國 立 交 通 大 學

電子工程學系 電子研究所碩士班

碩 士 論 文

IEEE 802.16e OFDM 上行及 OFDMA 下行通道估測技術與數位訊號處理器實現之探討

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

研 究 生：王治傑

指導教授：林大衛 博士

中 華 民 國 九 十 五 年 六 月

IEEE 802.16e OFDM 上行及 OFDMA 下行通道估測技術與數位信號處理器實現之探討

Research in Channel Estimation and DSP Implementation for IEEE 802.16e OFDM Uplink and OFDMA Downlink

研究生: 王治傑　　Student: Chih-Chieh Wang

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
in
Electronics Engineering
June 2006
Hsinchu, Taiwan, Republic of China

中華民國九十五年六月

IEEE 802.16e OFDM 上行及 OFDMA 下行通道估測技術與數位訊號處理器實現之探討

研究生：王治傑　　　　　　　　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

在過去數年間，多載波調變 (multicarrier modulation)，尤其是正交分頻多工 (OFDM) 技術已經應用在許多數位通訊應用中。採用 OFDM 一個最主要的原因是增加抗頻率選擇性衰變及窄頻干擾的能力。在單一載波的系統中，衰變或干擾可能會導致整個連結失效；但在一個多載波的系統，僅僅只有一小部分的載波會受到影響。我們聚焦在 IEEE 802.16e OFDM 上行及 OFDMA 下行的部分，並用數位信號處理器去實現通道估測的機制。此數位訊號處理器的環境是 Innovative Integration 公司的 Quixote 個人電腦插板，其上裝置為德州儀器公司的 TMS320C6416，是個擁有強大數學運算功能的處理器。

通道估測大致可以分成兩個階段。首先我們使用最小平方差的估測器來估計在導訊上的通道頻率響應，這是為了硬體的計算方便。而內插的方法我們則研究了多項式內插法及 Cubic Spline 內插法；而在用時域的資料改善的方法有下列三種：二維內插法、最小平方法(Least Squares)以及最小平均平方差適應(Normalized LMS adaptation)；最後我們將通道估測與信號偵測何在一起作已達到更好的效果(Joint Channel Estimation and Symbol Detection)。我們先在 AWGN 通道上驗證我們的模擬模型，然後在分別在靜態及時變的 Rayleigh 通道上模擬。

至於數位信號處理器實現的部份，考量其運算複雜度，只用簡單的內插法綜合二維內插。為了增進程式的執行效率，我們先將原始的浮點運算 C 程式版本修改為實數運算的程式版本，接著再考慮數位訊號處理器的特性來修改之前的程式

在本篇論文中，我們首先簡介 IEEE 802.16e OFD 上行及 OFDMA 下行的標準及機制。接著，我們介紹所用通道估測的技巧和 DSP 的實現環境。最後，我們描述各種通道估測技術的效果及數位訊號處理器實現方面的實驗結果。

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

Student：Chih-Chieh Wang　　　　Advisor：Dr. David W. Lin

Department of Electronics Engineering

Institute of Electronics

National Chiao Tung University

Abstract

Multicarrier modulation, in particular orthogonal frequency division multiplexing (OFDM), has been successfully applied to a wide variety of digital communications applications over the past several years. One of the main reason to use OFDM is to increase the robustness against frequency selective fading and narrowband interference. In a single carrier system, a single fade or interference can cause the entire link to fail, but in a multicarrier system, only a small percentage of subcarriers will affected. We focus on the OFDM uplink and OFDMA downlink channel estimation based on IEEE 802.16e. Also, we use digital signal processor to implement OFDM uplink channel estimation schemes. The digital signal processing environment is Innovative Integration's Quixote personal computer card, which hosts Texas Instruments' TMS320C6416 which is a powerful signal processor with strong arithmetic operation capability.

The channel estimation scheme can be separated into two stages. In the first stage, we use LS estimator for estimations of pilot subcarriers because of its low computational complexity. We study polynomial interpolations and cubic spline interpolations in frequency domain, NLMS adaptation algorithm, least squares in time domain, two-dimensional interpolations in time domain and maximum likelihood estimation. Finally, we did joint channel estimation and symbol detection to get better performance. Also we verify our simulation model on AWGN channel and then did the simulation on both static and time-variant fading channels.

As for the DSP implementation, combination of linear interpolation and 2D interpolation are chosen due to computational complexity. In order to increase the efficiency, we also rewrite the original floating-point C program to fixed-point version and further refine our codes by taking into account the features of the DSP chip.

In this thesis, we first introduce to standard of the IEEE 802.16e OFDM uplink and OFDMA downlink. Second, we describe the channel estimation techniques. Then the DSP implementation environment will be introduced. Finally, we discuss the channel estimation performance and the DSP implementation results.

誌謝

首先這篇論文能夠完成，要先感謝我的指導教授林大衛老師，在我遭遇困難時他總是能給予適當的方向去解決問題，當然最重要的是我也從他那裡學到不少獨立研究的精神及方法。

回首過去兩年我在新竹交大的日子，因為有實驗室同學及學長之間的切磋與指導，在在都讓我感到過去兩年也許是我求學生涯中最充實的那一段，所以在此要特別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。尤其要感謝吳俊榮學長、洪崑健學長的指導與建議，他們幫我解決不少學業上的疑惑，還有勇竹、國偉、育彰、冠彰、閔傑等同學，謝謝他們在這兩年來對我的幫助及帶給我歡樂。

當然家人對我的支持、鼓勵是我求學路上精神的慰藉，對他們的感謝，也是筆墨難以形容的。

最後由衷感謝所有幫助關懷過我的人。

王治傑

民國九十五年六月 於新竹

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Oncoming technologies such as WiFi and WiMAX are changing wireless communication. The demand for high data rate transmission and multimedia type traffic grows rapidly. Due to its inherent advantages in high-speed communication, orthogonal frequency division multiplexing (OFDM) has become the choice for a number of high wireless systems (e.g., DVB-T, WiFi, WiMAX).

OFDM system transmits data using a set of parallel low bandwidth subcarriers. The subcarriers are independent from each other even though their spectra overlap, which results in its bandwidth efficiency and resistance to the ICI (inter-carrier interference) effect. And due to the low bandwidth of subcarriers, each subcarrier can avoid worse subchannels. Thus ISI (inter-symbol-interference) is also reduced. High data rate systems are also achieved by using a large number of carriers. OFDM can be easily generated using an inverse fast Fourier transform (IFFT) and received using a fast Fourier transform (FFT).

The IEEE 802.16 standard committee has developed a group of standards for wireless metropolitan area networks (MANs). The original 802.16 standard specifies the air interface for fixed broadband wireless access systems supporting multimedia services. The medium access control layer (MAC) supports a primarily point-to-multipoint architecture, with an optional mesh topology. The MAC is structured to support multiple physical layer (PHY) specifications, each suited to a particular operational environment. For operational frequencies from 10–66 GHz, the WirelessMAN-SC PHY, based on single-carrier modulation, is specified. For frequencies below 11 GHz, where propagation without a direct line of sight must be accommodated, three alternatives are provided in the later 802.16a: WirelessMAN-OFDM (using orthogonal frequency-division multiplexing), WirelessMAN-OFDMA (using orthogonal frequency-division multiple access), and WirelessMAN-SCa (using single-carrier modulation).

IEEE 802.16 has developed the IEEE Standard 802.16-2004 for broadband wireless access systems, which provides a variety of services to fixed outdoor as well as nomadic indoor users. Work is underway on a mobile extension, referred to as Project 802.16e, supporting new capabilities needed for the mobile environment. The object of this thesis focus on the channel estimation part for WirelessMAN-OFDM and WirelessMAN-OFDMA.

This thesis is organized as follows. First, in chapter 2, we introduce some OFDM basics together with the IEEE 802.16e OFDM uplink and OFDMA downlink standard. In chapter 3, the various channel estimation techniques are introduced. In chapter 4, we describe the implementation platform, which consists of Texas Instrument's TMS320C6416 digital signal processor (DSP) on a cPCI board named Quixote made by Innovative Integration. Then in chapter 5, we discuss the performance of channel estimation methods of OFDM uplink and some DSP implementation issues. The simulation results of OFDMA downlink will be left to chapter 6. At last, we give the conclusion and discuss potential future work in chapter 7.

Chapter 2

Introduction to IEEE802.16e OFDM and OFDMA

This chapter presents a brief overview of the OFDM and the OFDMA techniques for multicarrier modulation. The OFDM uplink and OFDMA downlink specifications of IEEE 802.16e are also introduced.

2.1 Basics of OFDM and OFDMA

The basic idea of OFDM-like or OFDMA-like system is to split the data stream into several parallel streams, each transmitted on a separate subcarrier. Moreover, these subcarriers are made orthogonal is to allow spectral overlapping and better spectral efficiency can be achieved. Material in this section is taken from [1], [2] and [3].

Figure 2.1 shows the block diagram of a simplex transmission system using OFDM. The three main principles incorporated are as follows:

1. The IDFT and the DFT are used for modulating and demodulating the data constellations on the orthogonal subcarriers. These signal processing algorithms replace the banks of I/Q-modulators and demodulators that would otherwise be required. Usually,

N is taken as an integer power of two, enabling the application of the efficient FFT algorithms for modulation and demodulation.

2. The second key principle is the introduction of a cyclic prefix, whose length should exceed the maximum excess delay of the multipath propagation channel. Due to the cyclic prefix, the transmitted signal becomes periodic, and the effect of the time-dispersive multipath channel is equivalent to a cyclic convolution, after discarding the cyclic prefix at the receiver. Owing to the properties of the cyclic convolution, the effect of the multipath channel is limited to a pointwise multiplication of the data constellations by the channel transfer function. The only drawback of this principle is a slight loss of effective transmit power. The equalization (symbol demapping) required for detecting the data constellations (when there is no error control coding) is an elementwise multiplication of the DFT output by the inverse of the estimated channel transfer function.

3. FEC (forward error control) coding and interleaving are usually also applied. The frequency-selective radio channel may severely attenuate the data symbols transmitted on one or several subcarriers. Spreading the coded bits over the bandwidth, an efficient coding scheme can correct the erroneous bits and hence exploit the frequency diversity. Synchronization is a key issue in the design of a robust OFDM receiver. Time and frequency synchronization are paramount to identify the start of the OFDM symbol and to align local oscillator frequencies.

2.1.1 The OFDM Principle

For multicarrier modulation, the available bandwidth W is divided into a number N_c of subbands, commonly called subcarriers, each of width $\Delta f = W/N_c$. The subdivision of the

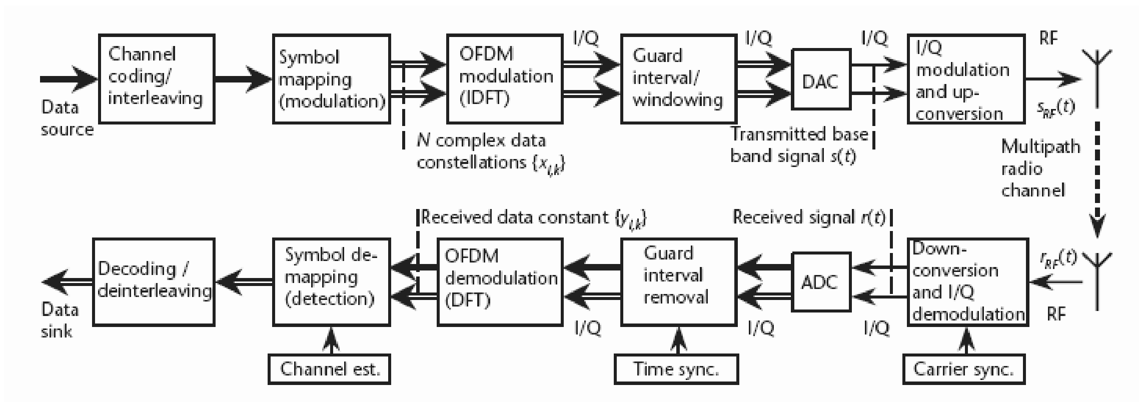

Data source
Channel coding/ interleaving
Symbol mapping (modulation)
N complex data constellations $\{x_{i,k}\}$
OFDM modulation (IDFT)
I/Q
Guard interval/ windowing
I/Q
DAC
Transmitted base band signal $s(t)$
I/Q
I/Q modulation and up-conversion
RF
$s_{RF}(t)$
Multipath radio channel
$r_{RF}(t)$
RF
Down-conversion and I/Q demodulation
Carrier sync.
I/Q
Received signal $r(t)$
ADC
I/Q
Guard interval removal
Time sync.
I/Q
OFDM demodulation (DFT)
Received data constant $\{y_{i,k}\}$
Symbol de-mapping (detection)
Channel est.
Decoding / deinterleaving
Data sink

Figure 2.1: OFDM system block diagram (from [2]).

bandwidth is illustrated in Fig. 2.2, where arrows represent the different subcarriers. Instead of transmitting the data symbols in a serial way, at a baud rate R, a multicarrier transmitter partitions the data stream into blocks of N_c data symbols that are transmitted in parallel by modulating the N_c carriers. The symbol duration for a multicarrier scheme is $T_s = N_c/R$.

As shown in Fig. 2.3, the general form of the multicarrier signal can be written as a set of modulated carriers:

$$s(t) = \sum_{m=-\infty}^{\infty} \Big(\sum_{k=0}^{N_c-1} x_{k,m}\psi_k(t - mT_s) \Big) \tag{2.1}$$

where $x_{k,m}$ is the data symbol modulating the kth subcarrier in the mth signalling interval and ψ_k is the waveform for the kth subcarrier.

The symbol duration can be made long compared to the maximum excess delay of the channel, $T_s \gg \tau_{max}$, or by choosing N_c sufficiently high. At the same time, the bandwidth of the subbands can be made small to reduce the coherence bandwidth of the channel ($B_{coh} \gg W/N_c$). The subbands then experience flat fading, which reduces equalization to a signal complex multiplication per carrier. Hence, increasing N_c reduces the ISI (intersymbol interference) effects.

However, the performance in time-variant channels is degraded by long symbols. If the

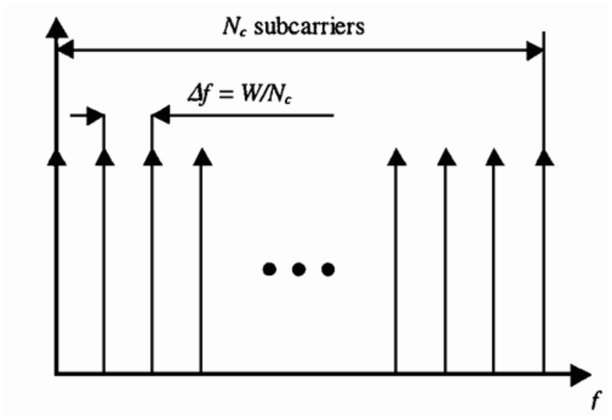


Figure 2.2: Subdivision of the bandwidth into N_c subbands (from [3]).

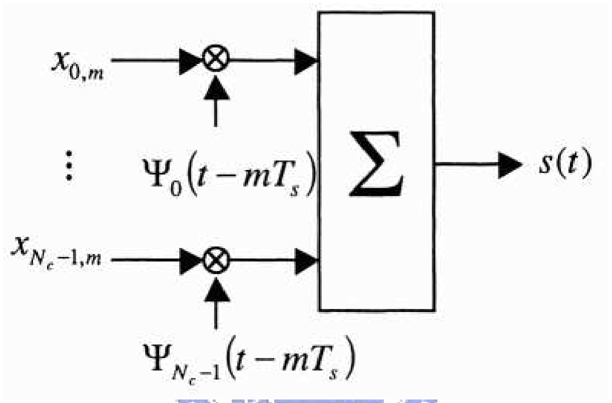


Figure 2.3: Multicarrier modulation (from [3]).

coherence time T_{coh} of the channel is small compared to T_s, the channel frequency response changes significantly during the transmission of one symbol. As a result, the coherence time of the channel defines an upper bound for the number of subcarriers. Together with the condition for flat fading within the subbands, a reasonable range for N_c is given by

$$\frac{W}{B_{coh}} \ll N_c \ll RT_{coh}. \tag{2.2}$$

To assure a high spectral efficiency, the subchannel waveforms have overlapping transmit

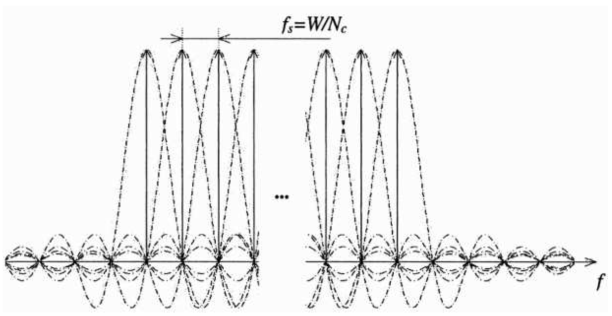


Figure 2.4: Spectrum of an OFDM signal (from [3]).

spectra. A general set of orthogonal waveforms is given by:

$$\psi_k(t) = \begin{cases} \frac{1}{\sqrt{T_s}} e^{j\omega_k t}, & t \in [0, T_s], \\ 0, & \text{otherwise}, \end{cases} \tag{2.3}$$

where $\omega_k = \omega_0 + k\omega_s$ and $k = 0, 1, \ldots, N_c - 1$. Here $f_k = \omega_k/2\pi$ is the subcarrier frequency and $f_0 = w_0/2\pi$ is the lowest frequency used ($k = 0$). The spacing between the adjacent subcarriers equals $\Delta f = \omega_s/2\pi = W/N_c$. Since the waveform $\psi_k(t)$ is restricted to the time window $[0, T_s]$, the intercarrier spacing must also satisfy $\triangle f = 1/T_s = R/N_c$. The windowing results in a convolution with $T_s \cdot exp(-j\pi f T_s) \cdot \frac{\sin(\pi f T_s)}{\pi f T_s}$ in the frequency domain. How the subbands overlap is shown in Fig. 2.4.

2.1.2 Cyclic Prefix

As mentioned, to overcome the ISI and ICI (interchannel interference) problem, the cyclic prefix (CP) is introduced. A cyclic prefix is a copy of the last part of the OFDM symbol (see Fig. 2.5). The cyclic prefix should be at least as long as the significant part of the channel impulse response. The benefit of the CP is twofold. First, it avoids ISI because it acts as a guard space between successive symbols. Second, it also converts the linear convolution with channel impulse response into a cyclic convolution. However, the transmitted energy

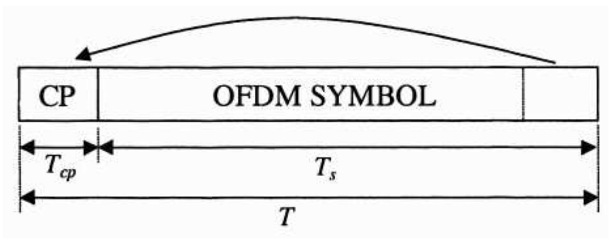


Figure 2.5: Cyclic prefix (form [3]).

increases with the length of the CP. The SNR loss is given by

$$SNR_{loss} = -10\log_{10}(1-\frac{T_g}{T_s}). \tag{2.4}$$

2.1.3 Discrete-Time Equivalent System Model

The discete-time baseband equivalent model is shown in Fig. 2.6. In the transmitter, the data stream is grouped in blocks of N_c data symbols. These groups are called OFDM symbols and can be represented by a vector $x_m = [x_{0,m}\ x_{1,m}\ \cdots\ x_{N_c-1,m}]^T$. Then an IDFT is performed on each data symbol block, and a cyclic prefix of length N_{cp} is added. The resulting complex baseband discrete time signal of mth OFDM-symbol can be written as

$$s_m(n) = \begin{cases} \frac{1}{N_c}\sum\limits_{k=0}^{N_c-1} x_{k,m}e^{j2\pi k(n-N_{cp})/N_c}, & \text{if } n \in [0, N_c+N_{cp}-1], \\ 0, & \text{otherwise.} \end{cases} \tag{2.5}$$

The complete time signal $s(n)$ is given by the concatenation of all OFDM symbols that are transmitted:

$$s(n) = \sum_{m=0}^{\infty} s_m(n-m(N_c+N_{cp})). \tag{2.6}$$

Here we assume that

- The channel fading is slow enough to consider it constant during one OFDM symbol.
- The transmitter and receiver are perfectly synchronized.

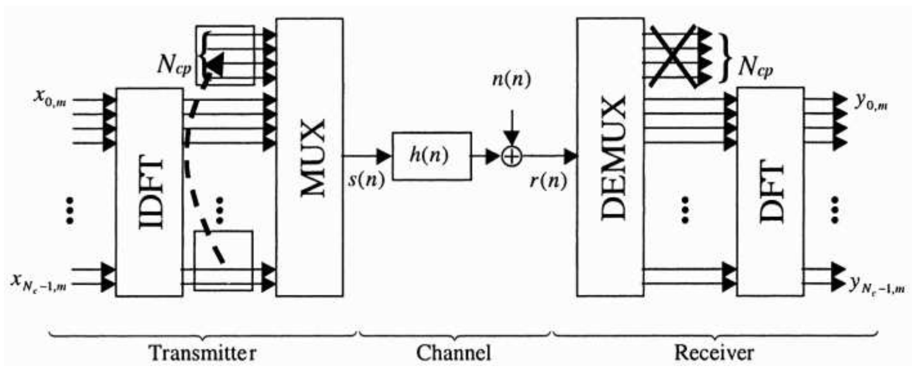


Figure 2.6: Discrete-time baseband equivalent model (from [3]).

- The CP is sufficiently long to accommodate the channel impulse response.

We can then write

$$r(n) = \sum_{\eta=0}^{N_{cp}-1} h(\eta)s(n-\eta) + n(n). \tag{2.7}$$

In the receiver, the incoming sequence is split into blocks and the cyclic prefix associated with each clock is removed. This results in a vector $r_m = [r(z_m)\ r(z_m+1)\ \cdots\ r(z_m+N_c-1)]^T$, with $z_m = m(N_c+N_{cp})+N_{cp}$. Performing DFT on r_m yields

$$y_{k,m} = \sum_{n=0}^{N_c-1} r(z_m+n)e^{-j2\pi kn/N_c}. \tag{2.8}$$

Substituting $r(n)$ in (2.7) into (2.8) gives

$$y_{k,m} = \sum_{n=0}^{N_c-1}\left[\sum_{\eta=0}^{N_{cp}-1} h(\eta)s_m(N_cp+n-\eta)\right]e^{-j2\pi kn/N_c} + \sum_{n=0}^{N_c-1} n(z_m+n)e^{-j2\pi kn/N_c}. \tag{2.9}$$

Then substituting $s_m(n)$ in (2.5) into it yields

$$y_{k,m} = \sum_{n=0}^{N_c-1}\left[\sum_{\eta=0}^{N_{cp}-1} h(\eta)\frac{1}{N_c}\sum_{k=0}^{N_c-1} x_{k,m}e^{j2\pi k(n-\eta)/N_c}\right]e^{-j2\pi kn/N_c} + n_{k,m} \tag{2.10}$$

where $n_{k,m} = \sum_{n=0}^{N_c-1} n(z_m+n)e^{-j2\pi kn/N_c}$ is the kth sample of the N_c-point DFT of $n(z_m+n)$. Since $n(n)$ is white Gaussian, $n_{k,m}$ is also white Gaussian.

Note that $h(\eta) = 0$ for all $\eta > N_{cp} - 1$. Additional swapping the two inner sums and reordering yields

$$y_{k,m} = \underbrace{\sum_{n=0}^{N_c-1} \Big[\underbrace{\frac{1}{N_c} \sum_{k=0}^{N_c-1} \Big(\sum_{\eta=0}^{N_c-1} h(\eta) e^{-j2\pi k\eta/N_c} \Big) x_{k,m} e^{j2\pi kn/N_c}}_{IDFT} \Big] e^{-j2\pi kn/N_c}}_{DFT} + n_{k,m}. \tag{2.11}$$

The first part of this expression consists of an IDFT operation nested in a DFT operation. The inner sum is the kth sample of the N_c-point DFT of $h(n)$, or h_k. The equation hence translates into

$$y_{k,m} = h_k x_{k,m} + n_{k,m}. \tag{2.12}$$

This equation demonstrates that the received data symbol $y_{k,m}$ on each subcarrier k equals the data symbol $x_{k,m}$, multiplied by the corresponding frequency-domain channel coefficient h_k in addition to the transformed noise $n_{k,m}$.

For a more compact notation, a matrix equivalent is often used. For a single OFDM symbol, it equals

$$\mathbf{y_m} = \mathbf{H} \circ \mathbf{x_m} + \mathbf{n_m} = \text{diag}(\mathbf{H}) \cdot \mathbf{x_m} + \mathbf{n_m} \tag{2.13}$$

where $\circ$ denotes the element-wise product, diag($\mathbf{H}$) is the diagonal matrix of the elements of $\mathbf{H}$, $\mathbf{y_m} = [y_{0,m}\ y_{1,m}\ \cdots\ y_{N_c-1,m}]^T$, $\mathbf{n_m} = [n_{0,m}\ n_{1,m}\ \cdots\ n_{N_c-1,m}]^T$ and $\mathbf{H} = [h_0\ h_1\ \cdots\ h_{N_c-1}]^T$.

2.2 Generic OFDM and OFDMA Symbol Description for IEEE 802.16e

2.2.1 Time Domain Description

IDFT creates the OFDM symbol waveforms. The time duration of each OFDM symbol is referred to as the useful symbol time T_s. A copy of the last T_{cp} of the useful symbol period is made the CP (see Fig. 2.5). The CP overhead and resultant SNR loss can be reduced

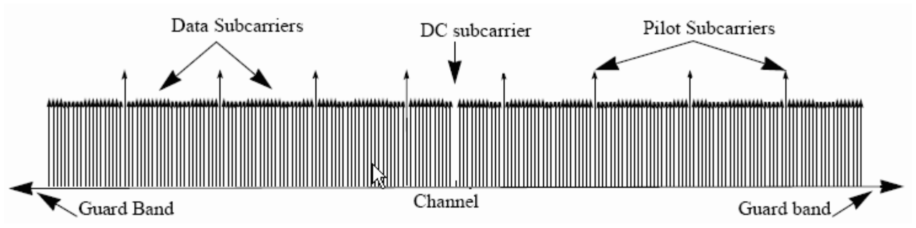


Figure 2.7: OFDM frequency description (from [4]).

by increasing the FFT size, which would however, among other things, adversely affect the sensitivity of the system to phase noise of the oscillators. Using a cyclic extension, the samples required for performing the FFT at the receiver can be taken anywhere over the length of the extended symbol. This provides multipath immunity as well as a tolerance for symbol time synchronization errors.

2.2.2 Frequency Domain Description

An OFDM (see Fig. 2.7) or OFDMA (see Fig. 2.8) symbol is made up from several carrier types:

- Data carriers: For data transmission.
- Pilot carriers: For various estimation purposes.
- Null carriers: No transmission ar all, for guard bands, non-active subcarriers and the DC subcarrier.

2.2.3 Primitive Parameters

Four primitive parameters characterize the OFDM and the OFDMA symbols:

- BW: The nominal channel bandwidth.

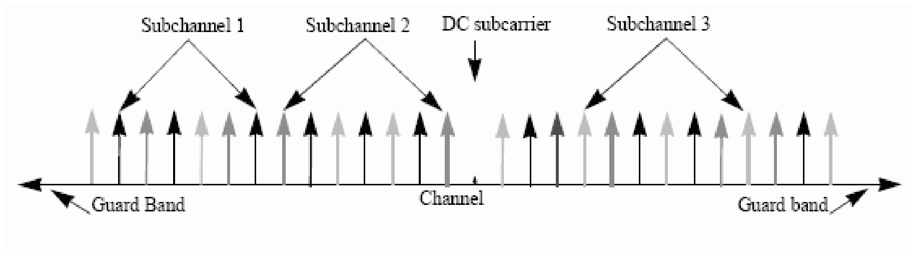


Figure 2.8: OFDMA frequency description (from [4]).

- N_{used}: Number of used subcarriers.
- n: Sampling factor. This parameter, in conjunction with BW and N_{used}, determines the subcarrier spacing and the useful symbol time.
- G: This is the ratio of CP time to "useful" time, i.e., T_{cp}/T_s.

2.2.4 Derived Parameters

The following parameters are defined in terms of the primitive parameters.

- N_{FFT}: Smallest power of two greater than N_{used}.
- Sampling frequency: $F_s = floor(\text{n}\cdot\text{BW}/8000) \times 8000$.
- Subcarrier spacing: $\triangle f = F_s/N_{FFT}$.
- Useful symbol time: $T_s = 1/\triangle f$.
- CP time: $T_{cp} = G \times T_s$.
- OFDM or OFDMA symbol time: $T = T_s + T_{cp}$.
- Sampling time: T_s/N_{FFT}.

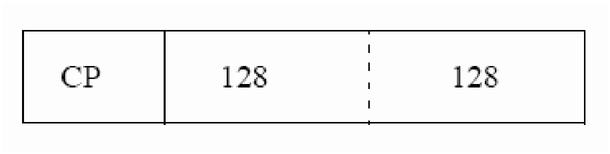


Figure 2.9: P_{EVEN} time domain structure (from [4]).

