Multi-Clock Pipeline Architecture for the IEEE 802.11a Baseband Transceiver

by

Maryam Mizani

B.Sc., University of Tehran, 2000
M.A.Sc., Tarbiat Modares University, 2002

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Applied Science

in the Electrical and Computer Engineering

Multi-Clock Pipeline Architecture for the IEEE 802.11a Baseband Transceiver

by

Maryam Mizani

B.Sc., University of Tehran, 2000
M.A.Sc., Tarbiat Modares University, 2002

Supervisory Committee

Dr. D. N. Rakhmatov, Supervisor(Department of Electrical and Computer Engineering)

Dr. A. Baniasadi, Department Member(Department of Electrical and Computer Engineering)

Dr. M. McGuire, Department Member(Department of Electrical and Computer Engineering)

Supervisory Committee

Dr. D. N. Rakhmatov, Supervisor(Department of Electrical and Computer Engineering)

Dr. A. Baniasadi, Department Member(Department of Electrical and Computer Engineering)

Dr. M. McGuire, Department Member(Department of Electrical and Computer Engineering)

Abstract

Demand for Wireless Local Area Networking (WLAN) has grown significantly during the past several years. WLAN systems need to support varying data rate applications and consume low amount of energy. This work presents a reconfigurable WLAN transceiver architecture that has the following key features: *Four-stage pipeline structure* to increase throughput and reduce dynamic power consumption, *Multiple adjustable clocks* to avoid excessive handshaking and buffering between pipeline stages, *Dynamic reconfigurability* to support different modes of operation, and *Low reconfiguration cost*, in terms of energy consumption and delay, to allow for efficient frame-by-frame adaptation.

We have chosen the IEEE 802.11a standard as the demonstration platform, however our ideas are extendable to other WLAN standards that are based on similar communication principles. For example, the popular IEEE 802.11g standard uses the same Orthogonal Frequency Division Multiplexing (OFDM) scheme as 802.11a. Consequently, both standards require somewhat similar data processing, i.e., our design techniques remain applicable. Our proposed architecture is prototyped on Xilinx

FPGA, and simulations show a relatively low power consumption in comparison with other 802.11a baseband processors.

Table of Contents

List of Tables

List of Figures

Acknowledgement

I would like to express my gratitude to my supervisor, Dr. Daler Rakhmatov, for his patience, advice and support. İ would also like to thank the members of the Electrical and Computer Engineering Department at the University of Victoria who mentored me during my Master's program.

This work would not have been possible without the support of my parents and my sisters, Fati and Shima, who have been a constant source of love and inspiration throughout my entire life. I also like to acknowledge my friends and lab mates: Bahar, Yasi, Mahsa, Solmaz, Marjan, and particularly Farshad who were always there for me throughout the ups and downs of the last few years.

And of course, there is Jochen, my dear husband, whose love and support always makes me stronger. His amazing positive attitude and pragmatic solutions helped me a lot in completing this work. Finally, I would like to thank my little son, my dear Tristan, for adding so much to the beauty of my life.

To my parents for their endless love and support

Chapter 1

Introduction

The field of wireless communications is growing fast and the vision of ubiquitous information access is becoming a reality [1]. Different types of networks that support various types of communication provide a universal information access anytime anywhere. For example, in an office environment, high-speed connectivity is provided by wireless local area networks (WLAN), while the global coverage is established over the cellular or satellite networks. Recently, WLAN networks have received a lot of interest due to their efficiency, lower cost, mobility, and flexibility. However, designing WLAN systems is a challenging task since they should support diverse applications that emphasize different Quality of Service (QoS) metrics, e.g., communication range, latency, bandwidth, reliability, etc. For example, a telephony application may need longer range at the expense of lower throughput in comparison to network-attached storage (NAS) [2]. Low power consumption is also of paramount importance in wireless applications. Modern terminals must consider the existing trade-offs for a specific application and adapt accordingly. In this work, we propose an architecture for the physical layer of WLAN systems that is optimized to consume low power and

introduce low overhead during data rate adaptation.

1.1 WLAN background

The first standard in the area of WLAN communications was the IEEE 802.11b [3] introduced in 1999[1]. Although IEEE 802.11a [5] was proposed at the same time as 802.11b, it has not been widely used due to its implementation difficulties. Recent advancements in CMOS technology have led to single-chip implementations of the 802.11a standard. Using an OFDM-based modulation scheme, 802.11a supports data rates of up to 54 Mbits/s which is nearly five times the data rate of 802.11b LANs [5–7]. IEEE 802.11g [8] is another WLAN standard that was introduced in 2003. It works on a similar basis as 802.11a and offers a high data rate of up to 54 Mbits/s, while incorporating backward compatibility with 802.11b. 802.11g works at 2.4 GHz which is a major drawback for this standard compared to 802.11a that works at 5 GHz. The 5 GHz band offers the advantages of more available spectrum and a better environment with less noise and interference from other electronic devices such as microwave ovens and cordless phones [9]. The upcoming WLAN standard, 802.11n [10], is an evolution of both 802.11a and 802.11g which supports Multiple-Input/Multiple-Output (MIMO) technology [11]. This standard is a proposed amendment to the IEEE 802.11-2007 standard and it is still in "draft" stage according to the IEEE. Table 1.1 shows a comparison between IEEE WLAN standards. The high throughput and less crowded frequency band of 802.11a makes it a good choice for single antenna WLAN communications.

[1]HIPERLAN (HIgh PErformance Radio LAN) [4], which is the European alternative for IEEE 802.11, started being planned in 1991, when planning of 802.11b was already going on. HIPERLAN has a similar basis as the IEEE 802.11a standard, however this work is focused on the latter.

Protocol	Operating frequency (GHz)	Max datarate (Mbps)	Range (Radius indoor) (m)	Range (Radius outdoor) (m)
802.11a	5	54	35	120
802.11b	2.4	11	38	140
802.11g	2.4	54	38	140
802.11n	2.4/5	248	70	250

Table 1.1: Comparison chart of IEEE WLAN standards [12].

1.2 Related work and motivation

Recent literature reports both software-based and hardware-based implementations of the IEEE 802.11a standard. Software-based implementations either use a single DSP processor [16, 17, 21] or a multiprocessor system [18–20] to implement 802.11a transceiver. These systems, known as Software Defined Radio (SDR) systems [22, 23], facilitate rapid design and offer a high degree of flexibility, but consume more power than hardware-based implementations. The reason is that the sequential nature of microprocessors calls for clock frequencies as high as 1.0 GHz, as reported in [16], to meet the performance requirements of 802.11a transceiver, and high clock frequencies result in high power consumption [13]. However, the system clock frequency can be reduced considerably in DSPs with parallel architecture or with hardware accelerators. The domain specific DSP architecture used to implement 802.11a processor in [18] is an example of such parallel architectures.

Custom hardware, on the other hand, can fully exploit application-specific parallelism to achieve high throughput at low clock frequencies. However, in the context of 802.11a, hardware-based implementations may suffer from insufficient flexibility to handle different operating modes. One solution to this problem is to use FPGA-based reconfiguration in which the functionality of individual hardware blocks and the interconnections between them are determined by uploading different bit streams into the FPGA [24, 25]. However, if many configuration bytes need to be loaded into the

FPGA, the energy overhead of the PHY layer reconfiguration becomes a bottleneck, preventing the MAC layer to adjust the data rate frequently. Low reconfiguration overhead of the PHY layer lets the MAC layer to constantly switch to the most power-efficient mode as channel conditions vary. In bad channel conditions, selecting modes with slower data rates saves the number of retransmissions. In good channel conditions, selecting modes with faster data rates saves the number of transmissions. The faster the MAC layer adjusts the data rate, the fewer packets need to be transmitted or retransmitted. The fewer packet transmissions translates into lower energy consumption.

The most efficient 802.11a systems, in terms of speed and energy consumption, are ASIC implementations [28–30]. For example, reference [31] reports an implementation of digital baseband 802.11a transceiver in 0.25 μm technology that consumes 393 mW power and works with a maximum 80 MHz clock frequency. This clock frequency is much lower than 1.0 GHz clock frequency of the software-based implementation reported in [16]. Unfortunately most of the software-based implementations do not report any power number to be compared with the power of the ASIC designs.

The proposed architecture in this work is prototyped on FPGA for the validation purpose, but it is transferable to ASIC with small modifications. Except for the memory blocks that use the FPGA *Coregen* library, we have not used any other FPGA predefined libraries or FPGA reconfiguration capabilities. This architecture employs a carefully pipelined hardware structure with multiple clocks, where data can be processed non-stop. Consequently, a given throughput target can be achieved at very low clock frequencies, i.e., 72 MHz is the highest clock frequency that is required in this design to support the fastest operating mode. Slower operating modes work with even lower clock frequencies. Lower clock frequencies enable lower supply voltages, which significantly reduces the dynamic power consumption. Adapting our hardware to work with eight different data rates is very efficient, as it requires only

two adjustments: (1) changing the value of a 3-bit mode signal, and (2) changing the frequency of two clock signals. These changes modify the operation of the datapath and controllers according to the selected mode. Multiple adjustable clocks also avoid excessive handshaking and buffering that is required in conventional single-clock systems. We divide our hardware blocks to four pipeline stages and apply a different clock to every stage of the pipeline: each stage of the pipeline has a separate clock, but all the blocks in one stage use the same clock. Thus, the processing rate of different hardware blocks are adjusted stage by stage in the pipeline structure and interfacing between pipeline stages is implemented using inter-stage buffers. The concept of employing multiple clocks for different pipeline stages can be applied to many communication systems and considerably decrease the clock frequency and therefore energy consumption. Finally, our design introduces low energy penalties for rate adaptation which allows the MAC layer to perform more efficient frame-by-frame reconfiguration.

1.3 Outline

The rest of this thesis is organized as follows. In Chapter 2, we introduce the IEEE 802.11a standard and explain the function of its hardware components. We discuss the hardware characteristics that can be utilized to optimize the 802.11a transceiver in Chapter 3. Based on these characteristics, we present a multi-clock pipeline structure for IEEE 802.11a transceiver and elaborate its architectural features in Chapter 4. These features can be utilized to design other WLAN transceivers as well. In chapter 5, we analytically derive the latency of the proposed architecture during transmission and reception. These latency values are expressed by the number of clock cycles, therefore they are independent of the implementation technology (ASIC, FPGA, etc). In chapter 6, we present the implementation results of prototyping our architecture

on Xilinx VirtexII-Pro FPGA and compare them with the implementation results of other 802.11a designs. In Chapter 7, we explain how this research can be extended in future, and finally we conclude our work in Chapter 8.

Chapter 2

IEEE 802.11a Physical Layer Transceiver

The IEEE 802.11a standard defines the transmission rules for the physical (PHY) layer of WLAN transceivers in North America. The PHY layer provides the interface between the MAC layer and wireless medium. The MAC layer controls transmission tasks such as acknowledgement checking and requesting for retransmissions. MAC protocol tasks are implemented in software or partly by using hardware accelerators [14]. The software implementation of the MAC layer is outside the scope of this work, we focus only on the PHY layer. The 802.11a PHY layer, as it is shown in Figure 2.1, consists of two sublayers.

- **PHY Layer Convergence Procedure (PLCP) sublayer,** which is the interface between the IEEE 802.11 MAC layer and medium dependent sublayer. This sublayer simplifies the PHY service interface and allows the MAC layer to operate with minimum dependence on wireless medium.

- **PHY Medium Dependent (PMD) sublayer,** which provides the means to send and receive data through air and contains mixed-signal data converters and RF circuitry.

During transmission, the PLCP sublayer receives MAC Protocol Data Units

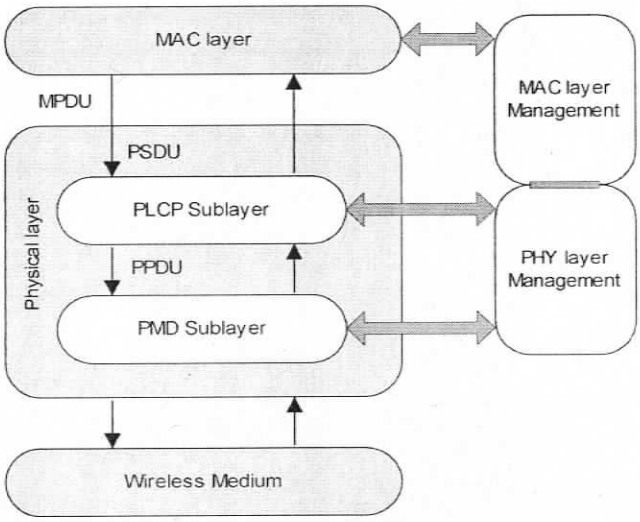

Figure 2.1: The IEEE 802.11a PHY layer provides the interface between the MAC layer and wireless medium.

(MPDU) from the MAC layer. In the IEEE 802.11a PHY layer, each MPDU coming from the MAC layer is one PHY Service Data Unit (PSDU)[1]. The PLCP sublayer encapsulates PSDUs into OFDM symbols and puts them into a framing format, called PLCP Protocol Data Unit (PPDU), which is suitable for the PMD sublayer. The PMD sublayer receives the constructed PPDUs, interpolates and upconverts them, and passes them to the antenna.

During reception, the PMD sublayer detects data reception and it passes the PPDUs to the PLCP sublayer. The PLCP sublayer extracts the binary data out of PPDUs and sends it to the MAC layer.

In this chapter, first we introduce the PPDU framing format that is used by the IEEE 802.11a PHY layer and explain how these frames are constructed at the transmitter and how binary data is extracted from them at the receiver. Then, we explain the functionality of all the hardware components in the transmission and reception chain of the PHY layer according to the IEEE 802.11a standard.

[1]In IEEE 802.11n, one PSDU can contain multiple MPDUs [10].

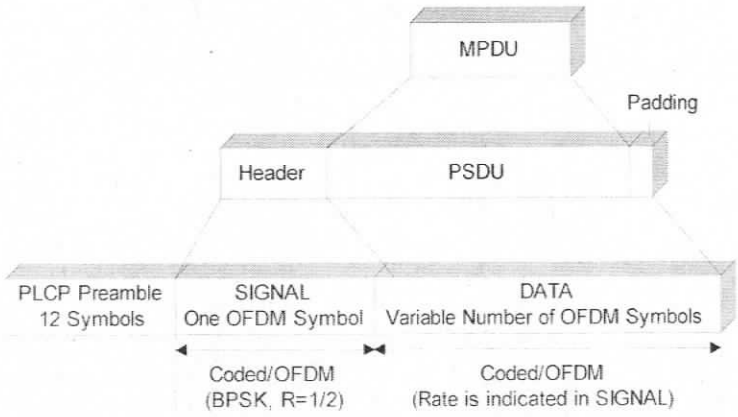

Figure 2.2: The PPDU framing format [5].

2.1 The PLCP sublayer framing format

The format of the PPDU frames is shown in Figure 2.2. Every PPDU includes a PLCP Preamble, SIGNAL, and DATA field.

- **PLCP Preamble:** consists of 12 symbols which are used for the synchronization purposes at the receiver side. The first ten symbols, called short training sequence (STS), are used at the receiver side for signal detection, automatic gain control (AGC), coarse frequency offset estimation and timing synchronization. The other two symbols of the preamble, called long training sequence (LTS), are used for channel estimation and fine frequency offset estimation.

- **PLCP SIGNAL:** contains header information that are used to process the rest of the frame, therefore it needs to be transmitted in the most robust mode which is a combination of BPSK modulation and a coding rate of R = 1/2. Figure 2.3 shows the bit assignment of the SIGNAL field. It contains 4 bits to define frame RATE (6, 9, 12, 18, 24, 36, 48, and 54), a reserved bit, 12 bits to define frame LENGTH (1-4095), an even parity bit, with 6 zero tail bits appended.

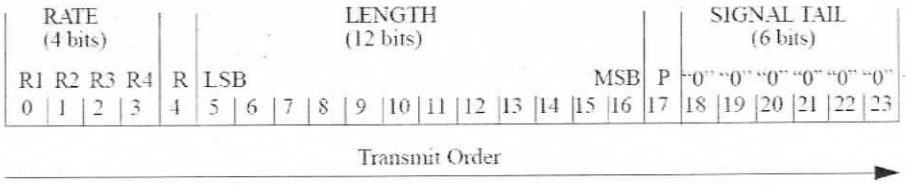

Figure 2.3: The bit assignment of the SIGNAL field [5].

- **PLCP DATA:** contains the SERVICE field of the PLCP header, PSDU, 6 zero tail bits to return the convolutional encoder of the receiver to the zero state and pad bits.

2.2 IEEE 802.11a transmitter

The PHY layer transmitter receives binary data from the MAC layer and sends them out through wireless medium. A block diagram of the IEEE 802.11a PHY layer transmitter is shown in Figure 2.4. DATA octets coming from the MAC layer (MPDUs) pass through the Scrambler, Coder, Interleaver and Modulator blocks. After modulation, pilot signals are inserted between data samples in order to make the coherent detection at the receiver robust against frequency offsets and phase noise. Between every 48 data samples, 4 pilot signals are inserted. These 52 samples, padded with 12 zeros, make a group of 64 samples in frequency domain. A 64-point Inverse Fast Fourier Transform (IFFT) is used to convert the 64 samples from frequency domain into time domain. After IFFT is performed, each group of 64 samples are cyclicly extended and windowed, resulting in an OFDM symbol with 80 samples. DATA OFDM symbols are prepended with a preamble (for receiver synchronization) and the SIGNAL field which carries the header information. Finally, the PPDU is sent to the PMD sublayer to be interpolated, upconverted and sent out through the antenna.

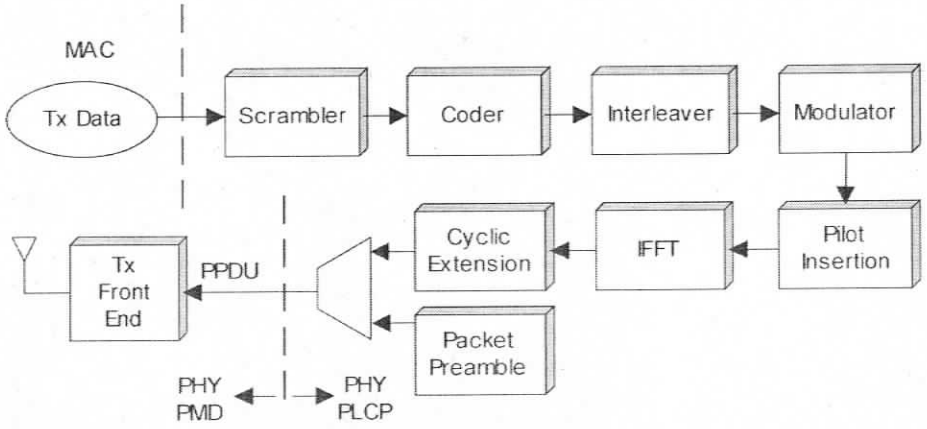

Figure 2.4: IEEE 802.11a PHY layer transmitter.

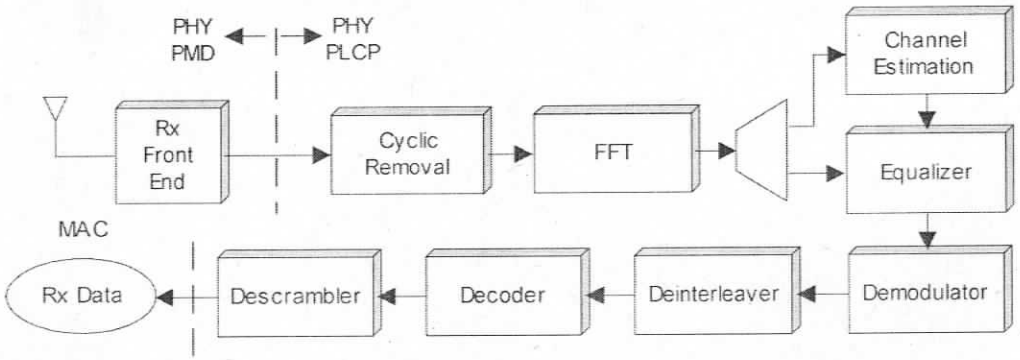

Figure 2.5: IEEE 802.11a PHY layer receiver.

2.3 IEEE 802.11a receiver

Upon receiving a frame, the PMD sublayer signals to the PLCP sublayer to start processing the received frame. First, the Preamble LTS enters the PLCP sublayer to be used for the channel estimation. Next, the SIGNAL is handled followed by the DATA field. The DATA field is processed according to the specifications provided by the SIGNAL. After the resulting MPDU is sent to the MAC layer, the PLCP sublayer returns to the idle state.

Figure 2.5 shows the block diagram of the processing steps required to extract data (MPDU) from a frame (PPDU). In the PLCP sublayer, first cyclic extension is removed from the received OFDM symbols. Then, symbols are converted to the

Data Rate Mode (Mbits/s)	Modulation	Coding Rate R	Coded Bits per Subcarrier N_{BPSC}	Coded Bits per OFDM Symbol N_{CBPS}	Data Bits per OFDM Symbol N_{DBPS}
6	BPSK	1/2	1	48	24
9	BPSK	3/4	1	48	36
12	QPSK	1/2	2	96	48
18	QPSK	3/4	2	96	72
24	16-QAM	1/2	4	192	96
36	16-QAM	3/4	4	192	144
48	64-QAM	2/3	6	288	192
54	64-QAM	3/4	6	288	216

Table 2.1: IEEE 802.11a modes and rate-dependent parameters [5].

frequency domain by the FFT block. If the FFT is working on the Preamble LTS, its output is sent to the Channel Estimation block, otherwise its output is sent to the Equalizer. The Channel Estimation block utilizes the Preamble LTS to calculate the equalization coefficients. These coefficients are used to equalize the SIGNAL and DATA fields. The equalized samples pass through the Demodulator, Deinterleaver, Decoder, and Descrambler blocks. Finally, the extracted data octets are sent to the MAC layer.

