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 OFDMA 同步技術之研究與數位訊號處理器實現

Research in and DSP Implementation of Synchronization Techniques for IEEE 802.16e

研 究 生: 劉耀鈞

指導教授: 林大衛 博士

中 華 民 國 九 十 六 年 六 月


E
S
A
1896

IEEE 802.16e OFDMA 同步技術之研究與數位訊號處理器實現

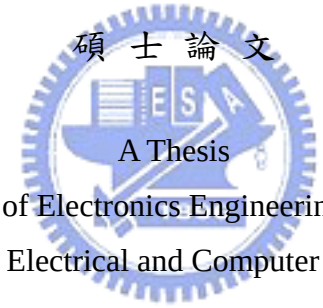
Research in and DSP Implementation of Synchronization Techniques for IEEE 802.16e

研 究 生: 劉耀鈞　　Student: Yao Chun Liu

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



碩 士 論 文

A Thesis

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 of Science

in

Electronics Engineering

June 2007

Hsinchu, Taiwan, Republic of China

中 華 民 國 九 十 六 年 六 月


E
S
A
1896

IEEE 802.16e OFDMA 同步技術之研究與數位訊號處理器實現

研究生：劉耀鈞　　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

摘要

本篇論文介紹 IEEE 802.16e 正交分頻多工存取(OFDMA)裡，同步的問題、演算法、以及實做方面的議題。

當一個行動電話在一開始要進入網路的時候，我們必須做起始的同步。在起始的同步中，包含了符碼時間偏移、載波偏移和前置符元序號(preamble index)需要同步。我們使用循環字首(cyclic prefix)的相關性(correlation)及在第一個下行次訊框(subframe)裡的資訊來估計出較準確的符碼時間偏移和小數部分載波偏移。之後我們在頻域上用補償之後的同步碼來聯合估計出整數載波偏移和前置符元序號。我們利用前置符元序列和前置符元的特性來做估計，另外我們也介紹了一些不同複雜度的方法。

在之後的次訊框中，行動電話只需要做到追蹤符碼時間偏移和小數部分載波偏移。我們再次使用循環字首的相關性並在每個符元間利用指數平均來求得較準確的結果。另外，因為我們在非起始的同步中，我們已經知道前置符元序號，因此我們可利用這個資訊來估計符碼時間偏移。這個方法主要是利用前置符元序列

之間的準正交性的特性來估計。我們首先是用浮點數運算來驗證，並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 120 km/h，並觀察其結果。

最後，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，但其結果依然可以接受。經過最佳化之後，同步的工作都能達到即時處理(real time)的要求。

Research in and DSP Implementation of Synchronization Techniques for IEEE 802.16e

Student: Yao Chun Liu Advisor: Dr. David W. Lin

Department of Electronics Engineering

Institute of Electronics

National Chiao Tung University

Abstract

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

In DL synchronization, the (mobile station) MS receiver needs to perform initial synchronization upon its initial entrance to the network. There are timing offset, carrier frequency offset (CFO), and preamble index needed to be estimated during initial synchronization. We use the information of the first DL subframe to estimate the more accurate timing and fractional CFO by cyclic prefix (CP) correlation. Then, we consider the joint detection of integer CFO and preamble index using the compensated preamble in the frequency domain. The preamble sequences and the feature of preamble symbol are exploited and a number of detection methods of different complexity are introduced.

In subsequent subframe, the MS only needs to track the timing offset and fractional CFO. We use CP correlation with exponential average over the symbols in the subframe to obtain a more accurate estimation. Besides, we also afford a data-aided method to estimate the symbol timing since we already know the preamble index during normal synchronization. This method exploits the quasi-orthogonality among the preamble sequences. We verify our system in floating-point computation,

and simulate our system in both AWGN and multipath Rayleigh fading channel, which the speed is as high as 120 km/h, and see the performance.

In the end, 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. After DSP optimization, the synchronization tasks can achieve the real-time requirement.

誌謝

這篇論文能夠順利完成，首先要感謝的人是我的指導教授林大衛老師，在兩年的研究所生涯當中，由於老師的細心指導及在專業領域的博學精深，使得我學習到不少研究的精神與方法。

此外，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。感謝洪崑健學長、吳俊榮學長和林鴻志學長給予我在研究過程上的指導與建議，還有柏昇、政達、介遠、依翎、凱庭、志岡、錫祺、怡禎等同學，因為能和你們共同討論、分享求學的經驗及一路上的相互扶持，讓這兩年的研究生涯充滿歡樂與回憶。

最後，我要感謝我的家人們，感謝他們一直都在背後支持我，在求學過程中總是不斷的鼓勵我，是我精神上最大的支柱。

在此，將此篇論文獻給所有幫助過我，陪伴我走過這一段歲月的師長，同學，朋友與家人。

劉耀鈞

民國九十六年六月 於新竹


E
S
A
1896

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The IEEE 802.16e, of which a subset is commonly known as Mobile WiMAX (Worldwide Interoperability of Microwave Access), is a broadband wireless access (BWA) system which has drawn much attention these days. IEEE 802.16e is originally suggested as an enhancement version of IEEE Std. 802.16-2004 to provide mobile station (MS) with mobility at vehicular speed. Therefore, it specifies BWA systems for both fixed and mobile MS simultaneously [1],[2].

One of the most promising modes in the IEEE 802.16e standard is the Orthogonal Frequency Division Multiple Access (OFDMA) mode, which is generally accepted as a performance efficient multiple access scheme. The Mobile WiMAX system also utilizes the bandwidth scalability, where the FFT size typically increases with the bandwidth. In this thesis, we consider the IEEE 802.16e WirelessMAN OFDMA system with a time-division duplex (TDD) mode, where downlink (DL) and uplink (UL) transmissions are time multiplexed in each TDD frame.

Our study can be divided into two parts. The first part is the synchronization techniques for IEEE 802.16e OFDMA. Synchronization in OFDMA system involves frequency and timing recovery. For operation under the current IEEE 802.16e OFDMA TDD specifications, the identification of the preamble index may also be considered part of the synchronization process. Therefore, in the present study we consider carrier frequency synchronization, timing synchronization, and preamble index identification, for both fixed and mobile communication channels.

The second part is the digital signal processor (DSP) implementation of the synchronization techniques. We implement them on Texas Instrument (TI)'s DSP. Moreover, we employ various optimization techniques to accelerate the execution speed of the programs in the DSP implementation.

This thesis is organized as follows. We first introduce the IEEE 802.16e Wireless-MAN OFDMA standard in chapter 2. Chapter 3 introduces the DSP implementation platform. In chapter 4, the system parameters and channel environments are discussed. The transmission filtering is also analyzed in chapter 4. We analyze the synchronization problems and present some solutions in chapter 5. Chapter 6 discusses the DSP optimization methods and presents the optimization results. Finally, the conclusion is given in chapter 7, where we also point out some potential future work.

Chapter 2

Overview of the IEEE 802.16e OFDMA Standard

In this chapter, we first introduce some basic concepts regarding OFDM and OFDMA. Then we give an overview of the IEEE 802.16e OFDMA standard. For the sake of simplicity, we only introduce the specifications that have use in our study. Other specifications like channel coding, MAP messages, transmit diversity, etc., are not our concern and are ignored in this introduction. For more details we refer the readers to [1] and [2], from which we take much of the material in this chapter.

2.1 Introduction to OFDM [3]

OFDM is a special case of multicarrier transmission technique, where a single datastream is transmitted over a number of subcarriers a lower rates. The concept of OFDM is to use parallel data transmission and frequency multiplexing. It divides the available spectrum into narrower subcarrier bands, and each subcarrier only transmits a portion of the total information.

The orthogonality of OFDM constitutes one major difference from the classical parallel data system, making its use of the available spectrum more efficient. Figure 2.1 shows the difference. As we can see, the subcarriers in an OFDM symbol can be arranged so that the sideband of each subcarrier overlaps but the received symbols still live with-

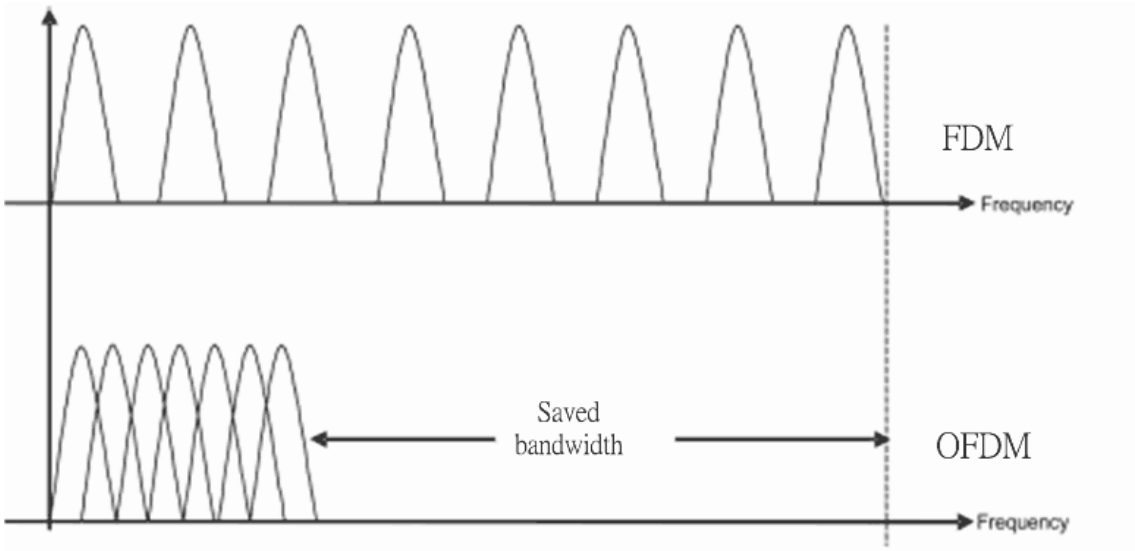


Figure 2.1: Bandwidth efficiency comparison of traditional FDM and OFDM systems.

out adjacent-carrier interference. This can be accomplished by using the discrete Fourier transform (DFT) proposed by Weinstein and Ebert in 1971 [4]. The complexity of DFT, however, is too expensive. Fortunately, modern advances in very-large-scale integration (VLSI) make it possible to use the fast Fourier transform (FFT) for a more efficient implementation of the DFT. The complexity is reduced from N^2 in DFT to $N \log_2 N$ in FFT.

One of the main reasons to use OFDM is to increase the robustness against frequency selective fading or narrowband interference. An OFDM system may encode data using forward error correction (FEC) coding and distribute them across several subcarriers. If frequency-selective fading causes errors in the reception of few subcarriers, the data bits in those subcarriers are recovered through FEC.

Another reason for choosing OFDM is its natural immunity to multipath. For a given overall data rate, the increasing number of carriers due to overlapping can reduce the data rate that each individual carriers must convey, and hence lengthen the symbol period. This means that the inter symbol interference (ISI) affects a smaller percentage of each symbol. Therefore complex equalization is normally not needed in the receiver.

In order to eliminate the ISI completely, a guard time (or guard interval, or cyclic prefix) is inserted. The guard time is chosen larger than the expected delay spread, such that

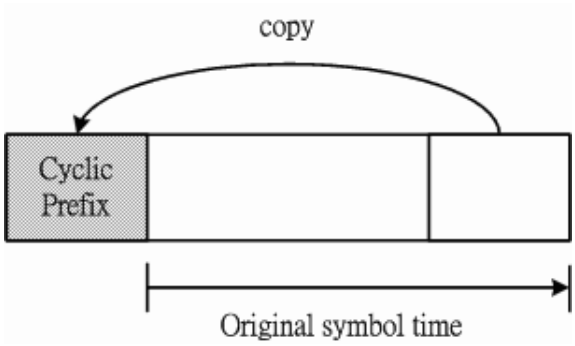


Figure 2.2: The use of cyclic prefix (from [5]).

multipath components from one symbol cannot interfere with the next symbol. However, if we insert zeros within the guard interval, the orthogonality among subcarriers will no longer exist, which causes serious intercarrier interference (ICI). To preserve the orthogonality among the subcarriers and eliminate ICI, the OFDM symbol should be cyclically extended in the guard time rather than just extended with zero. Figure 2.2 shows how to add cyclic prefix in front of an OFDM symbol. Hence if the maximum multipath delay is smaller than the guard time, there will not be ISI or ICI.

Finally, the advantages and disadvantages 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.

2.2 Introduction to OFDMA

OFDMA is a multiple access method based on OFDM signaling that allows simultaneous transmissions to and from multiple users along with the other advantages of OFDM. In

Table 2.1: OFDM Advantages and Disadvantages

Advantages	Disadvantages
Bandwidth efficiency	Sensitive to frequency offset
Immunity to multipath effect	Sensitive to timing offset
Robust against narrowband interference	Sensitive to phase noise
	Large peak-to-average power ratio

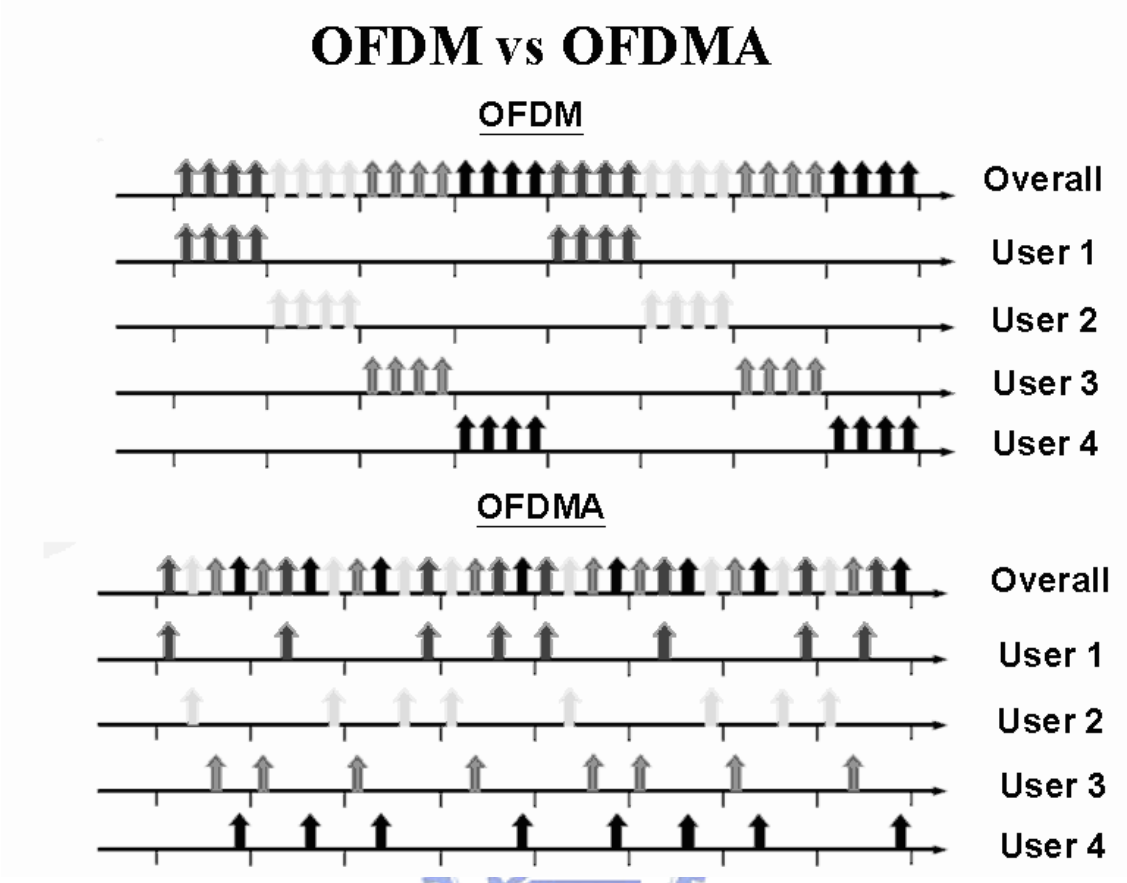


Figure 2.3: Comparison of subcarrier allocatins in OFDM and OFDMA (from [6]).

OFDM, a channel is divided into carriers which is 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 this way, the bandwidth can be allocated dynamically to the users according to their needs. A simple comparison of the subcarrier allocation of OFDM and OFDMA is shown in Fig. 2.3.

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 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 active sub-carriers.

Fig. 2.4 shows an example subcarrier allocation in an OFDMA symbol. The frequency

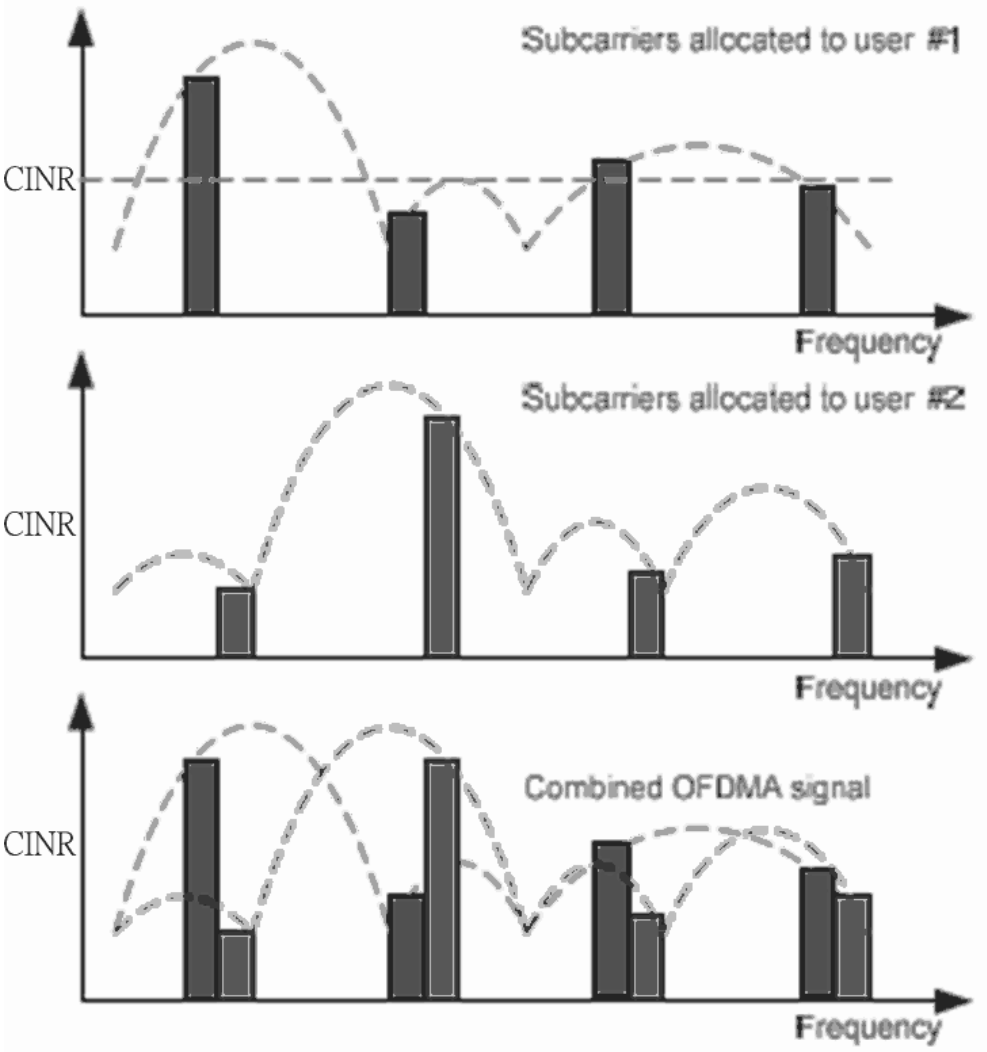


Figure 2.4: Subcarrier allocation in an OFDMA symbol (from [7]).

response of a typical broadband wireless channel is also depicted. In this example, the deep-fading condition and narrowband interference are considered. In the top plot, we see that when the channel is in deep fade, the subcarriers are not sufficiently energy efficient to carry information. These wasted subcarriers can be utilized by there uses in OFDMA, thus achieving higher efficiency and capacity. Very few, if any, subcarriers are likely to be wasted in OFDMA, since no particular subcarrier is likely to be bad for all users.

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.

2.3 Introduction to IEEE 802.16e

Since the publication of the IEEE 802.16 standard for fixed broadband wireless access in 2001, a number of revision and amendments have taken place. Like other IEEE 802 standards, the 802.16 standards are primarily concerned with physical (PHY) layer and medium access control (MAC) layer functionalities. The idea originally was to provide broadband wireless access to buildings through external antennas communicating with radio base stations (BSs).

To overcome the disadvantage of the line-of-sight (LOS) requirement between transmitters and receivers in the 802.16 standard, the 802.16a standard was approved in 2003 to support nonline-of-sight (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 specifications were devised primarily for fixed wireless users. The 802.16e task group 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 Febuary 2006. It specifies four air interfaces: WirelessMAN-SC PHY, WirelessMAN-SCa PHY, WirelessMAN-OFDM PHY, and WirelessMAN-OFDMA PHY. This study is concerned with WirelessMAN-OFDMA PHY in a mobile communication environment.

Some glossary we will often use in the following is as follows. The direction of transmission from the base station (BS) to the subscriber station (SS) is called downlink (DL), and the opposite direction is uplink (UL). The SS is considered synonymous as the mobile station (MS). It is sometimes termed the user. The BS is a generalized equipment set providing connectivity, management, and control of the SS.

2.3.1 OFDMA Basic Terms

In the OFDMA mode, the active subcarriers are divided into subsets of subcarriers, where each subset is termed a subchannel. The subcarriers forming one subchannel may, but need not be, adjacent. The concept is shown in Fig. 2.5.

Three basic types subchannel organization are defined: partial usage of subchannels (PUSC), full usage of subchannels (FUSC), and adaptive modulation and coding (AMC); among which the PUSC is mandatory and the other two are optional. In PUSC DL, the entire channel bandwidth is divided into three segments to be used separately. The FUSC is employed only in the DL and it uses the full set of available subcarriers so as to maximize the throughput.

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 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.
- For downlink FUSC and downlink optional FUSC using the distributed subcarrier permutation, one slot is one subchannel by one OFDMA symbol.

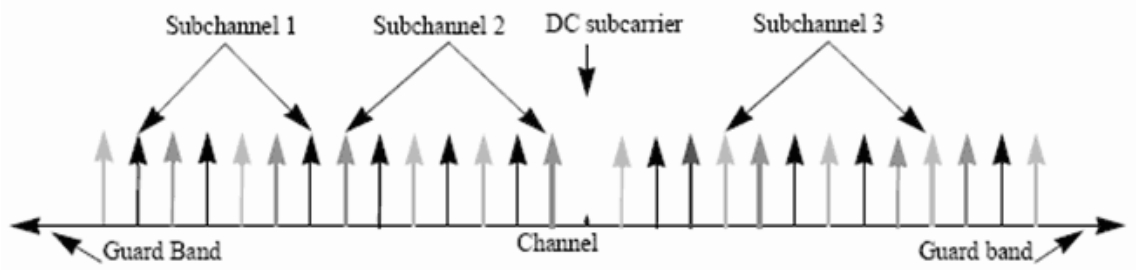


Figure 2.5: OFDMA frequency description (3-channel schematic example, from [1]).

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. 2.6.

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

A 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. The concept of permutation zone will be further elaborate later.

2.3.2 OFDMA Symbol Parameters

Some OFDMA symbol parameters are listed below.

- BW: Nominal channel bandwidth.
- N_{used}: Number of used subcarriers.

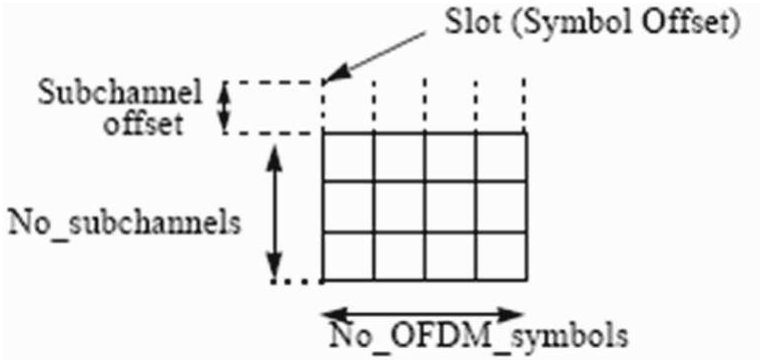


Figure 2.6: Example of the data region which defines the OFDMA allocation (from [1]).

- n: Sampling factor. This parameter, in conjunction with BW and N_{used}, determines the subcarrier spacing and the useful symbol time.
- G: Ratio of cyclic prefix (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}.

2.3.3 Scalable OFDMA [7]

One feature of the IEEE 802.16e OFDMA is the selectable FFT size, from 128 to 2048 in multiples of 2, excluding 256 to be used with OFDM. This has been termed scalable OFDMA (S-OFDMA). One use of S-OFDMA is that if the channel bandwidths are allocated based on integer power of 2 times a base bandwidth, then one may consider making the FFT size proportional to the allocated bandwidth so that all systems are based on the same subcarrier spacing and the same OFDMA symbol duration, which may simplify system design. For example, Table 2.2 lists some S-OFDMA parameters proposed by the WiMAX Forum [8]. S-OFDMA supports a wide range of bandwidth to flexibly address the need for various spectrum allocation and usage model requirements.

When designing OFDMA wireless systems the optimal choice of the number of subcarriers per channel bandwidth is a tradeoff between protection against multipath, Doppler shift, and design cost/coplexity. Increasing the number of subcarriers leads to better immunity to the ISI caused by multipath; on the other hand it increases the cost and complexity of the system (it leads to higher requirements for signal processing power and

Table 2.2: S–OFDMA Parameters Proposed by WiMAX Forum

Parameters	Values			
System Channel Bandwidth (MHz)	1.25	5	10	20
Sampling Frequency (MHz)	1.4	5.6	11.2	22.4
FFT Size	128	512	1024	2048
Subcarrier Spacing (Δf)	10.94 kHz			
Useful Symbol Time (T_b=1/Δf)	91.4 μsec			
Guard Time (T_g=T_b/8)	11.4 μsec			
OFDMA Symbol Duration (T_s=T_b+T_g)	102.9 μsec			

power amplifiers with the capability of handling higher peak-to-average power ratios). Having more subcarriers also results in narrower subcarrier spacing and therefore the system becomes more sensitive to Doppler shift and phase noise. Calculations show that the optimum tradeoff for mobile systems is achieved when subcarrier spacing is about 11 kHz [9] .