2.3 OFDM Uplink Specifications in IEEE 802.16e

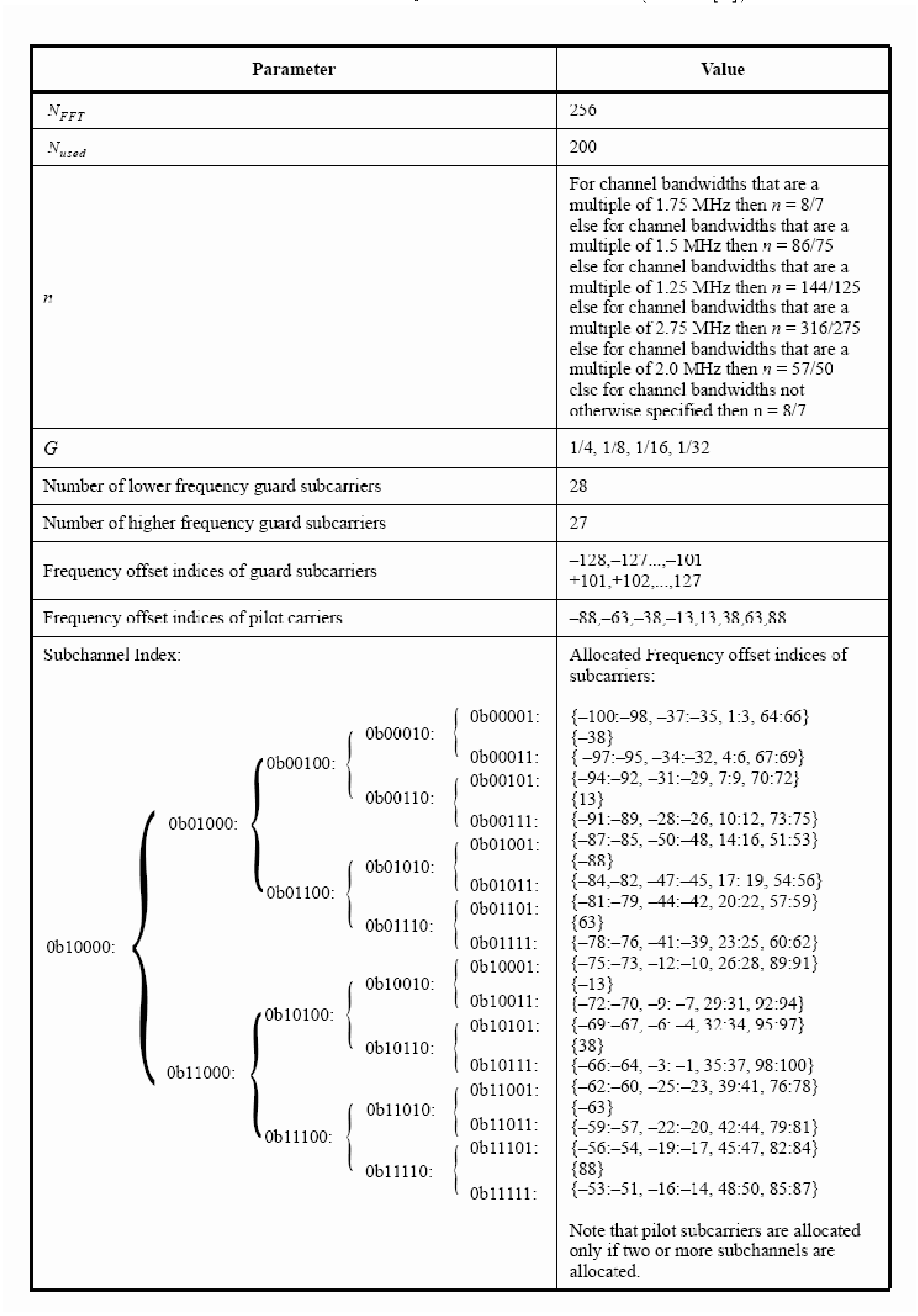
This section introduces the IEEE 802.16e OFDM uplink (UL) standard. The material is mainly taken from [4] and [5]. The parameters are given in Table 2.1. Note that Table 2.1 also indicates the pilot locations and the subchannelization method which will not be detailed later.

2.3.1 OFDM UL Preamble Structure and Modulation

In the uplink, the data preamble is as shown in Fig. 2.9. It consists of one OFDM symbol utilizing only even subcarriers. The time domain waveform consists of 2 times 128 samples preceded by a CP. The subcarrier values shall be set as

$$P_{EVEN}(k) = \begin{cases} \sqrt{2} \cdot P_{ALL}(k), & k \bmod 2 = 0, \\ 0, & k \bmod 2 \neq 0, \end{cases} \tag{2.14}$$

Table 2.1: OFDM Symbol Parameters (from [4])



Parameter	Value
N_{FFT}	256
N_{used}	200
n	For channel bandwidths that are a multiple of 1.75 MHz then n = 8/7 else for channel bandwidths that are a multiple of 1.5 MHz then n = 86/75 else for channel bandwidths that are a multiple of 1.25 MHz then n = 144/125 else for channel bandwidths that are a multiple of 2.75 MHz then n = 316/275 else for channel bandwidths that are a multiple of 2.0 MHz then n = 57/50 else for channel bandwidths not otherwise specified then n = 8/7
G	1/4, 1/8, 1/16, 1/32
Number of lower frequency guard subcarriers	28
Number of higher frequency guard subcarriers	27
Frequency offset indices of guard subcarriers	–128,–127...,–101 +101,+102,...,127
Frequency offset indices of pilot carriers	–88,–63,–38,–13,13,38,63,88
Subchannel Index:	Allocated Frequency offset indices of subcarriers:
0b10000:	
0b01000:	
0b00100:	
0b00001:	{–100:–98, –37:–35, 1:3, 64:66}
0b00010:	{–38}
0b00011:	{ –97:–95, –34:–32, 4:6, 67:69}
0b00101:	{–94:–92, –31:–29, 7:9, 70:72}
0b00110:	{13}
0b00111:	{–91:–89, –28:–26, 10:12, 73:75}
0b01100:	
0b01001:	{–87:–85, –50:–48, 14:16, 51:53}
0b01010:	{–88}
0b01011:	{–84,–82, –47:–45, 17: 19, 54:56}
0b01101:	{–81:–79, –44:–42, 20:22, 57:59}
0b01110:	{63}
0b01111:	{–78:–76, –41:–39, 23:25, 60:62}
0b11000:	
0b10100:	
0b10001:	{–75:–73, –12:–10, 26:28, 89:91}
0b10010:	{–13}
0b10011:	{–72:–70, –9: –7, 29:31, 92:94}
0b10101:	{–69:–67, –6: –4, 32:34, 95:97}
0b10110:	{38}
0b10111:	{–66:–64, –3: –1, 35:37, 98:100}
0b11100:	
0b11001:	{–62:–60, –25:–23, 39:41, 76:78}
0b11010:	{–63}
0b11011:	{–59:–57, –22:–20, 42:44, 79:81}
0b11101:	{–56:–54, –19:–17, 45:47, 82:84}
0b11110:	{88}
0b11111:	{–53:–51, –16:–14, 48:50, 85:87}
	Note that pilot subcarriers are allocated only if two or more subchannels are allocated.

where

$$\begin{aligned}P_{ALL}(-100:100) = \{&1-j, 1-j, -1-j, 1+j, 1-j, 1-j, -1+j, 1-j, 1-j,\\
&1-j, 1+j, -1-j, 1+j, 1+j, -1-j, 1+j, -1-j, -1-j, 1-j, -1+j, 1-j,\\
&1-j, -1-j, 1+j, 1-j, 1-j, -1+j, 1-j, 1-j, 1-j, 1+j, -1-j, 1+j,\\
&1+j, -1-j, 1+j, -1-j, -1-j, 1-j, -1+j, 1-j, 1-j, -1-j, 1+j, 1-j,\\
&1-j, -1+j, 1-j, 1-j, 1-j, 1+j, -1-j, 1+j, 1+j, -1-j, 1+j, -1-j,\\
&-1-j, 1-j, -1+j, 1+j, 1+j, 1-j, -1+j, 1+j, 1+j, -1-j, 1+j, 1+j,\\
&1+j, -1+j, 1-j, -1+j, -1+j, 1-j, -1+j, 1-j, 1-j, 1+j, -1-j, -1-j,\\
&-1-j, -1+j, 1-j, -1-j, -1-j, 1+j, -1-j, -1-j, -1-j, 1-j, -1+j,\\
&1-j, 1-j, -1+j, 1-j, -1+j, -1+j, -1-j, 1+j, 0, -1-j, 1+j, -1+j,\\
&-1+j, -1-j, 1+j, 1+j, 1+j, -1-j, 1+j, 1-j, 1-j, 1-j, -1+j, -1+j,\\
&-1+j, -1+j, 1-j, -1-j, -1-j, -1+j, 1-j, 1+j, 1+j, -1+j, 1-j, 1-j,\\
&1-j, -1+j, 1-j, -1-j, -1-j, -1-j, 1+j, 1+j, 1+j, 1+j, -1-j, -1+j,\\
&-1+j, 1+j, -1-j, 1-j, 1-j, 1+j, -1-j, -1-j, -1-j, 1+j, -1-j, -1+j,\\
&-1+j, -1+j, 1-j, 1-j, 1-j, 1-j, -1+j, 1+j, 1+j, -1-j, 1+j, -1+j,\\
&-1+j, -1-j, 1+j, 1+j, 1+j, -1-j, 1+j, 1-j, 1-j, 1-j, -1+j, -1+j,\\
&-1+j, -1+j, 1-j, -1-j, -1-j, 1-j, -1+j, -1-j, -1-j, 1-j, -1+j,\\
&-1+j, -1+j, 1-j, -1+j, 1+j, 1+j, 1+j, -1-j, -1-j, -1-j, -1-j, 1+j,\\
&1-j, 1-j\}.\end{aligned}$$

2.3.2 OFDM UL Pilot Modulation

Pilot subcarriers shall be inserted into each data burst in order to constitute the symbol. The PRBS (pseudo-random binary sequence) generator depicted in Fig. 2.10 shall be used

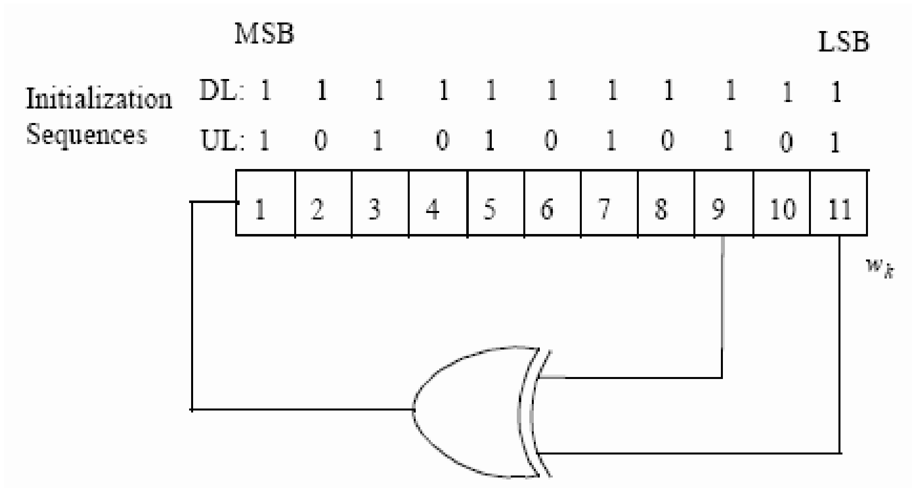


Figure 2.10: PRBS for pilot modulation (from [4]).

to produce a sequence, w_k. The polynomial for the PRBS generator is $X^{11} + X^9 + 1$.

The initialization sequence shown in Fig. 2.10 shall be used on the downlink (DL) and uplink. The value of the pilot modulation for OFDM symbol k is derived from w_k. On both uplink and downlink, the first symbol of the preamble is denoted by $k = 0$. For each pilot, the BPSK modulation shall be as follows:

$$\text{DL}: c_{-88} = c_{-38} = c_{63} = c_{88} = 1 - 2w_k \text{ and } c_{-63} = c_{-13} = c_{13} = c_{38} = 1 - 2\overline{w}_k, \tag{2.15}$$

$$\text{UL}: c_{-88} = c_{-38} = c_{13} = c_{38} = c_{63} = c_{88} = 1 - 2w_k \text{ and } c_{-63} = c_{-13} = 1 - 2\overline{w}_k. \tag{2.16}$$

2.3.3 OFDM UL Data Modulation

The OFDM data modulation schemes are shown in Fig. 2.11. The data bits are entered serially to the constellation mapper. BPSK, Gray-mapped QPSK, Gray-mapped 16QAM and Gray-mapped 64QAM must be supported. The indicated factor c is used to achieve equal average power.

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

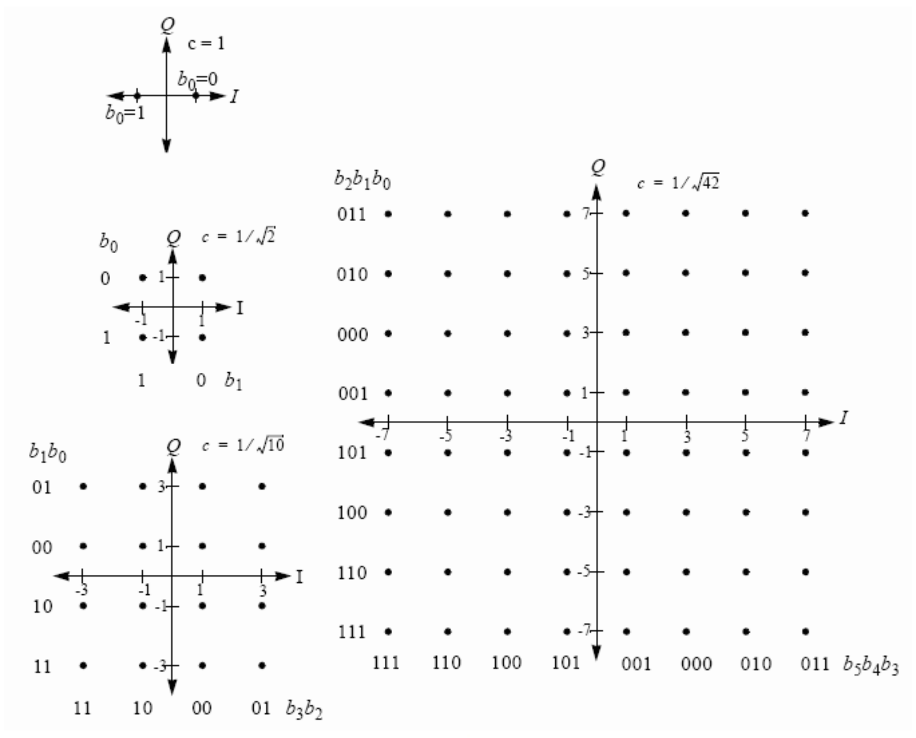


Figure 2.11: BPSK, QPSK, 16QAM and 64QAM constellations (from [4]).

constellation mapping shall be modulated onto the allocated subcarrier with the lowest frequency offset index.

2.4 OFDMA Downlink Specifications in IEEE 802.16e

This section briefs the IEEE 802.16e OFDMA downlink standard. Both the PUSC (Partial Usage of SubChannels) and FUSC (Full Usage of SubChannels) are introduced. The material is mainly taken from [4] and [5].

2.4.1 Definition of OFDMA Basic Terms

In OFDMA, a slot in the OFDMA PHY is a two-dimensional entity spanning both a time and a subchannel dimension. It is the minimum possible data allocation unit.

- For downlink FUSC, one slot is one subchannel by one OFDMA symbol.
- For downlink PUSC, one slot is one subchannel by two OFDMA symbols.

A Data Region is a two-dimensional allocation of a group of contiguous subchannels, in a group of contiguous OFDMA symbols; a segment is a subdivision of the set of available OFDMA subchannels.

The downlink data mapping rules are as follows:

1. Segment the data after the modulation block into blocks sized to fit into one OFDMA slot.
2. Each slot shall span one subchannel in the subchannel axis and one or more OFDMA symbols in the time axis, as per the slot definition mentioned before. Map the slots such that the lowest numbered slot occupies the lowest numbered subchannel in the lowest numbered OFDMA symbol.
3. Continue the mapping such that the OFDMA subchannel index is increased. When the edge of the Data Region is reached, continue the mapping from the lowest numbered OFDMA subchannel in the next available symbol.

Figure 2.12 illustrates the order in which OFDMA slots are mapped to subchannels and OFDMA symbols.

2.4.2 OFDMA DL Preamble Structure and Modulation

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 PN

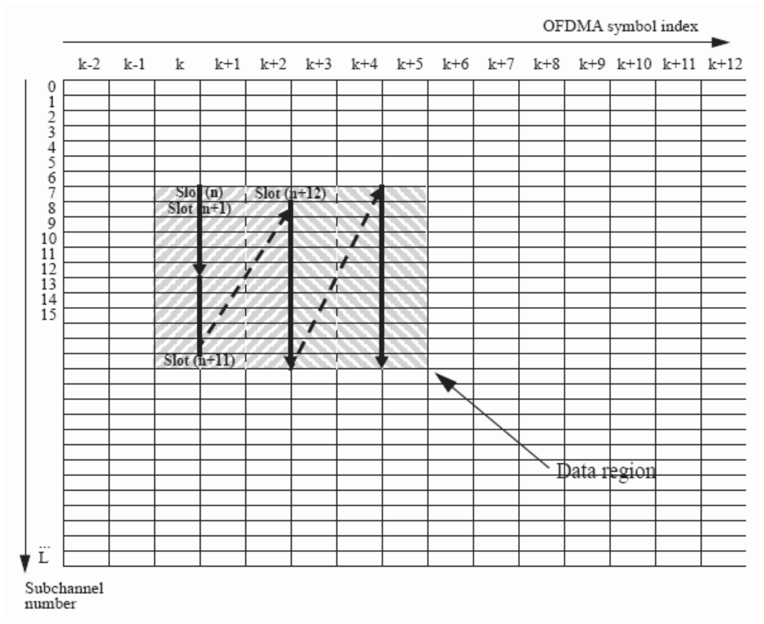


Figure 2.12: Example of mapping OFDMA slots to subchannels and symbols in the downlink in PUSC mode (from [5]).

(pseudo-noise) code. The PN series modulating the preamble carrier-sets can be found in [4, pp. 553–562]. The preamble carrier-sets are defined as

$$PreambleCarrierSet_n = n + 3 \cdot k, \tag{2.17}$$

where:

- $PreambleCarrierSet_n$ specifies all subcarriers allocated to the specific preamble,
- n is the number of the preamble carrier-set indexed 0, 1, 2,
- k is a running index 0,. . . ,567.

Each segment uses one type of preamble out of the three sets in the following manner: The DC carrier will not be modulated at all and the appropriate PN will be discarded. Therefore, DC carrier shall always be zeroed. For the preamble symbol, there will be 172 guard band

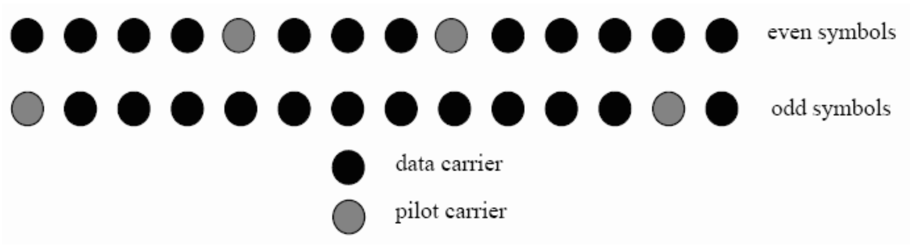


Figure 2.13: Cluster structure (from [5]).

subcarriers on both the left side and on the right side of the spectrum. Segment i uses preamble carrier-set i, where $i = 0, 1, 2$.

The pilot in downlink preamble shall be modulated as

$$\begin{aligned} &\Re\{PreamblePilotsModulated\} = 4 \cdot \sqrt{2} \cdot (\frac{1}{2} - w_k), \\ &\Im\{PreamblePilotsModulated\} = 0. \end{aligned} \tag{2.18}$$

2.4.3 OFDMA DL Carrier Allocation

For both uplink and downlink in OFDMA, N_{used} subcarriers are allocated to pilot subcarriers and data subcarriers. 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. Table 2.2 summarizes the parameters of the OFDMA PUSC symbol structure and Table 2.3 the corresponding parameters of OFDMA FUSC.

2.4.3.1 PUSC DL

The OFDMA symbol structure is constructed using pilots, data and zero subcarriers. The symbol is first divided into basic clusters and zero carriers are allocated. Pilots and data carriers are allocated within each cluster. Figure 2.13 shows the cluster structure with subcarriers from left to right in order of increasing subcarrier index.

The allocation of subcarriers to subchannels is performed using the following procedure:

Table 2.2: OFDMA DL Subcarrier Allocation under PUSC [4], [5]

Parameter	**Value**	**Comments**
Number of DC subcarriers	1	Index 1024 (counting from 0)
Number of guard subcarriers, left	184	
Number of guard subcarriers, right	183	
Number of used subcarriers (N_{used})	1681	Number of all subcarriers used within a symbol, including all possible allocated pilots and the DC carrier
Number of subcarriers per cluster	14	
Number of clusters	120	
Renumbering sequence	1	Used to renumber clusters before allocation to subchannels: 6,108,37,81,31,100,42,116,32,107,30,93,54,78, 10,75,50,111,58,106,23,105,16,117,39,95,7, 115,25,119,53,71,22,98,28,79,17,63,27,72,29, 86,5,101,49,104,9,68,1,73,36,74,43,62,20,84, 52,64,34,60,66,48,97,21,91,40,102,56,92,47, 90,33,114,18,70,15,110,51,118,46,83,45,76,57, 99,35,67,55,85,59,113,11,82,38,88,19,77,3,87, 12,89,26,65,41,109,44,69,8,61,13,96,14,103,2, 80,24,112,4,94,0
Number of data subcarriers in each symbol per subchannel	24	
Number of subchannels	60	
Basic permutation sequence 12 (for 12 subchannels)	12	6,9,4,8,10,11,5,2,7,3,1,0
Basic permutation sequence 8 (for 8 subchannels)	8	7,4,0,2,1,5,3,6

1) Dividing the subcarriers into the number ($N_{clusters}$) of physical clusters containing 14 adjacent subcarriers each (starting from carrier 0).

2) Renumbering the physical clusters into logical clusters using the following formula:

$$LogicalCluster = \begin{cases} RenumberingSequence(PhysicalCluster), & \text{first DL zone,} \\ RenumberingSequence\big((PhysicalCluster + \\ \qquad 13 \cdot DL\_PermBase) \text{mod } N_{clusters}\big), & \text{otherwise.} \end{cases}$$

3) Dividing the clusters into six major groups. Group 0 includes clusters 0–23, group 1 clusters 24–39, group 2 clusters 40–63, group 3 clusters 64–79, group 4 clusters 80–103 and group 5 clusters 104–119. These groups may be allocated to segments. If a segment is being used, then at least one group shall be allocated to it. (By default group 0 is allocated to sector 0, group 2 to sector 1, and group 4 to sector 2).

4) Allocating subcarriers to subchannels in each major group as

$$subcarrier(k,s) = N_{subchannels} \cdot n_k + \{p_s[n_k \; mod \; N_{subchannels}] + DL\_PermBase\} \text{mod } N_{subchannels}.$$

where:

- $subcarrier(k,s)$ is the subcarrier index of subcarrier k in subchannel s;
- $n_k = (k + 13 \cdot s) \text{mod } N_{subcarriers}$;
- $N_{subchannels}$ is the number of subchannels (for PUSC use number of subchannels in the currently partitioned group);
- $p_s[j]$ is the series obtained by rotating basic permutation sequence cyclically to the left s times;
- $N_{subcarriers}$ is the number of data subcarriers allocated to a subchannel in each OFDMA symbol;
- $DL\_PermBase$ is an integer from 0 to 31.

2.4.3.2 FUSC DL

In comparison with PUSC, the subcarrier allocation and subchannelization methods for FUSC are much simpler. For data subcarrier allocation and subchannelization, use the parameters listed in Table 2.3 and apply the same carrier allocation procedures mentioned in PUSC part. As for pilot allocation, use

$$\begin{aligned} PilotsLocation = \big[&VariableSet\#x + 6 \cdot (FUSC\_SymbolNumber\ mod\ 2)\big] \\ &\cup ConstantSet\#0 \cup ConstantSet\#1. \end{aligned}$$

2.4.4 OFDMA DL Pilot Modulation

In OFDMA, the polynomial for PRBS generator is $X^{11} + X^9 + 1$, which is the same as that in OFDM system. However, in this case, the initialization vector used to generate w_k shall refer to the MAC layer and is not introduced here. The interested reader can get detailed information from [5, pp. 631–632].

In both FUSC and PUSC modes, 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 as

$$\begin{aligned} \Re\{c_k\} &= \frac{8}{3}\Big(\frac{1}{2} - w_k\Big), \\ \Im\{c_k\} &= 0. \end{aligned} \tag{2.19}$$

2.4.5 OFDMA DL Data Modulation

As shown in Fig. 2.11, in the OFDMA system, the data bits are entered serially to the constellation mapper. Gray-mapped QPSK and Gray-mapped 16QAM shall be supported, whereas the support of 64QAM (also Gray-mapped) is optional and BPSK is not supported.

Table 2.3: OFDMA DL Subcarrier Allocation under FUSC [4], [5]

Parameter	**Value**	**Comments**
Number if DC subcarriers	1	Index 1024 (counting from 0)
Number of Guard subcarriers, Left	173	
Number of Guard subcarriers, Right	172	
Number of used subcarriers (N_{used})	1703	Number of all subcarriers used within a symbol, including all possible allocated pilots and the DC carrier
Pilots		
VariableSet #0	71	0,72,144,216,288,360,432,504,576,648,720,792,864, 936,1008,1080,1152,1224,1296,1368,1440,1512,1584, 1656,48,120,192,264,336,408,480,552,624,696,768, 840,912,984,1056,1128,1200,1272,1344,1416,1488, 1560,1632,24,96,168,240,312,384,456,528,600,672, 744,816,888,960,1032,1104,1176,1248,1320,1392, 1464,1536,1608,1680
ConstantSet #0	12	9,153,297,441,585,729,873,1017,1161,1305,1449, 1593
VariableSet #1	71	36,108,180,252,324,396,468,540,612,684,756,828, 900,972,1044,1116,1188,1260,1332,1404,1476,1548, 1620,1692,12,84,156,228,300,372,444,516,588,660, 732,804,876,948,1020,1092,1164,1236,1308,1380, 1452,1524,1596,1668,60,132,204,276,348,420,492, 564,636,708,780,852,924,996,1068,1140,1212,1284, 1356,1428,1500,1572,1644
ConstantSet #1	12	81,225,369,513,657,801,945,1089,1233,1377,1521, 1665
Number of data subcarriers	1536	
Number of data subcarriers per subchannel	48	
Number of sunchannels	32	
Basic permutation sequence	32	3,18,2,8,16,10,11,15,26,22,6,9,27,20,25,1,29, 7,21,5,28,31,23,17,4,24,0,13,12,19,14,30

It is worth noting that before mapping the data to the physical subcarriers, each subcarrier shall be multiplied by the factor $2 \times (1/2 - w_k)$ according to the subcarrier physical index k. This extra multiplicative factor is not used in the OFDM system.

Chapter 3

Channel Estimation Techniques

3.1 Pilot-Symbol-Aided Channel Estimation

Channel estimators usually need some kind of pilot information as a point of reference. A fading channel requires constant tracking, so pilot information has to be transmitted more or less continuously. Decision-directed channel estimation can also be used. But even in these types of schemes, pilot information has to be transmitted regularly to mitigate error propagation [6].

3.1.1 The Least-Squares (LS) Estimator

The simplest channel estimator one can imagine consists simply in dividing the received signal by the symbols that have been actually sent (and that are supposed to be known). Based on a priori known data, we can estimate the channel information on pilot carriers roughly by the least-squares (LS) estimator. An LS estimator minimizes the following squared error [7]:

$$||\mathbf{Y} - \hat{\mathbf{H}}_{LS}\mathbf{X}||^2 \tag{3.1}$$

where $\mathbf{Y}$ is the received signal and $\mathbf{X}$ is a priori known pilots, both in the frequency domain and both being $N \times 1$ vectors where N is the FFT size. $\hat{\mathbf{H}}_{LS}$ is an $N \times N$ matrix whose values are 0 except at pilot locations m_i where $i = 0, \cdots, N_p - 1$:

$$\hat{\mathbf{H}}_{LS} = \begin{bmatrix} H_{m_0,m_0} & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \cdots & H_{m_1,m_1} & \cdots & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \cdots & H_{m_2,m_2} & \cdots & 0 \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & H_{m_{N_p-1},H_{m_{N_p-1}}} \end{bmatrix}. \tag{3.2}$$

Therefore, (3.1) can be rewritten as

$$[Y(m) - \hat{H}_{LS}(m)X(m)]^2, \text{ for all } m = m_i. \tag{3.3}$$

Then the estimate of pilot signals, based on only one observed OFDM symbol, is given by

$$\hat{H}_{LS}(m) = \frac{Y(m)}{X(m)} = \frac{X(m)H(m) + N(m)}{X(m)} = H(m) + \frac{N(m)}{X(m)} \tag{3.4}$$

where $N(m)$ is the complex white Gaussian noise on subcarrier m. We collect $H_{LS}(m)$ into $\hat{\mathbf{H}}_{\mathbf{p},\mathbf{LS}}$, an $N_p \times 1$ vector where N_p is the total number of pilots, as

$$\begin{aligned} \hat{\mathbf{H}}_{p,LS} = & \ [H_{p,LS}(0)\ H_{p,LS}(1)\ \cdots H_{p,LS}(N_p-1)]^T \\ = & \ [\tfrac{Y_p(0)}{X_p(0)}, \tfrac{Y_p(1)}{X_p(1)}, \cdots, \tfrac{Y_p(N_p-1)}{X_p(N_p-1)}]^T. \end{aligned} \tag{3.5}$$

The LS estimate of $\mathbf{H}_p$ based on one OFDM symbol is susceptible to noise effects, and thus an estimator better than the LS estimator is preferable.

