

國立交通大學

電子工程學系 電子研究所碩士班

碩 士 論 文

IEEE 802.16e OFDM 上行及 OFDMA 下行同步技術
與數位訊號處理器實現之研究



Research in Synchronization Techniques and DSP
Implementation for IEEE 802.16e OFDM Uplink and
OFDMA Downlink

研 究 生: 紀國偉

指 導 教 授: 林大衛 博士

中 華 民 國 九 十 五 年 六 月

IEEE 802.16e OFDM 上行 OFDMA 下行同步技術與數位訊號處理器

實現之研究

**Research in Synchronization Techniques and DSP Implementation
for IEEE 802.16e OFDM Uplink and OFDMA Downlink**

研 究 生: 紀國偉

Student: Guo Wei Ji

指 導 教 授: 林大衛 博士

Advisor: Dr. David W. Lin

國 立 交 通 大 學

電子工程學系 電子研究所碩士班



Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master

in

Electronics Engineering

June 2006

Hsinchu, Taiwan, Republic of China

中 華 民 國 九 十 五 年 六 月

IEEE 802.16e OFDM 上行及 OFDMA 下行同步技術 與數位訊號處理器實現之研究

研究生：紀國偉

指導教授：林大衛 博士

國立交通大學

電子工程學系 電子研究所碩士班

The logo of National Central University (NCU) is a circular emblem with a gear-like border. Inside the circle, there is a stylized representation of a building or a bridge, and the year '1896' is inscribed at the bottom. The Chinese characters '摘要' (Abstract) are written in the center of the emblem.

摘要

本篇論文介紹 IEEE 802.16e 正交分頻多工(OFDM)和正交分頻多工存取(OFDMA)的同步。我們討論了他們同步的問題、演算法、以及實做方面的議題。

在正交分頻多工系統，我們首先設計一套同步系統用來解決符碼時間偏移和小數部分載波偏移的問題，並用浮點數運算來實做。符碼時間是利用固定不變的同步碼(preamble)來同步，而小數部分的載波偏移是計算靠循環字首(cyclic prefix)的相關性(correlation)來估計其值。我們同時在可加性白色高斯雜訊通道(AWGN)以及多路徑 Rayleigh 衰減通道下做模擬，模擬速度高達 60 km/h，並觀察其結果。

其次，我們把這些方法修改成定點運算的版本，並在數位訊號處理平台上，最佳化我們的程式的速度。雖然修改成定點運算會使效能衰減，但其接果依然可以接受。最後我們系統的每一塊功能區塊(function block)都能達到即時處理(real time)的要求。

在正交分頻多工存取系統中，除了符碼時間偏移和小數部分載波偏移之外，尚有整數部分載波偏移、取樣頻率偏移、和同步碼索引(preamble index)等需要同

步。因為使用者端事先無法知道確切的同步碼，所以我們把同步碼視為一般符碼，利用計算靠循環字首的相關性來估計時間偏移和小數載波偏移。我們同樣利用同步碼的特性來同步整數載波偏移和同步碼索引。小數載波偏移需要靠平均每個循環字首相關性的結果來得到較準的結果，而取樣頻率偏移可以和載波偏移一起同步，因為他們有一樣的錯誤比例。我們也提供比較準確的時間同步以改善同步碼索引同步的錯誤率。

就像正交分頻多工系統一樣，我們用浮點運算來實做這些方法，並同時在可加性白色高斯雜訊通道以及多路徑 Rayleigh 衰減通道下做模擬，不過這邊的模擬速度高達 300 km/h。



Research in Synchronization Techniques and DSP Implementation for IEEE 802.16e OFDM Uplink and OFDMA Downlink

Student: Guo Wei Ji

Advisor: Dr. David W. Lin

Department of Electronics Engineering

Institute of Electronics

National Chiao Tung University



Abstract

This thesis introduces the synchronization of IEEE 802.16e OFDM and OFDMA system. We discuss their synchronization problems, algorithms, and implementation issues.

In the OFDM system, we first design a synchronization system to overcome the problems of symbol timing offset and fractional carrier frequency offset (CFO), and implement them with floating-point. The symbol timing is synchronized by invariant preamble, and the fractional CFO is estimated by cyclic prefix (CP) correlation. We simulate our system in both AWGN and multipath Rayleigh fading channel, which the speed is as high as 60 km/h, and see the performance.

Next, we modified these methods into fixed-point version, and then optimize the speed of our programs on the digital signal processor (DSP) platform. Although the performance is degraded because of fixed-point modification, the results still can be accepted. Finally the function blocks of our system can all reach the requirement of real time.

In the OFDMA system, in addition to symbol timing offset and fractional CFO, it

still has integer CFO, sampling frequency offset (SFO), and preamble index need to be synchronized. Since the SS does not know the preamble in advance, we view preamble as a regular symbol and estimate the symbol timing and fractional CFO by CP correlation. We also use the feature of preamble to estimate the integer CFO and identify the preamble index. The fractional CFO needs be estimated by averaging every CP correlation result for a more accurate result, and the SFO can be synchronized with fractional CFO synchronization because they have the same error ratio. We also afford a fine timing estimation to improve the error rate of the preamble index identification.

Like in the OFDM system, we implement these methods in floating-point version, and simulate them in both AWGN and multipath Rayleigh fading channel, but the speed is as high as 300 km/h here.



誌謝

本篇論文方得以順利完成，首先想感謝林大衛老師。在兩年的研究所生涯裡，由於他的細心指導及在專業領域的博學精深，使得我在學習研究這條路上，一直都能順利地往前行。祝福老師在忙碌之餘，能保有健康的身體。

另外，感謝通訊電子與訊號處理實驗室所有的成員，包含各位師長、同學、學長姐與學弟妹們。感謝吳俊榮學長、洪崑健學長給予我在研究過程上的指導與建議，還有陳勇竹同學、王治傑同學、陳旻弘同學、黃育彰同學、劉建志同學、林鴻志等同學，因為能和你們共同討論及分享求學的經驗，使得實驗室一直是一個燈光美、氣氛佳的好地方。

最後，我要感謝我的家人和朋友們，感謝他們一直都在背後支持我，讓我能心無旁騖地完成學業。

在此，將此篇論文獻給所有給予我幫助的人。



紀國偉

民國九十五年六月 於新竹

Research in Synchronization Techniques and DSP Implementation for IEEE 802.16e OFDM Uplink and OFDMA Downlink

Prepared by Guo-Wei Ji

Directed by Prof. D. W. Lin

In Partial Fulfillment of the Requirements

for the Degree of
Master of Science

Department of Electronics Engineering

National Chiao Tung University

Hsinchu, Taiwan 300, R.O.C.

E-mail: lotus1982@gmail.com

June 20, 2006



Contents

1	Introduction	1
2	Overview of OFDM and OFDMA	4
2.1	OFDM	4
2.1.1	Introduction to OFDM	4
2.1.2	Mathematical Description of OFDM	6
2.2	OFDMA System	9
3	Overview of the IEEE 802.16e Standard	11
3.1	Introduction to IEEE 802.16 [11]	11
3.2	WirelessMAN-OFDM TDD Uplink [3]	12
3.2.1	OFDM Symbol Parameters	12
3.2.2	Point-to-Multipoint (PMP) Frame Structure	14
3.2.3	Modulation	16
3.2.4	Preamble Structure and Modulation	17
3.2.5	Frequency and Timing Requirements	18
3.3	WirelessMAN-OFDMA TDD Downlink [3]	19

3.3.1	OFDMA Basic Terms	19
3.3.2	OFDMA Symbol Parameters	21
3.3.3	Frame Structure	21
3.3.4	OFDMA Downlink Subcarrier Allocation	24
3.3.5	Modulation	31
3.3.6	Frequency and Timing Requirements	33
3.4	Transmit Spectral Mask	34
3.5	System Parameters	35
3.5.1	Uplink OFDM Transmission Parameters	35
3.5.2	Downlink OFDMA Transmission Parameters	36
3.6	Transmission Filters [7]	37
4	Introduction to the DSP Implementation Platform	39
4.1	The DSP Chip [16]	39
4.2	TI's Code Development Environment [17]	45
4.2.1	Code Development Flow [18]	46
4.2.2	Compiler Optimization Options [18]	48
4.2.3	Software Pipelining [19]	50
4.2.4	Intrinsics [18]	51
5	OFDM TDD Uplink Synchronization	52
5.1	OFDM Uplink Synchronization Problem and Techniques	52

5.1.1	Timing Offset and Fractional Carrier Frequency Offset	53
5.1.2	Integer Carrier Frequency Offset	55
5.1.3	Sampling Frequency Offset	55
5.2	Channel Model	57
5.2.1	Gaussian Noise	57
5.2.2	Slow Fading Channel	57
5.2.3	Fast Fading Channel	58
5.2.4	Power-Delay Profile Model	58
5.3	Floating-Point Simulation Results	59
5.3.1	Simulation Parameters and Environments	60
5.3.2	Symbol Timing Estimation	62
5.3.3	Carrier Frequency Synchronization	63
5.4	Fixed-Point Implementation	67
5.4.1	Modulation and Subcarrier Allocation	69
5.4.2	The IFFT and FFT	70
5.4.3	SRRC Filter with Oversampling and Downsampling	72
5.4.4	Synchronization	74
5.5	Fixed-Point Simulation Results	74
5.5.1	Symbol Timing Estimation	74
5.5.2	Carrier Frequency Synchronization	75
5.5.3	Bit Error Rate Performance	76

5.6	Program Optimization	79
5.6.1	The Modulation Function	80
5.6.2	The Pilot Generate Function	80
5.6.3	The Allocation Function	81
5.6.4	The IFFT, Add CP and Tx SRRC, Rx SRRC and downsample, FFT functions	82
5.6.5	The Preamble Synchronization Function	82
5.6.6	The CFO Synchronization Function	83
5.6.7	The Frequency Compensation Function	83
5.7	Profile of Optimized DSP Program	88
6	OFDMA TDD Downlink Synchronization	94
6.1	OFDMA Downlink Synchronization Problems and Techniques	94
6.1.1	Timing Offset and Fractional Carrier Frequency Offset	95
6.1.2	Integer Carrier Frequency Offset	96
6.1.3	Preamble Index Identification	98
6.1.4	Fine Symbol Timing Estimation	100
6.2	Floating-Point Simulation Results	101
6.2.1	Symbol Timing Estimation	101
6.2.2	Fractional CFO Estimation	102
6.2.3	Integer CFO Estimation	103
6.2.4	Preamble Index Identification	105

6.2.5	Fine Symbol Timing Estimation	106
6.2.6	Comparison of Preamble Indexes	107
7	Conclusion and Future Work	116
7.1	Conclusion	116
7.2	Future Work	117



List of Figures

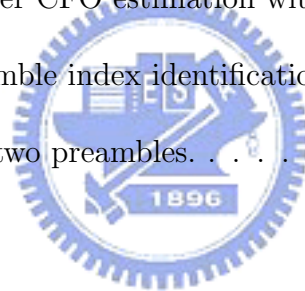
2.1	Bandwidth efficiency comparison of FDM and OFDM systems (from [5]). . .	5
2.2	The carrier has no crosstalk from other carriers at its center frequency (from [5]).	6
2.3	Three time domain waveforms of different carriers (from [5]).	7
2.4	Comparison of OFDM and OFDMA subcarriers allocation (from [12]). . . .	10
3.1	OFDM frame structure with TDD (from [2]).	15
3.2	PRBS generator for pilot modulation (from [3]).	17
3.3	Short preamble time domain structure (from [3]).	18
3.4	OFDMA frequency description (3-channel schematic example, from [2]). . . .	19
3.5	Example of the data region which defines the OFDMA allocation (from [2]).	20
3.6	Example of an OFDMA frame (with only mandatory zone) in TDD mode (from [3]).	22
3.7	FCH subchannel allocation for all 3 segments (from [3]).	24
3.8	Example of DL renumbering the allocated subchannels for segment 1 in PUSC (from [3]).	25
3.9	Cluster structure (from [3]).	28

3.10	Transmit spectral mask (from [2]).	34
3.11	Transmitter components that are related to synchronization.	36
3.12	Receiver components that are related to synchronization.	36
4.1	Functional block and CPU (DSP core) diagram [15].	40
4.2	The C64x CPU block diagram [16].	42
4.3	Code development flow of C6000 (from [18]).	47
4.4	Software-pipelined loop (from [16]).	50
5.1	The proposed synchronizer structure for the receiver.	53
5.2	Structure of J.-C. Lin's symbol timing and fractional carrier frequency synchronization method [21].	54
5.3	BER degradation at 5 ppm sampling clock error.	56
5.4	UL transmitter structure.	59
5.5	UL receiver structure.	60
5.6	Structure of the C program for synchronizer simulation.	61
5.7	Distribution of timing offset estimation errors.	63
5.8	Distribution of timing offset estimation errors using J.-C. Lin's method.	64
5.9	RMSE of symbol timing offset synchronization at SNR = 10 dB.	65
5.10	Symbol time synchronization error distribution under different speeds.	66
5.11	RMSE of fractional CFO synchronization.	67
5.12	Fractional CFO synchronization error distribution under different speeds.	68
5.13	RMSE of fractional CFO synchronization after averaging.	69

5.14	Fractional CFO synchronization error distribution under different speeds after averaging.	70
5.15	Fixed-point data formats used at different points in the transmitter.	71
5.16	Fixed-point data formats used at different points in the receiver.	71
5.17	Implementation of interpolation filter with polyphase decomposition [5].	73
5.18	Convolution kernel at the boundary of a finite-length sequence [7].	73
5.19	Distribution of timing offset estimation errors with fixed-point implementation.	76
5.20	RMSE of symbol timing offset estimation with fixed-point and floating-point implementation.	77
5.21	Symbol timing estimation error distribution under different speeds with fixed-point and floating-point implementation.	78
5.22	RMSE of fractional CFO synchronization with fixed-point and floating-point implementation.	79
5.23	Fractional CFO synchronization error distribution under different speeds with fixed-point and floating-point implementation.	80
5.24	RMSE of fractional CFO synchronization after averaging with fixed-point and floating-point implementation.	81
5.25	Fractional CFO synchronization error distribution under different speeds after averaging with fixed-point and floating-point and floating-point implementation.	82
5.26	BER performance after synchronization at 60 km/h.	83
5.27	A part of C and assembly code for pilot generate function.	84
5.28	A part of C code for allocation function.	85

5.29	Software-pipeline information and a part of assembly code for allocation function.	86
5.30	A part of C code for preamble synchronization function.	86
5.31	Software-pipeline information for preamble synchronization function.	87
5.32	A part of C code for CFO synchronization function.	87
5.33	A part of C code for frequency compensate function.	89
5.34	Software-pipeline information for frequency compensate function.	89
5.35	A part of assembly code for frequency compensate function–I.	90
5.36	A part of assembly code for frequency compensate function–II.	91
5.37	Percentage of DSP loading in the Tx and the Rx.	93
6.1	The proposed OFDMA synchronizer structure.	96
6.2	Multiplication complexity of two algorithms.	99
6.3	Distribution of timing offset estimation errors.	102
6.4	Symbol time synchronization error distribution under different speeds (i). . .	103
6.5	Symbol time synchronization error distribution under different speeds (ii). . .	104
6.6	RMSE of symbol timing offset synchronization.	105
6.7	RMSE of fractional CFO synchronization.	108
6.8	Fractional CFO synchronization error distribution under different speeds. . .	109
6.9	Error probability of integer CFO synchronization in multipath fading channel.	109
6.10	Error probability of preamble index synchronization in multipath fading channel.	110

6.11	Probability of error in either the identified preamble index or the estimated integer CFO.	110
6.12	Probability of error in either the identified preamble index or the estimated integer CFO, in perfect symbol timing.	111
6.13	Probability of error in either the identified preamble index or the estimated integer CFO of two methods.	111
6.14	Distribution of timing offset estimation errors of fine timing synchronization.	112
6.15	Probability of error in either the identified preamble index or the estimated integer CFO with different fine timing synchronization.	113
6.16	Error probability in symbol timing offset estimation with different preambles.	113
6.17	Error probability in integer CFO estimation with different preambles.	114
6.18	Error probability in preamble index identification with different preambles.	114
6.19	CP correlation values of two preambles.	115



List of Tables

2.1	OFDM Advantages and Disadvantages	7
3.1	OFDM Symbol Parameters	13
3.2	2048-FFT OFDMA DL Carrier Allocation Under PUSC	27
3.3	2048-FFT OFDMA DK Carrier Allocation Under FUSC	29
3.4	Transmit Sprctral Mask	34
4.1	The L. and S. Functional Units and Operations Performed [16]	43
4.2	The M. and D. Functional Units and and Operations Performed [16]	44
5.1	ETSI “Vehicular A” Channel Model in Different Units [13]	59
5.2	Receiver SNR Assumptions	60
5.3	Relation Between Speed and Doppler Shift at Carrier Frequency 5 GHz	62
5.4	Ranges of Modulated Signal Values	70
5.5	Profile of Synchronization Function Blocks	92
6.1	OFDMA Receiver SNR Assumptions	101

Chapter 1

Introduction

The IEEE 802.16 WirelessMAN standard provides specifications for an air interface for fixed, portable, and mobile broadband wireless access systems. The standard includes requirements for high data rate line-of-sight (LOS) operation in the 10–66 GHz range for fixed wireless networks as well as requirements for non-line-of-sight (NLOS) fixed, portable, and mobile systems operating in sub-11 GHz licensed and licensed-exempt bands.

The 802.16d upgrade to the 802.16a standard was approved in June 2004 (now code-named 802.16-2004), and primarily introduces some performance enhancement features in the uplink [1]. It consolidates IEEE Std 802.16, IEEE Std 802.16a, and IEEE Std 802.16c, retaining all modes and major features without adding modes [2]. The IEEE 802.16-2004 has been proposed to provide last-mile connectivity to fixed locations by radio links.

Just like IEEE802.16a, IEEE802.16-2004 deploys in two ranges of frequency bands. 10–66 GHz is used for line-of-sight propagation, and the air interface is designed “WirelessMAN-SC.” The 2–11 GHz band, both licensed and license-exempt, is what we are interested in. Design of the 2–11 GHz physical layer is driven by the need for NLOS operation. The three 2–11 GHz air interface specifications in 802.16d are:

- WirelessMAN-SCa: This uses a single-carrier modulation format.

- WirelessMAN-OFDM: This uses orthogonal frequency-division multiplexing (OFDM) with a 256-point transform. Access is by TDMA. This air interface is mandatory for licenseexempt bands.
- WirelessMAN-OFDMA: This uses orthogonal frequency-division multiple access (OFDMA) with a 2048-point transform. In this system, multiple access is provided by addressing a subset of the multiple carriers to individual receivers.

OFDMA is a variation scheme of OFDM, which is a special case of multicarrier transmission that transmits one data stream over a number of subchannels. What makes OFDMA different from OFDM is that multiple users can share one OFDM symbol. It is the combination of OFDM and frequency division multiple access (FDMA), but the guard band of each user could be neglected. OFDMA provides a highly flexible and efficient structure for mutliuser communication.

Mobility enhancements are considered in IEEE 802.16e, which was published in February 2006 [3]. Amendments for the physical (PHY) and medium access control (MAC) layers of 802.16-2004 for mobile operation are being developed by TGe of the 802.16 Working Group. The task group's responsibility is to develop enhancement specifications to the standard to support subscriber stations (SS) moving at vehicular speeds and thereby specify a system for combined fixed and mobile broadband wireless access. Functions to support optional PHY layer structures, mobile-specific MAC enhancements, higher-layer handoff between base stations (BS) or sectors, and security features are among those specified. Operation in mobile mode is limited to licensed bands suitable for mobility between 2 and 6 GHz [4].

The concept of scalable OFDMA is introduced to the IEEE 802.16 WirelessMAN OFDMA mode by the 802.16 TGe. A scalable physical layer enables standard-based solutions to deliver optimum performance in channel bandwidths ranging from 1.25 MHz to 20 MHz with

fixed subcarrier spacing for both fixed and portable/mobile usage models, while keeping the product cost low [4].

Either in the uplink (UL) or in the downlink (DL) direction, the receiver needs to know the exact timing and frequency information of received symbols. By taking advantage of the cyclic prefix or the preamble symbol, we can estimate the symbol timing and frequency offset. We also take fast Rayleigh fading into consideration for the purpose of wireless application.

Generally speaking, our study can be divided to two parts. The first part is based on the IEEE 802.16e OFDM time division duplex (TDD) UL system. We design synchronization algorithms according to the timing and frequency requirements in the IEEE 802.16e standard. In addition to running simulations in fixed and slowly moving channels, we implement the algorithms on Texas Instrument (TI)'s digital signal processor (DSP) employing the Code Composer Studio (CCS). The second part is designing OFDMA TDD DL timing and frequency synchronization algorithms, and running the corresponding simulation for fixed and fast moving channels. In this work, we mainly reference to [5], [6], and [7], where the intents were to design and implement the uplink synchronization scheme of IEEE 802.16a OFDMA by using DSP. Their works also included the implementation of the framing/deframing structure, IFFT/FFT blocks and transmitter/receiver square-root-raised-cosine (SRRC) filters.

This thesis is organized as follows. In chapter 2, we introduce the concepts of OFDM and OFDMA. Chapter 3 introduces IEEE 802.16e WirelessMAN OFDM and OFDMA standards, and gives the system parameters. The transmission filtering is also analyzed in chapter 3. Chapter 4 introduces the DSP platform. Chapter 5 discusses the synchronization issues of OFDM, and also presents the DSP optimization results. The channel environments are described in this chapter for simulations. In chapter 6 we study the synchronization works of OFDMA. Finally, chapter 7 gives the conclusion and future work.


Chapter 2

Overview of OFDM and OFDMA

The material in this chapter is largely taken from the contents of [8] and [9].

2.1 OFDM

2.1.1 Introduction to OFDM



OFDM is a special case of multicarrier transmission, where a single datastream is transmitted over a number of lower rate subcarriers. One of the main reasons to use OFDM is to increase the robustness against frequency selective fading or narrowband interference. In a single carrier system, a single fade or interference can cause the entire link to fail, but in a multicarrier system, only a small percentage of subcarriers will be affected. Error correction coding can then be used to correct for the few erroneous subcarriers.

In a classical parallel data system, the total signal frequency band is divided into non-overlapping frequency channels. It appears good to avoid spectral overlap of channels to eliminate inter-channel interference. However, this leads to inefficient use of the available spectrum. To cope with the inefficiency, the concept of using parallel data transmission by means of frequency division multiplexing (FDM) was published in mid-1960s. The idea was to use parallel data streams and FDM with overlapping carriers. Fig. 2.1 shows the

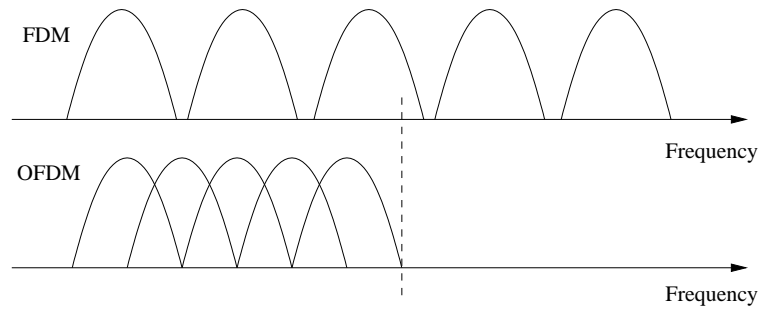


Figure 2.1: Bandwidth efficiency comparison of FDM and OFDM systems (from [5]).

comparison of the bandwidth utilization efficiency for FDM and OFDM. The bandwidth is saved nearly 50%. For a given overall data rate, the increasing number of carriers due to overlapping can reduce the data rate that each individual carrier must convey, and hence lengthen the symbol period. This means that the inter-symbol interference affects a smaller percentage of each symbol. Therefore complex equalization is normally not needed in the receiver.

However, to realize the overlapping multicarrier technique, the crosstalk between carriers is required to be reduced. This means we want orthogonality between the different modulated carriers. Figure 2.2 shows that at the center frequency of each carrier, there is no crosstalk from other carriers.

If the number of subchannels is large, the sinusoidal generators and coherent demodulators required in a parallel system would become extremely expensive and complex. Weinstein and Ebert [10] applied the discrete Fourier transform (DFT) to parallel data transmission system as part of the modulation and demodulation process. By this way, the banks of carrier oscillators and coherent demodulators were eliminated. Moreover, a completely digital implementation could be built around special-purpose hardware performing the fast Fourier transform (FFT).

As mentioned in the previous discussion, the symbol duration can be lengthened such

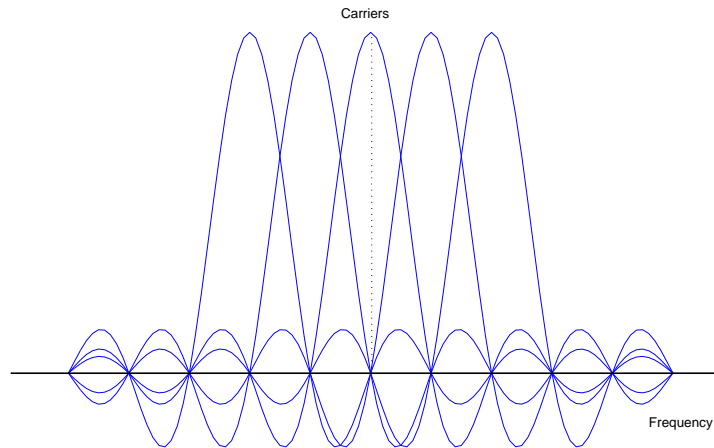


Figure 2.2: The carrier has no crosstalk from other carriers at its center frequency (from [5]).

that the multipath delay relative to the symbol time can be reduced. In order to eliminate the inter-symbol interference (ISI) completely, a guard time (or guard interval, or cyclic prefix) is inserted. Therefore the multipath portion of one symbol will only contaminate the guard interval of the next symbol. For the target data, it is ISI free. The cyclically extended guard interval is to replicate part of the OFDM time-domain waveform from the back to the front to create a guard period. By this way, cyclic convolution can still be applied between the OFDM signal and the channel response to model the transmission system. In addition, the cyclic extension property can be used in synchronization.

Finally, the advantages and disadvantages of OFDM are summarized in Table 2.1. The advantages are already discussed above. The first two disadvantages will be considered in this thesis, while the last two are ignored.

2.1.2 Mathematical Description of OFDM

Before discussing the mathematical definition of the OFDM signal, the orthogonal property is considered first. As described before, the orthogonality is the reason why the carriers can be

Table 2.1: OFDM Advantages and Disadvantages

OFDM advantages	OFDM disadvantages
Efficient to deal with multipath	Sensitive to frequency offset
Enhance channel capacity	Sensitive to timing errors
Robust against narrowband interference	Sensitive to phase noise
	Large peak-to-average power ratio

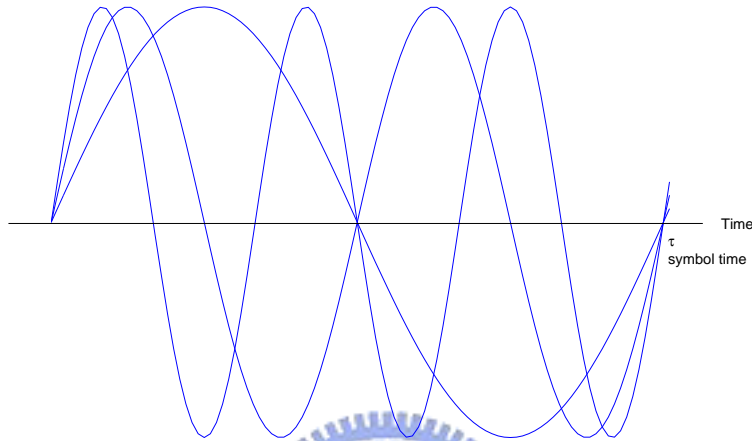


Figure 2.3: Three time domain waveforms of different carriers (from [5]).

overlapped. What is the required condition to maintain the orthogonality property? As the transmitted signal is FDM based, the receiver acts as a bank of demodulators $e^{jw_1}, e^{jw_2}, \dots$, translating each carrier down to DC. The resulting signal is then integrated over a symbol period to recover the raw data. As Fig. 2.3 shows, if the carriers have a whole number of cycles in the symbol period τ , then the integration process results in zero contribution from all the other carriers. Thus the carriers are orthogonal if the carrier spacing is a multiple of $1/\tau$.

The mathematical description of the OFDM system allows us to see how the signal is generated and how receiver must operate. Mathematically, each carrier can be described as a complex wave:

$$S_c(t) = A_c(t)e^{j[w_c t + \phi_c(t)]}. \quad (2.1)$$

The real signal is the real part of $s_c(t)$. Both $A_c(t)$ and $\phi_c(t)$, the amplitude and phase of the carrier, can vary on a symbol by symbol basis. The values of the parameters are constant over the symbol duration τ .

An OFDM signal consists of many carriers. Thus the complex signals $S_s(t)$ is represented by:

$$S_s(t) = \frac{1}{N} \sum_{n=0}^{N-1} A_n(t) e^{j[w_n t + \phi_n(t)]} \quad (2.2)$$

where $w_n = w_0 + n\Delta w$. This is a continuous-time signal. If we consider the waveforms of each component of the signal over one symbol period, then the variables $A_n(t)$ and $\phi_n(t)$ take on fixed values, which depend on the frequency of that particular carrier, and so can be rewritten as constants:

$$A_n(t) \Rightarrow A_n, \phi_n(t) \Rightarrow \phi_n.$$

If the signal is sampled using a sampling frequency of $1/T$, then the resulting signal is represented by:

$$S_s(kT) = \frac{1}{N} \sum_{n=0}^{N-1} A_n e^{j[(w_0 + n\Delta w)kT + \phi_n]}. \quad (2.3)$$

At this point, the time was restricted to be over which the signal can be analyzed to N samples. It is convenient to sample over the period of one data symbol, thus

$$\tau = NT.$$

To simplify the signals, let $w_0 = 0$. Then the signal becomes

$$S_s(kT) = \frac{1}{N} \sum_{n=0}^{N-1} A_n e^{j[(n\Delta w)kT + \phi_n]}. \quad (2.4)$$

As we know, the form of the inverse discrete Fourier transform (IDFT) is

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(w_k) e^{j\frac{2\pi kn}{N}}. \quad (2.5)$$

Since the factor $A_n e^{j\phi_n}$ is constant in the sampled frequency domain, (2.4) and (2.5) are equivalent if

$$\Delta f = \frac{\Delta w}{2\pi} = \frac{1}{NT} = \frac{1}{\tau}, \quad (2.6)$$

which is equivalent to the condition for orthogonality discussed earlier. Thus as a conclusion, using DFT to define the OFDM signal can maintain the orthogonality.

At the transmitter, the signal is defined in the frequency domain. It is a sampled digital signal, and it is defined such that the discrete Fourier spectrum exists only at discrete frequencies. Each OFDM carrier corresponds to one element of this discrete Fourier spectrum. The amplitudes and phases of the carriers depend on the data to be transmitted. The data transitions are synchronized for all carriers, and can be processed together, symbol by symbol.

2.2 OFDMA System

The basic idea of OFDMA is OFDM based frequency division multiple access. In OFDM, a channel is divided into carriers which are used by one user at any time. In OFDMA, the carriers are divided into subchannels. Each subchannel has multiple carriers that form one unit in frequency allocation. In the downlink, a subchannel may be intended for more than one receiver (user). In the uplink, a transmitter (user) may be assigned one or more subchannels, and several transmitters may transmit in parallel. By this way, the bandwidth can be allocated dynamically to the users according to their needs.

An additional advantage of OFDMA is the following. Due to the large variance in a mobile system's path loss, inter-cell interference is a common issue in mobile wireless systems. An OFDMA system can be designed such that subchannels can be composed from several distinct permutations of subcarriers. This enables significant reduction in inter-cell

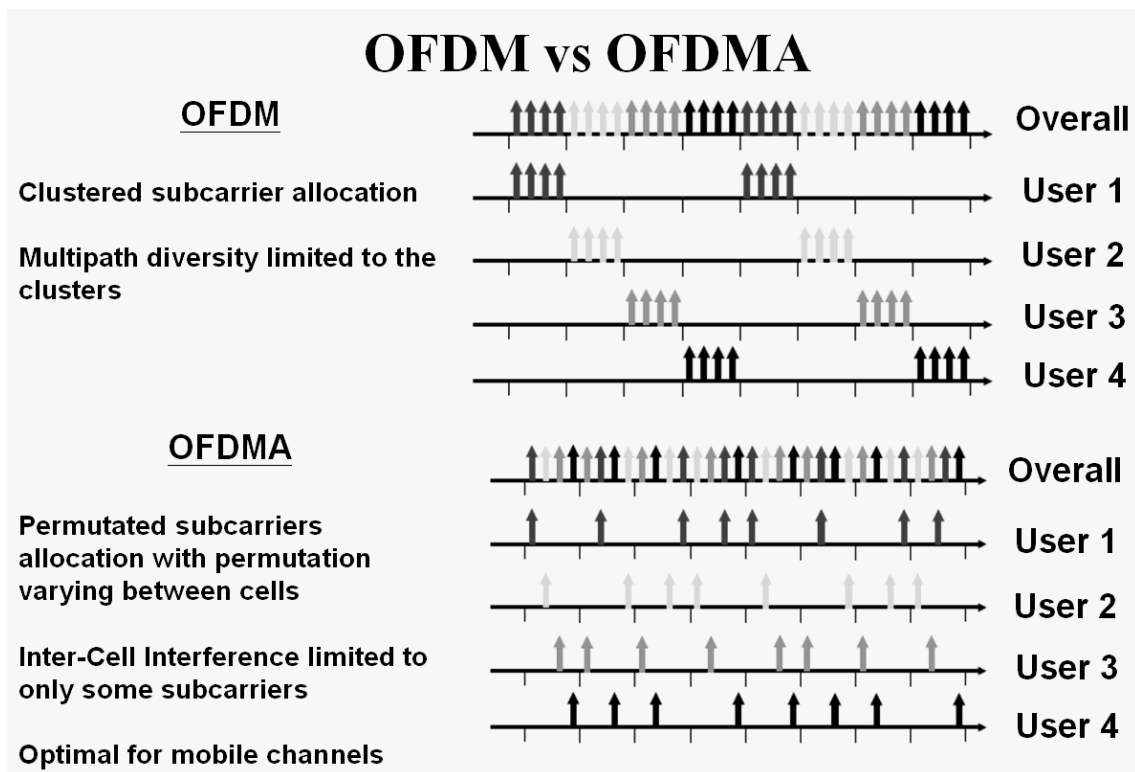


Figure 2.4: Comparison of OFDM and OFDMA subcarriers allocation (from [12]).

interference when the system is not fully loaded, because even on occasions where the same subchannel is used at the same time in two different cells, there is only a partial collision on the actual sub-carriers.