2.4 OFDMA Frame Structure

Duplexing Modes

In licensed bands, the duplexing method shall be either frequency-division duplex (FDD) or time-division duplex(TDD). FDD SSs may be half-duplex FDD (H-FDD). In license-exempt bands, the duplexing method shall be TDD.

Point-to-Multipoint (PMP) Frame Structure

When implementing a TDD system, the frame is composed of BS and SS transmissions. Figure 2.7 shows an example. Each frame in the downlink transmission begins with a preamble followed by a DL transmission period and an UL transmission period. In each frame, time gaps, denoted transmit/receive transition gap (TTG) and receive/transmit gap (RTG), are between the downlink and uplink subframes and at the end of each frame,

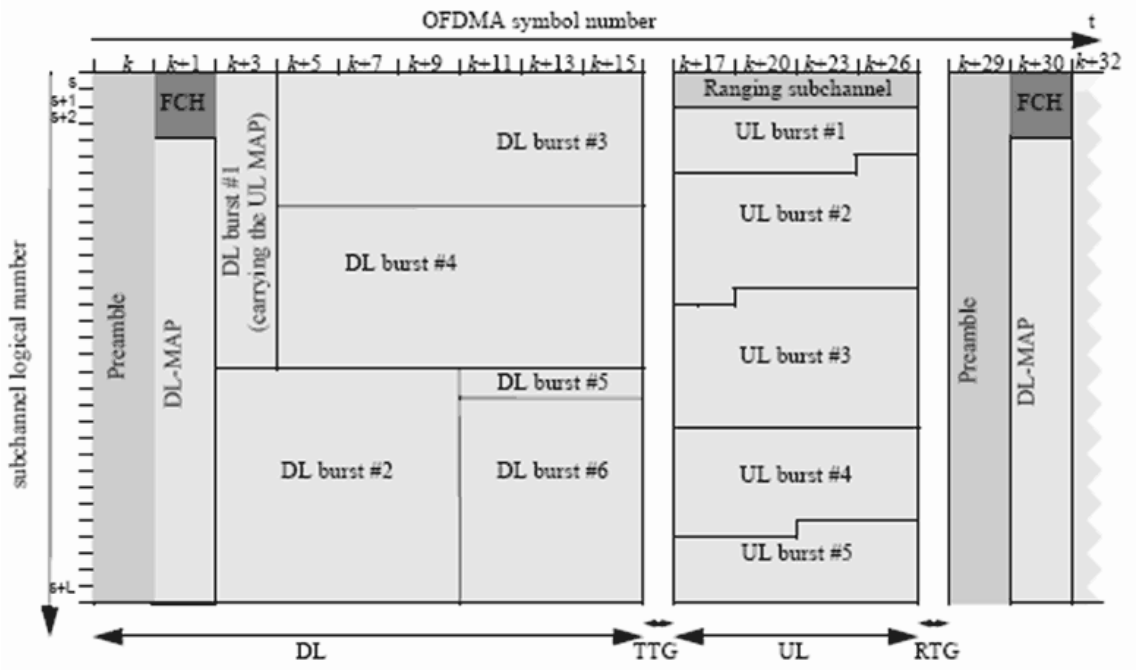


Figure 2.7: Example of an OFDMA frame (with only mandatory zone) in TDD mode (from [2]).

respectively placed. They allow transitions between transmission and reception functions.

Subchannel allocation in the downlink may be performed with PUSC where some of the subchannels are allocated to the transmitter or FUSC where all subchannels are allocated to the transmitter. The downlink frame shall start in PUSC mode with no transmit diversity. The 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 which 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

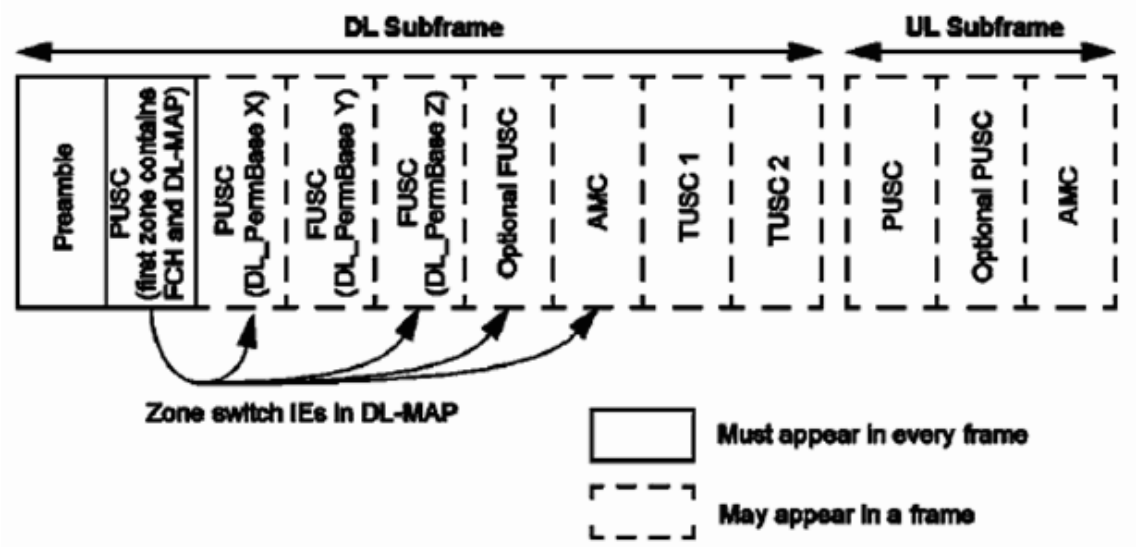


Figure 2.8: Illustration of OFDMA with multiple zones (from [2]).

the DL-Map. Figure 2.8 depicts an OFDMA frame with multiple zones.

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 groups #0, #2, and #4, respectively. Figure 2.9 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

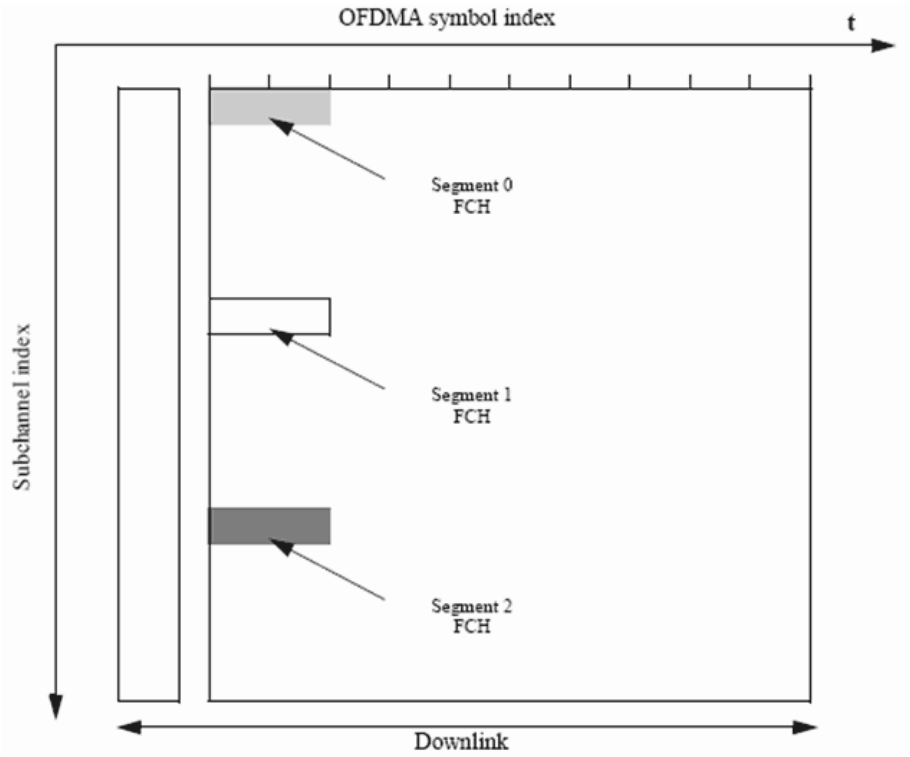


Figure 2.9: FCH subchannel allocation for all 3 segments (from [1]).

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. Figure 2.10 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.

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,

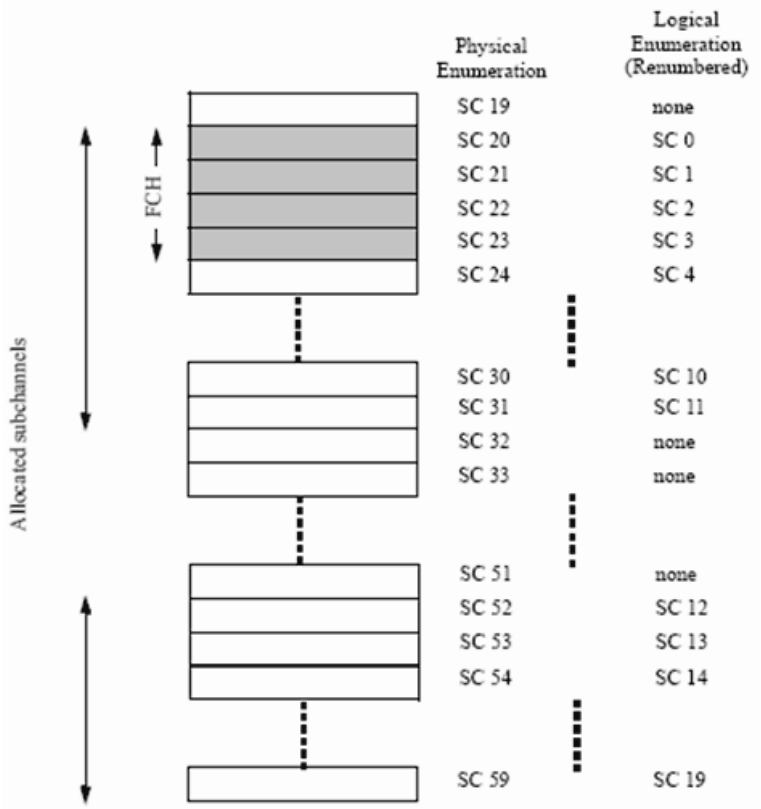


Figure 2.10: Example of DL renumbering the allocated subchannels for segment 1 in PUSC (from [1]).

and the respective slots may be used for transmission of MAP and data of other segments.

2.5 OFDMA Subcarrier Allocation

As mentioned, the OFDMA PHY defines four scalable FFT sizes: 2048, 1024, 512, and 128. For convenience, here we only take the 1024-FFT OFDMA subcarrier allocation for introduction. The subcarriers are divided into three types: null (guard band and DC), pilot, and data. Subtracting the guard tones from N_{FFT}, one obtains the set of "used" subcarriers N_{used}. 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 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 downlink, there is one set of common pilot subcarriers in each major group, but in PUSC uplink, each subchannel contains its own pilot subcarriers.

2.5.1 Downlink

Preamble

The first symbol of the downlink transmission is the preamble. There are three types of preamble carrier-sets, 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

$$PreambleCarrierSet_n = n + 3 \cdot k \tag{2.1}$$

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 0–283.

For the preamble symbol there are 86 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 manner that segment i uses preamble carrier-set i, where $i = 0, 1, 2$. Therefore, each segment eventually modulates each third subcarrier. In the case of segment 0, the DC carrier will be zeroed and the corresponding PN number will be discarded.

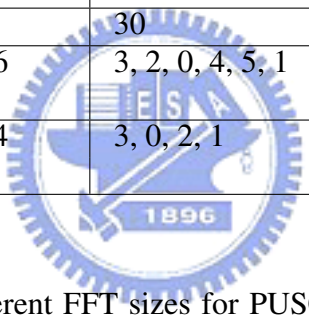
The 114 different PN series modulating the preamble carrier-set are defined in Table 309 of [1] for the 1024-FFT mode. The series modulated depends on the segment used and the IDcell parameter.

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 [2] summarizes the parameters

Table 2.3: 1024-FFT OFDMA DL Carrier Allocation for PUSC

Parameter	Value	Comments
Number of DC subcarriers	1	Index 512
Number of guard subcarriers, left	92	
Number of guard subcarriers, right	91	
Number of used subcarriers, N_{used}	841	
Renumbering sequence	6, 48, 37, 21, 31, 40, 42, 56, 32, 47, 30, 33, 54, 18, 10, 15, 50, 51, 58, 46, 23, 45, 16, 57, 39, 35, 7, 55, 25, 59, 53, 11, 22, 38, 28, 19, 17, 3, 27, 12, 29, 26, 5, 41, 49, 44, 9, 8, 1, 13, 36, 14, 43, 2, 20, 24, 52,4, 34, 0	Used to renumber clusters before allocation to subchannels.
Number of subcarriers per cluster	14	
Number of clusters	60	
Number of data subcarriers in each symbol per subchannel	24	
Number of subchannels	30	
Basic permutation sequence 6 (for 6 subchannels)	3, 2, 0, 4, 5, 1	
Basic permutation sequence 4 (for 4 subchannels)	3, 0, 2, 1	



of the symbol structure of different FFT sizes for PUSC mode. Here we only take the 1024-FFT OFDMA downlink carrier allocation for example, which is summarized in Table 2.3. Fig. 2.11 depicts the DL cluster structure.

Downlink Subchannels Subcarrier Allocation in PUSC

The subcarrier 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:

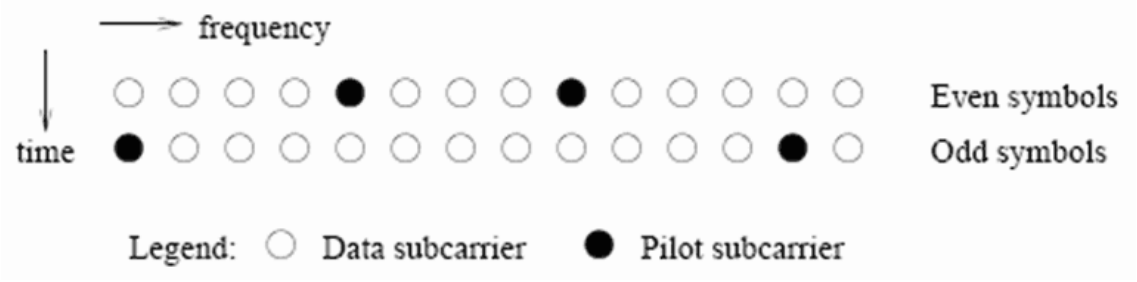


Figure 2.11: DL cluster structure (from[10]).

$$LogicalCluster = \begin{cases} RenumberingSequence(PhysicalCluster), & \text{first DL zone, or Use All SC indicator} \\ & \text{= 0 in STC\_DL\_Zone\_IE,} \\ RenumberingSequence((PhysicalCluster)+ & \text{otherwise.} \\ \quad 13 \cdot DL\_PermBase) mod N_{clusters}, & \end{cases} \tag{2.2}$$

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 renumbering 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 size. For FFT size = 1024, dividing the clusters into six major groups. Group 0 includes clusters 0–11, group 1 clusters 12–19, group 2 clusters 20–31, group 3 clusters 32–39, group 4 clusters 40–51, and group 5 clusters 52–59. These groups may be allocated to segments; if a segment is 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, and then partitioning all remaining data carriers 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 subchan-

nel, 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 parameters vary with FFT sizes. For FFT size = 1024, use the parameters from Table 2.3, with basic permutation sequence 6 for even numbered major groups and basic permutation sequence 4 for odd numbered major groups, to partition the subcarriers into subchannels containing 24 data subcarriers in each symbol. The exact partitioning into subchannels is according to the permutation formula:

$$subcarrier(k,s) = N_{subchannels} \cdot n_k + \{p_s[n_k \bmod N_{subchannels}] + DL\_PermBase\} \bmod N_{subcahnnels} \quad (2.3)$$

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

2.5.2 Uplink

The UL follows the DL model, therefore it also supports up to three segments. The UL supports 35 subchannels where each transmission uses 48 data subcarriers as the

minimal block of processing. Each new transmission for the uplink commences with the parameters as given in Table 2.4.

Symbol Structure for Subchannel (PUSC)

A slot in the uplink is composed of three OFDMA symbols and one subchannel. Within each slot, there are 48 data subcarriers and 24 fixed-location pilots. The subchannel is constructed from six uplink tiles, each tile has four successive active subcarriers and its configuration is illustrated in Fig. 2.12.

Partitioning of Subcarriers into Subchannels in the Uplink

The usable subcarriers in the allocated frequency band shall be divided into N_{tiles} physical tiles as defined in Fig. 2.12 with parameters from Table 2.4. The allocation of physical tiles to logical tiles in subchannels is performed in the following manner:

$$Tiles(s,n) = N_{subchannels} \cdot n + \{P_t[(s+n) \bmod N_{subchannels}] + UL\_PermBase\} \bmod N_{subcahnnels} \quad (2.4)$$

Table 2.4: 1024-FFT OFDMA UL Carrier Allocation for PUSC

Parameter	Value	Comments
Number of DC subcarriers	1	Index 512
N_{used}	841	
Guard subcarriers: left, right	92,91	
TilePermutation	11, 19, 12, 32, 33, 9, 30, 7, 4, 2, 13, 8, 17, 23, 27, 5, 15, 34, 22, 14, 21, 1, 0, 24, 3, 26, 29, 31, 20, 25, 16, 10, 6, 28, 18	used to allocate tiles to subchannels
$N_{subchannels}$	35	
N_{tiles}	210	
Tile per subchannel	6	

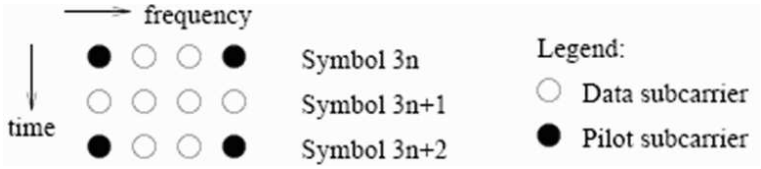


Figure 2.12: Description of an UL tile (from [10]).

where:

$Tiles(s,n)$	is the physical tile index in the FFT with tiles being ordered consecutively from the most negative to the most positive used subcarrier (0 is the starting tile index),
n	is the tile index 0,...,5 in a subchannel,
P_t	is the tile permutation,
$N_{subchannels}$	is the number of subchannels,
s	is the subchannel number in the range $\{0,...,N_{subchannels}-1\}$,
$UL\_PermBase$	is an integer value in the range $0...69$, which is assigned by a management entity.

After mapping the physical tiles in the FFT to logical tiles for each subchannel, the data subcarriers per slot are enumerated by the following process:

1. After allocating the pilot carriers within each tile, indexing of the data subcarriers within each slot is performed starting from the first symbol at the lowest indexed subcarrier of the lowest indexed tile and continuing in an ascending manner through the subcarriers in the same symbol, then going to the next symbol at the lowest indexed data subcarrier, and so on. Data subcarriers shall be indexed from 0 to 47.

2. The mapping of data onto the subcarriers will follow (2.5), which calculates the subcarrier index (as assigned in item 1) to which the data constellation point is to be mapped:

$$Subcarriers(n,s) = (n + 13 \cdot s) \bmod N_{subcarriers} \tag{2.5}$$

where:

$Subcarriers(n, s)$	is the permutated subcarrier index corresponding to data subcarrier n is subchannel s,
n	is the running index 0,...,47, indicating the data constellation point,
s	is the subchannel number,
$N_{subcarriers}$	is the number of subcarriers per slot.

2.6 Modulation

Subcarrier Randomization

The pseudo random binary sequence (PRBS) generator, as shown in Fig. 2.13, shall be used to produce a sequence w_k. The polynomial for the PRBS generator shall be $X^{11} + X^9 + 1$. 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, designated b10..b0, is defined as follows:

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.

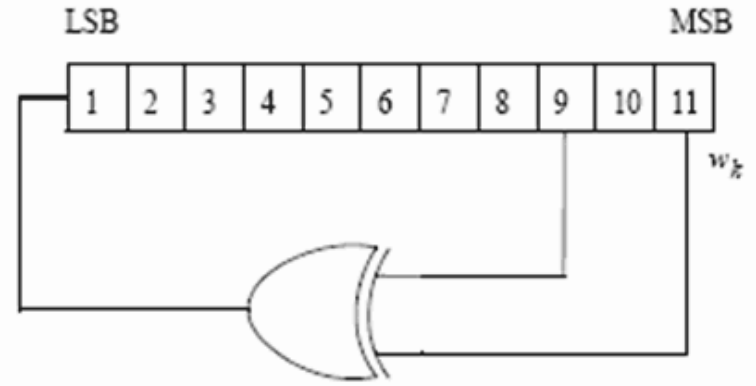


Figure 2.13: PRBS generator for pilot modulation (from [2]).

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.

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

$$\Re\{c_k\} = \tfrac{8}{3}(\tfrac{1}{2} - w_k) \cdot p_k,\ \Im\{c_k\} = 0, \tag{2.6}$$

where p_k is the pilot's polarity for SDMA (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

$$\Re\{PreamblePilotModulation\} = 4 \cdot \sqrt{2} \cdot (\frac{1}{2} - w_k), \tag{2.7}$$

$$\Im\{PreamblePilotModulation\} = 0. \tag{2.8}$$

2.7 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. 2.14 and

Table 2.5 in license-exempt bands. The measurements shall be made using 100 kHz resolution bandwidth and a 30 kHz video bandwidth. The 0 dBr level is the maximum power allowed by the relevant regulatory body. IEEE 802.16e dose not specify the power mask for the licensed bands.

2.8 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 an accuracy of $\pm 25\%$ of the minimum guard interval or better. For example, this translates into ± 8 samples in the case of 1024-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

Table 2.5: Transmit Sprctral Mask for License-Exempt Bands

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

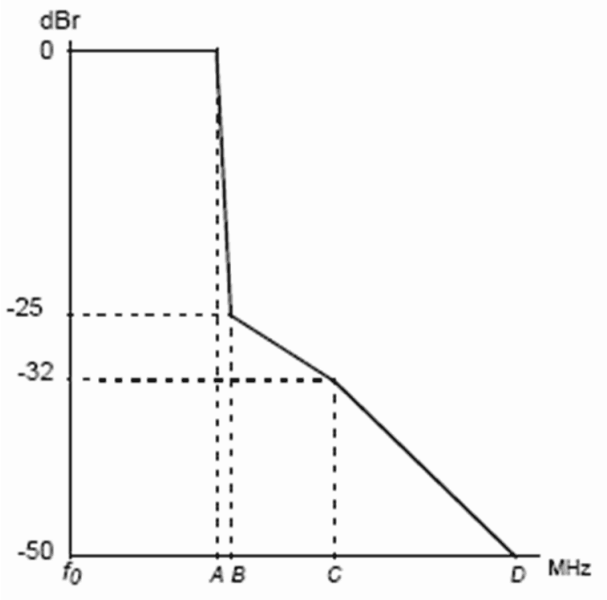


Figure 2.14: Transmit spectral mask for license-exempt operation (from [1]).

estimated UL frequency offset based on the downlink signal.

Chapter 3

Introduction to the DSP Implementation Platform

In this chapter, we introduce the DSP platform used in our implementation.We use the SMT395 DSP module made by Sundance housed on a Sundance PCI-plugin board. The DSP chip on the module is the TMS320C6416T made by Texas Instrument (TI). It also has a Xilinx Virtex II Pro FPGA. We will introduce the DSP card and the DSP chip. In addition, we will also introduce the software development tool, the Code Composer Studio (CCS), and the code development technique.

3.1 The DSP Card [11]

The DSP card used in our implementation is Sundance's SMT395 shown in Fig. 3.1. It houses a 1GHz 64-bit TMS320C6416T DSP of TI, manufactured on 90 nm technology. The SMT395 is supported by TI's Code Composer Studio and the 3L_Diamond_real-time operating system (RTOS) to enable multi-DSP systems with minimum programmer effort. It provides a flexible platform for various applications.

Some important features of the SMT395 module are as follows.

- 1 GHz TMS320C6416T fixed point DSP.
- 8000 MIPS peak DSP performance.
- Xilinx Virtex II Pro FPGA XC2VP30-6 in FF896 package.
- 256 Mbytes of SDRAM at 133MHz.

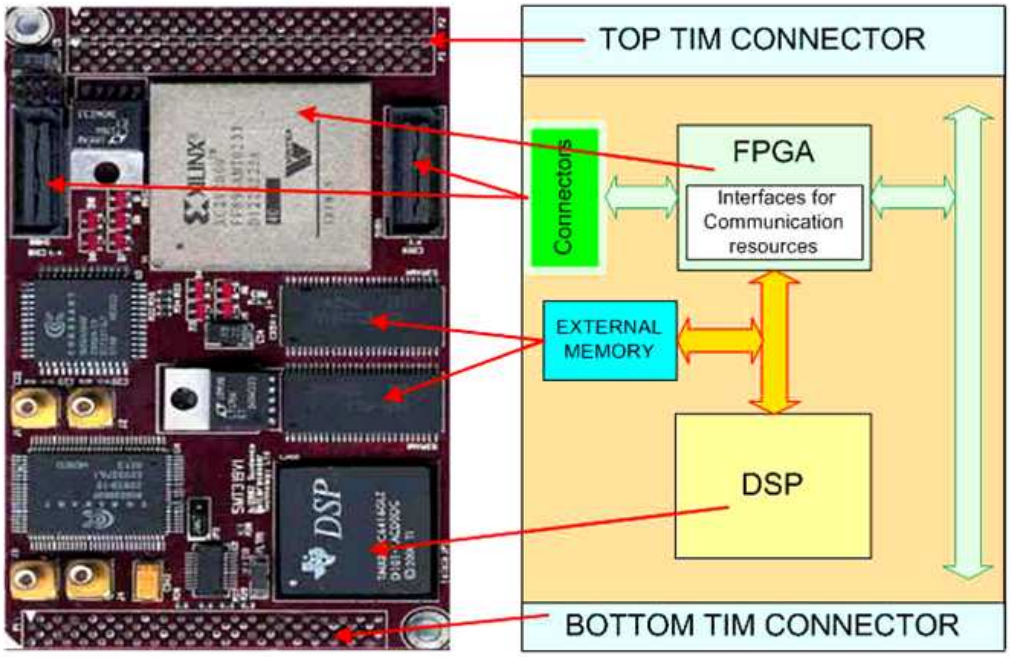


Figure 3.1: Sundance's SMT395 module

- Two Sundance High-speed Bus (50 MHz, 100 MHz or 200 MHz) ports at 32 bits width.
- Eight 2 Gbit/sec Rocket Serial Links (RSL) for inter-Module communications.
- Six common ports up to 20 MB per second for inter-DSP communication.
- 8 Mbytes flash ROM for configuration and booting.
- JTAG diagnostics port.