2.4 IEEE 802.11a PLCP components

The 802.11a standard supports eight different data rates, which translate into eight different modes of data transmission and reception. These operating modes, summarized in Table 2.1, determine the performance constraints for the PLCP components.

2.4.1 Scrambler and Descrambler

The Scrambler is the first component at the transmitter side which receives DATA field[2], composed of SERVICE, PSDU, tail, and pad parts, and scrambles them by a length-127 frame-synchronous scrambler. The octets of the PSDU are placed in the

[2]The frame SIGNAL field does not pass through the Scrambler.

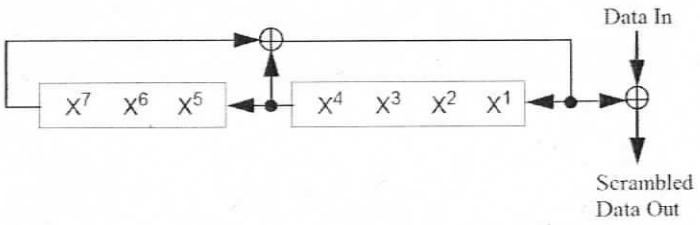

Figure 2.6: Data Scrambler and Descrambler [5].

transmit serial bit stream, bit 0 first and bit 7 last. The frame synchronous scrambler uses the generator polynomial $S(x) = x^7 + x^4 + 1$ as it is illustrated in Figure 2.6. The same scrambler is used at the receiver side to descramble received data.

2.4.2 Coder and Decoder

At the transmitter side, data is coded by a convolutional encoder of coding rate R=1/2, 2/3, or 3/4 [3], corresponding to the desired data rate. The convolutional coder uses the industry-standard generator polynomials, g0=1338 and g1=1718, of rate R=1/2, as shown in Figure 2.7. The bit denoted as A is output from the encoder before the bit denoted as B. Higher rates are derived from it by employing puncturing. The puncturer omits some of the coded bits in the transmitter in order to reduces the number of transmitted bits and increases the coding rate. The puncturing patterns defined in the IEEE 802.11a standard are shown in Figure 2.8.

At the receiver side, Decoder receives coded data that might be noisy after passing through the wireless channel. It uses a convolutional decoder (i.e., Viterbi [46] or MAP decoder [48]) to decode data[4]. If coding rate is more than R=1/2 due to puncturing, it inserts zeros in the punctured positions before starting the decoding process.

[3]The SIGNAL field always uses a coding rate of R=1/2, but DATA field can use other allowed coding rates.

[4]A Viterbi decoder is recommended by the IEEE 802.11a standard to be used at the receiver. The coding gain required by the 802.11a decoder is not high, and a Viterbi decoder can achieve it with lower price than a MAP decoder [49].

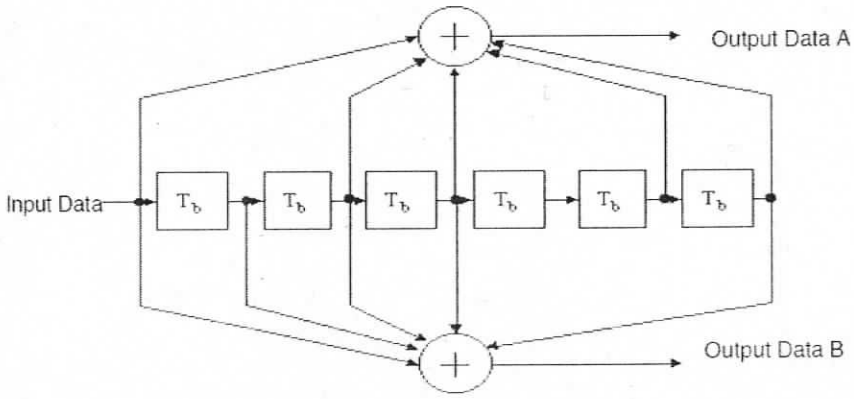

Figure 2.7: Convolutional Coder (k = 7) [5].

2.4.3 Interleaver and Deinterleaver

At the transmitter side, all encoded data bits are interleaved by a block interleaver with a block size corresponding to N_{CBPS} (The number of bits in a single OFDM symbol). The interleaver is defined by a two-step permutation. The first step ensures that adjacent coded bits are mapped onto nonadjacent subcarriers. The second step ensures that adjacent coded bits are mapped alternately onto less and more significant bits of the constellation and, thereby, long runs of low reliability bits (LSB) are avoided.

Let k and i respectively denote the index of the coded bit before and after the first permutation[5]. The first permutation is defined by

$$i = \lfloor k/16 \rfloor + (N_{CBPS}/16)(k \bmod 16) \tag{2.1}$$

The second permutation is defined by the following equation, assuming that j denotes the index of the coded bit after the second permutation.

$$j = s\lfloor i/s \rfloor + (i + N_{CBPS} - \lfloor 16 \times i/N_{CBPS} \rfloor) \bmod s \qquad s = \max\{1, N_{BPSC}/2\} \tag{2.2}$$

[5]The values of k and i are integers ranging from 0 to $(N_{CBPS} - 1)$.

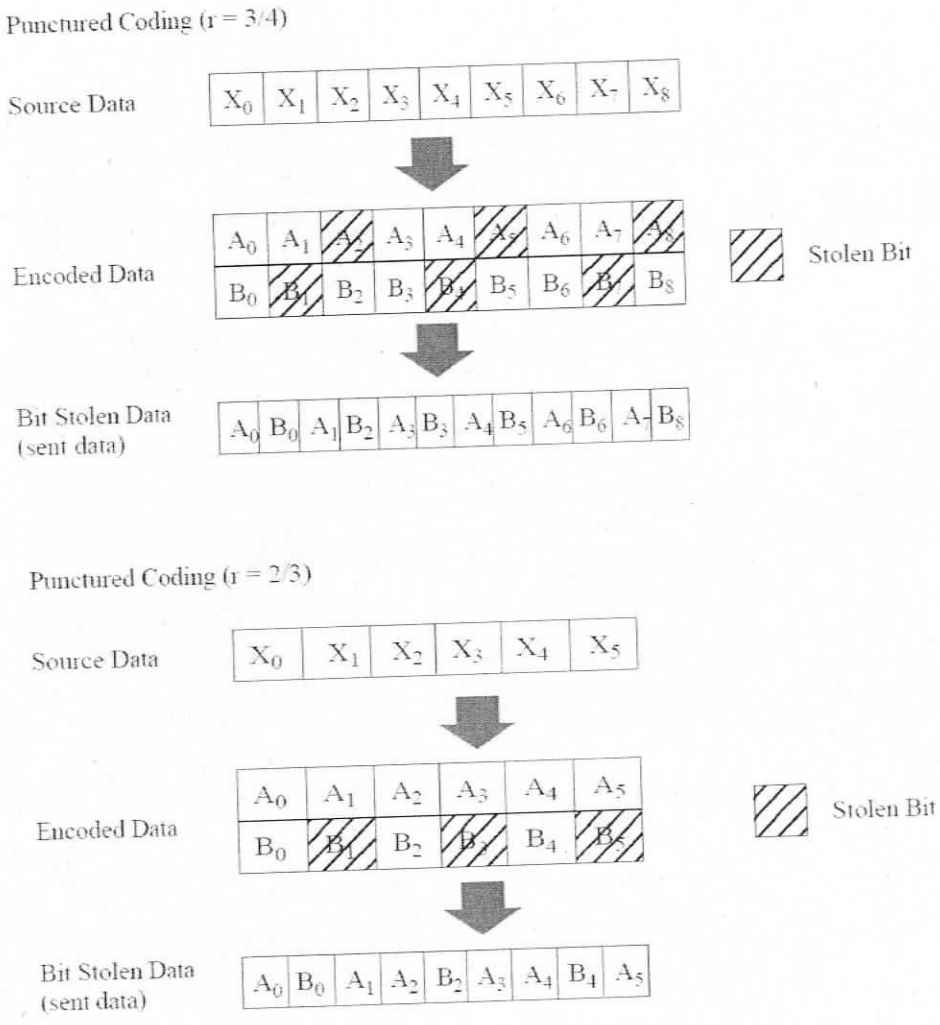

Figure 2.8: Puncturing patterns [5].

The Deinterleaver performs the reverse operation at the receiver side. Let index j represent the position of original received bit before the first permutation. i be the index after the first and before the second permutation, and k be the index after the second permutation, just prior to delivering the coded bits to the convolutional (Viterbi) decoder.

The first permutation of the Deinterleaver is defined by

$$i = s\lfloor j/s \rfloor + (j + \lfloor 16 \times j/N_{CBPS} \rfloor) \bmod s \qquad s = .\max\{1, N_{BPSC}/2\} \tag{2.3}$$

and the second permutation is defined by

$$k = 16 \times i - (N_{CBPS} - 1)\lfloor 16 \times i/N_{CBPS} \rfloor \tag{2.4}$$

2.4.4 Modulator and Demodulator

At the transmitter side, binary data should be modulated using BPSK, QPSK, 16-QAM, or 64-QAM, depending on the requested data rate. The encoded and interleaved binary serial input data is divided into groups of N_{BPSC} (1, 2, 4, or 6) bits and converted into complex numbers representing BPSK, QPSK, 16-QAM, or 64-QAM constellation points. The conversion is a Gray-coded constellation mapping illustrated in Figure 2.9 with the input bit, b0, being the earliest in the stream.

The output value, d, is formed by multiplying the resulting $(I + jQ)$ value by a normalization factor K_{MOD}, as described in Equation 2.5.

$$d = (I + jQ) \times K_{MOD} \tag{2.5}$$

The normalization factor, K_{MOD}, illustrated in Table 2.2, depends on the base modulation mode. The modulation type can be different from the start to the end

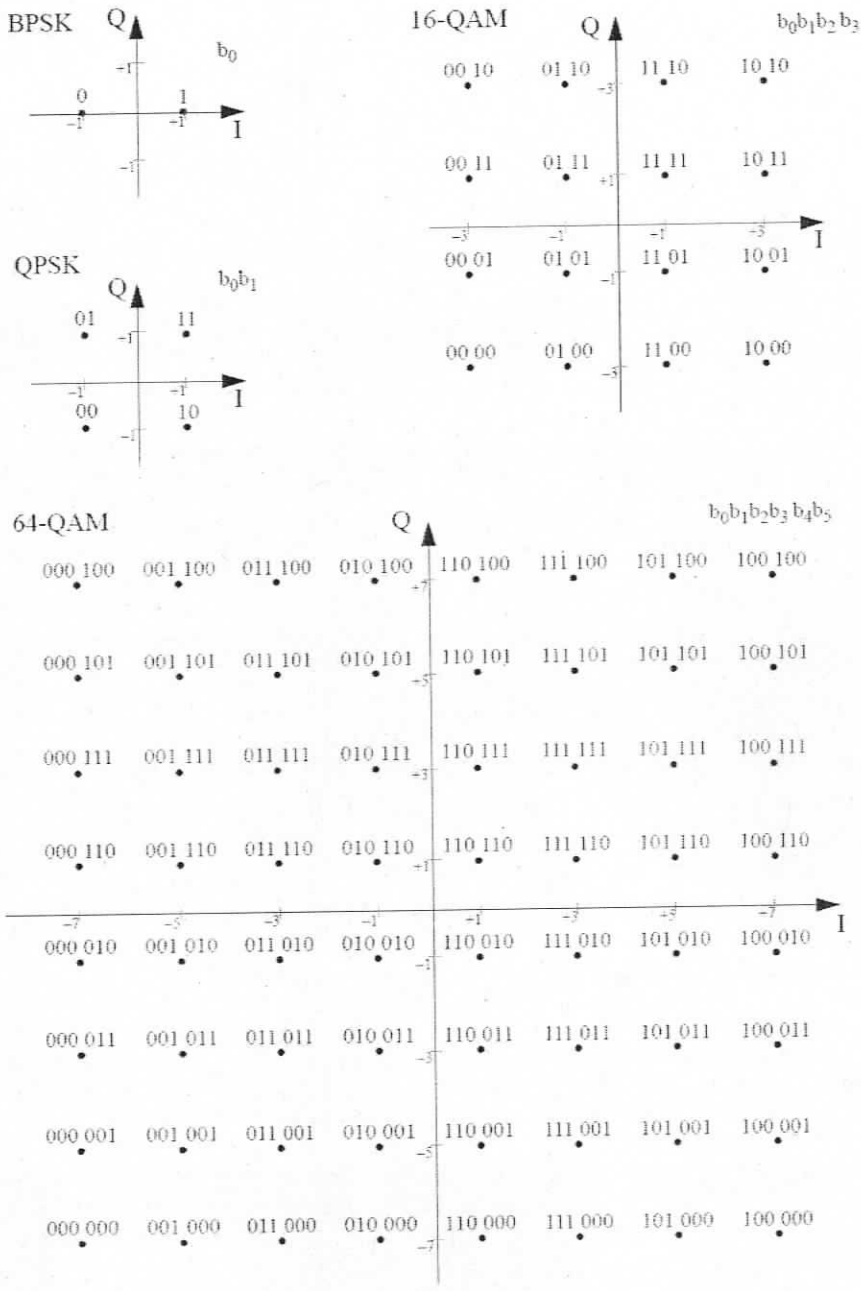

Figure 2.9: BPSK, QPSK, 16-QAM, and 64-QAM constellation bit encoding [5].

Modulation	K_{MOD}
BPSK	1
BPSK	1
BPSK	1
BPSK	1
QPSK	$1/\sqrt{2}$
16-QAM	$1/\sqrt{10}$
64-QAM	$1/\sqrt{42}$

Table 2.2: Modulation-dependent normalization factor K_{MOD} [5].

of the transmission, as data changes from SIGNAL to DATA field. The purpose of the normalization factor is to achieve the same average power for all mappings.

At the receiver side, the Demodulator performs reverse operation of the Modulator. First, it divides input data by K_{MOD} normalization value, then it assigns N_{BPSC} bit values to every complex number according to constellation mapping defined in Figure 2.9.

2.4.5 Pilot Insertion

After modulation at the transmitter side, the complex number string is divide into groups of $N_{SD} = 48$ complex numbers. Each such group of data subcarriers, with 4 pilot subcarriers, form one OFDM symbol. The pilot subcarriers are inserted between data subcarriers to make the coherent detection at the receiver robust against frequency offsets and phase noise.

The subcarrier frequency allocation is shown in Figure 2.10, where d_k denotes a complex number corresponding to k^{th} data subcarrier, k={0,..., N_{SD}–1}, and P_n denotes the n^{th} pilot subcarrier, n={–26, ..., 26}.

Pilot signals are inserted in subcarriers −21, 7, 7 and 21. The pilots should be BPSK modulated by a pseudo binary sequence to prevent the generation of spectral lines. The subcarrier falling at DC (0th subcarrier) is not used to avoid difficulties in D/A and A/D converter and RF system.

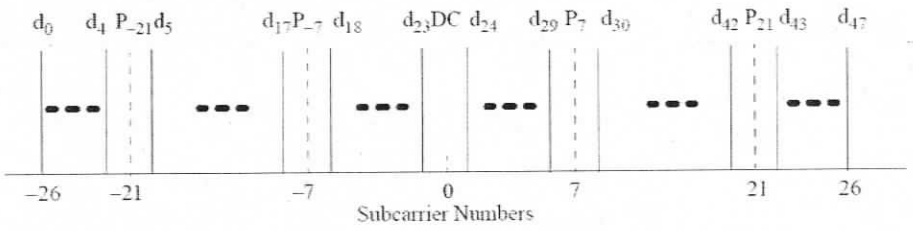

Figure 2.10: Subcarrier allocation of an OFDM symbol [5].

The contribution of the pilot subcarriers for the n^{th} OFDM symbol is produced by Fourier transform of sequence P, given by

$$P_{-26,26} = \{0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,$$
$$0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,-1,0,0,0,0,0\}$$

The polarity of the pilot subcarriers is controlled by the sequence, p_n, which is a cyclic extension of the 127 elements sequence and is given by

$$p_{0..126v} = \{1,1,1,1,-1,-1,-1,1,-1,-1,-1,-1,1,1,-1,1,-1,-1,1,1,-1,$$
$$1,1,-1,1,1,1,1,1,1,-1,1,1,1,-1,1,1,-1,-1,1,1,1,-1,1,-1,$$
$$-1,-1,1,-1,1,-1,-1,1,-1,-1,1,1,1,1,1,-1,-1,1,1,-1,-1,$$
$$1,-1,1,-1,1,1,-1,-1,-1,1,1,-1,-1,-1,-1,1,-1,-1,1,-1,1,$$
$$1,1,1,-1,1,-1,1,-1,1,-1,-1,-1,-1,-1,1,-1,1,1,-1,1,-1,$$
$$1,1,1,-1,-1,1,-1,-1,-1,1,1,1,-1,-1,-1,-1,-1,-1,-1\}$$

The sequence, p_n, can be generated by the scrambler defined in Figure 2.6, when the all ones initial state is used, and by replacing all 1s with -1 and all 0s with 1. Each sequence element is used for one OFDM symbol. The first element, p_0, multiplies the pilot subcarriers of the SIGNAL symbol, while the elements from p_1 on are used for the DATA symbols.

At the receiver side, the pilot signals are used for Channel estimation. We will discuss the function of the Channel estimation block in Section 2.4.10.

2.4.6 IFFT and FFT

After pilot insertion, the IFFT block of transmitter receives 52 input samples, it sets the remaining 12 IFFT points to zero, and performs 64-point IFFT operation. The output of the IFFT block is 64 complex samples in time domain. At the receiver side, the FFT block performs the reverse operation by receiving 64 samples in time domain and converting them to 64 samples in frequency domain. It is possible to perform IFFT with the same FFT block provided that the position of real and imaginary parts are exchanged before and after IFFT operation and the final result is divided by the length of input sequence (N=64) [43].

The FFT algorithm is performed on a series of N numbers as follows. Let $\{x(0), \ldots, x(N-1)\}$ denote the input complex numbers and $\{X(0), \ldots, X(N-1)\}$ denote the output complex numbers. The Discrete Fourier Transform (DFT) is defined by [41]:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-\frac{2\pi j}{N} nk} \qquad k = 0, \ldots, N-1. \tag{2.6}$$

The direct method to evaluate these sums takes $O(N^2)$ arithmetical operations. A fast Fourier transform (FFT) computes the same result in only $O(NlogN)$ operations. The most common algorithm for computing FFT is Cooley-Tukey algorithm [42], which is a divide and conquer algorithm that recursively breaks down the transform into two pieces of size $N/2$. The smallest transform in this recursion is called a radix-2 butterfly. Figure 2.11 shows a radix-2 butterfly signal flowgraph, where $x(0)$ and $x(1)$ are complex-valued input data points, W_N^{nk} is the complex valued FFT coefficient, or

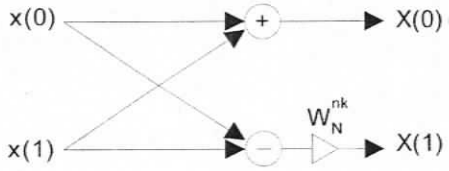

Figure 2.11: Radix-2 FFT butterfly.

twiddle factor, and $X(0)$ and $X(1)$ are the complex-valued output data points [43].

$$W_N^{nk} = e^{-\frac{2\pi j}{N}nk}, \qquad k = \{0, \ldots, N-1\} \tag{2.7}$$

If we separate the complex-valued inputs and outputs of the butterfly operation into real components, denoted by R subscript, and imaginary components, denoted by I subscript, the butterfly equations can be expressed as follows:

$$X_R(0) = x_R(0) + [x_R(1)W_R - x_I(1)W_I], \tag{2.8}$$

$$X_I(0) = x_I(0) + [x_I(1)W_R + x_R(1)W_I], \tag{2.9}$$

$$X_R(1) = x_R(0) - [x_R(1)W_R - x_I(1)W_I], \tag{2.10}$$

$$X_I(1) = x_I(0) - [x_I(1)W_R + x_R(1)W_I]. \tag{2.11}$$

2.4.7 Cyclic Extension and Removal

At the transmitter side, the transformed waveform should be prepended by a circular extension of itself to form a guide interval (GI) between every group of 64 samples. For this purpose, 16 last samples of every sequence of 64 transformed samples are cyclically extended at the beginning of the sequence. This creates a gap of GI=0.8μs between every group of 64 samples of duration 3.2 μs.