A simple comparison of the subcarrier allocation of OFDM and OFDMA is shown in Fig. 2.4.

In order to support multiple users, the control mechanism becomes more complex. Besides, the OFDMA system has some implementation issues which are more complicated to handle. For example, power control is needed for the uplink to make signals from different users have equal power at the receiver, and all users have to adjust their transmitting time to be aligned. We shall address some issues in the context of IEEE 802.16e.

Chapter 3

Overview of the IEEE 802.16e Standard

This chapter gives an overview of the IEEE 802.16e OFDM and OFDMA systems, and the main references are clauses of [2] and [3]. For the sake of simplicity, we only introduce the specifications that we must use in our study. Other specifications like channel coding, MAP messages, transmit diversity, etc., are not our concern, so we ignore these parts in our introduction. For more details we refer the readers to [2] and [3]. We describe the transmit spectral mask here, and introduce the square-root raised cosine (SRRC) filter used for shaping of the power spectrum and avoiding of the ISI (intersymbol interference).

3.1 Introduction to IEEE 802.16 [11]

Wireless local-area-networks (WLAN) based on the IEEE 802.11 standards have been widely deployed and used in airports, offices and homes. Building on this success, the IEEE 802.16 standard approved in 2001 specifies the air interface and MAC protocol for wireless metropolitan area networks (MANs). The idea there is to provide broadband wireless access to buildings through external antennas communicating with radio base stations (BSs).

To overcome the disadvantage of the LOS requirement between transmitters and receivers

in the 802.16 standard, the 802.16a standard was approved in 2003 to support NLOS links, operational in both licensed and unlicensed frequency bands from 2 to 11 GHz, and subsequently revised to create the 802.16d standard (now code-named 802.16-2004). With such enhancements, the 802.16-2004 standard has been viewed as a promising alternative for providing the last-mile connectivity by radio link. However, the 802.16-2004 specification was devised primarily for fixed wireless users. The 802.16e committee was subsequently formed with the goal of extending the 802.16-2004 standard to support mobile terminals.

The IEEE 802.16e has been published in February 2006, it specifies four air interfaces: WirelessMAN-SC PHY, WirelessMAN-SCa PHY, WirelessMAN-OFDM PHY, and WirelessMAN-OFDMA PHY. What we are interested in in this study are WirelessMAN-OFDM uplink and WirelessMAN-OFDMA downlink.

3.2 WirelessMAN-OFDM TDD Uplink [3]

The WirelessMAN-OFDM PHY is based on OFDM modulation and designed for NLOS operation in frequency bands below 11 GHz.

3.2.1 OFDM Symbol Parameters

The parameters of the transmitted OFDM signal are given in Table 3.1, and some parameter definitions are listed below.

- BW : Nominal channel bandwidth.
- N_{used} : Number of used subcarriers.
- n : Sampling factor. This parameter, in conjunction with BW and N_{used} , determines the subcarrier spacing and the useful symbol time.

Table 3.1: OFDM Symbol Parameters

Parameter	Value
N_{FFT}	256
N_{used}	200
G	1/4, 1/8, 1/16, 1/32
Number of lower frequency guard subcarriers	28
Number of higher frequency guard subcarriers	27
Frequency offset indices of guard subcarriers	-128,-127...,-101,+101,+102...,+127
Frequency offset indices of pilot carriers	-88,-63,-38,-13,13,38,63,88
n	For channel bandwidths that are a multiple of 1.75 MHz then $n = 8/7$, else for channel bandwidths that are a multiple of 1.5 MHz then $n = 86/75$, else for channel bandwidths that are a multiple of 1.25 MHz then $n = 144/125$, else for channel bandwidths that are a multiple of 2.75 MHz then $n = 316/275$, else for channel bandwidths that are a multiple of 2.0 MHz then $n = 57/50$, else for channel bandwidths not otherwise specified then $n = 8/7$

- G : Ratio of CP time to useful time.
- N_{FFT} : Smallest power of two greater than N_{used} .
- Sampling frequency: $F_s = \lfloor n \cdot BW/8000 \rfloor \times 8000$.
- Subcarrier spacing: $\Delta f = F_s/N_{FFT}$.
- Useful symbol time: $T_b = 1/\Delta f$.
- Cyclic prefix (CP) time: $T_g = G \cdot T_b$.
- OFDM symbol time: $T_s = T_b + T_g$.
- Sampling time: T_b/N_{FFT} .

3.2.2 Point-to-Multipoint (PMP) Frame Structure

In licensed bands, the duplexing method shall be either frequency division duplex (FDD) or TDD. FDD SSs may be half-duplex FDD (H-FDD). In licenseexempt bands, the duplexing method shall be TDD. In our study we used only TDD duplexing method. The frame interval contains transmissions (PHY PDUs, where PDU stands for payload data unit) of BS and SSs, gaps and guard intervals.

The OFDM PHY supports a frame-based transmission. A frame consists of a downlink subframe and an uplink subframe. A downlink subframe consists of only one downlink PHY PDU. An uplink subframe consists of contention intervals scheduled for initial ranging and bandwidth request purposes and one or multiple uplink PHY PDUs, each transmitted from a different SS.

A downlink PHY PDU starts with a long preamble, which is used for PHY synchronization. The preamble is followed by a frame control header (FCH) burst. The FCH burst is one OFDM symbol long and is transmitted using BPSK rate 1/2 with the mandatory coding scheme. The FCH contains DLFramePrefix to specify burst profile and length of one or several downlink bursts immediately following the FCH. A DL-MAP message, if transmitted in the current frame, shall be the first MAC PDU in the burst following the FCH. An UL-MAP message shall immediately follow either the DL-MAP message (if one is transmitted) or the downlink frame prefix (DLFP). If uplink channel descriptor (UCD) and downlink channel descriptor (DCD) messages are transmitted in the frame, they shall immediately follow the DL-MAP and UL-MAP messages. Although burst 1 contains broadcast MAC control messages, it is not necessary to use the most robust well-known modulation and coding. A more efficient modulation and coding may be used if it is supported and applicable to all the SSs of a BS.

The FCH is followed by one or multiple downlink bursts. Each downlink burst consists of an integer number of OFDM symbols. With the OFDM PHY, a PHY burst, either a downlink PHY burst or an uplink PHY burst, consists of an integer number of OFDM symbols, carrying MAC messages, i.e., MAC PDUs. To form an integer number of OFDM symbols, unused bytes in the burst payload may be padded by the bytes 0xFF. Then the payload should be randomized, encoded, and modulated using the burst PHY parameters specified by this standard.

In each TDD frame (see Fig. 3.1), the transmit/receive transition gap (TTG) and receive/transmit transition gap (RTG) shall be inserted between the downlink and uplink subframe and at the end of each frame, respectively, to allow the BS to turn around.

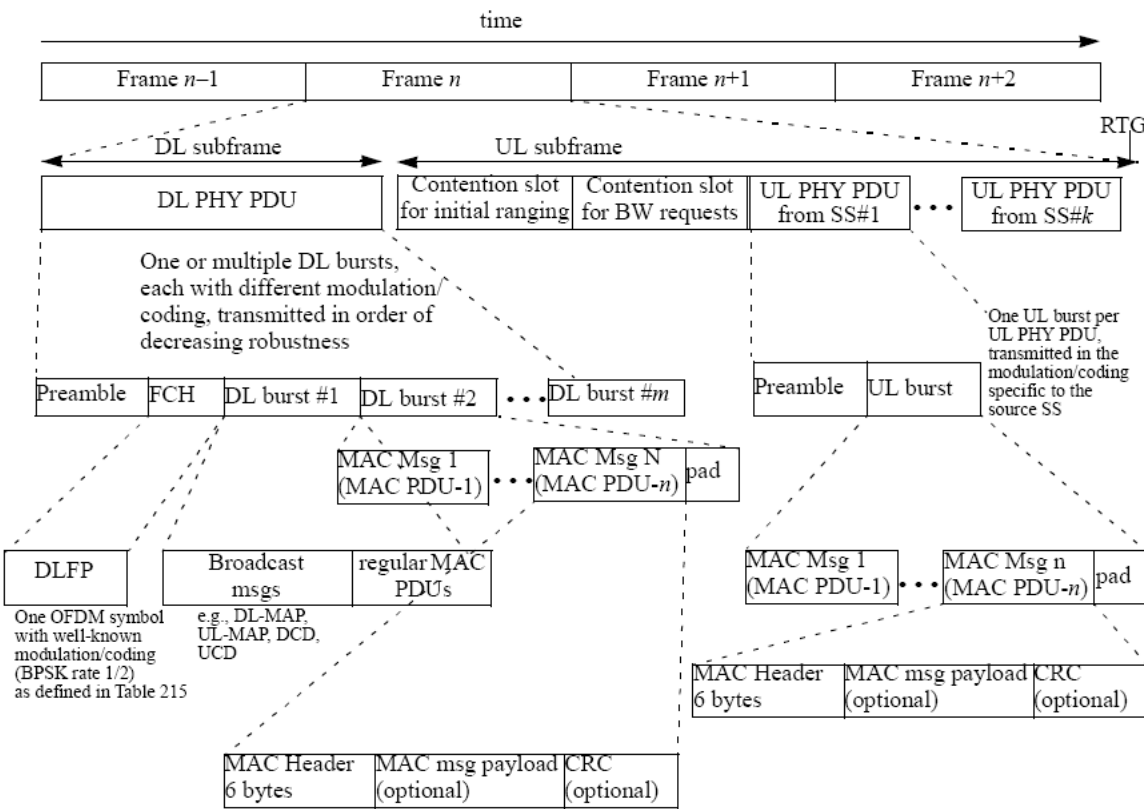


Figure 3.1: OFDM frame structure with TDD (from [2]).

3.2.3 Modulation

Data Modulation

After bit interleaving, the data bits are entered serially to the constellation mapper. The employed modulations are BPSK, Gray-mapped QPSK, 16-QAM, and 64-QAM, whereas the support of 64-QAM is optional for license-exempt bands. The constellations shall be normalized by multiplying the constellation point with the factor c (1 for BPSK, $\frac{1}{\sqrt{2}}$ for QPSK, $\frac{1}{\sqrt{10}}$ for 16-QAM, and $\frac{1}{\sqrt{42}}$ for 64-QAM) to achieve equal average power.

The constellation-mapped data shall be subsequently modulated onto all allocated data subcarriers in order of increasing frequency offset index. The first symbol out of the data constellation mapping shall be modulated onto the allocated subcarrier with the lowest frequency offset index.

Pilot Modulation

Pilot subcarriers shall be inserted into each data burst in order to constitute the symbol and they shall be modulated according to their carrier location within the OFDM symbol. The pseudo-random binary sequence (PRBS) generator depicted in Fig. 3.2 shall be used to produce a sequence, w_k . The polynomial for the PRBS generator is $X^{11} + X^9 + 1$.

The value of the pilot modulation for OFDM symbol k is derived from w_k . On the downlink the index k represents the symbol index relative to the beginning of the downlink subframe. On the uplink the index k represents the symbol index relative to the beginning of the burst. On both uplink and downlink, the first symbol of the preamble is denoted by $k = 0$. The initialization sequences that shall be used on the downlink and uplink are also shown in Fig. 3.2. For each pilot (indicated by frequency offset index), the BPSK modulation shall be derived as follows:

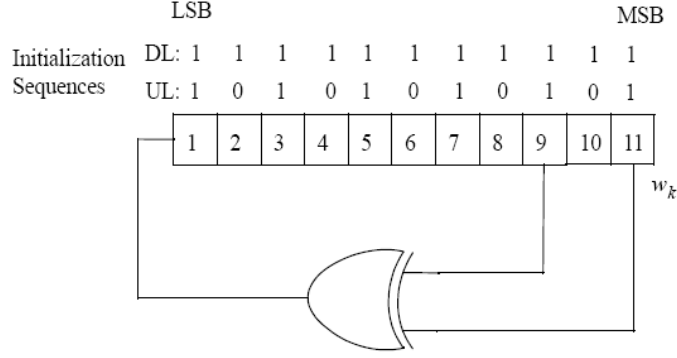


Figure 3.2: PRBS generator for pilot modulation (from [3]).

$$\text{DL: } c_{-88} = c_{-38} = c_{63} = c_{88} = 1 - 2w_k \text{ and } c_{-63} = c_{-13} = c_{13} = c_{38} = 1 - 2\overline{w_k},$$

$$\text{UL: } c_{-88} = c_{-38} = c_{13} = c_{38} = c_{63} = c_{88} = 1 - 2w_k \text{ and } c_{-63} = c_{-13} = 1 - 2\overline{w_k}.$$

3.2.4 Preamble Structure and Modulation

Both DL subframe and UL subframe have the preamble as their first symbol, here we only introduce the UL preamble.

In the uplink, when the entire 16 subchannels are used (which is the assumption in our work), the data preamble, as shown in Fig. 3.3 consists of one OFDM symbol utilizing only even subcarriers. The time domain waveform consists of two times 128 samples preceded by a CP. The subcarrier values shall be set according to the sequence P_{EVEN} . This preamble is referred to as the short preamble. This preamble shall be used as burst preamble on the downlink bursts when indicated in the DL-MAP_IE.

The frequency domain sequence for the two times 128 sequence P_{EVEN} is defined by:

$$P_{EVEN}(k) = \begin{cases} \sqrt{2} \cdot P_{ALL}(k), & k_{mod2} = 0, \\ 0, & k_{mod2} \neq 0. \end{cases} \quad (3.1)$$

The factor of $\sqrt{2}$ gives a 3 dB boost. Please see [2] for the values of the sequence P_{ALL} .

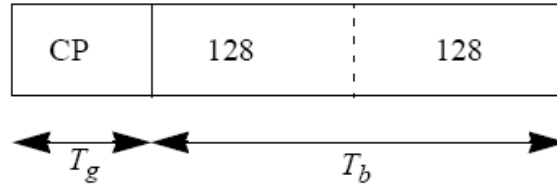


Figure 3.3: Short preamble time domain structure (from [3]).

3.2.5 Frequency and Timing Requirements

Knowing the frequency and timing requirements is very important for synchronization work. At the BS, the transmitted center frequency, receive center frequency and the symbol clock frequency shall be derived from the same reference oscillator. At the BS, the reference frequency tolerance shall be better than $\pm 8 \times 10^{-6}$ in licensed bands up to 10 years from the date of equipment manufacture.

At the SS, both the transmitted center frequency and the symbol sampling clock frequency shall be synchronized and locked to the BS with a tolerance of maximum 2% of the subcarrier spacing for the transmitted center frequency, and 5 ppm for the sampling clock frequency.

During the synchronization period, the SS shall acquire frequency synchronization within the specified tolerance before attempting any uplink transmission. During normal operation, the SS shall track the frequency changes and shall defer any transmission if synchronization is lost.

All SSs shall acquire and adjust their timing such that all uplink OFDM symbols arrive time coincident at the BS to an accuracy of $\pm 50\%$ of the minimum guard-interval or better, which means ± 4 samples or better.

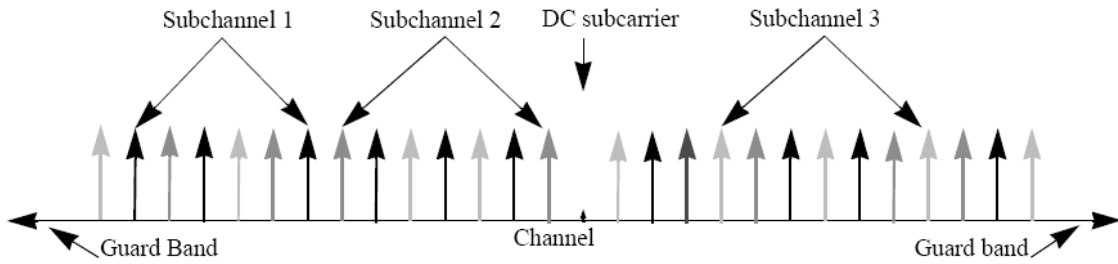


Figure 3.4: OFDMA frequency description (3-channel schematic example, from [2]).

3.3 WirelessMAN-OFDMA TDD Downlink [3]

The specification of OFDMA system is much more complex than the OFDM system. In the OFDMA mode, the active subcarriers are divided into subsets of subcarriers, each subset is termed a subchannel. In the downlink, a subchannel may be intended for different (groups of) receivers; in the uplink, a transmitter may be assigned one or more subchannels, several transmitters may transmit simultaneously. The subcarriers forming one subchannel may, but need not be adjacent. The concept is shown in Fig. 3.4.

3.3.1 OFDMA Basic Terms

Some basic terms we introduce below only appear in OFDMA PHY. These definitions may help readers to understand the concepts of subcarrier allocation of IEEE 802.16e OFDMA.

Slot and Data Region

The definition of an OFDMA slot depends on the OFDMA symbol structure, which varies for uplink and downlink, for FUSC and PUSC, and for the distributed subcarrier permutations and the adjacent subcarrier permutation.

- For downlink FUSC and downlink optional FUSC using the distributed subcarrier

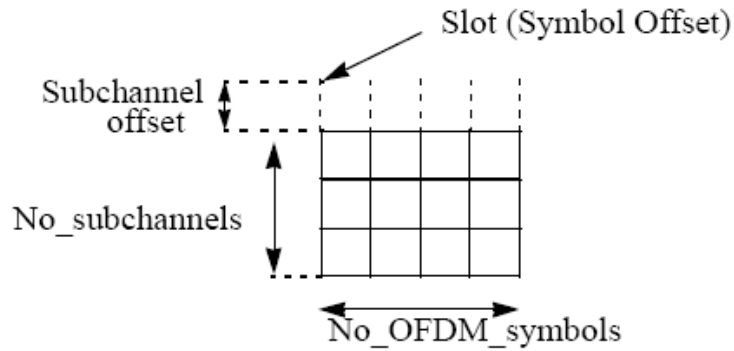


Figure 3.5: Example of the data region which defines the OFDMA allocation (from [2]).

permutation, one slot is one subchannel by one OFDMA symbol. (PUSC and FUSC will be defined later.)

- For downlink PUSC using the distributed subcarrier permutation, one slot is one subchannel by two OFDMA symbols.
- For uplink PUSC using either of the distributed subcarrier permutations, one slot is one subchannel by three OFDMA symbols.

In OFDMA, a Data Region is a two-dimensional allocation of a group of contiguous subchannels, in a group of contiguous OFDMA symbols. All the allocations refer to logical subchannels. This two-dimensional allocation may be visualized as a rectangle, such as the 4×3 rectangle shown in Fig. 3.5.

Segment

A Segment is a subdivision of the set of available OFDMA subchannels (that may include all available subchannels). One segment is used for deploying a single instance of the MAC.

Permutation Zone

Permutation Zone is a number of contiguous OFDMA symbols, in the DL or the UL, that use the same permutation formula. The DL subframe or the UL subframe may contain more than one permutation zone.

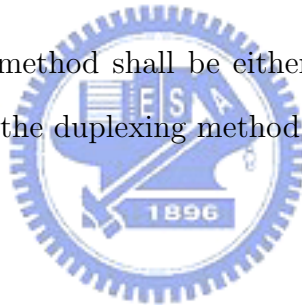
3.3.2 OFDMA Symbol Parameters

All parameters have the same definitions as in OFDM PHY. We do not list them here, but only give their values in later sections.

3.3.3 Frame Structure

Duplexing Modes

In licensed bands, the duplexing method shall be either FDD or TDD. FDD SSSs may be H-FDD. In license-exempt bands, the duplexing method shall be TDD.



PMP Frame Structure

See Fig. 3.6; when implementing a TDD system, the frame structure is built from BS and SS transmissions. Each frame in the downlink transmission begins with a preamble followed by a DL transmission period and an UL transmission period. In each frame, the TTG and RTG shall be inserted between the downlink and uplink and at the end of each frame, respectively, to allow the BS to turn around.

Subchannel allocation in the downlink may be performed in the following ways: Partial usage of subchannels (PUSC) where some of the subchannels are allocated to the transmitter, and full usage of the subchannels (FUSC) where all subchannels are allocated to the transmitter. The downlink frame shall start in PUSC mode with no transmit diversity. The

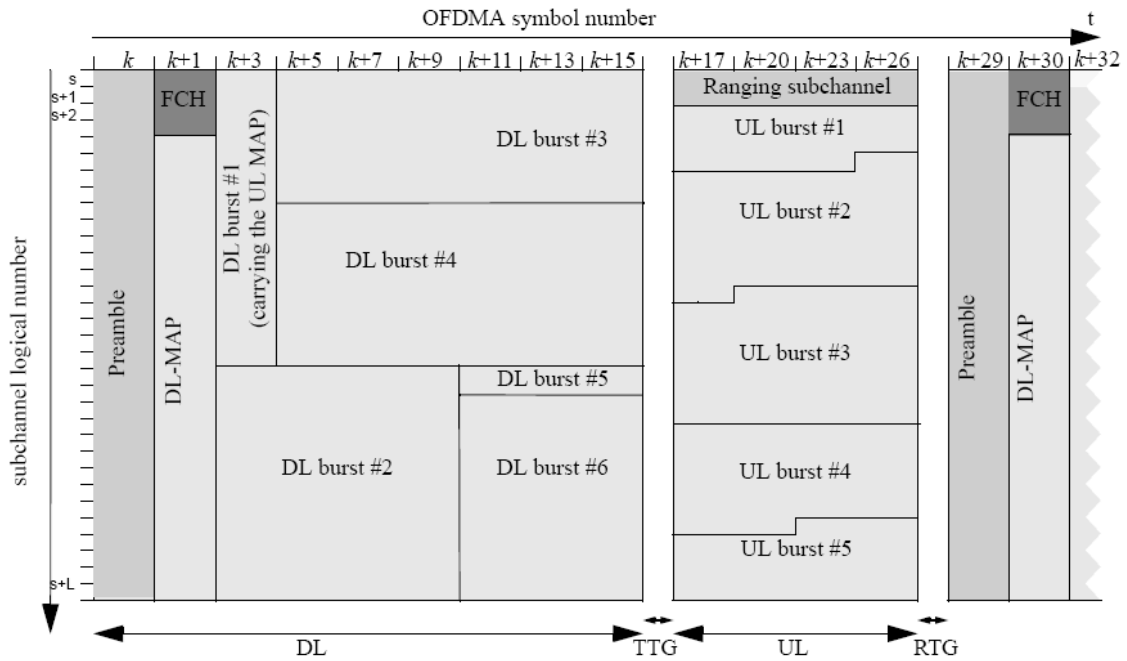


Figure 3.6: Example of an OFDMA frame (with only mandatory zone) in TDD mode (from [3]).

FCH shall be transmitted using QPSK rate 1/2 with four repetitions using the mandatory coding scheme (i.e., the FCH information will be sent on four subchannels with successive logical subchannel numbers) in a PUSC zone. The FCH contains the DL_Frame_Prefix, and specifies the length of the DL-MAP message that immediately follows the DL_Frame_Prefix and the repetition coding used for the DL-MAP message.

The transitions between modulations and coding take place on slot boundaries in time domain (except in AAS zone, where AAS stands for adaptive antenna system) and on subchannels within an OFDMA symbol in frequency domain. The OFDMA frame may include multiple zones (such as PUSC, FUSC, PUSC with all subchannels, optional FUSC, AMC, TUSC1, and TUSC2, where AMC stands for adaptive modulation and coding, and TUSC stands for tile usage of subchannels), the transition between zones is indicated in the DL-Map. The PHY parameters (such as channel state and interference levels) may change from

one zone to the next.

The maximum number of downlink zones is 8 in one downlink subframe. For each SS, the maximum number of bursts to decode in one downlink subframe is 64. This includes all bursts without connection identifier (CID) or with CIDs matching the SS's CIDs.

Allocation of Subchannels for FCH and DL-MAP, and Logical Subchannel Numbering

In PUSC, any segment used shall be allocated at least the same number of subchannels as in subchannel group #0. For FFT sizes other than 128, the first 4 slots in the downlink part of the segment contain the FCH as defined before. These slots contain 48 bits modulated by QPSK with coding rate 1/2 and repetition coding of 4. For FFT-128, the first slot in the downlink part of the segment is dedicated to FCH and repetition is not applied. The basic allocated subchannel sets for Segments 0, 1, and 2 are subchannel group #0, #2, and #4, respectively. Fig. 3.7 depicts this structure.

After decoding the DL.Frame.Prefix message within the FCH, the SS has the knowledge of how many and which subchannels are allocated to the PUSC segment. In order to observe the allocation of the subchannels in the downlink as a contiguous allocation block, the subchannels shall be renumbered. The renumbering, for the first PUSC zone, shall start from the FCH subchannels (renumbered to values 0–11), then continue numbering the subchannels in a cyclic manner to the last allocated subchannel and from the first allocated subchannel to the FCH subchannels. Fig. 3.8 gives an example of such renumbering for segment 1.

For uplink, in order to observe the allocation of the subchannels as a contiguous allocation block, the subchannels shall be renumbered, and the renumbering shall start from the lowest numbered allocated subchannel (renumbered to value 0), up to the highest numbered allocated subchannel, skipping nonallocated subchannels.

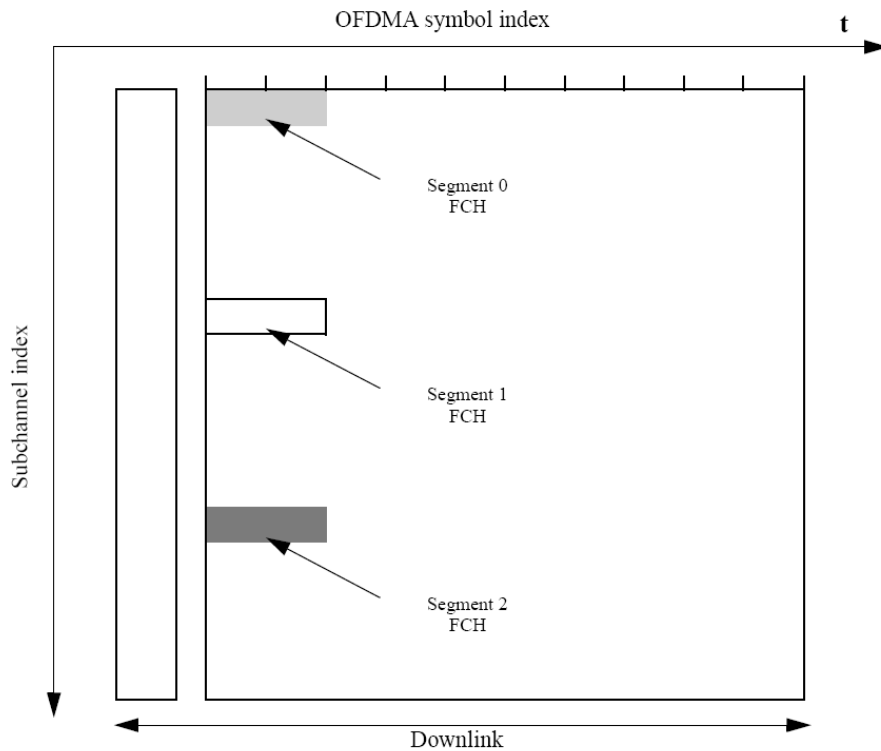


Figure 3.7: FCH subchannel allocation for all 3 segments (from [3]).

The DL-MAP of each segment shall be mapped to the slots allocated to the segment in a frequency first order, starting from the slot after the FCH (subchannel 4 in the first symbol, after renumbering), and continuing to the next symbols if necessary. The FCH of segments that have no subchannels allocated (unused segments) will not be transmitted, and the respective slots may be used for transmission of MAP and data of other segments.

3.3.4 OFDMA Downlink Subcarrier Allocation

Here we only describe DL subcarrier allocation since our study of OFDMA PHY is only on DL. For both uplink and downlink, these used subcarriers are allocated to pilot subcarriers and data subcarriers. However, there is a difference between the different possible zones. For FUSC and PUSC, in the downlink, the pilot tones are allocated first; what remains are data

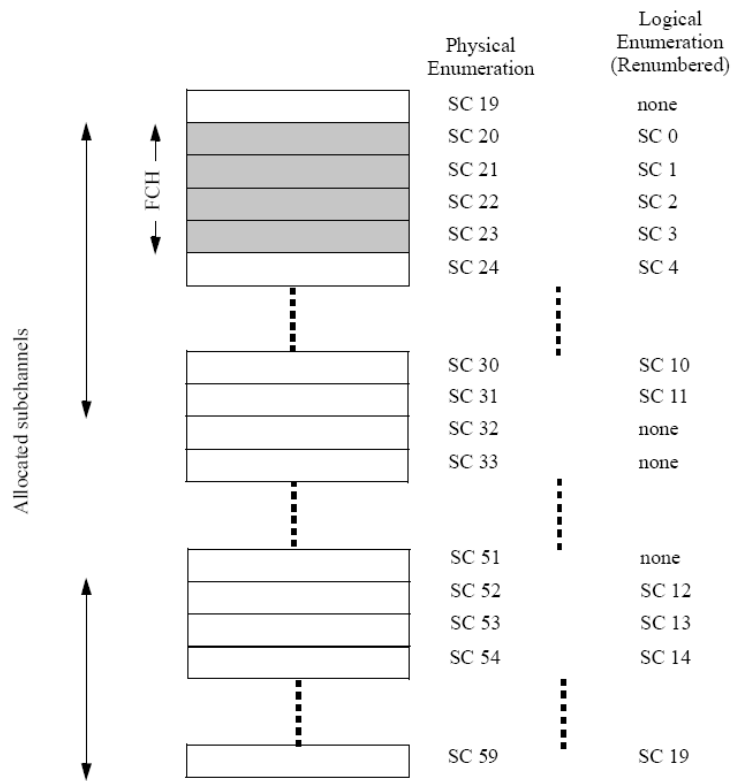


Figure 3.8: Example of DL renumbering the allocated subchannels for segment 1 in PUSC (from [3]).

subcarriers, which are divided into subchannels that are used exclusively for data. Thus, in FUSC, there is one set of common pilot subcarriers, and in PUSC of the downlink, there is one set of common pilot subcarriers in each major group.

The downlink can be divided into a three segment structure and includes a preamble which begins the transmission. In this preamble, subcarriers are divided into three carrier-sets. There are three possible groups consisting of a carrier-set each, that may be used by any segment.

Preamble

The first symbol of the downlink transmission is the preamble. There are 3 types of preamble carriersets, which are defined by allocation of different subcarriers for each one of them. The subcarriers are modulated using a boosted BPSK modulation with a specific pseudo-noise (PN) code. The preamble carrier-sets are defined using

$$PreambleCarrierSet_n = n + 3 \cdot k \quad (3.2)$$

where:

$PreambleCarrierSet_n$ specifies all subcarriers allocated to the specific preamble,

n is the number of the preamble carrier-set indexed 0–2,

k is a running index 0–567.

For the preamble symbol there will be 172 guard band subcarriers on the left side and the right side of the spectrum. Each segment uses a preamble composed of a carrier-set out of the three available carrier-sets in the following manner that segment i uses preamble carrier-set i , where $i = 0, 1, 2$. In the case of segment 0, the DC carrier will not be modulated at all and the appropriate PN will be discarded; therefore, DC carrier shall always be zeroed. Therefore, each segment eventually modulates each third subcarrier.

The 114 different PN series modulating the preamble carrier-set are defined in Table 309 of [2] for the 2k FFT mode. The series modulated depends on the segment used and the IDcell parameter.

Symbol Structure for PUSC

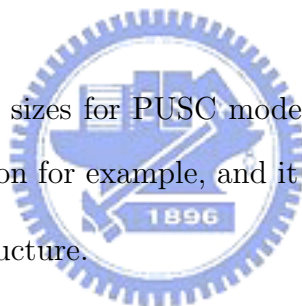
The symbol is first divided into basic clusters and zero carriers are allocated. Pilots and data carriers are allocated within each cluster. Table 310 of [3] summarizes the parameters of the

Table 3.2: 2048-FFT OFDMA DL Carrier Allocation Under PUSC

Parameter	Value	Comments
Number of DC subcarriers	1	Index 1024 (counting from 0)
Number of guard subcarriers, left	184	
Number of guard subcarriers, right	183	
Number of used subcarriers, N_{used}	1681	Including all possible pilots and DC
Number of subcarriers per cluster	14	
Number of clusters	120	
Renumbering sequence	1	Used to renumber clusters before allocation to subchannels, see [3]
Number of data subcarriers in each symbol per subchannel	4	
Number of subchannels	60	
Basic permutation sequence 12 (for 12 subchannels)		6,9,4,8,10,11,5,2,7,3,1,0
Basic permutation sequence 8 (for 8 subchannels)		7,4,0,2,1,5,3,6

symbol structure of different FFT sizes for PUSC mode. Here we only take the 2048-FFT OFDMA downlink carrier allocation for example, and it is summarized in Table 3.2.