3.2 The DSP Chip [12]

The TMS320C6416T DSP is the a fixed-point DSP in the TMS320C64x series of the TMS320C6000 DSP platform family. It is based on the advanced VelociTI very-long-instruction-word (VLIW) architecture developed by TI. The functional block and DSP core diagram of TMS320C64x series is shown in Fig. 3.2.

The C6000 core CPU consists of 64 general-purpose 32-bits registers and eight function units. Features of C6000 device include the following.

- VLIW CPU with eight functional units, including two multipliers and six arithmetic:
  - Executes up to eight instructions per cycle.
  - Allows designers to develop highly effective RISC-like code for fast development time.
- Instruction packing:
  - Gives code size equivalence for eight instructions executed serially or in parallel.
  - Reduces code size, program fetches, and power consumption.
- Conditional execution of all instructions:
  - Reduces costly branching.
  - Increases parallelism for higher sustained performance.
- Efficient code execution on independent functional units:
  - Efficient C complier on DSP benchmark suite.
  - Assembly optimizer for fast development and improved parallelization.
- 8/16/32-bit data support, providing efficient memory support for a variety of applications.
- 40-bit arithmetic options add extra precision for vocoders.
- 32x32-bit integer multiply with 32- or 64-bit result.

In the following subsections, three major parts of the TMS320C64x DSP are introduced respectively. They are the central processing unit, memory, and peripherals.

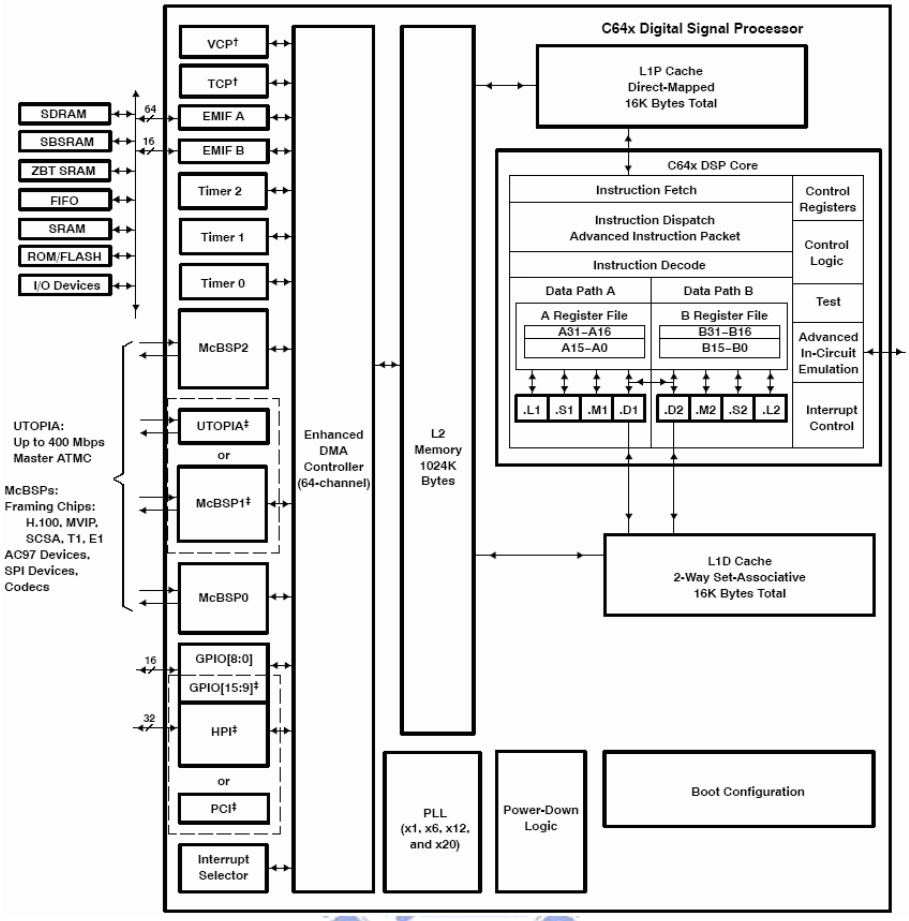


Figure 3.2: Functional block and CPU (DSP core) diagram [13].

3.2.1 Central Processing Unit

The C64x DSP core contains 64 32-bit general purpose registers, program fetch unit, instruction decode unit, two data paths each with four function units, control register, control logic, advanced instruction packing, test unit, emulation logic and interrupt logic. The program fetch, instruction fetch, and instruction decode units can arrange eight 32-bit instructions to the eight function units every CPU clock cycle. The processing of instructions occurs in each of the two data paths (A and B) shown in Fig. 3.2, each of which contains four functional units and one register file. The four functional units are as follows. The first unit is for multiplier operations (.M). The second unit is for arithmetic and logic operations (.L). The third is for branch, byte shifts, and arithmetic operations (.S). And the last is for linear and circular address calculation to load and store with external memory operations (.D). The details of the functional units are described in Table 3.1.

Table 3.1: Functional Units and Operations Performed [12]

Parameter	Value
.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 and Quad 8-bit arithmetic operations Dual 16-bit and 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 and Quad 8-bit compare operations Dual 16-bit and Quad 8-bit saturated arithmetic operations
.M unit (.M1, .M2)	16 x 16 multiply operations 16 x 32 multiply operations Dual 16 x 16 and Quad 8 x 8 multiply operations Dual 16 x 16 multiply with add/subtract operations Quad 8 x 8 multiply with add operations Bit expansion Bit interleaving/de-interleaving Variable shift operations Rotation Galois Field Multiply
.D unit (.D1, .D2)	32-bit add, subtract, linear and circular address calculation Loads and stores with 5-bit constant offset Loads and stores with 15-bit constant offset(.D2 only) Loads and stores doubles words with 5-bit constant Loads and store non-aligned words and double words 5-bit constant generation 32-bit logical operations

Each register file consists of 32 32-bit registers for each four functional units reads and writes directly within its own data path. That is, the functional units .L1, .S1, .M1, .D1 can only write to register file A. The same condition occurs in register file B. However, two cross-paths (1X and 2X) allow functional units from one data path to access a 32-operand from the opposite side register file. The cross path 1X allows data path A to read their source from register file B. The cross path 2X allows data path B to read their source from register file A. In the C64x, CPU pipelines data-cross-path accesses over multiple clock cycles. This allows the same register to be used as a data-cross-path operand by multiply functional units in the same execute packet.

3.2.2 Memory Architecture and Peripherals

The C64x is a two-level cache-based architecture. The level 1 cache is separated into program and data spaces. The level 1 program cache (L1P) is a 128 Kbit direct mapped cache and the level 1 data cache (L1D) is a 128 Kbit 2-way set-associative mapped cache. The level 2 (L2) memory consists of 8 Mbit memory space for cache (up to 256 Kbytes) and unified mapped memory.

The external memory interface (EMIF) provides interfaces for the DSP core and external memory, such as synchronous-burst SRAM (SBSRAM), synchronous DRAM (SRAM), SDRAM, FIFO and asynchronous memories (SRAM and EPROM). The EMIF also provides 64-bit-wide (EMIFA) and 16-bit-wide (EMIFB) memory read capability.

The C64x contains some peripherals such as enhanced direct-memory-access (EDMA), host-port interface (HPI), PCI, three multichannel buffered serial ports (McBSPs), three 32-bit general-purpose timers and sixteen general-purpose I/O pins. The EDMA controller handles all data transfers between the level-two (L2) cache/memory and the device peripheral. The C64x has 64 independent channels. The HPI is a 32-/16-bit wide parallel port through which a host processor can directly access the CPUs memory space. The PCI port supports connection of the DSP to a PCI host via the integrated PCI master/slave bus interface.

3.3 TI's Code Development Environment [14]

The Code Composer Studio (CCS) is a key element of the DSP software and development tools from Texas Instruments. The tutorial [15] introduces the key features of CCS and the programmer's guide [16] gives a reference for programming TMS320C6000 DSP devices. A programmer needs to be familiar with coding development flow and CCS for building a new project on the DSP platform efficiently.

3.3.1 Code Composer Studio

The CCS combines the basic code generation tools with a set of debugging and real-time analysis capabilities which supports all phases of the development cycle shown in Fig. 3.3. Some main features of the CCS 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.

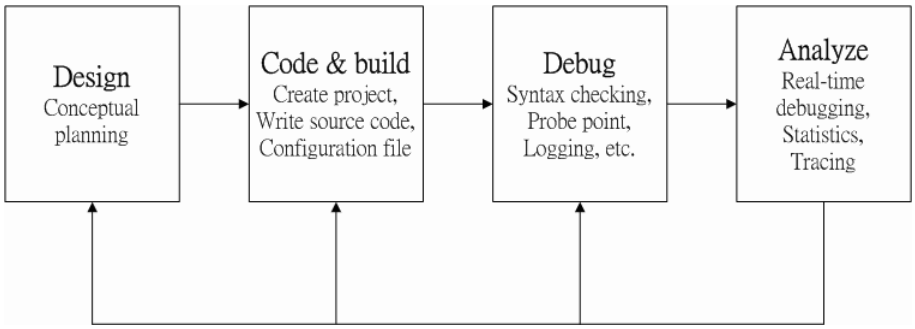


Figure 3.3: Code development cycle [15].

  - Interthread communication.
  - Interupt handing.

- 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.

3.3.2 Code Development Flow [16]

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. 3.4 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.

3.4 Code Optimization on TI DSP Platform

In this section, we describe several methods that can accelerate our code and reduce the execution time on the C64x DSP. First, we introduce two techniques that can be used to analyze the performance of specific code regions:

- 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.

3.4.1 Compiler Optimization Options [16]

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 use 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.

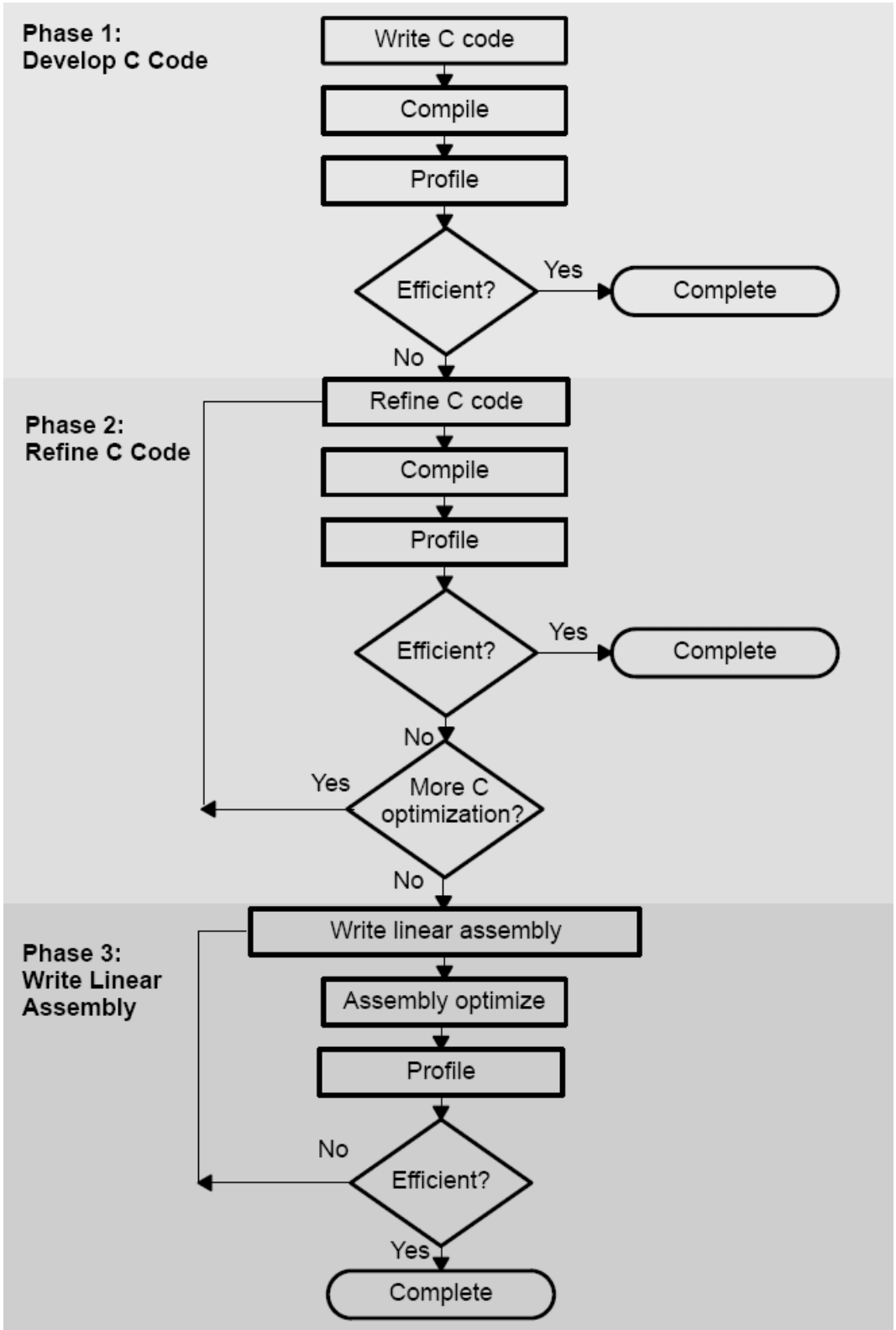


Figure 3.4: Code development flow for C6000 (from [16]).

  - -o3: highest level optimization, whose 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 addition, there is potential for performance improvement where interrupt registers may be utilized in high register pressure loops.

3.4.2 Software Pipelining [17]

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 feature we rely on to speed up our system. The compiler always attempts to software-pipeline. Fig. 3.5 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 [16]:

- 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.

3.4.3 Intrinsics [16]

We do not use any intrinsics in our code, but we introduce the concept of this technique here. The C6000 compiler provides intrinsics, which are special functions that

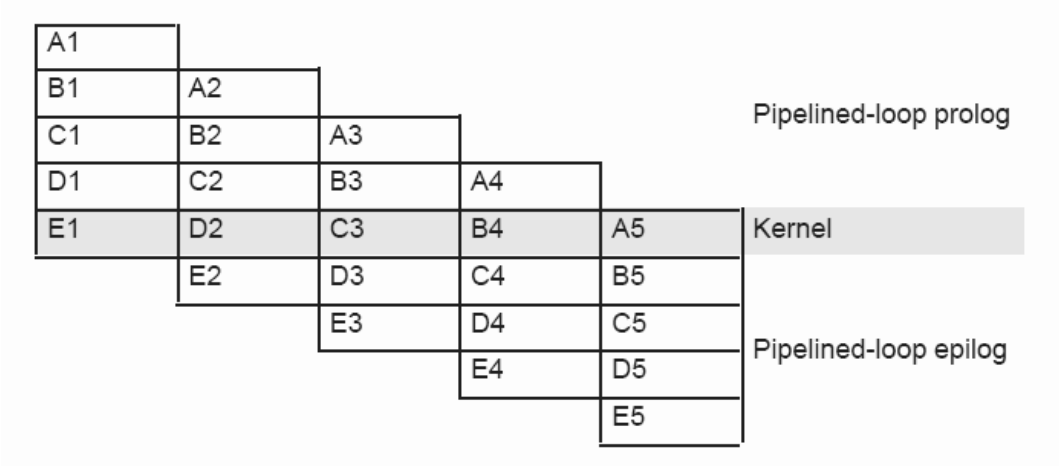


Figure 3.5: Software-pipelined loop (from [12]).

map directly to 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 [16].

Chapter 4

Transmission Environment and Transmission Filtering

In this chapter, we first discuss the transmission environment used in our study. Then we introduce the square–root raised cosine (SRRC) filter used for shaping of the power spectrum and controlling of the ISI.

4.1 System Parameters

We have to specify the system parameters so that the simulation environment can be constructed. The IEEE 802.16e standard is very flexible in choice of bandwidth and cyclic prefix length. However, it would be difficult to conduct the simulation and implementation study for all possible sets of parameters. Hence we pick a subset as follows.

The system profile we select is PMP, WirelessMAN-OFDMA PHY profile, TDD, and single-input single-output (SISO) operation. The FFT sizes are 1024 and 2048, and the carrier frequency is 3.5 GHz. We consider the mandatory PUSC permutation and use segment 0 to allocate data subcarriers.

The modulation could be QPSK, 16-QAM, or 64-QAM with randomly generated data. The frame duration could be 5 or 10 ms. Other parameter values are specified in Table 4.1.

Table 4.1: System Parameters Used in Our Study

Parameters	Values		
System Channel Bandwidth (MHz)	10	10	20
Sampling Frequency (MHz)	11.2	11.2	22.4
FFT Size	1024	2048	2048
Subcarrier Spacing (kHz)	10.94	5.47	10.94
Useful Symbol Time (μsec)	91.4	182.8	91.4
Guard Time (μsec)	11.4	22.8	11.4
OFDMA Symbol Time (μsec)	102.9	205.7	102.9

4.2 Channel Environments

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.

4.2.1 Gaussian Noise

The simplest kind of channel is the additive white Gaussian noise (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).

4.2.2 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

$$h(\tau) = \sum_{i=0}^{N-1} \alpha_i e^{j\theta_i} \delta(\tau - \tau_i) \tag{4.1}$$

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

4.2.3 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

$$h(\tau, t) = \sum_{i=0}^{N-1} \alpha_i(t) e^{j\theta_i(t)} \delta(\tau - \tau_i). \tag{4.2}$$

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 LOS path between the transmitter and the receiver, then 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. An improvement to Jakes' model is proposed in [18].

4.2.4 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.

The channel model used here is a modification of the Stanford University Interim (SUI) channel models proposed in [19]. These models, for a collect of scenarios, provide the parameters to model the various random phenomena involved with a simulation; of course there are many possible combinations of these parameters to obtain such channel descriptions.

A set of 6 typical channels was selected for the three most common terrain categories that are typical of the continental United States. The parametric view of the SUI channels is summarized in Tables 4.2 and 4.3.

Table 4.2: Terrain Type vs. SUI Channels

Terrain Type	SUI Channels
A: hilly terrain, heavy tree	SUI-5, SUI-6
B: between A and C	SUI-3, SUI-4
C: flat terrain, light tree	SUI-1, SUI-2

Table 4.3: General characteristics of SUI channels

Doppler	Low delay spread	Moderate delay spread	High delay spread
Low	SUI-1, SUI-2,SUI3		SUI-5
High		SUI-4	SUI-6

The scenario of SUI channels are:

- Cell size: 7 km.
- Base station antenna height: 30 m.
- Receiver antenna height: 6 m.
- Base station antenna beamwidth: 120°.
- Receiver antenna beamwidth: omnidirectional (360°) and 30°.
- Vertical polarization only.

We list the SUI channel models in Tables 4.4 to 4.9.

Another channel model chosen from one of the channel environments defined by ETSI is used in this thesis. The model is as shown in Table 4.10. This is a channel model for the vehicular test environment, where the tested speed is from 120 to 500 km/h. This environment is characterized by larger cells and higher transmit power, is valid for NLOS case only, and describes worse case propagation. Channel A is the low delay spread case that occurs frequently. See [20] for more details.

4.3 Transmission Filtering

We introduce the SRRC filter based on [21] here. 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. Figure 4.1 shows

Table 4.4: SUI-1 Channel Model

	Relative delay (μs or sample number)			Average power		
Tap	(μs)	(4×oversampling)	(normal)	(dB)	(normal scale)	(normalized)
1	0	0	0	0	1	0.9610
2	0.4	17	4	-15	0.0316	0.0303
3	0.9	40	10	-20	0.01	0.0096

Table 4.5: SUI-2 Channel Model

	Relative delay (μs or sample number)			Average power		
Tap	(μs)	(4×oversampling)	(normal)	(dB)	(normal scale)	(normalized)
1	0	0	0	0	1	0.9135
2	0.4	17	4	-12	0.0631	0.0576
3	1.1	49	12	-15	0.0316	0.0289

the spectrum of the signal after 4-times oversampling. The three duplicates of the original are undesired and need to be filtered out to meet the power mask. As the ideal lowpass interpolation filter cannot be implemented exactly, the easier realized SRRC filter is used instead. The impulse response of the filter is given by

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

where α is the roll-off factor. The reason for adopting the SRRC filter is that for this filter the transmitter and receiver filters are matched to each other and they introduce no inter-sample interference under ideal channels. Figure 4.2 shows that although the transmitter output signal contains inter-sample interference introduced by the SRRC filter due to its nonzero values at multiples of T_{sample}, the output of the receiver SRRC filter is ISI-free for the reason that the convolution of two SRRC filters has zero values at all multiples of T_{sample}.

Now, we have to decide the roll-off factor and the tap number of the SRRC filter (4-

Table 4.6: SUI-3 Channel Model

	Relative delay (μs or sample number)			Average power		
Tap	(μs)	(4×oversampling)	(normal)	(dB)	(normal scale)	(normalized)
1	0	0	0	0	1	0.7061
2	0.4	17	4	-5	0.3162	0.2233
3	0.9	40	10	-10	0.1	0.0706

Table 4.7: SUI-4 Channel Model

	Relative delay (μs or sample number)			Average power		
Tap	(μs)	(4×oversampling)	(normal)	(dB)	(normal scale)	(normalized)
1	0	0	0	0	1	0.6424
2	1.5	17	4	-4	0.3981	0.2557
3	4	67	16	-8	0.01585	0.1018

Table 4.8: SUI-5 Channel Model

	Relative delay (μs or sample number)			Average power		
Tap	(μs)	(4×oversampling)	(normal)	(dB)	(normal scale)	(normalized)
1	0	0	0	0	1	0.7061
2	4	179	45	-5	0.3162	0.2233
3	10	448	112	-10	0.1	0.0706

times oversampled). The frequency range from 0.212π to 0.288π in Fig. 4.1 corresponds to guard bands; hence the SRRC filter design can disregard this range. Thus the critical point for the design is the lowest frequency of the first duplicate (0.288π in Fig. 4.1). The power mask at this critical frequency is -26.63 dBr, so we have to design a filter which has smaller response than -26.63 dBr at this frequency. The magnitude responses of three different SRRC filters are shown in Fig. 4.3. Figure 4.3(a) is the response of a 57-taps SRRC filter (4-time oversampled) with roll-off factor $\alpha = 0.15$. If the roll-off factor is larger, say 0.155 as in Figure 4.3(b), the power will be larger than -26.63 dBr at the critical frequency. A roll-off factor smaller than 0.15 would satisfy the mask at the critical frequency. However, the power at high frequency becomes larger than -50 dBr as shown in Fig. 4.3(c). Moreover, if the tap number is smaller, the power will not meet the mask at high frequency no matter how small the roll-off factor is. Figure 4.4 shows the frequency response of 49-taps SRRC filter with $\alpha = 0.15$. Finally, we adopt the 57-taps SRRC filter with $\alpha = 0.15$.

The spectral density of the transmitted signal after oversampling and SRRC filtering is

Table 4.9: SUI-6 Channel Model

	Relative delay (μs or sample number)			Average power		
Tap	(μs)	(4×oversampling)	(normal)	(dB)	(normal scale)	(normalized)
1	0	0	0	0	1	0.8773
2	14	627	156	-10	0.1	0.0877
3	20	896	224	-14	0.0398	0.0349

Table 4.10: ETSI "Vehicular A" Channel Model in Different Units [20]

	Relative delay (μs or sample number)			Average power		
Tap	(μs)	(4×oversampling)	(normal)	(dB)	(normal scale)	(normalized)
1	0	0	0	0	1	0.4850
2	0.31	13	3	-1	0.7943	0.3852
3	0.71	31	8	-9	0.1259	0.061
4	1.09	48	12	-10	0.1	0.0485
5	1.73	77	19	-15	0.0316	0.0153
6	2.51	112	28	-20	0.01	0.0049

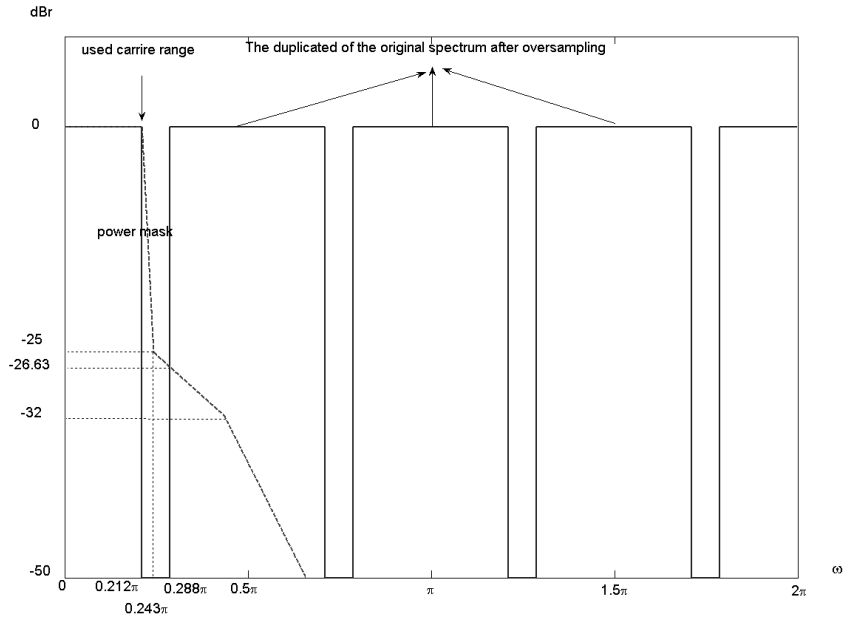


Figure 4.1: Frequency spectrum of the signal after 4 times oversampling and relation to the power mask.

shown in Fig. 4.5. The transmitted signal we have used here is the DL preamble symbol. We can see that the transmitted spectral density satisfies the required power mask.

In the receiver, we use the same SRRC filter followed by 4-times downsampling. We apply the polyphase technique for implementing the interpolation filter and the decimation filter to reduce the complexity. More details can be found in [21].