The resulting waveform is truncated afterwards by a windowing function defined

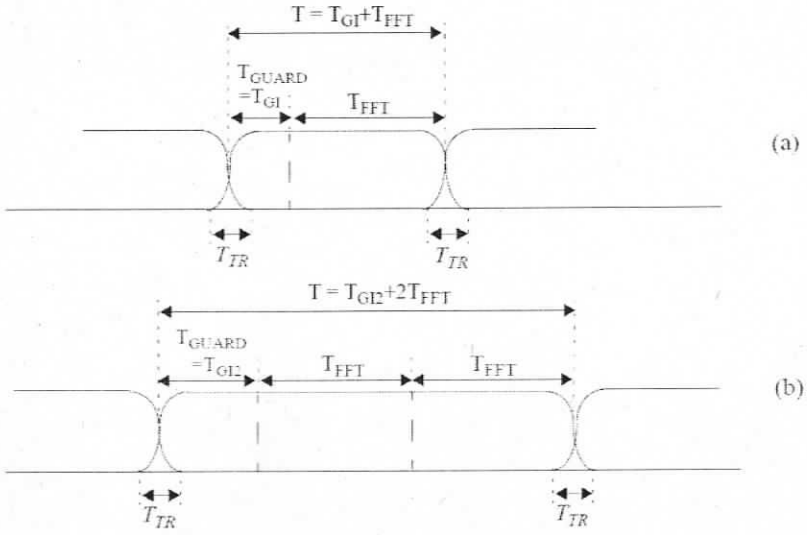

Figure 2.12: OFDM frame with cyclic extension and windowing for (a) single reception or (b) two receptions of the FFT period.

by

$$w_T[n] = w_T(nT_s) = \begin{cases} 1 & 1 \leq n \leq 79 \\ 0.5 & 0.80 \\ 0 & otherwise \end{cases} \tag{2.12}$$

where T_s is the sampling period (T_s= 50 ns). As it is shown in Figure 2.12.a., this truncation creates a small overlap of duration T_{TR} ($T_{TR} \simeq$ 100 ns) between OFDM symbols, leading to a smooth transition that is required to reduce the spectral sidelobes of the transmitted waveform.

Figure 2.12.b. illustrates the possibility of extending the windowing function over multiple FFT periods. This windowing function provides smoother transitions which is used for the transmission of long training sequence (LTS) in packet preamble.

The Cyclic Extension block of the transmitter performs the cyclic extension and windowing operations. The Cyclic removal block at the receiver side performs the reverse operation.

2.4.8 Packet Preamble

During transmission, a preamble should be appended at the beginning of each frame. This preamble contains a short training sequence (STS) of duration $T_{SHORT} = 8\mu s$ and a long training sequence (LTS) of duration $T_{LONG} = 8\mu s$.

The STS contains 10 short OFDM symbols[6] that are modulated by the elements of the sequence S, given by

$$\begin{aligned} S_{-26,26} = \sqrt{(13/6)} \times \{&0,0,1+j,0,0,0,-1-j,0,0,0,1+j,0,0,0,-1-j,0,0,0,\\ &-1-j,0,0,0,1+j,0,0,0,0,0,0,0,0,-1-j,0,0,0,-1-j,\\ &0,0,0,1+j,0,0,0,1+j,0,0,0,1+j,0,0,0,1+j,0,0\} \end{aligned}$$

The multiplication by a factor of $\sqrt{(13/6)}$ normalizes the average power of the resulting OFDM symbol, which utilizes 12 out of 52 subcarriers. The signal should be generated according to the following equation:

$$r_{SHORT}(t) = w_{TSHORT}(t) \times \sum_{k=-N_{ST}/2}^{N_{ST}/2} S_k e^{j2\pi k \Delta_F t} \tag{2.13}$$

where Δ_F is the subcarrier frequency spacing and equal to 0.3125 MHz. The fact that only spectral lines of $S_{-26,26}$ with indices that are a multiple of 4 have nonzero amplitude results in a periodicity of $T_{FFT}/4 = 0.8\ \mu s$. The interval T_{SHORT} is equal to ten 0.8 μs periods (i.e., 8 μs). The generated short training sequence is illustrated in Annex G of IEEE 802.11a standard [5].

The LTS contains 2 long OFDM symbols each consisting of 53 subcarriers (including a zero value at dc), which are modulated by the elements of the sequence L.

[6]Each of the STS OFDM symbols consists of 12 subcarriers.

given by

$$L_{-26,26} = \{1, 1, -1, -1, 1, 1, -1, 1, -1, 1, 1, 1, 1, 1, 1, -1, -1, 1, 1,$$
$$-1, 1, -1, 1, 1, 1, 1, 0, 1, -1, -1, 1, 1, -1, 1, -1, 1, -1,$$
$$-1, -1, -1, -1, 1, 1, -1, -1, 1, -1, 1, -1, 1, 1, 1, 1\}$$

A long OFDM training symbol shall be generated according to the following equation:

$$r_{LONG}(t) = w_{TLONG}(t) \times \sum_{k=-N_{ST}/2}^{N_{ST}/2} L_k e^{j2\pi k \Delta_F (t - T_{GI2})} \tag{2.14}$$

where $T_{GI2} = 1.6\ \mu s$.

Two periods of the long sequence are transmitted for improved channel estimation accuracy, yielding

$T_{LONG} = 1.6 + 2 \times 3.2 = 8 \mu s$.

The generated long training sequence is given in Annex G of the 802.11a standard [5].

2.4.9 Transmitter and Receiver Front End

During transmission, OFDM symbols generated by the PLCP sublayer enter the PMD sublayer where digital samples are first converted to analog domain and pass through a baseband filter. Then, analog streams are multiplied by sine and cosine waveforms to create the modulated intermediate frequency (IF) signals. The IF signal is brought up to RF by a local oscillator of frequency f_C and a multiplier. Finally, upconverted signal is passed through RF filter to be resided only in the approved bandwidth, amplified by a high power amplifier (HPA), and relieved into air [37]. Figure 2.13.a shows a high level block diagram of the transmitter front end.

During reception, the RF signal first passes through a low noise amplifier (LNA)

and then it is downconverted to IF. Based on the received signal, the synchronizer is adjusted for coarse frequency offset, and the automatic gain control amplifier (AGC) is adjusted to proper level. The receiver circuitry should detect the Preamble field in the received signal. For this purpose, a double sliding window is used that leverages the periodicity of the Preamble STS symbols [40]. This algorithm keeps two windows and checks the energy level. The first window observes the cross-correlation between the signal and its delayed version, and the second window observes the power level of the received signal. When there is a packet arrival, the ratio of the first to second observation shows a peak, otherwise it is approximately equal to zero since the cross-correlation of noise is zero. If the PMD sublayer continues to observe peaks, then it concludes that this is a frame arrival and starts decoding the signal[7]. After the packet detector provides an estimate of the start edge of the packet, the clock recovery circuit has the task of finding the precise edge of the OFDM symbol using the preamble training symbols [37]. Figure 2.13.b shows a high level block diagram of the receiver front end.

A mismatch in sampling instances of the digital to analog converter (DAC) and the analog to digital converter (ADC) causes a detection of slightly shifted version of the symbols that results in performance degradation. There are two approaches to do frequency offset estimation. The first approach uses the preamble short and long training symbols and the second approach utilizes the cyclic extension. However, frequency offset estimation can not perfectly eliminate the offset and still some residual frequency error remains which can rotate the constellation. Pilot carriers that are inserted symmetrically into every OFDM symbol are used to detect that residual frequency offset and adjust the receiver accordingly. Frequency offset estimator has an operating range which determines how large frequency offset can be estimated.

[7] Any high correlation can be expected to be the start of the first OFDM symbol, but the rest of the signal defines whether it is a valid preamble or not.

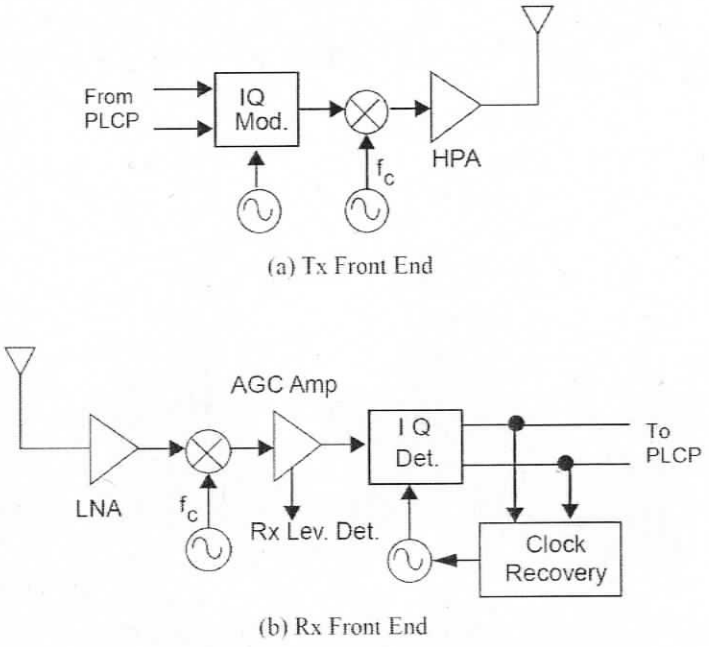

Figure 2.13: Front end of (a) the transmitter, (b) the receiver [5].

The limit can be defined as follows.

$$|f_\Delta| \leq \frac{N}{2D}\Delta_F \tag{2.15}$$

where $N = 64$ is the number of subcarriers, Δ_F=0.3125 MHz is the subcarrier frequency spacing, and D is the delay between the identical samples of two repeated symbols. For the short training symbols ($D = 16$), the limit is 625 KHz (coarse) and for the long training symbols ($D = 64$), the limit is 156.25 KHz (fine). In IEEE 802.11a, the standard specifies a maximum oscillator error of 20 parts per million (ppm) and if the channel is 5.3 GHz, then corresponding frequency offset is

$$|f_\Delta| = 40 \times 10^{-6} \times 5.3 \times 10^9 = 212KHz \tag{2.16}$$

which is 156.25 KHz (fine) < 212 KHz < 625 KHz (coarse). If initial frequency offset is very high, it may reduce the magnitude of peak and increase probability of

false alarm. A frequency offset of f_Δ amounts to peak reduction of

$$|\Sigma_{n=1}^{N} e^{j2\pi nTf_\Delta}| = \frac{sin(\pi NTf_\Delta)}{sin(\pi Tf_\Delta)} \tag{2.17}$$

In that case a peak detector can be implemented as a bank of correlators tuned to different frequency offset values.

2.4.10 Channel Estimation

WLAN systems are assumed to be stationary terminals in slowly time-varying environments. In such environments, the channel is highly correlated for the consecutive symbols, i.e., the channel can be considered non-varying during the transmission of one frame. Therefore, the channel can be estimated once at the beginning of each frame and that estimation is used to equalize the entire frame [37]. This type of estimation is called block type channel estimation. In IEEE 802.11a, the Preamble LTS consists of a guard interval and two long symbols that are known both at the transmitter and the receiver. Upon receiving these two symbols at the receiver, they are averaged and compared to the expected LTS. The result of this comparison shows the channel influence on the transmitted symbols. The averaging between two LTS symbols suppresses the channel noise and improves the quality of the channel estimation [33].

Let n^{th} sample of the t^{th} symbol at the output of the transmitter IFFT block be defined by

$$x(t,n) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X(t,k) e^{j\frac{2\pi}{N}kn}, \qquad 0 \leq n \leq N-1 \tag{2.18}$$

where N is the number of subcarriers ($N = 64$), and $X(t,k)$ is the IFFT complex input data of the k^{th} sub-carrier of symbol t. After cyclic extension is added, the OFDM symbol is transmitted through the multipath-fading channel corrupted by additive white Gaussian noise (AWGN). By using a guard interval longer than channel

impulse response (CIR), the received signal can be considered as the sum of AWGN and the convolution of the transmitted signal and CIR. The received signal then can be written as

$$y(t,n) = x(t,n) * h(n) + v(n) \tag{2.19}$$

where $h(n)$ represents the channel impulse response and $v(n)$ denotes the additive white Gaussian noise. Consequently, in the frequency domain the received signal of the k^{th} subcarrier of the t^{th} symbol can be written as

$$Y(t,k) = X(t,k) \times H(k) + V(k) \tag{2.20}$$

where $Y(t,k)$ represents the k^{th} output subcarrier of FFT in the t^{th} symbol. $V(k)$ denotes the FFT output of AWGN, $H(k)$ represents the channel frequency response on the k^{th} subcarrier, and $X(t,k)$ denotes the data subcarrier at the k^{th} subcarrier of the t^{th} symbol. Figure 2.14 shows the representation of the received signal at the input and output of the receiver blocks. We can rewrite Equation 2.20 as

$$X(t,k) = C(k) \times Y(t,k) + C(k) \times V(k) \tag{2.21}$$

Where $C(k) = 1/H(k)$ denotes the channel influence on the transmitted subcarrier k. To calculate $\hat{C}(k)$ for $1 \leq k \leq 64$, we use the known Preamble LTS symbols at the beginning of each frame, as it is shown in Equation 2.22 [34].

$$\hat{C}(k) = \frac{X(t_{LTS},k)}{Y(t_{LTS},k)} = \frac{X(t_{LTS},k) \times Y^*(t_{LTS},k)}{|Y(t_{LTS},k)|^2} \tag{2.22}$$

where $X(t_{LTS},k)$ and $Y(t_{LTS},k)$ respectively denote the known LTS data and the received LTS data for the k_{th} subcarrier of t^{th} LTS symbol. The $\hat{C}(k)$ corresponding to subcarrier k is calculated twice for the two symbols of LTS. The average of these two values is considered as the estimated channel coefficient for the corresponding

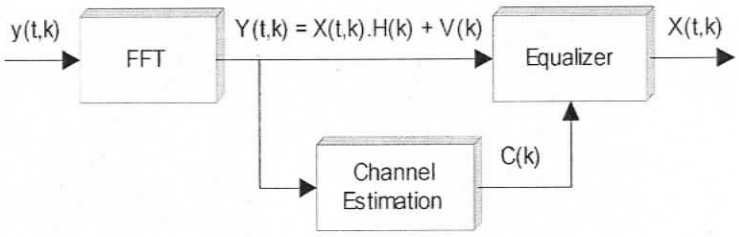

Figure 2.14: Channel Estimation and Equalization at the receiver [34].

subcarrier which is used to equalize the rest of the frame symbols.

2.4.11 Equalization

After receiving the LTS of preamble and estimating the channel behavior, a frequency domain equalizer uses the estimated channel model to eliminate intersymbol interference (ISI) within individual packets. The equalizer in OFDM systems does not need to cancel the ISI entirely, but to limit its length. Unlike a single carrier system in which equalization minimizes ISI, OFDM systems need to reduce it to a time span less than the length of guard interval and then remove the guard intervals after channel estimation and synchronization [37]. As long as the guard interval with duration T_G is longer than the channel impulse response (with duration T_h), there is no interference between the information symbols of successive blocks.

By neglecting channel noise in Equation (2.21), data on the k_{th} sub-carrier of the t^{th} symbol is equalized as follows.

$$\hat{X}(t,k) = \frac{Y(t,k)}{\hat{H}(k)} = \hat{C}(k) \times Y(t,k) \tag{2.23}$$

2.4.12 Phase Compensation

Upon receiving a packet, automatic frequency control (AFC) uses the Preamble symbols to compensate the carrier frequency offset (CFO). Unfortunately it is difficult for AFC to compensate the CFO completely and a residual CFO remains [35]. The

residual CFO causes phase distortion of the OFDM symbols in the frequency domain after FFT. Unlike amplitude distortion that is almost constant in one 802.11a frame[8], the phase distortion can change in a frame and needs to be compensated after channel equalization.

The pilot signals that are inserted between the data subcarriers of every OFDM symbol are utilized to track the amount of phase distortion during the frame transmission. In Section 2.4.5, we explained how these pilot signals are generated and positioned at the subcarrier locations K_P={-21, -7, 7, 21}. The rest of the subcarriers are data subcarriers located at positions K_D={-26, ..., -1, 1, ..., 26}. Since the pilot subcarriers are known, the phase distortion of OFDM symbol t, denoted by $\gamma(t)$, can be estimated as follows.

$$\gamma(t) = \frac{1}{4} \sum_{k \in K_p} arg(S(k), X(t,k)) \tag{2.24}$$

where $S(k)$ represents the expected pilot subcarrier k, and $X(t,k)$ represents the received pilot subcarrier k ($k \in K_p$). Using the estimated phase distortion $\gamma(t)$, we can calculate the compensated data subcarrier k of the symbol t as

$$D(t,k) = X(t,k) \times e^{-\gamma(t)} \tag{2.25}$$

Estimating the phase distortion first and compensating it later causes a long delay if done serially. The delay sometimes becomes fatal, preventing an ACK packet from being sent in time. To avoid such a long delay, it is desirable for the two operations to be done in parallel. A feedback type phase compensator realizes this. A feedback type phase compensator estimates the phase distortion by using the pilot subcarriers of the symbol that is phase compensated by the estimated phase distortions in previous

[8]The length of a frame in IEEE 802.11a is only a few milliseconds and amplitude distortion is almost constant during this short period.

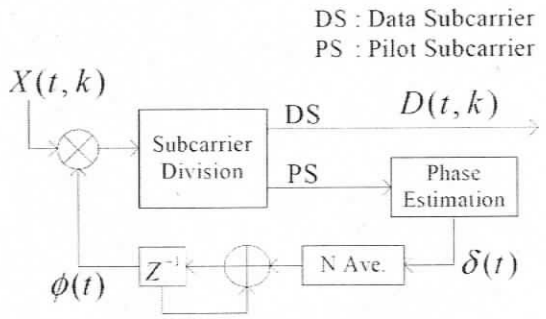

Figure 2.15: The block diagram of a feedback type phase compensation block [39].

symbols. Therefore, the estimated phase distortion $\delta(t)$ is calculated as

$$\delta(t) = \frac{1}{4} \sum_{k \in K_p} arg(S(k).D(t.k)) \tag{2.26}$$

$\delta(t)$ in 2.26 represents a phase distortion difference from one OFDM symbol to the next and it should be averaged before being accumulated to reduce the influence of AWGN. If the averaging number is chosen as N, the phase compensation value $\phi(t)$ is represented as follows.

$$\phi(0) = 0, \qquad \phi(t) = \phi(t-1) + \frac{1}{N} \sum_{i=t-N+1}^{t} \delta(i) \tag{2.27}$$

Then the phase compensation is executed as follows.

$$D(t.k) = X(t.k).e^{-\phi(t-1)} \tag{2.28}$$

The block diagram of the conventional phase compensation block implementing equations 2.26, 2.27, and 2.28 is shown in Figure 2.15

Chapter 3

Analysis of the IEEE 802.11a Hardware Architecture

In the previous chapter, we explained the data processing flow during transmission and reception, according to the IEEE 802.11a standard. Our objective is to map these processing steps onto a hardware architecture. The key design questions to be resolved are as follows:

- **Hardware requirements** – What hardware blocks and processing rates are required to perform necessary computations?
- **Hardware reuse** – Which hardware blocks can be reused for both transmission and reception?
- **Hardware reconfiguration** – Which hardware blocks should be flexible to support different operating modes?
- **Hardware pipelining** – How can hardware blocks be organized into pipeline stages to increase throughput?

In this chapter, we discuss the above mentioned questions in IEEE 802.11a transceiver architecture.

Hardware Block	Required Output Rate in Mode:								Units
	6-Mbps	9-Mbps	12-Mbps	18-Mbps	24-Mbps	36-Mbps	48-Mbps	54-Mbps	
Scrambler	6	9	12	18	24	36	48	54	Mbits/s
Coder	12	18	24	36	48	72	96	108	Mbits/s
Puncturer	12	12	24	24	48	48	72	72	Mbits/s
Interleaver	12	12	24	24	48	48	72	72	Mbits/s
Modulator	12	12	12	12	12	12	12	12	Msamples/s
Pilot/Zero Insertion	16	16	16	16	16	16	16	16	Msamples/s
IFFT	16	16	16	16	16	16	16	16	Msamples/s
Cyclic Extension	20	20	20	20	20	20	20	20	Msamples/s

Table 3.1: Main hardware blocks required for the transmitter.

Hardware Block	Required Output Rate in Mode:								Units
	6-Mbps	9-Mbps	12-Mbps	18-Mbps	24-Mbps	36-Mbps	48-Mbps	54-Mbps	
Cyclic Removal	16	16	16	16	16	16	16	16	Msamples/s
FFT	16	16	16	16	16	16	16	16	Msamples/s
Channel Estimation	16	16	16	16	16	16	16	16	Msamples/s
Equalizer	16	16	16	16	16	16	16	16	Msamples/s
Demodulator	12	12	24	24	48	48	72	72	Mbits/s
Deinterleaver	12	12	24	24	48	48	72	72	Mbits/s
Depuncturer	12	18	24	36	48	72	96	108	Mbits/s
Decoder	6	9	12	18	24	36	48	54	Mbits/s
Descrambler	6	9	12	18	24	36	48	54	Mbits/s

Table 3.2: Main hardware blocks required for the receiver.

3.1 Hardware Requirements

Tables 3.1 and 3.2 list the main hardware blocks required for the transmitter and receiver of the 802.11a PLCP sublayer. For each block, these tables also show the required rates of producing the output in eight operating modes, as specified by the 802.11a standard. Below we provide an explanation for these numbers.

During data transmission the allocated time window for an 80-sample OFDM symbol is 4.0 μs [5]. Consequently, the Cyclic Extension block must generate its output at the minimum rate of 20 Msample/s. To generate one group of 80 samples, the Cyclic Extension block needs 64 samples from the IFFT block; hence, the required output rate of the IFFT block is 16 Msamples/s. The Pilot/Zero Insertion block outputs one group of 64 samples, composed of 48 data samples from the Modulator block and 16 pilot/zero samples[1], every 4.0 μs. Therefore, the required output rate of the Pilot/Zero Insertion and Modulator blocks are 16 Msamples/s and 12 Msamples/s, respectively.