3.1.2 The LMMSE Estimator

The minimum mean-square error (MMSE) estimate has been shown to be better than the LS estimate for channel estimation in OFDM systems, but the major drawback of the MMSE estimate is its high complexity. A low-rank approximation results in a linear minimum mean squared error (LMMSE) estimator that uses the frequency-domain correlation of the channel [8]. The linear minimum mean-square error channel estimator tries to minimize the mean squared error between the actual and estimated channels, obtained by a linear transformation applied to $\hat{\mathbf{H}}_{\mathbf{p,LS}}$. The mathematical representation of the LMMSE estimator on pilot signals is

$$\begin{aligned}\hat{\mathbf{H}}_{p,lmmse} &= \mathbf{R}_{H_pH_{p,LS}}\mathbf{R}^{-1}_{H_{p,LS}H_{p,LS}}\hat{\mathbf{H}}_{p,LS} \\ &= \mathbf{R}_{H_pH_p}(\mathbf{R}_{H_pH_p} + \sigma_n^2(\mathbf{X}_p\mathbf{X}_p^H)^{-1})^{-1}\hat{\mathbf{H}}_{p,LS} \qquad (3.6)\end{aligned}$$

where $\hat{\mathbf{H}}_{p,LS}$ is the least-square estimate of $\mathbf{H}_p$ in (3.5), σ_n^2 is the variance of the Gaussian white noise, $\mathbf{X_p}$ is the vector of transmitted signal on pilot subcarriers, and the covariance matrices are defined by

$$\mathbf{R}_{H_pH_{p,LS}} = E\{\mathbf{H}_p\mathbf{H}^H_{p,LS}\}, \qquad (3.7)$$

$$\mathbf{R}_{H_{p,LS}H_{p,LS}} = E\{\mathbf{H}_{p,LS}\mathbf{H}^H_{p,LS}\}, \qquad (3.8)$$

$$\mathbf{R}_{H_pH_p} = E\{\mathbf{H}_p\mathbf{H}^H_p\}. \qquad (3.9)$$

Note that there is a matrix inverse involved in the MMSE estimator, which must be calculated every time, and the computation of matrix inversion requires $O(N_p^3)$ arithmetic operations [9]. We also need the statistical properties of the unknown channel. Therefore, we use the LS estimator which requires only $O(N_p)$ operations instead of the LMMSE due to the concerns of complexity and unknown information.

3.2 One-Dimensional Channel Estimators

By one-dimensional channel estimation, we mean that we only use channel information along the frequency domain. In other words, we use the channel information at pilot subcarriers obtained by the LS estimator to estimate the channel information at data subcarriers via interpolation or extrapolation. The material in this section is largely taken from [10].

3.2.1 Polynomial Interpolation and Extrapolation

The interpolation polynomial of degree $N-1$ through the N points $y_1 = f(x_1)$, $y_2 = f(x_2)$, $\ldots$, $y_N = f(x_N)$ is given by Lagrange's classical formula as

$$\begin{aligned} Px =& \frac{(x-x_2)(x-x_3)\ldots(x-x_N)}{(x_1-x_2)(x_1-x_3)\ldots(x_1-x_N)}y_1 + \frac{(x-x_1)(x-x_3)(x-x_N)}{(x_2-x_1)(x_2-x_3)(x_2-x_N)}y_2 \\ &+ \cdots + \frac{(x-x_1)(x-x_2)(x-x_{N-1})}{(x_N-x_1)(x_N-x_2)(x_N-x_{N-1})}y_N. \end{aligned} \tag{3.10}$$

There are N terms, each a polynomial of degree $N-1$ and each constructed to be zero at all of the x_i except one, at which it is constructed to be y_i.

A better method for constructing the interpolating polynomial is *Neville's algorithm* as follows: Let P_1 be the value at x of the unique polynomial of degree zero passing through the point (x_1, y_1); so $P_1 = y_1$. Likewise define $P_2, P_3, \ldots, P_N$. Now let P_{12} be the value at x of the unique polynomial of degree one passing through both (x_1, y_1) and (x_2, y_2). Likewise $P_{23}, P_{34}, \ldots, P_{(N-1)N}$. Similarly, for higher-order polynomials, compute up to $P_{123\ldots N}$, which is the value of the unique interpolating polynomial through all N points, i.e., the desired answer. The various Ps form a "tableau" on the left lading to a single "descendent" at the extreme right. For example, with $N = 4$,

$$\begin{array}{lllll} x_1: & y_1 = P_1 & & & \\ & & P_{12} & & \\ x_2: & y_2 = P_2 & & P_{123} & \\ & & P_{23} & & P_{1234} \\ x_3: & y_3 = P_3 & & P_{234} & \\ & & P_{34} & & \\ x_4: & y_4 = P_4 & & & \end{array} \tag{3.11}$$

Neville's algorithm is a recursive way of filling in the numbers in the tableau a column at a time, from left to right. It is based on the relationship between a "daughter" P and its two "parents" as

$$P_{i(i+1)\ldots(i+m)} = \frac{(x - x_{i+m})P_{i(i+1)\ldots(i+m-1)} + (x_i - x)P_{(i+1)(i+2)\ldots(i+m)}}{x_i - x_{i+m}}. \tag{3.12}$$

This recurrence works because the two parents already agree at points $x_{i+1}, \ldots, x_{i+m-1}$.

An improvement on the recurrence (3.12) is to keep track of the small differences between parents and daughters, namely, to define (for $m = 1, 2, \ldots, N-1$),

$$\begin{aligned} C_{m,i} &\equiv P_{i\ldots(i+m)} - P_{i\ldots(i+m-1)}, \\ D_{m,i} &\equiv P_{i\ldots(i+m)} - P_{(i+1)\ldots(i+m)}. \end{aligned} \tag{3.13}$$

Then one can easily derive from (3.12) the relations

$$\begin{aligned} D_{m+1,i} &= \frac{(x_{i+m+1} - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}}, \\ C_{m+1,i} &= \frac{(x_i - x)(C_{m,i+1} - D_{m,i})}{x_i - x_{i+m+1}}. \end{aligned} \tag{3.14}$$

At each level m, the Cs and Ds are the corrections that make the interpolation one order higher. The final answer $P_{1\ldots N}$ is equal to the sum of any y_i plus a set of Cs and/or Ds that form a path through the family tree to the rightmost daughter.

Usually, linear and second-order interpolations are employed due to the consideration of complexity, as discussed in [11], [12] and [13]. The mathematical expression of linear and second order interpolations are given below.

3.2.1.1 Linear interpolation

The linear interpolation is given by

$$H_e(k) = H_e(m + l) = (H_p(m+1) - H_p(m))\frac{l}{L} + H_p(m) \tag{3.15}$$

where $H_p(k), k = 0, 1, \cdots, N_p$, are the channel frequency responses at pilot subcarriers, L is the distance between the two given data, that is, the pilot subcarriers spacing, and $0 \le l < L$.

3.2.1.2 Second order interpolation

The second-order interpolation is given by

$$\begin{aligned} H_e(k) &= H_e(m+l) \\ &= c_1 H_p(m-1) + c_0 H_p(m) + c_{-1} H_p(m+1) \end{aligned} \tag{3.16}$$

where

$$\begin{aligned} c_1 &= \frac{\alpha(\alpha-1)}{2}, \\ c_0 &= -(\alpha-1)(\alpha+1), \\ c_{-1} &= \frac{\alpha(\alpha+1)}{2}, \\ \alpha &= \frac{l}{L}. \end{aligned}$$

Other notations are the same as in linear interpolation.

3.2.2 Rational Function Interpolation and Extrapolation

Some functions are not well approximated by polynomials, but are well approximated by rational functions. We denote by $R_{i(i+1)\ldots(i+m)}$ a rational function passing through the $m+1$ points $(x_i, y_i), \ldots, (x_{i+m}, y_{i+m})$. Suppose

$$R_{i(i+1)\ldots(i+m)} = \frac{P_\mu(x)}{Q_\nu(x)} = \frac{p_0 + p_1 x + \cdots + p_\mu x^\mu}{q_0 + q_1 x + \cdots + q_\nu x^\nu}. \tag{3.17}$$

Since there are $\mu + \nu + 1$ unknown μs and νs (q_0 being arbitrary), we must have $m + 1 = \mu + \nu + 1$.

Rational functions are sometimes superior to polynomials because of their ability to model functions with poles, that is, zeros of the denominator of (3.17). These poles might occur for real values of x, if the function to be interpolated itself has poles. More often, the function $f(x)$ is finite for all finite real x, but has an analytic continuation with poles in the complex x-plane. Such poles can ruin a polynomial approximation, especially those at real values of x.

Bulirsch and Stoer found an algorithm of the Neville type which performs rational function extrapolation on tabulated data. The algorithm is summarized by a recurrence relation:

$$R_{i(i+1)\dots(i+m)} = R_{(i+1)\dots(i+m)} + \frac{R_{(i+1)\dots(i+m)} - R_{i\dots(i+m-1)}}{\left(\frac{x-x_i}{x-x_{i+m}}\right)\left(1 - \frac{R_{(i+1)\dots(i+m)} - R_{i\dots(i+m-1)}}{R_{(i+1)\dots(i+m)} - R_{(i+1)\dots(i+m-1)}}\right) - 1}. \tag{3.18}$$

This recurrence generates the rational functions through $m+1$ points from the ones through m and (the term $R_{(i+1)\dots(i+m-1)}$) $m-1$ points. It is started with

$$R_i = y_i \qquad \text{and} \qquad R \equiv [R_{i(i+1)\dots(i+m)} \quad \text{with} \quad m = -1] = 0.$$

Now, we can convert the recurrence (3.18) to one involving only the small differences

$$\begin{aligned} C_{m,i} &\equiv R_{i\dots(i+m)} - R_{i\dots(i+m-1)}, \\ D_{m,i} &\equiv R_{i\dots(i+m)} - R_{(i+1)\dots(i+m)}. \end{aligned} \tag{3.19}$$

Note that these satisfy the relation

$$C_{m+1,i} - D_{m+1,i} = C_{m,i+1} - D_{m,i} \tag{3.20}$$

which is useful in proving the recurrences

$$\begin{aligned} D_{m+1,i} &= \frac{C_{m,i+1}(C_{m,i+1} - D_{m,i})}{\left(\frac{x-x_i}{x-x_{i+m+1}}\right)D_{m,i} - C_{m,i+1}}, \\ C_{m+1,i} &= \frac{\left(\frac{x-x_i}{x-x_{i+m+1}}\right)D_{m,i}(C_{m,i+1} - D_{m,i})}{\left(\frac{x-x_i}{x-x_{i+m+1}}\right)D_{m,i} - C_{m,i+1}}. \end{aligned} \tag{3.21}$$

3.2.3 Cubic Spline Interpolation [10], [14], [15]

Cubic spline is one very effective, well-behaved, computationally efficient interpolation. The approach is to fit cubic polynomials to adjacent pairs of points and choose the values of the two remaining parameters associated with each polynomial such that the polynomials covering adjacent intervals agree with one another in both slope and curvature at their common endpoint. The cubic spline interpolation is developed in the following.

Given a tabulated function $y_i = y(x_i)$ and its second order derivative y'', $i = 1, \ldots, N$, let us focus our attention on one particular interval, say between x_j and x_{j+1}. The goal of cubic spline interpolation is to get an interpolation formula that is smooth in the first derivative and continuous in the second derivative, both within the interval and at its boundaries. A little calculation shows that there is only one way to arrange this construction, that is,

$$y = Ay_i + By_{i+1} + cy''_j + Dy''_{j+1} \tag{3.22}$$

where A, B, C and D are given by

$$\begin{aligned} A &= \frac{x_{j+1} - x}{x_{j+1} - x_j}, & B &= 1 - A = \frac{x - x_j}{x_{j+1} - x_j}, \\ C &= \frac{1}{6}(A^3 - A)(x_{j+1} - x_j)^2, & D &= \frac{1}{6}(B^3 - B)(x_{j+1} - x_j)^2. \end{aligned} \tag{3.23}$$

Combined with (3.23), we take the derivatives of (3.22) with respect to x, yielding

$$\frac{dy}{dx} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{3A^2 - 1}{6}(x_{j+1} - x_j)y''_j + \frac{3B^2 - 1}{6}(x_{j+1} - x_j)y''_{j+1} \tag{3.24}$$

for the first derivative and

$$\frac{d^2y}{dx^2} = Ay''_j + By''_{j+1} \tag{3.25}$$

for the second derivative. Since $A = 1$ at x_j, $A = 0$ at x_{j+1}, while B is just the other way around, (3.25) shows that y'' is just the tabulated second derivative, and also that the second derivative will be continuous across the boundary.

The only problem now is that we supposed the y''_i's to be known, when actually they are not. The key idea of a cubic spline is to require the continuity of the first derivative and to use it to get equations for the second derivatives y''_i.

We set (3.24) evaluated for $x = x_j$ in the interval (x_{j-1}, x_j) equal to the same equation evaluated for $x = x_j$ but in the interval (x_j, x_{j+1}). With some arrangement, this gives, for $j = 2, \ldots, N - 1$,

$$\frac{x_j - x_{j-1}}{6}y''_{j-1} + \frac{x_{j+1} - x_{j-1}}{3}y''_j + \frac{x_{j+1} - x_j}{6}y''_{j+1} = \frac{y_{j+1} - y_j}{x_{j+1} - x_j} - \frac{y_j - y_{j-1}}{x_j - x_{j-1}}. \tag{3.26}$$

These are $N-2$ linear equations in the N unknowns y_i'', $i=1,\ldots,N$. Therefore there is a two-parameter family of possible solutions.

For a unique solution, we need to specify two further conditions, typically taken as boundary conditions at x_1 and x_N. The most common ways of doing this are either

- set one or both y_1'' and y_N'' equal to zero, giving the so-called natural cubic spline, or
- set either of y_1'' and y_N'' to some values so as to make the first derivative of the interpolating function have a specified value on either or both boundaries.

3.2.4 The Maximum Likelihood Channel Estimator

As mentioned before, the LMMSE estimator exploits channel correlations in time and frequency domains. It needs knowledge of the channel statistics and the operating SNR. As indicated in [8], although it can work with mismatched conditions on parameter values, its performance degrades if the assumed Doppler frequencies and the delay spread are smaller than the true ones.

The LMMSE estimator regards the channel impulse response as a random vector whose particular realization is to be estimated. On the contrary, in maximum likelihood estimation (MLE), the channel impulse response is viewed as a deterministic but unknown vector and no information on the channel statistics or the operating SNR is required in this scheme.

The MLE of $\mathbf{h}$ is give by [16]

$$\widehat{\mathbf{h}}_{\mathbf{MLE}} = \mathbf{D}^{-1}\mathbf{B}^{\mathbf{H}}\widehat{\mathbf{H}}_{\mathbf{p,LS}} \tag{3.27}$$

where $\mathbf{D}$ is a square matrix $\mathbf{D} = \mathbf{B^H B}$ whose entries are given by

$$\left[\mathbf{B}\right]_{n,k} = e^{-j2\pi k i_n/N}, \quad 0 \le n \le N_p - 1,\ 0 \le k \le L - 1, \tag{3.28}$$

$$\left[\mathbf{D}\right]_{n,k} = \sum_{m=0}^{N_p-1} e^{j2\pi(n-k)i_m/N}, \quad 0 \le n, k \le L - 1, \tag{3.29}$$

where $\widehat{\mathbf{H}}_{\mathbf{p,LS}}$ is given by (3.5), i_n are pilot locations, N_p is the number of pilots and L is channel length.

Equation (3.27) indicates that MLE requires the invertibility of $\mathbf{D}$. Such a condition is met if and only if $\mathbf{B}$ is full rank and $N_p \ge L$. this means that the number of pilots must be not smaller than the number of channel taps.

3.3 Two-Dimensional Channel Estimators

By two-dimensional channel estimation, we mean that in addition to using channel information along the frequency domain, we also use channel information along the time domain to get better performance.

3.3.1 Interpolation in Time Domain

Figure 3.1 shows a typical pilot pattern of OFDM symbols. Channel response is interpolated along the time axis based on that derived along the frequency axis. The interpolation methods are the same as that used in frequency-domain interpolation and are not detailed here.

3.3.2 Linear Prediction [10]

We consider the problem of fitting a set of N data points (x_i, y_i) to a straight-line model:

$$y(x) = a + bx. \tag{3.30}$$

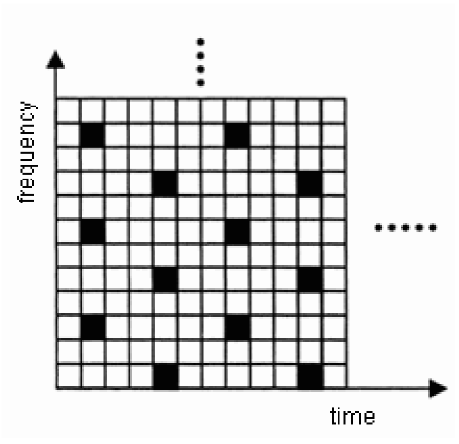


Figure 3.1: A typical pilot pattern (from [3]).

We assume that the uncertainty σ_i associated with each measurement y_i is known, and that the x_i's are known exactly.

3.3.2.1 Minimizing the chi-square merit function

To measure how well the model agrees with the data, we use the chi-square merit function, which in this case is

$$\chi^2(a,b) = \sum_{i=1}^{N} \left(\frac{y_i - a - bx_i}{\sigma_i} \right). \tag{3.31}$$

Equation (3.31) is minimized to determine a and b. At the minimum, derivatives of $\chi^2(a,b)$ with respect to a and b vanish as

$$\begin{aligned} 0 &= \frac{\partial \chi^2}{\partial a} = -2 \sum_{i=1}^{N} \frac{y_i - a - bx_i}{\sigma_i^2}, \\ 0 &= \frac{\partial \chi^2}{\partial b} = -2 \sum_{i=1}^{N} \frac{x_i(y_i - a - bx_i)}{\sigma_i^2}. \end{aligned} \tag{3.32}$$

Define the following sums:

$$\begin{aligned} S \equiv \sum_{i=1}^{N} \frac{1}{\sigma_i^2}, \quad S_x \equiv \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2}, \quad S_y \equiv \sum_{i=1}^{N} \frac{y_i}{\sigma_i^2}, \\ S_{xx} \equiv \sum_{i=1}^{N} \frac{x_i^2}{\sigma_i^2}, \quad S_{xy} \equiv \sum_{i=1}^{N} \frac{x_i y_i}{\sigma_i^2}. \end{aligned} \tag{3.33}$$

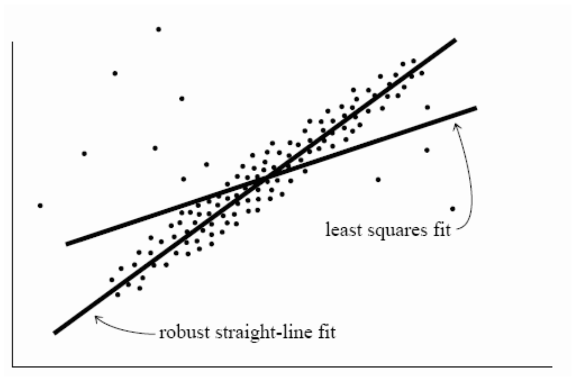


Figure 3.2: An example where robust statistical methods are desirable (from [10]).

With these definitions (3.32) becomes

$$
\begin{aligned}
aS + bS_x &= S_y, \\
aS_x + bS_{xx} &= S_{xy}.
\end{aligned}
\tag{3.34}
$$

The solution of these two equations in two unknowns is given by

$$
\begin{aligned}
\Delta &\equiv SS_{xx} - (S_x)^2, \\
a &= \frac{S_{xx}S_y - S_xS_{xy}}{\Delta}, \\
b &= \frac{SS_{xy} - S_xS_y}{\Delta}.
\end{aligned}
\tag{3.35}
$$

which gives the solution for the best-fit model parameters a and b.

3.3.2.2 Minimizing absolute deviation

Instead of using chi-square as the merit function, consider minimizing

$$
\sum_{i=1}^{N} |y_i - a - bx_i|. \tag{3.36}
$$

This merit function is more robust, i.e., insensitive to small departures from the idealized assumptions from which the estimator is optimized. An example where robust statistical methods are desirable are shown in Fig. 3.2.

The key simplification is based on the following fact: The median c_M of a set of number c_i is also that value which minimizes the sum of the absolute deviations

$$\sum_i |c_i - c_M|. \tag{3.37}$$

It follows that, for fixed b, the value of a that minimizes (3.36) is

$$a = \text{median}\{y_i - bx_i\}. \tag{3.38}$$

Employing (3.38) for the parameter b gives

$$0 = \sum_{i=1}^{N} x_i \text{sgn}(y_i - a - bx_i). \tag{3.39}$$

We use these two methods to predict the channel response along the time axis (see Fig. 3.1). Suppose this results in a predicted channel response $H_{predict}(k)$ together with an interpolated channel response (along the frequency axis) $H_{interp}(k)$. We combine the two results by averaging as $[H_{predict}(k) + H_{interp}(k)]/2$.

3.3.3 Time Averaging

Because we assume the noise is white Gaussian, averaging several channel responses over a period of time can mitigate the influence of noise. Now the problem is how long the period of time should we choose. Coherence time is a statistical measure of the time duration over which the channel impulse response is essentially invariant, and it quantifies the similarity of the channel response at different times.

Take OFDMA system for example. Assume the SS has a velocity of 60 km/h. The maximum Dopper shift with a center frequency 3.5 GHz can be calculated as

$$f_m = \frac{v}{\lambda} = 194.44 \text{ Hz}. \tag{3.40}$$

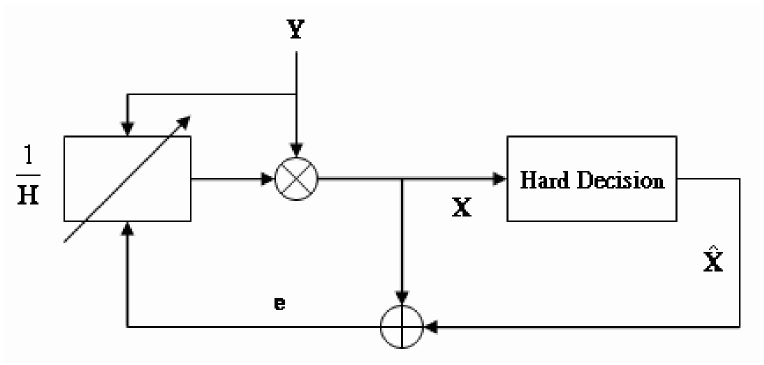


Figure 3.3: NLMS equalizer adaptation.

The corresponding coherence time can be approximately obtained as [17]

$$T_c \approx \frac{9}{16\pi f_m} = 920.83\,\mu\text{s}. \tag{3.41}$$

As the considered OFDMA system operates with bandwidth 10 MHz, the symbol period is then $(2048+64)/\left(\lfloor \frac{28}{25}\cdot 10M/8000\rfloor \times 8000\right) = 188.57\,\mu$s. Hence, the channel response over $\lfloor \frac{920.83}{188.57} \rfloor = 4$ symbols can be regarded static. Thus we use an averaging over 4 symbols to reduce noise effect as

$$H_{avg}(k) = \frac{H_0^{interp}(k) + H_{-1}^{interp}(k) + H_{-2}^{interp}(k) + H_{-3}^{interp}(k)}{4} \tag{3.42}$$

where $H_n^{interp}(k)$ is the interpolated channel response at the previous nth symbol time.

3.4 Adaptive Channel Estimators [18]

The LMS algorithm is the most widely used adaptive filtering algorithm in practice for its simplicity. Meanwhile, it is stable and robust against different channel conditions.

3.4.1 Model Based on Equalization

The first approach is to model the inverse of the channel response, i.e., $1/H(k)$, and the signal flow is shown in Fig. 3.3, where **X** is the input signal sent into the decision device, **H** is the channel frequency response, and $\mathbf{Y}$ is the channel output. The following equations apply to our work. Note that for the sake of simplicity, $H(n,k|\, n$: time index, k: frequency index) is denoted $H(n)$ and the frequency index is ignored.

- Estimation error:

$$\mathbf{e}(n) = \widehat{\mathbf{X}}(n) - \mathbf{X}(n) \quad \text{where} \quad \mathbf{X}(n) = \mathbf{w}(n) \cdot \mathbf{Y}(n). \tag{3.43}$$

- Weights updating function:

$$\mathbf{w}(n+1) = \mathbf{w}(n) - \nu \nabla ||\mathbf{e}^2(n)||. \tag{3.44}$$

Hence, we get

$$\mathbf{w}(n+1) = \mathbf{w}(n) + 2\nu \mathbf{e}^*(n)\mathbf{Y}(n). \tag{3.45}$$

The $\mathbf{w}(n)$ term is the estimated inverse of the channel frequency response. The final LMS adaptation equation can be written as

$$\frac{1}{\mathbf{H}(n+1)} = \frac{1}{\mathbf{H}(n)} + \mu \mathbf{e}^*(n) \frac{\mathbf{Y}(n)}{||\mathbf{Y}(n)||^2}. \tag{3.46}$$

3.4.2 Model Based on Channel Estimation

The second approach is to model the channel response, as shown in Fig 3.4. The equation is the same as before except that the estimation error is given by

$$\mathbf{e}(n) = \mathbf{Y}(n) - \widehat{\mathbf{Y}}(n) \quad \text{where} \quad \widehat{\mathbf{Y}}(n) = \mathbf{w}(n) \cdot \widehat{\mathbf{X}}(n). \tag{3.47}$$

Further derivation gives

$$\mathbf{w}(n+1) = \mathbf{w}(n) + 2\nu \mathbf{e}^*(n)\widehat{\mathbf{X}}(n). \tag{3.48}$$

Here, the $\mathbf{w}(n)$ is the estimated channel frequency response and the resulting NLMS adaptation equation is

$$\mathbf{H}(n+1) = \mathbf{H}(n) + \mu \mathbf{e}^*(n) \frac{\widehat{\mathbf{X}}(n)}{||\widehat{\mathbf{X}}(n)||^2}. \tag{3.49}$$

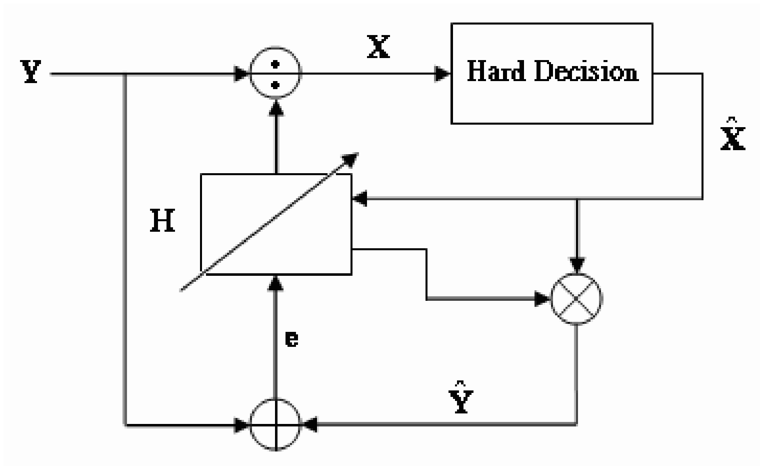


Figure 3.4: NLMS channel estimation.

3.5 Joint Channel Estimation and Symbol Detection

Recall that in the previous chapter, we derived the equation

$$Y(k) = X(k)H(k) + W(k) \tag{3.50}$$

where k is the subcarrier index. According to [19] and [20], the joint maximum likelihood (JML) solution of $X(k)$ and $H(k)$ can be obtained by

$$\{\widehat{X}, \widehat{H}\} = \arg\min p\,(Y|X, H) = \arg\Big\{\min \big|\big|Y - \widehat{X}\widehat{H}\big|\big|^2\Big\}. \tag{3.51}$$

3.5.1 Iterative Joint Maximum Likelihood Approximation

A natural way to approximate JML is to estimate alternately the channel maximizing the conditioned likelihood

$$L(H|\widehat{X}^{(i)}) \propto \exp\Big(-\big|\big|Y - H\widehat{X}^{(i)}\big|\big|^2/\sigma^2\Big) \tag{3.52}$$

by calculating

$$\begin{aligned}\widehat{H}^{(i)} &= \arg\max L(H|\widehat{X}^{(i)}) \\ &= \arg\min \big|\big|Y - H\widehat{X}^{(i)}\big|\big|^2\end{aligned} \tag{3.53}$$

and to detect the data maximizing the conditioned likelihood

$$L(X|\widehat{H}^{(i)}) \propto \exp\Big(-\big|\big|Y - \widehat{H}^{(i)}X\big|\big|^2/\sigma^2\Big) \tag{3.54}$$

by calculating

$$\begin{aligned} \widehat{X}^{(i+1)} &= \arg\max L(X|\widehat{H}^{(i)}) \\ &= \arg\min \big|\big|Y - \widehat{H}^{(i)}X\big|\big|^2, \end{aligned} \tag{3.55}$$

where i denotes the iteration counter.

3.5.2 Algorithm Summary

The algorithm is summarized as follows:

1)

$$\widehat{H}_{LS,\,i=0} = \frac{Y_p}{X_p}$$

where i is the iteration counter. This equation is the same as (3.5) and is repeated here for convenience. Y_p stands for the received signal at each pilot subcarrier; X_p stands for the transmitted signal at each pilot subcarrier and is assumed known for the receiver.

2)

$$\widehat{H}_{DFT,\,i=0} = (B^H B)^{-1} B^H \widehat{H}_{LS,\,i=0}. \tag{3.56}$$

This equation is identical to (3.27).

3)

$$\widehat{X}_{i=0} = \frac{Y}{\widehat{H}_{DFT,\,i=0}} \quad \text{and} \quad \widehat{S}_{i-1} = \text{Hard-Decision}(\widehat{X}_{i-1}) \tag{3.57}$$

where Y is the received signal, that is, it includes information both on data subcarriers and on pilot subcarriers. The data detection operations are done on each subcarrier rather than block detection.

4)

$$\widehat{H}_{LS,\,i} = \frac{Y}{\widehat{S}_{i-1}}. \tag{3.58}$$

Note that at this moment, we have the estimated information on all used subcarriers.