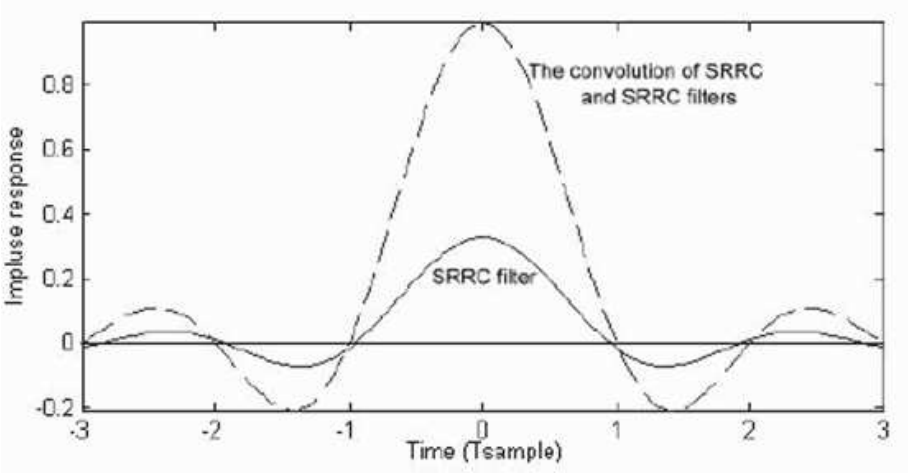


Figure 4.2: Impulse response of the SRRC filter (solid) and the convolution of the impulse responses of two SRRC filters (dashed) [21].

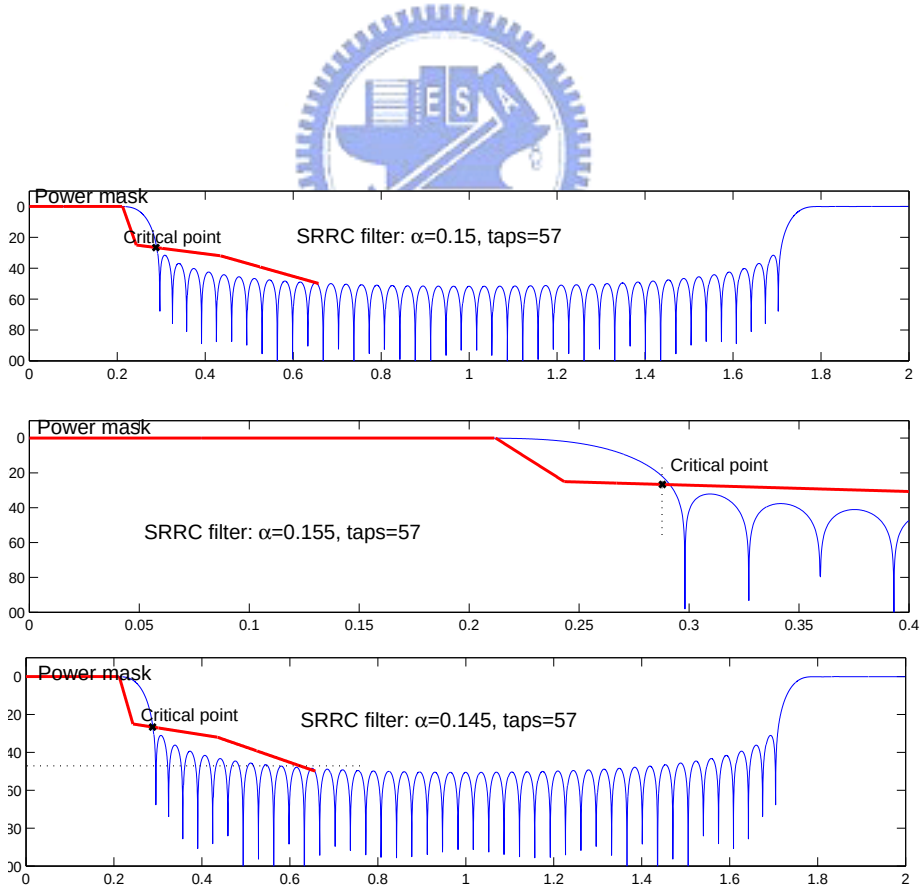


Figure 4.3: Magnitude responses of three different SRRC filters.

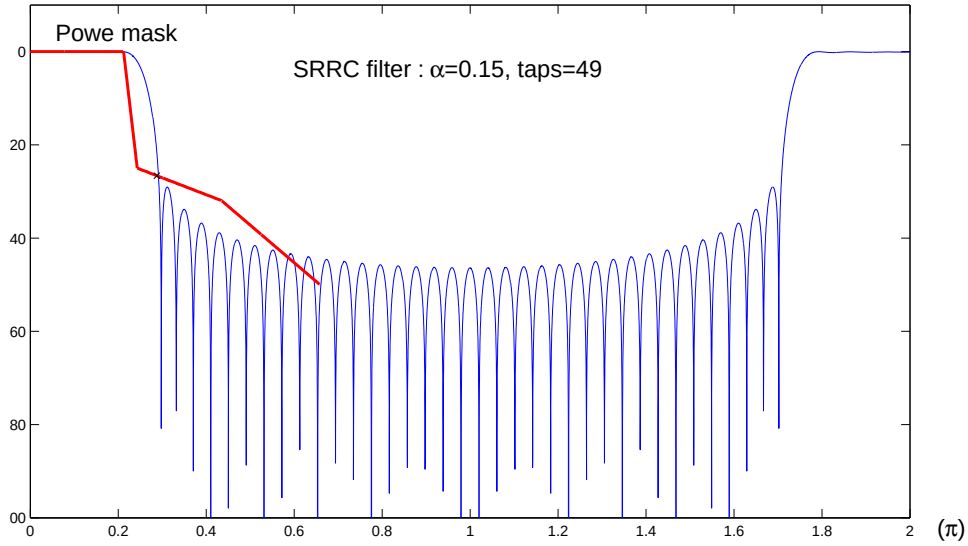


Figure 4.4: Magnitude responses of 49-taps SRRC filter with roll-off factor 0.15.

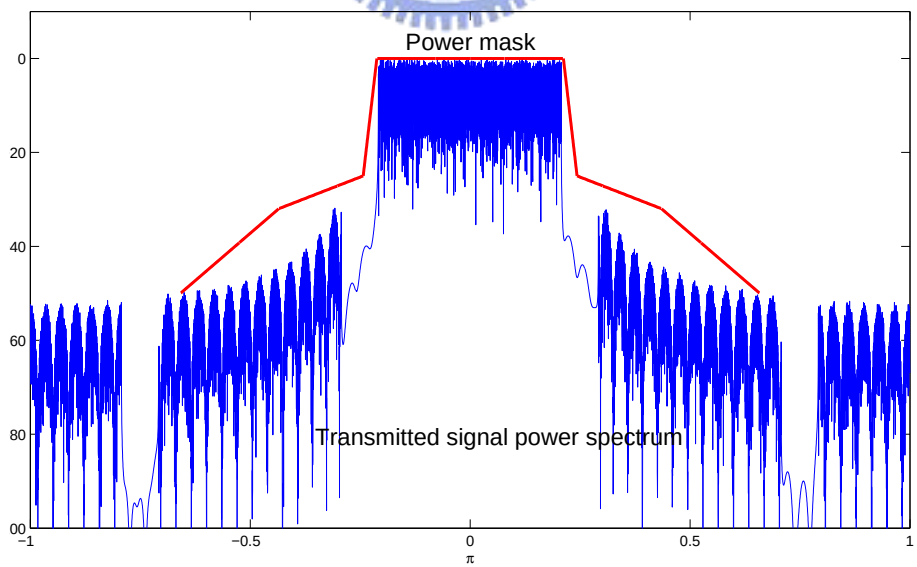


Figure 4.5: Spectral density of the signal after the interpolation and SRRC filtering, compared to the spectral mask.

Chapter 5

Synchronization Techniques for IEEE 802.16e OFDMA

In this chapter, we discuss the synchronization issues of the IEEE 802.16e OFDMA TDD system and present our synchronization techniques.

5.1 OFDMA Synchronization Problems and Techniques

Accurate demodulation and detection of an OFDM signal requires subcarrier orthogonality. Variations in the carrier oscillator, sampling clock or the symbol time affect this orthogonality. Therefore, before a receiver can demodulate the signal, it has to perform synchronization. The synchronizer estimates and compensates any offsets in carrier, sampling time, and OFDMA symbol time in the receiver in reference to the transmitter.

In OFDMA DL, there are two synchronization conditions: initial synchronization and normal synchronization. We apply initial DL synchronization when the mobile station (MS) receiver enters the network for the first time. We assume that the frame synchronization is done by monitoring the power of the received signal. Initial DL synchronization at the MS includes carrier recovery and timing recovery. Upon entering the network and upon a need to handover, the MS has to identify the preamble index of the BS segment that it will communicate with. Therefore, another important task needed to be done during initial synchronization is to find the preamble index. Carrier recovery involves estimation and compensation of the carrier frequency offset (CFO) and timing recovery, in principle, should include estimation and compensation of the sampling frequency offset (SFO), sampling timing offset, and the OFDMA symbol time offset. The carrier phase

offset may be considered part of the channel response and resolved by channel estimator. The sampling timing offset can also be absorbed into the channel response if the CP is long enough.

Note that the IEEE 802.16e OFDMA requires not only that the MS carrier frequency shall be synchronized to the BS within 2% of the subcarrier spacing, but also that the transmitted center frequency and the sampling frequency of the MS shall be derived from one reference oscillator. When these are true, the sampling phase difference from the beginning of an OFDMA symbol to the end of it will only differ by at most $2\% \times (1+1/4)$ of the true sample period, where the factor 1/4 accounts for the largest allowed CP time ratio. As a result, it appears unnecessary to perform separate SFO recovery in either the MS or the BS provided that the CFO can be accurately recovered. What remain to be synchronized, besides preamble index identification in the DL, are therefore the CFO and the OFDMA symbol timing.

Nonzero CFO results in ICI where nearby subcarriers interfere with one another. On the other hand, incorrect symbol timing estimation may lead to two different kinds of impairment. A negative timing error (or lead-error, where the estimated timing is earlier than the actual timing) amounts to adding a delay to the channel response. Hence a negative timing error causes no performance problem as long as the amount of error plus the original length of channel response is still within the CP length. Nevertheless, a smaller negative timing error can better ensure proper system operation if the length of channel response can vary from time to time. On the other hand, positive timing errors (i.e., lag-errors, where the estimated timing is later than the actual timing) is more detrimental to system performance because one would mistake some of next symbol's samples for part of the present symbol. As a result, it is desirable to minimize the probability of positive timing errors. In Fig. 5.1, we can see that if the length of CP exceeds the channel impulse response, as long as a receiver captures an OFDM symbol starting in the allowable region, the OFDM symbol appears cyclic, orthogonality is maintained, and ISI and ICI are avoided. Note that the above material is largely taken from the contents of [10].

In the DL, due to potentially large tolerance in the free-running oscillator frequency of the MS, and due to the motion-induced Doppler spread, there may be large CFO in the

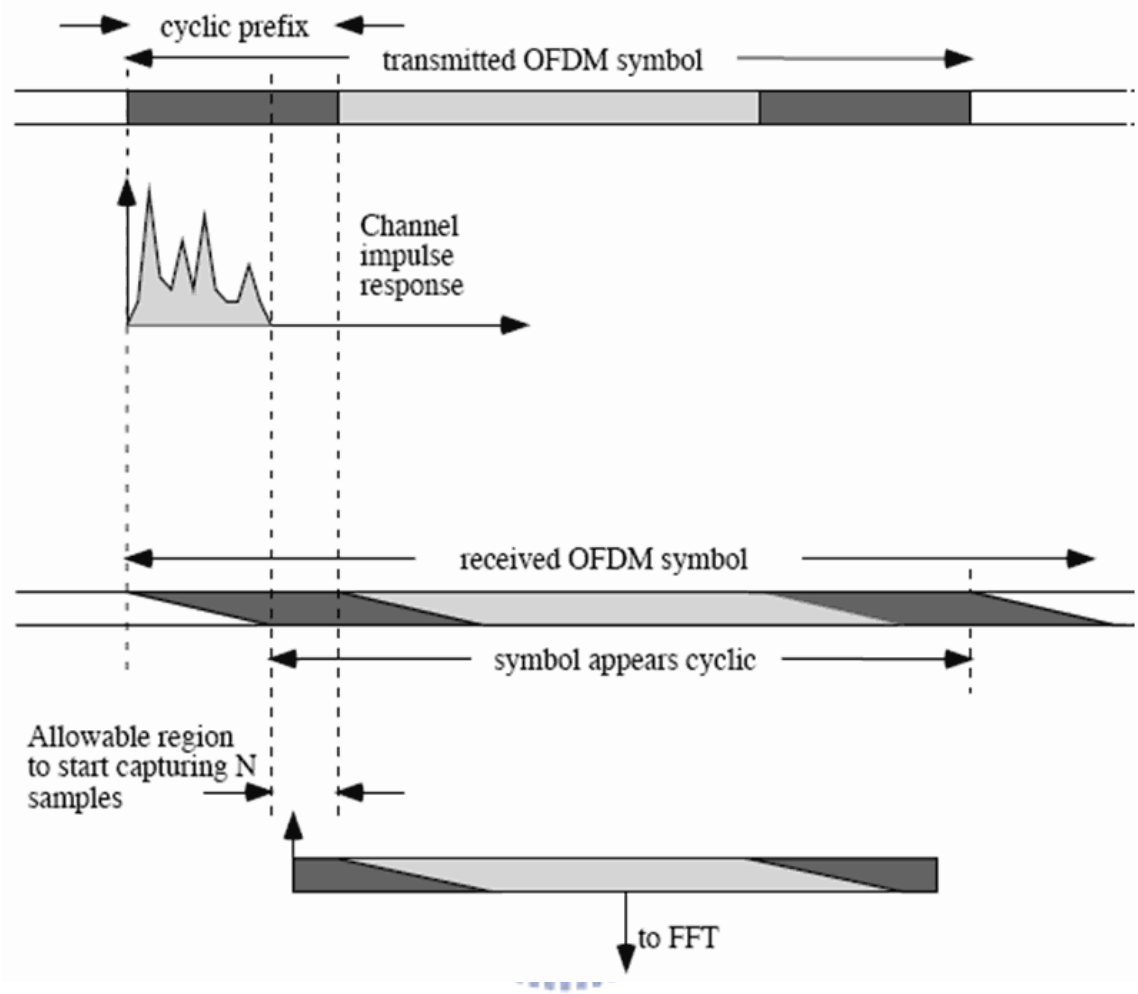


Figure 5.1: The symbol time offset requirement (from [22]).

received signal. The large CFO can be partitioned into the fractional part of "normalized CFO" (where normalization is with respect to the subcarrier spacing) and integral part of "normalized CFO". We call them "fractional CFO" and "integer CFO" respectively in this thesis.

In the DL normal synchronization, the synchronization task is easier, as the MS no longer has to reidentify the preamble index, nor does it have to do acquisition of the carrier and the timing but only need to track them. Disregarding the ranging operation, the key task in UL synchronization (at the BS) involves timing recovery only. We mainly focus on the DL synchronization rather than the UL synchronization in this thesis.

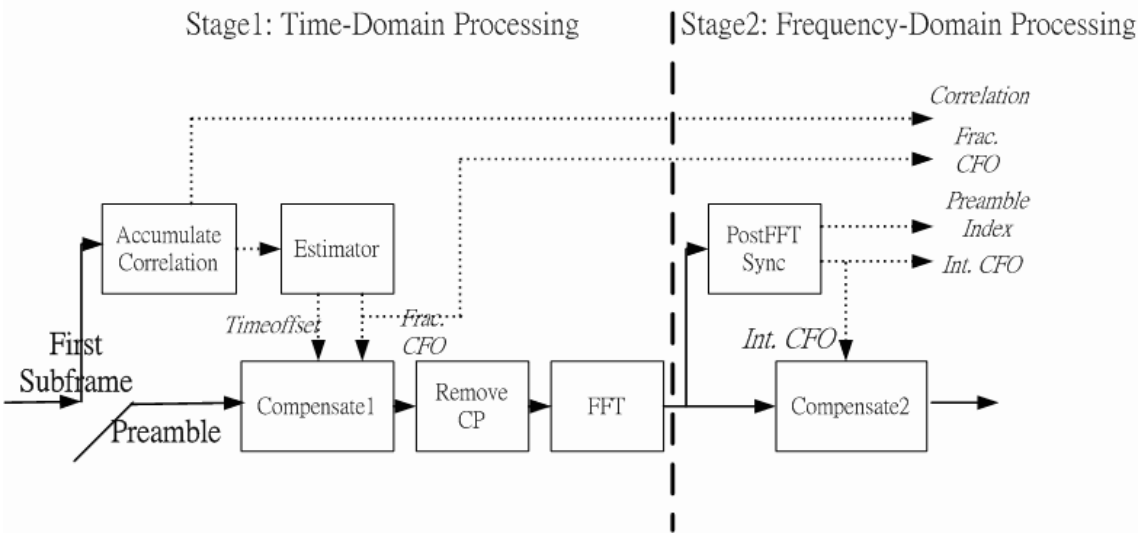


Figure 5.2: Structure of initial DL synchronization.

5.1.1 Initial DL Synchronization

As mentioned, we have to estimate the symbol timing offset, CFO, and preamble index during initial DL synchronization. We achieve the task of initial DL synchronization in several stages after considering the complexity for DSP implementation. Fig. 5.2 depicts the overall structure of the proposed initial DL synchronization. It is a two-stage organization where stage1 operates in the time domain to estimate the OFDMA symbol timing and the fractional CFO, and stage 2 operates in the frequency domain to detect the integer CFO and the preamble index jointly.

Below we use the parameters of the 1024-FFT system as an example to describe the synchronization techniques.

5.1.1.1 Timing Offset and Fractional Carrier Frequency Offset

A number of CFO estimation approaches have been proposed. One is the data-aided approach [23], [24], applicable when the preamble consists of known signal (or when a reliable decision on the preamble contents can be made). In the case of IEEE 802.16e OFDMA, it is not suitable because in DL, the preamble can be one of 114 choices, and in UL, the a prior known signal (i.e., the pilots) only consists of one-third of the received signal. The second approach is based on subspace analysis, e.g., via the ESPRIT algo-

rithm [25], [26]. While the resolution in CFO estimation of these methods can be high, the computational complexity can also be high. The third approach is completely blind estimation relying solely on the repetitive signal structure of OFDMA symbols, e.g., the presence of CP. This appears simplest and suitable for use in IEEE 802.16e OFDMA. Similarly, there are several approaches to symbol timing estimation, for example, an approach utilizing the quasi-periodic time-domain structure of the preamble is introduced in [27]. But, again, one simplest and appropriate for IEEE 802.16e OFDMA is blind estimation based on CP structure.

By taking advantage of CP, an algorithm proposed in [28] can estimate the symbol timing instance and frequency offset relatively accurately in AWGN, blindly with no assistance from pilot symbols. Nevertheless, it suffers considerable performance degradation in multipath propagation or Rayleigh fading [21]. A modified technique proposed in [29] is shown to have better performance in fast Rayleigh fading.

Let N be the FFT size and L be the CP length in number of samples. Figure 5.3 illustrates the algorithm structure proposed in [29]. Under the assumption that the received samples are jointly Gaussian, symbol time offset $\hat{\tau}$ and fractional CFO $\hat{\theta}$ are given by

$$\hat{\tau} = \arg\max\{c|\lambda(\tau)|^2\}, \tag{5.1}$$

$$\hat{\theta} = -\frac{1}{2\pi}\tan^{-1}\left(\frac{Im\{\lambda(\hat{\tau})\}}{Re\{\lambda(\hat{\tau})\}}\right), \tag{5.2}$$

respectively, where

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

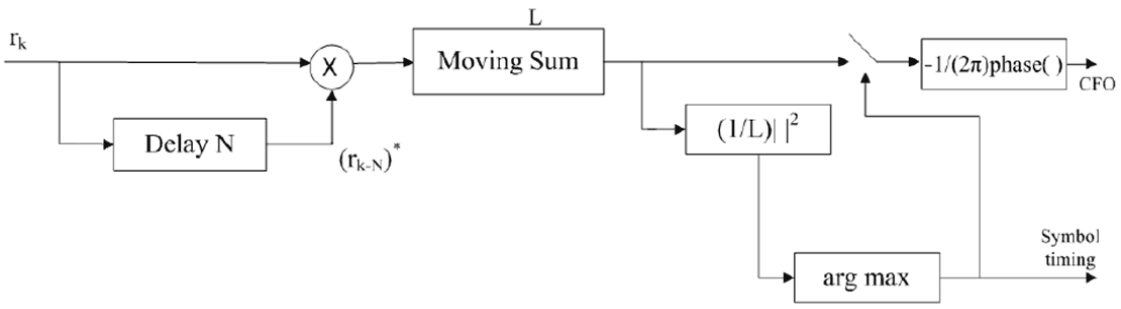


Figure 5.3: Structure of J.-C. Lin's symbol timing and fractional carrier frequency synchronization method [29].

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

As mentioned, one most important task needed to be done during initial DL synchronization is to find the preamble index. Consequently, we may sacrifice the signal in the first DL subframe in order to estimate the preamble index more accurately. As a result, we use all the information in the DL subframe to attempt a more accurate estimate of timing offset and fractional CFO, and then find the preamble index by using the preamble symbol compensated by the estimated timing offset and CFO. We accumulate the real part and the imaginary part of $\lambda(\tau)$ of all symbols in the first DL subframe to get $Re\{A(\tau)\}$ and $Im\{A(\tau)\}$. After that, we estimate the timing offset and fractional CFO in the last symbol of the DL subframe with

$$\hat{\tau} = \arg\max\{c|A(\tau)|^2\}, \tag{5.3}$$

$$\hat{\theta} = -\frac{1}{2\pi}\tan^{-1}(\frac{Im\{A(\hat{\tau})\}}{Re\{A(\hat{\tau})\}}). \tag{5.4}$$

5.1.1.2 Integer CFO Estimation and Preamble Index Identification

Since the preamble index must be estimated by using the preamble, we keep the received preamble signal in a buffer and compensate it with the fractional CFO and timing offset estimated in the last symbol of the first DL subframe. Then we estimate the integer CFO and identify the preamble index using the compensated preamble in the frequency domain.

Note that there are three types of preamble carrier-sets and each segment uses only one carrier-set. Carrier-set 0 uses subcarrier indexes 86, 89, 92, ..., 932, 935. The subcarrier indexes used by carrier-set 1 are those used by carrier-set 0 adding 1, and carrier-set 2, adding 2. Because of this, we cannot find the exact integer CFO until we find the correct preamble index which contains the information about the used carrier-sets. Here we consider several methods to find the preamble index and integer CFO jointly.

A. Correlation Method [30]

First, we exploit some signal structure to find the coarse integer CFO. In IEEE 802.16e OFDMA, the pilot subcarriers in a preamble are limited to the central part of the band and there are guard bands at both bandedges. The pilot subcarriers are BPSK modulated.

Hence, we may look for the lower and upper ends (in frequency) of the pilot subcarriers by examining the subcarrier amplitudes. Since the pilot subcarriers in a preamble sequence are spaced three subcarriers apart, we may add up the power of every third subcarrier starting from the lower-end subcarrier as identified above up to the number of pilot subcarriers in the preamble. To account for the possible mis-identification of the lower or upper end of nonzero subcarriers, we may repeat the above power sum computation starting from several subcarrier locations around the assumed lower end. Actually, there are many terms repeated in each sums, so we can discard those repeated terms without affecting the performance. Finially, all the power sums may be compared, and we can find the coarse integer CFO from the one with the largest sum.

Since the preamble carrier-set can be one of three possibilities, the estimated integer CFO, say f_I, through the above method does not necessarily give the actual integer CFO, but could be ± 1 away from it. The exact value will have to be determined after identification of the preamble index.

After that, we conduct an exhaustive search to identify the preamble index by correlating each of the 114 candidates with the received signal in the frequency domain and find the one with the largest magnitude of correlation. From the identified preamble index, we also obtain the corresponding segment and carrier-set as well as the actual integer CFO.

Correlation of the received preamble with a possible preamble sequence does not necessarily give a good indicator to their degree of match, because for it to be a good indicator, the preamble cannot span a frequency range much beyond the coherence bandwidth– but unfortunately this can be very far from the truth under the IEEE 802.16e OFDMA DL system parameters discussed previously.

Towards a solution, note that since the coherence bandwidth may cover several subcarrier spacings, we may employ a technique resembling differential detection but working in the frequency domain [31] prior to the cross-correlation in order to mitigate the corruption of either frequency selective fading or nonzero OFDM symbol timing offset. We introduce the differential method and a number of variations to simplify the computational complexity in the following. More details can be found in [32].

B. Differential Method [32]

First, we find the maximum power of all the subcarriers in the 3 possible carrier-sets which start from the $(86 - f_{max})$th subcarrier, the $(87 - f_{max})$th subcarrier, and the $(88 - f_{max})$th subcarrier of the received preamble, respectively, and determine the carrier-set used for estimation, where f_{max} is the maximum CFO search range. Note that the carrier-set we use here has length $284 + int(f_{max}/3) \times 2 + 1$, where 284 is the number of pilots in the DL preamble. Second, we derive the differential sequence from this carrier-set by

$$D_R[k] = Re\{r_k r_{k+1}^*\} = r_{re,k} r_{re,k+1} + r_{im,k} r_{im,k+1} \tag{5.5}$$

where $k = 0, 1, ..., 282 + int(f_{max}/3) \times 2 + 1$.

In writing the above, we have slightly abused the index k to let it indicate the kth nonzero preamble subcarrier rather than the kth OFDMA subcarrier. This is because in IEEE 802.16e OFDMA, the nonzero preamble subcarriers are not contiguous but are spaced three subcarriers apart.

Then we derive 114 possible differential sequences from the known preamble sequences by

$$D_j[k] = 1 - 2 \times q_j[k] \oplus q_j[k+1] \tag{5.6}$$

where $j = 0, 1, ..., 113$, $k = 0, 1, ..., 282$, $q_j[k] \in \{0, 1\}$ is the jth binary preamble sequence, and $\oplus$ denotes the "exclusive or" operation. In the end, we compute $114 \times (int(f_{max}/3) \times 2 + 1)$ possible metrics by using

$$M_{n,j} = \sum_{k=0}^{282} D_{R_n}[k] \times D_j[k] \tag{5.7}$$

where $n = 0, 1, ..., int(f_{max}/3) \times 2$ and D_{R_n} is the sequence started from the nth value of D_R, and the length of this sequence is 283 . Then we can find the preamble index $\hat{j}$ and $\hat{n}$ by

$$(\hat{n}, \hat{j}) = \arg \max_{n,j} M_{n,j}. \tag{5.8}$$

Note that $\hat{n}$ denotes the lower-end subcarrier in the carrier-set and the estimated preamble index $\hat{j}$ determines the lower-end subcarrier of the preamble symbol in the transmitter. Therefore, we can derive the integer CFO using $\hat{n}$ and $\hat{j}$.