An individual sample produced by the Modulator block represents a mapping of

[1]Among 64 samples, 4 are allocated to pilot subcarriers and the other 12 samples are zero subcarriers

coded bits to one of the 48 subcarriers. The number of coded bits per subcarrier N_{BPSC} is given in Table 2.1. As 48 samples must be produced every 4.0 μs, the number of coded bits arriving every 4.0 μs must be at least $48 \times N_{BPSC}$. Thus, the required output rate of the Interleaver block that feeds the Modulator block is $12 \times N_{BPSC}$ Mbits/s. The Interleaver block rearranges the coded bits, without changing the number of bits passing through it. Therefore, the minimum input rate for the Interleaver block, which is the required output rate of the Puncturer block, is also $12 \times N_{BPSC}$ Mbits/s.

The Coder block is a fixed convolutional encoder with the coding rate 1/2, i.e., for every data bit the Coder block always generates 2 coded bits. Then, if necessary, the Puncturer block removes some of the coded bits to adjust the coding rate R for a given operating mode, as shown in Table 2.1. Note that R equals to the number of data bits divided by the number of coded bits. While the Puncturer block produces final coded bits, the Scrambler block produces data bits for the Coder block. Hence, the required output rate of the Scrambler block is $12 \times N_{BPSC} \times R$ Mbits/s, which is also equal to the target data rate of the corresponding mode.[2] Since the Coder block always generates 2 output bits per 1 input bit, the required output rate of the Coder block is $12 \times N_{BPSC} \times R$ Mbits/s.

During data reception the processing flow is reversed. The required output rates of the receiver blocks are shown in Table 3.2 and correspond to the input rates[3] of the transmitter counterparts. For example, the output rate of the Demodulator block is $12 \times N_{BPSC}$ Mbits/s, which is the input rate of the Modulator block.

[2]For example, in the 9-Mbits/s mode $N_{BPSC} = 1$ and $R = 3/4$, which yields $12 N_{BPSC} R = 9$.

[3]The input rate of each block is equal to the output rate of the preceding block. For example, the input rate of the IFFT block is 16 Msample/s, which is the output rate of the Pilot/Zero Insertion block.

3.2 Hardware reuse

Table 3.1 lists transmitter blocks while Table 3.2 lists the corresponding receiver counterparts. If some transmitter block can be reused as its receiver counterpart, the amount of hardware in the transceiver can be reduced. In this section we describe such blocks – Scrambler/Descrambler, Puncturer/Depuncturer, Interleaver/Deinterleaver, and IFFT/FFT.

Scrambler/Descrambler: According to the IEEE 802.11a standard [5], the same generator polynomial S(x) is used to scramble transmit data and to descramble receive data, where $S(x) = x^7 + x^4 + 1$. Therefore, the same hardware structure can serve as both Scrambler and Descrambler.

Puncturer/Depuncturer: The 802.11a standard specifies three coding rates R = 1/2, 2/3, and 3/4. Recall that the Puncturer follows the Coder with a fixed coding rate of 1/2. Therefore, if R = 1/2, the Puncturer block is bypassed. If R = 2/3, one out of every four coded bits is punctured: in particular, the fourth bit in every 4-bit group is omitted. If R = 3/4, two out of every six coded bits are punctured: in particular, the fourth and the fifth bits in every 6-bit group are omitted. During depuncturing, the corresponding omitted bits are replaced by dummy bits, and the decoder treats these dummy bits as perfect matches with zero error metric.

There are many ways to design a hardware structure that can perform both puncturing and depuncturing. One can use a bit counter and a buffer to store the punctured bit sequence. During puncturing the bits from the Coder block are sequentially written into the buffer as long as the counter value does not correspond to the bit number to be omitted. During depuncturing, the bits are sequentially read from the buffer as long as the counter value does not correspond to the omitted bit number.[4]

[4]For example, let $R = 2/3$. Every four clock cycles the counting sequence is (0, 1, 2, 3). During puncturing the buffer receives a write signal and increments its address pointer when the counter value is 0, 1, or 2; when the counter values is 3, the buffer is idle. During depuncturing the buffer

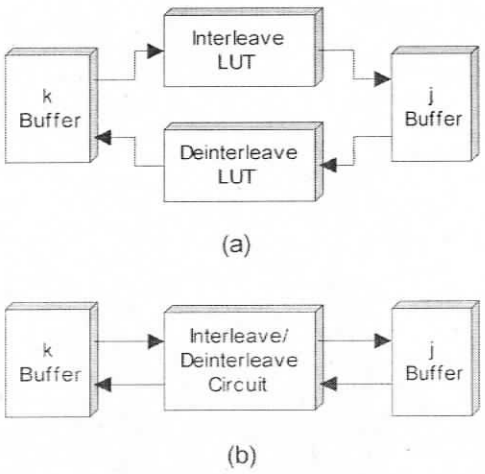

Figure 3.1: Interleaver/Deinterleaver design (a) using LUTs, (b) using address generator circuit.

Interleaver/Deinterleaver: Another hardware block that can be reused for both transmission and reception is the Interleaver/Deinterleaver. Recall the equations of interleaver and deinterleaver from Section 2.4.3, a simple approach would involve two different lookup tables (LUT) that generate the interleaving and deinterleaving addresses separately. Our approach, however, utilizes one circuit for both interleaving and deinterleaving.

Figure 3.1 illustrates these two approaches. Both designs require two RAM buffers, named k-Buffer and j-Buffer, that hold original bits and interleaved bits, respectively. The first approach, shown in Figure 3.1.a includes two LUTs of size $MAX\{N_{CBPS}\} = 288$. The proposed design, shown in Figure 3.1.b, includes a circuit with a counter counting from 0 to $(N_{CBPS} - 1)$, and an index generator implementing Equation (2.2). During interleaving, the counter provides a sequence of read addresses in increasing order for each bit in the k-Buffer, while the index generator provides the corresponding sequence of write addresses in interleaved order for each bit in the j-Buffer. During deinterleaving, the index generator provides a sequence of read addresses in interleaved order for each bit in the j-Buffer, while the counter

receives a read signal and increments its address pointer when the counter value is 0, 1, or 2; when the counter value is 3, a dummy zero-metric bit is supplied to the Decoder block.

provides the corresponding sequence of write addresses in increasing order (i.e., original deinterleaved order) for each bit in the k-Buffer. This design takes advantage of the fact that deinterleaving is an inverse of interleaving.[5]

IFFT/FFT: It is possible to perform IFFT with the same FFT block provided that the position of real and imaginary parts are exchanged before and after FFT operation and the result is normalized by the length of FFT operation (N=64) [43]. The exchange can be implemented by using simple multiplexor switches to control input and output interconnections. The normalization can be simply implemented by a shift register since division by 64 is six right shift operations.

3.3 Hardware reconfiguration

This section investigates the hardware blocks that need to be reconfigured according to the selected data rate. After the Modulator block has produced a sample during transmission, the sample processing is the same regardless of the mode, i.e., there is no need for the pilot insertion, IFFT, and cyclic extension operations to be flexible. Similarly, there is no need for the channel estimation, equalizer, FFT, and pilot removal operations to be flexible during multi-mode reception.

Except for the Scrambler/Descrambler, Coder, and Decoder, the operation of the remaining blocks are mode-dependent, i.e., the Puncturer/Depuncturer, Interleaver/Deinterleaver, Modulator, and Demodulator blocks must be flexible.

Puncturer/Depuncturer: Puncturing and depuncturing must support two coding rates: $R = 2/3$ for the 48-Mbits/s mode, and $R = 3/4$ for the 9/18/36/54-Mbits/s modes. Previously we described a simple structure with one buffer and one

[5]For example, in the 9-Mbits/s mode $k = 0, 1, 2, ..., 46, 47$ corresponds to $j = 0, 3, 6, ..., 44, 47$. During interleaving, bits are sequentially read from the k-Buffer locations 0, 1, 2, ..., 46, 47 and written into the j-Buffer locations 0, 3, 6, ..., 44, 47. During deinterleaving, bits are sequentially read from the j-Buffer locations 0, 3, 6, ..., 44, 47 and written into the k-Buffer locations 0, 1, 2, ..., 46, 47.

counter for puncturing and depuncturing. To make it flexible, we only need to alter the length of the counting sequence depending on R.

Interleaver/Deinterleaver: Interleaving and deinterleaving must support four pairs of values $\{N_{BPSC}, N_{CBPS}\}$ that affect interleaving and deinterleaving equations: $\{1, 48\}$ for the 6/9-Mbits/s modes, $\{2, 96\}$ for the 12/18-Mbits/s modes, $\{4, 192\}$ for the 24/36-Mbits/s modes, and $\{6, 288\}$ for the 48/54-Mbits/s modes. Previously we described a simple structure with two buffers, one counter, and one index generator for interleaving and deinterleaving. To make it flexible, we only need to alter the length of the counting sequence, and to switch between four different index generators depending on $\{N_{BPSC}, N_{CBPS}\}$.

Modulator: Modulation must support four schemes: BPSK for the 6/9-Mbits/s modes, QPSK for the 12/18-Mbits/s modes, 16-QAM for the 24/36-Mbits/s modes, and 64-QAM for the 48/54-Mbits/s modes. The simplest way to make the Modulator block flexible is to have four different circuits for each modulation scheme and switch between them depending on the mode.

Demodulator: Demodulation must support four schemes: BPSK for the 6/9-Mbits/s modes, QPSK for the 12/18-Mbits/s modes, 16-QAM for the 24/36-Mbits/s modes, and 64-QAM for the 48/54-Mbits/s modes. The simplest way to make the Demodulator block flexible is to have four different circuits for each demodulation scheme, and switch between them depending on the mode.

3.4 Hardware pipelining

If we can break transceiver datapath into pipeline stages, we can increase throughput and reduce dynamic power consumption. Table 3.3 lists the main datapath sections of a transceiver and the corresponding hardware blocks. Note that the datapath sections 1, 2, 4, and 7 containing the Scrambler/Descrambler, Coder/Decoder, In-

Datapath Section	Transceiver Operation		Flexible ?
	Transmission	Reception	
1	Scrambler/Descrambler		No
2	Coder	Decoder	No
3	Puncturer/Depuncturer		Yes
4	Interleaver/Deinterleaver		Yes
5	Modulator	Demodulator	Yes
6	Pilot/Zero Insertion	Channel Estimation & Equalizer	No
7	IFFT/FFT		No
8	Cyclic Extension	Cyclic Removal	No

Table 3.3: Sections of transceiver datapath.

terleaver/Deinterleaver, and IFFT/FFT blocks are the same for both transmission and reception. The remaining datapath sections 3, 5, 6, and 8 are split into two branches to be used either during transmission or during reception.[6] Also, note that the datapath sections 3, 4, and 5 must be flexible to support different data rate modes.

To break the transceiver datapath into several pipeline stages, we need to organize the pipeline so that:

- Different stages work on different data sets when the pipeline is full.
- During transmission a sample of an OFDM symbol is produced every clock cycle, and
- During reception a data bit is produced every clock cycle.

Our key design ideas to achieve these goals are as follows:

- Use different clock signals for different pipeline stages.
- Use dual-port RAMs as multi-bank interstage buffers for interfacing adjacent stages.

[6]For example, the datapath section 2 uses the branch with the Coder block during transmission, and switches to the branch with the Decoder block during reception.

- Depending on the selected mode, adjust clock frequencies accordingly, in addition to reconfiguring flexible datapath sections.

The need for multiple adjustable clocks naturally arises from considering, for example, datapath sections 1 (Scrambler/Descrambler) and 7 (IFFT/FFT). The Scrambler/Descrambler functionality is fixed, but its processing rate varies from 6 Mbits/s to 54 Mbits/s. To match these rate variations, we simply increase or decrease the clock frequency of the pipeline stage that contains the datapath section 1. On the other hand, the clock frequency of the pipeline stage containing the IFFT/FFT block is fixed because the IFFT/FFT processing rate is fixed. Thus, for two different pipeline stages we have two different clock signals, one of which is adjustable, while the other is fixed.

The interface between adjacent stages with different clocks must facilitate a synchronized data flow. The use of traditional handshaking for synchronization is not a good design choice as it requires high clock frequencies for all hardware blocks to match their processing rate. For example, the Coder which is a simple shift register should work with the same high clock frequency that is required for the FFT block. Applying high clock frequencies has the disadvantage of increasing the energy consumption. Instead of handshaking, we employ the dual-port RAM buffers with multiple banks. For example, consider an interface buffer with two banks between two pipeline stages. During transmission or reception, one stage always writes into this buffer while the other stage always reads from it. While the first stage fills the first bank, the second stage reads from the second bank previously written by the first stage. Next, the second stage will read from the first bank, while the first stage fills the second bank. For such a buffering scheme to work, the number of bits written by the first stage per unit time must match the number of bits read by the second stage per unit time.

There are two alternatives to our approach of using a multi-clock pipeline: (1)

single-clock pipelined design, and (2) multi-clock non-pipelined design.[7] The first alternative may suffer from overclocking and excessive buffering between adjacent pipeline stages with different processing rates. The second alternative may suffer from poor throughput and excessive handshaking between adjacent datapath sections with different clocks. Our approach overcomes these disadvantages – the details of our proposed design are presented in the next chapter.

[7]The other alternative, single-clock non-pipelined design, is inferior to single-clock pipelined design: the former must compensate its reduced throughput by using a higher clock frequency.

Chapter 4

Multi-Clock Pipeline Structure for the IEEE 802.11a PHY Layer Transceiver

In this chapter, we propose an architecture for the IEEE 802.11a PHY layer transceiver that addresses the following three issues:

- *Sufficient flexibility* to support eight different modes of operation.
- *Low power consumption* while processing data in each individual mode.
- *Low reconfiguration overhead* while switching from one mode to another.

We focus on the implementation of the digital hardware of the 802.11a transceiver, i.e., the PLCP sublayer. The PMD sublayer works with analog signals and its implementation is out of the scope of this work. In Section 4.1, we propose a hardware architecture that achieves the design goals discussed in Chapter 3. We present this architecture by considering data flow during transmission and reception. We describe the architecture of those transceiver blocks that are specifically designed to fit into our pipelined architecture in Section 4.2. Then, in Section 4.3 we elaborate the architectural features of our design. These features can be applied in designing other

WLAN transceivers as well. We discuss applicability of our design strategies to other WLAN standards in Section 4.4.

4.1 The IEEE 802.11a transceiver architecture

The proposed 802.11a transceiver architecture has four pipeline stages as shown in Figure 4.1. Each stage has its own clock signal. The frequencies of the Stage 1 clock CLK1 and Stage 2 clock CLK2 must be adjusted according to the selected mode. The frequencies of the Stage 3 clock CLK3 and Stage 4 clock CLK4 are fixed. During transmission, Stage 4 produces a sample of an OFDM symbol every cycle of CLK4. During reception, Stage 1 produces a data octet every cycle of CLK1.

There are three dual-port interstage buffers. Buffer1, between Stages 1 and 2 , has two 256×8-bit banks. Buffer2, between Stages 2 and 3, has five 64×64-bit banks and Buffer3, between Stages 3 and 4, has two 64×64-bit banks. The two ports of each buffer are denoted by Port A and Port B. During transmission, Port A is used for writing, and port B is used for reading. During reception, Port B is used for writing, and Port A is used for reading.

4.1.1 Transmitter operation

Stage 1: During transmission, data octets continuously enter the Input Buffer after receiving START-REQUEST signal from the MAC layer, until the END-REQUEST signal is received. The Input Buffer serially sends out received data to the Scrambler/Descrambler that performs scrambling. Note that before processing the data, the transmitter must prepare a header – during header processing, the Scrambler is bypassed. After scrambling, the Coder generates coded bits, and the Puncturer/Depuncturer performs mode-dependent puncturing. Stage 1 saves the resulting bits in the Buffer1 bank emptied last by Stage 2. That bank will be read by Stage 2

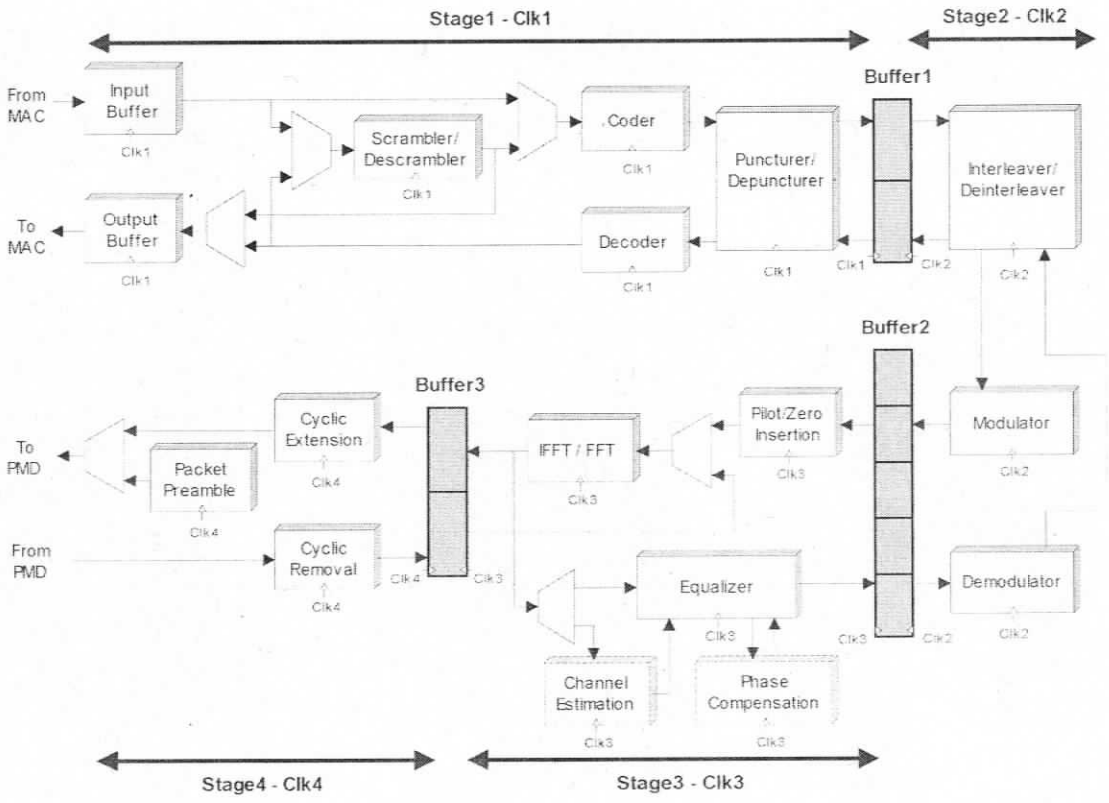

Figure 4.1: Multi-clock pipeline architecture of the 802.11a PLCP sublayer

when Stage 1 is filling the other bank.

Stage 2: The Interleaver/Deinterleaver reads data in the mode-dependent interleaved order from the Buffer1 bank filled last by Stage 1 (not currently used by Stage 1). The interleaved data bits are sent to the Modulator that maps them into a mode-dependent QAM constellation. After modulation, regardless of the mode, there will be 48 data samples (complex numbers) saved in one of the Buffer2 banks not currently used by Stage 3. Only two of the five available banks are alternately used during transmission.

Stage 3: The Pilot/Zero Insertion block inserts 4 pilots and 12 zero samples between 48 data samples of Buffer 2. The resulting group of 64 samples is passed to the 64-point IFFT/FFT block that performs the IFFT operation. The IFFT/FFT block has a pipeline structure as well, so that it does not stop the continuous flow of data. Stage 3 stores the 64-sample output of IFFT in the Buffer2 bank emptied last by Stage 4.

Stage 4: During transmission, the preamble and header must be appended at the beginning of data. When the first OFDM symbol of data is at the first stage of the IFFT pipeline, the PMD sublayer has already started reading the preamble. Once the preamble is completely transmitted, the PMD sublayer starts transmitting the header, previously processed and saved in the first bank of Buffer3. Meanwhile, the first OFDM symbol of the data is written into the second bank of Buffer 3, getting ready to be transmitted after the header. After the first OFDM symbol of the data, Stage 4 alternates between the first and the second bank of Buffer3, while saving the consecutive symbols. To form an 80-sample OFDM symbol, the Cyclic Extension block reads 64 IFFT samples from the Buffer3 bank filled last, cyclicly extends them by 16 samples, and performs windowing (see Section 2.4.7 for more details). During transmission, Stage 4 produces one sample of an OFDM symbol every cycle of its clock CLK4.

4.1.2 Receiver operation

Stage 4: During reception, the PMD sublayer detects the preamble and indicates it to the PLCP sublayer. The PLCP sublayer receives LTS of preamble, followed by SIGNAL and DATA symbols. For each received OFDM symbol, the first 16 samples (cyclic extension) are deleted, while the rest 64 samples are saved in one of the two (alternately used) banks of Buffer3.