Fig. 3.9 depicts the cluster structure.



Downlink Subchannels Subcarrier Allocation in PUSC

The carrier allocation to subchannels is performed using the following procedure:

1. Dividing the subcarriers into the number of clusters ($N_{clusters}$), where the physical clusters contain 14 adjacent subcarriers each (starting from carrier 0). The number of clusters varies with the FFT size.
2. Renumbering the physical clusters into logical clusters using the following formula:

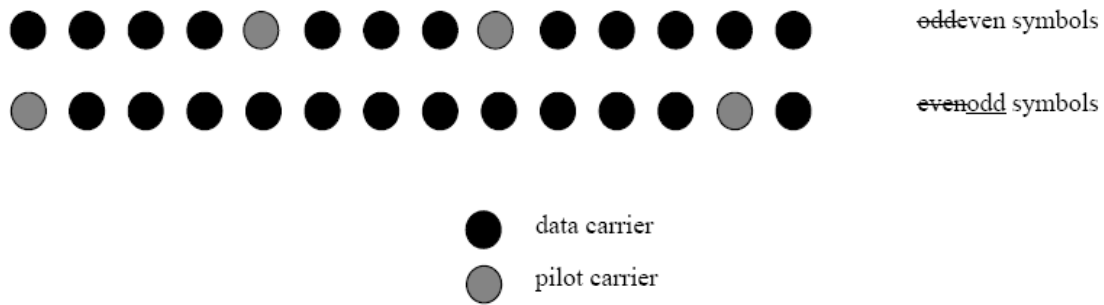


Figure 3.9: Cluster structure (from [3]).

$LogicalCluster =$

$$\left\{ \begin{array}{ll} RenumberingSequence(PhysicalCluster), & \text{First DL zone, or Use All SC indicator} \\ & = 0 \text{ in } STC_DL_Zone_IE, \\ RenumberingSequence((PhysicalCluster) + & \text{Otherwise.} \\ 13 \cdot DL_PermBase) \bmod N_{clusters}, & \end{array} \right. \quad (3.3)$$

In the first PUSC zone of the downlink (first downlink zone) and in a PUSC zone defined by $STC_DL_ZONE_IE()$ with “use all SC indicator = 0”, the default re-numbering sequence is used for logical cluster definition. For all other cases $DL_PermBase$ parameter in the $STC_DL_Zone_IE()$ or $AAS_DL_IE()$ shall be used.

3. Allocating logical clusters to groups. The allocation algorithm varies with FFT sizes. For FFT size = 2048, dividing the clusters into six major groups. Group 0 includes clusters 0–23, group 1 clusters 24–39, group 2 clusters 40–63, group 3 clusters 64–79, group 4 clusters 80–103, and group 5 clusters 104–119. These groups may be allocated to segments; if a segment is being used, then at least one group shall be allocated to it. By default group 0 is allocated to sector 0, group 2 to sector 1, and group 4 to sector 2).
4. Allocating subcarriers to subchannels in each major group, which is performed separately for each OFDMA symbol by first allocating the pilot carriers within each cluster,

Table 3.3: 2048-FFT OFDMA DK Carrier Allocation Under FUSC

Parameter	Value	Comments
Number of DC subcarriers	1	Index 1024 (counting from 0)
Number of guard subcarriers, left	173	
Number of guard subcarriers, right	172	
Number of used subcarriers, N_{used}	1703	Including all possible pilots and DC
Pilot sets	166	See Table 311 of [3]
Number of data subcarriers	1536	
Number of data subcarriers per subchannel	48	
Number of Subchannels	32	
Basic permutation sequence		3,18,2,8,16,10,11,15,26,22,6,9,27,20,25,1,29,7,21,5,28,31,23,17,4,24,0,13,12,19,14,30

and then taking all remaining data carriers within the symbol and using the same procedure described in the next subsection (Symbol Structure for FUSC). The parameters vary with FFT sizes. For FFT size = 2048, use the parameters from Table 3.2, with basic permutation sequence 12 for even numbered major groups and basic permutation sequence 8 for odd numbered major groups, to partition the subcarriers into subchannels containing 24 data subcarriers in each symbol.

Symbol Structure for FUSC

The symbol structure is constructed using pilots, data, and zero subcarriers. The symbol is first allocated with the appropriate pilots and with zero subcarriers, and then all the remaining subcarriers are used as data subcarriers (which are divided into subchannels).

There are two variable pilot-sets and two constant pilot-sets. In FUSC, each segment uses both sets of variable/constant pilot-sets. We only summarize the parameters of 2048-FFT OFDMA in Table 3.3.

The Variable set of pilots embedded within the symbol of each segment shall obey the

following rule:

$$PilotLocation = VariableSet\#x + 6 \cdot (FUSC_SymbolNumber \bmod 2) \quad (3.4)$$

where FUSC_SymbolNumber counts the FUSC symbols used in the current zone starting from 0.

Downlink Subchannels Subcarrier Allocation

Each subchannel is composed of 48 subcarriers. The subchannel indices are formulated using a Reed-Solomon series, and is allocated out of the data subcarriers domain. The data subcarriers domain includes $48 \times 32 = 1536$ subcarriers.

After mapping all pilots, the remainder of the used subcarriers are used to define the data subchannels. To allocate the data subchannels, the remaining subcarriers are partitioned into groups of contiguous subcarriers. Each subchannel consists of one subcarrier from each of these groups. The number of groups is therefore equal to the number of subcarriers per subchannel, and it is denoted $N_{subcarriers}$. The number of the subcarriers in a group is equal to the number of subchannels, and it is denoted $N_{subchannels}$. The number of data subcarriers is thus equal to $N_{subcarriers} \cdot N_{subchannels}$.

The exact partitioning into subchannels is according to the permutation formula:

$$subcarrier(k, s) =$$

$$N_{subchannels} \cdot n_k + \{p_s[n_k \bmod N_{subchannels}] + DL_PermBase\} \bmod N_{subchannels} \quad (3.5)$$

where:

$subcarrier(k, s)$ is the subcarrier index of subcarrier k in subchannel s ,

s is the index number of a subchannel, from the set $\{0, \dots, N_{subchannels} - 1\}$,

$n_k = (k + 13 \cdot s) \bmod N_{subcarriers}$, where k is the subcarrier-in-subchannel index from the set $\{0, \dots, N_{subcarriers} - 1\}$,

$N_{subchannels}$ is the number of subchannels (for PUSC use number of subchannels in the currently partitioned major group),

$p_s[j]$ is the series obtained by rotating basic permutation sequence cyclically to the left s times,

DL_PermBase is an integer ranging from 0 to 31, which is set to the preamble IDCell in the first zone and determined by the DL-MAP for other zones.

On initialization, an SS must search for the downlink preamble. After finding the preamble, the user shall know the IDcell used for the data subchannels.

3.3.5 Modulation

Subcarrier Randomization



The PRBS generator, as known in Fig. 3.2, shall be used to produce a sequence w_k . The value of the pilot modulation on subcarrier k shall be derived from w_k .

The initialization vector of the PRBS generator for both uplink and downlink shall be designated b10..b0, such that:

b0..b4 = five least significant bits of IDcell as indicated by the frame preamble in the first downlink zone and in the downlink AAS zone with Diversity_Map support, DL_PermBase following STC_DL_Zone_IE() and 5 LSB of DL_PermBase following AAS_DL_IE without Diversity_Map support in the downlink. Five least significant bits of IDcell (as determined by the preamble) in the uplink. For downlink and uplink, b0 is MSB and b4 is LSB, respectively.

b5..b6 = set to the segment number + 1 as indicated by the frame preamble in the first downlink zone and in the downlink AAS zone with Diversity_Map support, PRBS_ID as indicated by the STC_DL_Zone_IE or AAS_DL_IE without Diversity_Map support in other downlink zone. 0b11 in the uplink. For downlink and uplink, b5 is MSB and b6 is LSB, respectively.

b7..b10 = 0b1111 (all ones) in the downlink and four LSB of the Frame Number in the uplink. For downlink and uplink, b7 is MSB and b10 is LSB, respectively.

Data Modulation

After the repetition block, the data bits are entered serially to the constellation mapper. Gray-mapped QPSK and 16-QAM shall be supported, whereas the support of 64-QAM is optional. The Gray-mapped modulations are the same as modulations in OFDM PHY.

Pilot Modulation

In all permutations except uplink PUSC and downlink TUSC1, each pilot shall be transmitted with a boosting of 2.5 dB over the average non-boosted power of each data tone. The pilot subcarriers shall be modulated according to:

$$\Re\{c_k\} = \frac{8}{3}(\frac{1}{2} - w_k) \cdot p_k, \quad \Im\{c_k\} = 0. \quad (3.6)$$

where p_k is the pilot's polarity for SDMA (stands for spatial division multiple access) allocations in AMC AAS zone, and $p = 1$ otherwise.

Preamble Pilot Modulation

The pilots in the downlink preamble shall be modulated according to:

$$\Re\{PreamblePilotModulation\} = 4 \cdot \sqrt{2} \cdot (\frac{1}{2} - w_k), \quad (3.7)$$

$$\mathfrak{S}\{PreamblePilotModulation\} = 0. \quad (3.8)$$

3.3.6 Frequency and Timing Requirements

Timing Requirements

For any duplexing, all SSs shall acquire and adjust their timing such that all uplink OFDMA symbols arrive time coincident at the BS to a accuracy of $\pm 25\%$ of the minimum guard-interval or better. This translates into ± 16 samples in the case of 2048-FFT OFDMA.

Frequency Requirements

At the BS, the transmitted center frequency, receive center frequency, and the symbol clock frequency shall be derived from the same reference oscillator. At the BS, the reference frequency accuracy shall be better than $\pm 2 \times 10^{-6}$.

At the SS, both the transmitted center frequency and the sampling frequency shall be derived from the same reference oscillator. Thereby, the SS uplink transmission shall be locked to the BS, so that its center frequency shall deviate no more than 2% of the subcarrier spacing, compared to the BS center frequency.

During the synchronization period, the SS shall acquire frequency synchronization within the specified tolerance before attempting any uplink transmission. During normal operation, the SS shall track the frequency changes by estimating the downlink frequency offset and shall defer any transmission if synchronization is lost. To determine the transmit frequency, the SS shall accumulate the frequency offset corrections transmitted by the BS (for example in the RNG-RSP message), and may add to the accumulated offset an estimated UL frequency offset based on the downlink signal.

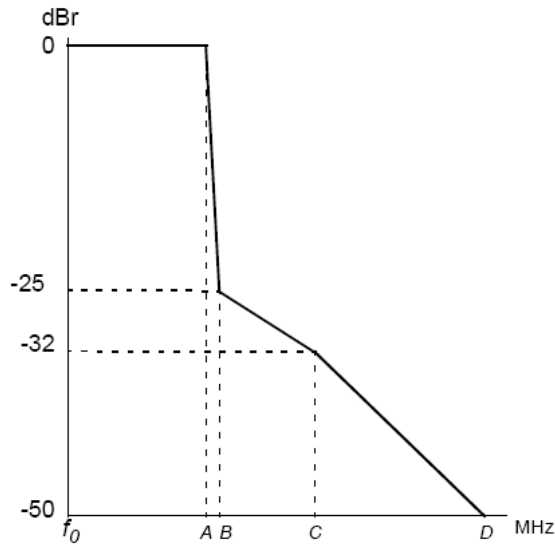


Figure 3.10: Transmit spectral mask (from [2]).

Table 3.4: Transmit Spectral Mask

Bandwidth (MHz)	A	B	C	D
10	9.5	10.9	19.5	29.5
20	4.75	5.45	9.75	14.75

3.4 Transmit Spectral Mask

Due to requirement of bandwidth-limited transmission, the transmitted spectral density of the transmitted signal shall fall within the spectral mask as shown in Fig. 3.10 and Table 3.4 in license-exempt bands. The measurements shall be made using 100 kHz resolution bandwidth and a 30 kHz video bandwidth. The 0 dBm level is the maximum power allowed by the relevant regulatory body. IEEE 802.16e does not specify the power mask for the license bands.

3.5 System Parameters

The standard is very flexible in choice of bandwidth and cyclic prefix length. However, it would be difficult to conduct the simulation and implementation study without a particular set of parameters. Hence we pick the set of parameters shown in this section.

3.5.1 Uplink OFDM Transmission Parameters

There are a number of system profiles defined in IEEE 802.16e standard, each characterized by five components: a MAC profile, a PHY profile, a RF profile, a duplexing selection, and a power class. The system profile we choose is PMP, WirelessHUMAN(-OFDM) PHY profile with 10 MHz channelization, TDD, and SISO operation for the uplink OFDM transmission. We assume a carrier frequency of 5GHz. Knowing the bandwidth, we can compute the other parameters as given in the last chapter:

- BW : 10 MHz (license-exempt band usage only).
- n : 57/50.
- G : 8.
- Sampling Frequency: 11.4 MHz.
- Subcarrier spacing: 44.53125 kHz.
- Useful symbol time: $22\frac{2}{9} \mu s$.
- CP time: $2\frac{7}{9} \mu s$.
- OFDM symbol time: $25 \mu s$.
- Sampling time: 86.81 ns.



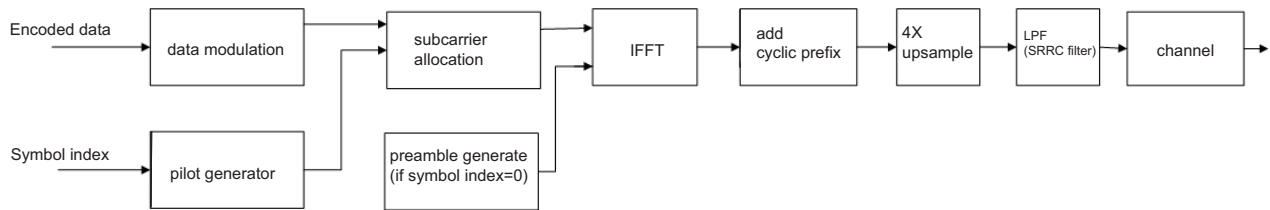


Figure 3.11: Transmitter components that are related to synchronization.

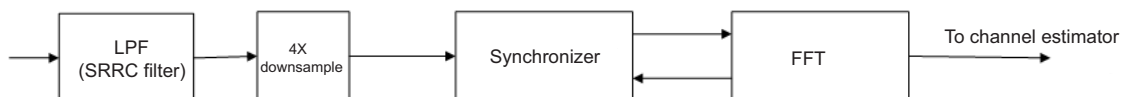


Figure 3.12: Receiver components that are related to synchronization.

The modulation could be BPSK, QPSK, 16-QAM, or 64-QAM by random generated binary data. The frame duration could be 5, 10, or 20 ms.

The transmitter and receiver components that are related to synchronization are shown in Figs. 3.11 and 3.12. In actual simulation, we may use an interpolator to change the sampling rate by an appropriate factor. In our study, we upsample by 4.

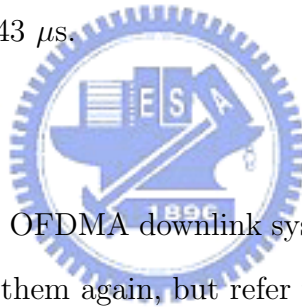
3.5.2 Downlink OFDMA Transmission Parameters

Like OFDM PHY, the OFDMA PHY also defines system profiles for systems operating with the WirelessMAN-OFDMA and WirelessHUMAN-OFDMA air interfaces. The system profile we select is PMP, WirelessHUMAN(-OFDMA) 10 MHz channel basic PHY profile, TDD, and SISO operation for the downlink OFDM transmission. The FFT size is 2048, and the carrier frequency is 3.5 GHz. We choose the PUSC permutation in our simulation, and use segment 0 with subchannel 0–19 to allocate data subcarriers.

The modulation could be QPSK, 16-QAM, or 64-QAM by random generated binary data.

The frame duration could be 2.5, 5, or 8 ms. Other parameter values are as follows:

- BW : 10 MHz (license-exempt band usage only).
- n : 28/25.
- G : 8.
- Sampling frequency: 11.2 MHz.
- Subcarrier spacing: 5.46875 kHz.
- Useful symbol time: 182.8571 μ s.
- CP time: 22.8571 μ s.
- OFDM symbol time: 205.7143 μ s.
- Sampling time: 89.2587 ns.



The transceiver components of the OFDMA downlink system are very similar to the OFDM uplink system, so we do not show them again, but refer the reader to Figs. 3.11 and 3.12.

3.6 Transmission Filters [7]

Reference [5] contains a detailed discussion on how to choose a suitable SRRC (square-root raised cosine) transmission filter. We use the filter designed in [5] directly. Below we give a simple introduction based on [5].

To avoid the complexity of an ideal lowpass filter and to simulate path delays at non-integer sample times, an interpolator is added to the transmitter to yield 4-times oversampled

transmitter output. The SRRC filter is used as the lowpass interpolation filter. The impulse response of this filter is given by

$$SRRC(t) = \frac{\sin\left(\pi\frac{t}{T_{sample}}(1-\alpha)\right) + 4\alpha\frac{t}{T_{sample}}\cos\left(\pi\frac{t}{T_{sample}}(1+\alpha)\right)}{\pi\frac{t}{T_{sample}}\left(1 - \left(4\alpha\frac{t}{T_{sample}}\right)^2\right)},$$

where α is the roll-off factor. One reason for adopting the SRRC filter is that for this filter the transmitter and the receiver filters are matched to each other and there is no inter-sample interference introduced by the filter when fully synchronized. The roll-off factor chosen is 0.155 which results in a filter of 57 taps, which is chosen to satisfy the power mask specified in 802.16a [5].



Chapter 4

Introduction to the DSP Implementation Platform

In this chapter, we introduce the DSP platform utilized in our implementation. The platform includes a DSP chip, and Texas Instruments (TI)'s code development environment. Note that although we just perform software implementation of the OFDM PHY UL system, we also need to know something about the DSP hardware environment. This chapter is mainly taken from chapters 3 and 4 of [7].



4.1 The DSP Chip [16]

The DSP chip on the load, TI's TMS320C6416, employs the "VelociTI" architecture, a variant of the traditional VLIW architecture, which consists of multiple execution units running in parallel, performing multiple instructions during one cycle time. It is a 32-bit fixed-point DSP, with processing speed at 600 MHz, delivering 4800 MIPS.

The C6416 core CPU, which is shown in Fig. 4.1, consists of 64 general-purpose 32-bit registers and eight functional units. These eight functional units contain two multipliers and six arithmetic units. It allows users to develop highly effective RISC-like code for fast development time.

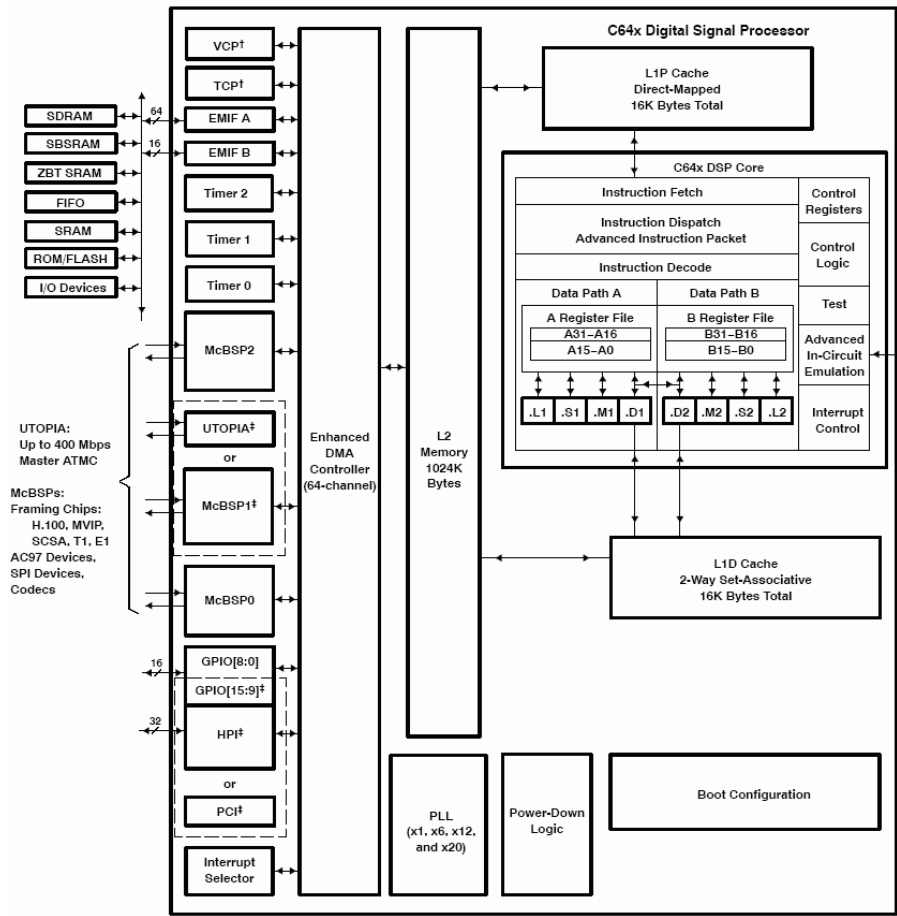


Figure 4.1: Functional block and CPU (DSP core) diagram [15].

The C6416 uses a two-level cache-based architecture with 16 kB of L1 data cache, 16 kB of L1 program cache, and 1 MB of L2 data/program cache. On-chip peripherals include two multichannel buffered serial ports (McBSPs), two timers, a 16-bit host port interface (HPI), a 32-bit external memory interface (EMIF), a direct memory access (DMA) controller and an enhanced direct memory access (EDMA) controller.

The following gives some sketch of the units just mentioned above:

- The EDMA controller transfers data between the memory without passing through the DSP core.

- McBSPs can buffer serial samples in memory automatically with the aid of the DMA/EDMA controller.
- HPI is a parallel port through which a host processor can directly access the CPU's memory space.
- EMIF provides the interface for the DSP core to connect with several external devices, allowing additional data and program space.

The C6416 has two 64-bit internal ports to access internal data memory. It supports double word loads and stores. There are four 32-bit paths for loading/storing data from memory to the register file. C6416 has two register files (A and B), each containing 32 32-bit registers for a total of 64 general-purpose registers. The general-purpose registers can be used for data, data address pointers, or condition registers. The C6416 register file supports packed 8-bit types and 64-bit fixed-point data types. Packed data types store either four 8-bit values or two 16-bit values in a single 32-bit register, or four 16-bit values in a 64-bit register pair. Note that the C6416 does not directly support floating-point data types.

The eight functional units in the C6416 data paths can be divided into two groups of four; each functional unit in one data path is almost identical to the corresponding unit in the other data path. The two sets of functional units, along with two register files, compose sides A and B of the DSP core. Fig. 4.2 illustrates the C6416 DSP CPU. From this figure, we see that the C6416 CPU contains:

- Program fetch unit.
- Instruction dispatch unit, with advanced instruction packing.
- Instruction decode unit.

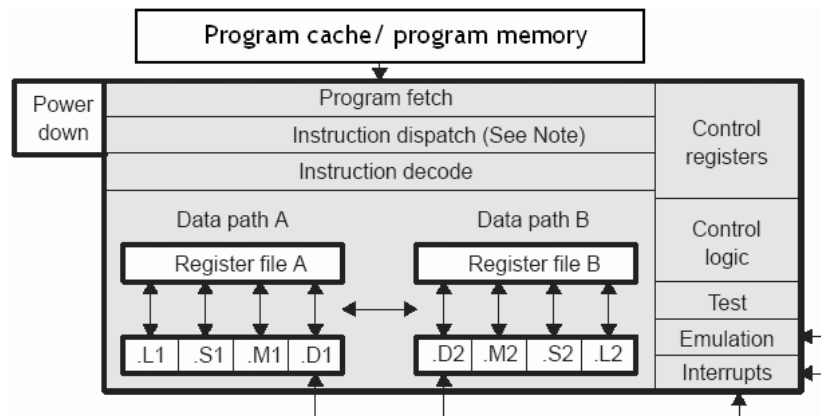


Figure 4.2: The C64x CPU block diagram [16].

- Control registers.
- Control logic.
- Test, emulation, and interrupt logic.

The details of each functional units are given in Table 4.1 and 4.2. Most data lines in the CPU support 32-bit operands, and some support long (40-bit) and double word (64-bit) operands. Each functional unit has its own 32-bit write port into a general-purpose register file. All units ending in 1 (for example, .L1) write to register file A, and all units ending in 2 write to register file B. Each functional unit has two 32-bit read ports for source operands src1 and src2. Four units (.L1, .L2, .S1, and .S2) have an extra 8-bit-wide port for 40-bit long writes, as well as an 8-bit input for 40-bit long reads. Because each unit has its own 32-bit write port, when performing 32-bit operations all eight units can be used in parallel every cycle.

Table 4.1: The L. and S. Functional Units and Operations Performed [16]

Functional Unit	Fixed-Point Operations
.L unit (.L1, .L2)	32/40-bit arithmetic and compare operations 32-bit logical operations Leftmost 1 or 0 counting for 32 bits Normalization count for 32 and 40 bits Byte shifts Data packing/unpacking 5-bit constant generation Dual 16-bit arithmetic operations Quad 8-bit arithmetic operations Dual 16-bit min/max operations Quad 8-bit min/max operations
.S unit (.S1, .S2)	32-bit arithmetic operations 32/40-bit shifts and 32-bit bit-field operations 32-bit logical operations Branches Constant generation Register transfers to/from control register file (.S2 only) Byte shifts Data packing/unpacking Dual 16-bit compare operations Quad 8-bit compare operations Dual 16-bit shift operations Dual 16-bit saturated arithmetic operations Quad 8-bit saturated arithmetic operations

Table 4.2: The M. and D. Functional Units and Operations Performed [16]

Functional Unit	Fixed-Point Operations
.M unit (.M1, .M2)	<p>16 x 16 multiply operations</p> <p>16 x 32 multiply operations</p> <p>Quad 8 x 8 multiply operations</p> <p>Dual 16 x 16 multiply operations</p> <p>Dual 16 x 16 multiply with add/subtract operations</p> <p>Quad 8 x 8 multiply with add operation</p> <p>Bit expansion</p> <p>Bit interleaving/de-interleaving</p> <p>Variable shift operations</p> <p>Rotation</p> <p>Galois Field Multiply</p>
.D unit (.D1, .D2)	<p>32-bit add, subtract, linear and circular address calculation</p> <p>Loads and stores with 5-bit constant offset</p> <p>Loads and stores with 15-bit constant offset (.D2 only)</p> <p>Load and store double words with 5-bit constant</p> <p>Load and store non-aligned words and double words</p> <p>5-bit constant generation</p> <p>32-bit logical operations</p>

4.2 TI's Code Development Environment [17]

We now introduce the software environment used in our work and how to successfully develop an efficient DSP code as quickly. We will introduce some important and useful techniques to improve the program speed performance.

The Code Composer Studio, TI's GUI code development tool, is the software platform that we use to develop and debug the projects. Some main features of it are listed below:

- Real-time analysis.
- Source code debugger common interface for both simulator and emulator targets.
 - C/C++ assembly language support.
 - Simple breakpoints.
 - Advanced watch window.
 - Symbol browser.
- DSP/BIOS support.
 - Pre-emptive multi-threading.
 - Interthread communication.
 - Interrupt handling.
- Chip Support Libraries (CSL) to simplify device configuration. CSL provides C-program functions to configure and control on-chip peripherals.
- DSP libraries for optimum DSP functionality. The DSP library includes many C-callable, assembly-optimized, general-purpose signal-processing and image/video processing routines. These routines are typically used in computationally intensive real-time



applications where optimal execution speed is critical. The TMS320C64x digital signal processor library (DSPLIB) provides some routines for:

- Adaptive filtering.
- Correlation.
- FFT.
- Filtering and convolution.
- Math.
- Matrix functions.
- Miscellaneous.

Some of these routines are used in our implementation, such as FFT and filtering. We introduce them in a later chapter.

4.2.1 Code Development Flow [18]

The recommended code development flow involves utilizing the C6000 code generation tools to aid in optimization rather than forcing the programmer to code by hand in assembly. Hence the programmer may let the compiler do all the laborious work of instruction selection, parallelizing, pipelining, and register allocation. This simplifies the maintenance of the code, as everything resides in a C framework that is simple to maintain, support, and upgrade. Fig. 4.3 illustrates the three phases in the code development flow. Because phase 3 is usually too detailed and time consuming, most of the time we will not go into phase 3 to write linear assembly code unless the software pipelining efficiency is too bad or the resource allocation is too unbalanced. The following techniques can be used to analyze the performance of specific code regions:

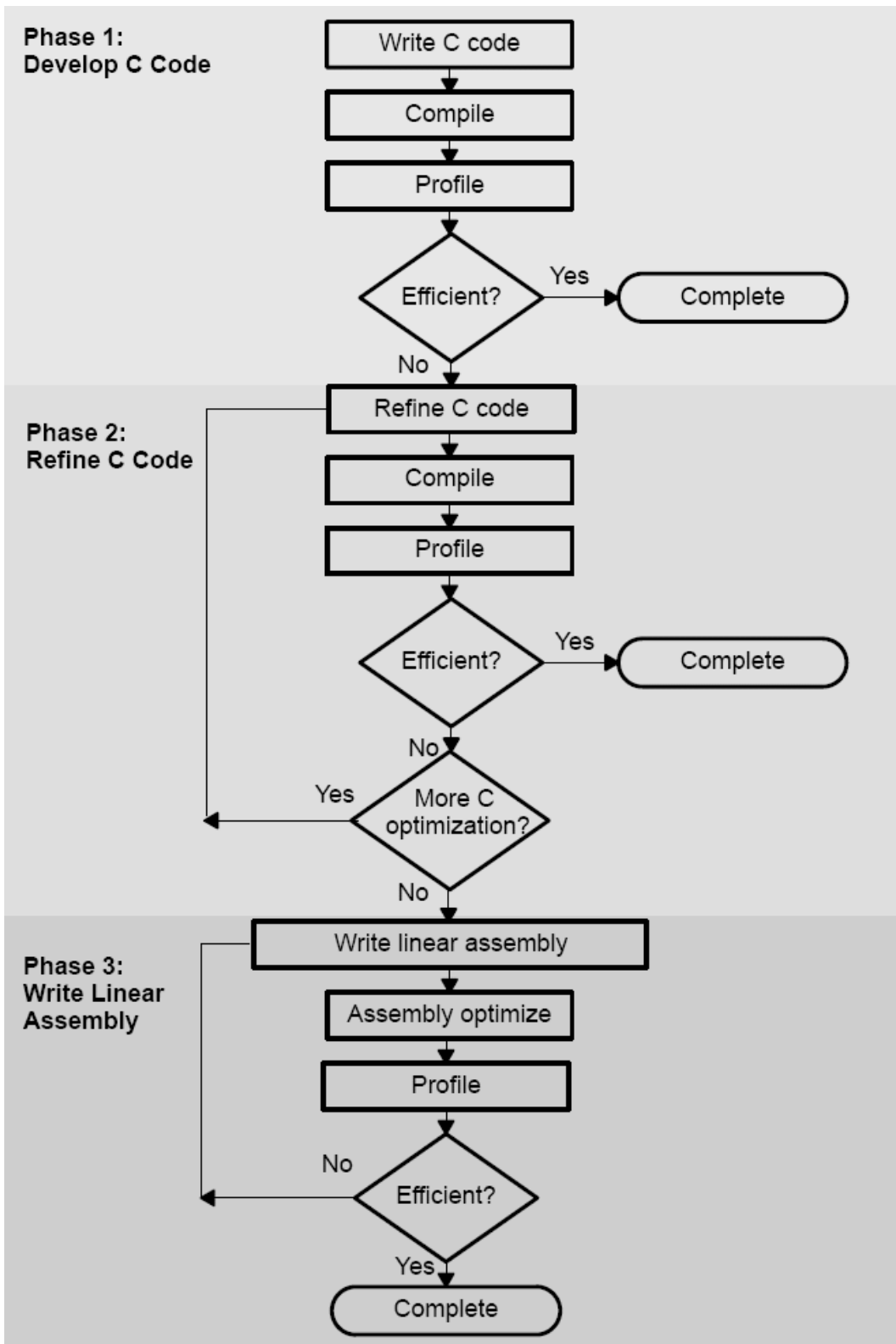


Figure 4.3: Code development flow of C6000 (from [18]).

- Use the `clock()` and `printf()` functions in C/C++ to time and display the performance of specific code regions. Use the stand-alone simulator (`load6x`) to run the code for this purpose.
- Use the profile mode of the stand-alone simulator. This can be done by compiling the code with the `-mg` option and executing `load6x` with the `-g` option. Then enable the clock and use profile points and the `RUN` command in the Code Composer debugger to track the number of CPU clock cycles consumed by a particular section of code. Use “View Statistics” to view the number of cycles consumed.