C. Early Dropping of Bad Candidates by Dynamic Metric Thresholding [32]

This method is used to reduce the computational complexity by dropping the bad candidates early on. We divide the summation over 283 subcarriers in the metric computation into a number of fixed-length windows. Rather than finish computing the metric for each (n, j) pair over the total number of 283 subcarriers and then compare for the best, we may set a threshold after each window. The (n, j) pairs that perform below the threshold are dropped. This continues until only one (n, j) pair remains or until we come to the end of the last window when all the surviving (n, j) pairs are compared.

There are several parameters that can be designed in this method. We can design the length of the window size and the threshold for each window. In our design, we set the threshold to a fraction of the largest metric of all the retained (n, j) pairs. It goes without saying that the detection performance depends on the window size and the threshold, and so is the computational complexity. In the end, considering that the channel may be subject to multipath fading, it may not be good to let each window consist of contiguous preamble subcarriers. Rather, the windows should be interleaved, with the subcarriers in each window spaced, for example, beyond the coherence bandwidth.

D. Reduction of Search Range Through Coarse Estimation of Integer CFO [32]

In this method, we use the guard bands to reduce the number of candidates for integer CFO. First, we use the same approach as the first step of the correlation method to find the coarse integer CFO. This estimated coarse integer CFO may not be the true integer CFO, but may be near the true value. Then we can apply the differential method discussed previously around this coarse integer CFO for a more accurate result, but now the search range can be more restricted than without the above coarse search. Furthermore, we may apply the early dropping method around the coarse integer CFO to further reduce the computational complexity.

E. Hardlimiting of the Differential Signal [32]

Here, we consider hardlimiting the "differential signal" to simplify the differential method. First, we still need to find the carrier-set used for estimation as the first step of the differ-

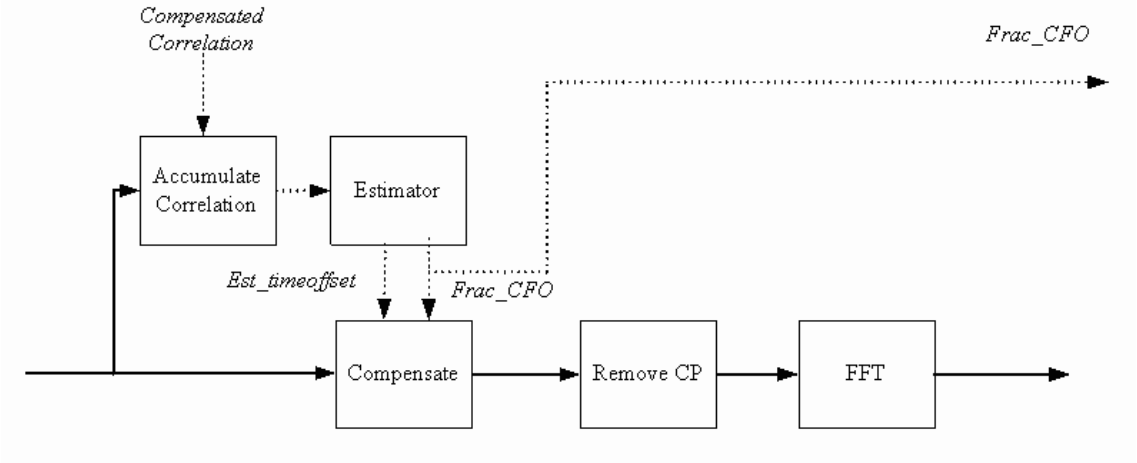


Figure 5.4: Structure of normal DL synchronization.

ential method. Then we derive the hardlimited differential sequence from this carrier-set by

$$D_R[k] = \begin{cases} 1, & sgn\{Re[r_k r_{k+1}^*]\} < 0, \\ 0, & sgn\{Re[r_k r_{k+1}^*]\} > 0, \end{cases} \tag{5.9}$$

where $k = 0, 1, ..., 282 + int(f_{max}/3) \times 2 + 1$. Then, we derive 114 possible differential sequences from known preamble sequences by

$$D_j[k] = q_j[k] \oplus q_j[k+1] \tag{5.10}$$

where $j = 0, 1, ..., 113$, $k = 0, 1, ..., 282$, and $q_j[k] \in \{0, 1\}$. In the end, we compute the Hamming distance between the $D_j[k]$ and $D_{R_n}[k]$ by

$$M'_{n,j} = \sum_{k=0}^{282} D_{R_n}[k] \oplus D_j[k] \tag{5.11}$$

where $n = 0, 1, ..., int(f_{max}/3) \times 2 + 1$. Then we can find the preamble index $\hat{j}$ and $\hat{n}$ by

$$(\hat{n}, \hat{j}) = \arg\min_{n,j} M'_{n,j}. \tag{5.12}$$

Again, we can derive the integer CFO through the preamble index $\hat{j}$ and $\hat{n}$ as we do in the differential method. This method is particularly useful for reducing the complexity of hardware implementation.

5.1.2 Normal DL Synchronization

During normal DL synchronization, we need to track the CFO and the timing offset in the receiver. Fig. 5.4 depicts the structure of normal DL synchronization. After initial DL synchronization, we compensate the oscillator frequency in the MS by using the CFO we estimated. As a result, there is no need to estimate the integer CFO during normal DL synchronization. Again, we use the CP correlation method [29] to estimate the fraction CFO and symbol timing. In order to get a more accurate estimation, we use the correlation value from the previous subframe in the present subframe. Since the oscillator frequency is compensated after every subframe, we also need to compensate the correlation value from the previous subframe every subframe by using

$$\hat{A}(\tau) = \frac{A(\tau)e^{-j\theta}}{N} \tag{5.13}$$

where $N = 25$ for the first subframe for normal synchronization and $N = 1$ otherwise, $\hat{A}(\tau)$ is the correlation value used for current symbol, $A(\tau)$ is the correlation value of previous subframe, and θ is the estimated fractional CFO of previous subframe.

Note that we assume that there are 25 symbols in one DL subframe. Since we use the accumulation method to get more accurate results during initial DL synchronization, we have to divide the compensated correlation value by 25 after the first DL subframe. Then, we employ exponential average over the symbols in the DL subframe during normal synchronization in order to obtain a more accurate estimation. We use

$$\begin{aligned} \hat{A}_0 &= \alpha\hat{A} + (1-\alpha)A_0, \\ \hat{A}_k &= \alpha\hat{A}_{k-1} + (1-\alpha)A_k, \quad k = 1, 2, ..., 24, \end{aligned} \tag{5.14}$$

to acquire the correlation value of each symbol, where A_k is the correlation value obtained in the current symbol and $\hat{A}_k$ is the exponential average result. In the end, symbol timing offset and fractional CFO for each symbol are given by

$$\hat{\tau}_k = \arg\max\{c|\hat{A}_k(\tau)|^2\}, \tag{5.15}$$

$$\hat{\theta}_k = -\frac{1}{2\pi}\tan^{-1}\left(\frac{Im\{\hat{A}_k(\hat{\tau})\}}{Re\{\hat{A}_k(\hat{\tau})\}}\right). \tag{5.16}$$

Here we introduce a data-aided method to estimate the symbol timing offset. Since we have identified the preamble index during initial DL synchronization, the preamble

symbol is known during normal DL synchronization. We may use the known preamble to estimate the timing offset during normal synchronization. The received preamble signal in the time domain after CP removal can be expressed as

$$r(k+m) = h(k) \circledast p(k)e^{j2\pi nk/N} + \eta(k) \tag{5.17}$$

where $k = 0, 1, ..., N-1$, $h(k)$ is the discrete channel response, $p(k)$ is the preamble symbol in the time domain, n is the CFO, m is the timing offset, $\circledast$ denotes the circular convolution of length N, and $\eta(k)$ is additive noise, assume white Gaussian.

Numerical evaluation shows that the preamble sequences in the time domain are quasi-orthogonal over a large range of time offset values, mutually and with self, in the sense that

$$|\sum_{k=0}^{N-1} p_i(k)p_j^*((k+m)\text{mod}N)e^{j2\pi nk/N}| \ll \sum_{k=0}^{N-1} |p_i(k)|^2 \tag{5.18}$$

unless $i = j$ and $n = 0$, for a large range of values of m around 0. For each possible time offset value m, we may perform the circular correlation of $p(k)$ with $r(k+m)$ over all lag values up to the length of CP. Note that $p(k)$ is the known preamble in the time domain. Let $y(m)$, $m = 0, 1, ..., L-1$, denote the result. Then the estimated timing offset is given by

$$\hat{m} = \arg\max_m y(m). \tag{5.19}$$

However, the complexity of the above approach can be very significant because it requires $L \times N$ complex multiplications. Another way to reduce the complexity is to use the frequency domain equivalent to calculate the circular correlation. Mathematically, we have

$$Y(k) = R(k)P(k)^*, \tag{5.20}$$

$$y(m) = \mathcal{F}^{-1}\{Y(k)\}, \tag{5.21}$$

where $\mathcal{F}$ denotes the fast Fourier transform (FFT). Then, we derive the timing offset using (5.19). This approach requires on order of $N \log_2 N$ complex multiplications for IFFT and on order of N_p complex multiplication for correlation, where N_p is the number of pilot subcarriers in the frequency-domain preamble sequence. Furthermore, since the preamble sequences in the frequency domain are BPSK, the multiplications are simple.

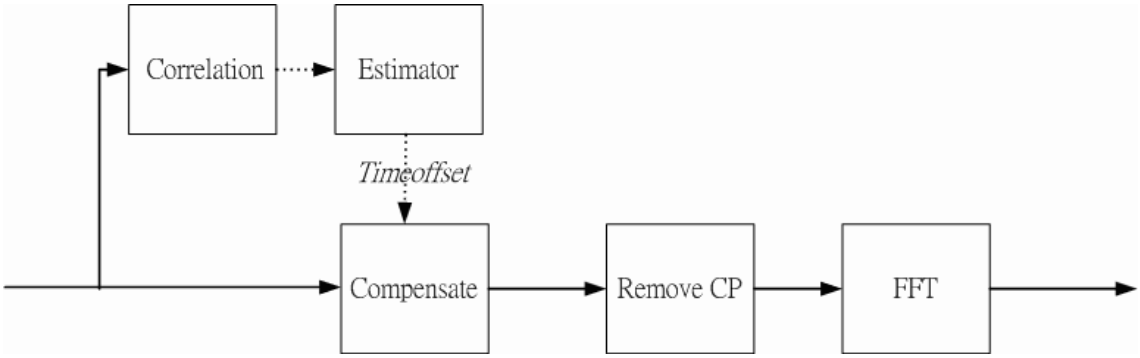


Figure 5.5: Structure of UL synchronization.

5.1.3 UL Synchronization

As mentioned before, we only apply timing recovery in the BS receiver during UL transmission. The techniques we use to estimate the symbol timing is the same as we use in normal DL synchronization. Fig. 5.5 depicts the structure of UL synchronization.

5.2 Floating-Point Simulation Results

The system parameters for our simulation are defined in Table 4.1, and the channel environments have been given in Chapter 4. We only use 16-QAM modulation in the simulation for simplicity. Note that the receiver SNR specified in the IEEE 802.16e OFDMA is from 5 to 20 dB (see Table 5.1). Our simulated SNR values are in the range 0 to 20 dB, which is a suitable range for 16-QAM modulation. The mobile speed is from 0 to 120 km/hr, and the CFO is 9.35 Δf. The symbol timing offset is 10 samples. Note that we do not take sampling inaccuracy caused by the SFO into consideration in our simulation. If not mentioned, the FFT size we use in this section is 1024, and we present some simulation results for FFT-2048 in the next chapter together with the fixed-point simulation results. Note that the signal-to-noise ratio (SNR) used in our simulations means the ratio of the variance of the signal samples to that of the noise samples.

5.2.1 Symbol Timing Estimation

Figure 5.6 shows timing error distribution in AWGN (upper two charts) and SUI3 channel (lower two charts) at SNR of 10 and 20 dB, respectively. The mobile speed is 120 km/hr.

Table 5.1: OFDMA Receiver SNR Assumptions [2]

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

We can see that in AWGN channel, the correct rates (the probability that we estimate the correct timing offset) for SNR of 10 and 20 dB are both 100%. This is because the length of 128-sample CP is long enough to alleviate the Gaussian noise effect. Also, we can see that the correct rate for SUI3 channel is not very high even at high SNR. This is because the speed 120 km/hr makes the channel response vary fast and greatly, causing the CP correlation to perform badly. We can also see in the figure that the estimation of timing offset for normal synchronization is more accurate than that for initial synchronization. The reason is that we not only use the information of initial synchronization, but also exponential average in the present symbol, to estimate the timing offset during normal synchronization.

In Fig. 5.7, we show how different SNRs affect the error distributions of symbol timing during initial synchronization under the Vehicular A channel. When SNR=10 dB, almost 99% of errors are under ± 8 samples, which is required by the specification ($1024/32 \times 25\% = 8$) during initial synchronization. Figure 5.8 shows the results under normal synchronization. Not surprisingly, the error rates are lower during normal synchronization. The mobile speed here is 120 km/hr.

Figure 5.9 shows the root mean-square error (RMSE) of symbol timing offset estimation in Vehicular A channel, where the RMSE is defined as $\sqrt{E\left\{|\tau - \hat{\tau}|^2\right\}}$, which is a measurement of how spread out a distribution is.

Now, we show some simulation results for symbol timing estimation using data-aided method during normal synchronization. These simulation results are obtained under 2048-FFT, bandwidth 20 MHz, and SNR 10 dB.

In Fig. 5.10, we show the timing error distribution of the two algorithms with mobil

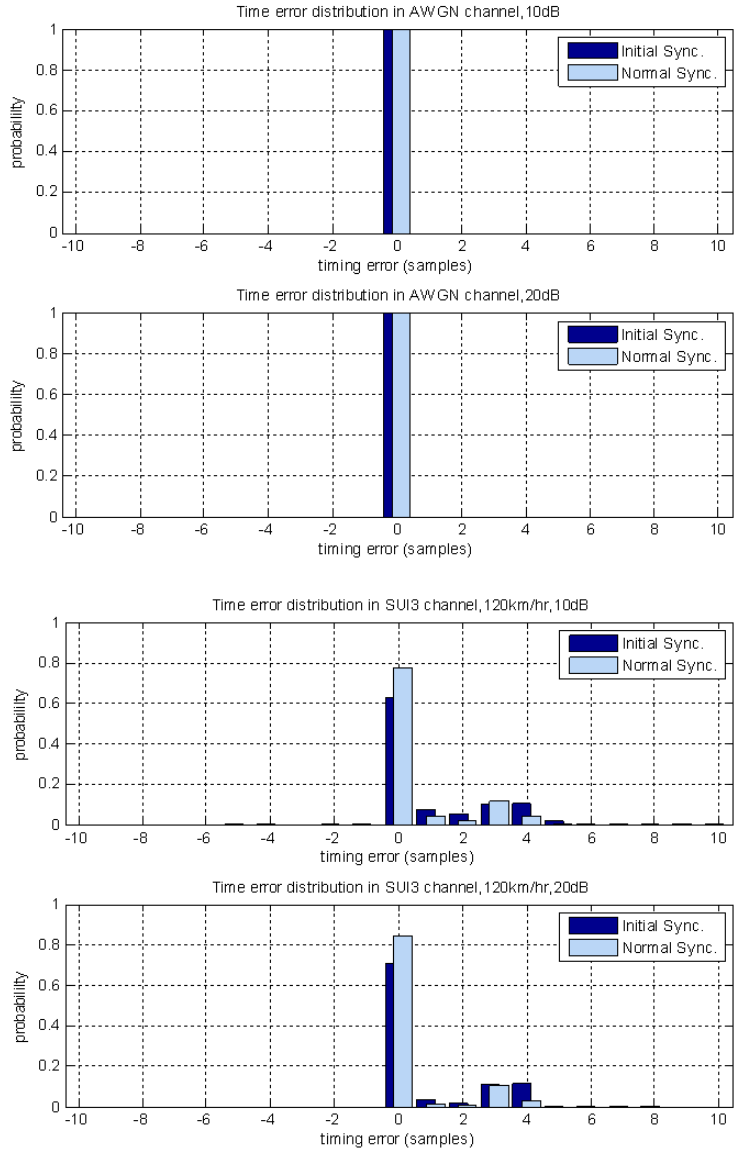


Figure 5.6: Distribution of timing offset estimation errors.

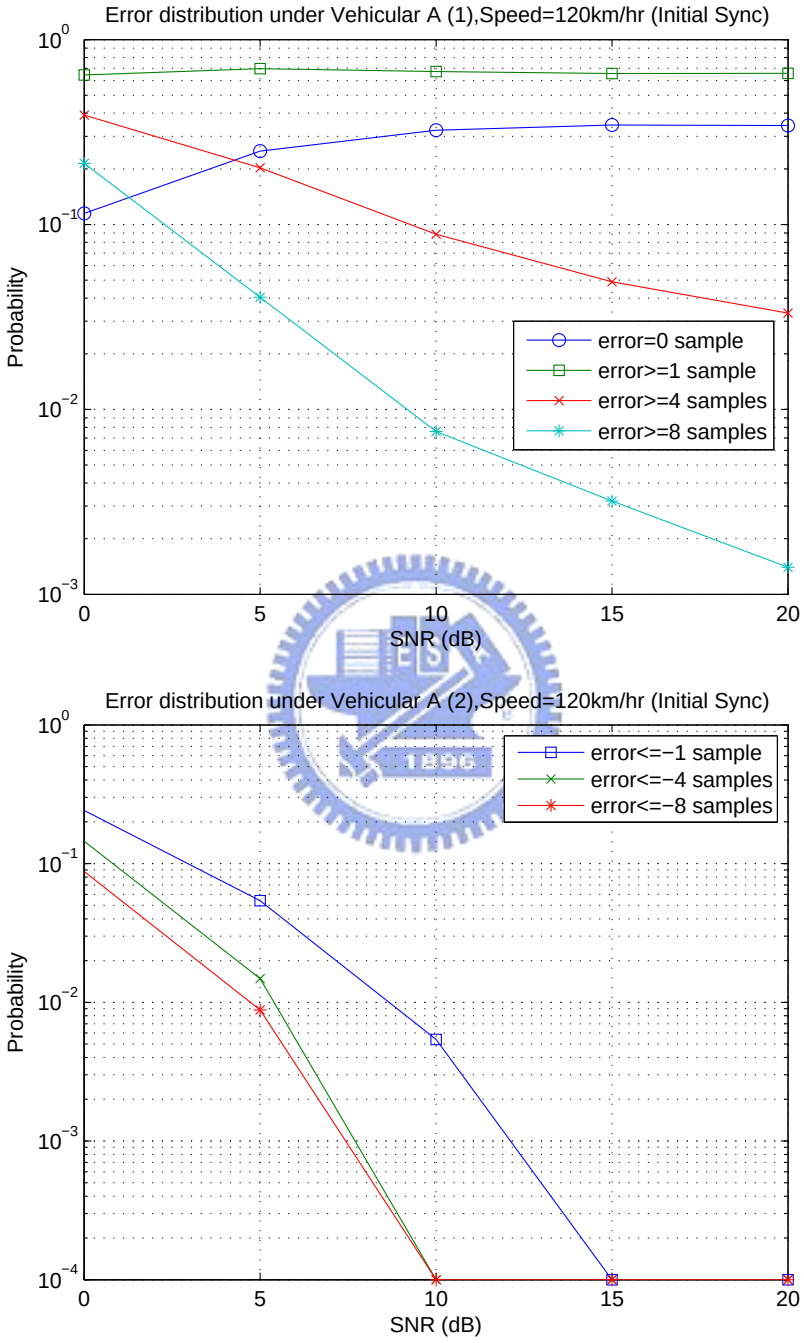


Figure 5.7: Symbol time synchronization error distribution under different SNRs (initial synchronization).

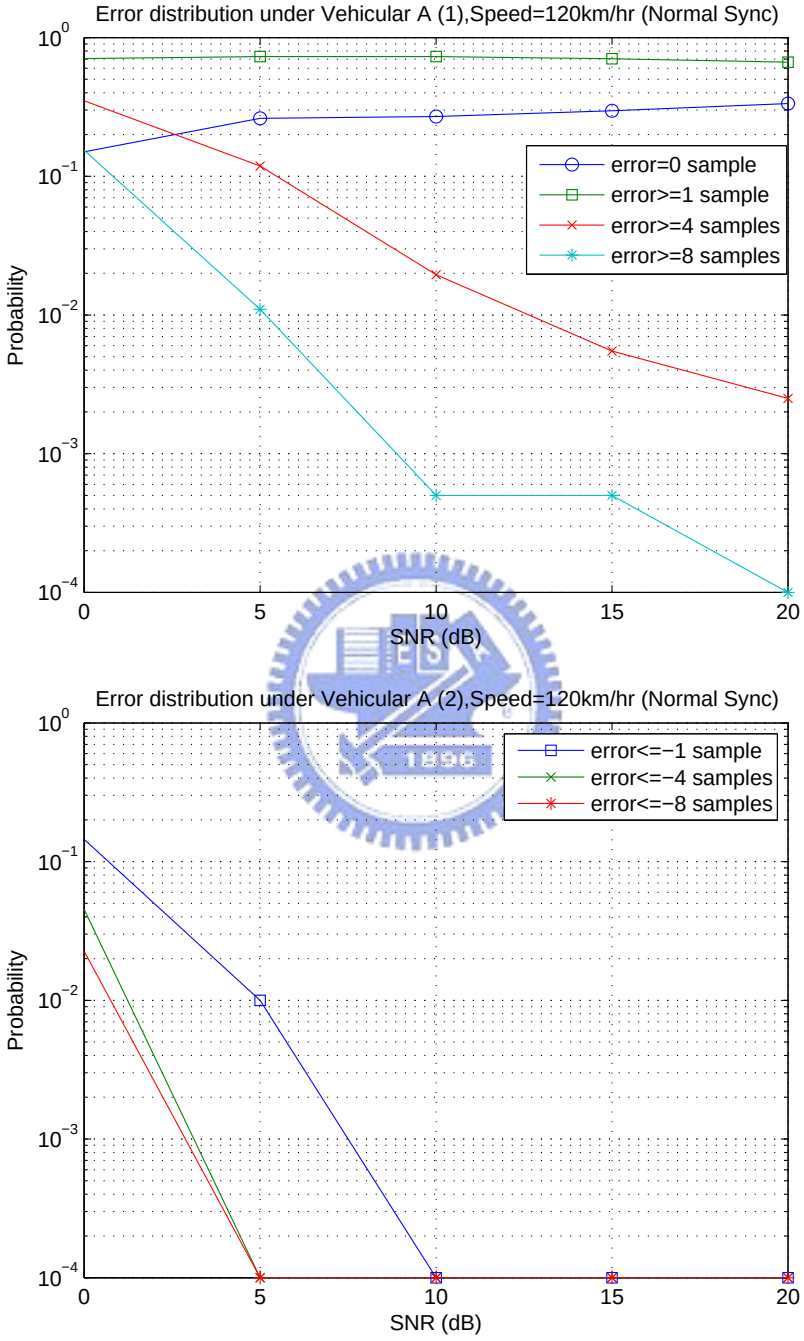


Figure 5.8: Symbol time synchronization error distribution under different SNRs (normal synchronization).

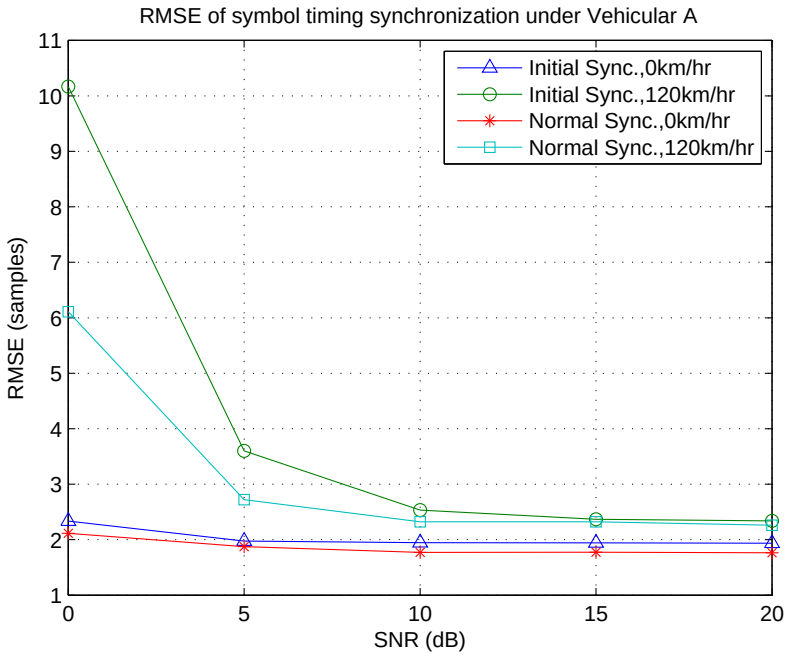


Figure 5.9: RMSE of symbol timing offset synchronization for Vehicular A channel.

speed 120 km/hr under SUI3 channel. We see that using CP correlation can get more accurate symbol timing. But when using the data-aided method, the distribution of the time synchronization errors is closely related to the power-delay profile of the SUI3 channel. Fig. 5.11 shows not only the power-delay profile of the Vehicular A channel with normal sample numbers and with normalized average power, but also the time synchronization error distribution using data-aided method under Vehicular A channel. We see that the different time offsets obtained at the synchronizer output almost coincide the sample number of the multipath delays. Furthermore, the occurrence probabilities at the different time offsets are proportional to the relative average power of the paths.