Stage 3: The IFFT/FFT block reads 64 samples from the Buffer3 bank filled last by Stage 4, and performs the 64-point FFT operation. If the Fourier transformed samples are LTS of preamble, they will be sent to the Channel Estimation block. The Channel Estimation block compares the received LTS samples with expected LTS samples and generates 64 coefficients accordingly. These coefficients are saved in an internal RAM of the Channel Estimation block to be used for the equalization of the rest of the frame symbols. If the output of the IFFT/FFT block is not LTS, it enters the Equalizer block. The Equalizer multiplies one received sample by the corresponding coefficient (previously generated by the Channel Estimator) every clock cycle. The equalized sample is then sent to the Phase Compensation block that compensates the phase distortion of the received sample on the fly. 4 pilot signals that were inserted equally between 48 data samples are used in this block to calculate the phase compensation value. The phase compensator receives one sample every clock cycle, it refines it according to the previously received pilots, and sends the output back to the Equalizer. Equalizer outputs 64 refined samples in 64 clock cycles where only 48 of them are data samples. 48 data samples are written in a Buffer2 bank not currently used by Stage 2. Note that one bank of Buffer 2 will hold the header until the header information reaches the MAC layer (i.e., until the header exits Stage 1). While the header is processed by Stages 1 and 2, the data symbols are saved in the remaining four banks of Buffer2. Once the MAC layer receives the header, it identifies the operation mode, and the appropriate blocks in Stages 1 and 2

are reconfigured accordingly. Only after reconfiguration, the data samples can leave Buffer2 and enter Stage 2.

Stage 2: The Demodulator reads the Buffer2 bank holding the oldest unread data samples, and assigns soft-bits according to the mode-dependent modulation type. We use three bits to represent one soft-bit. The Interleaver/Deinterleaver writes these soft-bits in mode-dependent deinterleaved order in the Buffer1 bank not currently used by Stage 1.

Stage 1: The Depuncturer reads the soft-bits out of Buffer1 bank filled last by Stage 2, and inserts zero soft-bits in mode-dependent punctured locations. The resulting soft-bit stream is passed to the Decoder. The Decoder block has a pipeline structure as well, so that it does not stop the continuous flow of data. After decoding, data bits are descrambled by the Scrambler/Descrambler block, and written in the Output Buffer to be sent out to the MAC layer in octets. During reception, Stage 1 produces one data octet every clock cycle of its clock CLK1.

4.2 Hardware blocks for pipelined architecture

As discussed in Chapter 2, IEEE 802.11a defines the functionality of transceiver blocks without restricting their hardware implementation. This section describes the architecture of transceiver blocks that are specifically designed to fit into our pipelined architecture.

4.2.1 Pipelined IFFT/FFT block

The FFT block has a long processing delay, therefore it should be broken into pipeline stages to raise its throughput to the acceptable flow rate. The IFFT/FFT operation is performed on 64 samples and requires 192 butterfly operations as discussed in Section 2.4.6. To perform these 192 butterfly operations, we use 6 pipeline stages.

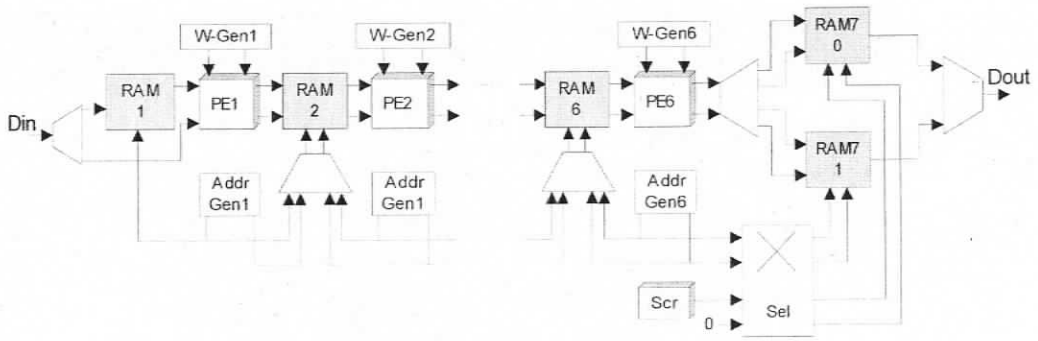

Figure 4.2: Pipelined architecture of the IFFT/FFT block.

There is one Processing Element (PE) per stage, performing one butterfly operation on 64 numbers over 32 clock cycles. There is one more stage for scrambling the output data. Figure 4.2 shows the structure of our IFFT/FFT block. The Addr-Gen blocks generate the proper addresses for reading and writing the real and imaginary sample parts stored in RAMs. The W-Gen blocks provide the corresponding coefficients $Re(W_N^{nk})$ and $Im(W_N^{nk})$. The size of RAM1 is 32×64 bits, while the others are 64×64-bit RAMs.

It takes 32 cycles to complete each stage of the IFFT/FFT pipeline. First, 32 complex numbers enter RAM1. In the following 32 cycles, these numbers and the next 32 numbers entering IFFT/FFT will enter PE1 in pairs, and the results will be saved in RAM2. The other stages work in a similar manner, as shown in Figure 4.3. In the last stage, data is saved in either of the two banks of RAM7, i.e. RAM70 or RAM71, and it will be read out in a scrambled order using the addresses that are generated by the local scrambler. Once the IFFT/FFT pipeline becomes full, an output sample is produced every clock cycle.

4.2.2 Pipelined Decoder block

We use the Viterbi algorithm [46] for decoding. The three major components in the Viterbi decoder are shown in Figure 4.4. They are the Branch Metric Unit (BMU),

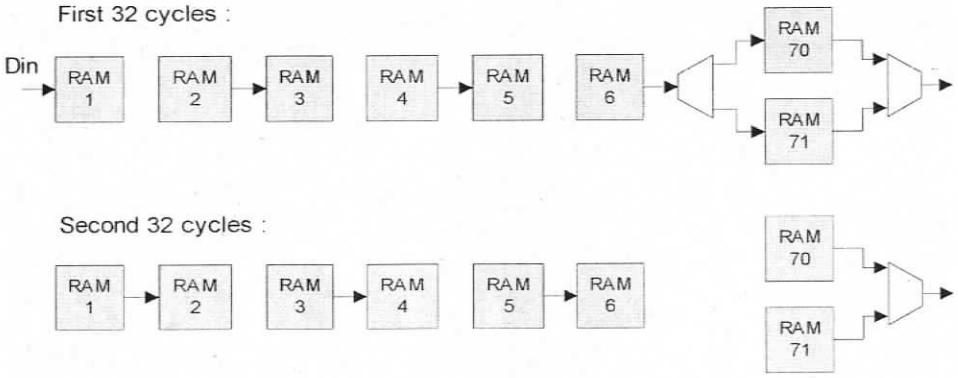

Figure 4.3: Flow of data in pipeline at the first and second 32 cycles of IFFT/FFT.

Add-Compare-Select (ACS), and Survivor Path Unit (SPU).

The BMU receives the input symbols and computes the branch metrics. According to the value of Soft/Hard control signal, the computation is performed on soft symbols (3-bits) or hard symbols (1-bit). The MAC layer decides about the soft/hard computations, exploring the tradeoff between decoded data reliability and decoder power consumption.

The ACS block contains 32 ACS_n blocks where $n = 1, 2, ..., 64$. Each ACS_n block receives two pairs of path metrics and branch metrics, and computes the next state of path metrics. It also provides the 64 decision bits to the SPU on every clock cycle.

The SPU saves the survivor paths every clock cycle. The survivor path converges to the correct value after a certain number of stages. The number of SPU stages, or length L, is adjustable in our design between four different values. Although not required by the 802.11a standard, we have decided to make L flexible to control the tradeoff between the high data reliability (large L) and low power consumption (small L). This idea of adjustable L originates from [47]. The MAC layer decides about the SPU length L.

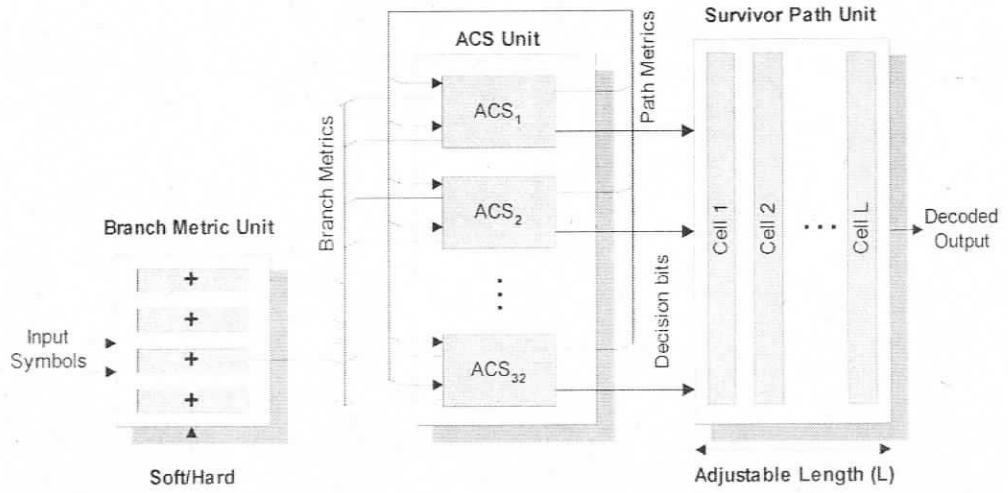

Figure 4.4: Adjustable Viterbi Decoder [47].

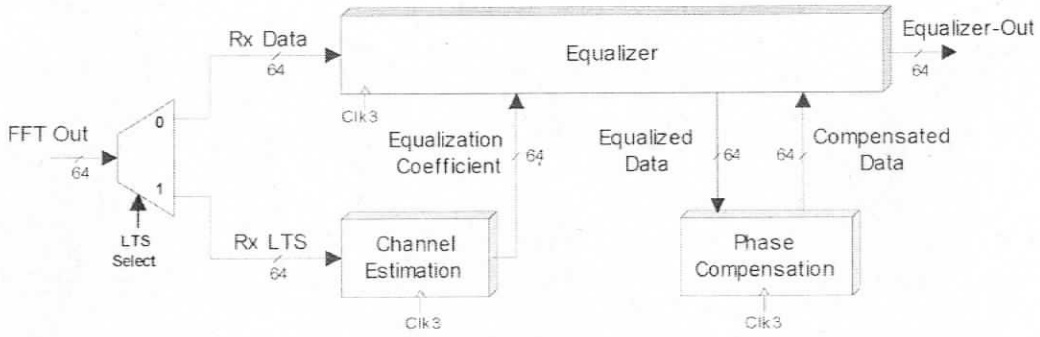

Figure 4.5: The interface between the Equalizer, Channel Estimation, and Phase Compensation blocks.

4.2.3 Equalizer, Channel Estimation, and Phase Compensation

In our proposed architecture, the Equalizer block works in parallel with the Channel Estimation and Phase compensation blocks. Our design does not implement the Channel Estimation and Phase Compensation blocks and provides the interface to attach these blocks externally. This interface is shown in Figure 4.5. During LTS reception, a control signal called LTS-Select is 1, leading the FFT output to the Channel Estimation block. The FFT output at this time is the received LTS, called Rx-LTS in Figure 4.5, which is used to compute the channel coefficients for the

corresponding frame[1]. The estimated coefficients (Equalization-Coefficients) will be saved in an internal buffer of the Channel Estimation block. After LTS reception the LTS-Select becomes 0, and the Channel Estimation block outputs one 64-bit Equalization Coefficient every clock cycle. When LTS-Select is 0, the FFT output subcarriers are sent to the Equalizer block. The FFT output at this time is the Signal or Data field of the frame, called Rx-Data in Figure 4.5. The Equalizer multiplies the Rx-Data subcarriers by the corresponding Equalization-Coefficients. Next, the equalized data subcarriers enter the Phase Compensation block through a 64-bit bus, called Equalized-Data. The Phase Compensation block uses the pilot subcarriers of the Equalized-Data to compensate the phase distortion. The phase compensation takes one clock for every subcarrier using the hardware block introduced in [39].[2] The block diagram of this phase compensator is shown in Figure 4.6. This block diagram is a slight modification of the conventional designs (see Section 2.4.12) as it utilizes a coefficient α ($0 < \alpha \leq 1$) that prevents the vibration of phase distortion values. The Phase Estimation block in this design works differently to remove the residual phase distortion completely. The reader is referred to [39] for more details about this hardware. After Phase compensation is performed, the corrected subcarriers are sent back to the Equalizer block through a 64-bit bus called Compensated-Data. The Equalizer block has to wait 3 clock cycles to receive the corrected data and write it into Buffer2.

[1]The Channel Estimation block implements Equation 2.22 as explained in Chapter 2. Implementing this operation with a short delay and low power consumption is a challenging task since it requires CoOrdinate Rotation DIgital Computer (CORDIC) algorithm [38]. The channel estimator in [28] is an example of a low-latency ASIC implementation of this block that can be used in our design.

[2]If a Phase Compensation block with higher latency is employed, the Equalizer block in our design should extend its waiting clock cycles.

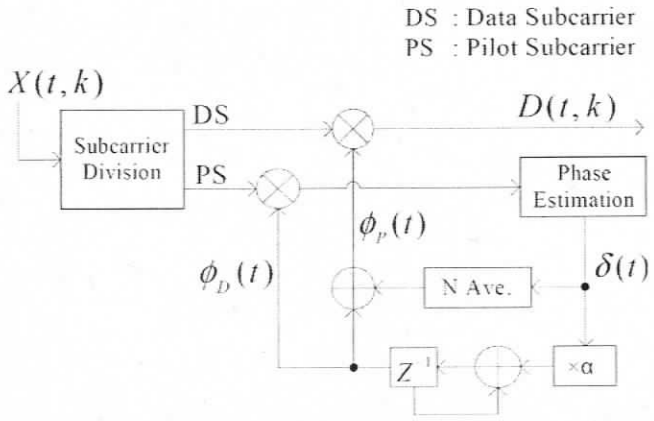

Figure 4.6: The phase compensation block diagram [39].

4.3 Architectural features

In this section, we elaborate the architectural features of our design that are applicable to other WLAN standards as well. Table 4.1 summarize the key performance parameters in terms of the number of clock cycles. These cycle counts, which are independent of a specific implementation platform (e.g., FPGA, ASIC, etc.), define the minimum acceptable clock frequency to reach the IEEE 802.11a standard data rates. To achieve the target data rates specified by the standard, each hardware block should generate its output unit at a certain rate (see Table 3.1 and 3.2) in Chapter 3. Putting the hardware blocks with the same output unit rate in a category, four categories of blocks are distinguished which correspond to our four pipeline stages. To follow the standard data rates, the hardware blocks corresponding to each pipeline stage should generate outputs with a minimum frequency as it is shown in Table 4.1. In our proposed architecture, the hardware blocks of each pipeline stage work with this minimum frequency, except for Stage 3 which works with a slightly higher clock frequency [3]. Table 4.2 shows the minimum clock frequency of each pipeline stage in our proposed architecture.

[3]IFFT block in stage 3 generates one output unit (64 samples) every 72 cycles. To ensure standard's predefined data rate (16 Msample/s), the clock frequency should be $72/64 \times 16 = 18$ MHz.

Stage	Minimum Output Rate (MHz) in Mode:							
	6-Mbps	9-Mbps	12-Mbps	18-Mbps	24-Mbps	36-Mbps	48-Mbps	54-Mbps
1	6	9	12	18	24	36	48	54
2	12		24		48		72	
3	16							
4	20							

Table 4.1: Minimum output rates of hardware blocks in each pipeline stage according to the IEEE 802.11a standard.

Clock	Minimum Frequency (MHz)	**Comment:**
CLK1	$\phi_1^{min} = 1 \times 12 \times N_{BPSC} \times R$	Stage 1 processes one bit every clock cycle, and its required throughput is $12N_{BPSC}R$ Mbits/s
CLK2	$\phi_2^{min} = N_{BPSC} \times 12$	Stage 2 processes one sample every N_{BPSC} cycles, and its required throughput is 12 Msamples/s
CLK3	$\phi_3^{min} = 72/64 \times 16 = 18$	Stage 3 processes 64 samples every 72 cycles, and its required throughput is 16 Msamples/s
CLK4	$\phi_4^{min} = 1 \times 20$	Stage 4 processes one sample every clock cycle, and its required throughput is 20 Msamples/s

Table 4.2: Minimum clock frequencies of hardware blocks corresponding to each pipeline stage required by the proposed architecture.

The maximum frequencies, however, are limited by the critical path delays, which are implementation-dependent. These maximum frequencies will be given in Chapter 6 for the FPGA prototype of our architecture. The difference between the maximum frequency and the minimum frequency for a given stage can be used to scale the supply voltage accordingly, thus reducing the dynamic power consumption of that stage.

4.3.1 Four-stage pipeline structure

The pipeline structure of the proposed architecture brings the following benefits:

- **Increased throughput:** Pipelining yields increased throughput, as different pipeline stages can process different data sets at the same time. Ideally, once

pipeline becomes full, it should produce a valid output every clock cycle. In our architecture, during transmission, Stage 4 produces one sample of an OFDM symbol every clock cycle and during reception, Stage 1 produces one bit of a data octet every clock cycle. In non-pipelined designs, it may take several clock cycles to produce a valid output. Therefore, our pipelined structure can process more data over a fixed number of clock cycles in comparison to non-pipelined designs. In other words, the total number of clock cycles needed to transmit or receive a frame is reduced when using a pipelined structure.

- **Reduced power consumption:** Pipelining can be used to reduce dynamic power consumption. Since the 802.11a standard fixes required throughput, fewer number of clock cycles translate into lower clock frequencies that are required to meet the throughput targets. Lower clock frequencies imply longer clock periods that can be tolerated, which increases the available delay slack that can be utilized by voltage scaling. Table 4.2 shows that the clock frequencies required by our architecture are relatively low. Ideally, the implementation-dependent critical path delays would be much smaller than the corresponding required clock periods, so that one can aggressively scale the supply voltage down to save dynamic power. The potential power savings will be given in Chapter 6 for the FPGA prototype of our architecture.

4.3.2 Multiple adjustable clocks

Our architecture has four clocks shown in Table 4.2, among which CLK3 and CLK4 are respectively fixed at 18 MHz and 20 MHz, and CLK1 and CLK2 are respectively adjustable depending on $N_{BPSC}R$ and N_{BPSC}. By using these clocks, we avoid excessive buffering and handshaking. For example in Stage 2, instead of having an individual input buffer and output buffer for the Interleaver/Deinterleaver, Modulator

and Demodulator blocks, data is only buffered in Buffer 2 and Buffer 3. Therefore, not only buffering area and power is reduced, but also the handshaking cycles to communicate between individual blocks are saved. As an instance, in the former example only stage 2 and stage 3 need to perform handshaking not all the individual blocks inside them.

The main disadvantage of using multiple adjustable clocks is the need for a flexible clock generator. For example in our proposed architecture, we need to generate 10 different clock frequencies: 6, 9, 12, 18, 24, 20, 36, 48, 54, and 72 MHz, as shown in Table 4.2. However, these clock frequencies are multiples of three master clocks and they can be generated by a simple frequency divider circuit. The three master clocks are 108 MHz (covering 72, 36, 18, 9 ,and 54 MHz), 48 MHz (covering 48, 24, 12, and 6 MHz), and 20 MHz.

4.3.3 Eight different modes of operation

The 802.11a standard specifies eight different modes of operation corresponding to eight different rates. Mode selection affects puncturing and depuncturing, interleaving and deinterleaving, as well as modulation and demodulation. In our architecture, the Puncturer/Depuncturer, Interleaver/Deinterleaver, Modulator, and Demodulator blocks are reconfigurable and can be adapted to the selected mode. Their reconfiguration is fairly simple and outlined in Section 3.3.

Our architecture does not support other modes that may arise from different WLAN standards (e.g., 802.11b, 802.11g, 802.11n, HYPERLAN/2, etc). However, this disadvantage positively affects the efficiency of operation in each of the supported modes. In general, the performance and power consumption of a more specialized design are better than those of a more flexible design. Our architecture is specialized for the 802.11a standard in order to improve its efficiency.

4.3.4 Efficient frame-by-frame adaptation

One of the main advantages of our architecture is its low-overhead reconfigurability that allows for efficient frame-by-frame adaptation. There is a special 3-bit mode signal indicating which mode is selected by the MAC layer. This signal controls the frequency adjustments of clocks CLK1 and CLK2, and reconfigures the appropriate hardware blocks. The reconfiguration mechanism is as follows:

- During transmission, the 24-bit header enters the pipeline first, starting at Stage 1. According to the 802.11a standard, the header is always processed in the 6-Mbits/s mode, therefore the pipeline is initially working in that mode. When header information are saved in the first bank of the Buffer 2 and start passing through the Stage3, the MAC layer informs the pipeline controller of the desired mode, so that the configuration and clocks of Stages 1 and 2 can be changed accordingly. After frequency adjustment and hardware reconfiguration are complete, the pipeline starts processing the data according to the selected mode.

- During reception, the header and the data continuously enter the pipeline, starting at Stage 4. While header goes through Stages 1 and 2 (configured for the 6-Mbits/s mode), the incoming data is temporarily saved in Buffer 2. Once the MAC layer receives header, it informs the pipeline controller of the mode. Then, the configuration and clocks of Stages 1 and 2 are changed accordingly, and data is allowed to proceed through these stages.

The reconfiguration overhead is very low in our architecture. The frequency dividers do not spend much effort to produce a new clock signal, i.e., clock adjustments do not result in long delays or high energy consumption. Hardware changes do not require any downloads of configuration bitstreams, i.e., hardware reconfigurations do not result in long delays or high energy consumption. Consequently, the pipeline

adaptation to a different mode is not a costly operation, and it can be performed efficiently on a frame-by-frame basis.