Usually, we use the second technique above to analyze the C code performance. The feedback of the optimization result can be obtained with the `-mw` option. It shows some important results of the assembly optimizer for each code section. We take these results into consideration in improving the computational speed of certain loops in our program.

4.2.2 Compiler Optimization Options [18]

In this subsection, we introduce the compiler options that control the operation of the compiler. The CCS compiler offers high-level language support by transforming C/C++ code into more efficient assembly language source code. The compiler options can be used to optimize the code size or the executing performance.

The major compiler options we utilize are `-o3`, `-k`, `-pm -op2`, `-mh<n>`, `-mw`, and `-mi`.

- `-on`: The “*n*” denotes the level of optimization (0, 1, 2, and 3), which controls the type and degree of optimization.
 - `-o3`: highest level optimization, main features are:
 - * Performs software pipelining.

- * Performs loop optimizations, and loop unrolling.
 - * Removes all functions that are never called.
 - * Reorders function declarations so that the attributes of called functions are known when the caller is optimized.
 - * Propagates arguments into function bodies when all calls pass the same value in the same argument position.
 - * Identifies file-level variable characteristics.
- -k: Keep the assembly file to analyze the compiler feedback.
 - -pm -op2: In the CCS compiler option, -pm and -op2 are combined into one option.
 - -pm: Gives the compiler global access to the whole program or module and allows it to be more aggressive in ruling out dependencies.
 - -op2: Specifies that the module contains no functions or variables that are called or modified from outside the source code provided to the compiler. This improves variable analysis and allowed assumptions.
 - -mh<n>: Allows speculative execution. The appropriate amount of padding, n , must be available in data memory to insure correct execution. This is normally not a problem but must be adhered to.
 - -mw: Produce additional compiler feedback. This option has no performance or code size impact.
 - -mi: Describes the interrupt threshold to the compiler. If compiler knows that no interrupts will occur in the code, it can avoid enabling and disabling interrupts before and after software-pipelined loops for improvement in code size and performance. In

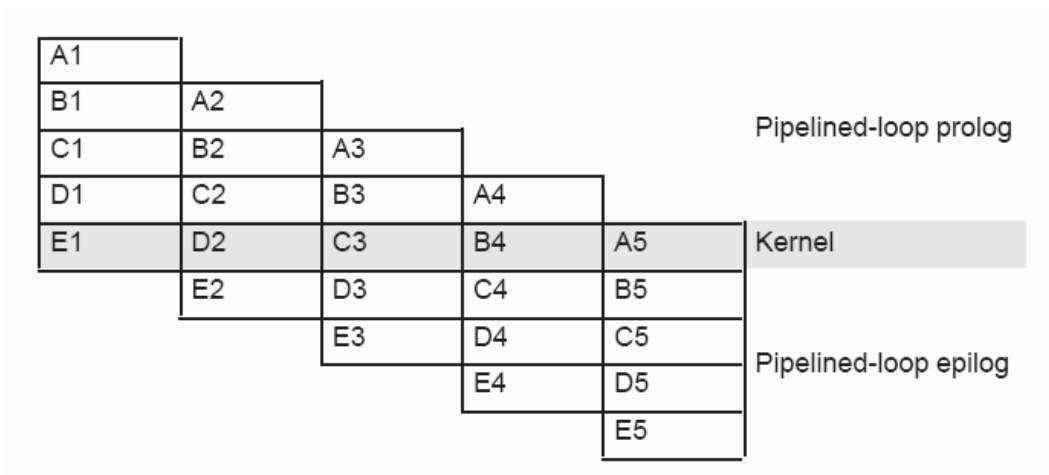


Figure 4.4: Software-pipelined loop (from [16]).

addition, there is potential for performance improvement where interrupt registers may be utilized in high register pressure loops.

4.2.3 Software Pipelining [19]

Software pipelining is a technique used to schedule instructions from a loop so that multiple iterations of the loop execute in parallel. This is the most important technique we use to speed up our system. The compiler always attempts to software-pipeline. Fig. 4.4 illustrates a software pipelined loop. The stages of the loop are represented by A, B, C, D, and E. In this figure, a maximum of five iterations of the loop can execute at one time. The shaded area represents the loop kernel. In the loop kernel, all five stages execute in parallel. The area above the kernel is known as the pipelined loop prolog, and the area below the kernel the pipelined loop epilog.

But under the conditions listed below, the compiler will not do software pipelining [18]:

- If a register value lives too long, the code is not software-pipelined.
- If a loop has complex condition code within the body that requires more than five

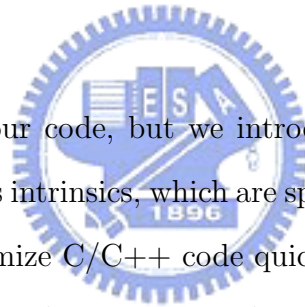
condition registers, the loop is not software pipelined.

- A software-pipelined loop cannot contain function calls, including code that calls the run-time support routines.
- In a sequence of nested loops, the innermost loop is the only one that can be software-pipelined.
- If a loop contains conditional break, it is not software-pipelined.

Usually, we should maximize the number of loops that satisfy the requirements of software pipelining. Software pipelining is a very important technique for optimization; its importance cannot be overemphasized.

4.2.4 Intrinsic [18]

We did not use any intrinsic in our code, but we introduce the concept of this technique here. The C6000 compiler provides intrinsics, which are special functions that map directly to inlined C64x instructions, to optimize C/C++ code quickly. All assembly instructions that are not easily expressed in C/C++ code are supported as intrinsics. A table of TMS320C6000 C/C++ compiler intrinsics can be found in [18].



Chapter 5

OFDM TDD Uplink Synchronization

In this chapter, we consider the uplink synchronization issues under the IEEE 802.16e OFDM. Further, we describe the considered channel environment. The system profiles we use are given in chapter 3.

5.1 OFDM Uplink Synchronization Problem and Techniques



Accurate demodulation and detection of an OFDM signal requires carrier orthogonality. Variations of the carrier oscillator, sample clock or the symbol time affect the orthogonality of the system. Then, before an OFDM receiver can demodulate the carriers, it has to perform three synchronization tasks. First, timing estimation is needed to detect the proper frame start time. Secondly, it has to estimate and correct the carrier frequency offset (CFO) of the received signal. Third, sampling frequency offset (SFO), or symbol clock offset) should be detected. Note that, in normal uplink transmission, frame synchronization is not needed because the base station knows roughly when the signal from each SS should arrive. Figure 5.1 shows the proposed synchronizer structure for the receiver.

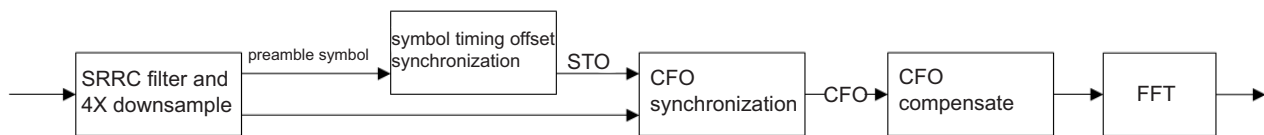


Figure 5.1: The proposed synchronizer structure for the receiver.

5.1.1 Timing Offset and Fractional Carrier Frequency Offset

A popular algorithm to estimate timing offset and (fractional) CFO is proposed in [20]. By taking advantage of the cyclic prefix, the proposed technique can accurately estimate the symbol timing instant and frequency offset relatively accurately in additive white Gaussian noise (AWGN), blindly with no assistance from pilot symbols. However, it suffers considerable performance degradation in multipath propagation or Rayleigh fading [5]. A modified technique proposed in [21] is shown to have better performance in fast Rayleigh fading. In addition, it can obtain a symbol timing estimate in parallel to the frequency offset estimate.

Figure 5.2 illustrates the algorithm structure proposed in [21]. Under the assumption that received samples are jointly Gaussian, symbol time offset $\hat{\theta}$ and fractional CFO $\hat{\varepsilon}$ is given by

$$\hat{\theta} = \arg \max \{c_3 |\lambda_1(\theta)|^2\}, \quad (5.1)$$

$$\hat{\varepsilon} = -\frac{1}{2\pi} \tan^{-1} \left(\frac{\text{Im}\{\lambda_1(\hat{\theta})\}}{\text{Re}\{\lambda_1(\hat{\theta})\}} \right), \quad (5.2)$$

respectively, where

$$\lambda_1(\theta) = \sum_{k=\theta}^{\theta+L-1} r(k)r^*(k+N)$$

and c_3 is set to a constant $1/L$.

To get a more accurate CFO estimation, we can average the estimated values over multiple OFDM symbols.

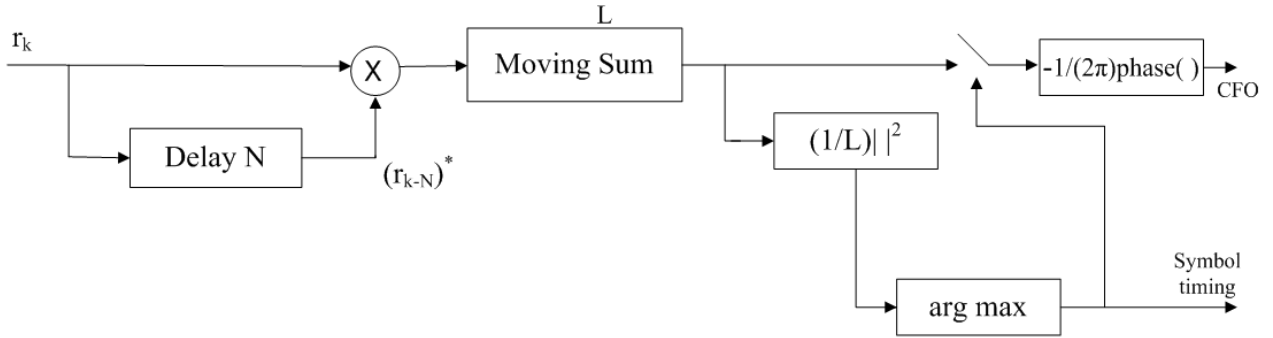


Figure 5.2: Structure of J.-C. Lin's symbol timing and fractional carrier frequency synchronization method [21].

Our earlier study has considered the above approach. The uplink signal structure defined in the IEEE 802.16e standard, however, motivates another approach which could yield better performance in multipath fading. According to the standard, each UL burst contains a preamble, which consists 2 times 128 samples in the time domain. Although the contents of preamble is known, it seems good (we will see their performance later) to perform blind symbol timing detection based on the 128-sample periodical structure as

$$\hat{\theta} = \arg \max_i \sum_{k=i}^{i+128} |r(k+L)r(k+L+128)^* - r(k+L+128)r(k+L+256)^*| \quad (5.3)$$

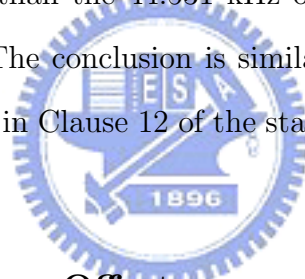
where $i = 0, \dots, 8$ because all SSs should adjust their timing such that all OFDM symbols arrive time coincident at the BS to an accuracy of ± 4 samples. Note that in the second term we subtract out the effect of CP, which interferes with the estimation of correct timing point.

The reason why we do not use the whole known preamble to correlate with the received preamble is because the received symbol is corrupted by channel response, so the correlation result of the corrupted preamble and the original preamble is not very well. On the other hand, if the mobile speed is not very high, the channel response is almost the same during one symbol duration, so the two halves of the preamble, although still are corrupted, are

very similar and the performance of their correlation is better than using the whole known preamble.

5.1.2 Integer Carrier Frequency Offset

The technique discussed above can only estimate fractional CFO. Hence, theoretically, we still need to estimate any possible integer CFO between the transmitter and the receiver. But practically, this is unnecessary in the UL OFDM transmission because of the wide subscriber spacing. According to the specifications of the standard, an SS should synchronize their frequency to the BS to within a maximum tolerance of 0.02 times the subscriber spacing. Now consider a mobile speed as high as 240 km/h. Then considering a 10 MHz signal bandwidth at a carrier frequency of 5 GHz, the maximum Doppler shift is on the order of 1 kHz, which is much smaller than the 44.531 kHz of subcarrier spacing even with the 2% maximum frequency error. (The conclusion is similar for the profile with the smallest bandwidth, namely, *profP3-1.75*, in Clause 12 of the standard.) Therefore, there is no need to estimate the integer CFO.



5.1.3 Sampling Frequency Offset

From [22], we know the frequency-domain symbol with phase rotation caused by SFO can be modeled as

$$z_{l,k} = (e^{j2\pi((lN_s+N_g)/N)\zeta k})\alpha(\phi_k)a_{l,k}H_k + n_{\Omega;l,k} + n_{l,k}. \quad (5.4)$$

where l is symbol number, $N_s = N + N_g$, k is subcarrier index, $\zeta = (T' - T)/T$ with T' being the sampling frequency of receiver and T the sampling frequency of transmitter, $\alpha(\phi_k) = \text{sinc}(\pi\phi_k)$ is very close to 1, $a_{l,k}$ is transmitted data symbol, H_k is channel impulse response of subcarrier k , which is assumed to stay constant over two consecutive symbols, $n_{\Omega;l,k}$ is interchannel interference (ICI), and $n_{l,k}$ is AWGN. In the IEEE 802.16e OFDM,

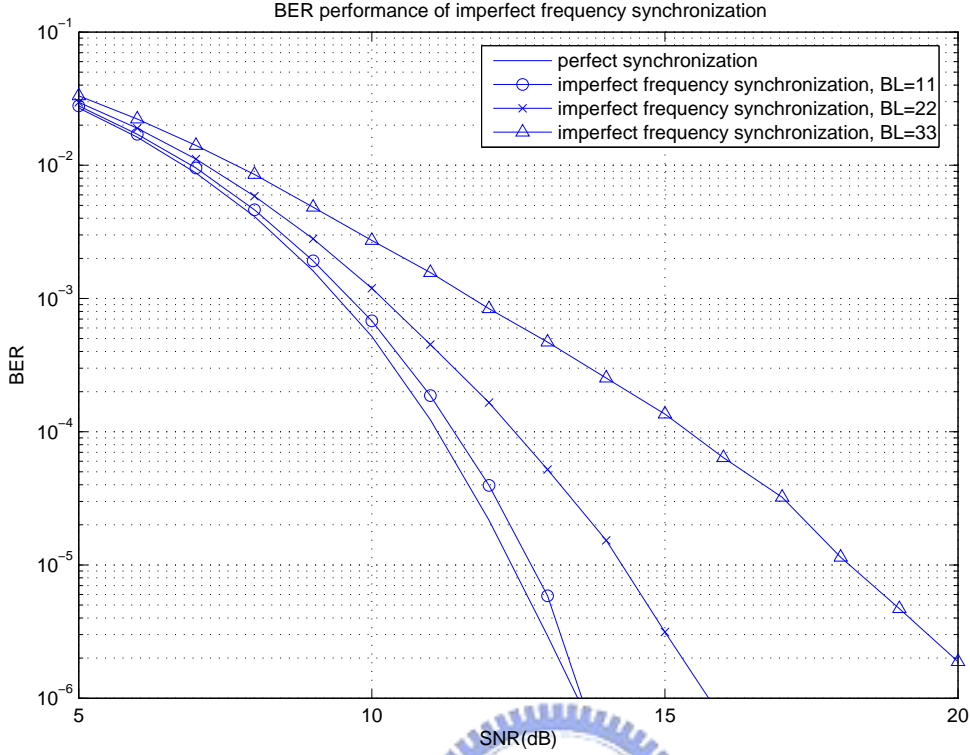


Figure 5.3: BER degradation at 5 ppm sampling clock error.

$N = 256$, and we let $N_g = 32$; hence $N_s = 288$. Therefore, in two consecutive symbols, the phase increment is given by

$$\Delta\varphi_k = 2\pi(N_s/N)\zeta k. \quad (5.5)$$

Note that the IEEE 802.16e standard has specified the maximum tolerance for the sampling clock frequency at the SS as 5 ppm. This rule can simplify the synchronization work at BS, because the performance degradation caused by SFO is not serious if the transmission burst length is not too long. Figure 5.3 shows our simulation results. The modulation is 16-QAM, in AWGN channel. A burst with length 11 OFDM symbols transmits about 1kb data (200 used carriers \times 4 bits/sample \times 11 = 8800 bits) each time. We can see the BER degradation is very small when burst length is small.

Since the effects of SFO can be ignored, there is no need to do SFO synchronization.

5.2 Channel Model

Typical models of the wireless communication channel include additive noise and multipath fading. For channel simulation, noise and multipath fading are described as random processes, so they can be algorithmically generated as well as mathematically analyzed.

5.2.1 Gaussian Noise

The simplest kind of channel is the AWGN channel, where the received signal is only subject to added noise. A major source of this noise is the thermal noise in the amplifiers which may be modeled as Gaussian with zero mean and constant variance. In computer simulations, random number generators may be used to generate Gaussian noise of given power to obtain a particular signal-to-noise ratio (SNR).

5.2.2 Slow Fading Channel

In slow fading, multipath propagation may exist, but the channel coefficients do not change significantly over a relatively long transmission period. The channel impulse response over a short time period can be modeled as

$$h(\tau) = \sum_{i=0}^{N-1} \alpha_i e^{j\theta_i} \delta(\tau - \tau_i). \quad (5.6)$$

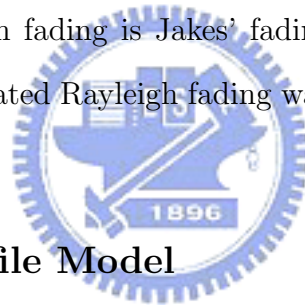
where N is the number of multipaths, α_i and τ_i are respectively the amplitude and the delay of the i th multipath, and θ_i represents the phase shift associated with path i . These parameters are time-invariant in a short enough time period.

5.2.3 Fast Fading Channel

With sufficiently fast motion of either the transmitter or the receiver, the coefficient of each propagation path becomes time varying. The equivalent baseband channel impulse response can then be better modeled as

$$h(\tau, t) = \sum_{i=0}^{N-1} \alpha_i(t) e^{j\theta_i(t)} \delta(\tau - \tau_i) \quad (5.7)$$

Note that α_i and θ_i are now functions of time. But τ_i is still time-invariant, because the path delays usually change at a much slower pace than the path coefficients. The channel coefficients are often modeled as complex independent stochastic processes. If there is no line-of-sight (LOS) path between the transmitter and the receiver, each path may be made of the superposition of many reflected paths, yielding a Rayleigh fading characteristic. A commonly used method to simulate Rayleigh fading is Jakes' fading model, which is a deterministic method for simulating time-correlated Rayleigh fading waveforms. A recent improvement to Jakes' model is proposed in [13].



5.2.4 Power-Delay Profile Model

For simplicity in analysis and simulation, the delay τ_i in the above two models can be discretized to have a certain easily manageable granularity. This results in a tapped-delay-line model for the channel impulse response, where the spacing between any two taps is an integer multiple of the chosen granularity. For convenience, one may excise the initial delay and make $\tau_0 = 0$. Often, it is convenient to normalize the path powers relative to the strongest path. And, often, the first path has the highest average power.

We consider the ETSI “Vehicular A” model considered in [14]. The model is as shown in Table 5.1. This is a channel model for the vehicular test environment, which the tested speed is from 120 km/h to 500 km/h. This environment is characterized by larger cells and higher

Table 5.1: ETSI “Vehicular A” Channel Model in Different Units [13]

tap	relative delay (nsec or sample number)			average power		
	(nsec)	(4× oversampling)	(normal)	(dB)	(normal scale)	(normalized)
1	0	0	0	0	1.0000	0.4850
2	310	14	3 or 4	-1.0	0.7943	0.3852
3	710	32	8	-9.0	0.1259	0.0610
4	1090	50	12 or 13	-10.0	0.1000	0.0485
5	1730	79	20	-15.0	0.0316	0.0153
6	2510	115	29	-20.0	0.0100	0.0049

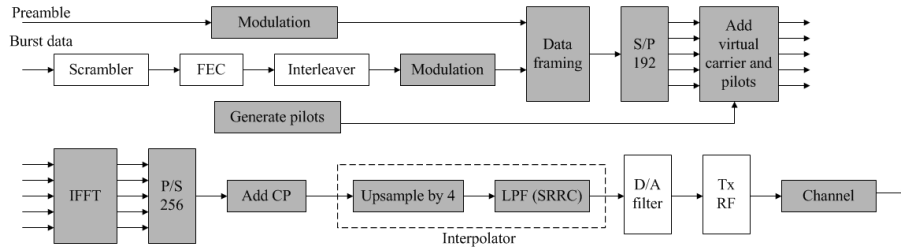


Figure 5.4: UL transmitter structure.

transmit power, and is valid for NLOS case only and describes worse case propagation. Channel A is the low delay spread case that occurs frequently. Please see [14] for more details.

5.3 Floating-Point Simulation Results

Before modifying the algorithms we discussed above to fixed-point version, consider the performance of floating-point version first for comparison with the performance of fixed-point version later. Figures 5.4 and 5.5 are complete structures of OFDM transmitter and receiver systems. The blocks that have gray color are the functions we implement. The C program structure is shown in Figure 5.6.

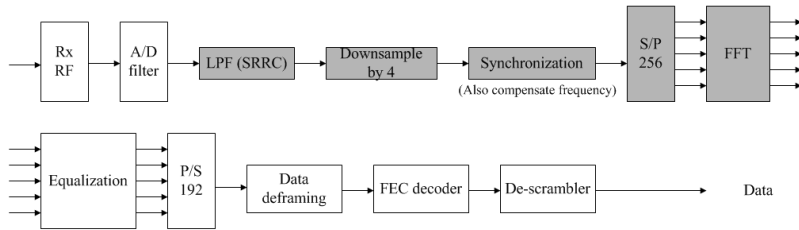


Figure 5.5: UL receiver structure.

Table 5.2: Receiver SNR Assumptions

Modulation	Coding rate	Receiver SNR(dB)
BPSK	1/2	3.0
QPSK	1/2	6.0
QPSK	3/4	8.5
16-QAM	1/2	11.5
16-QAM	3/4	15.0
64-QAM	2/3	19.0
64-QAM	3/4	21.0

5.3.1 Simulation Parameters and Environments

AWGN channel and multipath Rayleigh fading have been described previously. We use fast fading channel for more practical simulation. Note that the receiver SNR specified in the IEEE 802.16e OFDM is from 3 dB to 21 dB (see Table 5.2), and at least 11.5 dB for 16-QAM modulation, so our simulated range is chosen to be 5 dB to 24 dB in AWGN channel, and from 5 dB to 20 dB, or fixed at 10 dB in fading channel.

Since the OFDM system is designed for low mobility environments, the tested speed we choose is from 0 km/h to 60 km/h; their corresponding Doppler shifts are shown in Table 5.3. We can see even speed is as high as 60 km/h, the maximum Doppler shift is still much smaller than $0.02\Delta f = 890.625$ Hz, so we assume in the simulations that CFO is no larger than $0.1\Delta f$, which is more than enough. The 802.16e specifies that the uplink timing offset should be within ± 4 samples. Thus we let the offset be random in this range. We also

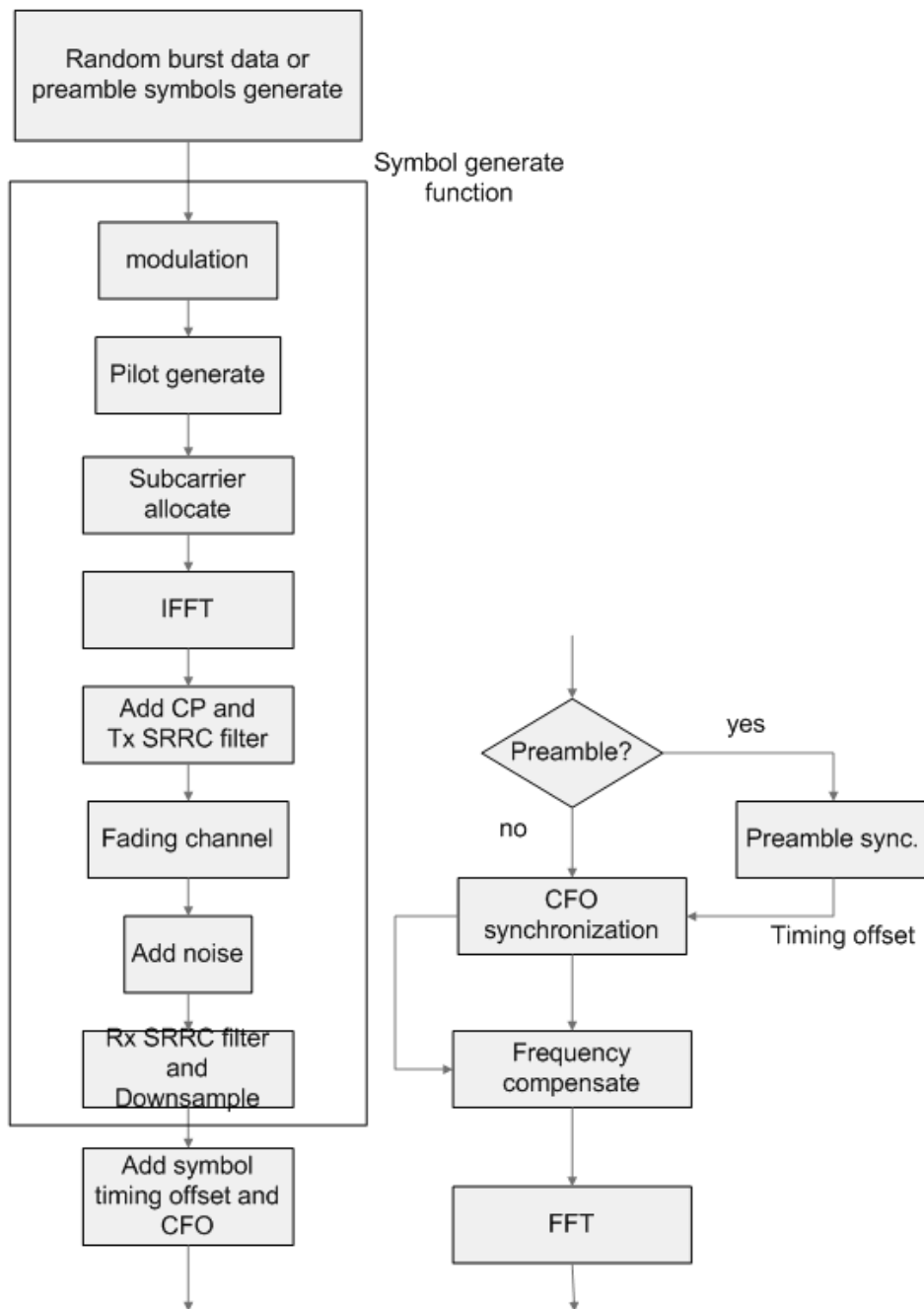


Figure 5.6: Structure of the C program for synchronizer simulation.

assume that there is no sampling clock error in our simulation.

Table 5.3: Relation Between Speed and Doppler Shift at Carrier Frequency 5 GHz

Speed (km/h)	Doppler Shift (Hz)	$f_d T_s$
0	0	0
10	46.296	0.001157
20	92.593	0.002315
30	138.889	0.003472
40	185.185	0.004630
50	231.482	0.005787
60	277.778	0.006944

5.3.2 Symbol Timing Estimation

Figure 5.7 is timing error distribution in AWGN (upper two charts) and 6-path fast fading channel (lower two charts) at SNR 5 dB and 16 dB, respectively, using (5.3). The mobile speed is 60 km/h. For comparison, Figure 5.8 shows the simulation results by using (5.1), which is a guard interval correlation.

We can see that when in AWGN channel, using method in [21] has some what better performance than ours, but when in multipath fading channel, our method has much higher correct rate. This is because our algorithm can delete the power delay-spread caused by multipath propagation.

Figure 5.9 shows the root mean-square error (RMSE) of our method at SNR = 10 dB, the RMSE is defined as $\sqrt{E \left\{ \left| \theta - \hat{\theta} \right|^2 \right\}}$. The RMSE is a measurement of how spread out a distribution is.

In Figure 5.10, we can see how different speeds affect the error distribution of symbol timing estimation. Almost 99.5% of errors are under 2 samples. The probabilities of error larger than 0 sample are more than that of error smaller than 0 sample, this is caused by the delay profile. Note that we run 10000 symbols for simulation, so the error rate 1×10^{-5} means no error. The SNR is 10 dB.

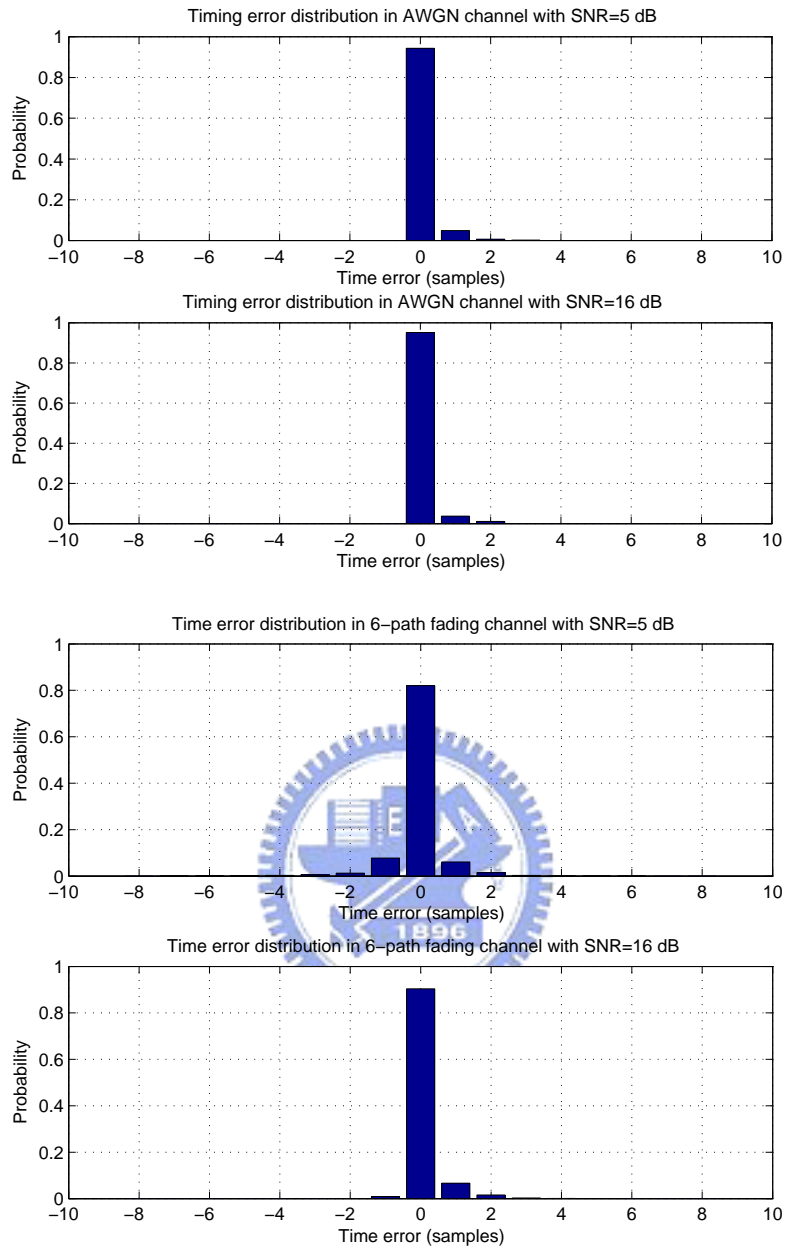


Figure 5.7: Distribution of timing offset estimation errors.