5)

$$\widehat{H}_{DFT,\,i} = (F^H F)^{-1} F^H \widehat{H}_{LS,\,i} \tag{3.59}$$

where $\left[F\right]_{n,k} = e^{-j2\pi k i_n/N},\ 0 \le n \le N_{used} - 1,\ 0 \le k \le L - 1$, with N_{used} being the number of used subcarriers and L the channel length.

6)

$$\widehat{X}_i = \frac{Y}{\widehat{H}_{DFT,\,i}} \quad \text{and} \quad \widehat{S}_i = \text{Hard-Decision}(\widehat{X}_i). \tag{3.60}$$

7) Check to see if $\widehat{S}_i = \widehat{S}_{i-1}$, and if not, go to step 4) to continue the iteration.

Figure 3.5 shows the flowchart of the iterative joint channel estimation and symbol detection (JCESD) algorithm.

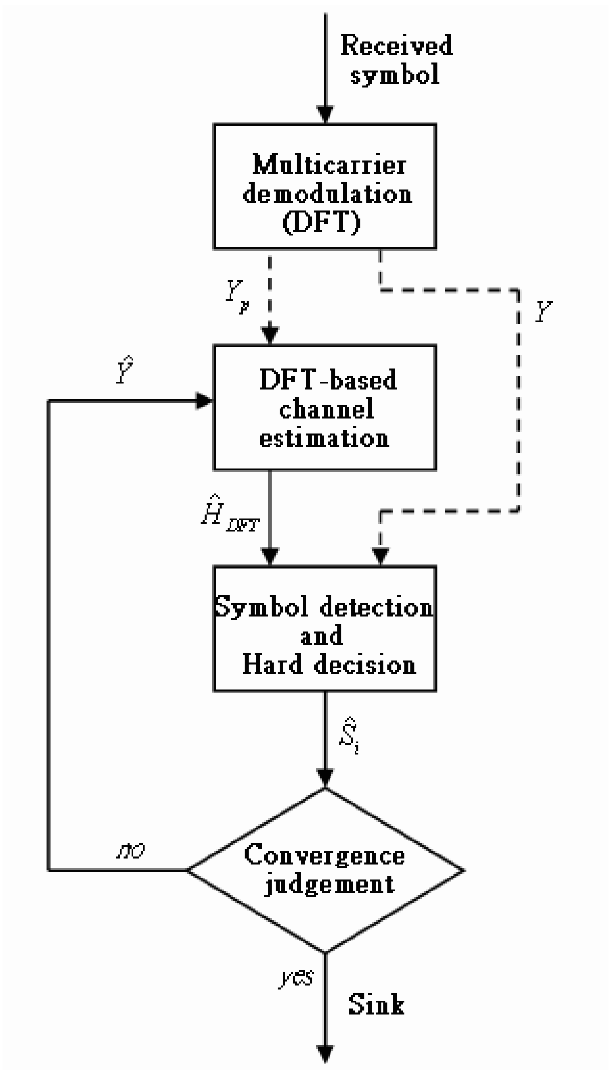

Received symbol
Multicarrier demodulation (DFT)
Y_p
Y
$\hat{Y}$
DFT-based channel estimation
$\hat{H}_{DFT}$
Symbol detection and Hard decision
$\hat{S}_i$
no
Convergence judgement
yes
Sink

Figure 3.5: Flowchart of the iterative joint channel estimation and symbol detection (JCESD) algorithm.

Chapter 4

The DSP Hardware and Associated Software Development Environment

4.1 The TMS320C6416 DSP

The TMS320C64x DSPs are the highest-performance fixed-point DSP generation of the TMS320C6000 DSP devices, with a performance of up to 600 million instructions per second (MIPS) and an efficient C compiler. The TMS320C64x device is based on the second-generation high-performance, very-long-instruction-word (VLIW) architecture developed by Texas Instruments (TI). The C6416 device has two high-performance embedded coprocessors, Viterbi Decoder Coprocessor (VCP) and Turbo Decoder Coprocessor (TCP) that significantly speed up channel-decoding operations on-chip. But they do not apply to the work reported in this thesis. Material in this section is mainly taken from [22].

4.1.1 Features and Options of the TMS320c64x

The C64x core CPU consists of 64 general-purpose 32-bits registers and 8 function units, which are described further below. Features of C6000 devices include :

- Advanced VLIW CPU with eight functional units, including two multipliers and six arithmetic units:

  - 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 compiler 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 applications requiring it.

- Saturation and normalization provide support for key arithmetic operations.

- Field manipulation and instruction extract, set, clear, and bit counting support common operation found in control and data manipulation applications.

The additional features of C64x include:

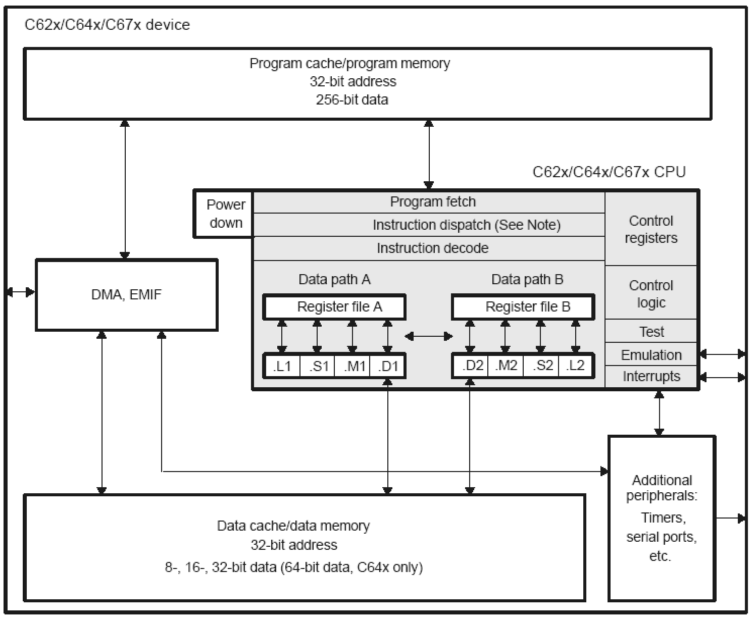


Figure 4.1: Block diagram of the TMS320C6416 DSP [22].

- Each multiplier can perform two 16×16 bits or four 8×8 bits multiplies every clock cycle.
- Quad 8-bit and dual 16-bit instruction set extensions with data flow support.
- Support for non-aligned 32-bit (word) and 64-bit (double word) memory accesses.
- Special communication-specific instructions have been added to address common operations in error-correcting codes.
- Bit count and rotate hardware extends support for bit-level algorithms.

4.1.2 Central Processing Unit

The C64x CPU, shaded in Fig. 4.1, contains:

- Program fetch unit.

- Instruction dispatch unit.
- Instruction decode unit.
- Two data paths, each with four functional units.
- 64 32-bit registers.
- Control registers.
- Control logic.
- Test, emulation, and interrupt logic.

The program fetch, instruction dispatch, and instruction decode units can deliver up to eight 32-bit instructions to the functional units every CPU clock cycle. The processing of instructions occurs in each of the two data paths (A and B), each of which contains four functional units (.L, .S, .M, and .D) and 32 32-bit general-purpose registers for the C6416.

4.1.2.1 Pipeline Structure

The TMS320C64x DSP pipeline provides flexibility to simplify programming and improve performance. The pipeline can dispatch eight parallel instructions every cycle. These two factors provide this flexibility:

- Control of the pipeline is simplified by eliminating pipeline interlocks.
- Increased pipelining eliminates traditional architectural bottlenecks in program fetch, data access, and multiply operations. This provides single cycle throughput.

The pipeline phases are divided into three stages as shown in Fig. 4.2.

- Fetch has 4 phases:

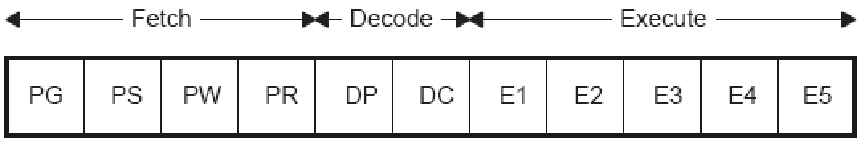


Figure 4.2: Pipeline phases of TMS320C6416 DSP [22].

  - PG (program address generate): The address of the fetch packet is determined.
  - PS (program address send): The address of the fetch packet is sent to memory.
  - PW (program access ready wait): A program memory access is performed.
  - PR (program fetch packet receive): The fatch packet is at the CPU boundary.

- Decode has two phases:

  - DP (instruction dispatch): The next execute packet in the fetch packet is determined and sent to the appropriate functional units to be decoded.
  - DC (instruction decode): Instructions are decoded in functional units.

- Execute has five phases: E1 to E5.

The pipeline operation of the C62x/C64x instructions can be categorized into seven instruction types. Six of these are shown in Table 4.1, which gives a mapping of operations occurring in each execution phase for the different instruction types. The delay slots associated with each instruction type are listed in the bottom row.

The execution of instructions can be defined in terms of delay slots. A delay slot is a CPU cycle that occurs after the first execution phase (E1) of an instruction. Results from instructions with delay slots are not available until the end of the last delay slot. For example, a multiply instruction has one delay slot, which means that one CPU cycle elapses before the results of the multiply are available for use by a subsequent instruction. However,

Table 4.1: Execution Stage Length Description for Each Instruction Type [22]

		Instruction Type					
		Single Cycle	**16 X 16 Single Multiply/ C64x .M Unit Non-Multiply**	**Store**	**C64x Multiply Extensions**	**Load**	**Branch**
Execution phases	E1	Compute result and write to register	Read operands and start computations	Compute address	Reads operands and start computations	Compute address	Target-code in PG‡
	E2		Compute result and write to register	Send address and data to memory		Send address to memory	
	E3			Access memory		Access memory	
	E4				Write results to register	Send data back to CPU	
	E5					Write data into register	
Delay slots		0	1	0†	3	4†	5‡

results are available from other instructions finishing execution during the same CPU cycle in which the multiply is in a delay slot.

4.1.2.2 Functional Units

The eight functional units in the C6000 data paths can be divided into two groups of four; each functional unit in one data path is almost identical to the corresponding unit in the other data path. The functional units are described in Table 4.2.

Besides being able to perform 32-bit operations, the C64x also contains many 8-bit to 16-bit extensions to the instruction set. For example, the MPYU4 instruction performs four 8×8 unsigned multiplies with a single instruction on an .M unit. The ADD4 instruction performs four 8-bit additions with a single instruction on an .L unit.

The data line in the CPU supports 32-bit operands, long (40-bit) and double word (64-bit) operands. Each functional unit has its own 32-bit write port into a general-purpose

Table 4.2: Functional Units and Operations Performed [22]

Function Unit	Operations
.L unit (.L1, .L2)	32/40-bit arithmetic and compare operations
	32-bit logical operations
	Leftmost 1 or 0 counting for 32 bits
	Normalization count for 32 and 40 bits
	Byte shifts
	Data packing/unpacking
	5-bit constant generation
	Dual 16-bit arithmetic operations
	Quad 8-bit arithmetic operations
	Dual 16-bit min/max operations
	Quad 8-bit min/max operations
.S unit (.S1, .S2)	32-bit arithmetic operations
	32/40-bit shifts and 32-bit bit-field operations
	32-bit logical operations
	Branches
	Constant generation
	Register transfers to/from control register file (.S2 only)
	Byte shifts
	Data packing/unpacking
	Dual 16-bit compare operations
	Quad 8-bit compare operations
	Dual 16-bit shift operations
	Dual 16-bit saturated arithmetic operations
	Quad 8-bit saturated arithmetic operations
.M unit (.M1, .M2)	16 x 16 multiply operations
	16 x 32 multiply operations
	Quad 8 x 8 multiply operations
	Dual 16 x 16 multiply operations
	Dual 16 x 16 multiply with add/subtract operations
	Quad 8 x 8 multiply with add operation
	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)
	Load and store double words with 5-bit constant
	Load and store non-aligned words and double words
	5-bit constant generation
	32-bit logical operations

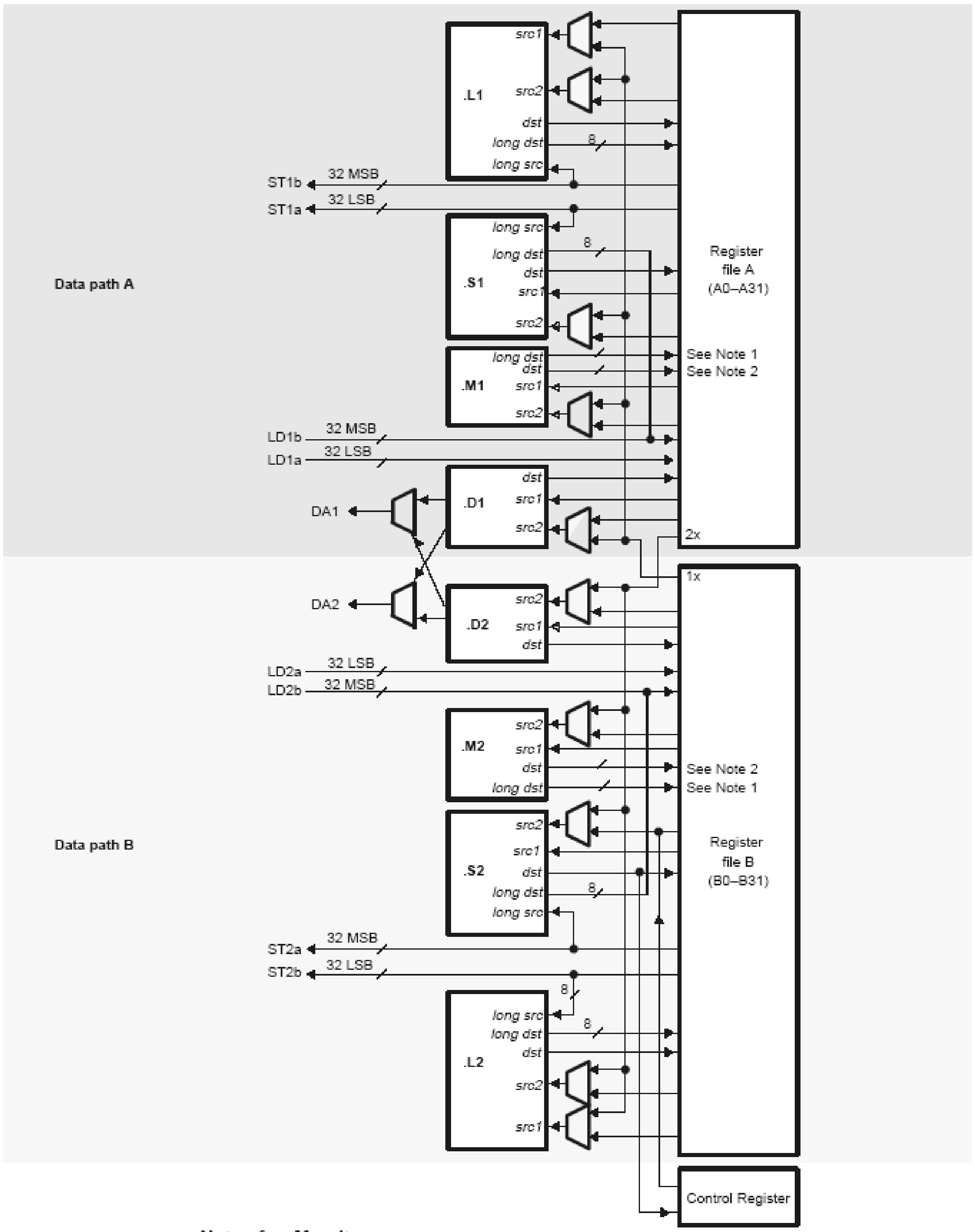
register file (listed in Fig. 4.3). All units ending in 1 (for example, .L1) write to register file A, and all units ending in 2 write to register file B. Each functional unit has two 32-bit read ports for source operands src1 and src2. Four units (.L1, .L2, .S1, and .S2) have an extra 8-bit-wide port for 40-bit long writes, as well as an 8-bit input for 40-bit long reads. Because each unit has its own 32-bit write port, when performing 32-bit operations all eight units can be used in parallel every cycle.

4.1.3 Memory Architecture

The C64x has a 32-bit, byte-addressable address space. Internal (on-chip) memory is organized in separate data and program spaces. When off-chip memory is used, these spaces are unified on most devices to a single memory space via the external memory interface (EMIF). The C64x has two 64-bit internal ports to access internal data memory have and a single internal port to access internal program memory, with an instruction-fetch width of 256 bits.

A variety of memory options are available for the C6000 platform. In our system, the memory types we can use are:

- On-chip RAM, up to 875 MB.
- Program cache.
- 32-bit external memory interface supports SDRAM, SBSRAM, SRAM, and other asynchronous memories.
- Two-level caches [23]. Level 1 cache is split into program (L1P) and data (L1D) cache. Each L1 cache is 16 KB. Level 2 memory is configurable and can be split into L2 SRAM (addressable on-chip memory) and L2 cache for caching external memory locations. The size of L2 is 1 MB. External memory can be several MB large. The



Notes for .M unit:
1. *long dst* is 32 MSB
2. *dst* is 32 LSB

Figure 4.3: TMS320C64x CPU data paths [22].

access time depends on the memory technology used but is typically around 100 to 133 MHz. In our system, the external memory usable by the DSP is a 32 MB SDRAM.

4.2 The Quixote cPCI Board [21]

The Quixote is one of Innovative Integration's Velocia-family baseboard for applications requiring speed and processing power. Quixote features a processing core built around Texas Instruments' fixed-point TMS320C6416 and Xilinx Virtex2 with 32 MB of DSP RAM and 2 MB of FPGA computation RAM (optional). The TI C6416 DSP operating at 600 MHz offers a processing power of 4800 MIPS. The analog IO features of the board include dual channels of 105 MHz A/D and D/A (2 in, 2 out). A block diagram of the Quixote board is shown in Fig. 4.4.

The Quixote board has a 32 MB SDRAM for use by the DSP. When used with the advanced cache controller on the C6416, the SDRAM provides a large, fast external memory pool for DSP data and code. The Quixote has a serial EEPROM for storing data such as board identification, calibration coefficients, and other data that needs to be stored permanently on the board. This memory is 16 Kbits in size. Functions for using the serial EEPROM are included in the Pismo Toolset that allow the software application programmer to easily write and read from the memory without controlling the low-level interface.

The Caliente subsystem handles the details of interacting with the baseboard in streaming mode. There are 3 ways for data transmission between host PC and DSP: data streaming, block mode data streams and message packet I/O.

Data Streaming. To address high-bandwidth data transfer applications, Quixote is capable of continuous transmission and reception of data via the PCI bus, using a mechanism called streaming. When streaming, the target DSP, which must be running a downloaded

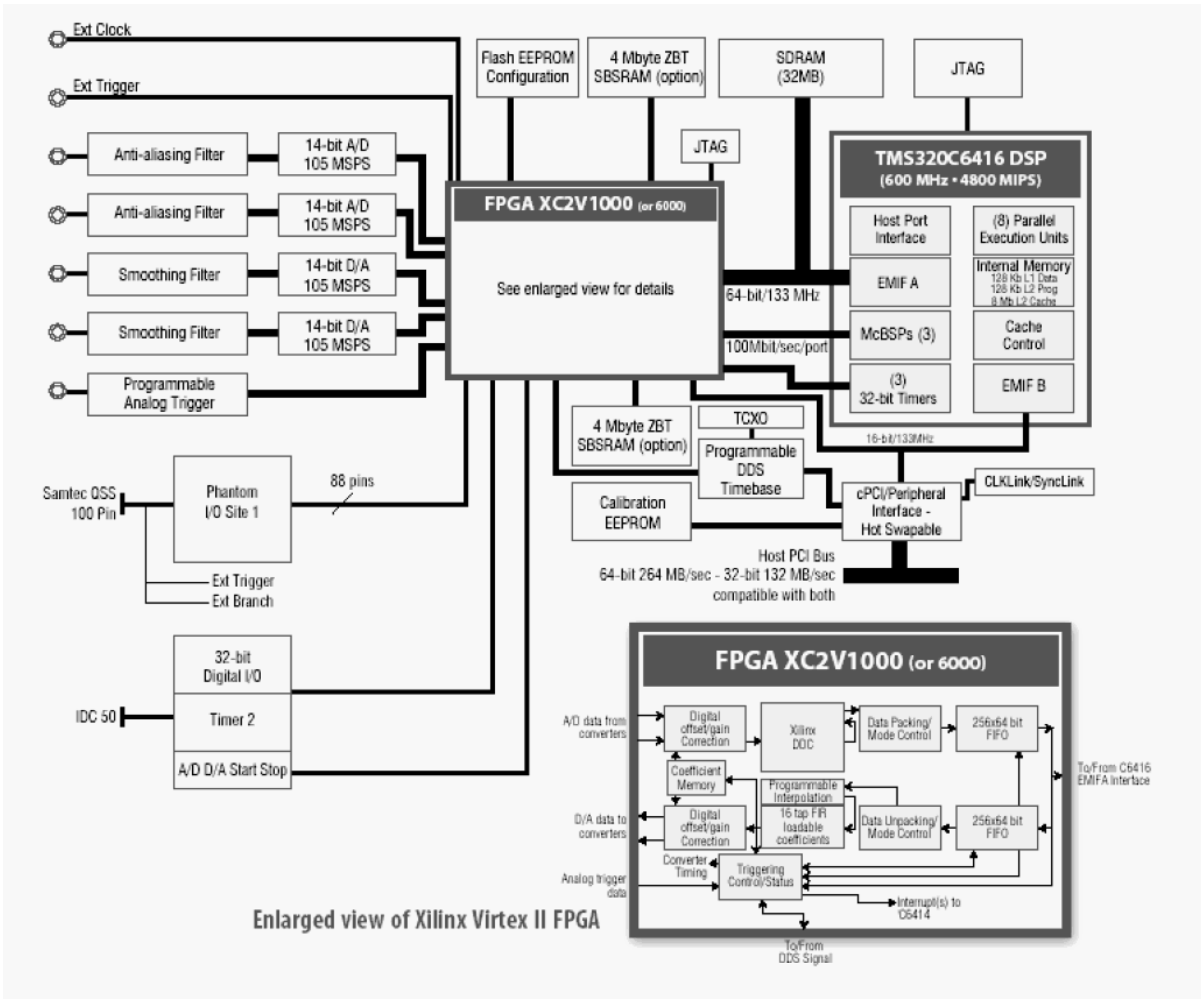


Figure 4.4: Block diagram of the Quixote board [21].

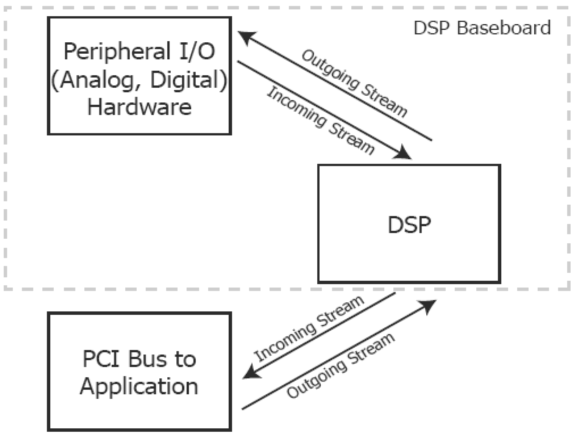


Figure 4.5: Illustration of the DSP streaming mode [21].

DSP application, transfers data between target DSP memory and host PC memory automatically with no host intervention. Streaming input is independent of streaming output. It is possible to acquire data from any number and mix of input devices at a programmed rate. Simultaneously, data may be streamed out to a variety of output devices at a different programmed rate. Data flow is fully controlled by use of device drivers called from within the DSP target application.

During data streaming on baseboards, data flows between peripherals and a dedicated, onboard, digital signal processor (DSP) while simultaneously flowing data between the DSP and the host application software. The dedicated DSP can extensively process data as it travels between peripherals and the host application. Fig. 4.5 illustrates the data streaming operation.

Block Mode Data Streams. An alternate data flow paradigm is supported for non-channelized peripherals. This mode is referred to as block mode streaming. In block mode, the splitter/merger features of Caliente are bypassed, and raw, binary data in peripheral-specific format is consumed and supplied by the application program. Devices that produce data that can be channelized may elect to use block mode because of its higher inherent efficiency. For very high rate applications, any processing done to each point may result in a reduction in the maximum data rate that can be achieved. Since block mode does no implicit processing on a point-by-point basis, the fastest data rates are achievable using this mode.

Message Packet I/O. In many applications, there is a need for additional, low bandwidth channels in addition to a high rate data stream. Velocia baseboards feature a means to support the asynchronous interchange of low-bandwidth data in conjunction with high-bandwidth streaming mode I/O. Messages packets consist of a command code and channel number plus up to 14 additional 32-bit parametric data values. Messages may be asynchronously transmitted and received from any number of distinct channels by any number of threads running on both the target DSP and the host PC. Message transfers have no deleterious effect on data streaming and consume virtually none of the bandwidth of the DSP, so they may be freely used even in conjunction with full rate data streaming.

In our implementation, we use block mode data streams the most and also use message packet I/O [24]. The Virtex2 FPGA includes 18×18 hardware multipliers and contains up to 12 digital clock managers, each providing 256 subdivisions of phase shifting and frequency synthesis capabilities to deliver flexibility in managing both on-chip and off-chip clock domains and synchronization. On-chip memory blocks in the Virtex-II fabric provide convenient high-speed memory elements for FIFOs, dual-port RAM and local processing memory. But our implementation is purely in DSP software and does not make use of the FPGA.

4.3 The Code Composer Studio Development Tools [25], [26]

TI supports a useful GUI development tool set to DSP users for developing and debugging their projects: the Code Composer Studio (CCS). The CCS development tools are a key element of the DSP software and development tools from TI. The fully integrated development environment includes real-time analysis capabilities, easy to use debugger, C/C++ compiler, assembler, linker, editor, visual project manager, simulators, XDS560 and XDS510 emulation drivers and DSP/BIOS support.

Some of CCS's fully integrated host tools include:

- Simulators for full devices, CPU only and CPU plus memory for optimal performance.
- Integrated visual project manager with source control interface, multi-project support and the ability to handle thousands of project files.
- 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 host tooling support (configure, real-time analysis and debug).
- Data transfer for real time data exchange between host and target.
- Profiler to analyze code performance.

CCS also delivers "foundation software" consisting of:

- DSP/BIOS kernel for the TMS320C6000 DSPs.
  - Pre-emptive multi-threading.
  - Interthread communication.
  - Interrupt handling.
- TMS320 DSP Algorithm Standard to enable software reuse.
- Chip Support Libraries (CSL) to simplify device configuration. CSL provides C-program functions to configure and control on-chip peripherals.

TI also supports some optimized DSP functions for the TMS320C64x devices: the TMS320C64x digital signal processor library (DSPLIB). This source code library includes C-callable functions (ANSI-C language compatible) for general signal processing mathematical and vector functions [27]. The routines included in the DSP library are organized as follows:

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

4.4 Code Optimization Methods [28]

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. This makes the compiler do all the laborious work of instruction selection, parallelizing, pipelining, and register allocation, which simplifies the maintenance of the code, as everything resides in a C framework that is simple to maintain, support, and upgrade.

The recommended code development flow for the C6000 involves the phases described in Fig. 4.6. The tutorial section of the Programmer's Guide [28] focuses on phases 1 and phase 2, and the Guide also instructs the programmer about the tuning stage of phase 3. What is learned is the importance of giving the compiler enough information to fully maximize its potential. An added advantage is that this compiler provides direct feedback on the entire program's high MIPS areas (loops). Based on this feedback, there are some simple steps the programmer can take to pass complete and better information to the compiler to maximize the compiler performance.

The following items list the goal for each phase in the software development flow shown in Fig. 4.6.

- Developing C code (phase 1) without any knowledge of the C6000. Use the C6000 profiling tools to identify any inefficient areas that we might have in the C code. To improve the performance of the code, proceed to phase 2.
- Use techniques described in [28] to improve the C code. Use the C6000 profiling tools to check its performance. If the code is still not as efficient as we would like it to be, proceed to phase 3.
- Extract the time-critical areas from the C code and rewrite the code in linear assembly.

