------)

Dur: 00 00

Addr1: 00 30 b4 01 22 c8

Addr2: 00 40 96 46 84 03

Addr3: 00 40 96 46 84 03

SeqCnt: 30 00

seq: 03

frg: 00

rcvdCRC32: bc 4e 89 aa

calcCRC32: bc 4e 89 aa