According to the simulation results, positive timing errors and ISI may result unless we find the actual timing. In order to solve this problem, we may try to find the first path after the symbol timing search. For this, we may set a threshold after we obtain all the values of $y(m)$, $m = 0, 1, ..., L-1$. Then we examine these values.The earliest symbol timing is given by the smallest value of m for which $y(m)$ exceeds the threshold. Figure 5.12 shows the simulation results with mobile speed 120 km/hr under SUI3 channel. We can see that we obtain more accurate symbol timing by using the threshold to find the

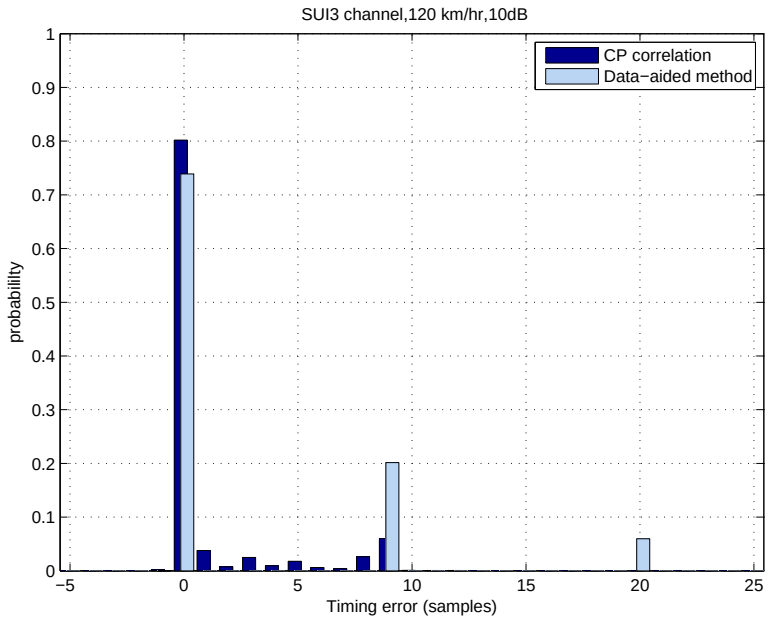


Figure 5.10: Error distributions of two algorithms during normal synchronization.

earliest timing.

5.2.2 Fractional CFO Estimation

Fig. 5.13 shows the RMSE of fractional CFO estimation under AWGN channel for both initial synchronization and normal synchronization, where the RMSE is defined as $\sqrt{E\left\{\left|\theta-\hat{\theta}\right|^2\right\}}$. It is found in the figure that the values of RMSE for normal synchronization are smaller than those for initial synchronization.

Figure 5.14 shows the RMSE of fractional CFO estimation under SUI3 channel. The top figure gives results of estimated CFO for initial synchronization and the bottom figure is results for normal synchronization. We can see that the RMSE decreases as the SNR increases. For initial synchronization when SNR exceeds 5 dB, the RMSE of higher mobile speed is bigger than the RMSE of lower mobile speed. In normal synchronization, the RMSE of higher mobile speed is always bigger than that of lower mobile speed.

We can learn how SNR affects the error distribution of carrier frequency synchronization from Fig. 5.15. It is found in the figure that when SNR = 10 dB, in 90% of the cases the correct frequency offset is under 2% of the subcarrier spacing (as required by IEEE

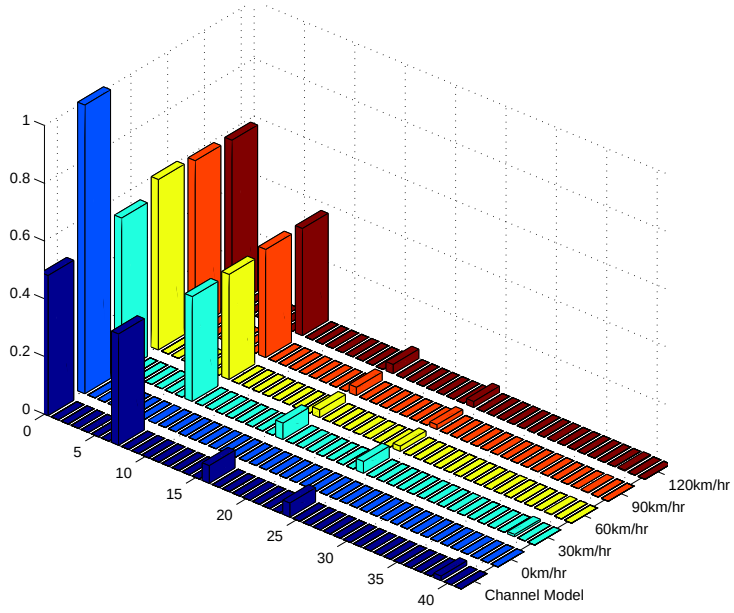


Figure 5.11: Error distributions at different speeds in Vehicular A channel.

802.16e) during initial synchronization, and in more than 98% of the cases is under 2% of the subcarrier spacing during normal synchronization. The mobile speed here is 120 km/hr, and the channel model is SUI3 channel.

5.2.3 Integer CFO Estimation and Preamble Index Identification

Now we assume that the symbol timing and fractional CFO offset are perfect estimated and compensated. The maximum search range for integer CFO is set to $\pm 12\ \Delta f$, and we let the preamble index be 31. We present results for the following five joint detection methods: the correlation method, the differential method, the reduction method, the early dropping method, and the hardlimited differential method. The threshold fraction in the early dropping method is set to $1/2$.

Figure 5.16 shows the simulation results on detection error probability under Vehicular A channel with FFT size 1024, where "error" means incorrect identification of the integer CFO or the preamble index or both. It is observed in the figure that the differential method gives the best performance in error probability. This is followed by the hardlimited differential method, the early dropping method, the reduction method, and the correlation

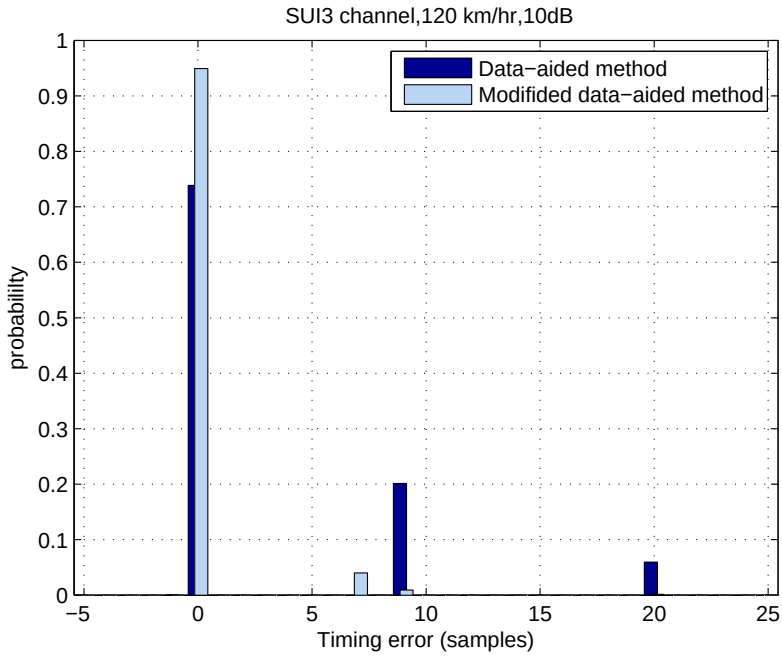


Figure 5.12: Error distributions of data-aided method and modified data-aided method during normal synchronization.

method, in that order. For the early dropping method, larger window yields lower error probabilities. For the reduction method, we can find in the figure that the the performance of reduction with differential method is almost the same as that of reduction with early dropping method. We may conclude that the coarse integer CFO estimation affects the error probability significantly from this simulation result.

Figure 5.17 shows the simulation results under SUI3 channel with FFT size 1024. Again, the differential method performs best, and the error probability for correlation method is very high. But it is observed that the probability of early dropping method with window size 20 is lower than that of hardlimited differential method under higher SNR.

Figure 5.18 shows the simulation results under SUI3 channel with FFT size 2048. Comparing it with with FFT size 1024, we see that the curves with the correlation and reduction methods are very close, but the performance of the differential method is somewhat better. This is because we do more correlations for FFT size 2048.

Now we consider the computational complexity for each method. Note that all the data we present in the following are the cases with FFT size 1024 under Vehicular A

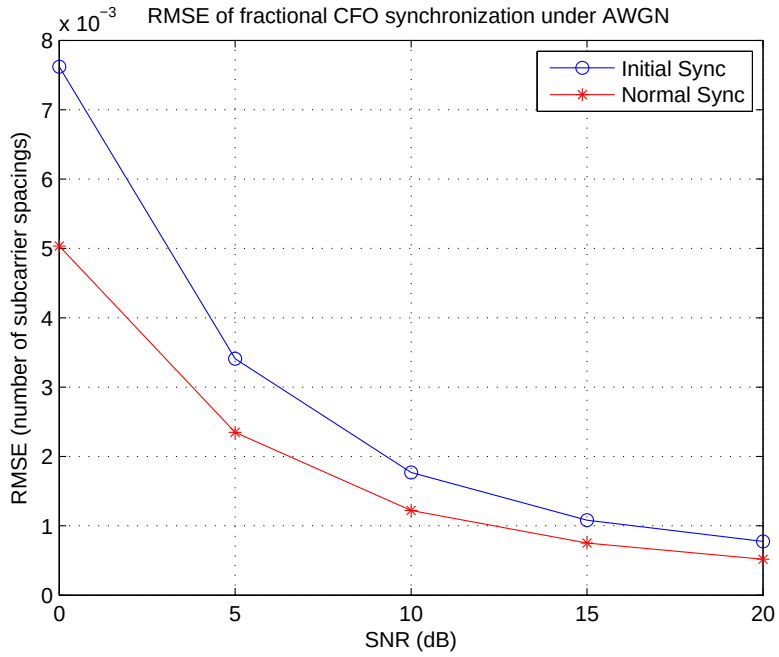


Figure 5.13: RMSE of fractional CFO under AWGN channel.

channel. First of all we consider the computational complexity of the correlation method. Since the maximum search range of integer CFO is $\pm 12\,\Delta f$, it needs $(12/3)\times 2\times 4\times(2\times 12+1)=800$ multiplications to find the coarse integer CFO [30]. Then, if multiplication by a signed binary number is also counted a multiplication operation, the amount of real multiplications to find the preamble index is equal to $114\times(284+1)=32490$. In total, it needs $800+32490=33290$ multiplications for the correlation method.

For the differential method and the early dropping method, we need to find the carrier-set used for the estimation and derive the differential sequence from the carrier-set first. It needs $293\times 2\times 3+292\times 2=2342$ multiplications for both methods to do these. Due to the previously described preamble structure, we need to search among $1+(12/3)\times 2=9$ different integer CFO conditions for each preamble index, for a total 114×9 possible CFO-preamble combinations. The amount of multiplications is equal to $2342+283\times 9\times 114=99880$ for the differential method. For the early dropping method, the amount of multiplications is related to the window size, the threshold, and the iteration numbers. We compute the number of multiplications in our simulation and average them. We can find that the amounts of multiplications are equal to $2342+11535=13877$ and $2342+$

$21572 = 23914$, respectively, for the early dropping method with window sizes 10 and 20.

For the hardlimited differential method, we need to find the carrier-set first and it costs $293 \times 2 \times 3 = 1758$ multiplications. Then we need $292 \times 2 = 584$ logic operations to derive the hardlimited differential sequence and $283 \times 9 \times 114 = 290358$ logic operations to find the integer CFO and preamble index. Therefore, it needs 1758 multiplications and $584 + 290358 = 290942$ logic operations in total.

For the reduction method, we first find the coarse integer CFO and it costs $(12/3) \times 2 \times 4 \times (2 \times 12 + 1) = 800$ multiplications. Then we assume the smaller search range for integer CFO is $\pm 3\ \Delta f$; so we need to search among $1 + (3/3) \times 2 = 3$ different integer CFO conditions for each preamble index. It needs $287 \times 2 \times 3 + 286 \times 2 = 2294$ to find the carrier-set and obtain the differential sequence. The total amount of multiplications is equal to $800 + 2294 + 283 \times 3 \times 114 = 99880$ for the reduction method with full differential search. And it costs totally $800 + 2294 + 7377 = 10471$ multiplications for the reduction method with early dropping and window size 10. Note that 3812 is a average result in our simulation.

Table 5.2 lists the amounts of multiplications required for all methods under different FFT sizes. Note that the amount of multiplications for the early dropping method needs to be obtained through simulations. As we only simulate the cases with FFT sizes 1024 and 2048, the amounts of multiplications for the early dropping method with FFT sizes 128 and 512 are not available. Complexity of the other methods under FFT sizes 128 and 512 can be obtained by calculation.

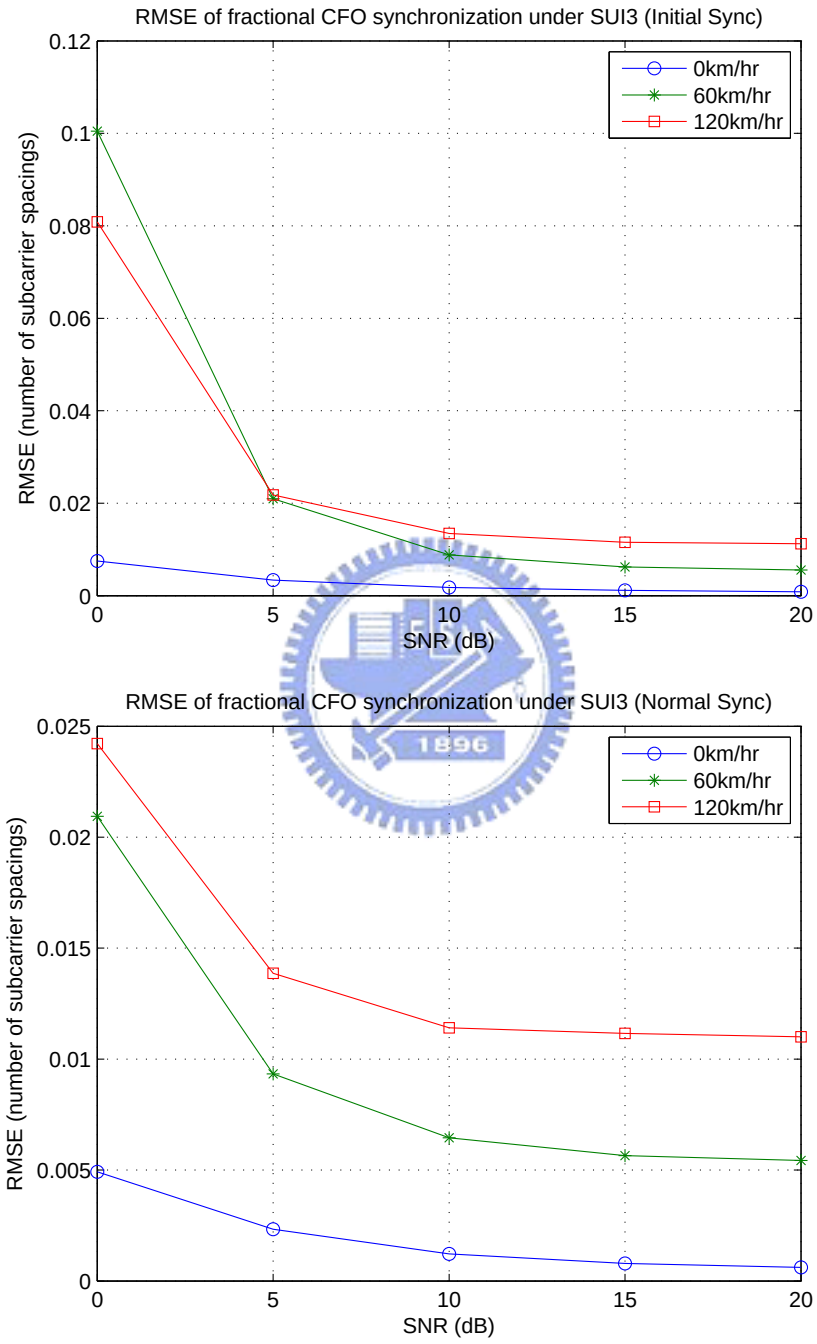


Figure 5.14: RMSE of fractional CFO under SUI3 channel.

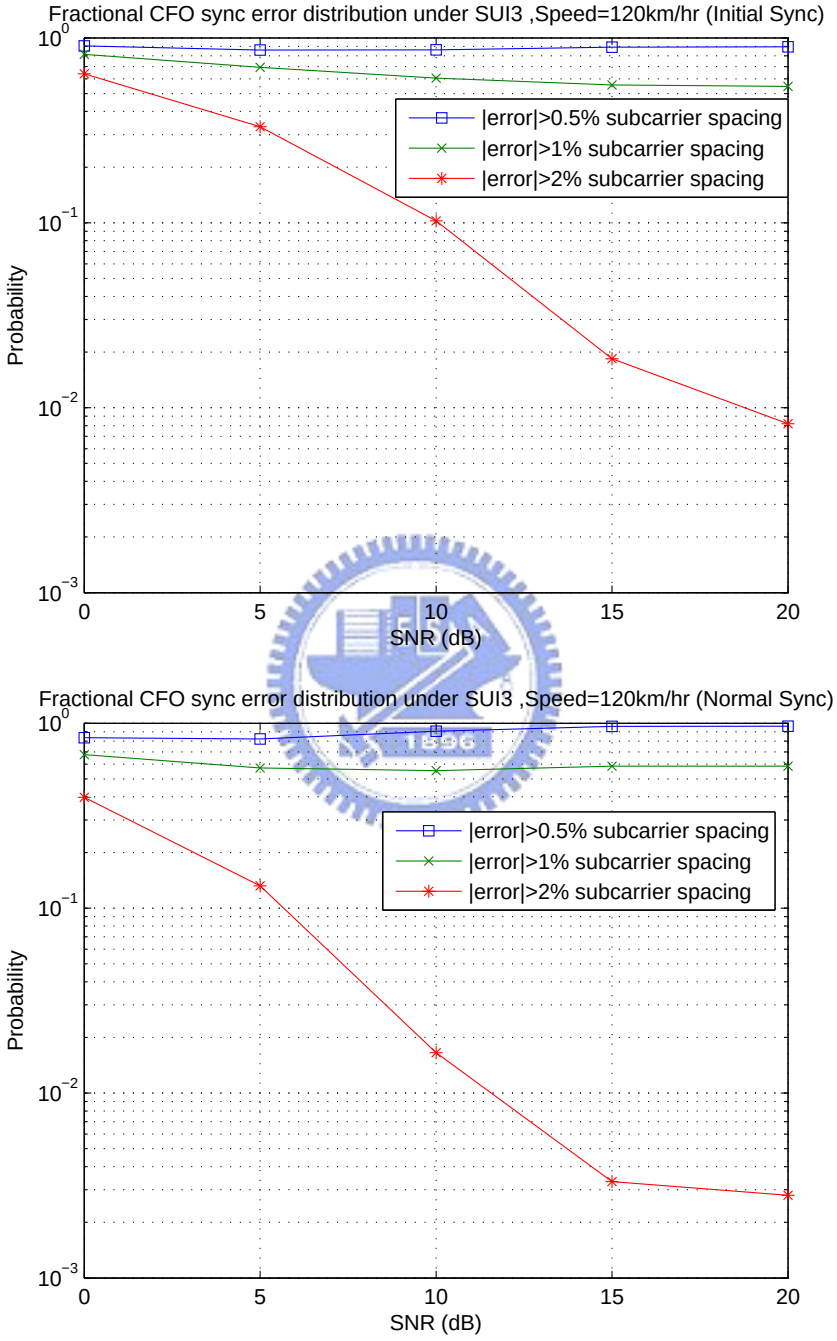


Figure 5.15: Fractional CFO synchronization error distribution under different SNRs.

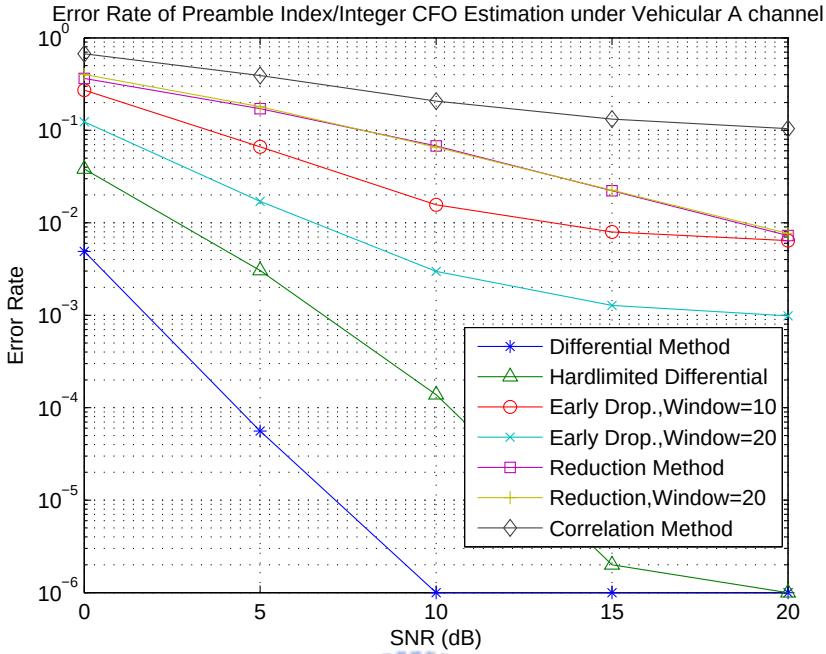


Figure 5.16: Error probability in either the identified preamble index or the estimated integer CFO under Vehicular A channel, where FFT size = 1024.

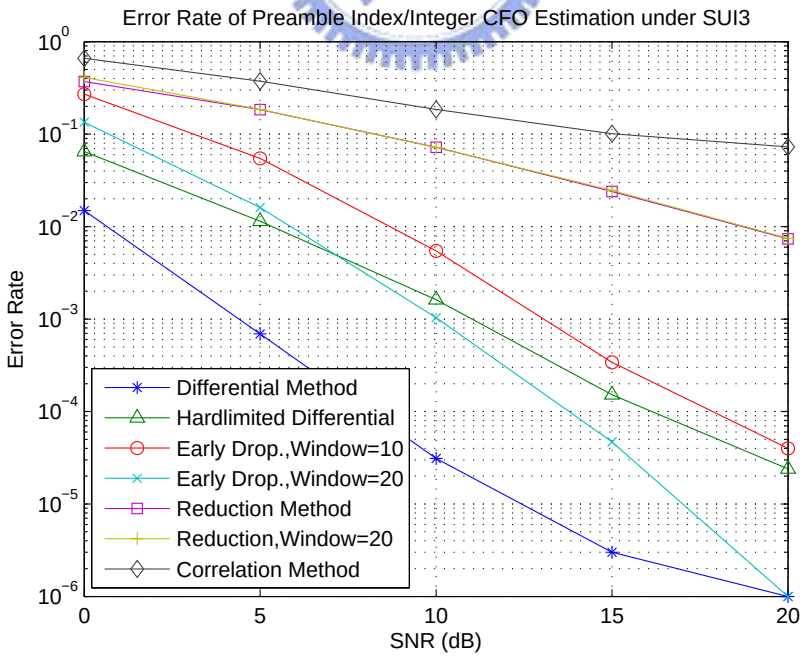


Figure 5.17: Error probability in either the identified preamble index or the estimated integer CFO under SUI3 channel, where FFT size = 1024.

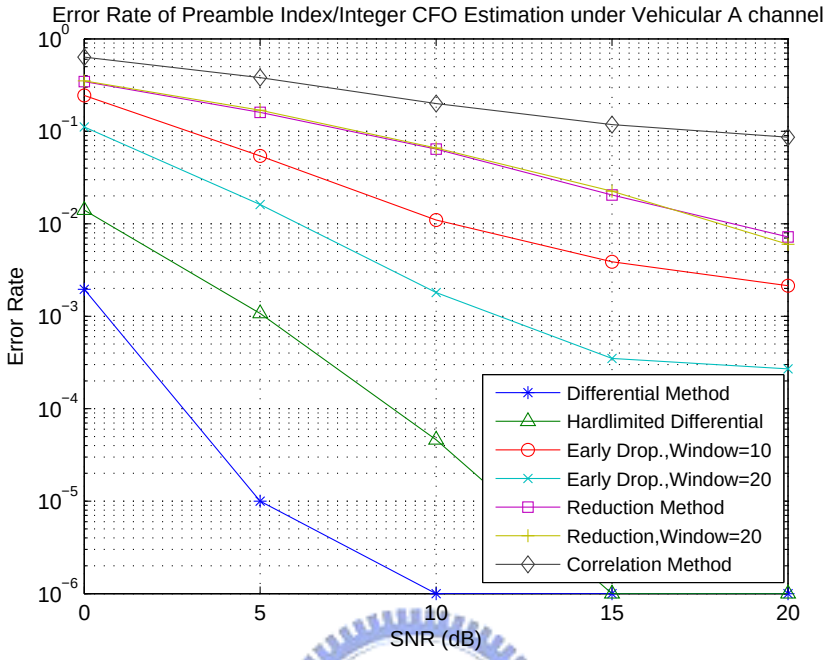


Figure 5.18: Error probability in either the identified preamble index or the estimated integer CFO under Vehicular A channel, where FFT size = 2048.

Table 5.2: Computational Complexity for Integer CFO Estimation and Preamble Index Identification

Multiplications				
FFT Size	128	512	1024	2048
Correlation method	5018	17216	33290	65666
Differential method	36268	146906	292700	586356
Hardlimited differential	270+ 35998 (logic)	912+ 145994 (logic)	1758+ 290942 (logic)	3462+ 582894 (logic)
Reduction method	13080	50530	99880	199280
Reduction with window size=20	N.A.	N.A.	10471	13551
Early drop. with window size=10	N.A.	N.A.	13877	16117
Early drop. with window size=20	N.A.	N.A.	23914	26056

Chapter 6

Fixed-Point DSP Implementation

We have introduced the synchronization techniques for IEEE 802.16e OFDMA in previous chapters. Now we consider the fixed-point implementation of the synchronization techniques on DSP in this chapter.

6.1 Fixed-Point Implementation

In algorithm development, it is often convenient to employ floating-point computation to verify the performance of the algorithms. But for power, speed, and hardware cost reasons, practical transceiver implementations normally use fixed-point computation. The DSP employed in this work, TI's TMS320C6416T, 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 C6416T 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 1 GHz, the peak performance is 8000 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 se-

lection 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 [16]. The chosen data formats are as shown in Figs. 6.1 and 6.2 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 some blocks of the system in the following subsections.