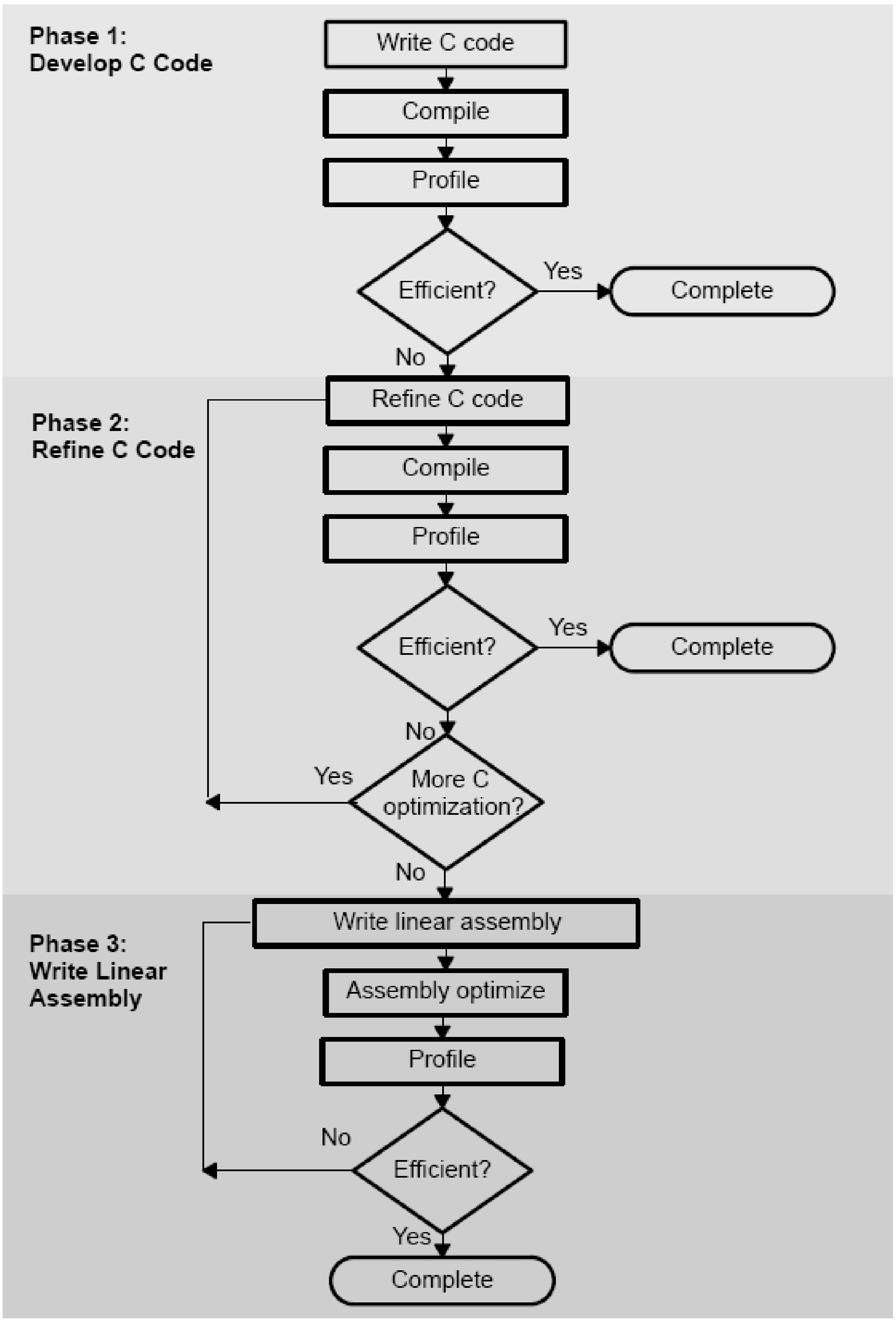


Figure 4.6: Code development flow for TI C6000 DSP [28].

We can use the assembly optimizer to optimize this code.

TI provides high performance C program optimization tools, and they do not suggest the programmer to code by hand in assembly. In this thesis, the development flow is stopped at phase 2. We do not optimize the code by writing linear assembly. Coding the program in high level language keeps the flexibility of porting to other platforms.

4.4.1 Compiler Optimization Options [25], [26]

The compiler supports several options to optimize the code. The compiler options can be used to optimize code size or execution performance. Our primary concern in this work is the execution performance. Hence we do not care very much about the code size. The easiest way to invoke optimization is to use the cl6x shell program, specifying the -on option on the cl6x command line, where n denotes the level of optimization (0, 1, 2, 3) which controls the type and degree of optimization:

- -o0.
  - Performs control-flow-graph simplification.
  - Allocates variables to registers.
  - Performs loop rotation.
  - Eliminates unused code.
  - Simplifies expressions and statements.
  - Expands calls to functions declared inline.
- -o1. Performs all -o0 optimization, and:
  - Performs local copy/constant propagation.

  - Removes unused assignments.
  - Eliminates local common expressions.

- -o2. Performs all -o1 optimizations, and:
  - Performs software pipelining.
  - Performs loop optimizations.
  - Eliminates global common subexpressions.
  - Eliminates global unused assignments.
  - Converts array references in loops to incremented pointer form.
  - Performs loop unrolling.

- -o3. Performs all -o2 optimizations, and:
  - Removes all functions that are never called.
  - Simplifies functions with return values that are never used.
  - Inlines calls to small functions.
  - 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.

The -o2 is the default if -o is set without an optimization level.

The program-level optimization can be specified by using the -pm option with the -o3 option. With program-level optimization, all of the source files are compiled into one intermediate file called a module. The module moves through the optimization and code generation passes of the compiler. Because the compiler can see the entire program, it performs several optimizations that are rarely applied during file-level optimization:

- If a particular argument in a function always has the same value, the compiler replaces the argument with the value and passes the value instead of the argument.
- If a return value of a function is never used, the compiler deletes the return code in the function.
- If a function is not called directly or indirectly, the compiler removes the function.

When program-level optimization is selected in Code Composer Studio, options that have been selected to be file-specific are ignored. The program level optimization is the highest level optimization option. We use this option to optimize our code.

4.4.2 Using Intrinsics

The C6000 compiler provides intrinsics, which are special functions that map directly to C64x instructions, to optimize the C code performance. All instructions that are not easily expressed in C code are supported as intrinsics. Intrinsics are specified with a leading underscore (_) and are accessed by calling them as we call a function. A table of TMS320C6000 C/C++ compiler intrinsics can be found in [28].

Chapter 5

IEEE 802.16e OFDM Uplink Channel Estimation and DSP Implementation

The aim of this chapter is focus on the DSP implementation of IEEE802.16e OFDM uplink. Hence, we use only simple channel estimation techniques, such as linear interpolation and second order interpolation. Detailed comparison between various channel estimation techniques will be left to the next chapter. We evaluate the performance of each channel estimation approach mainly via symbol error rate (SER) and mean square error (MSE).

5.1 Simulation Parameters and Channel Model

This section gives the parameters and introduce the channel model used in our simulation work.

5.1.1 OFDM Uplink System Parameters

In chapter 2, we introduce the primitive parameters and derived parameters of the system. The system parameters used in our simulation are listed in Table 5.1.

Table 5.1: OFDM Uplink Parameters

Parameters	**Values**
Bandwidth	10M Hz
Central frequency	5G Hz
N_{used}	201
Sampling factor n	1
G	$\frac{1}{32}$
N_{FFT}	256
Sampling frequency	11.52M Hz
Subcarrier spacing	45k Hz
Useful symbol time	22.2 μs
CP time	694.4 ns
OFDM symbol time	22.92 μs
Sampling time	86.81 ns

Table 5.2: Power-Delay Profile of the ETSI “Vehicular A” Channel

Tap	Relative Delay (4× oversampled)	Average Power (dB)
1	0	0
2	14	-1.0
3	32	-9.0
4	50	-10.0
5	79	-15.0
6	115	-20.0

5.1.2 Simulation Channel Model

We employ the ETSI “Vehicular A” model with the path delays slightly adjusted for convenience in our simulation. The model is as shown in Table 5.2.

5.2 Simulation Flow and Modified Estimation Techniques

Figure 5.1 illustrates the block diagrams of the simulated system. For channel estimation simulation, we assume perfect synchronization. After channel estimation, we can calculate

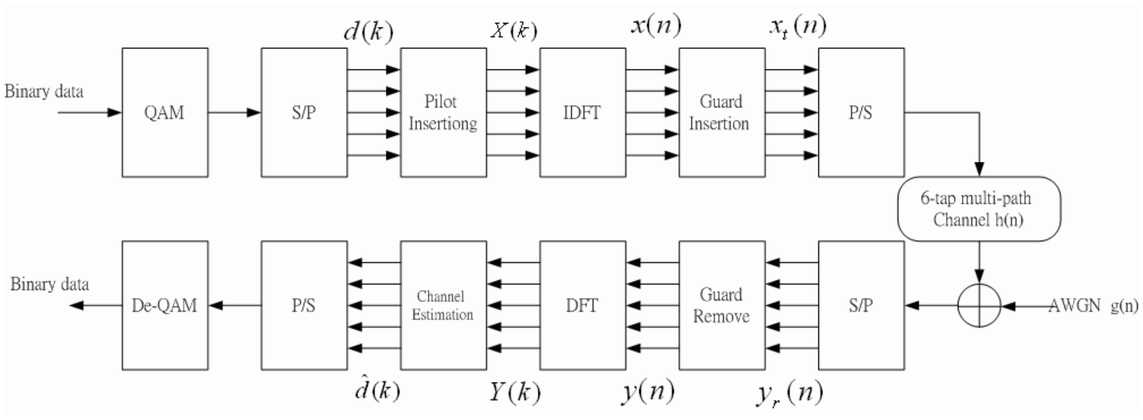
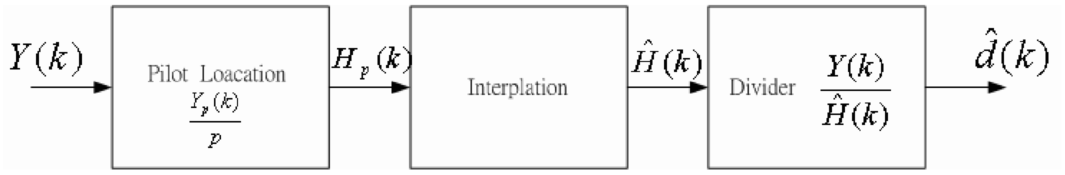


Figure 5.1: Block diagram of the simulated system (from [29]).



$Y(k)$
Pilot Loacation $\frac{Y_p(k)}{p}$
$H_p(k)$
Interplation
$\hat{H}(k)$
Divider $\frac{Y(k)}{\hat{H}(k)}$
$\hat{d}(k)$

Figure 5.2: Channel estimation steps (from [29]).

the channel MSE between the real channel and the estimated one. The symbol error rate (SER) can also be obtained after demapping, where the SER denotes the average symbol error at the specific E_s/N_0. Likewise, the MSE also denotes the average $|\widehat{H} - H|^2$ at the E_s/N_0, where the above averages are taken over the subcarriers.

The channel estimation simulation contains several steps:

- Estimate the channel response at each pilot location.
- Interpolate for the whole channel response using the estimated values at pilot locations.
- Estimate the transmitted signal using a divider.

These steps are illustrated in Fig. 5.2.

5.2.1 Validation of Simulation Model

Before considering multipath channels, we do simulation with an AWGN channel to validate the simulation model, which means we transmit the data through a one-path channel with $h[0] = 1$, and then add AWGN noise to it. We validate this model by comparing theoretical SER curves and SER curves resulting from simulations. This comparison is illustrated here for uncoded QPSK modulations.

Expressions for the symbol error rate of rectangular QAM are not hard to derive but yield rather unpleasant expression. For an even number of bits per symbol, exact expressions are available. They are most easily expressed in a per carrier sense:

$$P_s \cong 4\left(1 - \frac{1}{\sqrt{M}}\right) Q\left(\sqrt{\frac{3}{M-1}\frac{E_{av}}{N_0}}\right) \tag{5.1}$$

where

- M = Number of symbols in modulation constellation,
- E_{av} = Average signal energy,
- N_0 = Noise power spectral density (W/Hz),
- P_s = Probability of a symbol-error,
- $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-t^2/2} dt, \ x \geq 0.$

In Fig. 5.3, the theoretical symbol error rate (SER) curve versus E_s/N_0 for uncoded QPSK is plotted together with the SER curve resulting from the simulation. In this figure, we do not take the channel estimation error into account. This validates the simulation (we use C/C++ programming language and TI's code composer studio).

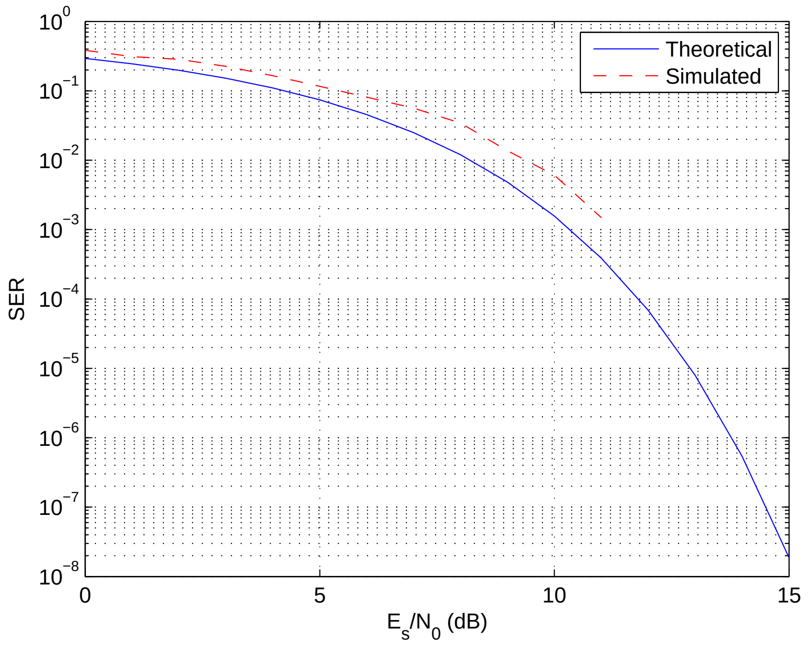
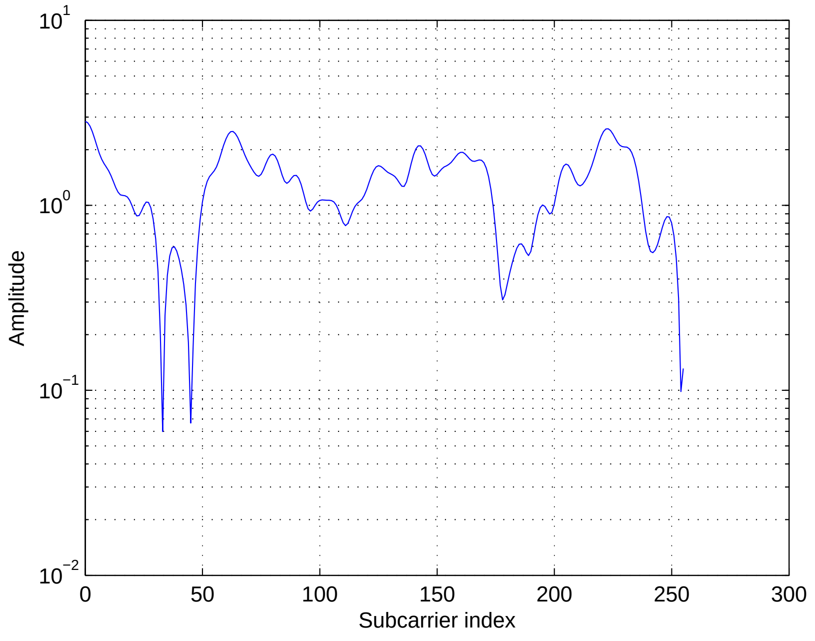


Figure 5.3: The SER curve for uncoded QPSK resulting from simulation matches the theoretical one.

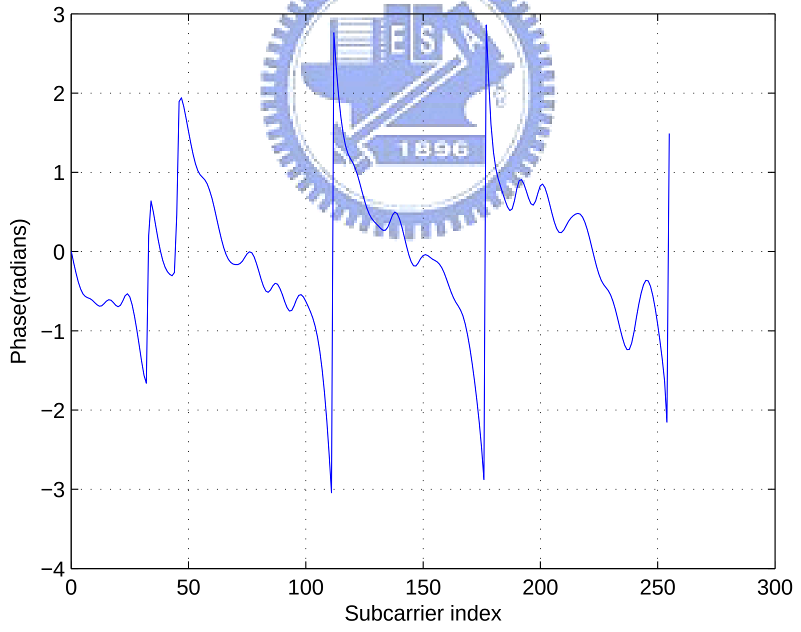
5.2.2 Utilizing Preamble in Static Multipath Channel

As mentioned before, we adopt the ETSI Vehicular A channel model and the channel impulse response is listed in Table 5.2. The amplitude and phase responses of this channel at a particular time instant are shown in Fig. 5.4.

The utilization of the preamble is vital in OFDM uplink channel estimation. If we do not utilize the preamble, the performance would be as shown in Fig. 5.5. In this figure, we shown the performance resulting from one-dimensional linear interpolation (1D-linear), one-dimensional second-order interpolation (1D-2nd), maximum likelihood estimation (MLE) and time averaging. Obviously, the performance is very poor. Hence, we must use the preamble information in our channel estimation work. A simple but not theory-based method is as follows:



(a)



(b)

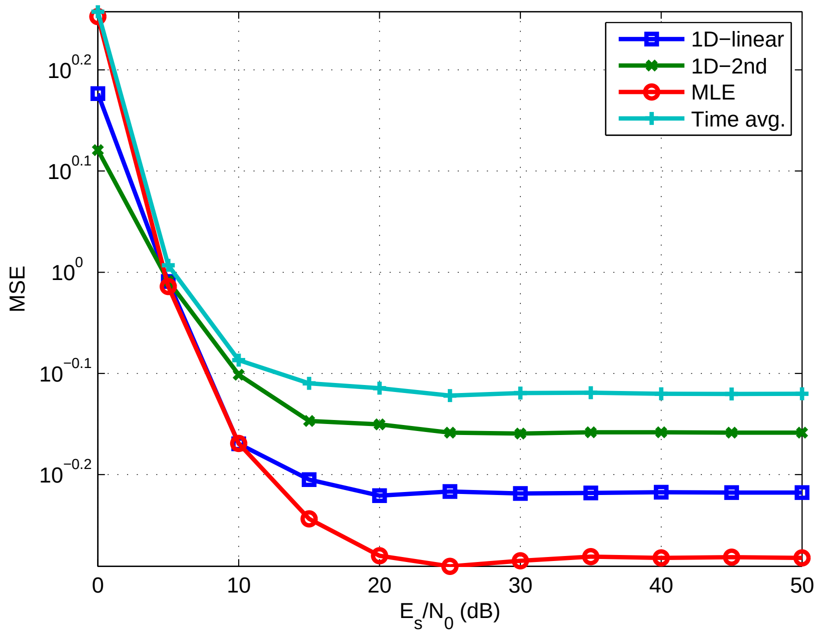
Figure 5.4: (a) Amplitude response and (b) phase response of the multipath channel at a certain time instant.

- Each block starts with a preamble and block length is about 10 to 20 OFDM symbols, depending on the channel coherence time.
- In the uplink, the preamble utilizes only even subcarriers.
- We interpolate the channel response for all subcarriers using the estimates on even subcarriers. In consequence, we have raw channel estimation on all subcarriers, called $H_{preamble}(k)$.
- When receiving usual OFDM symbols, we subtract $H_{preamble}(k_p)$ from $H_{symbol}(k_p)$, where k_p is the pilot subcarrier index and $H_{symbol}(k_p)$ is the estimated channel response in ordinary OFDM symbol.
- Then, we interpolate the difference $H_{symbol}(k_p) - H_{preamble}(k_p)$ to fill in all used subcarriers, called $H_{diff}(k)$.
- Finally, add $H_{diff}(k)$ to $H_{preamble}(k)$ to derive the final channel estimate.

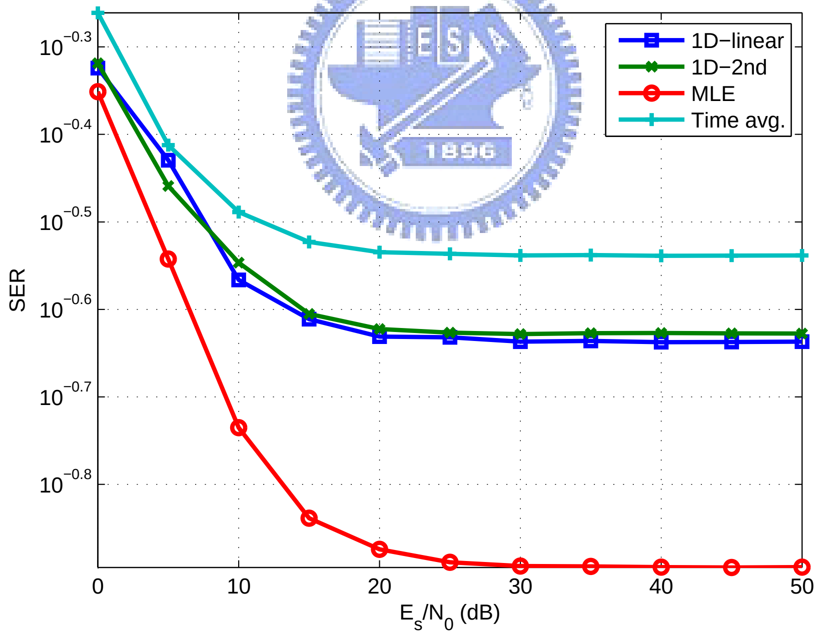
5.3 DSP Implementation

5.3.1 Fixed-Point Implementation

In algorithm development, it is often convenient to employ floating-point computation to acquire better accuracy. However, for the sake of power consumption, execution speed, and hardware costs, practical implementations usually adopt fixed-point computations. The DSP chip used in our work, TI's TMS320C6416 is also of the fixed-point category. It means that fixed-point computations are executed more efficiently than floating-point ones on this platform. Due to these facts, we do simulation in 16-bit fixed-point domain, which is more accurate than 8-bit computation. Meanwhile, compared with 32-bit computation, it has better efficiency and negligible accuracy loss.



(a)



(b)

Figure 5.5: (a) MSE and (b) SER for QPSK in multipath channel estimation without utilization of the preamble.

Table 5.3: Q1.14 Bit Fields

Bits	15	14	13	...	1	0
Value	S	QI0	Q14	...	Q1	Q0

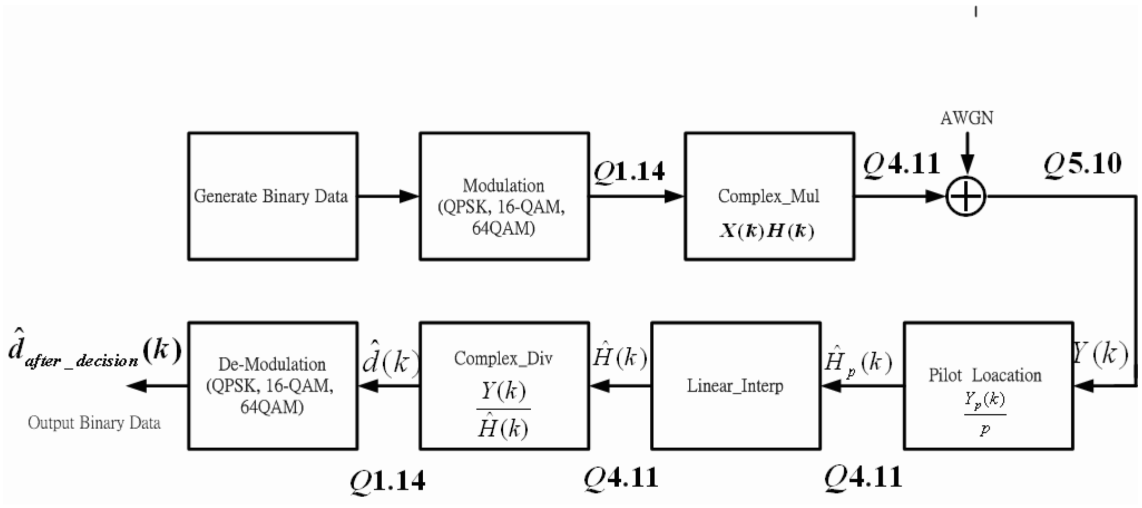


Figure 5.6: Fixed-point data formats in our design.

Taking modulation into account, the widest range occurs in the case of 64QAM, which is $[-7/\sqrt{42}, 7/\sqrt{42}]$. Therefore we must have at least one bit for the integer part, one bit for sign, with the remaining 14 bits for fractional part. Hence Q1.14 is the chosen data format as shown in Table 5.3.

The fixed-point data formats used in our design based on linear frequency-domain interpolation are as shown in Fig. 5.6.

5.3.2 Code Profile

Table 5.4 shows the code size and the execution speed of the DSP implementation, where "load" gives the fractional consumption of a DSP's real-time computing power of each function. Each function in the table corresponds to a block shown in Fig. 5.6. (When "Second_Interp" is used, it replaces the "Linear_Interp" function shown in Fig. 5.6.) Note that

Table 5.4: Profile of the Channel Estimator Implementation

Function	Code Size (bytes)	Max. Count (cycles)	Min. Count (cycles)	Avg. Count (cycles)	Load (# DSPs)
Modulation (QPSK)	168	604	603	603	0.04
Modulation (16QAM)	296	9039	9039	9039	0.598
Modulation (64QAM)	484	12568	12432	12498	0.826
Complex_Mul	844	1479	1479	1479	0.098
Linear_Interp	916	6980	6872	6923	0.46
Second_Interp	936	15945	15912	15930	1.054
Complex_Div	360	20561	20241	20443	1.352
Demodulation (QPSK)	236	634	634	634	0.042
Demodulation (16QAM)	444	11763	11763	11763	0.778
Demodulation (64QAM)	1052	18462	16550	17345	1.147

the key functions for channel estimation are "Linear_Interp" and "Second_Interp." The other functions merely play a supporting role in this part of the work. For example, the "Modulation" functions have been addressed more fully in the synchronization part of the study, and the "Demodulation" functions should be considered in the channel decoding part.

Acknowledgeably, the implemented "Linear_interp" and "Second_interp" functions have not been fully optimized. For example, we have not eliminated all the divisions shown in the earlier equations for linear interpolation and second-order interpolation. The functions are expected to run faster when these divisions are replaced, for example, by multiplications with the dividers' inverses.

We list the program structure as well as the fixed-point c code and assembly code of some basic functions in appendix.

5.4 Simulation Results

We examine the transmission performance of the implemented channel estimator by simulating transmission over multipath channels. The simulated multipath channel is derived from ETSI's "Vehicular A" channel given in Table 5.2, with the following characteristics: 1) each path delay is rounded to an integer times the sample spacing for convenience of simulation; 2) we make the channel is static, also for convenience of simulation; and 3) each path coefficient is the square root of its average power.

Fig. 5.7 shows the MSE in channel estimtion for multipath channel and the resulting SNR, where the solid represents the linear interpolation simulation and dashed line represents the second-order interpolation. And It shows that the second-order interpolation outperforms the liner interpolation.

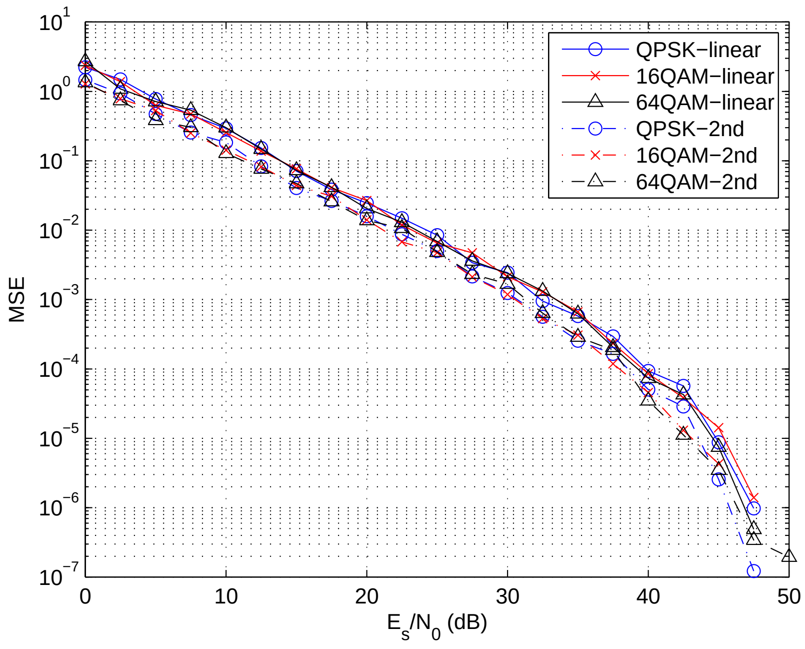
Fig. 5.8 shows the MSE and SER spread over all used subcarrier. By observing the channel gain spread in this figure, we can find that the best subcarrier is about 60 and the worst subcarrier is about 180. Fig. 5.9 shows the performance at the best and the worst subcarriers. It shows that the performance at the worst subcarrier has higher noise flow. This point corresponds to our expectation.

5.5 Appendix

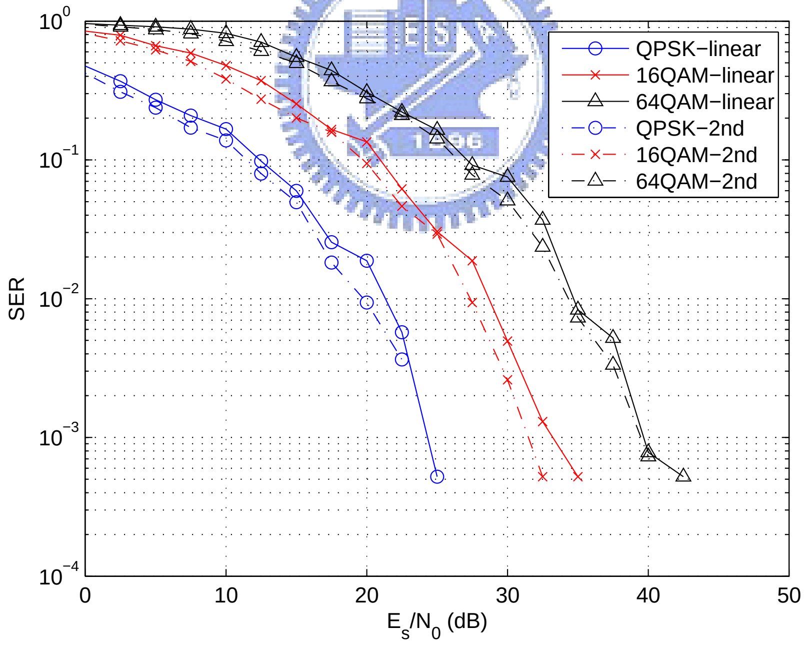
Fig. 5.10 shows program structure of the implemented system, where the key function in channel estimation is Linear-Interp and SecondOrder-Interp; other are in supporting role.