5.3.3 Carrier Frequency Synchronization

The SNR here, if not mentioned, is 10 dB. Figure 5.11 shows the RMSE of fractional CFO estimation under $\epsilon = 0.1\Delta f$, where the RMSE is defined as $\sqrt{E\{|\epsilon - \hat{\epsilon}|^2\}}$. We can see that

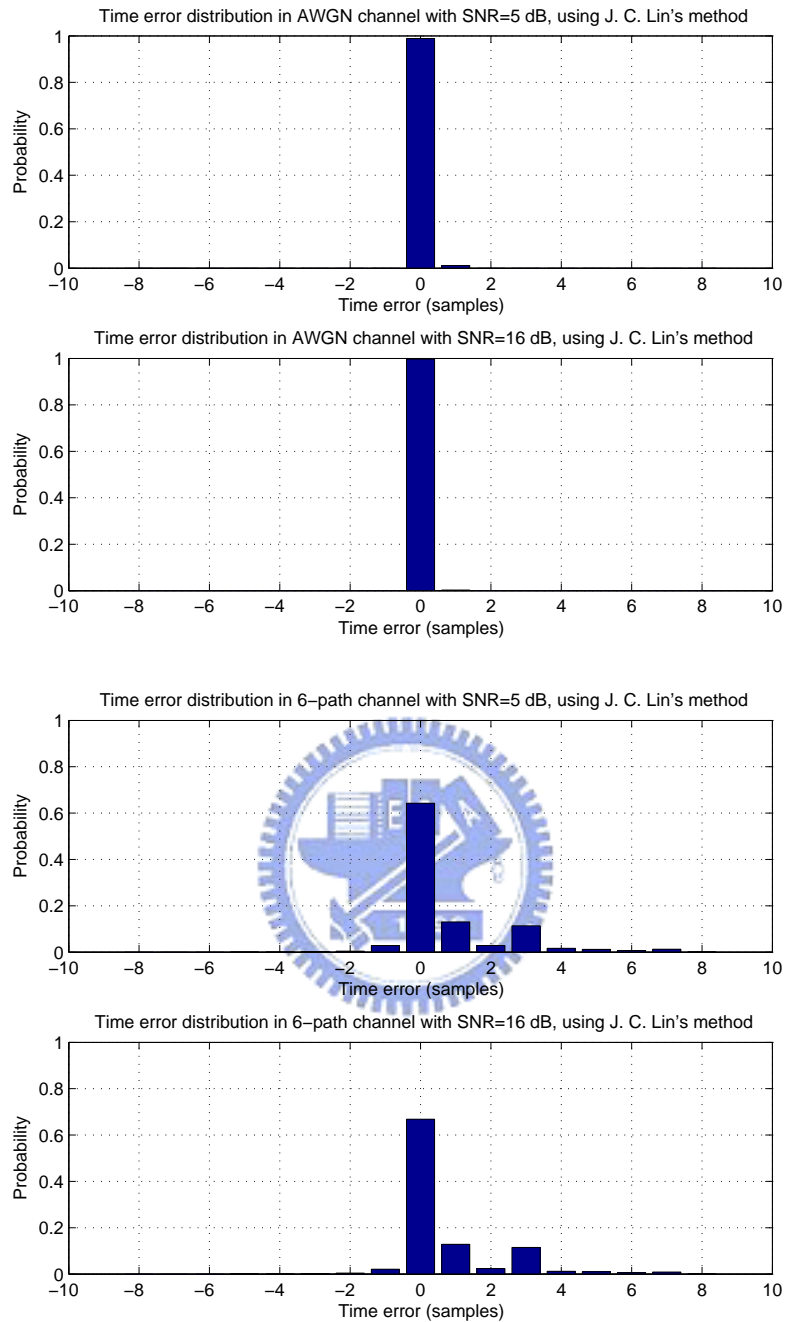


Figure 5.8: Distribution of timing offset estimation errors using J.-C. Lin's method.

the RMSE grows as the the speed increases. Figure 5.12 shows how the motion speed affects the error distribution of carrier frequency synchronization. After estimation, in 99.7% of the

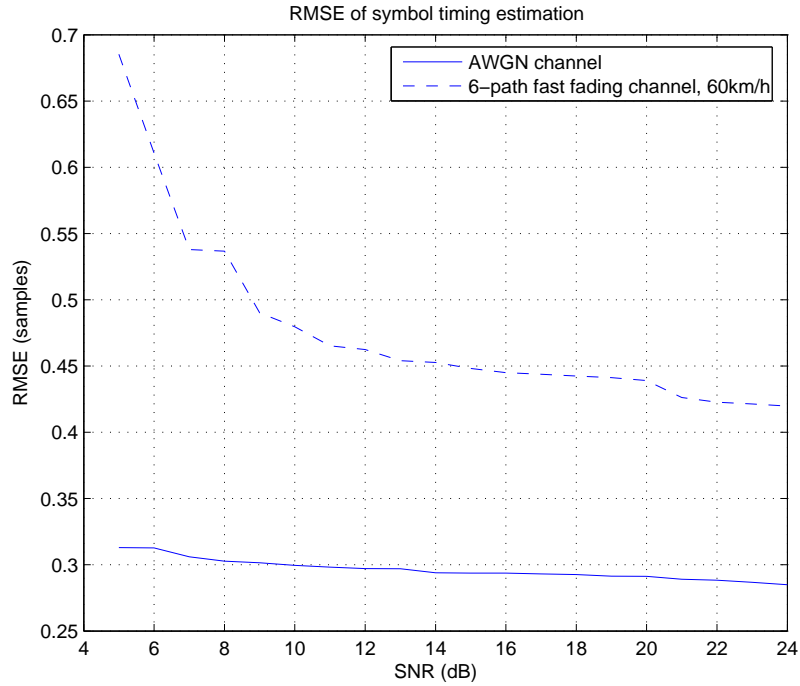


Figure 5.9: RMSE of symbol timing offset synchronization at SNR = 10 dB.

cases the corrected frequency offset is under 2% of subcarrier spacing, as required by IEEE 802.16e, and in more than 90% of the cases is under 1% of subcarrier spacing.

For enhanced accuracy, we average consecutive estimated fractional CFOs of received OFDM symbols in the same burst. Here we set the length of transmission burst to 11 OFDM symbols. The average is calculated as

$$\hat{\epsilon}_{k+1} = \frac{\hat{\epsilon}_k \cdot k + \epsilon_k}{k + 1}. \quad (5.8)$$

where $\hat{\epsilon}_{k+1}$ is the $k + 1$ st averaged CFO, ϵ_k is the k th estimated CFO, and k is the symbol index in a burst. In our case k goes from 0 to 10. Figures 5.13 and 5.14 show the simulation results for $\hat{\epsilon}$. Both are obviously better than the results shown in Figures 5.11 and 5.12. Again, the error rate 10^{-5} means no error occurred in our testing.

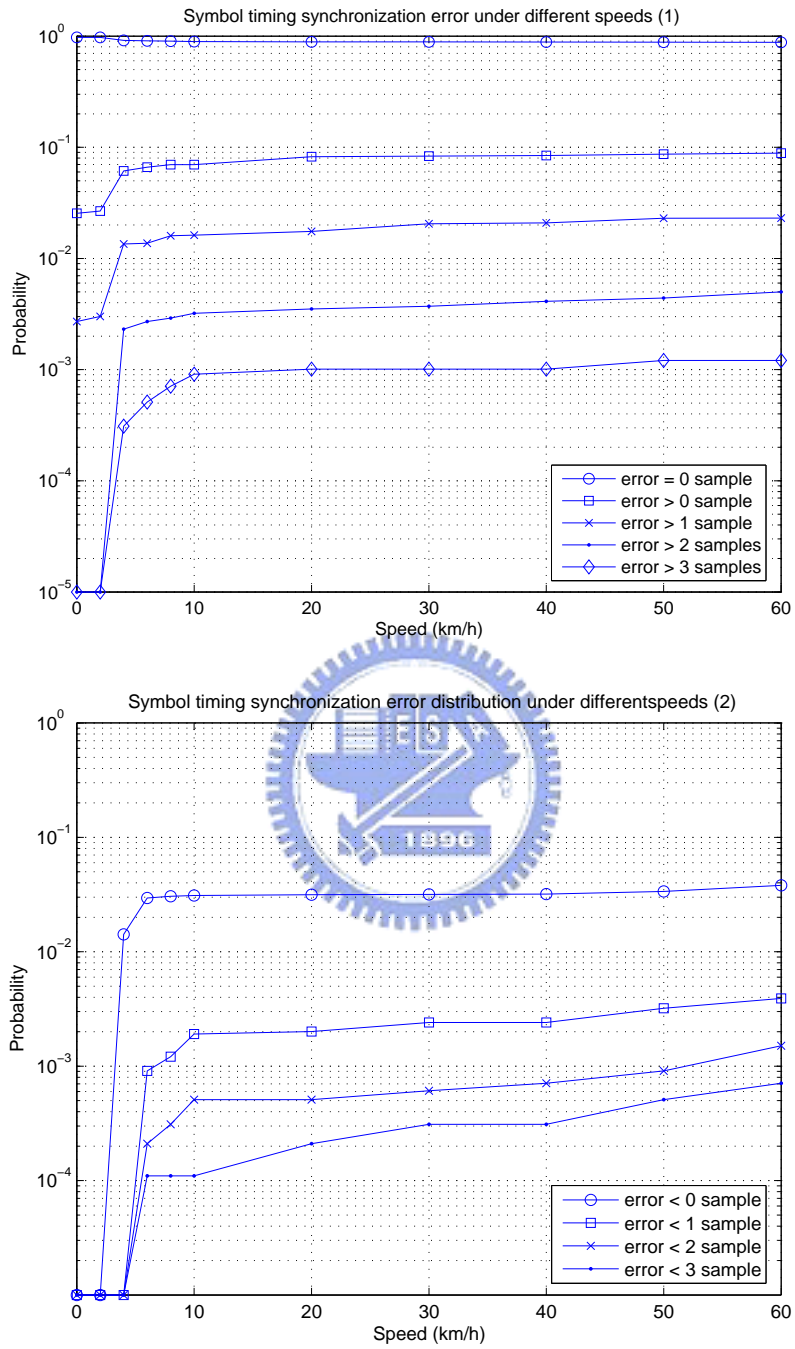


Figure 5.10: Symbol time synchronization error distribution under different speeds.

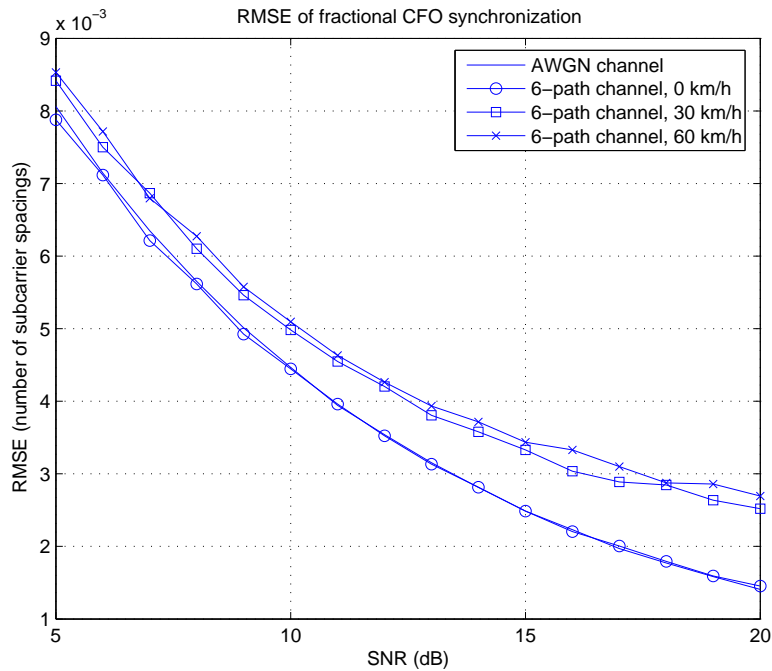


Figure 5.11: RMSE of fractional CFO synchronization.

5.4 Fixed-Point Implementation

In algorithm development, it is often convenient to employ floating-point computation. But for power, speed, and hardware cost reasons, practical transceiver implementations normally use fixed-point computation. The DSP employed in this work, TI's TMS320C6416, is also of the fixed-point category, which can perform fixed-point computations more efficiently than floating-point ones. We consider fixed-point DSP implementation in this work, which entails careful conversion of the original program used in algorithm development from floating-point to fixed-point. We also try to making the resulting program run fast by making efficient use of the DSP's features.

The C6416 CPU contains 8 parallel 32-bit function units, two of which are multipliers and the remaining six can do a number of arithmetic, logic, and memory access operations.

There is also flexibility in arranging the data so that each function unit can do double 16-bit or quadruple 8-bit operations. Running at 600 MHz, the peak performance is 4800 MIPS. For many transmission systems, 32-bit computations are an overkill and 8-bit computations do not provide the necessary accuracy. This appears to be the case with the present system, too. Hence we choose to use the 16-bit data type mostly, with careful selection of dynamic range of the data at different points in the function chain. Simulation results confirm that this is an appropriate choice. In fact, a TI document also suggests use of the short data type (16-bit) for fixed-point multiplication inputs whenever possible [18]. The chosen data formats are as shown in Figures 5.15 and 5.16 for the transmitter and the receiver, respectively, where $Qx.y$ means there are x bits before the binary points and y bits after. In every case, $x + y = 15$ because the sign takes one bit. We discuss the details of each block in the following subsections.

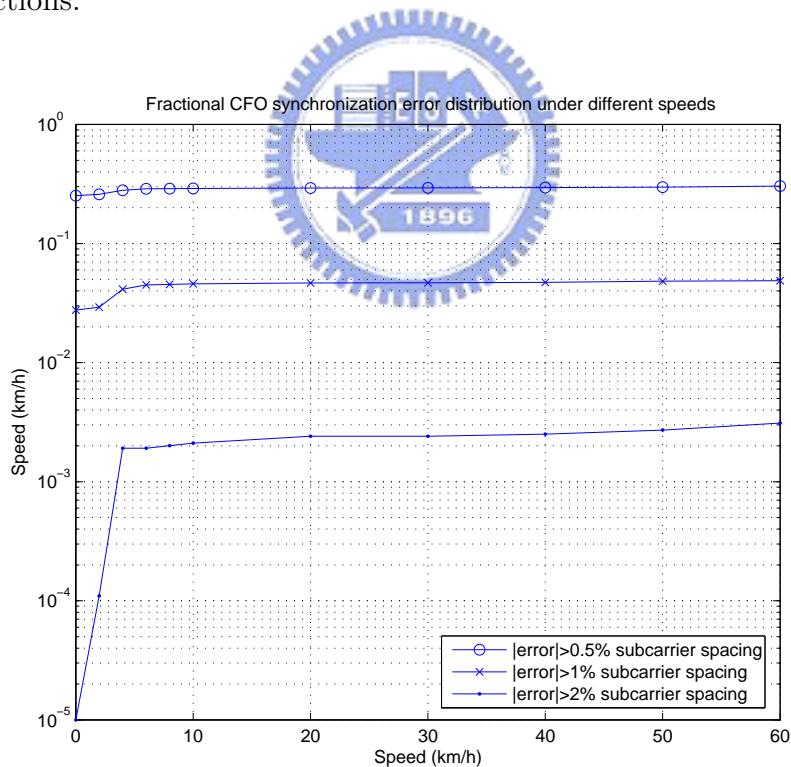


Figure 5.12: Fractional CFO synchronization error distribution under different speeds.

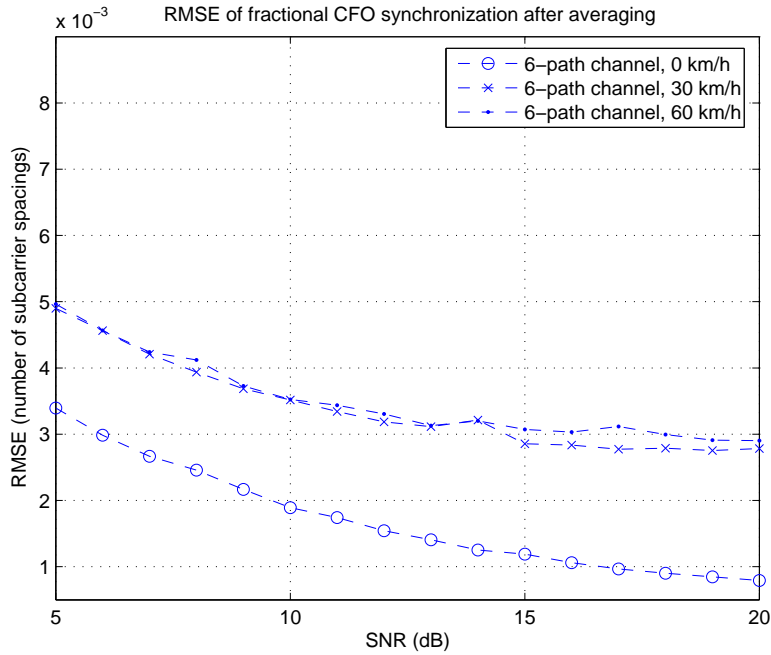


Figure 5.13: RMSE of fractional CFO synchronization after averaging.

5.4.1 Modulation and Subcarrier Allocation

The types of modulation supported in the IEEE 802.16e standard are BPSK, QPSK, 16-QAM and, optionally, 64-QAM. The output signals of the modulators have normalized symbol energy, with the range of signal values of each modulation type as shown in Table 5.4. The widest range occurs in the case of 64-QAM, which is $[-\frac{7}{\sqrt{42}}, \frac{7}{\sqrt{42}}]$. Therefore we must have at least one bit for the integer part of the signal value. With one bit for sign, there remains 14 fractional bits. Hence Q1.14 is the chosen data format, whose range covers $[-2, 2)$. It can cover the ranges of pilot and preamble modulations as well.

The subcarrier allocation block simply allocates the modulation data samples, null samples and pilot samples to their assigned subcarriers. There is no need to change data format in this block.

Table 5.4: Ranges of Modulated Signal Values

Modulation	Range
BPSK	$[-1, 1]$
QPSK	$[-1/\sqrt{2}, 1/\sqrt{2}]$
16-QAM	$[-3/\sqrt{10}, 3/\sqrt{10}]$
64-QAM	$[-7/\sqrt{42}, 7/\sqrt{42}]$

5.4.2 The IFFT and FFT

Since the signals after the IFFT are in the range $[-1, 1]$, we choose Q.15 as the data format after IFFT and before FFT. For efficiency reason, we employ some functions provided by TI in the DSPLIB for C64x to implement the IFFT and the FFT.

The DSPLIB contains FFT functions employing 32×32 -bit and 16×16 -bit multiplica-

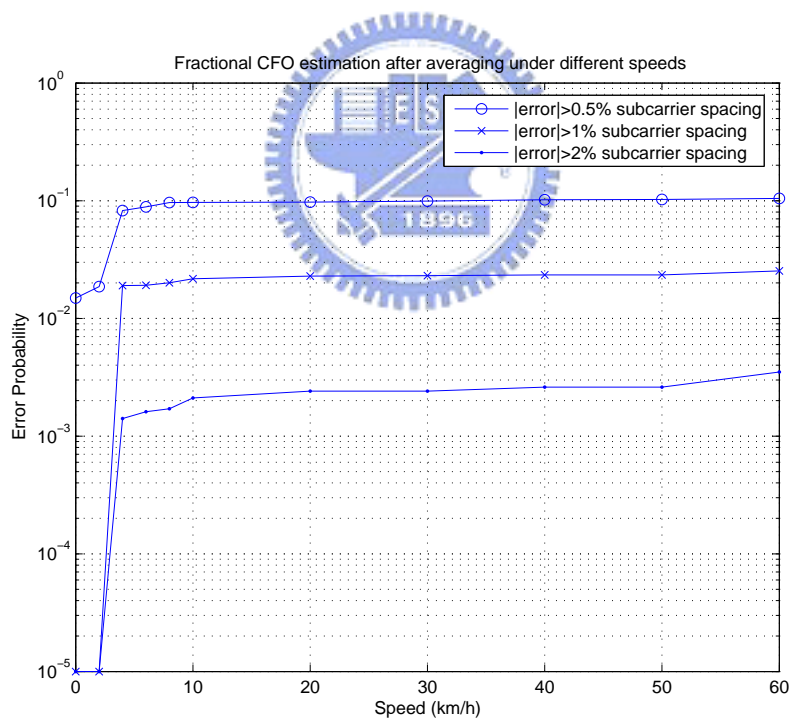


Figure 5.14: Fractional CFO synchronization error distribution under different speeds after averaging.

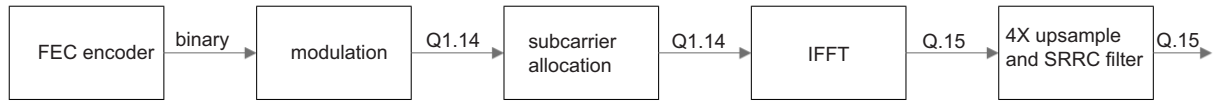


Figure 5.15: Fixed-point data formats used at different points in the transmitter.

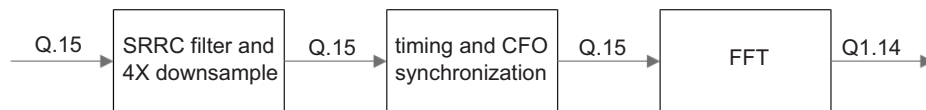


Figure 5.16: Fixed-point data formats used at different points in the receiver.

tions. The former has higher computational complexity. We resolve to use the latter.

The function `DSP_fft16x16r()` computes a complex forward mixed radix FFT with scaling, rounding and digit reversal. The input data $x[]$ and the coefficients $w[]$ are arrays of complex numbers, with the numbers stored in interleaved 16-bit real and imaginary parts. The output data are returned in a separate array $y[]$ in normal order, also complex with interleaved 16-bit real and imaginary parts. The code uses a special ordering of FFT coefficients (also called twiddle factors). These twiddle factors are generated by using the function `tw_fft16x16()` provided by TI.

The DSPLIB does not contain a 16×16 -bit IFFT routine. Hence we modify the `DSP_fft16x16r()` routine to compute the IFFT. The modification is based on the following identity:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} y[k] W_N^{-kn} = \frac{1}{N} \sum_{k=0}^{N-1} y[k] (W_N^{kn})^* = \frac{1}{N} \left(\sum_{k=0}^{N-1} y[k]^* W_N^{kn} \right)^*, \quad n = 0, \dots, N-1, \quad (5.9)$$

where $y[]$ is the input, $x[]$ is the output, and W_N is the twiddle factor. Therefore, we first

conjugate the input, then perform FFT, and then conjugate the output to obtain the desired IFFT.

In `DSP_fft16x16r()`, scaling by 2 (i.e., right shift by 1 bit) takes place at each radix-4 stage except the last one. A radix-4 stage could give a maximum bit-growth of 2 bits, which would require scaling by 4. To prevent overflows, the input data in general should be scaled by 2^{BT-BS} , where $BT = \log_2 N$ (total number of bit growth) and $BS = \lceil \log_4 N - 1 \rceil$ (2's exponent of scaling), with N being the length of the FFT. All shifts are rounded to reduce the truncation noise power by 3 dB.

Recall that the length of IFFT/FFT in our system is 256. Hence $BT = \log_2 256 = 8$ and $BS = \lceil \log_4 256 - 1 \rceil = 3$ and theoretically we need to shift the input to the right by 5 bits. But we find that, in our case, scaling the IFFT input by 4 bits is enough to prevent output overflow, and this can reduce the noise from scaling by 3 dB. Thus, in principle, the IFFT output is scaled for a total of $BS + 4 = 7$ bits. But as far as fixed-point binary numbers are concerned, such scaling amounts merely to relocating the binary point, which can be relocated anywhere (equivalent to applying an arbitrary integer-power-of-2 scaling) for the convenience of fixed-point computation. Thus we interpret the IFFT output as in Q.15 format. For the FFT, we find that right-shift of the input by 1 bit is enough to prevent output overflow.

5.4.3 SRRC Filter with Oversampling and Downsampling

In order to provide the ability to simulate path delays at non-integer sample times, an interpolator is induced in the transmitter to yield 4-times oversampled transmitter output. In our system, we adopt the 57-taps square-root raised-cosine (SRRC) filter with $\alpha = 0.155$. We implement a polyphase system, shown in Figure 5.17. This implementation would involve applying filter coefficients only to input values that are nonzero. In our work, $L = 4$. When

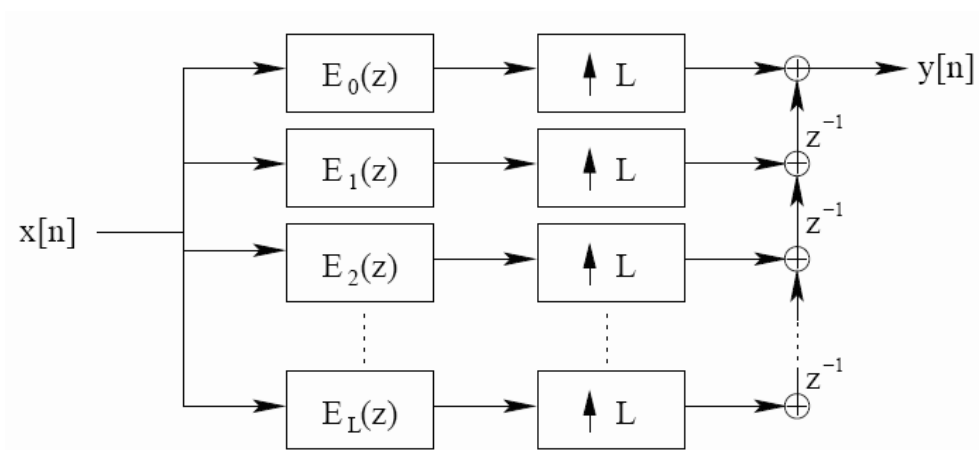


Figure 5.17: Implementation of interpolation filter with polyphase decomposition [5].

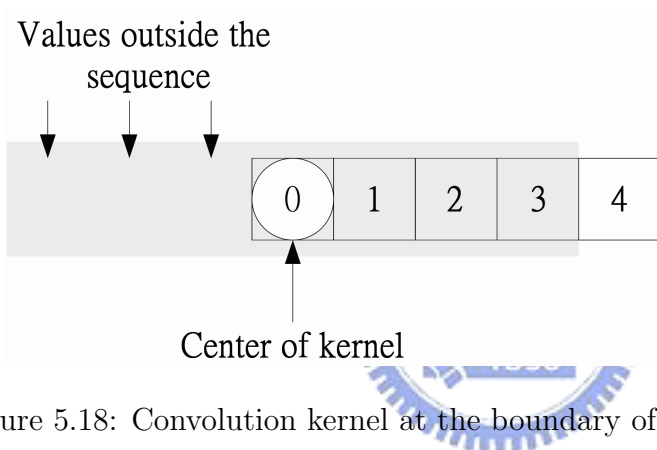


Figure 5.18: Convolution kernel at the boundary of a finite-length sequence [7].

computing an output value at the boundary of a sequence, a portion of the convolution or correlation kernel is usually off the edge of sequence, as illustrated in Figure 5.18. We assume the values outside the data sequence to be 0, that is, we do zero padding. Thus, we can avoid using many if-else statements to handle the boundary values when doing convolution.

The output of Figure 5.17 is equivalent to oversampling input by 4 times and passing it through the SRRC filter. In the receiver, we just convolve the input signals with the SRRC filter, which is like the convolution in the transmitter, and downsample the output by 4

times. The data formats of the input and the output are the same.

5.4.4 Synchronization

The detailed synchronization method has been presented in previous sections. Besides translating floating data type to short data type, here we only make two points relating to fixed-point implementation:

- In fractional CFO estimation, we use a lookup table to implement the $\arctan()$ function. The table contains 2048 entries covering the range $[\tan 0, \tan 0.4\pi]$ uniformly, for $\hat{\epsilon}$ in $[0, 0.2]$ times the subcarrier spacing. The table also applies to negative values of $\hat{\epsilon}$ since the tangent function is symmetric.
- In frequency offset compensation, we construct two tables for the $\sin()$ and the $\cos()$ functions, each containing 2048 entries covering the range $[0, \pi]$ uniformly.

5.5 Fixed-Point Simulation Results

We present some simulation results in this subsection. All simulations are just like we ran in Section 5.3, but in fixed-point implementation. This is for convenience comparing their performance.

5.5.1 Symbol Timing Estimation

Since the preamble correlation performs better than CP correlation, we only show the results of the former method. Figure 5.19 shows the performance in AWGN channel and multipath fast Rayleigh fading channel, in both 5 dB and 16 dB of SNR. Comparing with the performance of floating-point simulation, the results in AWGN channel perform better.

This is because in the fixed-point receiver, the received signal has been truncated from 32-bit to 16-bit data type, which can increase the SNR of received signal indirectly. But in multipath fading channel, fixed-point implementation reduces the performance somewhat. Observe that when the SNR is high, the probability of synchronizing to the correct timing in a multipath fading channel can be on the order of 0.9 or better.

Figure 5.20 shows the RMSE in AWGN channel and multipath Rayleigh fading channel at speed 60 km/hr. We put the results of floating-point and fixed-point simulations together for ease of comparison. In Figure 5.21 we can see how speeds affect the synchronization of symbol timing, and the SNR is 10 dB. The floating-point version has better performance especially when the error is smaller than 0 sample. All RMSE results have a little worse performance for the fixed-point implementation, but in a range we can accept.

5.5.2 Carrier Frequency Synchronization

Figure 5.22 shows the RMSE of fractional CFO estimation of both fixed-point and floating-point simulations under $\epsilon = 0.1\Delta f$. We see that the performance at 30 km/h and 60 km/h is worse than the other conditions, but is still under 2% of subcarrier spacing. In high SNR, the RMSE are about 0.012 times the subcarrier spacing. Figure 5.23 shows how different speeds affect the error distribution of carrier frequency synchronization, we can see that when the speed is not 0, the performance decreases obviously comparing with the floating-point version.

Since the performance degrades much when in mobile environment, averaging estimated fractional CFOs is a good solution here. Figures 5.24 and 5.25 show the simulation results. Although they are still not as good as the floating-point version, the performance is improved. The RMSEs can be as small as about 0.002 times the subcarrier spacing.

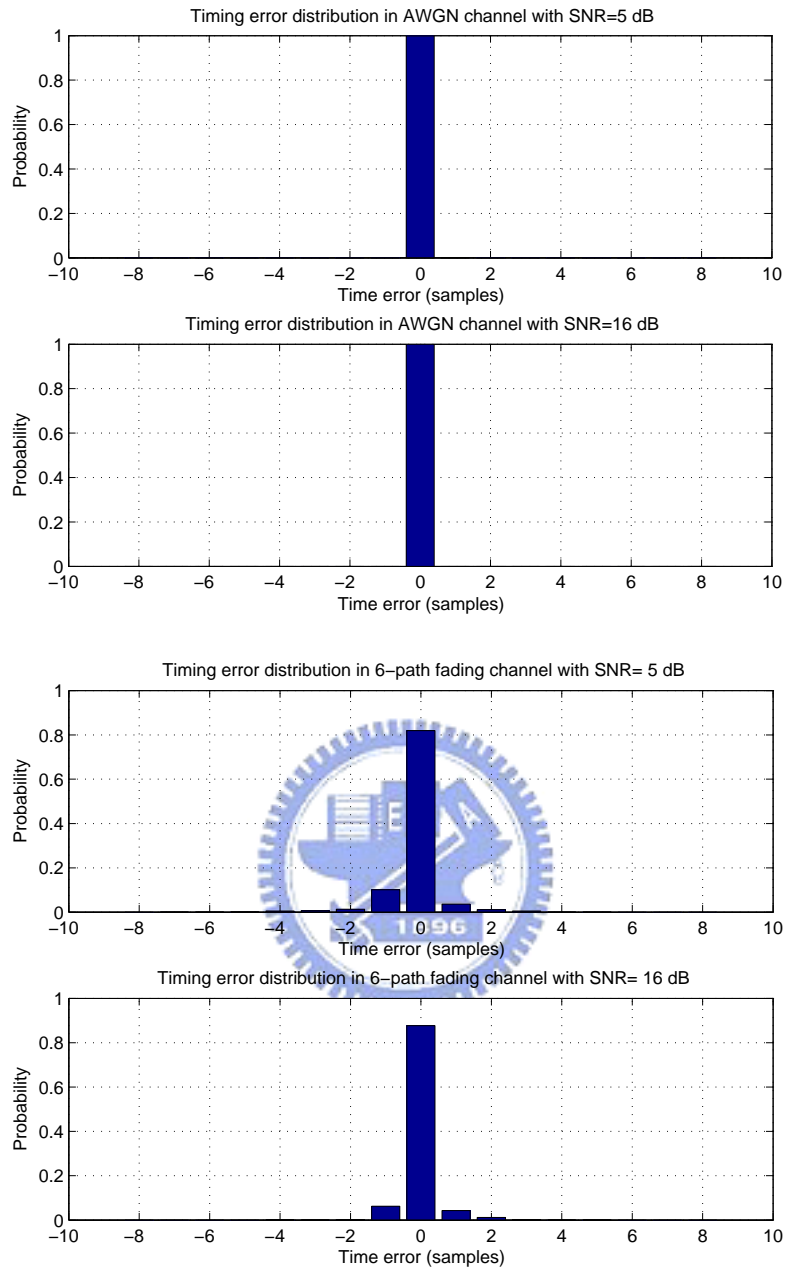


Figure 5.19: Distribution of timing offset estimation errors with fixed-point implementation.

5.5.3 Bit Error Rate Performance

To see the implication on bit error rate (BER) performance of the foregoing CFO synchronization methods, we simulate a transmission system without error-control coding. Figure

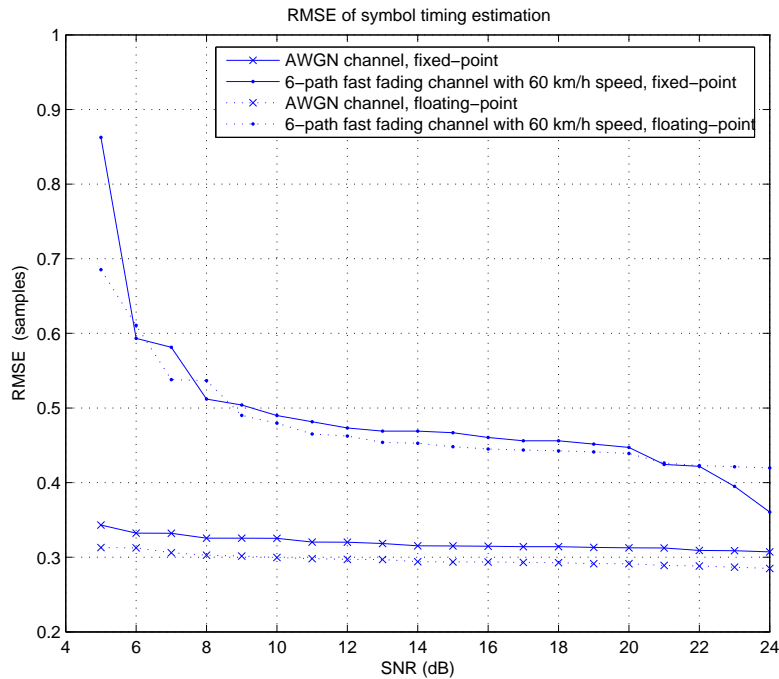
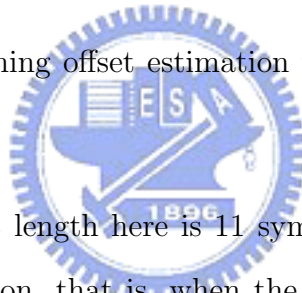


Figure 5.20: RMSE of symbol timing offset estimation with fixed-point and floating-point implementation.