4.4 Extension to other WLAN standards

Even though the proposed architecture specifically targets the 802.11a transceiver, many of our design ideas are applicable to other WLAN standards. For example, HIPERLAN/2 [4] is very similar to 802.11a at the physical layer; The differences between the two standards are minimal and reside in the scrambler initializations, the supported data rates, the form of preambles, etc. Our design is also compatible with another important alternative, the IEEE 802.11g standard [8] since it employs a similar OFDM basis as 802.11a, supporting the data rate up to 54 Mbits/s. The major difference between 802.11g and 802.11a is that IEEE 802.11g works at 2.4 GHz instead of 5 GHz which makes the implementation of PMD sublayer different than PMD sublayer of 802.11a. The upcoming WLAN standard 802.11n [10] is an evolution of both 802.11a and 802.11g. It uses Multiple-Input/Multiple-Output (MIMO) technology. However, OFDM is the basic channel coding and transport mechanism in 802.11n as well [10]. Hence, similar design strategies may be used to pipeline multiple antenna chains of the 802.11n transceiver.

Chapter 5

Latency Analysis of the Proposed Architecture

The latency of the PHY layer is a critical parameter for the WLAN systems. The pipeline structure of the proposed 802.11a architecture introduces a short latency in terms of the number of required clock cycles, since the only latency overhead imposed by this architecture is the latency of the inter-stage buffers. The short latency of the PHY layer leaves a longer time interval for the MAC layer to perform its operations. In this chapter, we derive the latency of the transmitter and receiver analytically and we express them in terms of the number of clock cycles, i.e., the latency values are independent of the implementation platform (FPGA, ASIC,etc). In chapter 6, we validate our latency equations by comparing their numerical values with the simulation results.

5.1 The latency of the IEEE 802.11a transmitter and receiver

Tables 5.1, 5.2, 5.3, 5.4 show the output units of the hardware blocks in pipeline stages 1, 2, 3, and 4 respectively, and the latency to generate one of those output units. Table 5.5 shows the latency of inter-stage buffers, i.e., the number of required

clock cycles to write or read each buffer port.

We define the latency of the transmitter as the time between the first bit of data entering the transmitter and the last sample of the PLCP frame leaving the PLCP sublayer. We similarly define the latency of the receiver as the time between the first sample of data being received by the PLCP sublayer and the last bit of data entering the MAC layer. The latency here is the delay of processing one OFDM symbol, and it is different from the delay of filling/flushing the pipeline. For example, during transmission the latency of filling the pipeline is the delay between the first bit entering the pipeline and the first sample leaving the pipeline[1]. After this delay, we get an output every clock cycle, i.e., after the pipeline filling latency, we need to wait for the entire samples of one OFDM symbol to get out of the pipeline.

The equations for calculating the transmitter latency and receiver latency are derived in the following section.

5.1.1 Transmitter latency

During transmission, the latency of each pipeline stage in terms of the number of its corresponding clock cycles is as follows.

Stage 1 latency:

$$N_{S1-Tx} = N_{DBPS} + N_{InBuf} + N_{Scr/Descr} + N_{Cod} + N_{Punc/DePunc} + 1 \qquad (5.1)$$

It should be noted that according to Table 5.5, it takes N_{DBPS} clock cycles for Buffer 1 to write one of its banks. We do not consider those clock cycles in the latency summation of Stage 1, since writing into Buffer 1 overlaps with entering data into Stage 1 that is already considered in the latency equation. The same reasoning is applied to not consider Buffer 2 and Buffer 3 write latencies during transmission.

[1]The latency of flushing the pipeline is the delay between the last bit entering the pipeline and the last sample leaving the pipeline.

Blocks	Output unit	Latency in terms of number of CLK1 cycles	**Comment:**
Input Buffer	1 bit	$N_{InBuf} = 1$	The Input Buffer starts sending out data bits in the next clock cycle after it receives its input data octet.
Output Buffer	8-bit octet	$N_{OutBuf} = 8$	The Output Buffer sends out one 8-bit octet 8 clock cycles after it receives its first data bit.
Scrambler/ Descrambler	1 bit	$N_{Scr/Descr} = 1$	The Scrambler/Descrambler sends out data in the next clock cycle after it receives its input data.
Coder	2 bits	$N_{Cod} = 1$	The Coder starts sending out data the next clock cycle after it receives its first input data bit.
Decoder	1 bit	$N_{Dec} = L$	L is the latency of the Decoder that can be 64, 72, 80, 88 in our design.
Puncturer	$1/R$ bits	$N_{Punc/Depunc} = 2$	$R = 1/2$, 2/3, or 3/4.
Depuncturer	2 bits	$N_{Punc/Depunc} = 2$	Depuncturer needs 2 clock cycles to insert zeros in punctured positions (if required) and generate 2 output bits.

Table 5.1: Latency of Stage 1 hardware blocks in terms of required CLK1 cycles.

Blocks	Output unit	Latency in terms of number of CLK2 cycles	**Comment:**
Interleaver/ Deinterleaver	1 address	$N_{Int/Deint} = 1$	It takes 1 clock cycle to generate interleved addresses.
Modulator	1 sample	$N_{Mod} = N_{BPSC}$	It takes N_{BPSC} clock cycles for the Modulator to read N_{BPSC} input bits and generate one output sample.
Demodulator	N_{BPSC} bits	$N_{Demod} = N_{BPSC}$	Demodulator should wait N_{BPSC} clock cycles before processing the next input sample.

Table 5.2: Latency of Stage 2 hardware blocks in terms of required CLK2 cycles.

as well as Buffer 2 and Buffer 1 write latencies during reception.

Stage 2 latency:

$$N_{S2-Tx} = N_{Buf1B} + 1 + N_{Mod} + 1 \tag{5.2}$$

Stage 3 latency:

$$N_{S3-Tx} = N_{Buf2B} + N_{Pilot-ZeroIns} + (N_{IFFT/FFT} - 64) + 1 \tag{5.3}$$

It takes 64 clock cycles to input data into IFFT. These 64 clock cycles overlap with reading data from Buffer 2, therefore they are subtracted from IFFT number of clock cycles.

Stage 4 latency:

$$N_{S4-Tx} = N_{Buf3B} + N_{Cyclic-Ext} \tag{5.4}$$

To calculate the latency in terms of time units, we need to divide the latency in terms of clock cycles by the frequency of the corresponding clock cycle. Therefore,

Blocks	Output unit	Latency in terms of number of CLK3 cycles	**Comment:**
Pilot/Zero Insertion	1 samples	$N_{Pilot/ZeroIns} = 1$	It takes one clock cycle for Pilot-Zero Insertion block to generate its output sample.
Channel Estimation	1 coefficient	$N_{ChEst} = 1$	Channel Estimator reads one sample out of its RAM every clock cycle.
Phase Compensation	1 sample	$N_{PhComp} = 3$	Phase compensation block needs 1 cycle to read input sample, 1 cycle to compensate phase distortion, and 1 cycle to output the result.
Equalizer	1 sample	$N_{Equal} = 5$	Equalizer needs 1 cycle to equalize the input sample, 1 cycle to pass it to the Phase compensation block, 2 cycle to receive the phase compensated sample, and 1 cycle to send it out.
IFFT/FFT	64 samples	$N_{IFFT/FFT} = 64 + (32 \times 6) + 64 + 33 = 353$	IFFT/FFT block needs 64 cycles to read 64 input samples, 32×6 cycles to pass data through 6 butterfly stages, 64 cycles to scramble output data and overally 33 extra cycles for the gaps required between every 64 input samples (These gaps are required by the FFT controller).

Table 5.3: Latency of Stage 3 hardware blocks in terms of required CLK3 cycles.

Blocks	Output unit	Latency in terms of number of CLK4 cycles	**Comment:**
Cyclic Extension	1 sample	$N_{CyclicExt} = 1$	Cyclic Extension block outputs 1 sample every clock cycle.
Cyclic Removal	1 sample	$N_{CyclicRem} = 1$	Cyclic Removal block outputs 1 sample every cycle.

Table 5.4: Latency of Stage 4 hardware blocks in terms of required CLK4 cycles.

Interstage Buffer	Port A		Port B		**Comments:**
	Clock	# Cycles	Clock	# Cycles	
Buffer 1	CLK1	$N_{Buf1A} =$ N_{DBPS}	CLK2	$N_{Buf1B} =$ N_{CBPS}	N_{CBPS}. and N_{DBPS} depend on the mode – see Table 2.1.
Buffer 2	CLK2	N_{Buf2A}=48	CLK3	N_{Buf2B}=64	48 and 64 are the number of data samples in one OFDM symbol before and after pilot/zero insertion respectively.
Buffer 3	CLK3	N_{Buf3A}=64	CLK4	N_{Buf3B}=80	64 and 80 are the length of OFDM symbol before and after cyclic extension respectively.

Table 5.5: Number of clock cycles required to write/read one buffer bank.

the overall latency of the transmitter can be calculated by

$$Tx\text{–}Latency = \frac{N_{S1-Tx}}{\phi_1^{min}} + \frac{N_{S2-Tx}}{\phi_2^{min}} + \frac{N_{S3-Tx}}{\phi_3^{min}} + \frac{N_{S4-Tx}}{\phi_4^{min}} \tag{5.5}$$

Table 4.2 shows the value of ϕ_1^{min}, ϕ_2^{min}, ϕ_3^{min}, and ϕ_4^{min} that represent the clock frequencies of Stage 1, 2, 3, and 4 respectively.

5.1.2 Receiver latency

During reception, the latency of each pipeline stage in terms of the number of its corresponding clock cycles is as follows.

Stage 4 latency:

$$N_{S4-Rx} = N_{Buf3B} + N_{Cyclic-Rem} \tag{5.6}$$

Stage 3 latency:

$$N_{S3-Rx} = N_{Buf3A} + (N_{IFFT/FFT} - 64) + N_{Equal} + 1 \tag{5.7}$$

It takes 64 clock cycles to input data into IFFT. These 64 clock cycles overlap with reading data from Buffer 3, therefore they are subtracted from FFT number of clock cycles. Also, the latency of Channel-estimation and Phase-Compensation blocks are not considered in latency summation of Stage 3, since they work in parallel with Equalizer block and their active clock cycles overlap with the ones of the Equalizer block.

Stage 2 latency:

$$N_{S2-Rx} = N_{Buf2A} + N_{Demod} + N_{Int/Dcint} + 1 \tag{5.8}$$

Stage 1 latency:

$$N_{S1-Rx} = N_{Buf1A} + N_{Punc/DePunc} + N_{Dec} + N_{Scr/Descr} \tag{5.9}$$

The active clock cycles of Output Buffer, N_{OutBuf}, overlaps with entering data into Stage 1, and we do not consider them in latency summation of Stage 1.

The overall latency of the receiver can be calculated by

$$Rx\text{-}Latency = \frac{N_{S1-Rx}}{\phi_1^{min}} + \frac{N_{S2-Rx}}{\phi_2^{min}} + \frac{N_{S3-Rx}}{\phi_3^{min}} + \frac{N_{S4-Rx}}{\phi_4^{min}} \tag{5.10}$$

Table 4.2 shows the value of ϕ_1^{min}, ϕ_2^{min}, ϕ_3^{min}, and ϕ_4^{min} that represent the clock frequencies of Stage 1, 2, 3, and 4 respectively.

5.2 Numerical latency values

Replacing ϕ_1^{min}, ϕ_2^{min}, ϕ_3^{min}, and ϕ_4^{min} in equations 5.5 and 5.10 by the value of clock frequencies defined in Table 4.2, we obtain the numerical values of transmission and reception latency in our architecture as it is shown in Table 5.6. As an example, calculating the transmission and reception latency in mod 6-Mbits/s is shown in Equation 5.11 and 5.12 respectively.

$$
\begin{aligned}
Tx\text{–}Latency =& \frac{N_{S1-Tx}}{\phi_1^{min}} + \frac{N_{S2-Tx}}{\phi_2^{min}} + \frac{N_{S3-Tx}}{\phi_3^{min}} + \frac{N_{S4-Tx}}{\phi_4^{min}} \\
=& \frac{N_{DBPS} + N_{InBuf} + N_{Scr/Descr} + N_{Cod} + N_{Punc/DePunc} + 1}{\phi_1^{min}} + \\
& \frac{N_{Buf1B} + 1 + N_{Mod} + 1}{\phi_2^{min}} + \\
& \frac{N_{Buf2B} + N_{Pilot-ZeroIns} + (N_{IFFT/FFT} - 64) + 1}{\phi_3^{min}} + \\
& \frac{N_{Buf3B} + N_{Cyclic-Ext}}{\phi_4^{min}} \\
=& \frac{24 + 1 + 1 + 1 + 2 + 1}{1 \times 12 \times 1 \times (1/2)} + \frac{48 + 1 + 1 + 1}{1 \times 12} + \\
& \frac{64 + 1 + 353 - 64 + 1}{18} + \frac{80 + 1}{20} \\
=& 5.00 + 4.25 + 19.72 + 4.05 \\
=& 33.02 \mu s
\end{aligned} \tag{5.11}
$$

$$
\begin{aligned}
Rx\text{–}Latency =& \frac{N_{S1-Rx}}{\phi_1^{min}} + \frac{N_{S2-Rx}}{\phi_2^{min}} + \frac{N_{S3-Rx}}{\phi_3^{min}} + \frac{N_{S4-Rx}}{\phi_4^{min}} \\
=& \frac{N_{Buf1A} + N_{Punc/DePunc} + N_{Dec} + N_{Scr/Descr}}{\phi_1^{min}} + \\
& \frac{N_{Buf2A} + N_{Demod} + N_{Int/Deint} + 1}{\phi_2^{min}} + \\
& \frac{N_{Buf3A} + (N_{IFFT/FFT} - 64) + N_{Equal} + 1}{\phi_3^{min}} + \\
& \frac{N_{Buf3B} + N_{Cyclic-Rem}}{\phi_4^{min}} \\
=& \frac{24 + 2 + 88 + 1}{1 \times 12 \times 1 \times (1/2)} + \frac{48 + 1 + 1 + 1}{1 \times 12} + \\
& \frac{64 + 353 - 64 + 5 + 1}{18} + \frac{80 + 1}{20} \\
=& 19.16 + 4.25 + 19.94 + 4.05 = \\
=& 47.41 \mu s
\end{aligned} \tag{5.12}
$$

Mode (Mbits/s)	Latency (μs)	
	Transmission	Reception
6	33.02	47.41
9	32.69	42.35
12	32.43	39.73
18	32.26	37.20
24	32.15	35.91
36	32.06	34.65
48	32.01	34.00
54	31.99	33.79

Table 5.6: Numerical latency values.

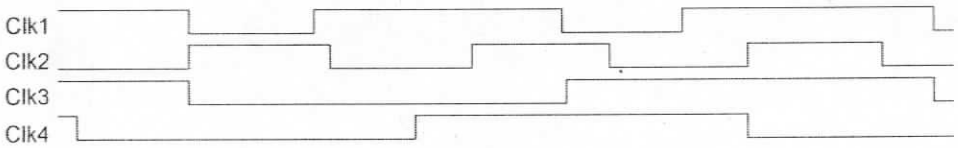

Figure 5.1: The waveform of clock signals in mode 36-Mbps corresponding to four pipeline stages.

5.3 Latency error

There is an error involved in Equation 5.5 and 5.10 due to the use of multiple clocks. The frequency of these clocks are not necessarily multiples of each other in different operating modes. therefore the rising edge and falling edge of these clocks are not aligned. For example, Figure 5.1 shows the waveform of CLK1, CLK2, CLK3, and CLK4 that correspond to Stage 1, 2, 3, and 4 of the pipeline respectively. The frequency of these clocks are depicted in Table 4.2. When a task is finished in one stage at the rising edge of the clock, there is a short gap till the rising edge of the other clock in the next stage. These gaps are not considered in Equation 5.5 and 5.10 and they create a small error between the result of these equations and simulation results. In Chapter 6 we show that the amount of this error in our design is less than 1.5% for all operating modes during transmission and reception.

Chapter 6

Implementation Results

To demonstrate specific power-performance figures, we have implemented our architecture on a Virtex-II Pro XC2VP50 FPGA manufactured by Xilinx [45]. In this chapter, first we report post place-and-route area, latency, and power consumption of this prototype. Then, we compare the potential power savings in our implementation against others.

6.1 Prototype area

The prototype occupies 9,740 slices, 28 embedded 18-Kbit RAM blocks, 132 embedded 18-bit multipliers, and 423 I/O Buffers. From the 28 embedded 18-Kbit RAM blocks, 21 of them are used for the IFFT/FFT block, 2 for each of the Buffers 1, 2 and 3, and 1 for the Preamble block. From the 132 embedded multipliers, 96 of them are used for the IFFT/FFT block, 24 for the Equalizer, and 12 for the Demodulator block. Figure 6.1 shows a high level floorplan of the prototype on Virtex-II Pro XC2VP50 FPGA. The approximate location of the hardware blocks that occupy large area of the FPGA is shown in Figure 6.1[1]. There is also two PowerPC (PPC)

[1]The boundaries shown in Figure 6.1 are approximate, the colors should be noticed to distinguish precise locations. The blocks that occupy a small area on FPGA are not specified by boundaries (only by color) to make the figure more readable. Scrambler/Descrambler block is the only block that is inside the IFFT/FFT boundary with the same color. The area of this block is negligible compared to the IFFT/ FFT area.

Mode (Mbits/s)	Latency (μs)	
	Transmission	Reception
6	33.46	47.40
9	33.18	42.31
12	32.71	39.67
18	32.56	37.13
24	32.36	35.88
36	32.26	34.62
48	32.21	33.98
54	32.20	33.77

Table 6.1: Estimated latency of processing a frame with only one OFDM symbol.

cores on the FPGA that can be utilized to implement the MAC layer.

6.2 Prototype latency

Table 6.1 shows the latency of our prototype during transmission and reception, when a PLCP frame with only one OFDM symbol is being processed. Figure 6.2 demonstrates these latency values. This figure shows that the transmitter has the advantage of almost constant latency. The latency of the receiver decreases as data rate goes up. The reason is the CLK1 of Stage 1 which is faster for higher data rates and therefore lowers the latency for those rates.

As mentioned in Chapter 5, there is a slight mismatch between the simulation results of Table 6.1 and what we were expecting from the latency equations 5.5 and 5.10. This latency mismatch is due to the the use of multiple clocks that might not be always aligned. Table 6.2 shows the amount of error between simulation results of Table 6.1 and numerical values of latency equations shown in Table 5.6. The error is less than 0.5 μs during transmission and reception, which is less than 1.5% in the worst case. Figure 6.3 demonstrates the latency mismatch of the transmitter and the receiver.

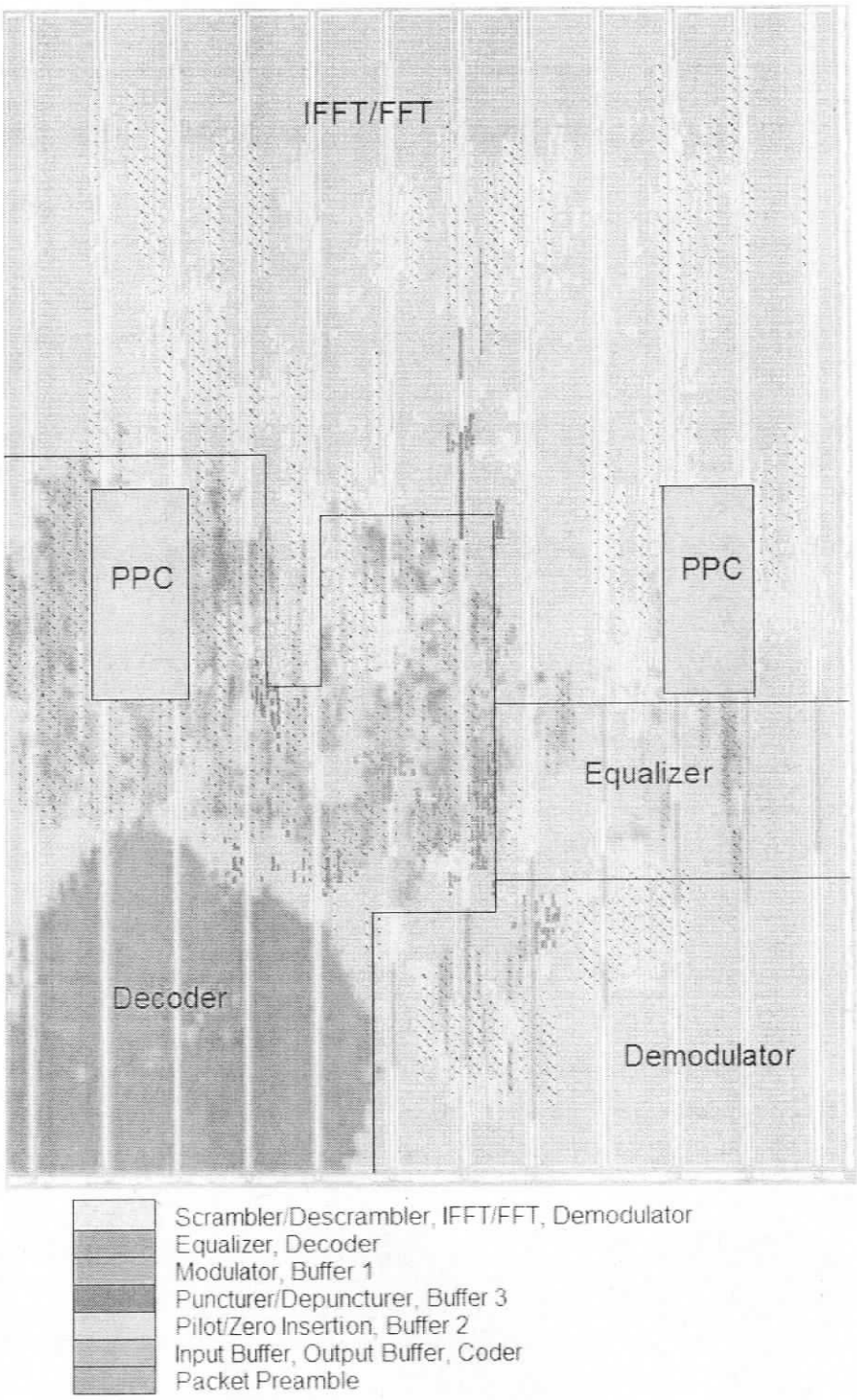

Figure 6.1: The floor plan of the prototype on Virtex-II Pro XC2VP50 FPGA.