Function *Modulation(QPSK, 16QAM, 64QAM)* maps binary data to the constellation points. We only show the fixed-point code for QPSK in Fig. 5.11 for example.

Function *Complex_Mul* is a multiplier which computes complex multiplication to simulate channel effects. The original code is shown in Fig. 5.12.



(a)



(b)

Figure 5.7: (a)MSE and (b)SER for different interpolation order.

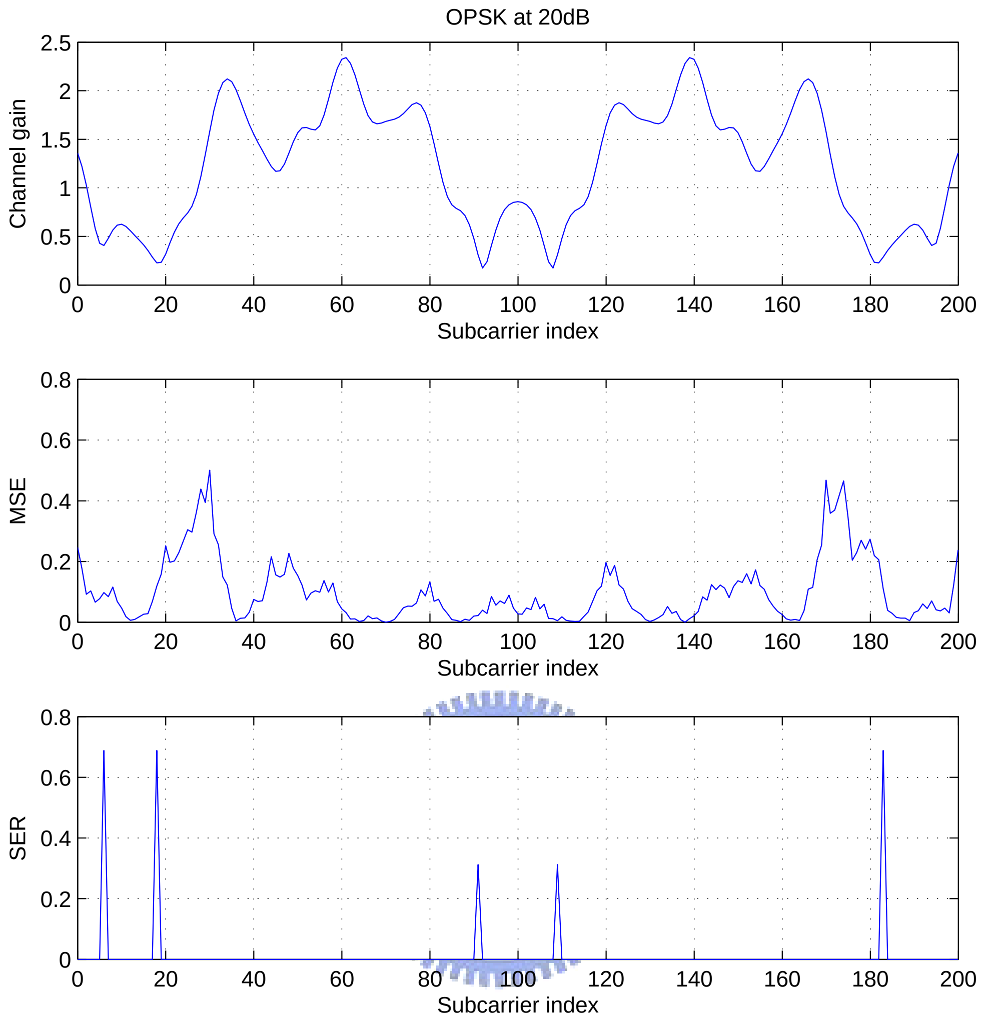
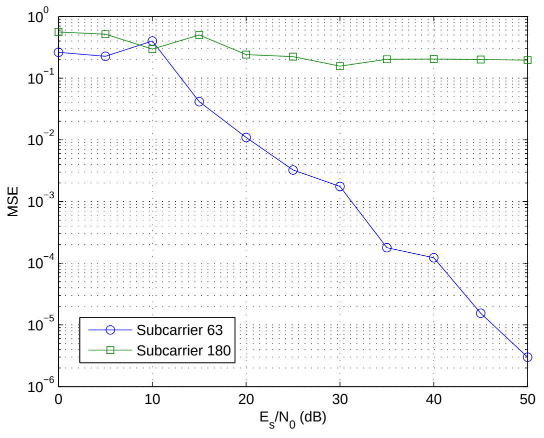


Figure 5.8: MSE and SER spread over used subcarriers in QPSK at 20 dB.

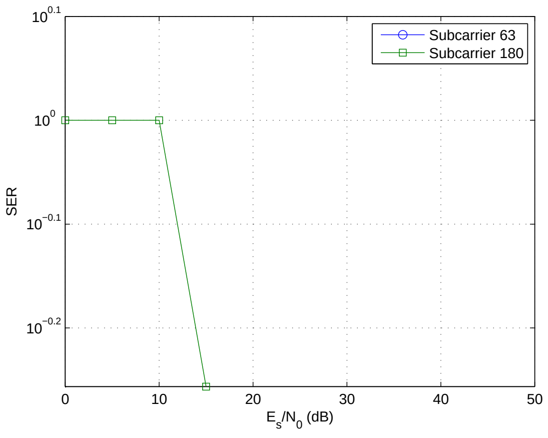
We add AWGN in the main function instead of an individual function.

The operation in the block *Pilot Location* is that received signal $\mathbf{Y(f)}$ is multiplied by p=1 or -1 at pilot locations, i.e., the LS estimator. This function is for convenience in simulation; in real system implementation it can be absorbed into later block.

Function *Linear_Interp* and *SecondOrder_Interp* are the interpolation part which plays



(a)



(b)

Figure 5.9: Performance at worst and best subcarriers. (a) MSE. (b) SER.

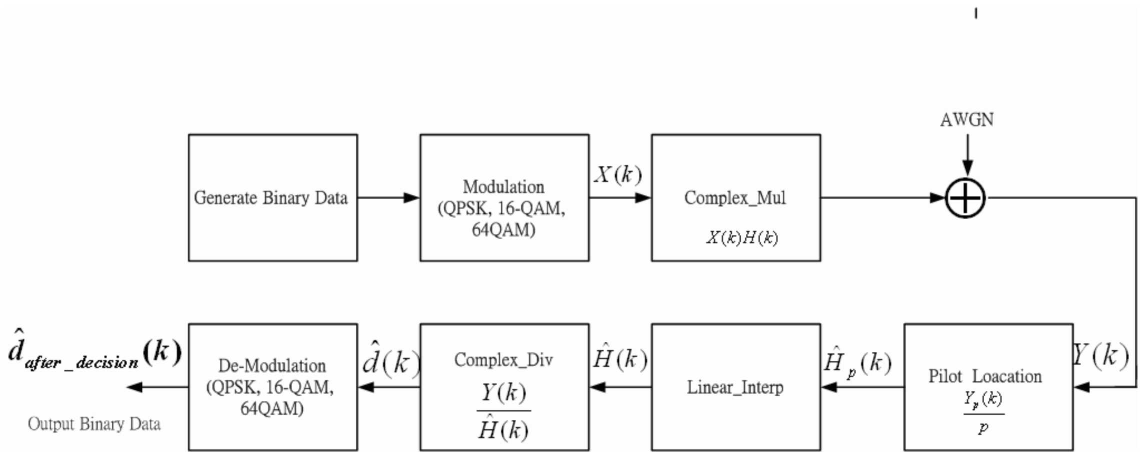
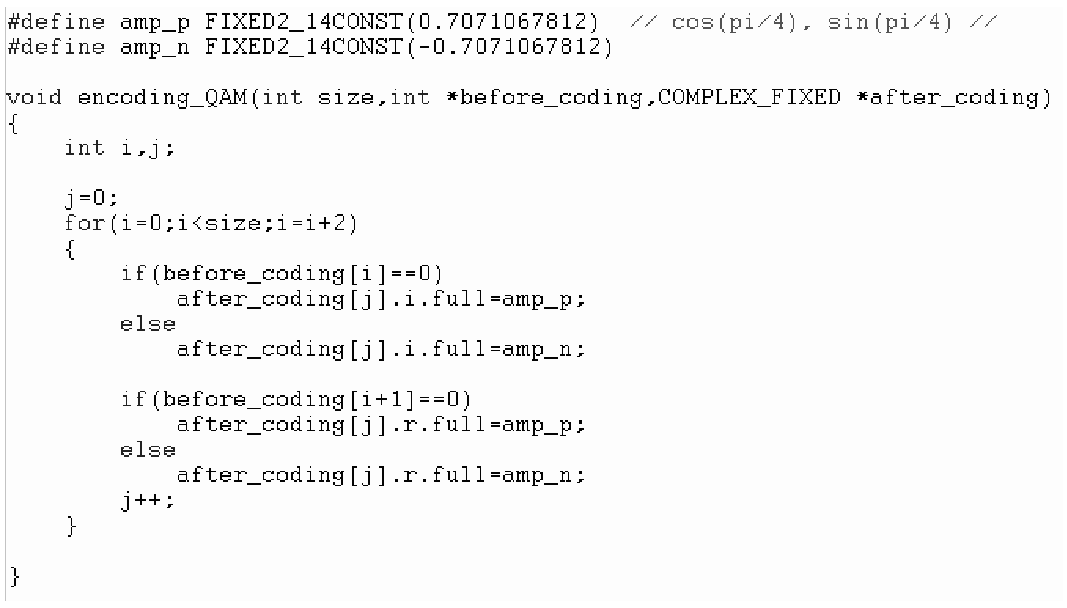

Generate Binary Data
Modulation (QPSK, 16-QAM, 64QAM)
$X(k)$
Complex_Mul $X(k)H(k)$
AWGN
$Y(k)$
Pilot Loacation $\frac{Y_p(k)}{p}$
$\hat{H}_p(k)$
Linear_Interp
$\hat{H}(k)$
Complex_Div $\frac{Y(k)}{\hat{H}(k)}$
$\hat{d}(k)$
De-Modulation (QPSK, 16-QAM, 64QAM)
$\hat{d}_{after\_decision}(k)$
Output Binary Data

Figure 5.10: Program structure for channel estimation.

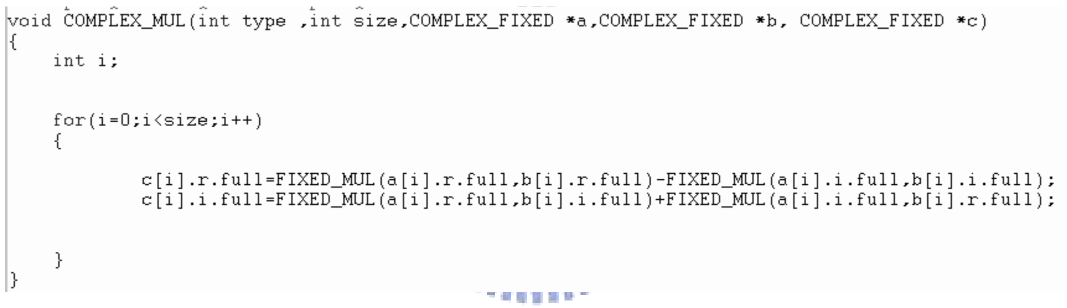
an important role in the channel estimation scheme. The original code is shown in Figs. 5.13 and 5.14.

Function *Complex_Div* is an equalizer where received signal is divided by the estimated channel response and $\hat{d}(k)$ is the output. The code is shown in Fig. 5.15. This function is also for convenience of simulation; in real implementation its function can be absorbed in the demodulator and the decoder.



```c
#define amp_p FIXED2_14CONST(0.7071067812)  // cos(pi/4), sin(pi/4) //
#define amp_n FIXED2_14CONST(-0.7071067812)

void encoding_QAM(int size,int *before_coding,COMPLEX_FIXED *after_coding)
{
    int i,j;

    j=0;
    for(i=0;i<size;i=i+2)
    {
        if(before_coding[i]==0)
            after_coding[j].i.full=amp_p;
        else
            after_coding[j].i.full=amp_n;

        if(before_coding[i+1]==0)
            after_coding[j].r.full=amp_p;
        else
            after_coding[j].r.full=amp_n;
        j++;
    }

}
```

Figure 5.11: Function *Modulation (QPSK)*.



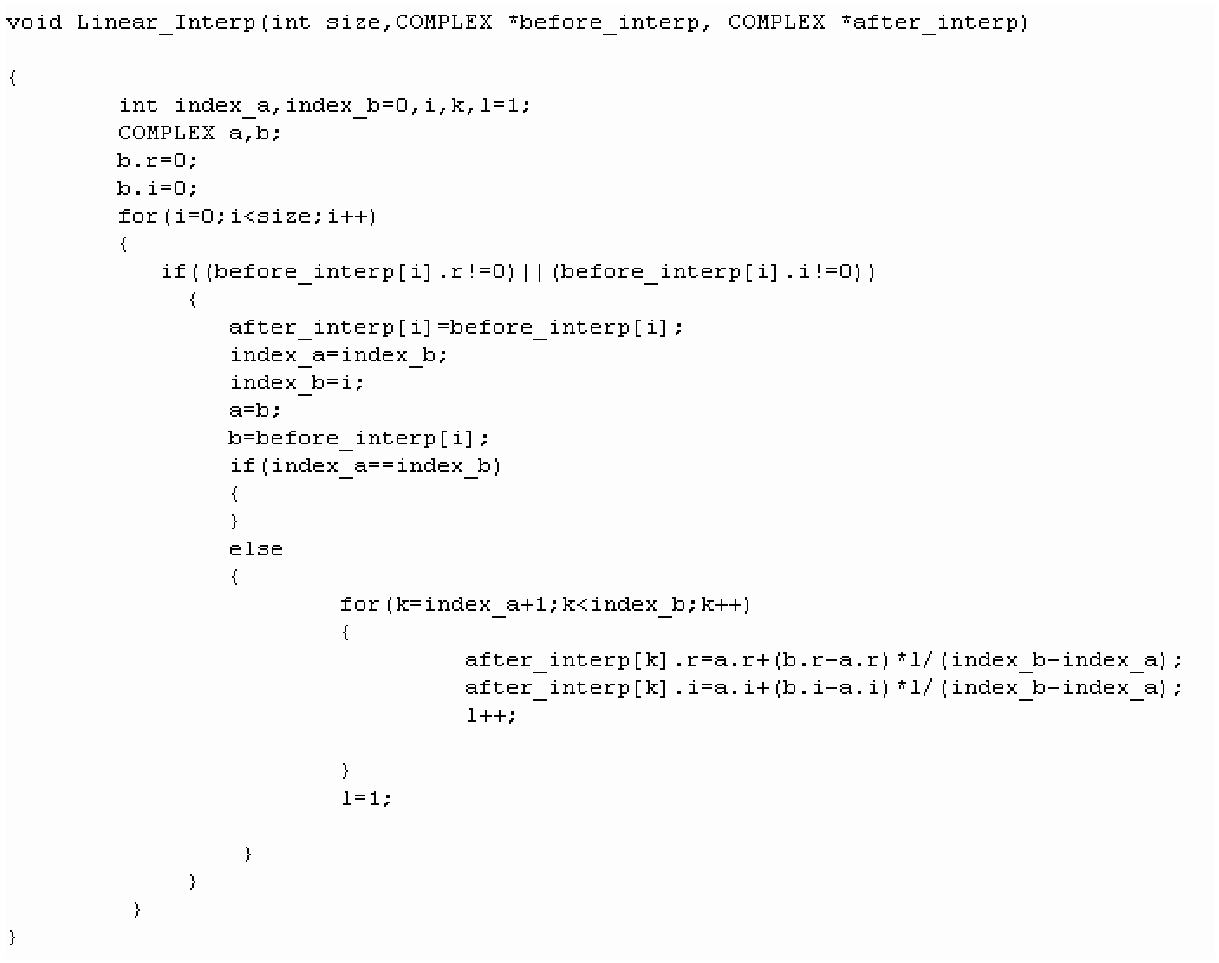
```c
void COMPLEX_MUL(int type ,int size,COMPLEX_FIXED *a,COMPLEX_FIXED *b, COMPLEX_FIXED *c)
{
    int i;

    for(i=0;i<size;i++)
    {

            c[i].r.full=FIXED_MUL(a[i].r.full,b[i].r.full)-FIXED_MUL(a[i].i.full,b[i].i.full);
            c[i].i.full=FIXED_MUL(a[i].r.full,b[i].i.full)+FIXED_MUL(a[i].i.full,b[i].r.full);


    }
}
```

Figure 5.12: Function *Complex_Mul*.

Function *De_Modulation* is the de-mapping function which outputs binary data and the mapped data $\hat{d}_{after\_decision}$ in the constellation. The original code is shown in Fig. 5.16.

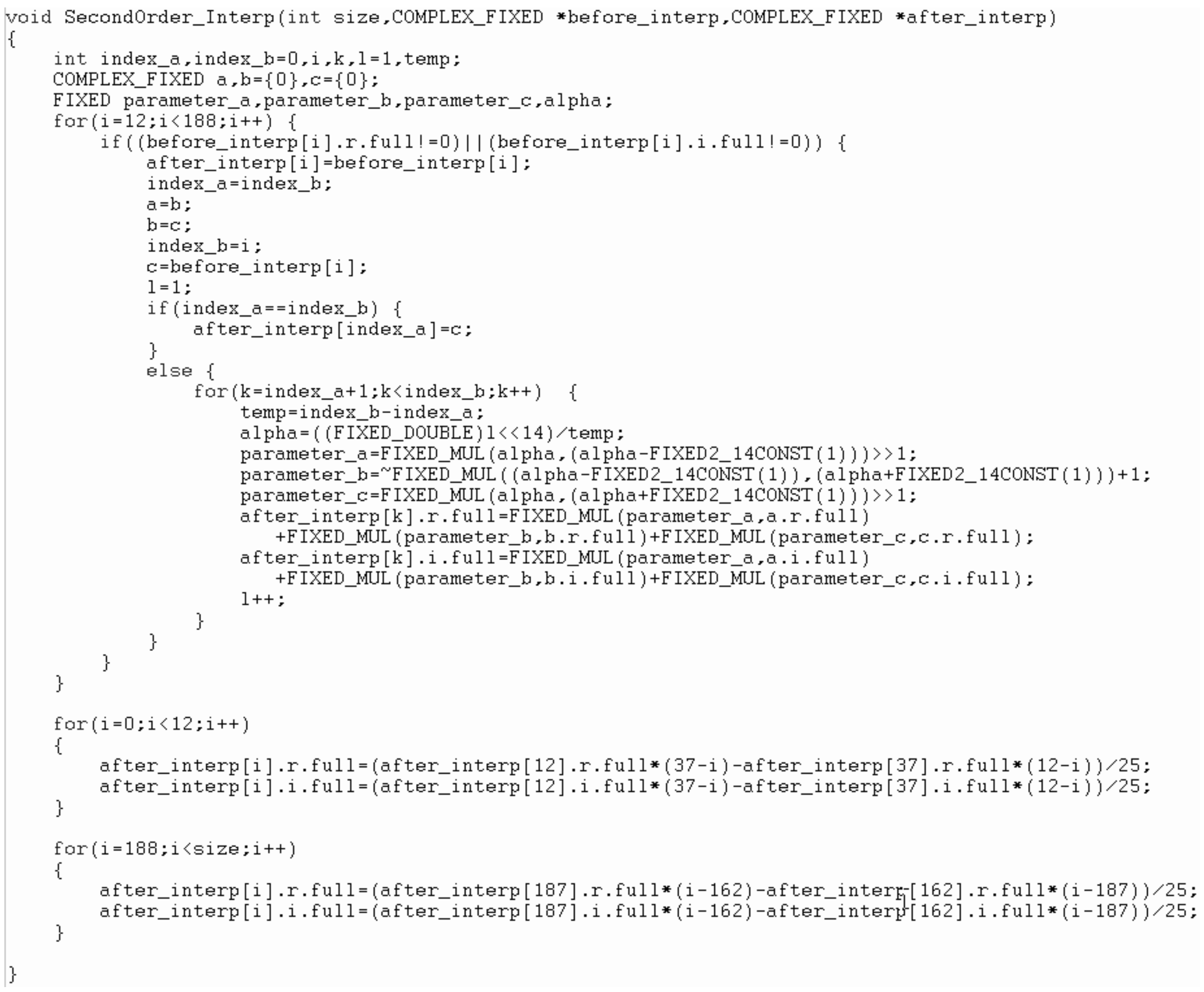
Their corresponding assembly codes are listed below:

We use -o3 option to do optimization. From the compile feedback information, we show that the function *Complex_Mul* can be accelerated most because it utilize most registers after software pipelining, as shown in Fig. 5.23.



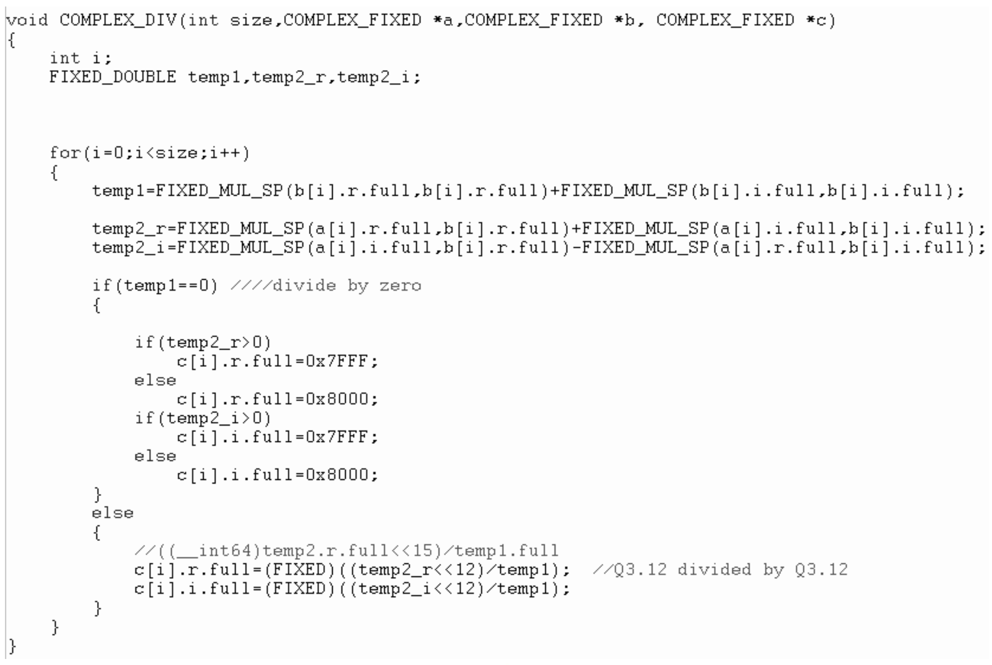
```
void Linear_Interp(int size,COMPLEX *before_interp, COMPLEX *after_interp)

{
        int index_a,index_b=0,i,k,l=1;
        COMPLEX a,b;
        b.r=0;
        b.i=0;
        for(i=0;i<size;i++)
        {
           if((before_interp[i].r!=0)||(before_interp[i].i!=0))
             {
                after_interp[i]=before_interp[i];
                index_a=index_b;
                index_b=i;
                a=b;
                b=before_interp[i];
                if(index_a==index_b)
                {
                }
                else
                {
                        for(k=index_a+1;k<index_b;k++)
                        {
                                after_interp[k].r=a.r+(b.r-a.r)*l/(index_b-index_a);
                                after_interp[k].i=a.i+(b.i-a.i)*l/(index_b-index_a);
                                l++;

                        }
                        l=1;

                 }
             }
         }
}
```

Figure 5.13: Function *Linear_Interp*.



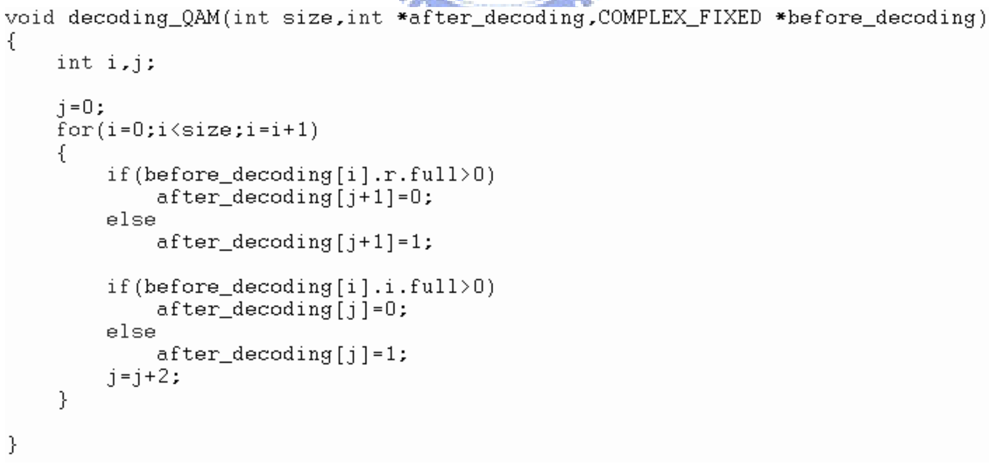
```
void SecondOrder_Interp(int size,COMPLEX_FIXED *before_interp,COMPLEX_FIXED *after_interp)
{
    int index_a,index_b=0,i,k,l=1,temp;
    COMPLEX_FIXED a,b={0},c={0};
    FIXED parameter_a,parameter_b,parameter_c,alpha;
    for(i=12;i<188;i++) {
        if((before_interp[i].r.full!=0)||(before_interp[i].i.full!=0)) {
            after_interp[i]=before_interp[i];
            index_a=index_b;
            a=b;
            b=c;
            index_b=i;
            c=before_interp[i];
            l=1;
            if(index_a==index_b) {
                after_interp[index_a]=c;
            }
            else {
                for(k=index_a+1;k<index_b;k++)  {
                    temp=index_b-index_a;
                    alpha=((FIXED_DOUBLE)l<<14)/temp;
                    parameter_a=FIXED_MUL(alpha,(alpha-FIXED2_14CONST(1)))>>1;
                    parameter_b=~FIXED_MUL((alpha-FIXED2_14CONST(1)),(alpha+FIXED2_14CONST(1)))+1;
                    parameter_c=FIXED_MUL(alpha,(alpha+FIXED2_14CONST(1)))>>1;
                    after_interp[k].r.full=FIXED_MUL(parameter_a,a.r.full)
                       +FIXED_MUL(parameter_b,b.r.full)+FIXED_MUL(parameter_c,c.r.full);
                    after_interp[k].i.full=FIXED_MUL(parameter_a,a.i.full)
                       +FIXED_MUL(parameter_b,b.i.full)+FIXED_MUL(parameter_c,c.i.full);
                    l++;
                }
            }
        }
    }

    for(i=0;i<12;i++)
    {
        after_interp[i].r.full=(after_interp[12].r.full*(37-i)-after_interp[37].r.full*(12-i))/25;
        after_interp[i].i.full=(after_interp[12].i.full*(37-i)-after_interp[37].i.full*(12-i))/25;
    }

    for(i=188;i<size;i++)
    {
        after_interp[i].r.full=(after_interp[187].r.full*(i-162)-after_interp[162].r.full*(i-187))/25;
        after_interp[i].i.full=(after_interp[187].i.full*(i-162)-after_interp[162].i.full*(i-187))/25;
    }

}
```

Figure 5.14: Function *SecondOrder_Interp*.