5.26 shows the results. The burst length here is 11 symbols. The bottom curve gives the result under perfect synchronization, that is, when the OFDM symbol timing, CFO, and SFO are estimated and corrected perfectly. The next curve is the case with only 5 ppm of sampling frequency offset. The following two curves correspond to the case with perfect OFDM symbol timing but have 5 ppm SFO and imperfect CFO from using the proposed techniques. We have only considered the simulated CFO synchronization results under 60 km/h multipath fading channel, and have ignored the fading caused by the channel (in other words, we assumed a perfect channel estimator). It is obvious that averaging estimated CFOs is important when the SNR is high.

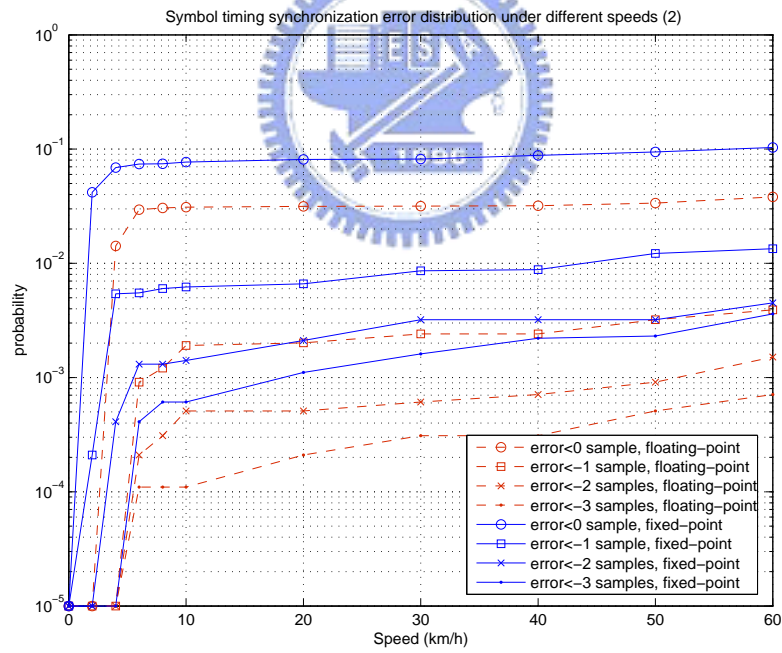
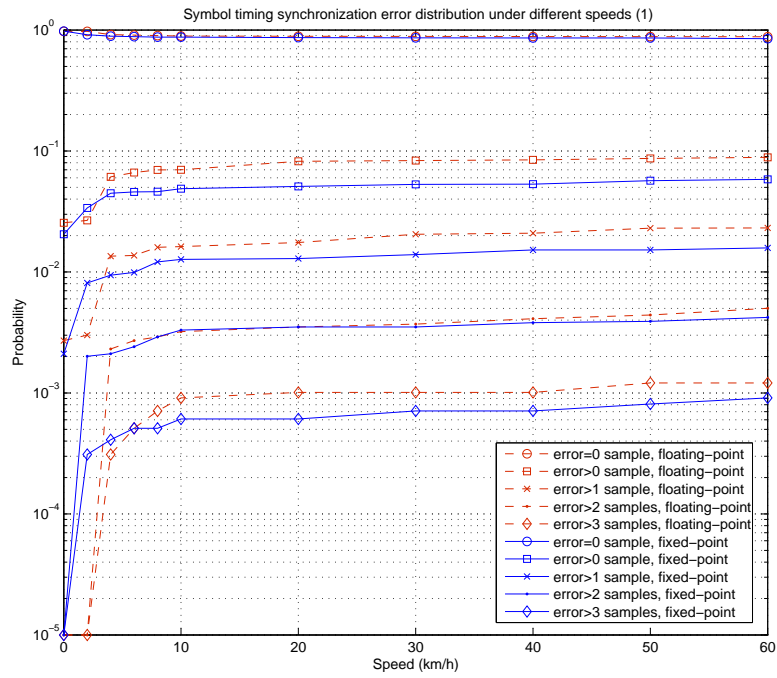


Figure 5.21: Symbol timing estimation error distribution under different speeds with fixed-point and floating-point implementation.

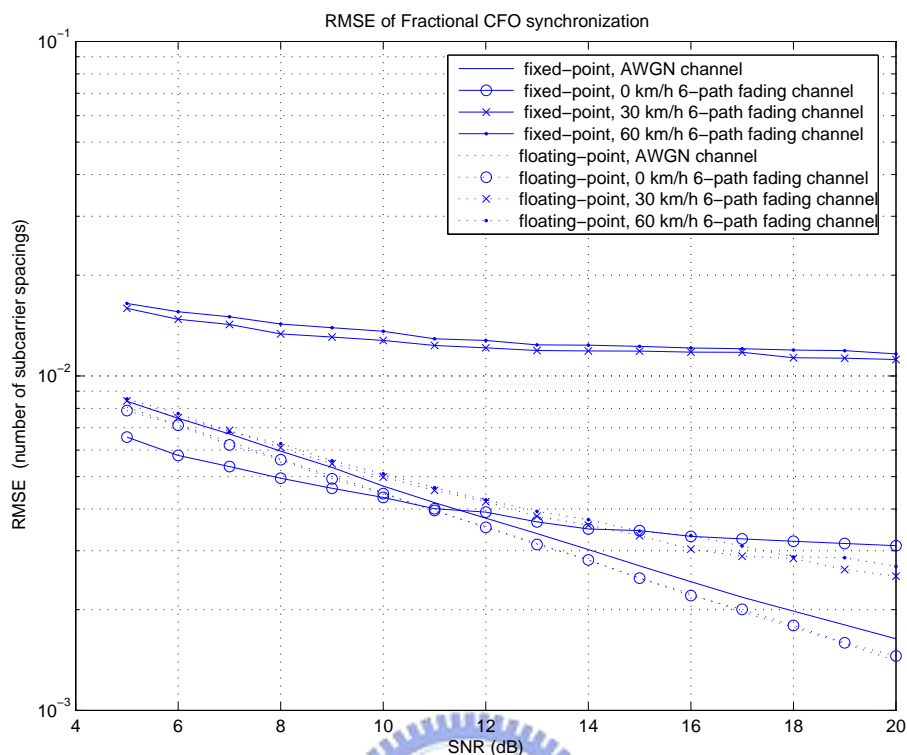


Figure 5.22: RMSE of fractional CFO synchronization with fixed-point and floating-point implementation.

5.6 Program Optimization

This section, we discuss the optimization techniques we use, and the optimized results of the function blocks as in Figure 5.6. Besides utilizing the compiler options we mentioned before, we have to adjust the program for compiler to software-pipeline the loop automatically. Software-pipeline is the most important technique in our optimization work, also, for simplicity, we only show the software-pipelined loops of our functions in this section (except for the “Pilot Generate” function). Usually the loops is the most important part of a function. The optimized profile is given in the later subsection.

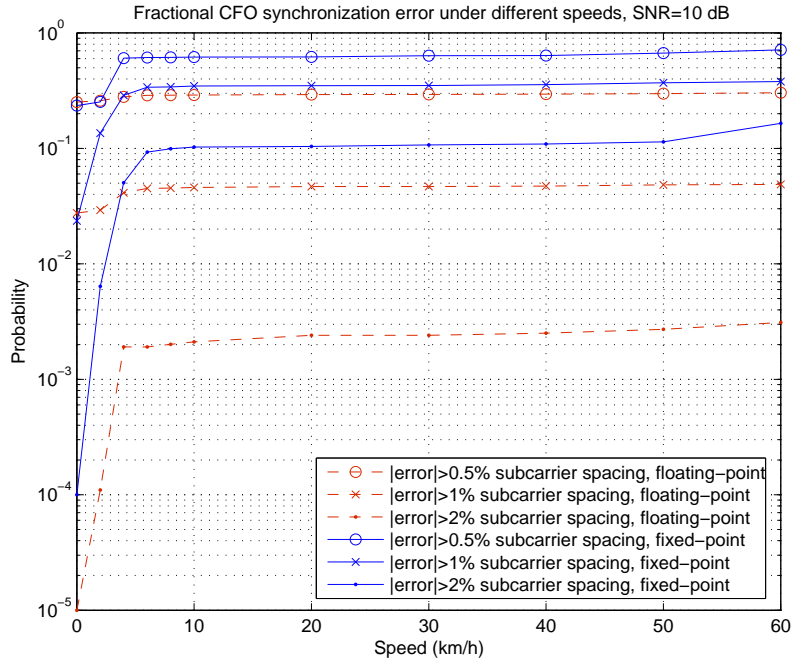


Figure 5.23: Fractional CFO synchronization error distribution under different speeds with fixed-point and floating-point implementation.

5.6.1 The Modulation Function

The modulation function we use is very similar to that used in [7] expect for the length of modulated data. We refer readers to [7] for more information.

5.6.2 The Pilot Generate Function

“Pilot Generate” is a function that works as a PRBS generator for pilot modulation. Figure 5.27 is a part of C and assembly code of this function. Since this function has a very small load for the DSP (see the optimized profile later), we do not further optimize this function for software-pipeline.

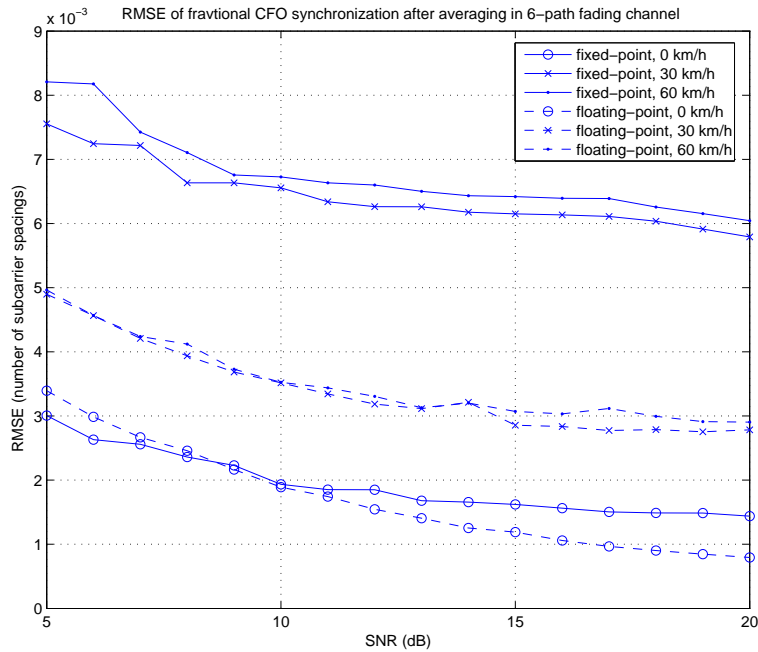


Figure 5.24: RMSE of fractional CFO synchronization after averaging with fixed-point and floating-point implementation.

5.6.3 The Allocation Function

This function is used to allocate pilots, guard band, DC, and data subcarriers. A software-pipelined loop cannot contain control code [18], we must allocate these 4 different kinds of subcarriers in different loops. Figure 5.28 is the code for allocating pilots, DC, guard band, and a part of data subcarriers. Figure 5.29 shows the software-pipeline information (top) and the kernel code for the loop in line 41 of Figure 5.28. We see that this loop needs 2 instructions per cycles. The information of other loops are like this one, so we just show one of them.

5.6.4 The IFFT, Add CP and Tx SRRC, Rx SRRC and down-sample, FFT functions

These four functions are like those used in [7] expect for the processing length, so please see [7] for more information.

5.6.5 The Preamble Synchronization Function

This function is used to estimation symbol timing offset by the preamble. The algorithm has been mentioned in Section 5.1. Figure 5.30 is the loop of the program, and it does the correlation work. Form Fig. 5.31 we see this loop needs a lot of .M and .X instructions. The ii is 14, and the loop unroll multiple is 4, so this loop needs 14/4 instructions per cycle.

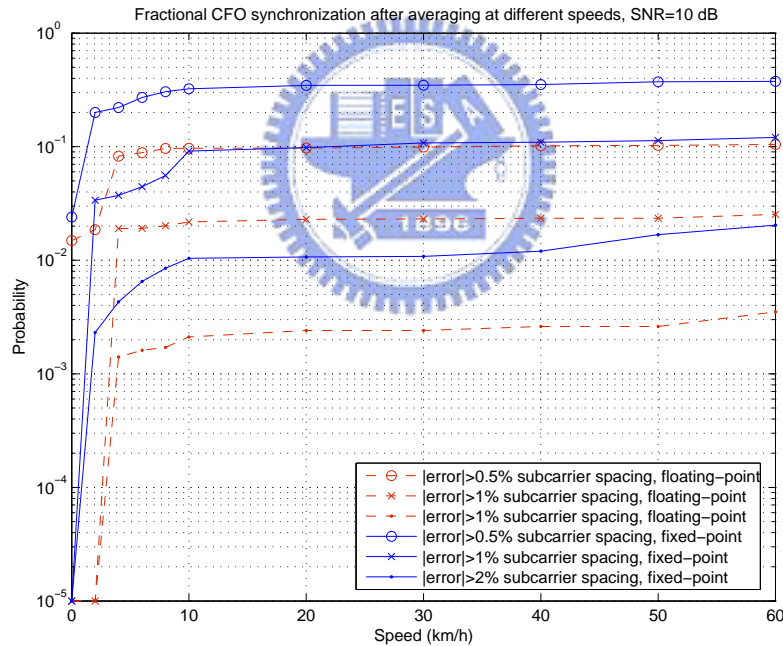


Figure 5.25: Fractional CFO synchronization error distribution under different speeds after averaging with fixed-point and floating-point and floating-point implementation.

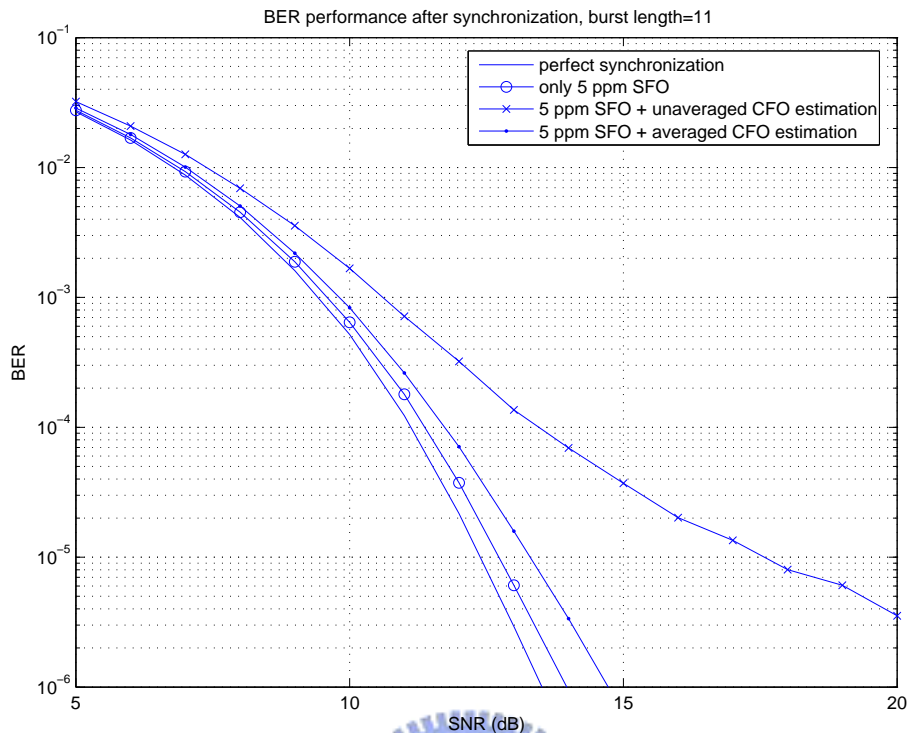


Figure 5.26: BER performance after synchronization at 60 km/h.

5.6.6 The CFO Synchronization Function

This function estimates the fractional CFO by CP correlation. Figure 5.32 shows the loop of this function. We unroll the loop by 4 to speed up by our selves because the compiler cannot software-pipeline this loop no matter how we adjust it. The compiler can only software-pipeline the small loops which is not critical in the overload in this function. Note that this function contains a $\arctan()$ function table, so the code size is large

5.6.7 The Frequency Compensation Function

This function use the CFO value estimated by the CFO synchronization function to compensate the received symbol. Figure 5.33 shows the loop of this function. Since the estimated

```

13 short pilot_gen(int order_sym)
14 {
15     short temp;//for recording PRBS[9]^PRBS[11]
16     short pilot;
17     int j;
18
19     {
20         pilot=PRBS[10];
21         temp=PRBS[8]^PRBS[10];
22         for(j=11;j>0;j--)
23             PRBS[j]=PRBS[j-1];
24
25         PRBS[0]=temp;
26     }
27     return pilot;
28 }
29
73 ;*****
74 _pilot_gen:
75 ;** -----
76     .line    2
77     .sym     _order_sym,4, 4, 17, 32
78     .sym     _temp,59, 3, 4, 16
79     .line    8
80     MVK     .S2      (_PRBS-$bss),B4      ; |20|
81     ADD     .D2      DP,B4,B4              ; |20|
82     LDH     .D2T2    *+B4(20),B5          ; |20|
83     .line   9
84     LDH     .D2T2    *+B4(16),B6          ; |21|
85     NOP
86     XOR     .D2      B5,B6,B7              ; |21|
87     EXT     .S2      B7,16,16,B22         ; |21|
88     .line  11
89     LDH     .D2T2    *+B4(18),B16         ; |23|
90     LDH     .D2T2    *+B4(14),B17         ; |23|
91     LDH     .D2T2    *+B4(12),B18         ; |23|
92     LDH     .D2T2    *+B4(10),B19         ; |23|
93     LDH     .D2T2    *+B4(8),B20          ; |23|
94     LDH     .D2T2    *+B4(6),B21          ; |23|
95     LDH     .D2T2    *+B4(4),B9           ; |23|
96     LDH     .D2T2    *+B4(2),B8           ; |23|
97     LDH     .D2T2    *+DP(_PRBS),B7       ; |23|
98     STH     .D2T2    B5,*+B4(22)         ; |23|
99     STH     .D2T2    B16,*+DP(_PRBS+20) ; |23|
100    STH     .D2T2    B17,*+DP(_PRBS+16) ; |23|
101    STH     .D2T2    B18,*+DP(_PRBS+14) ; |23|
102    STH     .D2T2    B19,*+DP(_PRBS+12) ; |23|
103    STH     .D2T2    B20,*+DP(_PRBS+10) ; |23|
104    STH     .D2T2    B21,*+DP(_PRBS+8)  ; |23|
105    STH     .D2T2    B9,*+DP(_PRBS+6)   ; |23|
106    STH     .D2T2    B8,*+DP(_PRBS+4)   ; |23|
107    STH     .D2T2    B7,*+DP(_PRBS+2)   ; |23|
108    STH     .D2T2    B6,*+DP(_PRBS+18) ; |23|
109    .line  13
110    STH     .D2T2    B22,*+DP(_PRBS)     ; |25|
111    .line  15
112    MV      .D1X     B5,A4                 ; |27|
113    .line  16
114    RETNOP  .S2      B3,5                   ; |28|
115    ; BRANCH OCCURS ; |28|
116    .endfunc      28,000000000h,0
117

```

Figure 5.27: A part of C and assembly code for pilot generate function.


```

16 // allocate DC value
17 RealOut[128]=ImageOut[128]=0;
18 // allocate 8 pilots
19 RealOut[40]=p1;ImageOut[40]=0;
20 RealOut[90]=p1;ImageOut[90]=0;
21 RealOut[141]=p1;ImageOut[141]=0;
22 RealOut[166]=p1;ImageOut[166]=0;
23 RealOut[191]=p1;ImageOut[191]=0;
24 RealOut[216]=p1;ImageOut[216]=0;
25 RealOut[65]=p2;ImageOut[65]=0;
26 RealOut[115]=p1;ImageOut[115]=0;
27 // allocate guard band
28 for(i=0;i<=26;i++)
29 {
30     RealOut[i]=0;RealOut[i+229]=0;
31     ImageOut[i]=0;ImageOut[i+229]=0;
32 }
33 RealOut[27]=p1;ImageOut[27]=0;
34 // allocate 192 data subcarriers
35 for(i=28;i<40;i++)
36 {
37     RealOut[i]=RealIn[j];
38     ImageOut[i]=ImageIn[j];
39     j++;
40 }
41 for(i=41;i<65;i++)
42 {
43     RealOut[i]=RealIn[j];
44     ImageOut[i]=ImageIn[j];
45     j++;
46 }
47 for(i=66;i<90;i++)
48 {
49     RealOut[i]=RealIn[j];
50     ImageOut[i]=ImageIn[j];
51     j++;
52 }

```

Figure 5.28: A part of C code for allocation function.

```

;-----*
;*
;* SOFTWARE PIPELINE INFORMATION
;*
;* Loop source line : 41
;* Loop opening brace source line : 42
;* Loop closing brace source line : 46
;* Known Minimum Trip Count : 24
;* Known Maximum Trip Count : 24
;* Known Max Trip Count Factor : 24
;* Loop Carried Dependency Bound(^) : 0
;* Unpartitioned Resource Bound : 2
;* Partitioned Resource Bound(*) : 2
;* Resource Partition:
;*
;* A-side B-side
;* .L units 0 0
;* .S units 1 0
;* .D units 2* 2*
;* .M units 0 0
;* .X cross paths 0 0
;* .T address paths 2* 2*
;* Long read paths 0 0
;* Long write paths 0 0
;* Logical ops (.LS) 0 0 (.L or .S unit)
;* Addition ops (.LSD) 0 0 (.L or .S or .D unit)
;* Bound(.L .S .LS) 1 0
;* Bound(.L .S .D .LS .LSD) 1 1
;*
;* Searching for software pipeline schedule at ...
;* ii = 2 Schedule found with 3 iterations in parallel
;-----*
L4: ; PIPED LOOP KERNEL

[ A0] BDEC .S1 I4,A0 ; |46| <2,0>
|| LDH .D2T2 *B4++,B6 ; |43| <2,0>
|| LDH .D1T1 *A4++,A5 ; |44| <2,0>

|| STH .D2T2 B6,*B5++ ; |43| <0,5>
|| STH .D1T1 A5,*A3++ ; |44| <0,5>

```

Figure 5.29: Software-pipeline information and a part of assembly code for allocation function.



```

28 //=====128 points correlation=====//
29 for(i=0;i<9;i++)
30 {
31     MulSum=0;
32     for(k=i;k<i+128;k++)
33     {
34         RealMulTemp=(RealPream[k+L]*RealPream[k+L+128]+ImagPream[k+L]*ImagPream[k+L+128])>>16;
35         ImagMulTemp=(ImagPream[k+L]*RealPream[k+L+128]-RealPream[k+L]*ImagPream[k+L+128])>>16;
36         RealMulTemp-=
37             (RealPream[k+L+128]*RealPream[k+L+256]+ImagPream[k+L+128]*ImagPream[k+L+256])>>16;
38         ImagMulTemp-=
39             (ImagPream[k+L+128]*RealPream[k+L+256]-RealPream[k+L+128]*ImagPream[k+L+256])>>16;
40         MulSumTemp=(RealMulTemp*RealMulTemp+ImagMulTemp*ImagMulTemp)>>7;
41         MulSum+=MulSumTemp;
42     }
43     if(MulSum>PreMulSum)
44     {
45         MaxPosi=i;
46         PreMulSum=MulSum;
47     }
48 }
49 SyncResult[0]=MaxPosi;

```

Figure 5.30: A part of C code for preamble synchronization function.

```

-----*
* SOFTWARE PIPELINE INFORMATION
*
* Loop source line : 32
* Loop opening brace source line : 33
* Loop closing brace source line : 42
* Loop Unroll Multiple : 4x
* Known Minimum Trip Count : 32
* Known Maximum Trip Count : 32
* Known Max Trip Count Factor : 32
* Loop Carried Dependency Bound(^) : 2
* Unpartitioned Resource Bound : 12
* Partitioned Resource Bound(*) : 13
* Resource Partition:
*
* A-side B-side
* .L units 0 0
* .S units 9 12
* .D units 3 3
* .M units 12 12
* .X cross paths 12 13*
* .T address paths 6 6
* Long read paths 0 0
* Long write paths 0 0
* Logical ops (.LS) 5 1 (.L or .S unit)
* Addition ops (.LSD) 11 18 (.L or .S or .D unit)
* Bound(.L .S .LS) 7 7
* Bound(.L .S .D .LS .LSD) 10 12
*
* Searching for software pipeline schedule at ...
* ii = 13 Did not find schedule
* ii = 14 Schedule found with 3 iterations in parallel

```

Figure 5.31: Software-pipeline information for preamble synchronization function.



```

105 for(i=0;i<(L+12)/4;i++)
106 {
107     RealMul1=(RealIn[4*i]*RealIn[4*i+N]+ImagIn[4*i]*ImagIn[4*i+N])>>15; RealSum1=RealSum1+RealMul1-RealWindow[L-1];
108     ImagMul1=(ImagIn[4*i]*RealIn[4*i+N]-RealIn[4*i]*ImagIn[4*i+N])>>15; ImagSum1=ImagSum1+ImagMul1-ImagWindow[L-1];
109     RealMul2=(RealIn[4*i+1]*RealIn[4*i+N+1]+ImagIn[4*i+1]*ImagIn[4*i+N+1])>>15; RealSum2=RealSum1+RealMul2-RealWindow[L-2];
110     ImagMul2=(ImagIn[4*i+1]*RealIn[4*i+N+1]-RealIn[4*i+1]*ImagIn[4*i+N+1])>>15; ImagSum2=ImagSum1+ImagMul2-ImagWindow[L-2];
111     RealMul3=(RealIn[4*i+2]*RealIn[4*i+N+2]+ImagIn[4*i+2]*ImagIn[4*i+N+2])>>15; RealSum3=RealSum2+RealMul3-RealWindow[L-3];
112     ImagMul3=(ImagIn[4*i+2]*RealIn[4*i+N+2]-RealIn[4*i+2]*ImagIn[4*i+N+2])>>15; ImagSum3=ImagSum2+ImagMul3-ImagWindow[L-3];
113     RealMul4=(RealIn[4*i+3]*RealIn[4*i+N+3]+ImagIn[4*i+3]*ImagIn[4*i+N+3])>>15; RealSum4=RealSum3+RealMul4-RealWindow[L-4];
114     ImagMul4=(ImagIn[4*i+3]*RealIn[4*i+N+3]-RealIn[4*i+3]*ImagIn[4*i+N+3])>>15; ImagSum4=ImagSum3+ImagMul4-ImagWindow[L-4];
115
116
117     for(j=L-1;j>=4;j--)
118     {
119         RealWindow[j]=RealWindow[j-4];
120         ImagWindow[j]=ImagWindow[j-4];
121     }
122
123     RealWindow[3]=RealMul1;     corre_amp[4*i]=(RealSum1*RealSum1+ImagSum1*ImagSum1)>>5;// ">>5" means divides by 32!!
124     ImagWindow[3]=ImagMul1;     corre_phase[0][4*i]=ImagSum1;
125     RealWindow[2]=RealMul2;     corre_phase[1][4*i]=RealSum1;
126     ImagWindow[2]=ImagMul2;     corre_amp[4*i+1]=(RealSum2*RealSum2+ImagSum2*ImagSum2)>>5;// ">>5" means divides by 32!!
127     RealWindow[1]=RealMul3;     corre_phase[0][4*i+1]=ImagSum2;
128     ImagWindow[1]=ImagMul3;     corre_phase[1][4*i+1]=RealSum2;
129     RealWindow[0]=RealMul4;     corre_amp[4*i+2]=(RealSum3*RealSum3+ImagSum3*ImagSum3)>>5;// ">>5" means divides by 32!!
130     ImagWindow[0]=ImagMul4;     corre_phase[0][4*i+2]=ImagSum3;
131                                 corre_phase[1][4*i+2]=RealSum3;
132     corre_amp[4*i+3]=(RealSum4*RealSum4+ImagSum4*ImagSum4)>>5;// ">>5" means divides by 32!!
133     corre_phase[0][4*i+3]=ImagSum4;
134     corre_phase[1][4*i+3]=RealSum4;
135
136     // Find the position of the correlation values
137     if(corre_amp[4*i]>corre_amp[max])
138     max=4*i;
139     if(corre_amp[4*i+1]>corre_amp[max])
140     max=4*i+1;
141     if(corre_amp[4*i+2]>corre_amp[max])
142     max=4*i+2;
143     if(corre_amp[4*i+3]>corre_amp[max])
144     max=4*i+3;
145
146     RealSum1=RealSum4;
147     ImagSum1=ImagSum4;
148 }

```

Figure 5.32: A part of C code for CFO synchronization function.

CFO may be larger than zero or smaller than zero, we divide our program into two loops for the compiler can software-pipeline them. Figure 5.34 shows the software-pipeline information of the loop in line 169. We see this loop needs $12/4 = 3$ instructions per cycles. Please see the Figs. 5.35 and 5.36 for the assembly code of this loop. The loop in line 180 has the similar information. Note that this function contains a $\sin()$ and a $\cos()$ tables, so the code size is large.