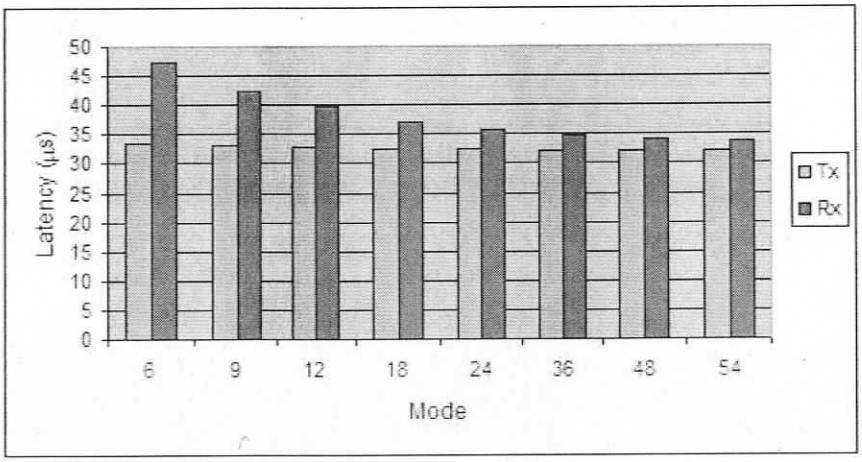

Figure 6.2: The latency of the transmitter (Tx) and the receiver (Rx).

Mode (Mbits/s)	Latency Mismatch (μs)		Latency Mismatch %	
	Transmission	Reception	Transmission	Reception
6	0.44	0.01	1.31%	0.02%
9	0.49	0.04	1.47%	0.09%
12	0.28	0.06	0.87%	0.15%
18	0.30	0.07	0.92%	0.18%
24	0.22	0.03	0.67%	0.08%
36	0.20	0.03	0.61%	0.08%
48	0.20	0.02	0.62%	0.05%
54	0.21	0.02	0.65%	0.05%

Table 6.2: Mismatch between simulation results and numerical values of latency equations 5.5 and 5.10.

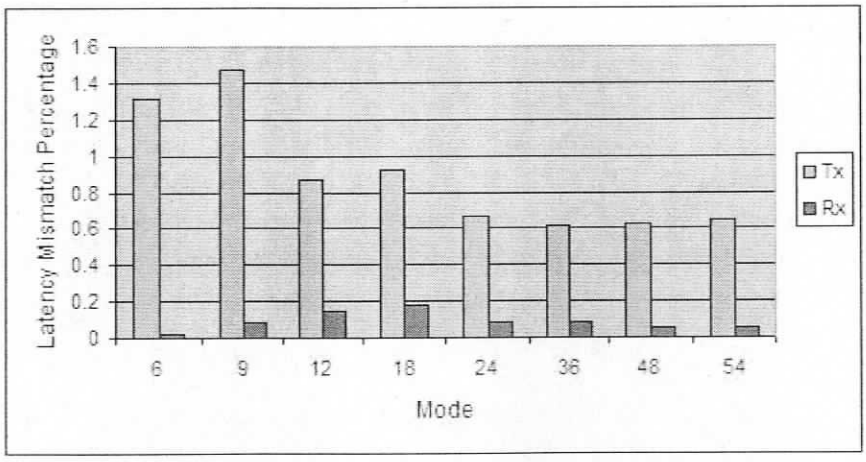

Figure 6.3: The latency mismatch of the transmitter (Tx) and the receiver (Rx).

6.3 Prototype power

The dynamic power consumption of our prototype is estimated in each mode of operation using different input data sequences. Tables 6.3, 6.4, 6.5 respectively show the power numbers when transmitting a data sequence of toggling bits (101010...10), a data sequence of all one bits (111...1), and a data sequence of all zero bits (000...0). The worst case power number in each mode[2] is considered as the power consumption of that operating mode. The power consumption of the transmitter and receiver is shown in Table 6.6 and plotted in Figure 6.4. Figure 6.4 shows that the power consumption during transmission and reception are almost similar which can be misleading, as the receiver does more work compared to the transmitter. The factor that should be considered to compare the work load of the receiver and transmitter is the energy consumption, not power consumption, which depends on the latency as well. The energy consumption of processing one OFDM symbol, which is the power consumption multiplied by the processing time of one OFDM symbol, is always higher in the receiver compared to the transmitter. Table 6.7 shows the energy consumption of the transmitter and receiver per OFDM symbol. These energy values are obtained by multiplying the power values of Table 6.6 by the corresponding latency values of Table 6.1. The energy consumption of the transmitter and receiver are demonstrated in Figure 6.5. This figure shows that the energy consumption of the transmitter is increasing as data rate increases. The reason is the increasing value of transmitter power consumption which is multiplied by an almost constant value of transmitter latency. The energy consumption of the receiver, however, decreases first and then increases as data rate increases. Note that the latency of the receiver is decreasing as the data rate increases (Figure 6.2), while the power consumption is increasing (Figure 6.4). The energy consumption of the receiver decreases first because the latency decreases faster than the power increases. After 18 Mbps, the situation is

[2]The worst case power consumption in each mode is highlighted in tables by the bold font.

Mode (Mbits/s)	Power (mW)	
	Transmission	Reception
6	**85**	**86**
9	**89**	**89**
12	93	**93**
18	115	**118**
24	**133**	**133**
36	**145**	**144**
48	163	**165**
54	**167**	**168**

Table 6.3: Estimated dynamic power consumption for the toggling bit sequence.

Mode (Mbits/s)	Power (mW)	
	Transmission	Reception
6	84	83
9	88	87
12	98	92
18	**120**	113
24	132	129
36	144	141
48	**164**	162
54	**167**	164

Table 6.4: Estimated dynamic power consumption for the all-ones sequence.

reversed: The power increases faster than the latency decreases. Hence, the energy consumption increases at higher data rates. Figure 6.5 also shows that the energy consumption of the receiver is always higher than the transmitter, and their difference becomes smaller at higher data rates.

6.4 Potential power saving

The four pipeline stages in our FPGA implementation allow for maximum clock frequencies that are shown in Table 6.8. These clock frequencies are greater than the minimum clock frequencies specified in Table 4.2 that were required to achieve the target data rates. The maximum-to-minimum frequency ratio can be used as a scaling factor for reducing the supply voltage, under the assumption that the relationship

Mode (Mbits/s)	Power (mW)	
	Transmission	Reception
6	84	85
9	88	86
12	**99**	91
18	118	114
24	**133**	132
36	**145**	**144**
48	161	162
54	164	165

Table 6.5: Estimated dynamic power consumption for the all-zeros sequence.

Mode (Mbits/s)	Power (mW)	
	Transmission	Reception
6	85	86
9	89	89
12	99	93
18	120	118
24	133	133
36	145	144
48	164	165
54	167	168

Table 6.6: Estimated dynamic power consumption (worst case power).

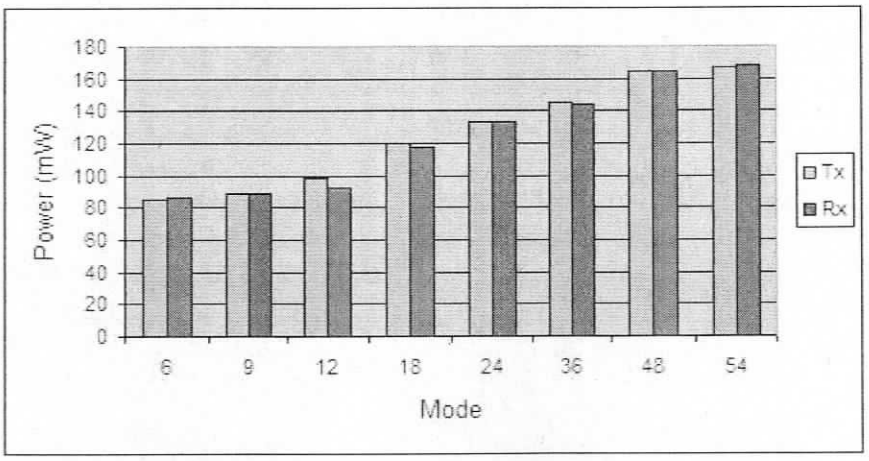

Figure 6.4: The power consumption of the transmitter (Tx) and the receiver (Rx).

Mode (Mbits/s)	Energy (μJ)	
	Transmission	Reception
6	2.844	4.076
9	2.953	3.765
12	3.042	3.689
18	3.907	4.381
24	4.303	4.772
36	4.677	4.985
48	5.282	5.606
54	5.377	5.673

Table 6.7: Estimated energy consumption of the FPGA prototype.

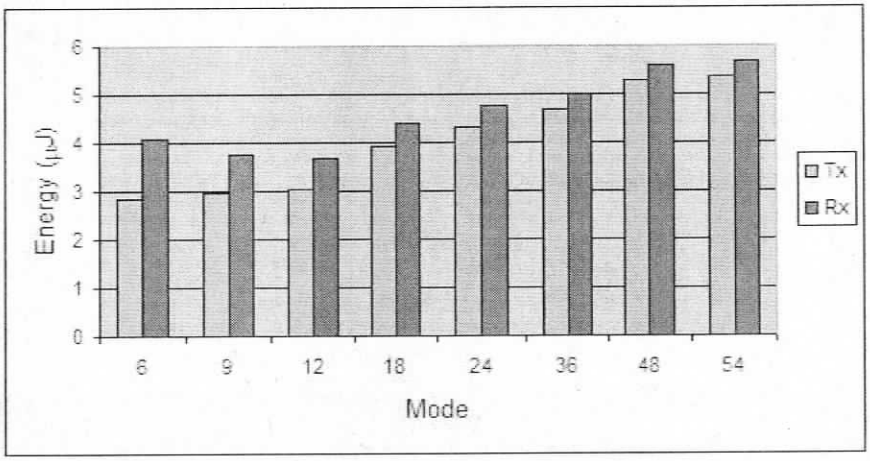

Figure 6.5: The energy consumption of the transmitter (Tx) and the receiver (Rx).

Clock Frequency	Maximum Value (MHz) Allowed by FPGA Prototype
ϕ_1^{max}	71.78
ϕ_2^{max}	85.32
ϕ_3^{max}	44.68
ϕ_4^{max}	210.97

Table 6.8: Maximum clock frequencies allowed by the FPGA prototype.

Mode	Scaling Factor S	Power Savings
6	2.48	83.7%
9	2.48	83.7%
12	2.48	83.7%
18	2.48	83.7%
24	1.78	68.4%
36	1.78	68.4%
48	1.18	28.2%
54	1.18	28.2%

Table 6.9: Voltage scaling factor and potential power savings.

between the voltage and the corresponding clock frequency is approximately linear [44]. It should be noted that we can not scale down the frequency in our design since we are already using the minimum clock frequencies that are required to support the 802.11a standard data rates. For a given pipeline stage i clocked at ϕ_i^{min}, the ratio $\phi_i^{max}/\phi_i^{min}$ can be interpreted as the voltage scaling factor S_i (if feasible). Reducing the voltage by the factor of S_i reduces the dynamic power consumption by the factor of S_i^2 [44]. If we were to use the same voltage for all four stages, our scaling factor would be $S = min\{S1, S2, S3, S4\}$, leading to the estimated power savings of $(1 - 1/S^2) \times 100\%$. Table 6.9 shows the scaling factors for different operating modes and the corresponding potential power savings of at least 28.2 %. Note that factors 2.48 and 1.78 in Table 6.9 may be infeasible due to the physical limits of voltage scaling.

6.5 Design comparison

Our design implements the PLCP sublayer of the IEEE 802.11a which is only a part of the complete 802.11a PHY layer. As a result, comparison with other 802.11a PHY layers is possible only if they provide their implementation results of the PLCP sublayer and PMD sublayer individually. Unfortunately commercial products do not provide enough detail about their internal components which makes the comparison with these products impossible. We compare our architecture only with the academic implementations that can be categorized to ASIC, FPGA-based, and software-based implementations. We consider different factors in these comparisons. The power consumption of the transceiver is one factor since low power dissipation is of paramount importance in mobile transceivers. The clock frequency of the system is also important as lower clock frequencies yield a bigger slack for dynamic voltage scaling, resulting in lower power consumption. The reconfiguration overhead is another factor to be considered, since lowering this overhead at the PHY layer lets the MAC layer to select proper data rate according to the channel condition more frequently.

6.5.1 Comparison with ASIC implementations

The dynamic power consumption of our prototype is the worst in the 54 Mbps mode: 335 mW after adding the transmitter power and the receiver power, or 241 mW after 28.2% saving. This is better than 351 mW power consumption reported in [31] which is an ASIC implementation of the 802.11a baseband transmitter and receiver excluding the synchronizer (consuming 25 mW), channel estimator (consuming 15 mW), and BIST circuitry (consuming 2 mW). We have excluded the power consumption of these blocks to have a fair comparison with our design which does not implement them. Although the ASIC design of [31] implements the same functions as our prototype, the comparison between them is still difficult since [31] employs 0.25 μm process

technology while our FPGA uses 0.13 μm technology. If that ASIC design was using 0.13 μm technology, it would consume less power. On the other hand, if our FPGA prototype was implemented on ASIC, it would consume less power as well.

The other ASIC implementation of 802.11a transceiver is reported in [29]. This design employs two chips: one front-end analog chip and one baseband processing chip that implements both 802.11a PHY layer and 802.11 MAC layer. The analog front end dissipates 790 mW of power during transmission and 430 mW during reception. The baseband transceiver, implemented in 0.25 μm technology, consumes 326 mW during active transmit and 452 mW during active receive. If we exclude the 24 mW power consumption of the MAC hardware, we obtain a power consumption of 302 mW for the 802.11a transmitter and 428 mW for the 802.11a receiver. The 302 mW power consumption of the baseband transmitter is already higher than our 167 mW power report for transmitter before dynamic voltage scaling. The receiver power consumption includes the power dissipation of the synchronizer and channel estimator blocks as well. From the detailed power report in [31], we can conclude that synchronizer and channel estimator dissipate a low portion of the receiver power, about 12.4%. By subtracting 12.4% of the 428 mW, we can estimate a power consumption of 375 mW for the receiver excluding the synchronizer and channel estimator blocks. This power number is again higher than our receiver power consumption of 168 mW before dynamic voltage scaling. The above discussion is still not a fair comparison because of difference between implementation platforms (ASIC vs. FPGA) and fabrication technologies (0.25 μm vs. 0.13 μm).

One interesting point that is noted from the power reports in [29] is the comparison between the power consumption of digital baseband transceiver versus the power consumption of analog front-end. In the transmitter chain, digital baseband transceiver consumes 27.7% of the system power versus 72.3 % of the analog front-end. This portion is 49.9% versus 51.1% in the receiver chain. These numbers show that the

digital baseband transceiver dissipates a big portion of the overall system power and it worth while being optimized for low power. The portion of digital baseband power consumption will be even more noticeable if analog front-end is power-optimized too. Reference [32] introduces such a low-power front-end. The transceiver front-end in this implementation uses 0.18 μm CMOS technology, and it consumes 207 mW in the receive mode and 247 mW in the transmit mode, while transmitting at full output power. The transmit mode power consumption is a strong function of the output power, reducing to 135 mW at the lowest output power level.

6.5.2 Comparison with FPGA-based implementations

FPGA-based implementations of 802.11a transceiver offer a performance close to ASIC designs and provide flexibility close to software implementations. Partial programmability offered in modern FPGAs eliminates the need for reprogramming the entire device to change the operation mode. Reference [24] implements 802.11a baseband system on Virtex II FPGA, using SelectMAP interface [26] that provides the means to reconfigure individual columns of FPGA. Reconfiguring this system to work with a different data rate takes 8.5 μs using a 32-bit PCI bus working at 33 MHz clock frequency. This reconfiguration overhead becomes a system bottleneck if data rate needs to be frequently adjusted by the MAC layer. In our implementation, system reconfiguration only needs the adjustment of the frequency dividers and changing the value of the mode signal.

The other FPGA implementation reported in [25] employs the Xilinx System Generator^{TM} for DSP [27] to implement some of the hardware blocks of OFDM transceiver. The System Generator tool facilitates rapid design, but may not result in an optimized hardware solution. Using Xilinx Virtex-II Pro FPGA for the implementation in [25], the long preamble correlator consumes 1100 slices and two embedded multipliers, and the channel estimator and frequency domain equalizer to-

gether occupy 776 logic slices, 2 block memories and 10 embedded multipliers. These number of slices are 18.76% of the number of slices that are used for our design, i.e., adding the channel estimation block, phase compensation block and the synchronizer block to our architecture increases the area of the FPGA prototype about 20%. The clock frequency employed for the hardware blocks of [25] is 100 MHz that is higher than our worst case clock frequency (72 MHz in the fastest mode) that will be only applied to the second stage of the pipeline. Reference [25] does not report latency or power numbers.

6.5.3 Comparison with software-based implementations

Software implementations of 802.11a introduce the most flexibility compared to FPGA and ASIC deigns, but generally provide the lowest performance. Reference [16] employs a programmable and reconfigurable Asynchronous Array of simple Processors (AsAP) and it achieves 54 Mbits/s data rate using a clock frequency of 1.0 GHz. Reference [17] can operate at the maximum rate of 136 Mbits/s using the same 1.0 GHz clock frequency by employing TI TMS320C64x DSP. Both of these implementations need a high clock frequency to support the highest data rate of 802.11a, while in our design a clock frequency of 54 Mbits/s is enough to support 54 Mbits/s data rate.

Another software implementation of the 802.11a transceiver, reported in [21], employs a TMS320C6201 DSP connected to a PC through a PCI bus. This system achieves a data rate of 24 Mbits/s during burst transmission. To provide 100% throughput using this DSP, it was estimated that 1626 MIPS would be required which creates the need for some form of hardware acceleration to support the DSP.

Reference [18] reports an implementation of 802.11a baseband transceiver on a domain specific programmable parallel DSP architecture called MaRS (Macro-pipelined Reconfigurable System). This system improves the performance of the 802.11a transceiver by using parallel processing elements to implement FFT and

Viterbi decoder blocks. The parallel architecture of FFT in this system needs 102 clock cycles to perform 64-point FFT operation which puts a lower bound of 31 MHz on the clock frequency of the FFT. Our pipelined FFT architecture takes 72 clock cycles to perform 64-point FFT operation which calls for a clock frequency of 18 MHz for the FFT block. This architecture requires 6 clock cycles to process 1 bit during Viterbi decoding, while our architecture needs only 1 clock cycle to process 1 bit during decoding.

Reference [19] has fabricated a 32-bit RISC Processor in 0.18 μm technology to specifically perform 802.11a MAC and PHY layer operations. This reference is the most optimized software-based implementation compared to all the above implementations. This design includes 4-Mb SRAM, PLL, A/D and D/A converters, and 1.3-million-gate random logics in addition to the RISC processor. It employs three clock frequencies: 20, 40, and 80 MHz and consumes 958 mW in total from 1.5-V, 2.5-V, or 3.3-V supplies. Unfortunately there is lack of details in this paper about individual blocks of transceiver which makes it difficult to compare it with our design.

Chapter 7

Future Work and Extensions

The proposed architecture in this work presents a short latency and low-power consumption for digital baseband processing of the IEEE 802.11a standard. It can also result in a short overhead to reconfigure the system to work with different data rates. This architecture is implemented on FPGA for test of functionality. This work can be extended in several ways by, for example, (1) Adding the synchronization, channel estimation, and equalization circuitry and implementing this hardware on ASIC to further reduce the energy consumption, (2) using adaptive bit-precision for hardware components that can work with lower bit-precision in robust operation modes, (3)implementing an efficient MAC layer which takes advantage of the architectural features of this PHY layer, and (3) supporting other PHY layer standards in this architecture as well. These extensions are discussed in this chapter.

7.1 ASIC implementation

The proposed architecture in this work is fully customized which makes it easily transferable to an ASIC platform. The ASIC implementation can include the Channel Estimation, Phase Compensation and Synchronization blocks that were not included in our design. The ASIC implementation also facilitates voltage scaling which leads to lower energy consumption.