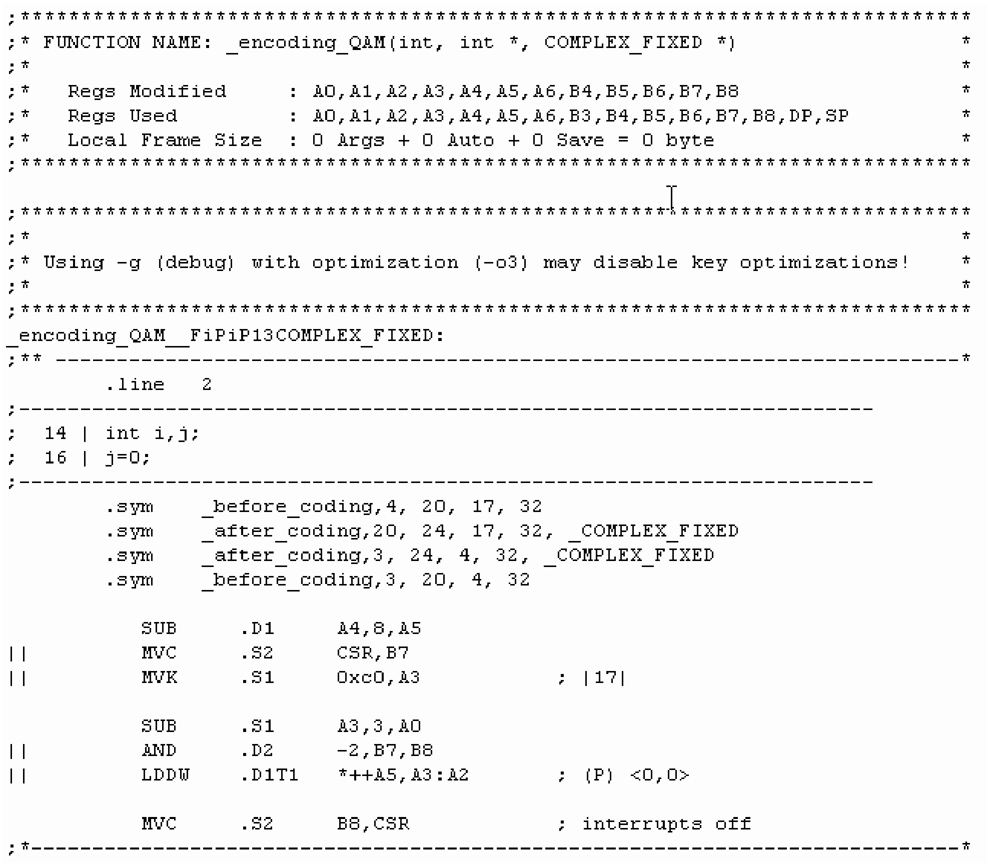
```
void COMPLEX_DIV(int size,COMPLEX_FIXED *a,COMPLEX_FIXED *b, COMPLEX_FIXED *c)
{
    int i;
    FIXED_DOUBLE temp1,temp2_r,temp2_i;


    for(i=0;i<size;i++)
    {
        temp1=FIXED_MUL_SP(b[i].r.full,b[i].r.full)+FIXED_MUL_SP(b[i].i.full,b[i].i.full);

        temp2_r=FIXED_MUL_SP(a[i].r.full,b[i].r.full)+FIXED_MUL_SP(a[i].i.full,b[i].i.full);
        temp2_i=FIXED_MUL_SP(a[i].i.full,b[i].r.full)-FIXED_MUL_SP(a[i].r.full,b[i].i.full);

        if(temp1==0) ////divide by zero
        {

            if(temp2_r>0)
                c[i].r.full=0x7FFF;
            else
                c[i].r.full=0x8000;
            if(temp2_i>0)
                c[i].i.full=0x7FFF;
            else
                c[i].i.full=0x8000;
        }
        else
        {
            //((__int64)temp2.r.full<<15)/temp1.full
            c[i].r.full=(FIXED)((temp2_r<<12)/temp1);  //Q3.12 divided by Q3.12
            c[i].i.full=(FIXED)((temp2_i<<12)/temp1);
        }
    }
}
```

Figure 5.15: Function *Complex_Div*.



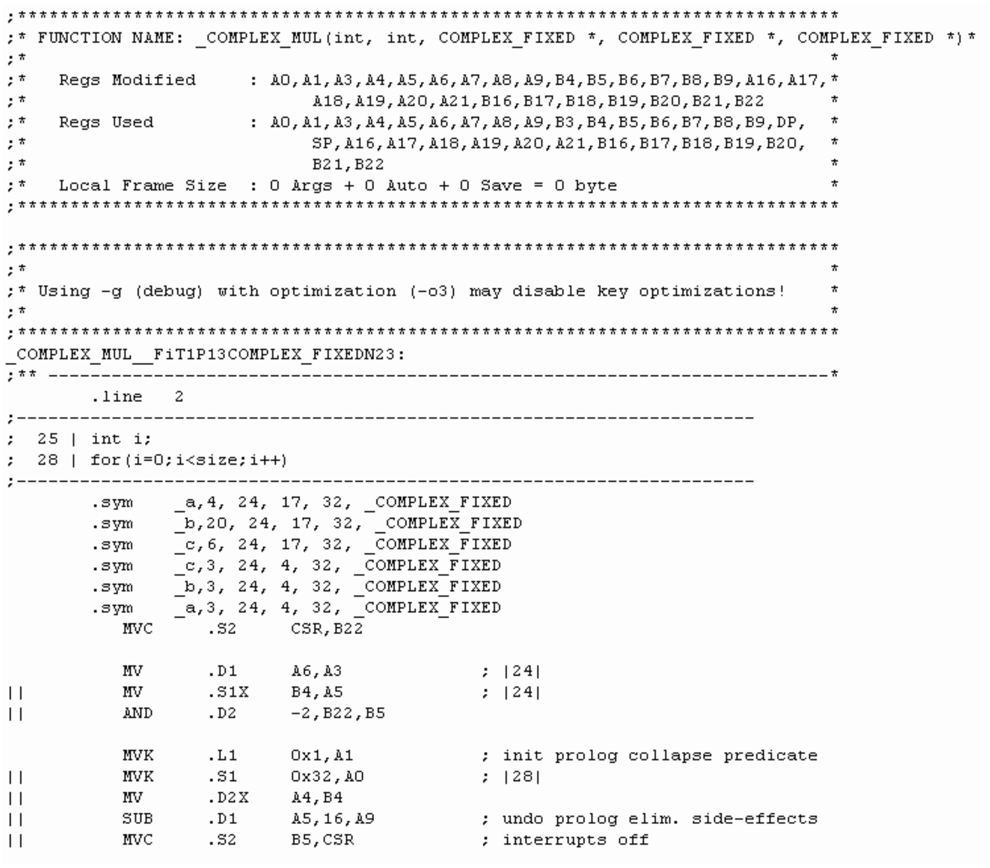
```
void decoding_QAM(int size,int *after_decoding,COMPLEX_FIXED *before_decoding)
{
    int i,j;

    j=0;
    for(i=0;i<size;i=i+1)
    {
        if(before_decoding[i].r.full>0)
            after_decoding[j+1]=0;
        else
            after_decoding[j+1]=1;

        if(before_decoding[i].i.full>0)
            after_decoding[j]=0;
        else
            after_decoding[j]=1;
        j=j+2;
    }

}
```

Figure 5.16: Function *De-modulation(QPSK)*.



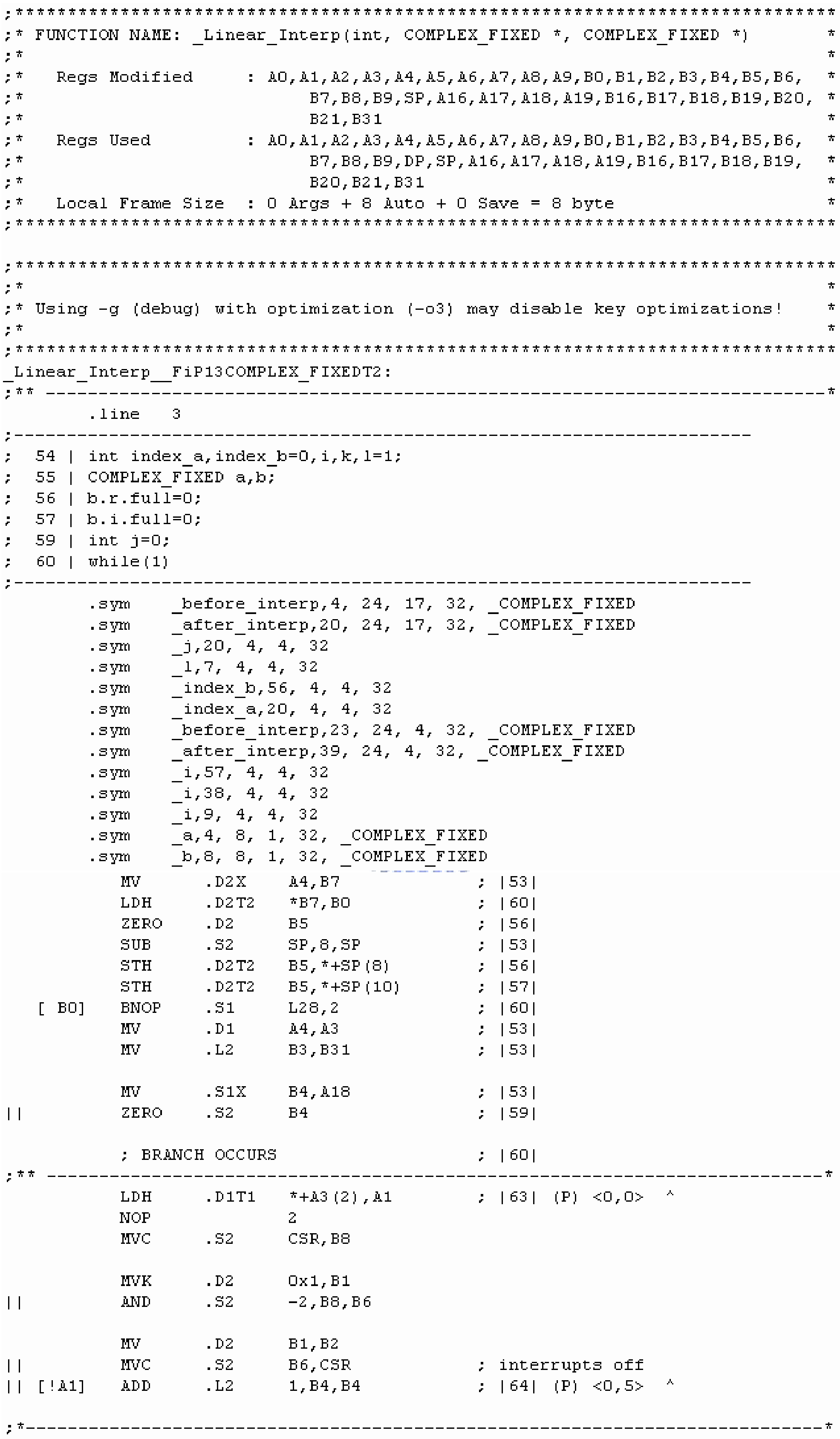
```
;******************************************************************************
;* FUNCTION NAME: _encoding_QAM(int, int *, COMPLEX_FIXED *)                  *
;*                                                                            *
;*   Regs Modified     : A0,A1,A2,A3,A4,A5,A6,B4,B5,B6,B7,B8                  *
;*   Regs Used         : A0,A1,A2,A3,A4,A5,A6,B3,B4,B5,B6,B7,B8,DP,SP         *
;*   Local Frame Size  : 0 Args + 0 Auto + 0 Save = 0 byte                    *
;******************************************************************************

;******************************************************************************
;*                                                                            *
;* Using -g (debug) with optimization (-o3) may disable key optimizations!    *
;*                                                                            *
;******************************************************************************
_encoding_QAM__FiPiP13COMPLEX_FIXED:
;** -------------------------------------------------------------------------*
        .line   2
;----------------------------------------------------------------------
;  14 | int i,j;
;  16 | j=0;
;----------------------------------------------------------------------
        .sym    _before_coding,4, 20, 17, 32
        .sym    _after_coding,20, 24, 17, 32, _COMPLEX_FIXED
        .sym    _after_coding,3, 24, 4, 32, _COMPLEX_FIXED
        .sym    _before_coding,3, 20, 4, 32

           SUB     .D1     A4,8,A5
||         MVC     .S2     CSR,B7
||         MVK     .S1     0xc0,A3          ; |17|

           SUB     .S1     A3,3,A0
||         AND     .D2     -2,B7,B8
||         LDDW    .D1T1   *++A5,A3:A2      ; (P) <0,0>

           MVC     .S2     B8,CSR           ; interrupts off
;*---------------------------------------------------------------------------*
```

Figure 5.17: Assembly code of function *Modulation (QPSK)*.



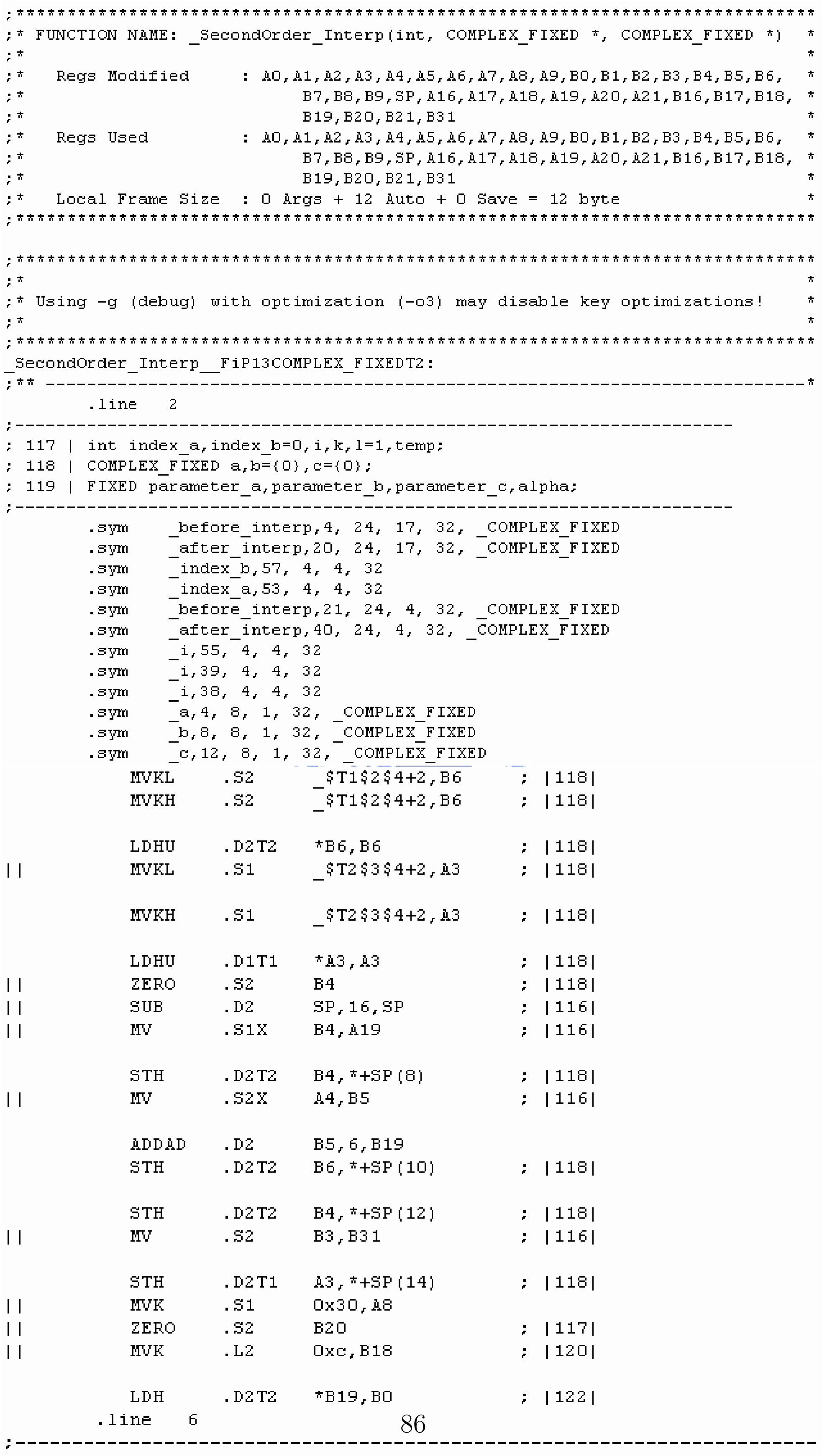
```
;******************************************************************************
;* FUNCTION NAME: _COMPLEX_MUL(int, int, COMPLEX_FIXED *, COMPLEX_FIXED *, COMPLEX_FIXED *)*
;*                                                                            *
;*   Regs Modified     : A0,A1,A3,A4,A5,A6,A7,A8,A9,B4,B5,B6,B7,B8,B9,A16,A17,*
;*                          A18,A19,A20,A21,B16,B17,B18,B19,B20,B21,B22       *
;*   Regs Used         : A0,A1,A3,A4,A5,A6,A7,A8,A9,B3,B4,B5,B6,B7,B8,B9,DP,  *
;*                          SP,A16,A17,A18,A19,A20,A21,B16,B17,B18,B19,B20,  *
;*                          B21,B22                                           *
;*   Local Frame Size  : 0 Args + 0 Auto + 0 Save = 0 byte                    *
;******************************************************************************

;******************************************************************************
;*                                                                            *
;* Using -g (debug) with optimization (-o3) may disable key optimizations!    *
;*                                                                            *
;******************************************************************************
_COMPLEX_MUL__FiT1P13COMPLEX_FIXEDN23:
;** --------------------------------------------------------------------------*
        .line   2
;----------------------------------------------------------------------
;  25 | int i;
;  28 | for(i=0;i<size;i++)
;----------------------------------------------------------------------
        .sym    _a,4, 24, 17, 32, _COMPLEX_FIXED
        .sym    _b,20, 24, 17, 32, _COMPLEX_FIXED
        .sym    _c,6, 24, 17, 32, _COMPLEX_FIXED
        .sym    _c,3, 24, 4, 32, _COMPLEX_FIXED
        .sym    _b,3, 24, 4, 32, _COMPLEX_FIXED
        .sym    _a,3, 24, 4, 32, _COMPLEX_FIXED
           MVC      .S2     CSR,B22

           MV       .D1     A6,A3             ; |24|
||         MV       .S1X    B4,A5             ; |24|
||         AND      .D2     -2,B22,B5

           MVK      .L1     0x1,A1            ; init prolog collapse predicate
||         MVK      .S1     0x32,A0           ; |28|
||         MV       .D2X    A4,B4
||         SUB      .D1     A5,16,A9          ; undo prolog elim. side-effects
||         MVC      .S2     B5,CSR            ; interrupts off
```

Figure 5.18: Assembly code of function *Complex_Mul*.



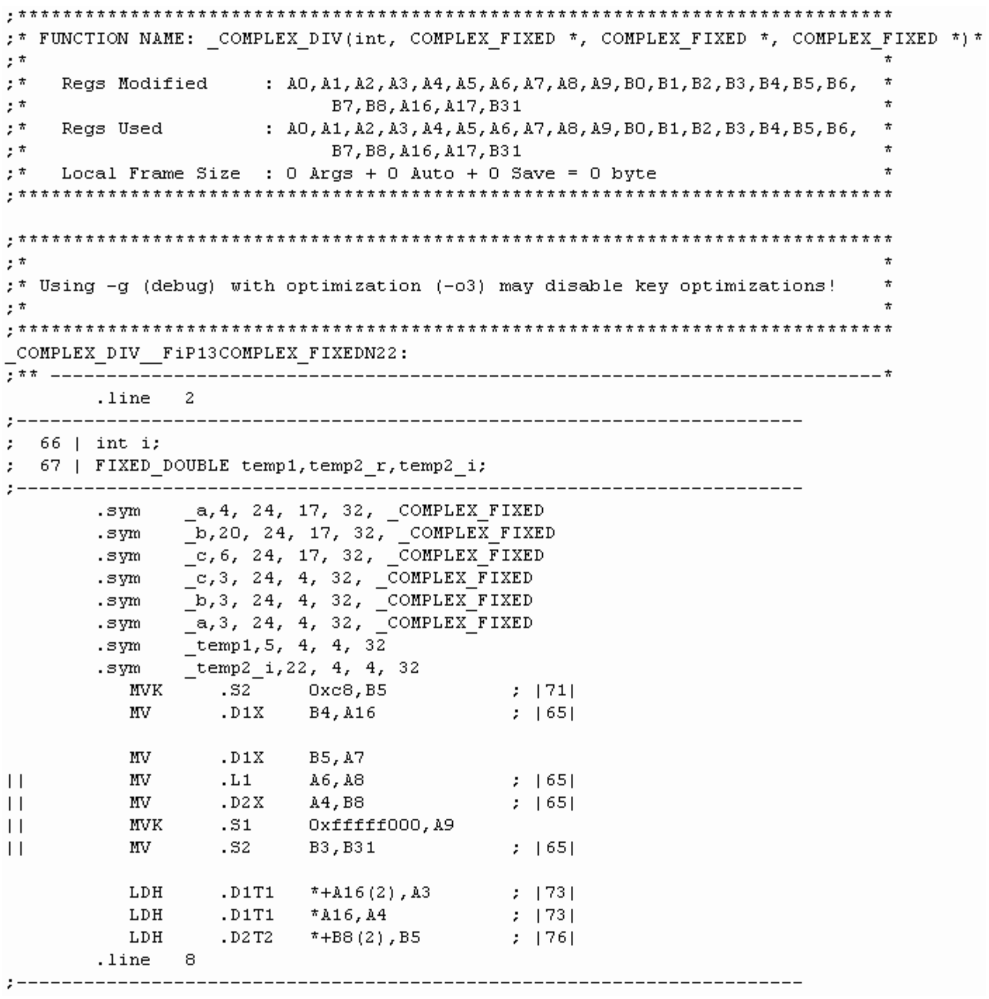
```
;******************************************************************************
;* FUNCTION NAME: _Linear_Interp(int, COMPLEX_FIXED *, COMPLEX_FIXED *)       *
;*                                                                            *
;*   Regs Modified     : A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,B0,B1,B2,B3,B4,B5,B6,  *
;*                           B7,B8,B9,SP,A16,A17,A18,A19,B16,B17,B18,B19,B20, *
;*                           B21,B31                                          *
;*   Regs Used         : A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,B0,B1,B2,B3,B4,B5,B6,  *
;*                           B7,B8,B9,DP,SP,A16,A17,A18,A19,B16,B17,B18,B19,  *
;*                           B20,B21,B31                                      *
;*   Local Frame Size  : 0 Args + 8 Auto + 0 Save = 8 byte                    *
;******************************************************************************

;******************************************************************************
;*                                                                            *
;* Using -g (debug) with optimization (-o3) may disable key optimizations!    *
;*                                                                            *
;******************************************************************************
_Linear_Interp__FiP13COMPLEX_FIXEDT2:
;** --------------------------------------------------------------------------*
        .line   3
;----------------------------------------------------------------------
;  54 | int index_a,index_b=0,i,k,l=1;
;  55 | COMPLEX_FIXED a,b;
;  56 | b.r.full=0;
;  57 | b.i.full=0;
;  59 | int j=0;
;  60 | while(1)
;----------------------------------------------------------------------
        .sym    _before_interp,4, 24, 17, 32, _COMPLEX_FIXED
        .sym    _after_interp,20, 24, 17, 32, _COMPLEX_FIXED
        .sym    _j,20, 4, 4, 32
        .sym    _l,7, 4, 4, 32
        .sym    _index_b,56, 4, 4, 32
        .sym    _index_a,20, 4, 4, 32
        .sym    _before_interp,23, 24, 4, 32, _COMPLEX_FIXED
        .sym    _after_interp,39, 24, 4, 32, _COMPLEX_FIXED
        .sym    _i,57, 4, 4, 32
        .sym    _i,38, 4, 4, 32
        .sym    _i,9, 4, 4, 32
        .sym    _a,4, 8, 1, 32, _COMPLEX_FIXED
        .sym    _b,8, 8, 1, 32, _COMPLEX_FIXED
           MV      .D2X    A4,B7             ; |53|
           LDH     .D2T2   *B7,B0            ; |60|
           ZERO    .D2     B5                ; |56|
           SUB     .S2     SP,8,SP           ; |53|
           STH     .D2T2   B5,*+SP(8)        ; |56|
           STH     .D2T2   B5,*+SP(10)       ; |57|
   [ B0]   BNOP    .S1     L28,2             ; |60|
           MV      .D1     A4,A3             ; |53|
           MV      .L2     B3,B31            ; |53|

           MV      .S1X    B4,A18            ; |53|
||         ZERO    .S2     B4                ; |59|

           ; BRANCH OCCURS                   ; |60|
;** --------------------------------------------------------------------------*
           LDH     .D1T1   *+A3(2),A1        ; |63| (P) <0,0>  ^
           NOP             2
           MVC     .S2     CSR,B8

           MVK     .D2     0x1,B1
||         AND     .S2     -2,B8,B6

           MV      .D2     B1,B2
||         MVC     .S2     B6,CSR            ; interrupts off
|| [!A1]   ADD     .L2     1,B4,B4           ; |64| (P) <0,5>  ^

;*--------------------------------------------------------------------------*
```

Figure 5.19: Assembly code of function *Linear_Interp*.



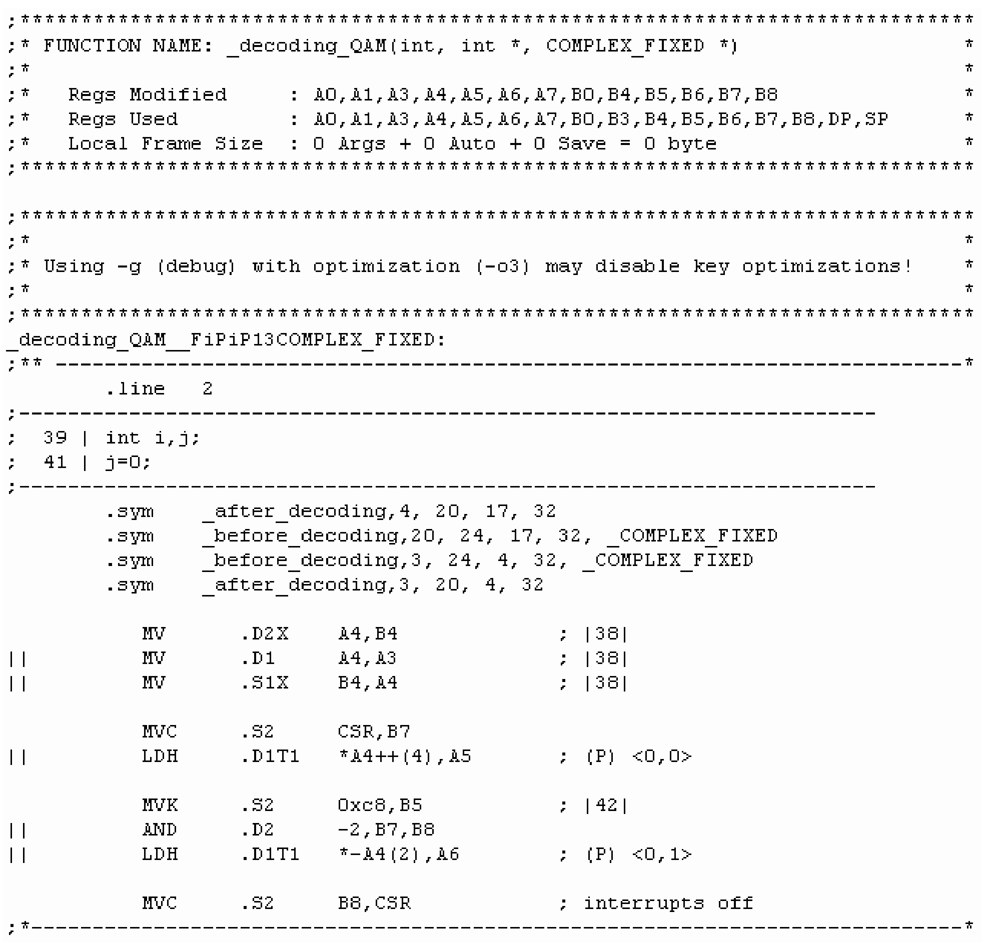
```
;******************************************************************************
;* FUNCTION NAME: _SecondOrder_Interp(int, COMPLEX_FIXED *, COMPLEX_FIXED *)   *
;*                                                                            *
;*   Regs Modified     : A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,B0,B1,B2,B3,B4,B5,B6,   *
;*                           B7,B8,B9,SP,A16,A17,A18,A19,A20,A21,B16,B17,B18, *
;*                           B19,B20,B21,B31                                  *
;*   Regs Used         : A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,B0,B1,B2,B3,B4,B5,B6,   *
;*                           B7,B8,B9,SP,A16,A17,A18,A19,A20,A21,B16,B17,B18, *
;*                           B19,B20,B21,B31                                  *
;*   Local Frame Size  : 0 Args + 12 Auto + 0 Save = 12 byte                  *
;******************************************************************************

;******************************************************************************
;*                                                                            *
;* Using -g (debug) with optimization (-o3) may disable key optimizations!    *
;*                                                                            *
;******************************************************************************
_SecondOrder_Interp__FiP13COMPLEX_FIXEDT2:
;** --------------------------------------------------------------------------*
        .line   2
;----------------------------------------------------------------------
; 117 | int index_a,index_b=0,i,k,l=1,temp;
; 118 | COMPLEX_FIXED a,b={0},c={0};
; 119 | FIXED parameter_a,parameter_b,parameter_c,alpha;
;----------------------------------------------------------------------
        .sym    _before_interp,4, 24, 17, 32, _COMPLEX_FIXED
        .sym    _after_interp,20, 24, 17, 32, _COMPLEX_FIXED
        .sym    _index_b,57, 4, 4, 32
        .sym    _index_a,53, 4, 4, 32
        .sym    _before_interp,21, 24, 4, 32, _COMPLEX_FIXED
        .sym    _after_interp,40, 24, 4, 32, _COMPLEX_FIXED
        .sym    _i,55, 4, 4, 32
        .sym    _i,39, 4, 4, 32
        .sym    _i,38, 4, 4, 32
        .sym    _a,4, 8, 1, 32, _COMPLEX_FIXED
        .sym    _b,8, 8, 1, 32, _COMPLEX_FIXED
        .sym    _c,12, 8, 1, 32, _COMPLEX_FIXED
           MVKL    .S2     _$T1$2$4+2,B6     ; |118|
           MVKH    .S2     _$T1$2$4+2,B6     ; |118|

           LDHU    .D2T2   *B6,B6            ; |118|
||         MVKL    .S1     _$T2$3$4+2,A3     ; |118|

           MVKH    .S1     _$T2$3$4+2,A3     ; |118|

           LDHU    .D1T1   *A3,A3            ; |118|
||         ZERO    .S2     B4                ; |118|
||         SUB     .D2     SP,16,SP          ; |116|
||         MV      .S1X    B4,A19            ; |116|

           STH     .D2T2   B4,*+SP(8)        ; |118|
||         MV      .S2X    A4,B5             ; |116|

           ADDAD   .D2     B5,6,B19
           STH     .D2T2   B6,*+SP(10)       ; |118|

           STH     .D2T2   B4,*+SP(12)       ; |118|
||         MV      .S2     B3,B31            ; |116|

           STH     .D2T1   A3,*+SP(14)       ; |118|
||         MVK     .S1     0x30,A8
||         ZERO    .S2     B20               ; |117|
||         MVK     .L2     0xc,B18           ; |120|

           LDH     .D2T2   *B19,B0           ; |122|
         .line   6
;--------------------------------------------------------------------------
```

Figure 5.20: Assembly code of function *SecondOrder_Interp*.