5.7 Profile of Optimized DSP Program


We make use of the techniques described in the previous chapter to optimize the fixed-point program. In our system, the clock frequency of DSP is 600 MHz, and one symbol duration is $25 \mu s$ (288 samples). Therefore, the available execution clock cycles are 15000 in a symbol duration, averaging to approximately 52 in a sample duration. To achieve real-time processing speed, one symbol must execute less than 15000 instruction cycles. Table 5.5 shows the code size and the number of clock cycles each block in Figure 5.6 takes, where “load” gives the fraction of a DSP’s real-time computing power. (Note that in programming terms, an “optimized” program does not mean that a program has been made ultimately efficient without any possibility of further improvement. It merely means that suitable programming techniques have been used in writing the program to make it reasonably efficient compared to one without using such techniques.) In Table 5.5 we can see that every block can be executed in real-time individually. The total requirement for the transmitter functions is approximately 1.17 DSP chips’ processing power, where that for the receiver functions is 3.08 DSP chips’ power. Further reduction in clock cycles should be possible, but is not considered in the present work. Figure 5.37 shows the percentage of each function in the Tx or the Rx.

```

161 //=====CFO compensate and remove CP=====//
162 for(i=0;i<(N+L);i++)
163 {
164     index[i]=(i*offset)>>10;
165 }
166
167 if(SyncResult[1]<0)
168 {
169     for(i=0;i<(N);i++)
170     {
171         cos_temp1=cos_tab[(-index[i+L])];
172         sin_temp1=-sin_tab[(-index[i+L])];
173
174         RealOut[i]=(FIXED)((RealIn[i+L]*cos_temp1-ImagIn[i+L]*sin_temp1)>>15);
175         ImagOut[i]=(FIXED)((RealIn[i+L]*sin_temp1+ImagIn[i+L]*cos_temp1)>>15);
176     }
177 }
178 if(SyncResult[1]>=0)
179 {
180     for(i=0;i<(N);i++)
181     {
182         cos_temp1=cos_tab[index[i+L]];
183         sin_temp1=sin_tab[index[i+L]];
184
185         RealOut[i]=(FIXED)((RealIn[i+L]*cos_temp1-ImagIn[i+L]*sin_temp1)>>15);
186         ImagOut[i]=(FIXED)((RealIn[i+L]*sin_temp1+ImagIn[i+L]*cos_temp1)>>15);
187     }
188 }

```

Figure 5.33: A part of C code for frequency compensate function.



```

;-----*
;* SOFTWARE PIPELINE INFORMATION
;*
;* Loop source line : 169
;* Loop opening brace source line : 170
;* Loop closing brace source line : 176
;* Loop Unroll Multiple : 4x
;* Known Minimum Trip Count : 64
;* Known Maximum Trip Count : 64
;* Known Max Trip Count Factor : 64
;* Loop Carried Dependency Bound(^) : 3
;* Unpartitioned Resource Bound : 11
;* Partitioned Resource Bound(*) : 12
;* Resource Partition:
;*
;* A-side B-side
;* .L units 0 0
;* .S units 6 7
;* .D units 10 12*
;* .M units 11 5
;* .X cross paths 8 11
;* .T address paths 12* 12*
;* Long read paths 0 0
;* Long write paths 0 0
;* Logical ops (.LS) 2 9 (.L or .S unit)
;* Addition ops (.LSD) 11 7 (.L or .S or .D unit)
;* Bound(.L .S .LS) 4 8
;* Bound(.L .S .D .LS .LSD) 10 12*
;*
;* Searching for software pipeline schedule at ...
;* ii = 12 Schedule found with 4 iterations in parallel
;*

```

Figure 5.34: Software-pipeline information for frequency compensate function.

```

L5:      ; PIPED LOOP KERNEL

      [ B0]  MPYSU  .M2   2,B0,B0      ; <0,25>
||        MPYHL  .M1X  A19,B18,A5    ; |175| <0,25>
||        LDNDW  .D2T1 *B4++(8),A5:A4 ; |174| <1,13>
||        NEG    .S2X  A9,B20       ; |171| <1,13>
||        NEG    .L2   B17,B18      ; |171| <1,13>

      EXT    .S2   B19,16,16,B7     ; |172| <0,26>
||        MPY    .M1X  A8,B16,A5    ; |175| <0,26>
||        SUB    .L2X  A5,B7,B21    ; |174| <0,26>
|| [ A2]  LDH    .D2T2 *+B23[B20],B24 ; |172| <1,14>
|| [ A1]  LDW    .D1T1 *++A22(16),A25 ; |171| <2,2>

      MPYHL  .M1   A3,A26,A3       ; |175| <0,27>
||        MPYHL  .M2X  A3,B19,B8    ; |174| <0,27>
||        ADD    .L1   A17,A5,A3    ; |175| <0,27>
||        SHR    .S2   B21,15,B19   ; |174| <0,27>
|| [ A2]  LDH    .D2T1 *+B8[B18],A27 ; |171| <1,15>
||        NEG    .L2   B16,B21     ; |172| <1,15>
||        ADD    .S1X  8,SP,A20     ; |172| <2,3>
|| [ A1]  LDW    .D1T2 *+A22(8),B17  ; |171| <2,3>

      ADD    .L1   A23,A5,A8       ; |175| <0,28>
||        SHR    .S1   A3,15,A24    ; |175| <0,28>
||        MPYHL  .M1X  A8,B7,A6    ; |175| <0,28>
||        EXT    .S2   B21,16,16,B20 ; |172| <1,16>
|| [ A2]  LDH    .D2T1 *+B22[B20],A26 ; |171| <1,16>
||        ADD    .L2X  A18,SP,B22   ; |171| <2,4>
|| [ A1]  LDW    .D1T2 *+A22(4),B17  ; |171| <2,4>

      [ A0]  SUB    .S1   A0,1,A0    ; |176| <0,29>
||        SHR    .S2   B9,15,B9     ; |175| <0,29>
||        SUB    .L2X  A25,B17,B17  ; |174| <0,29>
|| [!B0]  STH    .D1T1 A4,*+A16(8)  ; |174| <0,29>
||        MPY    .M1X  A6,B16,A6    ; |174| <1,17>
|| [ A2]  LDH    .D2T2 *+B25[B18],B18 ; |172| <1,17>
||        MV     .L1   A6,A19       ; |174| <1,17>

      SHR    .S2   B17,15,B18     ; |174| <0,30>
||        ADD    .L1   A3,A6,A26    ; |175| <0,30>
||        SUB    .S1X  A9,B8,A9     ; |174| <0,30>
||        MV     .D1   A5,A3       ; |174| <1,18>
|| [ A1]  LDNDW  .D2T1 *B5++(8),A7:A6 ; |174| <2,6>
||        ADD    .L2X  A18,SP,B8    ; |171| <2,6>

```

Figure 5.35: A part of assembly code for frequency compensate function-I.

```

[!B0]   STH     .D2T2  B9,+++B6(8)      ; |175| <0,31>
|| [ A0]   B      .S1    L5                      ; |176| <0,31>
||         MV     .L1    A4,A23                 ; |174| <1,19>
||         MPY    .M1X   A4,B21,A4             ; |174| <1,19>
||         ADD    .S2X   A18,SP,B9            ; |171| <2,7>
|| [ A1]   LDW    .D1T1  **A22(12),A9         ; |171| <2,7>

[!B0]   STH     .D2T1  A24,++B6(2)          ; |175| <0,32>
|| [!B0]   STH     .D1T2  B19,++A16(2)       ; |174| <0,32>
||         SHR     .S1    A26,15,A25         ; |175| <0,32>
||         MPY    .M2X   A19,B20,B20        ; |175| <1,20>
||         MPY    .M1    A7,A17,A25         ; |174| <1,20>
||         ADD    .L2    8,SP,B25           ; |172| <2,8>
||         NEG    .L1    A25,A24           ; |171| <2,8>
||         NEG    .S2    B17,B19           ; |171| <2,8>

[!B0]   STH     .D1T2  B18,++A16(4)         ; |174| <0,33>
||         SHR     .S1    A8,15,A4          ; |175| <0,33>
|| [!B0]   STH     .D2T1  A25,++B6(6)       ; |175| <0,33>
||         SUB     .L1    A6,A4,A6          ; |174| <1,21>
||         MPY    .M2X   A23,B16,B21        ; |175| <1,21>
||         MPYHL  .M1    A23,A27,A17        ; |175| <1,21>
||         ADD    .S2    8,SP,B23          ; |172| <2,9>
||         ADD    .L2    8,SP,B16          ; |172| <2,9>

[!B0]   STH     .D2T1  A4,++B6(4)           ; |175| <0,34>
||         MV     .L1    A7,A8              ; |174| <1,22>
||         NEG    .S2    B18,B18           ; |172| <1,22>
||         NEG    .L2    B7,B7             ; |172| <1,22>
||         MPY    .M1    A3,A17,A23        ; |175| <1,22>
||         SHR     .S1    A6,15,A4         ; |174| <1,22>
|| [ A1]   LDH     .D1T2  **A20[A24],B16     ; |172| <2,10>

[ A2]   SUB     .L1    A2,1,A2              ; <0,35>
||         SHR     .S1    A9,15,A20         ; |174| <0,35>
||         ADD    .L2    B21,B20,B9        ; |175| <1,23>
||         MPYHL  .M2X   A23,B18,B7        ; |174| <1,23>
||         EXT    .S2    B18,16,16,B18     ; |172| <1,23>
||         MPYHL  .M1    A19,A27,A5        ; |174| <1,23>
|| [ A1]   LDH     .D2T1  **B9[B19],A17     ; |171| <2,11>
|| [ A1]   LDH     .D1T2  **A21[A24],B16     ; |171| <2,11>

[ A1]   SUB     .S1    A1,1,A1              ; <0,36>
|| [!B0]   STH     .D1T1  A20,++A16(6)       ; |174| <0,36>
||         MPYHL  .M1    A8,A26,A9         ; |174| <1,24>
||         NEG    .L2    B24,B19           ; |172| <1,24>
||         MPY    .M2X   A5,B7,B17         ; |174| <1,24>
||         EXT    .S2    B7,16,16,B16     ; |172| <1,24>

|| [ A1]   LDH     .D2T2  **B16[B19],B7     ; |172| <2,12>
||         ADD    .L1X   A18,SP,A21        ; |171| <3,0>

; ** -----

```

Figure 5.36: A part of assembly code for frequency compensate function-II.

Table 5.5: Profile of Synchronization Function Blocks

Blocks	Code Size (Bytes)	Avg. Clock Cycles	Load (# DSPs)	
Modulation	472	537	0.0358	T x
Pilot Generate	120	38	0.0025	
Subcarrier Alloc.	1368	1064	0.0709	
IFFT*	676	6802	0.4535	
Add CP and Tx SRRC	2660	9143	0.6096	
Rx SRRC and Downsample	3872	13584	0.9056	R x
Preamble Sync.	976	4860	0.3240	
CFO Estimation	1592	9621	0.6414	
Frequency Compensate	3484	11229	0.7486	
FFT*	536	6906	0.4604	

* Note: TI DSPLIB employed.

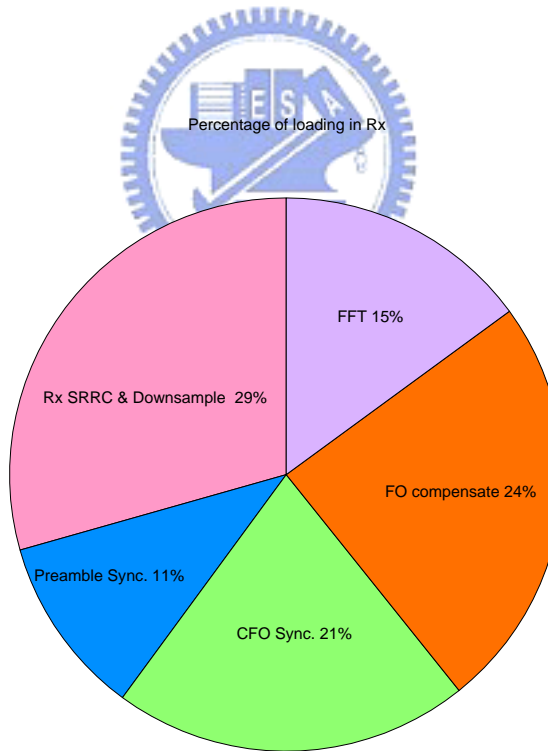
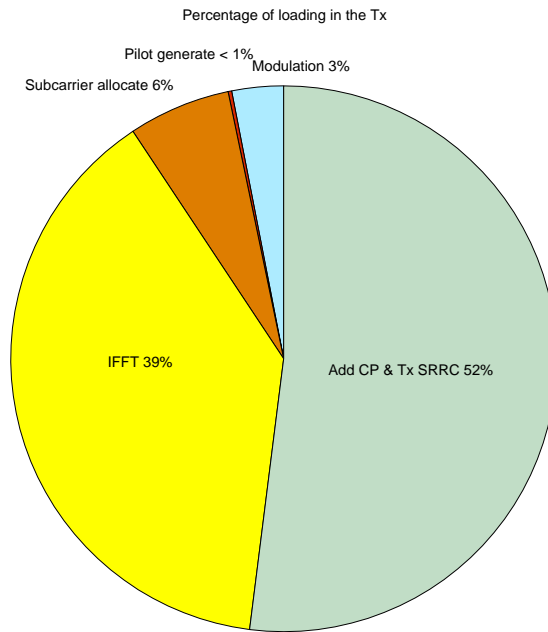
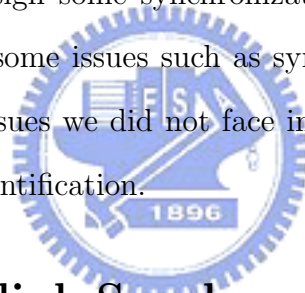


Figure 5.37: Percentage of DSP loading in the Tx and the Rx.

Chapter 6

OFDMA TDD Downlink Synchronization

In this chapter, we discuss synchronization problems of the IEEE 802.16e OFDMA TDD downlink system, and find or design some synchronization techniques to overcome them. OFDMA is much like OFDM in some issues such as symbol timing offset, fractional CFO and SFO, but it also has other issues we did not face in the last chapter like integer CFO correction and preamble index identification.



6.1 OFDMA Downlink Synchronization Problems and Techniques

There are some differences between OFDMA downlink and OFDM uplink:

1. In OFDMA downlink, the SS does not know which preamble is used in advance. Since there are 114 different preambles, an SS must find the preamble index used in the currently received frame.
2. Different preamble indexes correspond to different IDcells and different used segments, so an SS must find the preamble index before processing the following symbols.

3. In the downlink, due to potentially large tolerance in the free-running oscillator frequency of the SS, and due to motion-induced Doppler spread, there may be large carrier frequency offset and sampling frequency offset in the received signal. Hence there may be integer CFO, and the SFO may be large that the SS cannot ignore it.
4. The 802.16e specifies that the transmitted center frequency and symbol clock frequency must be derived from the same reference oscillator both at the BS and the SS, so the CFO and the SFO shall have the same error ratio, and we can estimate and correct them together.

First we use the preamble symbol to estimate the timing offset and the integer CFO since these two parameters are fixed in the whole frame even at high mobile speed. Because the preamble index decides the IDcell and the used segment, the preamble index should be found before the subsequent symbols. The fractional CFO is estimated in every symbol including the preamble, and the estimated values are averaged for more accurate results. Since the transmitted center frequency and symbol clock frequency are reference to the same oscillator, sampling frequency synchronization can be ignored. Fig. 6.1 shows the synchronizer structure of the SS. The dotted block “fine timing synchronization” can be present or absent.

6.1.1 Timing Offset and Fractional Carrier Frequency Offset

In timing offset and fractional CFO estimation, we use a similar technique as that used in Chapter 5. Thus we do not repeat the detail here. We only note that since we do not yet know which preamble is used in this step, we just view preamble symbol as a regular symbol and estimate timing offset and fractional CFO by using CP correlation [21]. However, the particular quasi-periodic use of the subcarrier in the preamble may give use to some time-domain symbol structure that can be exploited to some advantage; we leave this to potential

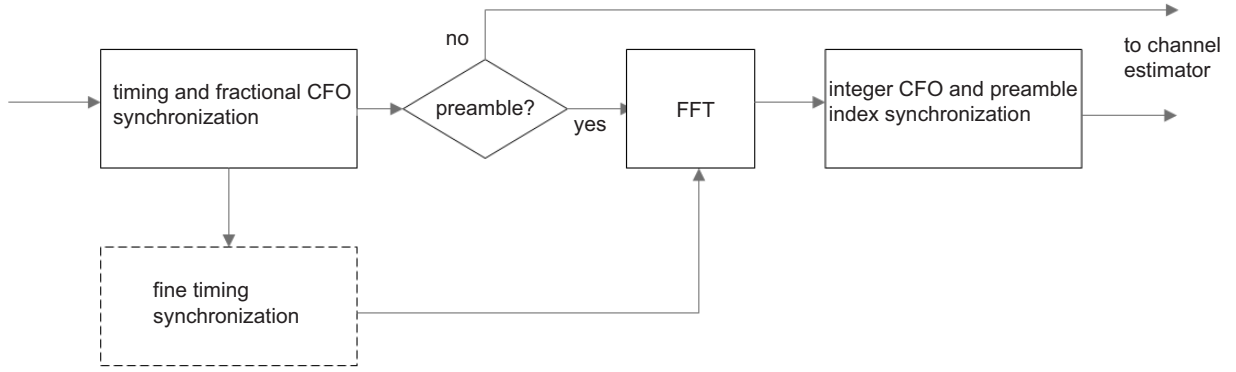


Figure 6.1: The proposed OFDMA synchronizer structure.

future work.

The equation we use to average estimated CFOs for more accuracy is also different from that for OFDM. After many experiments, we choose the equation below:

$$\begin{aligned} \hat{\epsilon}_{k+1} &= \frac{\hat{\epsilon}_k \cdot k + \epsilon_k}{k+1} \text{ for } 0 < k < 10, \\ \hat{\epsilon}_{k+1} &= \hat{\epsilon}_{k+1} \times 0.99 + \epsilon_k \times 0.01 \text{ for } k \geq 10. \end{aligned} \quad (6.1)$$

where $\hat{\epsilon}_k$, ϵ_k , and k are as in (5.6).

6.1.2 Integer Carrier Frequency Offset

The integer CFO should be estimated during the preamble symbol duration and before estimating the preamble index. If we do not know the integer CFO we cannot estimate the preamble index correctly, and the following processing will be meaningless.

In our work, we synchronize the integer CFO by using the frequency-domain structure preamble. Note that there are three types of preamble carrier-sets, and each segment uses only one carrier-set. Carrier-set 0 uses subcarrier indexes 172, 175, 178, ..., 1870, 1873, the subcarrier indexes that carrier-set 1 uses are those used by carrier-set 0 adding 1, and adding 2 for carrier-set 2. If the SS knows which carrier-set the BS uses, it can synchronize to

the correct integer CFO exactly by shifting the preamble symbol first, and then computing the power of the subcarriers in the carrier-set. For example if the integer CFO is f_I , the maximum possible integer CFO is f_{max} , and the BS uses carrier-set 1, then we can use the following algorithm to determine the integer CFO:

```

for( $i = -f_{max}; i \leq f_{max}; i++$ )
{
    sum=0;
    for( $j = 173; j \leq 1871; j+ = 3$ )
    {
        sum+=pow(Real_preamble[ $j + i$ ],2)+pow(Image_preamble[ $j + i$ ],2);
    }
}

```

where “pow” is the C function used to compute the power in the preamble subcarriers. The i which has the maximum “sum” is the estimated integer CFO.

From the algorithm, we see that for each i , there needs $567(\text{subcarrier number in the carrier-set}) \times 2(\text{real-part and imaginary-part}) = 1134$ multiplications, and 1134 additions. But actually there are many terms repeated across different i , so we can disregard those repeated terms without affection the performance. If we set the f_{max} as $15\Delta f$ (about ± 23 ppm accuracy of SS's oscillator), we can reduce the number of required multiplications and additions to $20 \times 2 = 40$, about 96.5% of reduction in complexity. The inner loop of the algorithm then becomes:

```

for( $j = 173; j \leq 173 + 9 \times 3; j+ = 3$ )
{
    sum+=pow(Real_preamble[ $j + i$ ],2)+pow(Image_preamble[ $j + i$ ],2);
}

```

```

for( $j = 1874 - 9 \times 3; j \geq 1874; j += 3$ )
{
    sum += pow(Real_preamble[j + i], 2) + pow(Image_preamble[j + i], 2);
}

```

Let us further analyze the complexity of the two above algorithms in general. The original algorithm needs a total of $567 \times 2 \times (2f_{max} + 1)$ multiplications and additions. In the modified algorithm, if $f_{max} \bmod 3$ is 0 or 1, then it needs a total of $\text{floor}(f_{max}/3) \times 2 \times 4 \times (2f_{max} + 1)$, otherwise it needs $(\text{floor}(f_{max}/3) \times 2 + 1) \times 4 \times (2f_{max} + 1)$. If the accuracy of the SS's oscillator will be no more than 100 ppm, then the largest possible integer CFO is $64\Delta f$. We show the complexity of the two algorithms for f_{max} in the range 1 to $64\Delta f$ in Fig. 6.2. The additions have the same complexity as multiplications.

Actually the SS does not know the used carrier-set in advance, but there still is a simple relationship of the estimated integer CFO between different carrier-sets. If the SS takes carrier-set 1 to synchronize the integer CFO as above, and the estimated integer CFO is f_I , then the result will be $f_I + 1$ if the SS worked on carrier-set 0, and $f_I - 1$ if the SS worked on carrier-set 2. So we can take carrier-set 1 as the carrier-set to estimate the integer CFO, but keep in mind that the actual situation may be $f_I + 1$ with carrier-set 0, f_I with carrier-set 1, or $f_I - 1$ with carrier-set 2. The actual situation will be decided after preamble index identification, which is discussed in the next subsection.

6.1.3 Preamble Index Identification

In our design, the preamble index identification is performed right after the integer CFO synchronization. The previous stage gives us three possibilities: $f_I + 1$ with carrier-set 0, f_I with carrier-set 1, or $f_I - 1$ with carrier-set 2. In this stage we will find which one is correct

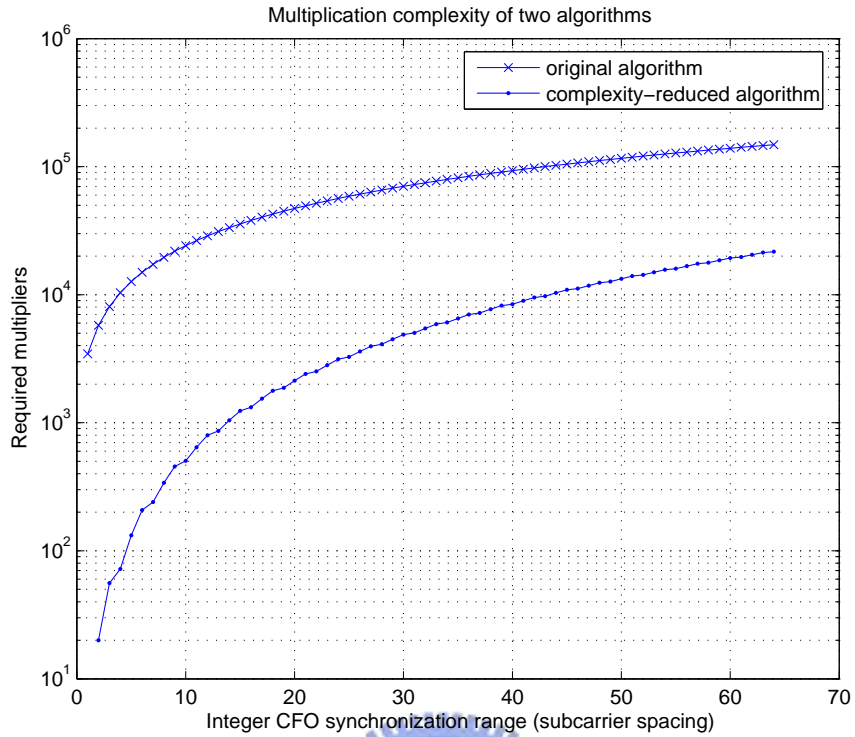


Figure 6.2: Multiplication complexity of two algorithms.

and find the correct preamble index out of 114 candidates.

The method we use to identify the preamble index is intuitive. First we generate a known preamble, and shift its subcarriers depending on the possible integer CFO of the carrier-set it belongs to, then we correlate this frequency-shifted preamble with the received symbol. After 114 preambles are shifted and correlated, choose one that has the largest correlation value, and this is the estimated preamble index. For example, if we use preamble index 0 to correlate with the received symbol, then we have the following algorithm:

```

i = fI + 1
for(j = 172; j ≤ 1873; j += 3)
{
    sum_real += Real_preamble[j + i] * known_Preamble[j];
}

```

```

sum_imag+=Imag_preamble[j + i]*known_Preamble[j];
}
sum=pow(sum_real,2)+pow(sum_imag,2);

```

where “sum” is the correlation value, “Real_preamble” is the real part of the received symbol, and “Imag_preamble” is the imaginary part of received symbol. In total we need $114 \times (567 \times 2 + 2)$ multiplications.

There is also another simple method to synchronize integer CFO and preamble index at once, but the complexity is surprisingly large, and the performance is inferior: After receiving a preamble, we may generate known preambles and shift subcarriers of every preamble in the range from $-f_I$ to f_I , then, correlate these $114 \times (2f_I + 1)$ subcarriers shifted preambles with the received preamble, choose one which has the largest correlation value. We give a simple comparison of their performance in the later section.

6.1.4 Fine Symbol Timing Estimation

Generally speaking, the effects caused by negative errors (resp. positive errors) in symbol timing synchronization can be eliminated (resp. mitigated) by channel estimation. But an accurate timing estimation can improve the performance of the integer CFO and preamble index identification; we will discuss this point later.

Note that the symbol timing offset in a burst (or even in a subframe) are all the same even in high speed environment. Although each symbol may have error in its timing estimation, theoretically speaking, if we observe many consecutive symbols, the correct offset value should appear most frequently. So, we observe consecutive estimated symbol timings first, and then see which one appears most frequently. The observed length may be 10, 25, or 50 symbols in our tries. This method can attain a high accuracy, but the disadvantage is that we need to store many symbols before making the final decision.

Table 6.1: OFDMA Receiver SNR Assumptions

Modulation	Coding rate	Receiver SNR(dB)
QPSK	1/2	5.0
QPSK	3/4	8.0
16-QAM	1/2	10.5
16-QAM	3/4	14.0
64-QAM	2/3	18.0
64-QAM	3/4	20.0

6.2 Floating-Point Simulation Results

The system profile parameters we use have been given in Chapter 3, and the channel environments are given in Chapter 5. Like in OFDM, we only use 16-QAM modulation in our simulation for simplicity. The specifications on the receiver SNR are shown in Table 6.1. Our simulated SNR values are in the range of 0 dB to 20 dB, which is a suitable range for 16-QAM modulation. The mobile speed is from 0 to 300 km/h, and the CFO is $9.25\Delta f$. The symbol timing offset is a random number between 0 to 49 samples. Note that we do not take sampling inaccuracy caused by the SFO into consideration in our simulation.

6.2.1 Symbol Timing Estimation

Figure 6.3 shows timing error distribution in 6-path fast Rayleigh fading channel with SNR at 10 dB and 20 dB. The mobile speed is 240 km/h. We can see that the correct rate (the probability of timing offset 0) is not very high even at high SNR. This is because the speed 240/h makes the channel response vary fast and greatly even in one symbol duration, which causes the CP correlation to perform badly. If we run simulation in AWGN channel, the correct rate are almost 100%, similar to the upper two charts in Fig. 5.8. This is because the length of 256-sample CP is long enough to alleviate white Gaussian noise effect.

In Figures 6.4 and 6.5, we see how different speeds affect the error distribution of symbol

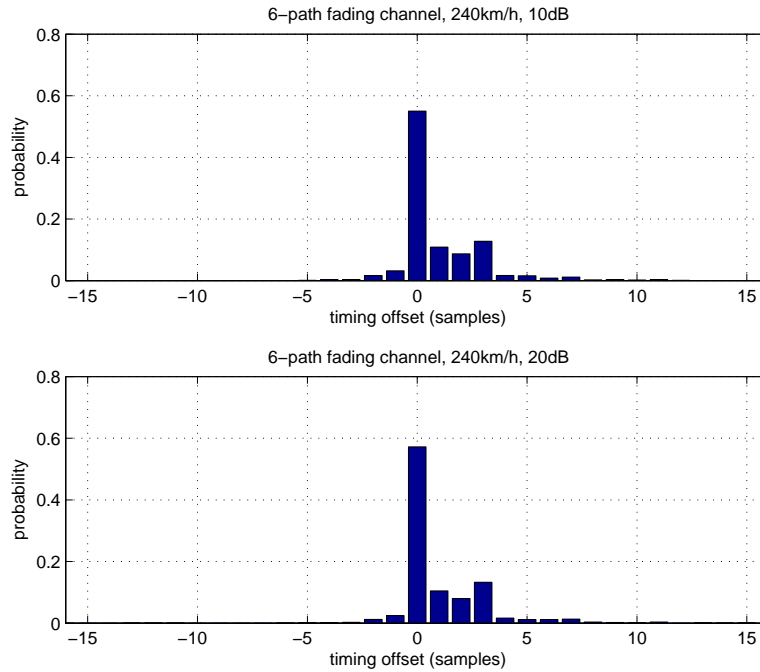


Figure 6.3: Distribution of timing offset estimation errors.

timing synchronization. Almost 99.5% of errors are under ± 16 samples, which is required by the specification ($2048/32 \times 25\% = 16$). Note that we run 5000 symbols for simulation, so the error rate 1×10^{-4} actually means no error has occurred in our simulation. The SNR here is 9 dB.

Figure 6.6 shows the RMSE, which has the same definition in the previous chapter, of our method.

6.2.2 Fractional CFO Estimation

Figure 6.7 shows the RMSE of fractional CFO estimation in the multipath fading channel. The top figure is result of estimated CFO without averaging, and the bottom figure is result after averaging estimated CFOs by (6.1). Obviously, the bottom figure has better

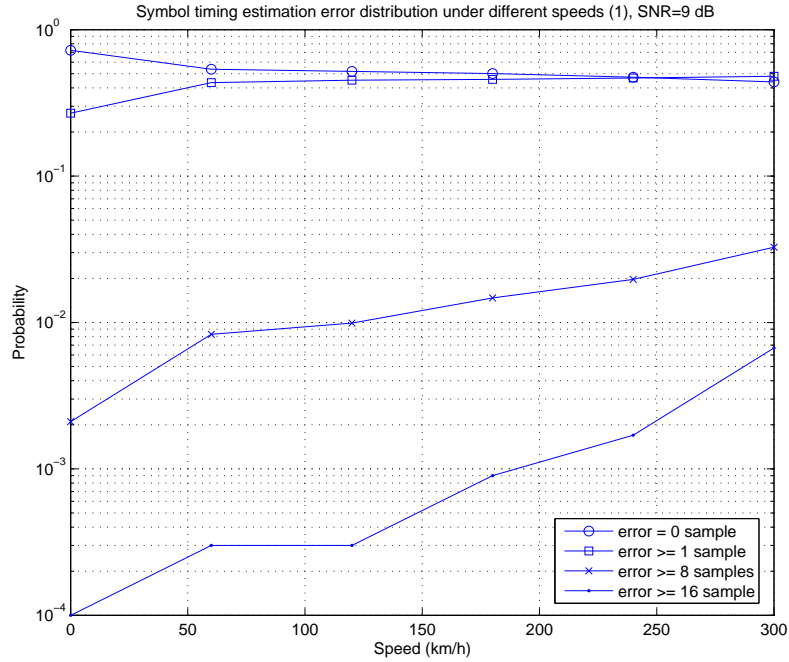


Figure 6.4: Symbol time synchronization error distribution under different speeds (i).

performance than the top one. Actually we cannot satisfy the frequency requirement of the IEEE 802.16e without taking the average, so let us only examine how different speeds affect the error distribution of fractional CFO synchronization with averaging. In Fig. 6.8 we can discover that the speed affects the synchronization results greatly. Since the transmitted center frequency and symbol clock frequency reference to the same oscillator, $2\% \Delta f$ CFO means 9.766 ppm SFO, $1\% \Delta f$ CFO means 4.883 ppm SFO, and $0.5\% \Delta f$ CFO means 2.442 ppm SFO. So in reference to the BS, the offset in the sampling clock of the SS has a high probability smaller than 5 ppm if the mobile speed is lower than 240km/h.

6.2.3 Integer CFO Estimation

We assume that the SS knows the correct symbol timing and look at the performance of integer CFO synchronizer first. Since the SS does not know which preamble it receives,

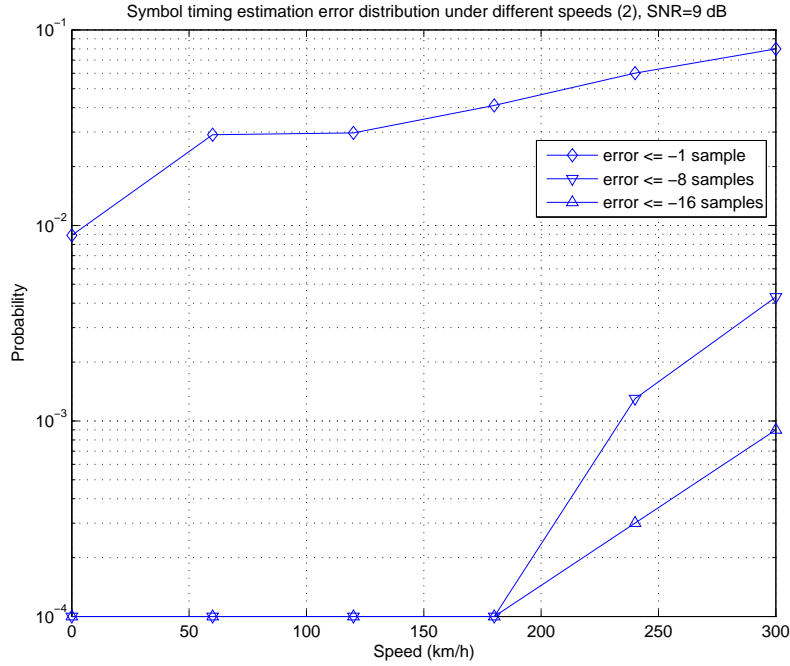


Figure 6.5: Symbol time synchronization error distribution under different speeds (ii).

we use subcarriers on the locations of carrier-set 1 to synchronize the integer CFO, and see whether it can synchronize to the correct value corresponding to the carrier-set. For example, if the transmitted preamble belongs to carrier-set 0 with integer CFO f_I , then the correctly synchronized result must be $f_I - 1$. The synchronized integer CFO must be exactly the same as the true value or the following processing will be meaningless, so we use its error probability to measure its performance.

Figure 6.9 shows the error rate of integer CFO synchronization in the 6-path Rayleigh fading channel. Here we assume the maximum possible integer CFO is $\pm 15\Delta f$. We can see that when SNR is high enough (above 10 dB), the error probability is under 1% no matter what the speed is. If the channel is AWGN and the SNR is above 0 dB, the error rate is always 0 when testing 5000 symbols. The preamble we use is index 33. We shall compare the performance of different preamble sequences in a later subsection where we will see that

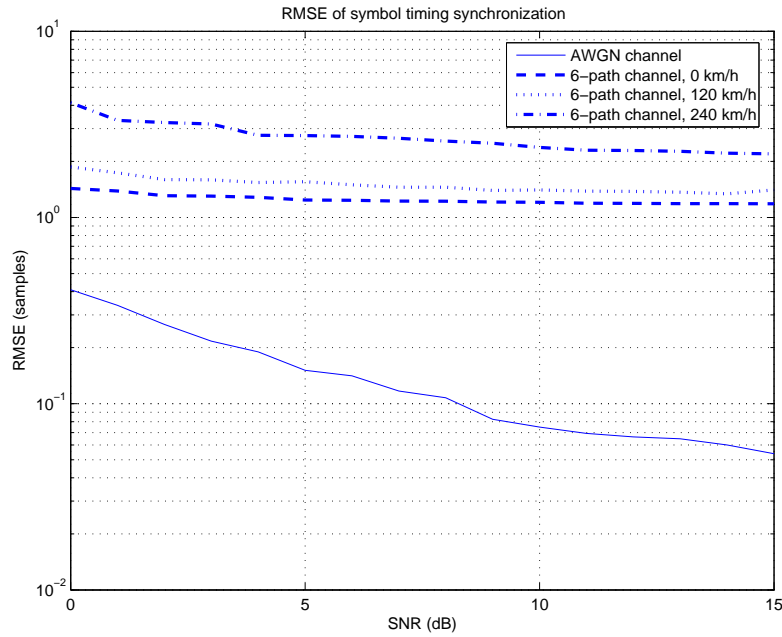


Figure 6.6: RMSE of symbol timing offset synchronization.

different preambles do not differ very greatly in integer CFO estimation performance.

6.2.4 Preamble Index Identification

We assume that the symbol timing and integer CFO synchronizations are perfect, and see whether the SS can identification the correct preamble index by using preamble correlation first. Figure 6.10 shows the error rate of preamble index identification in the 6-path Rayleigh fading channel. We can see that the error probability does not vary greatly even in different speeds or different SNRs, except when the speed is 0 km/h. The error probability is always under 1% in our simulation environments.