6.1.1 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, but pilots for DL preamble and data symbols are power boosted. The range of signal values of each modulation type are shown in Table 6.1. The widest range occurs in the case of BPSK, which is $[-2\sqrt{2}, 2\sqrt{2}]$. Therefore we must have at least two bits for the integer part of the signal value. With one bit for sign, there remains 13 fractional bits. Hence Q2.13 is the chosen data format, whose range covers $[-4, 4)$. It can cover the ranges of pilot and data 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 6.1: Ranges of Modulated Signal Values

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

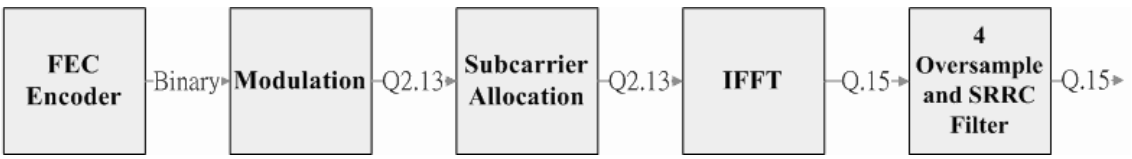


Figure 6.1: Fixed-point data formats used at different points in the transmitter.

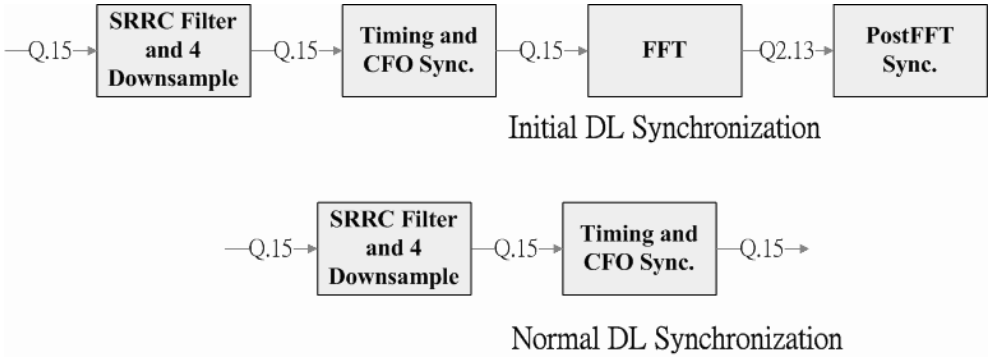


Figure 6.2: Fixed-point data formats used at different points in the receiver.

6.1.2 IFFT, FFT and SRRC Filter

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. Also, the data format of the input to and the output from the SRRC filter are the same. For efficiency reason, we employ some functions provided by TI in the DSPLIB for C64x to implement the IFFT, the FFT, and the SRRC filter. It will be introduced in the next section.

6.1.3 Synchronization

The detailed synchronization method has been presented in Chapter 5. Besides translating floating data type to short data type, here we only make two points related to fixed-point implementation:

- In fractional CFO estimation, we use a lookup table to implement the $\arctan()$ function. Since the corresponding $\arctan()$ curve for the entries covering the range $[\tan 0, \tan 0.5\pi)$ is not linear, it may lose the precision to estimate the fractional CFO if we construct a table for $\arctan()$. Hence, we construct a table for the $\arcsin()$ function to estimate the fractional CFO instead of a table for the $\arctan()$ function. The table contains 2048 entries covering the range $[\sin 0, \sin 0.25\pi]$ uniformly. By using some trigonometric calculations, we can estimate the fractional CFO precisely.

- 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.

6.2 DSP Optimization

In this section, we introduce some optimization techniques used in the system. Besides utilizing the compiler options we mentioned before, we adjust the program for compiler to software-pipeline the loop automatically. Moreover, we unroll the loop by 4 to speed up by ourselves for some cases that the compiler cannot software-pipeline. Fig. 6.3 shows an example to loop unroll a loop of compensation function. Figures 6.4 and 6.5 are the assembly code for this loop. Besides, we utilize the TI C64x DSP library that includes many C-callable, assembly optimized, and general-purpose signal-processing routines to accelerate our system. We introduce some of them in the following subsections.

6.2.1 The IFFT and FFT

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

6.2.2 Synchronization

We adopt the differential method to estimate the preamble index and integer CFO for DSP implementation. As we introduced in chapter 5, it needs to do a lot of correlations and, in the end, find the maximum correlation value with the differential method. We use the DSP library function `DSP_dotprod()` to calculate the correlation and the function `DSP_maxidx()` to find the maximum correlation value.

```
for(i=0;i<NFFT;i+=4)
{

    index = Fstate>>(12-11+2);
    index1 = Fstate1>>(12-11+2);
    index2 = Fstate2>>(12-11+2);
    index3 = Fstate3>>(12-11+2);

    Fstate=(Fstate+Foffset+16384)&(16384-1);
    Fstate1=(Fstate1+Foffset+16384)&(16384-1);
    Fstate2=(Fstate2+Foffset+16384)&(16384-1);
    Fstate3=(Fstate3+Foffset+16384)&(16384-1);

    rc = cos_tab[index];
    rs =  sin_tab[index];
    rc1 = cos_tab[index1];
    rs1 =  sin_tab[index1];
    rc2 = cos_tab[index2];
    rs2 =  sin_tab[index2];
    rc3 = cos_tab[index3];
    rs3 =  sin_tab[index3];

    RealOut[i]=(RealIn[Est_timeoffset+i]*rc
        -ImagIn[Est_timeoffset+i]*rs)>>15;
    ImagOut[i]=(ImagIn[Est_timeoffset+i]*rc
        +RealIn[Est_timeoffset+i]*rs)>>15;
    RealOut[i+1]=(RealIn[Est_timeoffset+i+1]*rc1
        -ImagIn[Est_timeoffset+i+1]*rs1)>>15;
    ImagOut[i+1]=(ImagIn[Est_timeoffset+i+1]*rc1
        +RealIn[Est_timeoffset+i+1]*rs1)>>15;
    RealOut[i+2]=(RealIn[Est_timeoffset+i+2]*rc2
        -ImagIn[Est_timeoffset+i+2]*rs2)>>15;
    ImagOut[i+2]=(ImagIn[Est_timeoffset+i+2]*rc2
        +RealIn[Est_timeoffset+i+2]*rs2)>>15;
    RealOut[i+3]=(RealIn[Est_timeoffset+i+3]*rc3
        -ImagIn[Est_timeoffset+i+3]*rs3)>>15;
    ImagOut[i+3]=(ImagIn[Est_timeoffset+i+3]*rc3
        +RealIn[Est_timeoffset+i+3]*rs3)>>15;
}
```

Figure 6.3: A part of C code for compensation function.

```
000C2444 018081A9            MVK.S1      0x0103,A3
000C2448 0280802B  ||         MVK.S2      0x0100,B5
000C244C 0C1B005A  ||         SUB.L2      B6,8,B24
000C2450 018D2C81             MPY.M1      A9,A3,A3
000C2454 0301002A  ||         MVK.S2      0x0200,B6
000C2458 0202002A             MVK.S2      0x0400,B4
000C245C 0D0E5208             EXTU.S1     A3,18,18,A26
000C2460          L14:
000C2460 0200006B             MVKH.S2     0x0000,B4
000C2464 0018205B  ||         ADD.L2      1,B6,B0
000C2468 0294E843  ||         ADD.D2      B5,B7,B5
000C246C 0000A359  ||         MVK.L1      0,A0
000C2470 0B8A0029  ||         MVK.S1      0x1400,A23
000C2474 01800040  ||         MVK.D1      0,A3
000C2478 0D8403E3             MVC.S2      CSR,B27
000C247C 000000E9  ||         MVKH.S1     0x10000,A0
000C2480 0B471058  ||         SUB.L1X     B17,8,A22
000C2484 0296104B             EXT.S2      B5,16,16,B5
000C2488 0FEFCF5B  ||         AND.L2      -2,B27,B31
000C248C 0B800069  ||         MVKH.S1     0x0000,A23
000C2490 0A901FD8  ||         OR.L1X      0,B4,A21
000C2494 0083F059             SUB.L1X     B0,1,A1
000C2498 00FC03A3  ||         MVC.S2      B31,CSR
000C249C 0B24BA43  ||         ADDAH.D2    B9,B5,B22
000C24A0 04C7405B  ||         SUB.L2      B17,6,B9
000C24A4 010DEF88  ||         SET.S1      A3,15,15,A2
000C24A8 0BC0BA42             ADDAH.D2    B16,B5,B23
000C24AC          L15:
000C24AC 0D4DEDA3             SHR.S2      B19,0xf,B26
000C24B0 041CB483  ||         MPYHL.M2X   B5,A7,B8
000C24B4 031A7C81  ||         MPY.M1X     A19,B6,A6
000C24B8 D9DC37A6  || [!A0]   LDNDW.D2T2  *B23++[0x1],B19:B18
000C24BC BD1A5209     [!A2]   EXTU.S1     A6,18,18,A26
000C24C0 08232483  ||         MPYHL.M2    B25,B8,B16
000C24C4 094E40FB  ||         SUB.L2      B18,B19,B18
000C24C8 0218F881  ||         MPYLH.M1X   A7,B6,A4
000C24CC DAD837A6  || [!A0]   LDNDW.D2T2  *B22++[0x1],B21:B20
000C24D0 0294D483             MPYHL.M2X   B6,A5,B5
000C24D4 0310C079  ||         ADD.L1      A6,A4,A6
000C24D8 0349EDA3  ||         SHR.S2      B18,0xf,B6
000C24DC B86092D7  || [!A2]   STH.D2T2    B16,*++B24[0x4]
000C24E0 D1DD4A45  || [!A0]   LDH.D1T1    *+A23[A10],A3
000C24E4 03E9B008  ||         EXTU.S1     A26,13,16,A7
000C24E8 0DE80FD9             OR.L1       0,A26,A27
000C24EC 03C6007B  ||         ADD.L2      B16,B17,B7
000C24F0 0394B483  ||         MPYHL.M2X   B5,A5,B7
000C24F4 0299EDA1  ||         SHR.S1      A6,0xf,A5
000C24F8 D410EAC7  || [!A0]   LDH.D2T2    *+B4[B7],B8
000C24FC D2554A44  || [!A0]   LDH.D1T1    *+A21[A10],A4
000C2500 2003E05B     [ B0]   SUB.L2      B0,1,B0
000C2504 02950843  ||         ADD.D2      B5,B8,B5
000C2508 041DEDA3  ||         SHR.S2      B7,0xf,B8
000C250C 03692079  ||         ADD.L1      A9,A26,A6
000C2510 DFC65209  || [!A0]   EXTU.S1     A17,18,18,A31
000C2514 D35C6A46  || [!A0]   LDH.D1T2    *+A23[A3],B6
000C2518 B36022D7     [!A2]   STH.D2T2    B6,*+B24[0x1]
000C251C B4589255  || [!A2]   STH.D1T1    A8,*++A22[0x4]
000C2520 089C92FB  ||         SUB.L2X     A4,B7,B17
000C2524 DC4A5209  || [!A0]   EXTU.S1     A18,18,18,A24
000C2528 03C806A3  ||         OR.S2       0,B18,B7
000C252C 08FD2078  ||         ADD.L1      A9,A31,A17
000C2530 B42492D7     [!A2]   STH.D2T2    B8,*++B9[0x4]
000C2534 2FFFF193  || [ B0]   B.S2        L15
000C2538 DCC25209  || [!A0]   EXTU.S1     A16,18,18,A25
000C253C 02CE7C83  ||         MPY.M2X     B19,A19,B5
000C2540 01D69C81  ||         MPY.M1X     A20,B21,A3
000C2544 D3DCEA45  || [!A0]   LDH.D1T1    *+A23[A7],A7
000C2548 0CD00FDB  ||         OR.L2       0,B20,B25
000C254C 09612078  ||         ADD.L1      A9,A24,A18
000C2550 0815EDA3             SHR.S2      B5,0xf,B16
000C2554 BD6042D7  || [!A2]   STH.D2T2    B26,*+B24[0x2]
000C2558 D2D4EA45  || [!A0]   LDH.D1T1    *+A21[A7],A5
000C255C 090E9C83  ||         MPY.M2X     B20,A3,B18
000C2560 021C7C81  ||         MPY.M1X     A3,B7,A4
```

Figure 6.4: A part of assembly code for compensation function-I.

```
000C2560 021C7C81  ||         MPY.M1X       A3,B7,A4
000C2564 086S2079  ||         ADD.L1        A9,A25,A16
000C2568 03E5B008  ||         EXTU.S1       A25,13,16,A7
000C256C 0EFC0FD9             OR.L1         0,A31,A29
000C2570 08925C83  ||         MPY.M2X       B18,A4,B17
000C2574 03E49C81  ||         MPY.M1X       A4,B25,A7
000C2578 89D4EA45  || [ A1]   LDH.D1T1      *+A21[A7],A19
000C257C 0561B008  ||         EXTU.S1       A24,13,16,A10
000C2580 C0004F01     [ A0]   MPYSU.M1      2,A0,A0
000C2584 0A45EDA3  ||         SHR.S2        B17,0xf,B20
000C2588 0F600FD9  ||         OR.L1         0,A24,A30
000C258C 099472FB  ||         SUB.L2X       A3,B5,B19
000C2590 091B2483  ||         MPYHL.M2      B25,B6,B18
000C2594 02CC08F3  ||         OR.D2         0,B19,B5
000C2598 8A5CEA45  || [ A1]   LDH.D1T1      *+A23[A7],A20
000C259C 01FDB008  ||         EXTU.S1       A31,13,16,A3
000C25A0 A1084F01     [ A2]   MPYSU.M1      2,A2,A2
000C25A4 B2D84255  || [!A2]   STH.D1T1      A5,*+A22[0x2]
000C25A8 BA6062D7  || [!A2]   STH.D2T2      B20,*+B24[0x3]
000C25AC 0E640FD9  ||         OR.L1         0,A25,A28
000C25B0 0898E483  ||         MPYHL.M2      B7,B6,B17
000C25B4 08C640FB  ||         SUB.L2        B18,B17,B17
000C25B8 0210E1E1  ||         ADD.S1        A7,A4,A4
000C25BC 035406A2  ||         OR.S2         0,B21,B6
000C25C0 8087E059     [ A1]   SUB.L1        A1,1,A1
000C25C4 B8586257  || [!A2]   STH.D1T2      B16,*+A22[0x3]
000C25C8 09A0E483  ||         MPYHL.M2      B7,B8,B19
000C25CC 0845EDA3  ||         SHR.S2        B17,0xf,B16
000C25D0 0411EDA1  ||         SHR.S1        A4,0xf,A8
000C25D4 02169C81  ||         MPY.M1X       A20,B5,A4
000C25D8 038C1FDA  ||         OR.L2X        0,A3,B7
000C25DC                DW$L$_Compensate$5$E:
000C25DC 041CB483             MPYHL.M2X     B5,A7,B8
000C25E0 031A7C81  ||         MPY.M1X       A19,B6,A6
000C25E4 0D4DEDA2  ||         SHR.S2        B19,0xf,B26
000C25E8 02781FDB             OR.L2X        0,A30,B4
000C25EC 08232483  ||         MPYHL.M2      B25,B8,B16
000C25F0 094E45E3  ||         SUB.S2        B18,B19,B18
000C25F4 0218F881  ||         MPYLH.M1X     A7,B6,A4
000C25F8 0D1A5208  ||         EXTU.S1       A6,18,18,A26
000C25FC 0294D483             MPYHL.M2X     B6,A5,B5
000C2600 0310C079  ||         ADD.L1        A6,A4,A6
000C2604 086092D7  ||         STH.D2T2      B16,*++B24[0x4]
000C2608 0349EDA2  ||         SHR.S2        B18,0xf,B6
000C260C 036022D7             STH.D2T2      B6,*+B24[0x1]
000C2610 04589255  ||         STH.D1T1      A8,*++A22[0x4]
000C2614 0DE80FD9  ||         OR.L1         0,A26,A27
000C2618 03C6007B  ||         ADD.L2        B16,B17,B7
000C261C 0299EDA1  ||         SHR.S1        A6,0xf,A5
000C2620 0394B482  ||         MPYHL.M2X     B5,A5,B7
000C2624 02A0A07B             ADD.L2        B5,B8,B5
000C2628 041DEDA2  ||         SHR.S2        B7,0xf,B8
000C262C 0815EDA3             SHR.S2        B5,0xf,B16
000C2630 089C92FB  ||         SUB.L2X       A4,B7,B17
000C2634 042492D6  ||         STH.D2T2      B8,*++B9[0x4]
000C2638 0D6042D7             STH.D2T2      B26,*+B24[0x2]
000C263C 0A45EDA2  ||         SHR.S2        B17,0xf,B20
000C2640 00EC03A3             MVC.S2        B27,CSR
000C2644 02D84255  ||         STH.D1T1      A5,*+A22[0x2]
000C2648 0A6062D6  ||         STH.D2T2      B20,*+B24[0x3]
000C264C 08586256             STH.D1T2      B16,*+A22[0x3]
000C2650 0E80025C             STH.D2T1      A29,*+DP[0x2]
000C2654 0E00035C             STH.D2T1      A28,*+DP[0x3]
000C2658 0200005E             STH.D2T2      B4,*+DP[0x0]
000C265C 0D80045C             STH.D2T1      A27,*+DP[0x4]
```

Figure 6.5: A part of assembly code for compensation function-II.

6.3 Fixed-Point Simulation Results

We present some simulation results in this section. All simulation parameters and environments are similar to those in Section 5.2, but here we have fixed-point implementation. For comparison, floating-point simulation results are also presented together with the fixed-point results. Most simulation results in this section use FFT size 2048 with bandwidth 20 MHz (if not mentioned), but we also show some simulation results of FFT size 2048 with bandwidth 10 MHz for comparison.

6.3.1 Symbol Timing Estimation

Figure. 6.6 shows the RMSE of symbol timing offset estimation in AWGN channel for both initial synchronization and normal synchronization. As we discussed in Section 5.2, the performance of normal synchronization is better than that of initial synchronization.

In Fig. 6.7, we see the error distributions of symbol timing estimation under different SNRs during initial synchronization under SUI3 channel. When SNR = 10 dB, it is seen that 99.7% of errors for both floating-point and fixed-point implementations are under ± 16 samples, the requirement of the specification ($2048/32 \times 25\% = 16$) during initial synchronization. Also, Fig. 6.8 shows the results under normal synchronization. The error rates are lower during normal synchronization. The mobile speed here is 120 km/hr.

We can see that all curves of fixed-point simulations are very close to those of floating-point simulations when we estimate the symbol timing.

6.3.2 Fractional CFO Estimation

Figure 6.9 shows the RMSE of fractional CFO estimation in AWGN channel for both initial and normal synchronization. Again, the performance for normal synchronization is better than initial synchronization.

Figures 6.10 and 6.11 show the RMSE of fractional CFO estimation under SUI3 channel for FFT size 2048 with bandwidths 20 MHz and 10 MHz, respectively. From these figures, we can see that the RMSE for bandwidth 10 MHz is much higher than that for bandwidth 20 MHz. When mobile speed is 120 km/hr, the RMSE for bandwidth 10MHz is higher than 2% of subcarrier spacing even in high SNRs. This is because under same

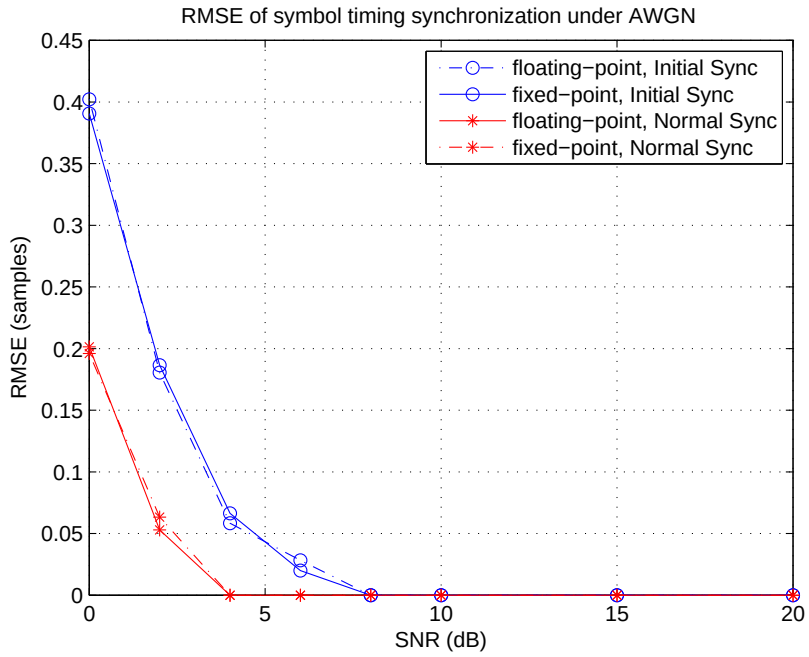


Figure 6.6: RMSE of symbol timing offset estimation in AWGN with fixed-point and floating-point implementation.

FFT size, subcarrier spacing for smaller bandwidth is narrower, and the system becomes more sensitive to Doppler shift. We may conclude that it is suitable to use bandwidth 20 MHz for FFT size 2048 as mentioned in Table 2.2 as proposed by the WiMAX Forum [8].

We can learn how the SNR affects the error distribution of carrier frequency synchronization from Fig. 6.12. It is seen that when SNR = 10 dB, about 99% of the cases the corrected frequency offset is under 2% of the subcarrier spacing, as required by IEEE 802.16e, for both initial and normal synchronization.

From the above figures on fractional CFO estimation results, we can also see that the performance curves for fixed-point and floating-point implementations are almost the same.

6.3.3 Integer CFO Estimation and Preamble Index Identification

Figure 6.13 shows the probability of either integer CFO or preamble index identification error with fixed-point and floating-point implementations under Vehicular A channel. The floating-point version has better performance when the SNR is under 10 dB, but within

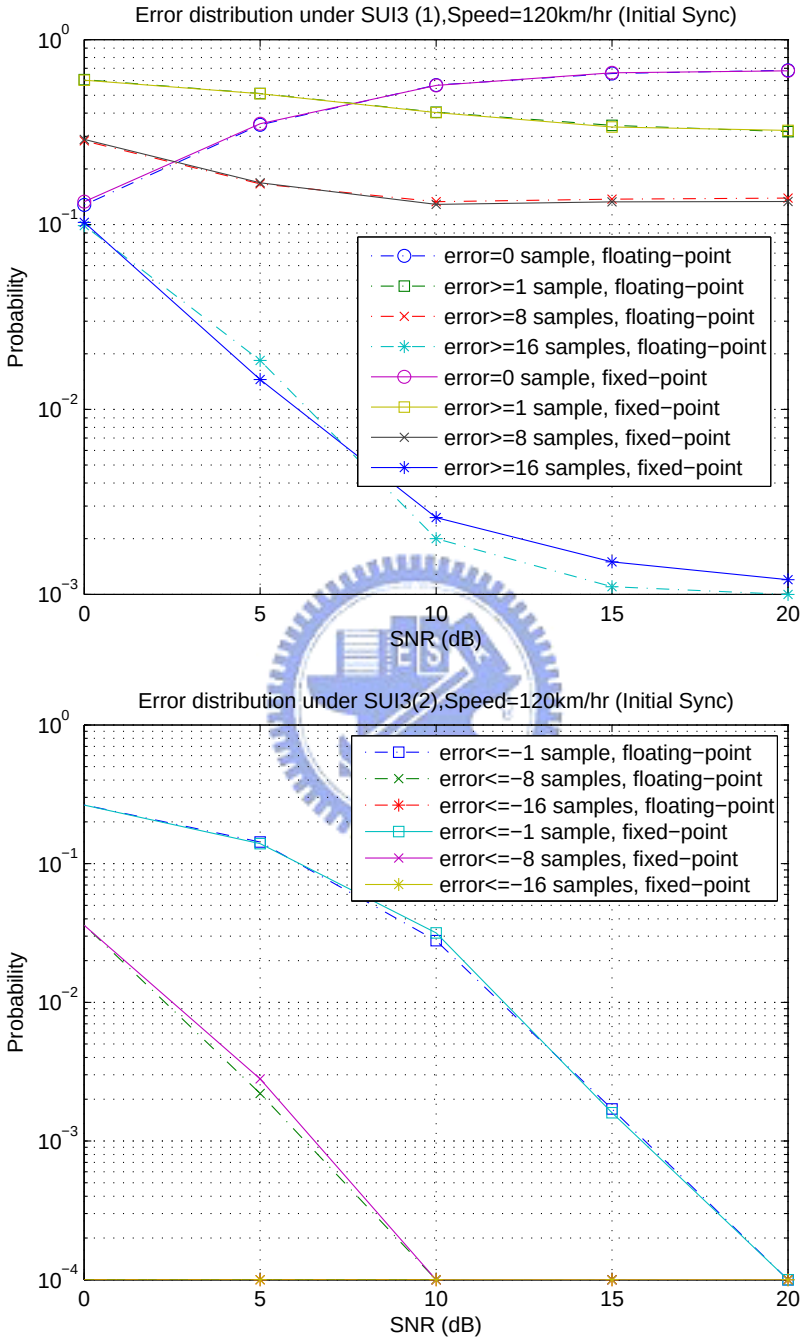


Figure 6.7: Symbol time synchronization error distribution under different SNRs (initial synchronization).