7.2 Adaptive bit-precision

In our work, we considered a fixed bit-precision for all the hardware blocks in the PHY layer. However, the bit-precision of different hardware blocks such as the Modulator and Demodulator in Stage 2, all the hardware blocks in Stage 3 and Stage 4, and also the A/D and D/A converters in the PMD sublayer can be made adjustable. In robust operation modes such as the ones that work with BPSK modulation, reducing the bit-precision of these hardware blocks results in further energy reduction. The clock gating technique can be used to design adaptive bit-precision hardware.

7.3 Efficient mode adaptation

The MAC layer decides about the data rate of the PHY mode according to different factors such as the channel condition, application constraints, energy budget, etc. In [50], for example, mode adaptation is based on a look-up table that is indexed by the system status. The system status is characterized by data payload length, the receiver-side SNR value to quantify the wireless channel condition, and frame retransmission count. This look-up table defines the best PHY mode in terms of maximizing the expected effective goodput under the corresponding system status. This table is calculated offline and is used at runtime to determine the best PHY mode for the next frame transmission attempt.

Other wireless systems might define different system status that MAC layer considers to adjust the PHY mode. The faster the MAC layer adopts the transmission mode of the next frame with the system status, the less transmission or retransmissions would be required, resulting in less transmission time and energy consumption. One factor that defines how fast MAC layer adapts the transmission mode is the overhead of system reconfiguration during mode adjustment. Sometimes, the delay or energy overhead of reconfiguration is more than the time or energy that is saved

by mode adjustment. Our proposed architecture introduces a low reconfiguration overhead, i.e., efficient MAC algorithms can be devised to take advantage of the possibility for frame-by-frame mode adaptation.

7.4 Scheduling policy

When different multimedia streams such as voice and video are sent through the same transmitter, a scheduling policy is required to regulate the traffic. Most of the scheduling algorithms in the literature usually do not consider the energy and time spent for the reconfiguration of the physical layer. They usually assume frame-by-frame reconfiguration [51], which can be acceptable only when the frames are very large, so that the configuration time is negligible compared to the frame transmission time. However, practical implementations that use serial streams of control bits to change the configuration of the hardware, may have significant time and energy penalties during reconfiguration. These reconfiguration penalties may not be ignored easily.

One of the extensions can target the scheduler of the MAC layer that decides on the size of the data units that are sent from each stream, with the goal of decreasing the number of reconfiguration events without violating the required QoS for each stream.

7.5 Supporting other WLAN standards

As it was explained in Chapter 4.4, IEEE 802.11g and HIPERLAN/2 have a similar basis as IEEE 802.11a standard. One future work is to make our design compatible with these two standards as well, i.e., adjusting the hardware blocks of the proposed architecture to support the data rate and functionalities of these two standards.

The new upcoming WLAN standard, IEEE 802.11n, also has an OFDM basis but

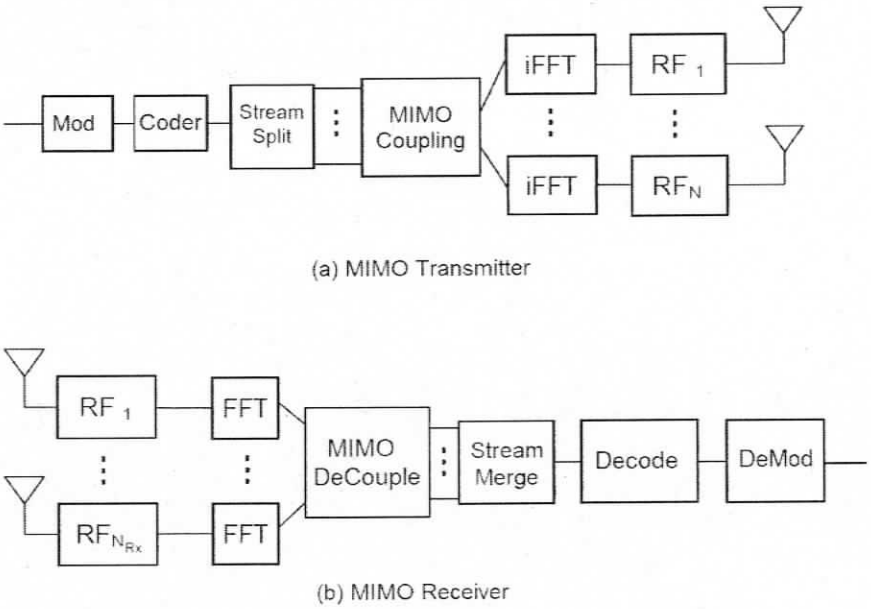

Figure 7.1: Block diagram of a MIMO (a) transmitter. (b) receiver [11].

for multiple antennas [10]. Figure 7.1 shows a simple block diagram of the transmitter and receiver of such Multiple Input Multiple Output (MIMO) systems [11]. For these systems, the energy consumption is a critical issue since the front-end circuit of every antenna consumes a considerable amount of energy. A similar multi-clock pipeline architecture proposed can be employed to implement the 802.11n standard. This architecture can reduce the energy consumption by applying low clock frequencies to different pipeline stages and also by providing low reconfiguration overhead which lets the MAC layer make better decisions about selecting the PHY mode and selecting the number of antennas.

Chapter 8

Conclusion

We proposed a 4-stage 4-clock structure for the IEEE 802.11a baseband transceiver. This architecture supports eight different data rates specified by the standard. Reconfiguring this system to work with different data rates is fast as it requires the adjustment of two clock frequencies. This architecture reduces the energy consumption by: (1) using pipeline structure that increases the throughput and allows for lower clock frequencies which provide the opportunity for dynamic voltage scaling and consequently energy saving. (2) using multiple clocks that reduce the handshaking and buffering inside pipeline stages, and (3) reducing the reconfiguration energy overhead which allows for frame-by-frame reconfiguration. The implementation of this architecture on Xilinx Virtex-II Pro FPGA occupies about 10.000 slices and consumes 335 mW for transmission and reception together. Low clock frequencies applied in this design provide a delay slack for dynamic voltage scaling and therefore power saving of up to 28.2%. This work can be extended in future to include all the hardware components of the PHY layer: the channel estimation and phase compensation circuitry added to the PLCP sublayer, and the transceiver front end, which includes the RF/IF circuitry and the synchronization block, attached to the PLCP sublayer. The future work can also focus on the optimal MAC layer algorithms that decide about the operating mode and frame length based on the energy and perfor-

mance constraints. The MAC layer can be implemented using two PowerPC cores on the Xilinx Virtex-II Pro FPGA.

Bibliography

[1] H. Yang, "A road to future broadband wireless access: MIMO-OFDM-Based air interface," IEEE Communications Magazine, vol. 43, pp. 53-60, Jan. 2005.

[2] Broadcom, "802.11n: Next-generation wireless LAN technology," White paper, Apr. 2006.

[3] IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks - specific requirement. Part 11: wireless LAN medium access control (MAC) and Physical layer (PHY) specifications. Amendment 2: higher-speed physical lyaer (PHY) extension in the 2.4 GHz band. 1999.

[4] ETSI DTS/BRAN 030003-1. Broadband radio access networks (BRAN). HIPERLAN type 2 functional specifications, Part 1 Physical (PHY) layer, Jun. 1999.

[5] Information technology- telecommunications and information exchange between systems- local and metropolitan area networks- specific requirements part 11: wireless lan medium access control (MAC) and physical layer (PHY) specifications amendment 1: high-speed physical layer in the 5 GHz band. 1999.

[6] W. Eberle, V. Derudder, L. Van Der Perre, G. Vanwijnsberghe, M. Vergara, L. Deneire, B. Gyselinckx, M. Engels, I. Bolsens, H. De Man, "A digital 72 Mbps

64-QAM OFDM transceiver for 5GHz wireless LAN in 0.18 μm CMOS", IEEE Solid-State Circuits Conference, Digest of Technical Papers, pp. 336-337, Feb. 2001.

[7] P. Ryan, T. Arivoli, L. De Souza, G. Foyster, R. Keaney, T. McDermott, A. Moini, S. Al-Sarawi, L. Parker, G. Smith, N. Weste, G. Zyner, "A single chip PHY COFDM modem for IEEE 802.11a with integrated ADCs and DACs", IEEE Solid-State Circuits Conference, Digest of Technical Papers, pp. 338-339, Feb. 2001.

[8] IEEE standard for information technology- telecommunications and information exchange between systems- local and metropolitan area networks- specific requirements Part II: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, 2003.

[9] A. Nogee, *WLAN chipset market - The incredible journey is just beginning*, In-Stat Publications, Mar. 2002.

[10] J. Lorincz, D. Begusic, "Physical layer analysis of emerging IEEE 802.11n WLAN standard," IEEE Conference on Advanced Communication Technology, vol. 1, pp. 189-194, Feb. 2006.

[11] J. M. Gilbert, C. Won-Joon, S. Qinfang, "MIMO technology for advanced wireless local area networks," ACM Proceedings on Design Automation, pp. 413-415, 2005.

[12] *http://en.wikipedia.org/wiki/IEEE_*802.11

[13] R. Jejurikar, R. Gupta, "System design methodologies: Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems," International Symposium on Low Power Electronics and Design, pp. 78-81, 2004.

[14] Y. Kim, H. Jung, H. H. Lee, K. R. Cho, "MAC implementation for IEEE 802.11 wireless LAN," IEEE International Conference on ATM (ICATM 2001) and High Speed Intelligent Internet, pp. 191-195, Apr. 2001.

[15] Breezecom Wireless Communications, "A technical tutorial on the IEEE 802.11 standard," Jul. 1996.

[16] M. J. Meeuwsen, O. Sattari, B. M. Baas, "A full-rate software implementation of an IEEE 802.11a compliant digital baseband transmitter," IEEE Workshop on Signal Processing Systems, pp. 124-129, 2004.

[17] Y. Tang, L. Qian, Y. Wang, "Optimized software implementation of full-rate IEEE 802.11a compliant digital baseband transmitter on digital signal processor," IEEE Global Telecommunications Conference, pp. 2194-2198, 2005.

[18] A. Kamalizad, N. Tabrizi, N. Bagherzadeh, A. Hatanaka, "A programmable DSP architecture for wireless communication systems," IEEE International Conference on Application-Specific Systems, Architecture Processors, pp. 231-238, Jul. 2005.

[19] T. Fujisawa, J. Hasegawa, K. Tsuchie, T. Shiozawa, T. Fujita, T. Saito, Y. Unekawa, "A single-chip 802.11a MAC/PHY with a 32-b RISC processor," IEEE Journal of Solid-State Circuits, vol. 38, pp. 2001-2009, Nov. 2003.

[20] T. Shono, Y. Shirato, H. Shiba, K. Uehara, K. Araki, M. Umehira, "IEEE 802.11 wireless LAN implemented on software defined radio with hybrid programmable architecture," IEEE Transactions on Wireless Communications, vol. 4, pp. 2299-2308, Sep. 2005.

[21] M. F. Tariq, Y. Baltaci, T. Horseman, M. Butler, A. Nix, "Development of an OFDM based high speed wireless LAN platform using the TI C6x DSP," IEEE International Conference on Communications, vol. 1, pp. 522-526, 2002.

[22] J. Craninckx, S. Donnay, "4G terminals: how are we going to design them?," ACM Design Automation Conference, pp. 79-84, Jun. 2003.

[23] SDR Forum, http://www.sdrforum.org

[24] P. Coulton and D. Carline, "An SDR inspired design for the FPGA implementation of 802.11a baseband system". IEEE International Symposium on Consumer Electronics, pp. 470-475, Sep. 2004.

[25] C. Dick, F. Harris, "FPGA implementation of an OFDM PHY," IEEE Asilomar Conference on Signals, Systems and Computers, vol. 1, pp. 905-909, Nov. 2003.

[26] Xilinx Inc., Using a microprocessor to configure Xilinx FPGAs via Slave Serial or SelectMAP Mode, v 1.4, 2002.

[27] Xilinx Inc., "System Generator for DSP," Reference Guide, Apr. 2008.

[28] M. Krstic, K. Maharatna, A. Troya, E. Grass, U. Jagdhold, "Implementation of an IEEE 802.11a compliant low-power baseband processor," IEEE International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Service, vol. 1, pp. 97-100, Oct. 2003.

[29] T. H. Meng, "Design and implementation of an all-CMOS 802.11a wireless LAN chipset," IEEE Communications Magazine, vol. 41, pp. 160-168, Aug. 2003.

[30] H. Zou, B. Daneshrad, "VLSI implementation for a low power mobile OFDM receiver ASIC," IEEE Wireless Communications and Networking Conference, vol. 4, pp. 2120-2124, Mar. 2004.

[31] A. Troya, K. Maharatna, M. Krstic, E. Grass, U. Jagdhold, R. Kraemer, "Low-power VLSI implementation of the inner receiver for OFDM-based WLAN systems," IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, vol. 55, pp. 672-686, Mar. 2008.

[32] R. Ahola, "A single-chip CMOS transceiver for 802.11a/b/g wireless LANs", IEEE Journal of Solid-State Circuits, vol. 39, pp. 2250 - 2258, Dec. 2004.

[33] J. Heiskala, J. Terry, *OFDM Wireless LANs: A Theoretical and Practical Guide*, SAMS Publishing, 2001.

[34] W. Zhong, "Design and VLSI architecture of a channel equalizer based on adaptive modulation for IEEE 802.11a WLAN," IEEE Asia Pacific Conference on Circuits and Systems, pp. 1699-1702, 2006.

[35] M. Speth, S. A. Fechtel, G. Fock, H. Meyr, Optimum receiver design for wireless broad-band systems using OFDM-Part I, IEEE Transactions on Communications, vol.47, no.11, pp. 1668-1677, Nov.1999.

[36] J. J. van de Beek, O. Edfors, M. Sandell, S. K. Wilson, P. O. Borjesson, "On channel estimation in OFDM systems", IEEE 45th Vehicular Technology Conference, pp. 815-819, 1995.

[37] A. R. S. Bahai, B. R. Saltzberg, M. Ergen, *Multi-carrier digital communications, theory and applications of OFDM*, Springer Science + Business Media Inc., Second Edition, 2004.

[38] Y. H. Hu, "CORDIC-based VLSI architectures for digital signal processing," IEEE Signal Procersing Magazine, pp. 16-35, Jul. 1992.

[39] K. Akita, R. Sakata, K. Sato, "A phase compensation scheme using feedback control for IEEE 802.11a receiver," IEEE Vehicular Technology Conference, vol. 7, pp. 4789-4793, 2004.

[40] R. V. Nee, R. Prasad, *OFDM for Wireless Multimedia Networks*, Artech House, 2001.

[41] *http://en.wikipedia.org/wiki/Discrete_Fourier_transform*

[42] J. Cooley, J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," Math. Comput. 19, pp. 297301, 1965.

[43] Altera , "FFT MegaCore function", Errata Sheet, MegaCore Version 7.0, 2007.

[44] T. Burd and R. Brodersen, "Energy efficient microprocessor design", MA: Kluwer, 2002.

[45] Xilinx, "Virtex-II Pro Paltform FPGA Handbook", 2002.

[46] G. D. Forney,"The Viterbi algorithm", IEEE Proceedings, pp. 218278, 1973.

[47] G. Manfred, T. Hollstein, L. Soares, P. Zipf, T. Pionteck, M. Petrov, H. Zimmer, T. Murgan, "Reconfigurable platforms for ubiquitous computing," ACM Computing Frontiers, pp. 377-389, 2004.

[48] S. J. Lee, N. R. Shanbhag, A. C. Singer, "Area-efficient high-throughput MAP decoder architectures," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 13, pp. 921-933, Aug. 2005.

[49] F. Huang, "Evaluation of Soft Output Decoding for Turbo Codes," MSc. Thesis, Virginia Tech, 1997.

[50] D. Qiao, S. Choi, K. G. Shin, "Goodput analysis and link adaptation for IEEE 802.11a wireless LANs," IEEE Transactions on Mobile Computing, vol. 1, pp. 278 - 292, Oct-Dec 2002.

[51] V. Raghunathan, S. Ganeriwal, C. Schurgers, M. Srivastava, "E^2WFQ: An energy efficient fair scheduling policy for wireless systems ," International Symposium on Low Power Electronics and Design, pp. 30-35, 2002.

Appendix A

Tool Setting

We have used ISE Project Navigator (version 9.1i) that provides a complete set of tools for simulation, synthesis, placement and routing, and power analysis for our FPGA implementation.

The first task in creating a new ISE project is to specify the configuration properties as shown in Figure 8.1. Next, we have to include the Verilog description of our hardware components by importing the corresponding V-files. Each of the Verilog components is first tested individually to ensure that it is synthesizable and functions correctly. We also have to generate the memory blocks of our architecture, listed in Table 8.1, using the Xilinx Coregen library. All of these blocks use the "Enable Pin" and all their inputs are sensitive to the rising edge of the clock. Note that the initialization values for ROM memory blocks have to be loaded from the corresponding

Memory Name	Type	Size
Buffer 1	Dual port RAM	2x4x512
Buffer 2	Dual port RAM	64x320
Buffer 3	Dual port RAM	64x128
Preamble Buffer	Single port ROM	64x256
FFT-RAM1	Single port RAM	63x32
FFT-RAM2.3.4.5.6.7-0.7-1	Dual port RAM	64x64
FFT-W-Gen	Single port ROM	64x32

Table 8.1: The type and size of memory blocks.

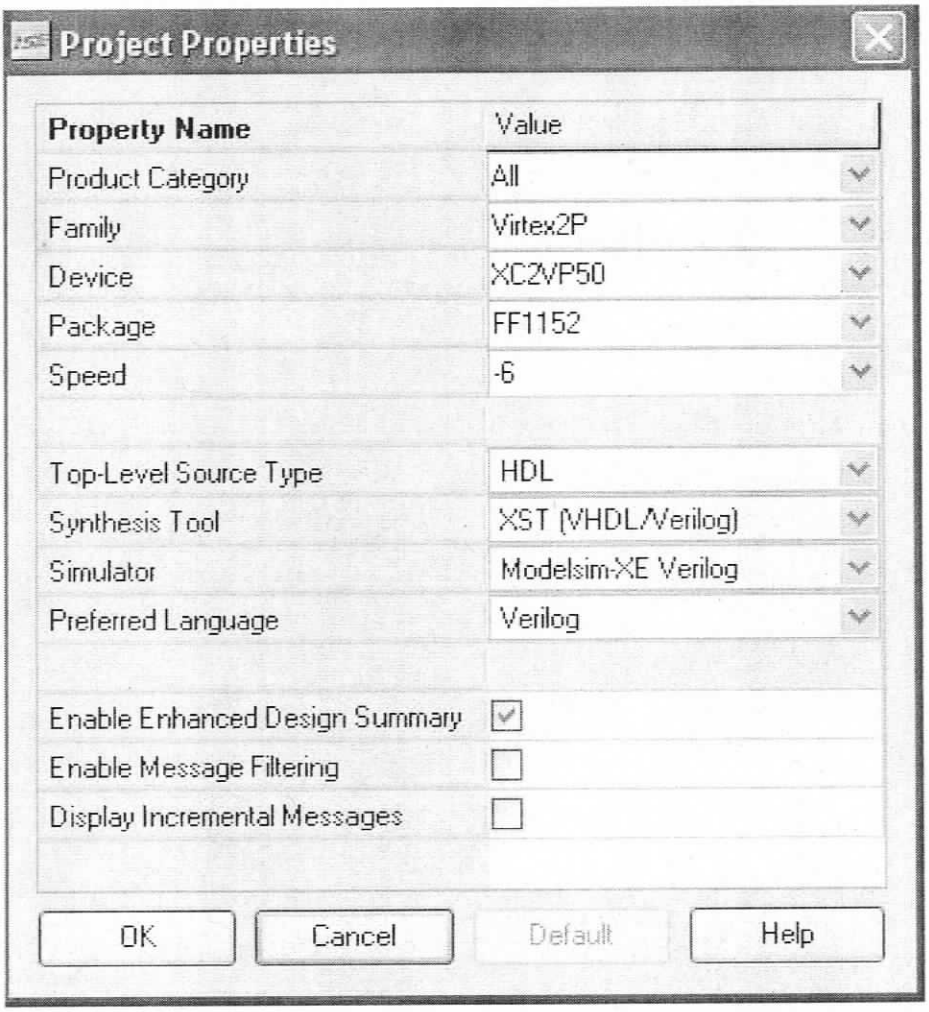

Figure 8.1: ISE Project setup.

COE-files.

After including all the hardware components in the project, we run a Verilog description of our testbench to simulate the behavioral model of our design using Modelsim XE III/Starter (version 6.2g). After verifying the behavioral model of the entire architecture, we synthesize it, setting "Speed" as the optimization goal. Then, we implement our design on the FPGA by executing the translation, mapping, and placement and routing steps.

Before placement and routing, we generate a constraint file (UCF-file) which defines minimum clock periods. The goal is to force Xilinx to place and route hardware of Stage 1 and Stage 2 of the pipeline with minimum critical path delay[1]. After placement and routing, we set the option of generating a VCD-file in the post-place-and-route simulation properties and run post-place-and-route simulation using Modelsim. We stop the simulation once the transmission task is complete and use the generated VCD-file to measure the amount of power consumption during transmission. XPower is the tool in ISE that measures the static and dynamic power consumption using the corresponding VCD-file. Next, we flush the VCD-file and resume the simulation to generate a new VCD file for the receiver, and then we measure the receiver power consumption using the corresponding VCD-file.

[1]The critical path delay of Stage 3 and Stage 4 are short enough that Xilinx have no problem to place and route them using the clock period of these stages.