If we take imperfect symbol timing and integer CFO synchronization into consideration, Fig. 6.11 shows the probability of either integer CFO or preamble index identification error with imperfect symbol timing synchronization. We would like to see the error rate of integer

CFO and preamble index together because they are done together in practice. In Fig. 6.11 we can see that the error rate is somewhat large. This is because timing offset degrades the performance of preamble index identification very much. The main influence of timing offset is phase rotation, which is not a big problem to power calculation of subcarriers in integer CFO synchronization, but can result in serious performance degradation in preamble correlation. For comparison, Fig. 6.12 shows the case of perfect symbol timing synchronization; the performance is much better than in Fig. 6.11. It is desirable to find a method that can yield better preamble identification performance. However, a 10% error rate may not be unacceptable if the error are statistically independent from one frame to the next, because then it would only need several frames to yield a highly accurate identification.

Figure 6.13 shows a comparison of performance of the two methods we mentioned in 6.1.3. Method I means the intuitive method, and method II is what we use in our design. We see method II performs better if the SNR is larger than 5 dB.

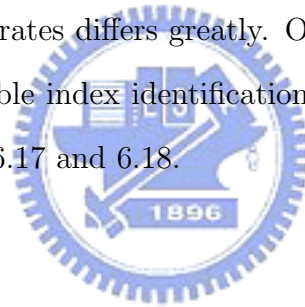
6.2.5 Fine Symbol Timing Estimation

From Figs. 6.11 and 6.12, we see that the performance of timing offset estimation affect the error rate of integer CFO and preamble index estimation severely. So we can improve the error rate by a finer symbol timing synchronization. Use the method we mentioned previously, we try to trace consecutive 10, 25, and 50 symbols, and see their performance. Figure 6.14 shows the error distributions of these three cases. Clearly, the correct rate grows as the length of traced symbol increases, but the drawback, as we already mentioned, is that we must store up to 50 symbols and wait a long time before making a decision. The SNR here is 9 dB.

Figure 6.15 shows the error probability of integer CFO and preamble index estimation with fine timing synchronization, which does help in improving the performance.

6.2.6 Comparison of Preamble Indexes

Since there are 114 different preambles in the IEEE 802.16e, we would like to see some synchronization performance of each of them. Here we test three tasks: symbol timing estimation, integer CFO estimation and preamble index identification, and see the relative difference. The SNR here is 5 dB. From Fig. 6.16 to Fig. 6.18 are the error rates in these three synchronization tasks. Note that the error of symbol timing synchronization means the estimated timing offset is different from the added one. Clearly, the error rate of timing synchronization differs greatly for each preamble. This is because different preamble sequences have different CP correlation values, Fig. 6.19 is an example of two CP correlation values of a good-performance preamble and a bad-performance one in AWGN channel. We see that the preamble which has lower error rate has a sharper correlation curve than the other, and this is the reason why their error rates differs greatly. On the other hand, the error rate in integer CFO estimation or preamble index identification under different preambles are less significantly different, from Figs. 6.17 and 6.18.



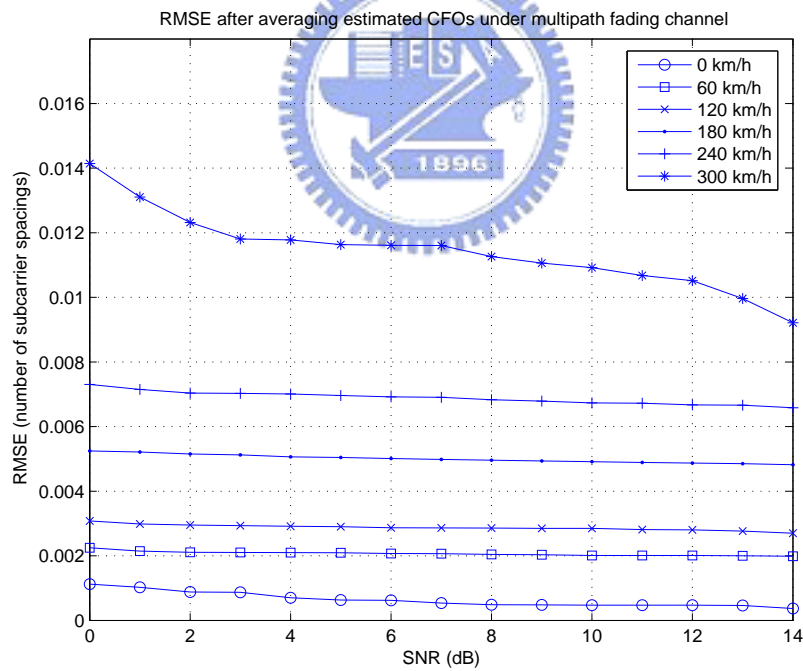
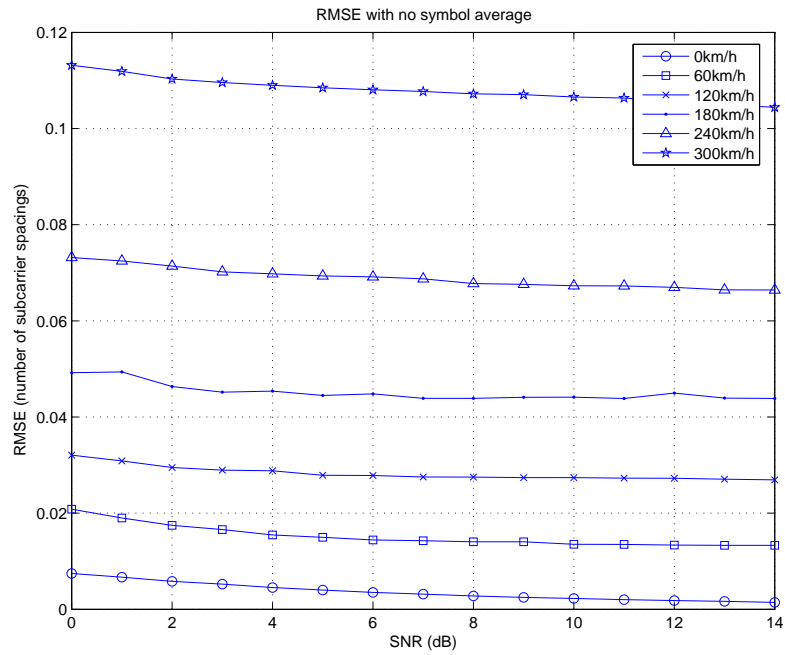


Figure 6.7: RMSE of fractional CFO synchronization.

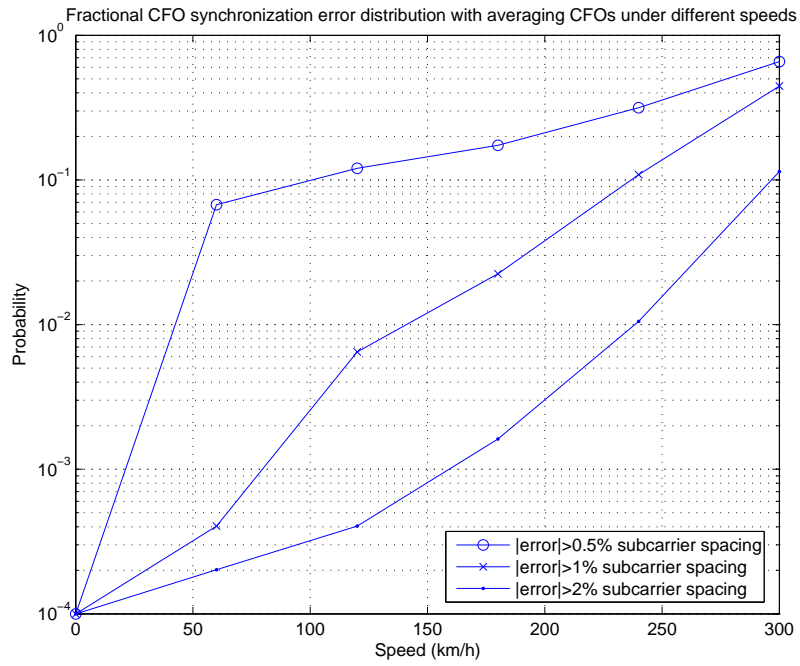


Figure 6.8: Fractional CFO synchronization error distribution under different speeds.

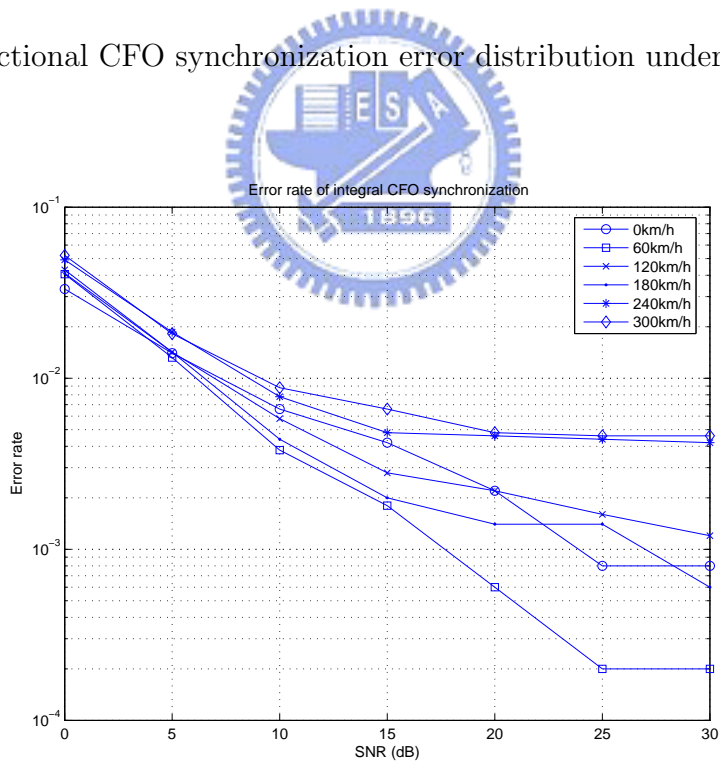


Figure 6.9: Error probability of integer CFO synchronization in multipath fading channel.

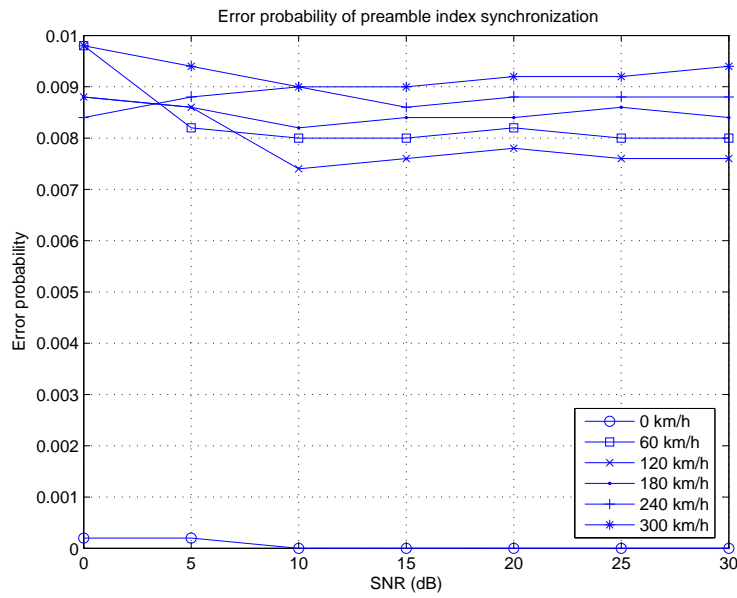


Figure 6.10: Error probability of preamble index synchronization in multipath fading channel.

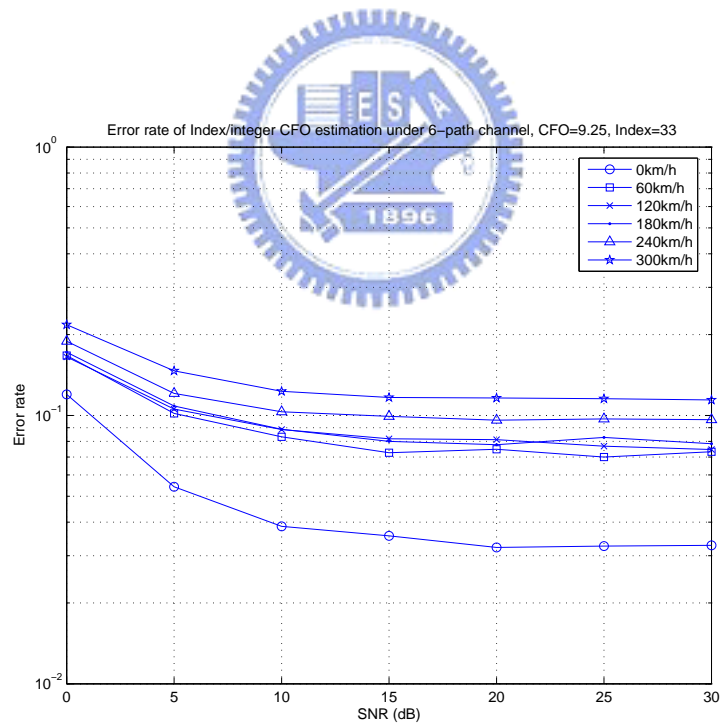


Figure 6.11: Probability of error in either the identified preamble index or the estimated integer CFO.

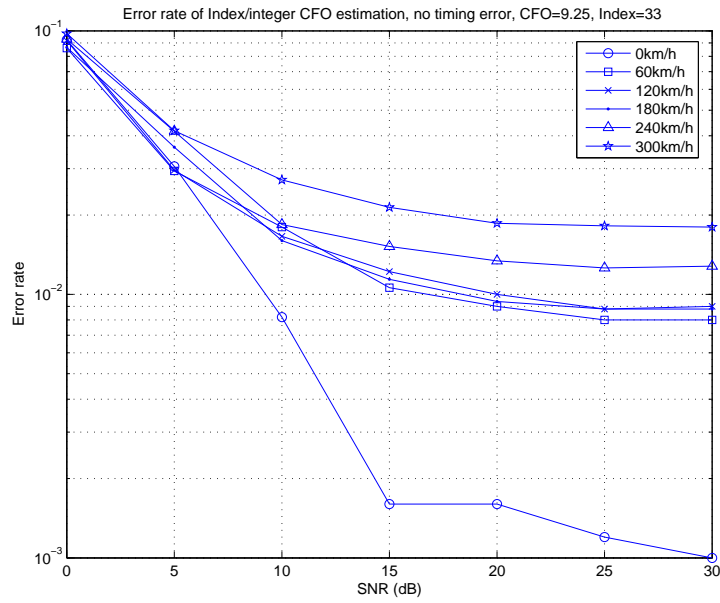


Figure 6.12: Probability of error in either the identified preamble index or the estimated integer CFO, in perfect symbol timing.

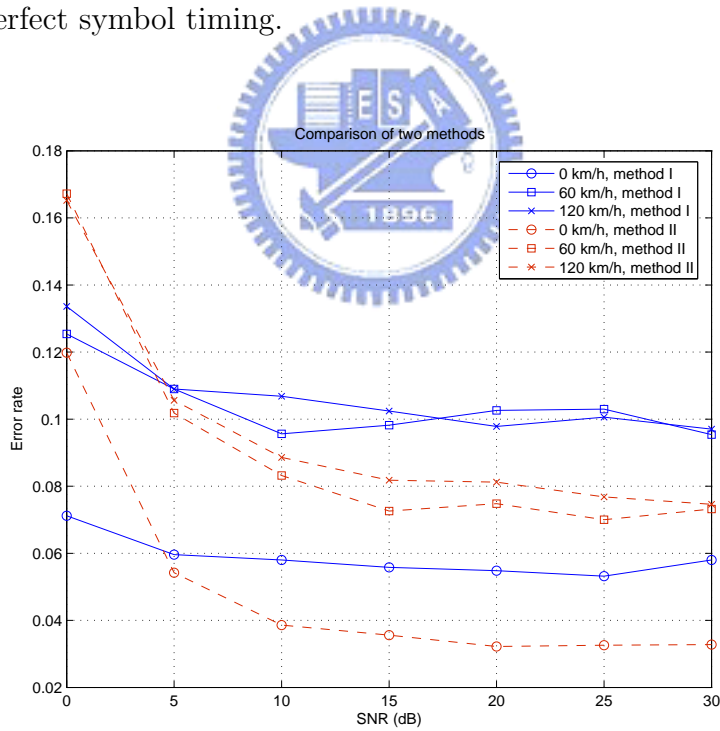


Figure 6.13: Probability of error in either the identified preamble index or the estimated integer CFO of two methods.

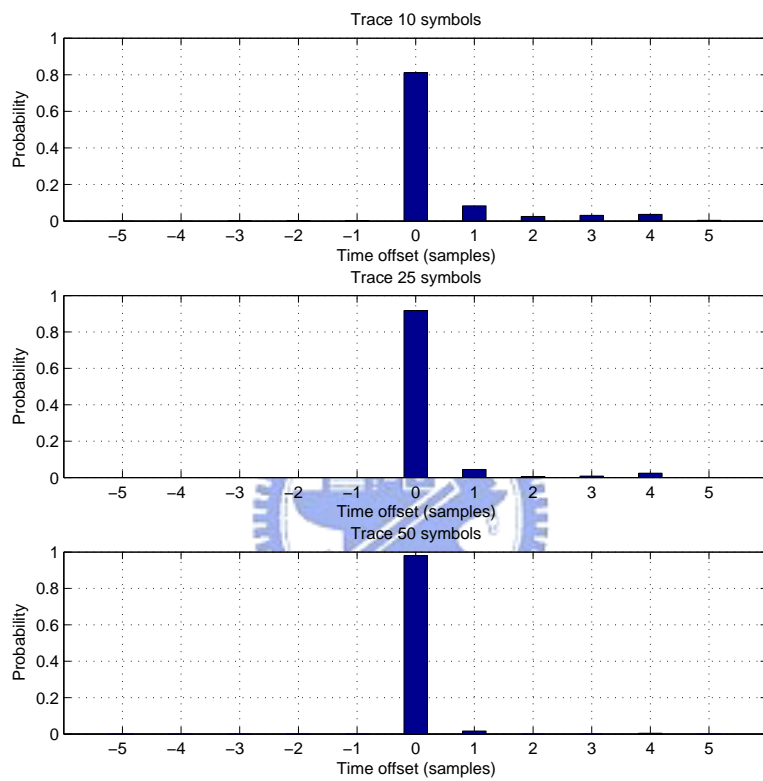


Figure 6.14: Distribution of timing offset estimation errors of fine timing synchronization.

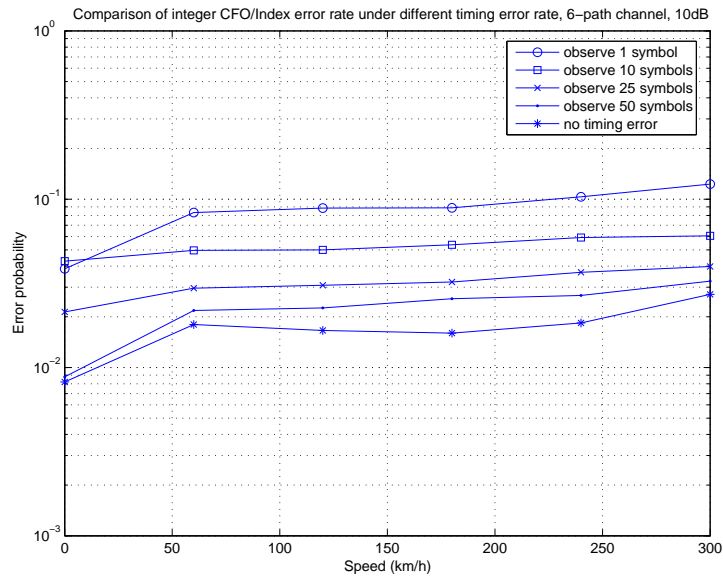


Figure 6.15: Probability of error in either the identified preamble index or the estimated integer CFO with different fine timing synchronization.

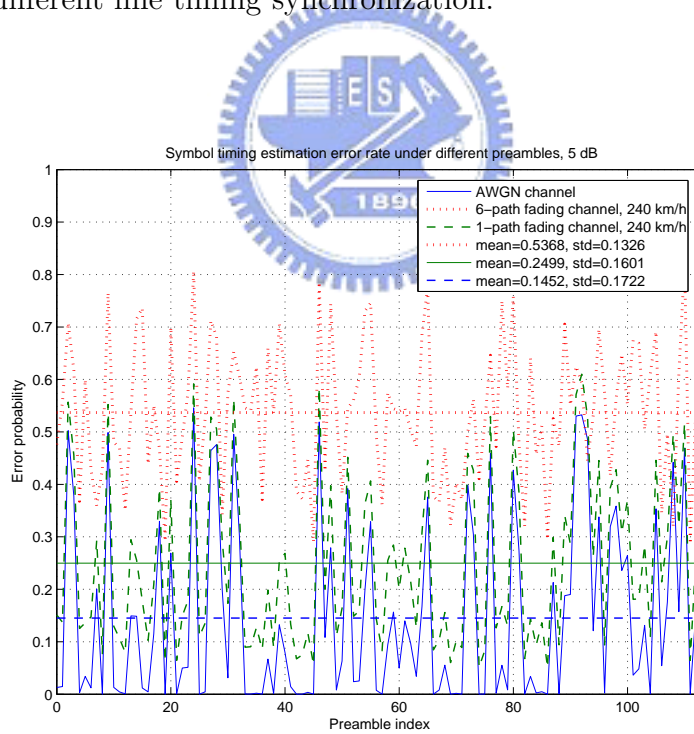


Figure 6.16: Error probability in symbol timing offset estimation with different preambles.

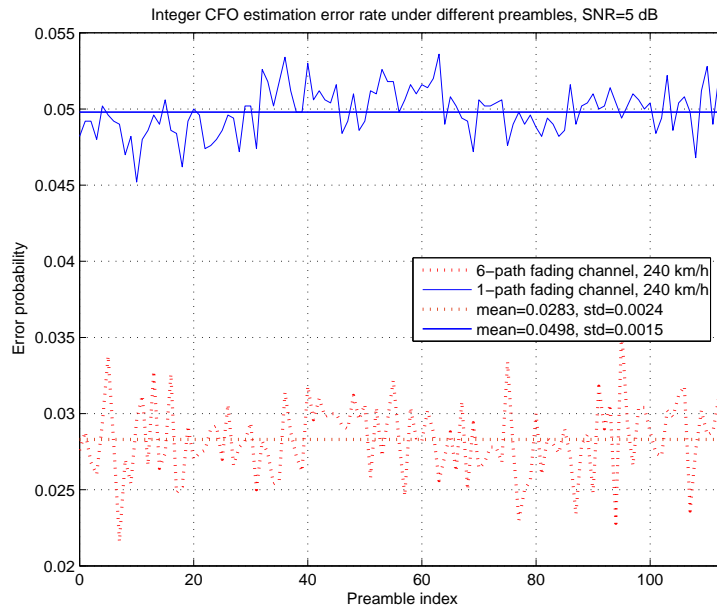


Figure 6.17: Error probability in integer CFO estimation with different preambles.

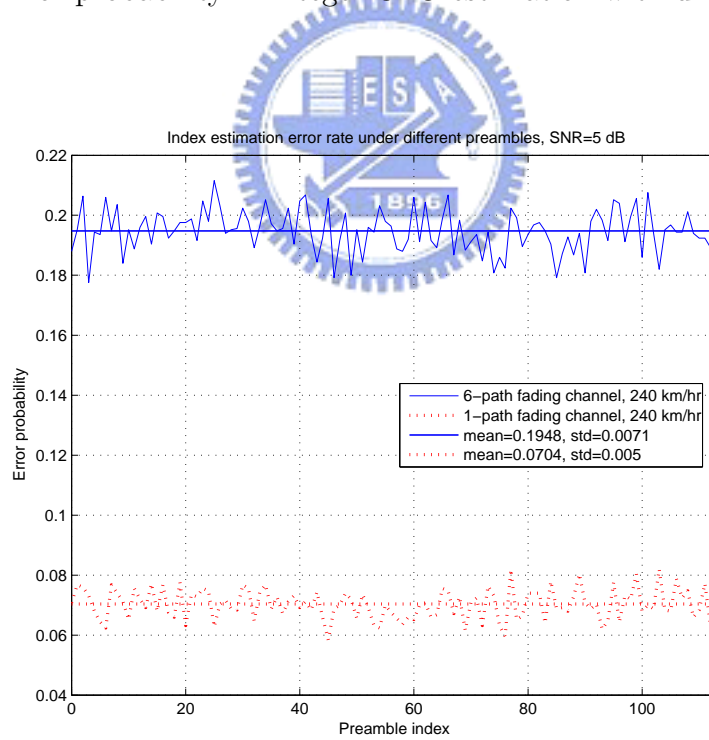


Figure 6.18: Error probability in preamble index identification with different preambles.

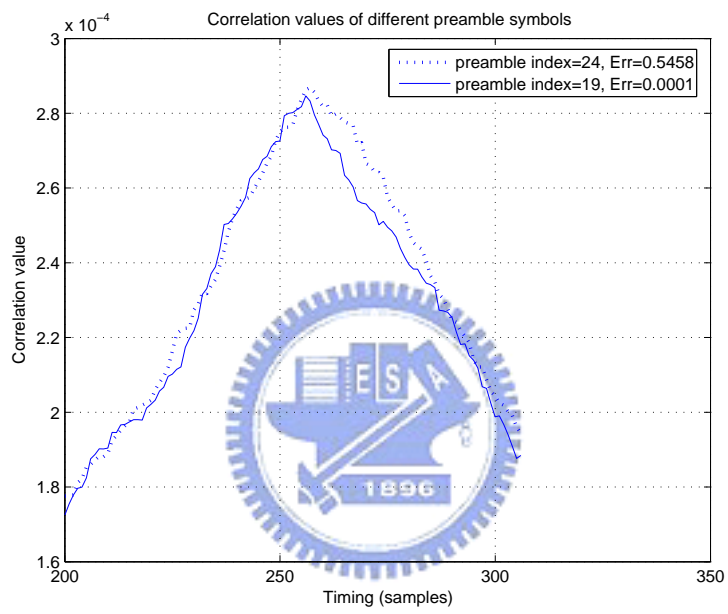


Figure 6.19: CP correlation values of two preambles.

Chapter 7

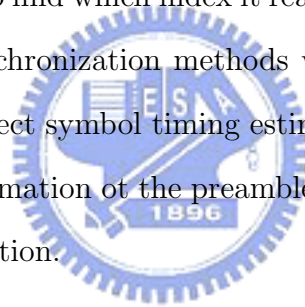
Conclusion and Future Work

7.1 Conclusion

In this thesis, we first discussed the synchronization problems of the IEEE 802.16e OFDM TDD UL system, modified to fixed-point version, and implemented them on TI's C6416 digital signal processor. Second, we also discussed the synchronization problems of the OFDMA TDD DL system, which has different issues from OFDM.

In the OFDM system, we designed a new algorithm to estimate symbol timing by taking advantage of the time-domain structure of the preamble. This algorithm had a well performance (more than 80% correction rate) even in the 60 km/h multipath Rayleigh fading channel. A CP correlation in [21] was used to synchronize fractional CFO of each symbol, and we averaged each estimated values for more accurate results. After averaging, more than 99% of synchronized results can reach the requirement of the specification. The integer CFO and symbol sampling frequency offset synchronization could be ignored. Next, we modified the whole system to fixed-point version, and used some optimization techniques to accelerate each block as fast as we can on TI's DSP. The fixed-point modification degraded the performance of original methods in a range we can accept, and after optimization, every block could achieve real-time rate.

In the OFDMA system, SFO, integer CFO, and preamble index identification were three problems needed to be considered. Since an SS does not know the exact subcarrier values of the preamble, we just viewed a preamble symbol as a regular symbol and estimated timing and fractional CFO by CP correlation in [21]. Like in the OFDM system, we estimated symbol timing offset only by the preamble, but estimated fractional CFO by averaging the estimated values of each symbol for more accuracy. After averaging, more than 99% of results can reach the requirement of the specification if the mobile speed was less than 240 km/h. We developed a method to estimate symbol timing offset more accurately by observing consecutive symbols, this can help to improve the performance of preamble index identification. The integer CFO estimation and preamble index identification were done together. First we estimated the possible integer CFO, then we used this estimated value and its corresponding carrier-set to find which index it really was. If the SNR was reasonable, both error rates of these two synchronization methods were less than 1% even the mobile speed was high. If we took imperfect symbol timing estimation into consideration, the error rate of either the integer CFO estimation or the preamble index identification was about 1% when using the fine timing estimation.



7.2 Future Work

There are several possible extensions for our research:

- Take the effect caused by sampling frequency offset into consideration. This is for a more practical simulation.
- Consider to deal with SFO synchronization in addition, especially in the high mobile speed environment, this can help the performance of BER.
- Run fixed-point simulation for the OFDMA system. We must change it to fixed-point

version so that we can run the system in the hardware.

- Try to use the quasi-periodic time-domain structure of the preamble to estimation symbol timing offset.
- Analyze the effects of different length of guard interval. The guard interval length may effect the performance of fractional CFO and symbol timing.



Bibliography

- [1] A. Ghosh, D. R. Wolter, J. G. Andrews, and R. Chen, “Broadband wireless access with WiMAX/802.16: current performance benchmarks and future potential,” *IEEE Commun. Mag.*, vol. 43, pp. 129–136, Feb. 2005.
- [2] IEEE Std 802.16-2004, *IEEE Standard for Local and Metropolitan Area Networks — Part 16: Air Interface for Fixed Broadband Wireless Access Systems*. New York: IEEE, June 2004.
- [3] IEEE Std 802.16e, *IEEE Standard for Local and Metropolitan Area Networks — Part 16: Air Interface for Fixed Broadband Wireless Access Systems*. New York: IEEE, Feb. 2006.
- [4] H. Yaghoobi, “Scalable OFDMA physical layer in IEEE 802.16 WirelessMAN,” *Intel Technology Journal*, vol. 8, pp. 201–212, Aug 2004.
- [5] M.-T. Lin, “Fixed and mobile wireless communication based on IEEE 802.16a TDD OFDMA: Transmission filtering and synchronization,” M.S. thesis, Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C., June 2003.
- [6] H.-C. Lin, “Study and DSP implementation of IEEE 802.6a TDD OFDMA uplink synchronization,” M.S. thesis, Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C., June 2004.

- [7] C.-C. Tung, "IEEE 802.16a OFDMA TDD uplink transceiver system integration and optimization on DSP platform," M.S. thesis, Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C., June 2005.
- [8] R. van Nee and R. Prasad, *OFDM for Wireless Multimedia Communications*. Boston: Artech House, 2000.
- [9] D. Matiae, "OFDM as a possible modulation technique for multimedia application in the range of mm waves," <http://www.ubicom.tudelft.nl/MMC/Docs/introOFDM.pdf>.
- [10] S. B. Weinstein and P. M. Ebert, "Data transmission by frequency-division multiplexing using the discrete Fourier transform," *IEEE Trans. Commun. Technol.*, vol. COM-19, pp. 628–634, Oct. 1971.
- [11] K. K. Leung, S. Mukherjee, and G. E. Rittenhouse, "Mobility support for IEEE 802.16d wireless network," *IEEE Commun. Society*, vol. 3, pp. 1446–1452, March 2005.
- [12] J. Puthenkulam, and M. Goldhammer, "802.16 overview and coexistence aspects," <http://grouper.ieee.org/groups/802/secmail/ppt00009.ppt>.
- [13] P. Dent, G. E. Bottomley, and T. Croft, "Jakes' fading model revisited," *Electron. Lett.*, vol. 29, no. 13, pp. 1162–1163, June 1993.
- [14] ETSI TR 101 112, "Selection procedures for the choice of radio transmission technologies of the UMTS," *ETSI Techbical Report*, V3.0.2, pp. 38–43, Apr. 1994.
- [15] Texas Instruments, *TMS320C6414T, TMS320C6415T, TMS320C6416T Fixed-Point Digital Signal Processors*. Literature no. SPRS226A, Mar. 2004.
- [16] Texas Instruments, *TMS320C6000 CPU and Instruction Set Reference Guide*. Literature no. SPRU189F, Oct. 2000.

- [17] Texas Instruments, *Code Composer Studio User's Guide*. Literature no. SPRU328B, Feb. 2000.
- [18] Texas Instruments, *TMS320C6000 Programmer's Guide*. Literature no. SPRU198G, Aug. 2002.
- [19] Texas Instrument, *TMS320C6000 Optimizing Compiler User Guide*. Literature no. SPRU187K, Oct. 2002.
- [20] J. J. van de Beek *et al.*, "ML estimation of time and frequency offset in OFDM systems," *IEEE Trans. Signal Processing*, vol. 45, no. 7, pp. 1800–1805, July 1997.
- [21] J.-C. Lin, "Maximum-likelihood frame timing instant and frequency offset estimation for OFDM communication over a fast Rayleigh-fading channel," *IEEE Trans. Vehicular Technology*, vol. 52, no. 4, pp. 1049–1062, July 2003.
- [22] M. Speth *et al.*, "Optimum Receiver Design for Wireless Broad-Band Systems Using OFDM—Part I," *IEEE Trans. Commun.*, vol. 47, pp. 1668–1677, Nov. 1999.