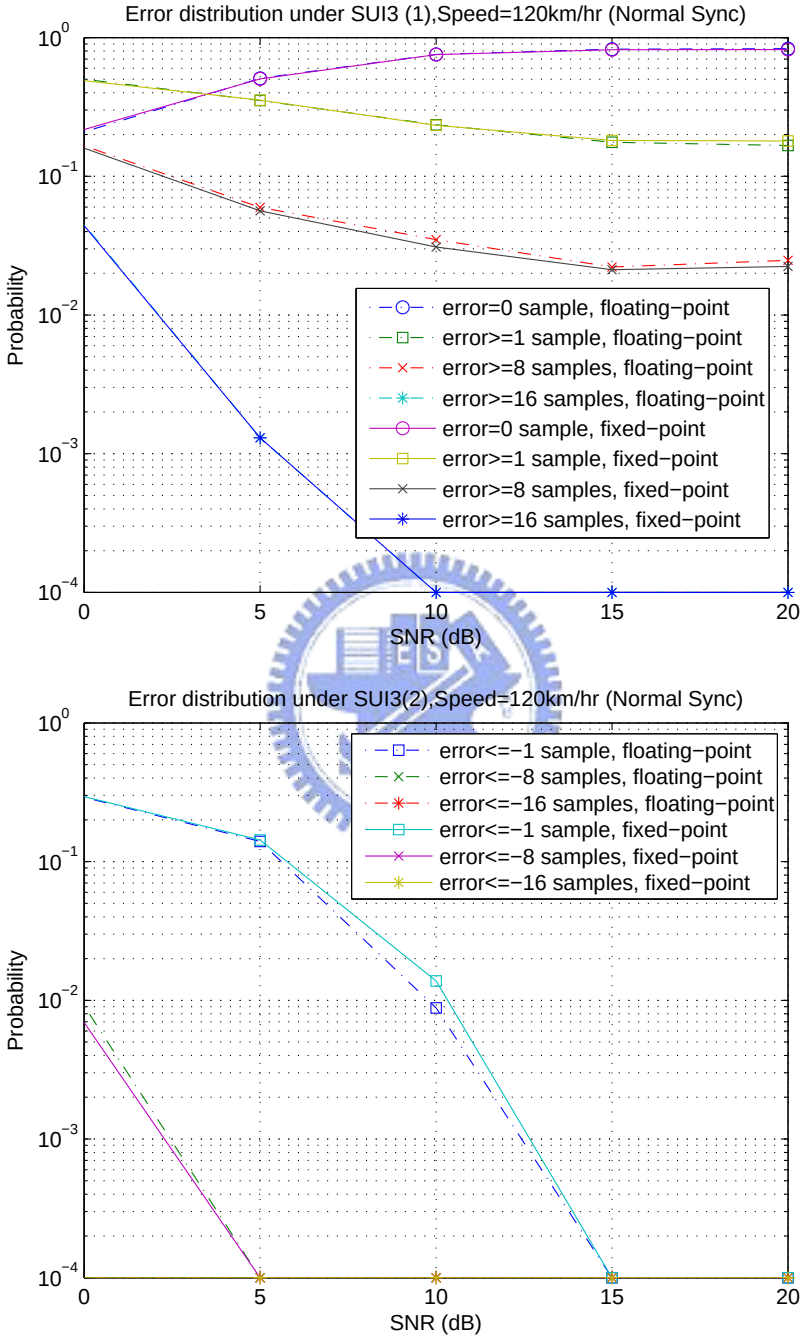


Figure 6.8: Symbol time synchronization error distribution under different SNRs (normal synchronization).

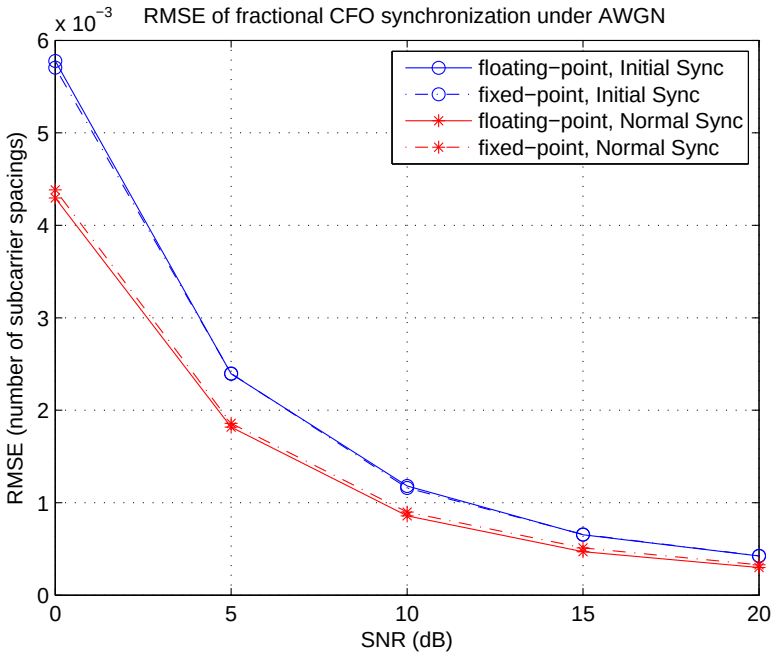


Figure 6.9: RMSE of fractional CFO estimation in AWGN with fixed-point and floating-point implementation.

an acceptable difference.

6.4 DSP Optimization Results

In our system, the clock frequency of DSP is 1 GHz and one subframe duration is 2.5 ms for both FFT size 1024 with bandwidth 10 MHz and FFT size 2048 with bandwidth 20 MHz. Therefore, the available execution clock cycles are 2,500,000 in a subframe duration. To achieve real-time processing speed, one subframe must execute less than 2,500,000 instruction cycles.

Table 6.2 shows the number of clock cycles for each function used in the synchronization procedures, 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.) For FFT size 1024, the total requirement for initial synchronization is approximately 0.1239

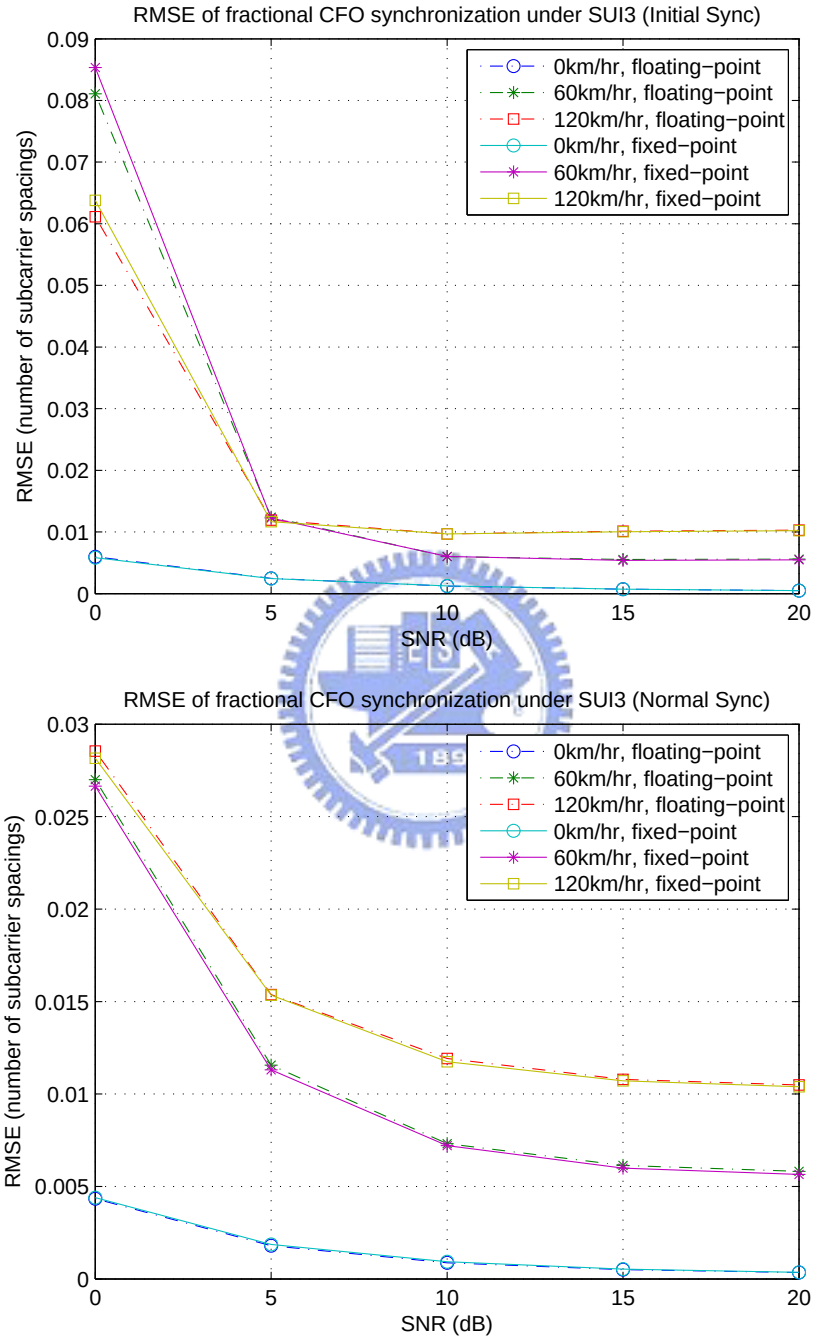


Figure 6.10: RMSE of fractional CFO under SUI3 channel for FFT size 2048 with bandwidth 20 MHz.

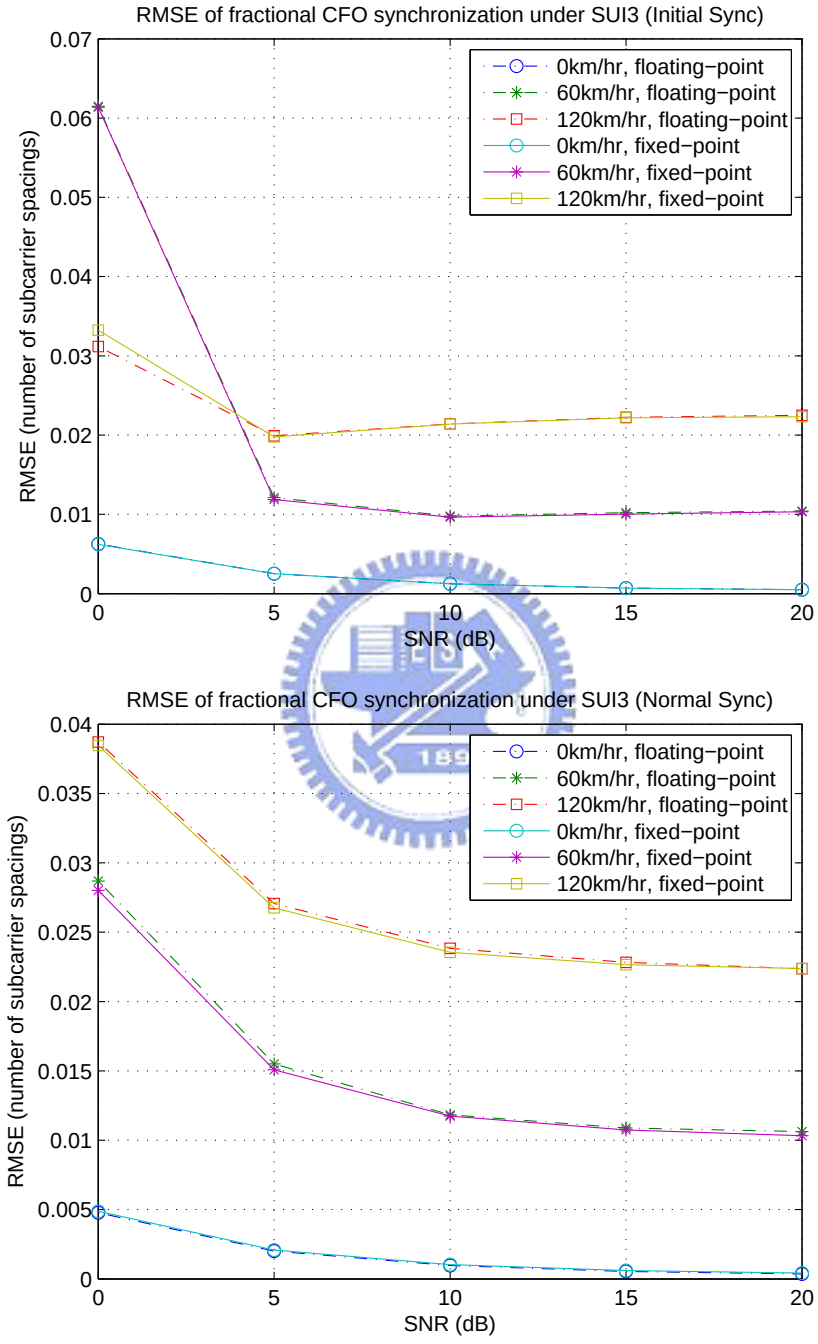


Figure 6.11: RMSE of fractional CFO under SUI3 channel for FFT size 2048 with bandwidth 10 MHz.

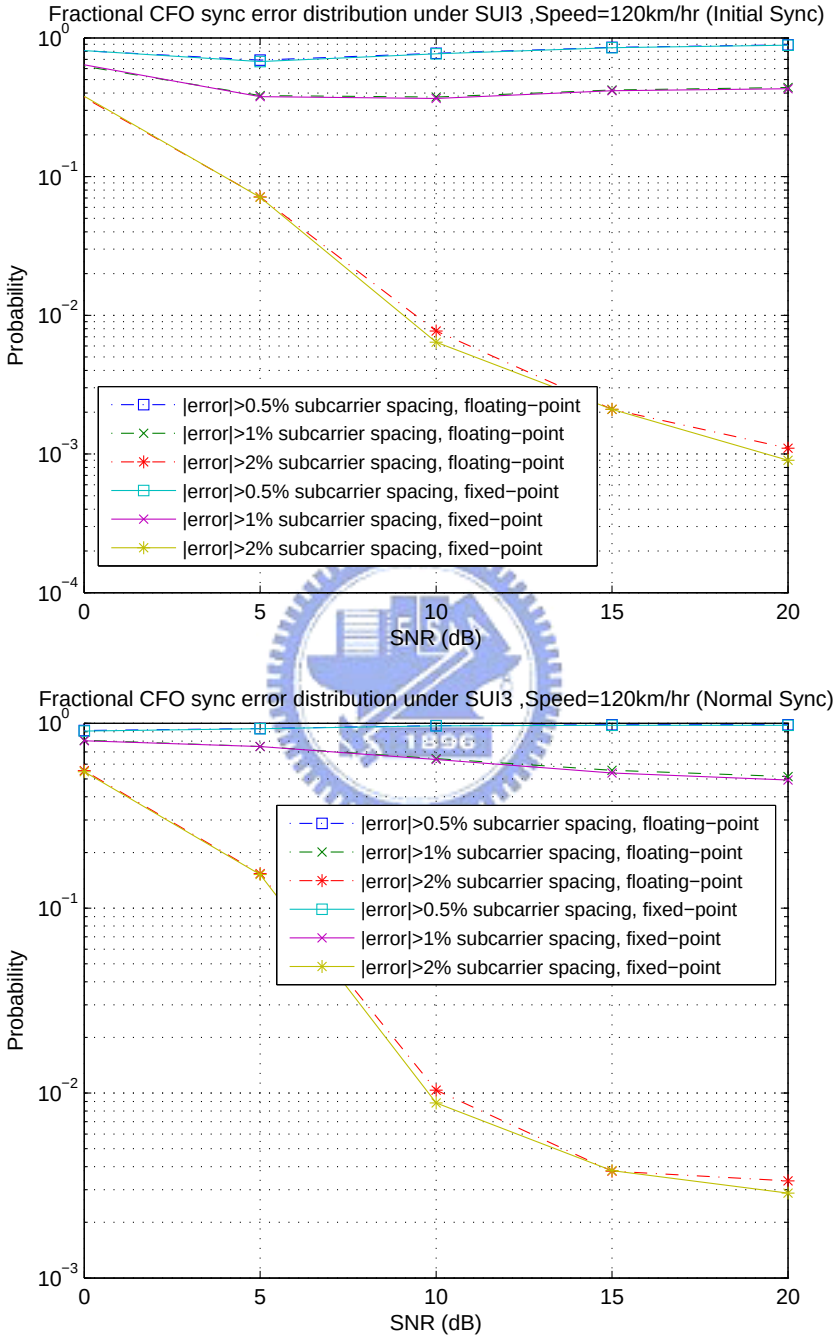


Figure 6.12: Fractional CFO synchronization error distribution under different SNRs.

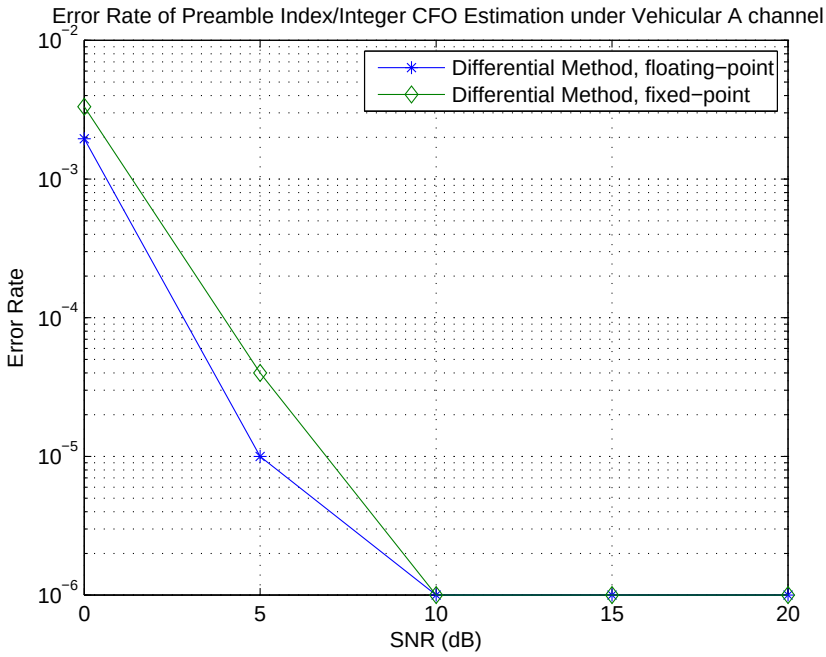


Figure 6.13: Error probability in either the identified preamble index or the estimated integer CFO with fixed-point and floating-point implementation under Vehicular A channel.

DSP chips' processing power, while that for normal synchronization is 0.0926 DSP chips' power. For FFT size 2048, the total requirement for initial synchronization is approximately 0.2295 DSP chips' processing power, while that for normal synchronization is 0.1373 DSP chips' power.

Using the above results for FFT sizes 1024 and 2048, we may extrapolate for the DSP computational loads for the cases of FFT sizes 128 and 512. Assume that the subframe duration is also 2.5 ms, that is, the bandwidth for FFT size 128 is 1.25 MHz and the that for FFT size 512 is 5 MHz. After rough estimation, the DSP loading for FFT size 128 is 0.0326 and 0.05 for initial synchronization and normal synchronization, respectively. For FFT size 512, the initial synchronization needs 0.072 DSPs and the normal synchronization needs 0.07 DSPs.

Table 6.2: DSP Optimization Results

	Function	Avg. Clock Cycles		DSP Computational Load (1 DL subframe / 2.5ms)	
		1024 (10 MHz)	2048 (20 MHz)	1024 (10 MHz)	2048 (20 MHz)
Initial Sync.	Correlation	28620	47729	0.1239	0.2295
	Estimator	4035	3911		
	Compensation	4130	8127		
	FFT	11456	27122		
	PostFFTSync	260013	485351		
	Correlation Compensation	1486	1594		
Normal Sync.	Correlation (tracking)	35561	51181	0.0926	0.1373
	Estimator (tracking)	98371	98164		
	Compensation	96087	192475		
	Correlation Compensation	1484	1542		

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we first presented the synchronization techniques of the IEEE 802.16e OFDMA TDD system, and verified them through floating-point computation. Second, we modified them to fixed-point version and compare the difference of the performance between floating-point and fixed-point versions. In the end, we implemented them on TI's C6416T digital signal processor.

We considered two DL synchronization conditions: initial synchronization and normal synchronization. There were timing offset, CFO, and preamble index needed to be estimated during initial synchronization. We used the information of the first DL subframe to estimate the more accurate timing and fractional CFO by CP correlation [29]. Then, we considered the joint detection of integer CFO and preamble index using the compensated preamble symbol in the frequency domain. Several methods of different complexity were used and simulation results showed that the differential method performed rather well even in low SNR.

During DL normal synchronization, we tracked the timing offset and fractional CFO in the receiver. Again, we used the CP correlation to estimate the timing and CFO. We employed exponential average over the symbols in the subframe to obtain a more accurate estimation. More than 99% of the results can reach the requirement of the specification if the SNR was more than 10dB when mobile speed was 120 km/hr under SUI3 channel. Furthermore, the performance of normal synchronization performed better than that of initial synchronization.

Also, we proposed a data-aided method to estimate the symbol timing offset during normal synchronization. But we can only find the timing offset related to the power delay profile of the channel in the beginning, and it causes ISI. We modified it with an adaptive threshold to find the earliest possible timing, and the simulation results showed that the performance became much better.

In the end, we modified the whole system to fixed-point version, and used some optimization techniques to accelerate each function of synchronization as fast as we can on TI's DSP. Simulation results showed that all curves of fixed-point simulations were very close to those of floating-point simulations. After DSP optimization, the synchronization tasks achieved the real-time requirements.

7.2 Future Work

There are several possible extensions for our research:

- Consider the ranging process for UL synchronization since we do not lay stress on UL synchronization in this thesis.
- 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.
- Try to implement the complexity-reduced methods to estimate the integer CFO and preamble index on DSP since we only implement the differential method on DSP in this thesis.
- 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] 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.

[2] IEEE Std 802.16e-2005 and IEEE Std 802.16-2004/Cor1-2005, *IEEE Standard for Local and Metropolitan Area Networks — Part 16: Air Interface for Fixed Broadband Wireless Access Systems — Amendment 2: Physical and Medium Access Control Layers for Combined Fixed and Mobile Operation in Licensed Bands and Corrigendum 1*. New York: IEEE, Feb. 28, 2006.

[3] R. van Nee and R. Prasad, *OFDM for Wireless Multimedia Communications.* Boston: Artech House, 2000.

[4] 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.

[5] 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.

[6] J. Puthenkulam, and M. Goldhammer, "802.16 overview and coexistence aspects," http://grouper.ieee.org/groups/802/secmail/ppt00009.ppt.

[7] V. Bykovnikov, "The advantages of SOFDMA for WiMAX,"

http://mail.com.nthu.edu.tw/ jmwu/LAB/SOFDMA-for-WiMAX.pdf.

[8] WiMAX Forum, "Mobile WiMAX — Part 1: A technical overview and performance evalution," June 2006,

http://www.wimaxforum.org/news/downloads/Mobile_WiMAX_Part1_Overview_and_Performance.pdf

[9] H. Yaghoobi, "Scalable OFDMA physical layer in IEEE 802.16 WirelessMAN," *Intel Technology Journal*, vol. 8, pp. 201–212, Aug 2004.

[10] K.-C. Hung and D. W. Lin, "Wireless MAN physical layer specifications: signal processing perspective," Book Chapter, Dec. 2006.

[11] Sundance home page: http://www.sundance.com

[12] Texas Instruments, *TMS320C6000 CPU and Instruction Set Reference Guide.* Literature no. SPRU189F, Oct. 2000.

[13] Texas Instruments, *TMS320C6414T, TMS320C6415T, TMS320C6416T Fixed-Point Digital Signal Processors.* Literature no. SPRS226A, Mar. 2004.

[14] Texas Instruments, *Code Composer Studio User's Guide.* Literature no. SPRU328B, Feb. 2000.

[15] Texas Instruments, *TMS320C6000 Code Composer Studio Tutorial.* Literature no. SPRU301CI, Feb. 2000.

[16] Texas Instruments, *TMS320C6000 Programmer's Guide.* Literature no. SPRU198I, Mar. 2006.

[17] Texas Instrument, *TMS320C6000 Optimizing Compiler User Guide.* Literature no. SPRU187K, Oct. 2002.

[18] P. Dent, G. E. Bottomley, and T. Croft, "Jakes' fading model revisited," *Electron. Lett.*, vol. 29, no. 13, pp. 1162–1163, June 1993.

[19] V. Erceg et al., "Channel models for fixed wireless applications," IEEE 802.16.3c-01/29r4, July 2001.

[20] ETSI TR 101 112, "Selection procedures for the choice of radio transmission technologies of the UMTS," *ETSI Technical Report*, V3.0.2, pp. 38–43, Apr. 1994.

[21] 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.

[22] J. J. van de Beek, P. O. Borjesson, M. L. Boucheret, D. Landstromo J. M. Arenas, P. Odling, and S. K. Wilson, "Three non-pilot-based time and frequency estimators for OFDM," Research Report 1998:08, Division of Signal Processing, Lulea University of Technology, Sep. 1998.

[23] T. M. Schmidl and D. C. Cox, "Robust frequency and timing synchronization for OFDM," *IEEE Trans. Commun.,* vol. 45, no. 12, pp. 1613–1621, Dec. 1997.

[24] M. Morelli and U. Mengali, "An improved frequency offset estimator for OFDM applications," *IEEE Commun. Lett.,* vol. 3, pp. 75–77, Mar. 1999.

[25] U. Tureli, H. Liu., and M. D. Zoltowski, "OFDM blind carrier offset estimation: ESPRIT," *IEEE Trans. Commun.,* vol. 48, pp. 1459–1461, Sep. 2000.

[26] X. Ma, C. Tepedelenlioglu, G. B. Giannakis, and S. Barbarossa, "Non-data-aided carrier offset estimator for OFDM with nullsubcarriers: Identifiabilit, algorithms, and performance," *IEEE J. Select. Areas Commun.,* vol. 19, pp. 2504–2515, Dec. 2001.

[27] T. Bhatt, V. Sundaramurthy, J. Zhang, and D. McCain, "Initial synchronization for 802.16e downlink," *Proc. Asilomar Conf. Signals Systems Computers.,* Nov. 2006, pp. 701-706 .

[28] 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.

[29] 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.

[30] G.-W. Ji, “Research in synchronization techniques and DSP implementation for IEEE 802.16e OFDM uplink and OFDMA downlink,“ M.S. thesis, Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C., June 2006.

[31] H. Lim and D. S. Kwon, “Initial synchronization for WiBro,” in *Proc. Asia-Pacific Conf. Commun.,* Oct. 2005, pp. 284–288.

[32] K.-C. Hung and D. W. Lin, “Joint detection of integral carrier frequency offset and preamble index in OFDMA WiMAX downlink synchronization,” in *Proc. IEEE Wireless Commun. Networking Conf.,* Mar. 2007, pp. 1959-1964.

作者簡歷

學生劉耀鈞，民國七十一年十一月出生於台北市。民國九十四年六月畢業於國立交通大學電信工程系，並於同年九月進入國立交通大學電子研究所就讀，從事 OFDMA 通訊系統方面相關研究。民國九十六年六月取得碩士學位，碩士論文題目為『IEEE 802.16e OFDMA 同步技術之研究與數位信號處理器實現』。