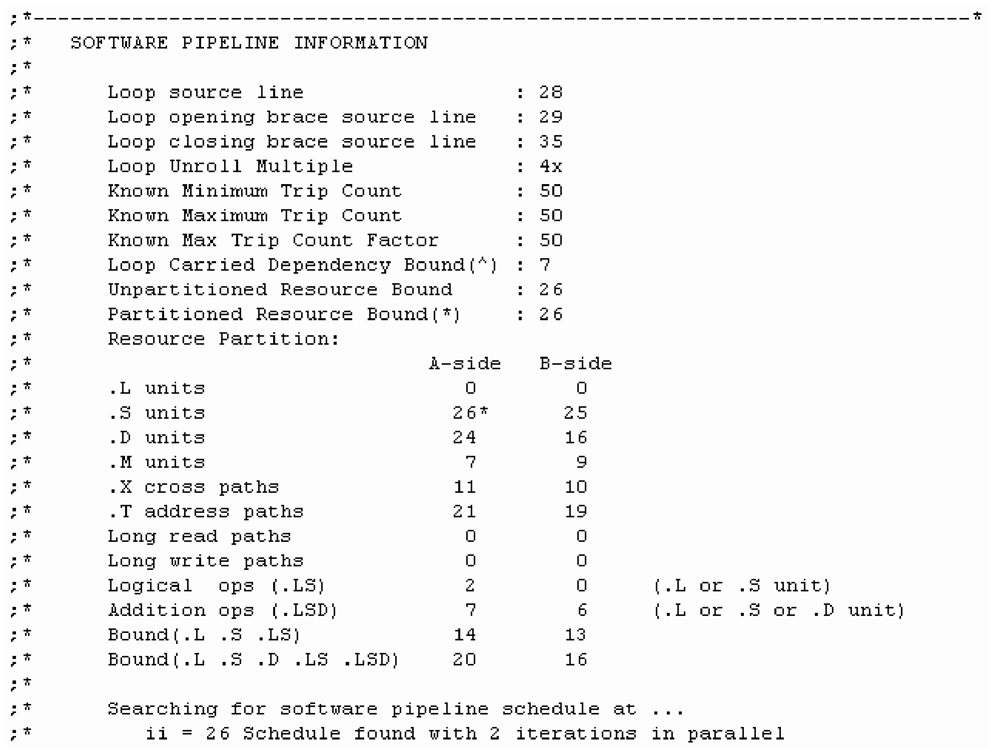
```
;******************************************************************************
;* FUNCTION NAME: _COMPLEX_DIV(int, COMPLEX_FIXED *, COMPLEX_FIXED *, COMPLEX_FIXED *)*
;*                                                                            *
;*   Regs Modified     : A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,B0,B1,B2,B3,B4,B5,B6,  *
;*                           B7,B8,A16,A17,B31                                *
;*   Regs Used         : A0,A1,A2,A3,A4,A5,A6,A7,A8,A9,B0,B1,B2,B3,B4,B5,B6,  *
;*                           B7,B8,A16,A17,B31                                *
;*   Local Frame Size  : 0 Args + 0 Auto + 0 Save = 0 byte                   *
;******************************************************************************

;******************************************************************************
;*                                                                            *
;* Using -g (debug) with optimization (-o3) may disable key optimizations!    *
;*                                                                            *
;******************************************************************************
_COMPLEX_DIV__FiP13COMPLEX_FIXEDN22:
;** -------------------------------------------------------------------------*
        .line   2
;----------------------------------------------------------------------
;  66 | int i;
;  67 | FIXED_DOUBLE temp1,temp2_r,temp2_i;
;----------------------------------------------------------------------
        .sym    _a,4, 24, 17, 32, _COMPLEX_FIXED
        .sym    _b,20, 24, 17, 32, _COMPLEX_FIXED
        .sym    _c,6, 24, 17, 32, _COMPLEX_FIXED
        .sym    _c,3, 24, 4, 32, _COMPLEX_FIXED
        .sym    _b,3, 24, 4, 32, _COMPLEX_FIXED
        .sym    _a,3, 24, 4, 32, _COMPLEX_FIXED
        .sym    _temp1,5, 4, 4, 32
        .sym    _temp2_i,22, 4, 4, 32
           MVK     .S2     0xc8,B5           ; |71|
           MV      .D1X    B4,A16            ; |65|

           MV      .D1X    B5,A7
||         MV      .L1     A6,A8             ; |65|
||         MV      .D2X    A4,B8             ; |65|
||         MVK     .S1     0xfffff000,A9
||         MV      .S2     B3,B31            ; |65|

           LDH     .D1T1   *+A16(2),A3       ; |73|
           LDH     .D1T1   *A16,A4           ; |73|
           LDH     .D2T2   *+B8(2),B5        ; |76|
        .line   8
;----------------------------------------------------------------------
```

Figure 5.21: Assembly code of function *Complex_Div*.



```
;******************************************************************************
;* FUNCTION NAME: _decoding_QAM(int, int *, COMPLEX_FIXED *)                  *
;*                                                                            *
;*   Regs Modified     : A0,A1,A3,A4,A5,A6,A7,B0,B4,B5,B6,B7,B8               *
;*   Regs Used         : A0,A1,A3,A4,A5,A6,A7,B0,B3,B4,B5,B6,B7,B8,DP,SP      *
;*   Local Frame Size  : 0 Args + 0 Auto + 0 Save = 0 byte                    *
;******************************************************************************

;******************************************************************************
;*                                                                            *
;* Using -g (debug) with optimization (-o3) may disable key optimizations!   *
;*                                                                            *
;******************************************************************************
_decoding_QAM__FiPiP13COMPLEX_FIXED:
;** -------------------------------------------------------------------------*
        .line   2
;----------------------------------------------------------------------
;  39 | int i,j;
;  41 | j=0;
;----------------------------------------------------------------------
        .sym    _after_decoding,4, 20, 17, 32
        .sym    _before_decoding,20, 24, 17, 32, _COMPLEX_FIXED
        .sym    _before_decoding,3, 24, 4, 32, _COMPLEX_FIXED
        .sym    _after_decoding,3, 20, 4, 32

           MV      .D2X    A4,B4             ; |38|
||         MV      .D1     A4,A3             ; |38|
||         MV      .S1X    B4,A4             ; |38|

           MVC     .S2     CSR,B7
||         LDH     .D1T1   *A4++(4),A5       ; (P) <0,0>

           MVK     .S2     0xc8,B5           ; |42|
||         AND     .D2     -2,B7,B8
||         LDH     .D1T1   *-A4(2),A6        ; (P) <0,1>

           MVC     .S2     B8,CSR            ; interrupts off
;*-------------------------------------------------------------------------*
```

Figure 5.22: Assembly code of function *De-modulation(QPSK)*.



```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop source line                 : 28
;*      Loop opening brace source line   : 29
;*      Loop closing brace source line   : 35
;*      Loop Unroll Multiple             : 4x
;*      Known Minimum Trip Count         : 50
;*      Known Maximum Trip Count         : 50
;*      Known Max Trip Count Factor      : 50
;*      Loop Carried Dependency Bound(^) : 7
;*      Unpartitioned Resource Bound     : 26
;*      Partitioned Resource Bound(*)    : 26
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                     0        0
;*      .S units                    26*      25
;*      .D units                    24       16
;*      .M units                     7        9
;*      .X cross paths              11       10
;*      .T address paths            21       19
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           2        0     (.L or .S unit)
;*      Addition ops (.LSD)          7        6     (.L or .S or .D unit)
;*      Bound(.L .S .LS)            14       13
;*      Bound(.L .S .D .LS .LSD)    20       16
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 26 Schedule found with 2 iterations in parallel
```

Figure 5.23: Software pipelining information of 16-bit fixed-point *Complex_Mul*.

Chapter 6

IEEE 802.16e OFDMA Downlink Channel Estimation

This chapter focuses on the performance comparison between various channel estimation techniques. First, let us review some simulation parameters and channel characteristics. Then simulation results of OFDMA downlink PUSC and OFDMA downlink FUSC will be illustrated separately.

6.1 Simulation Parameters and Channel Model

This section gives the parameters and introduce the channel model used in our simulation work.

6.1.1 OFDMA Downlink System Parameters

These system parameters used in our simulation are listed in Table 6.1. We have given primitive parameters and derived parameters in the table.

Table 6.1: OFDMA Downlink Parameters

Parameters	Values
Bandwidth	10M Hz
Central frequency	3.5G Hz
N_{used}	1681 for PUSC 1703 for FUSC
Sampling factor n	1
G	$\frac{1}{32}$
N_{FFT}	2048
Sampling frequency	11.2M Hz
Subcarrier spacing	5.47k Hz
Useful symbol time	182.86 μs
CP time	5.71 μs
OFDM symbol time	188.57 μs
Sampling time	89.29 ns

6.1.2 Simulation Channel Model

We employ the ETSI "Vehicular A" model, the same as that in the last chapter. The model has been shown in Table 5.2. However, in this chapter, the simulation is done under nonsample-spaced multipath condition.

There are six paths in this model and we express each path using Jakes fading model. Jakes fading model is a deterministic method for simulating time-correlated Rayleigh fading waveforms. The model assumes N equal-strength rays arriving at the moving receiver with uniformly distributed arrival angles, so the kth fading waveform can be modelled as [30]

$$T_k(t) = \sqrt{\frac{2}{N_0}} \sum_{n=1}^{N_0} A_k(n) \left(\cos\beta_n + j\sin\beta_n\right) \cos(\omega_M \cos\alpha_n \cdot t + \theta_n) \qquad (6.1)$$

where $k = 1, 2, \ldots, N_0$, $N_0 = N/4$, $\alpha_n = (2\pi n/N) - (\pi/N)$, $\beta_n = \pi n/N_0$, $\omega_M = 2\pi\nu/\lambda$ is the maximum Doppler frequency shift and θ_n are the independent random phases, each of which uniformly distributed in $[0, 2\pi)$. $A_k(n)$ is the kth Walsh-Hadamard codeword in n

which satisfies

$$\frac{1}{N_0}\sum_{n=1}^{N_0} A_k^*(n)A_l(n) = \begin{cases} 1, & k = l, \\ 0, & k \neq l. \end{cases}$$

The time variations of amplitude at different velocities are as shown in Fig. 6.1. The x-axis denotes the symbol sample, that is, 188.57 μs per sample and the y-axis denotes the amplitude in decibel. Meanwhile, the Doppler rate (maximum Doppler frequency shift multiplied by symbol time) in each condition is also listed.

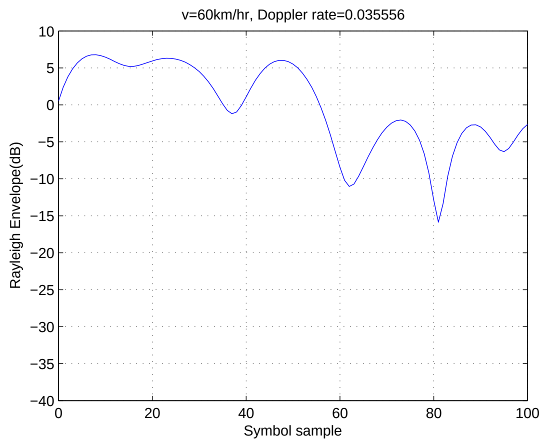
6.1.2.1 Some Effects of Nonsample-Spaced Multipath Channel

A sample-spaced channel has all delayed impulses of its channel impulse response at integer multiples of the system sampling interval. Hence, the continuous Fourier transform of the channel impulse response is a channel frequency response with non-zero values at multiples of the system's sampling rate. Due to this particular frequency response, the samples of the channel frequency response coincides exactly with the DFT of the channel impulse response.

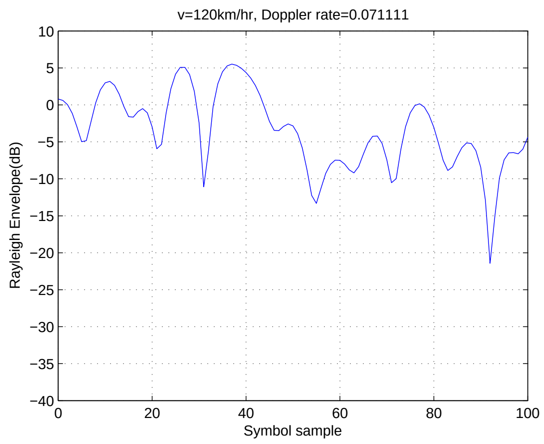
For the nonsample-spaced channel, like the one listed in Table 5.2, the channel is actually resampled in the receiver sampling process. As a consequence, there is no exact correspondence between the channel frequency response and the DFT of the sampled channel impulse response. In fact, the resampling process results in an extension of the equivalent sample-spaced channel response (as shown in Fig 6.2). This leads to the need for a longer guard interval to ease synchronization. It can also cause performance loss of the channel estimator, if it is based on a limited channel length assumption.

6.2 Simulation Flow

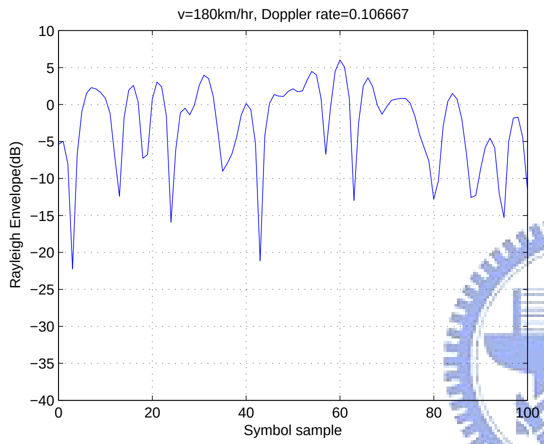
Figure 5.1 illustrates the simulation flow of the OFMDA system and Fig. 5.2 accounts for the channel estimation steps. These steps are already described in the last chapter.



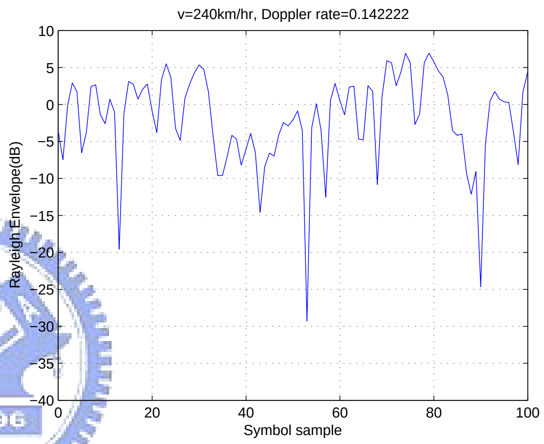
(a)



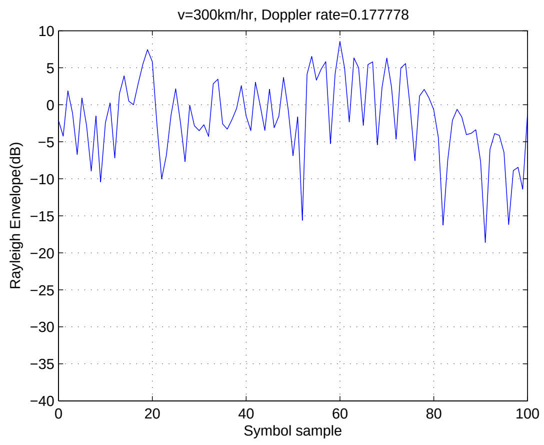
(b)



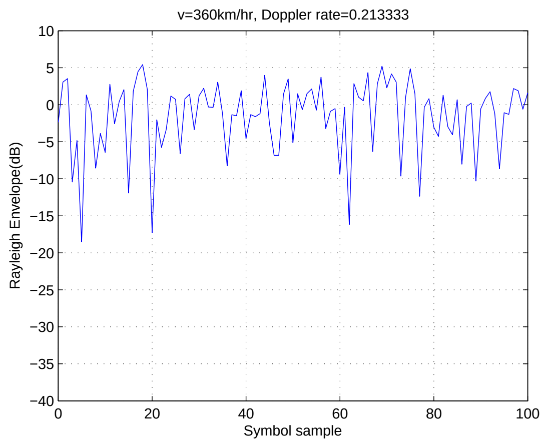
(c)



(d)



(e)



(f)

Figure 6.1: Amplitude variations for different velocities. (a) v = 60 km/h. (b) v = 120 km/h. (c) v = 180 km/h. (d) v = 240 km/h. (e) v = 300 km/h. (f) v = 360 km/h.

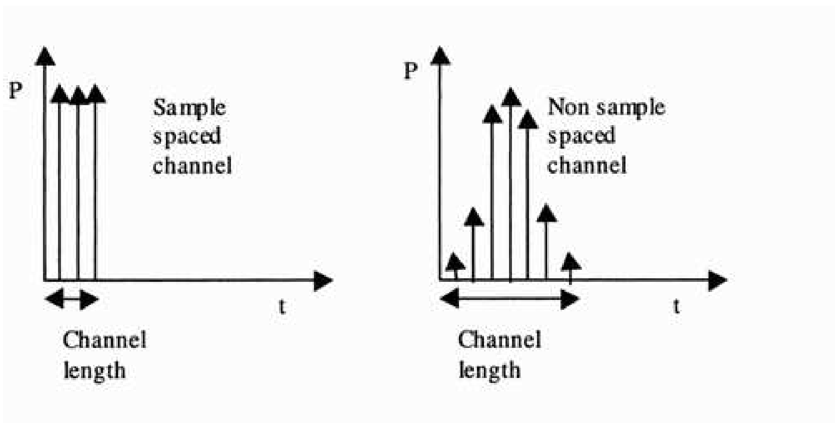


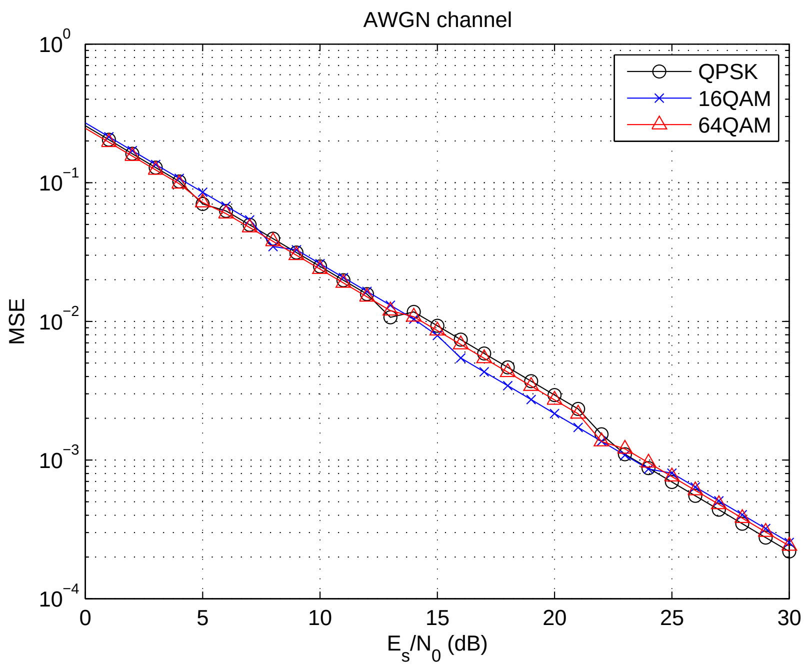
Figure 6.2: Resampling a nonsample-spaced channel extends the channel (from [3]).

Before considering multipath channels, we must validate our simulation on an AWGN channel by comparing theoretical SER curves and SER curves resulting from simulations. This comparison is illustrated here for uncoded QPSK, 16QAM and 64QAM modulations. Fig. 6.3(a) shows the MSE results from simulation. Fig 6.3(b) shows their corresponding SER values and compare them with the theory given in (5.1). As seen, the simulation results match well with the theory, validating our simulation model. In Fig. 6.3, we use simple one-dimensional linear interpolation for channel estimation.

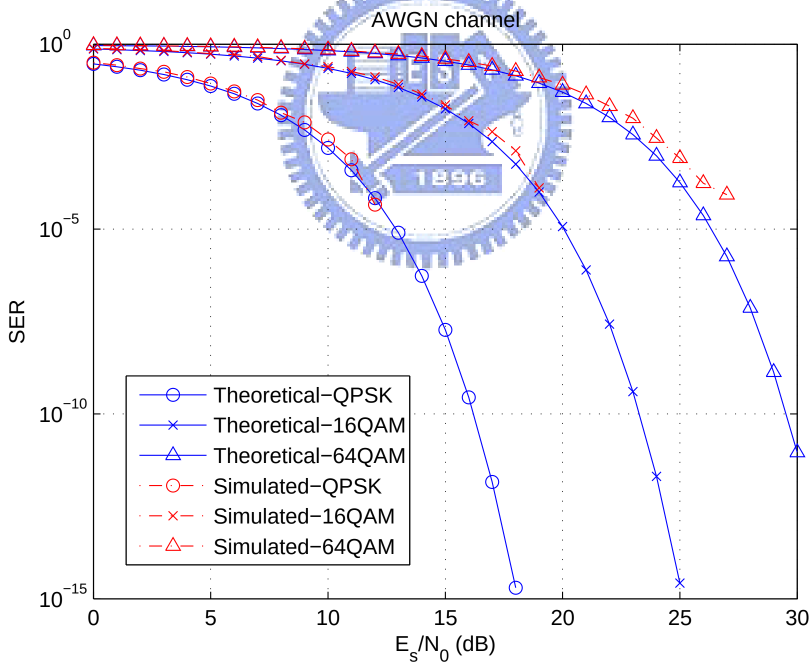
6.3 OFDMA DL PUSC Channel Estimation Simulation

Here, we assume that only one of three segments are used, say, Segment 0. And we use the subcarriers belonging to clusters of Group 0, which is assigned to Segment 0 by default. More specifically, we use 288 data subcarriers and 48 pilot subcarriers.

Due to nonsample-spaced multipath channels, the path extension effect appears and is as shown in Fig. 6.4.

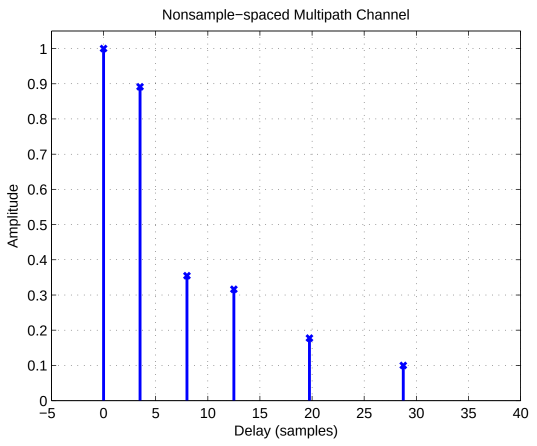


(a) MSE

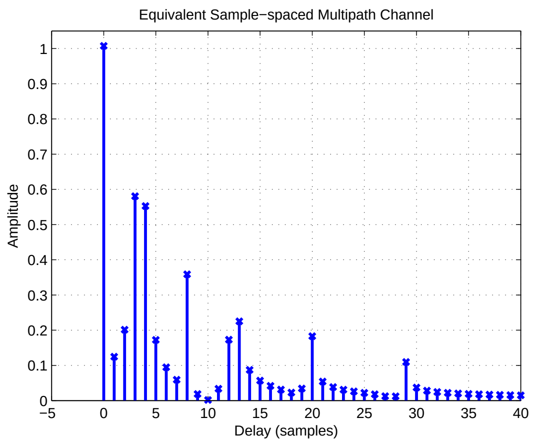


(b) SER

Figure 6.3: Simulation results (1D linear interpolation) for AWGN channel. (a) MSE. (b) SER.



(a)



(b)

Figure 6.4: The path extension effect in our model. (a) Nonsample-spaced multipath channel. (b) Equivalent sample-spaced multipath channel.

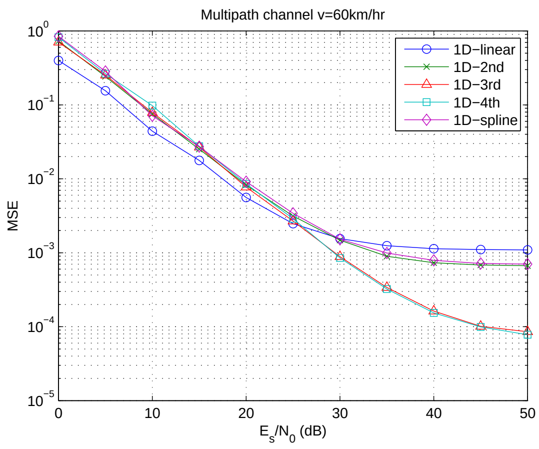
6.3.1 One-Dimensional Channel Estimators

6.3.1.1 Polynomial Interpolation Results

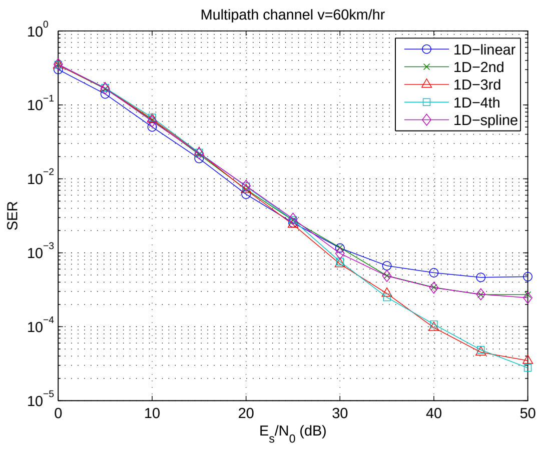
Fig. 6.5 shows the MSE and SER in each modulation for v = 60 km/h. It indicates that third-order or fourth-order interpolation has the best performance, and followed by spline interpolation, second-order interpolation, and linear interpolation. The results of MSE are unrelated to the modulation type because the pilots are boosted-BPSK modulated in each modulation case. And the channel response is interpolated only using the pilot information.

Fig. 6.6 shows the MSE and the SER spread over used subcarriers. We can find that the best subcarrier is around 310 and the worst subcarrier is around 600. Fig. 6.7 shows performance at best and worst subcarriers.

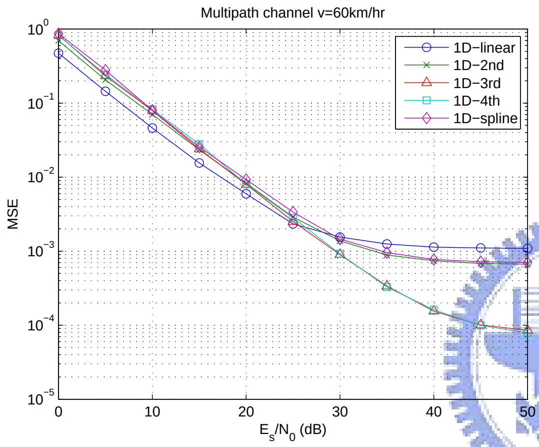
We also want to know the impact of velocity on channel estimation performance. Some results are shown in Fig. 6.8. The 1D interpolation-based channel estimation seems unrelated to channel time variations because the time variations over OFDMA symbols is not taken into account in this method.



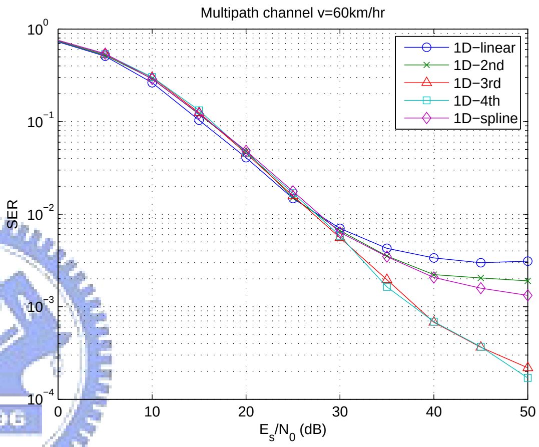
(a)



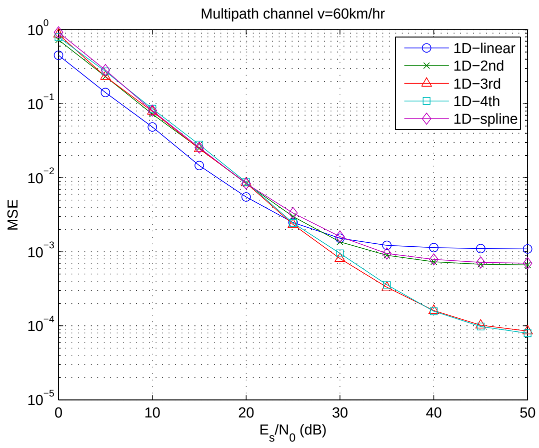
(b)



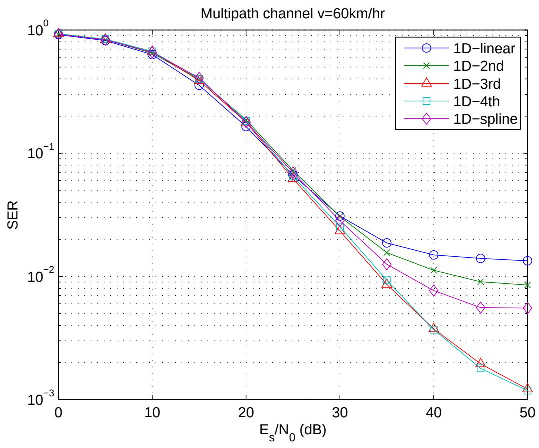
(c)



(d)



(e)



(f)

Figure 6.5: OFDMA DL PUSC MSE and SER for v = 60 km/h in each modulation scheme. (a) MSE for QPSK. (b) SER for QPSK. (c) MSE for 16QAM. (d) SER for 16QAM. (e) MSE for 64QAM. (f) SER for 64QAM.

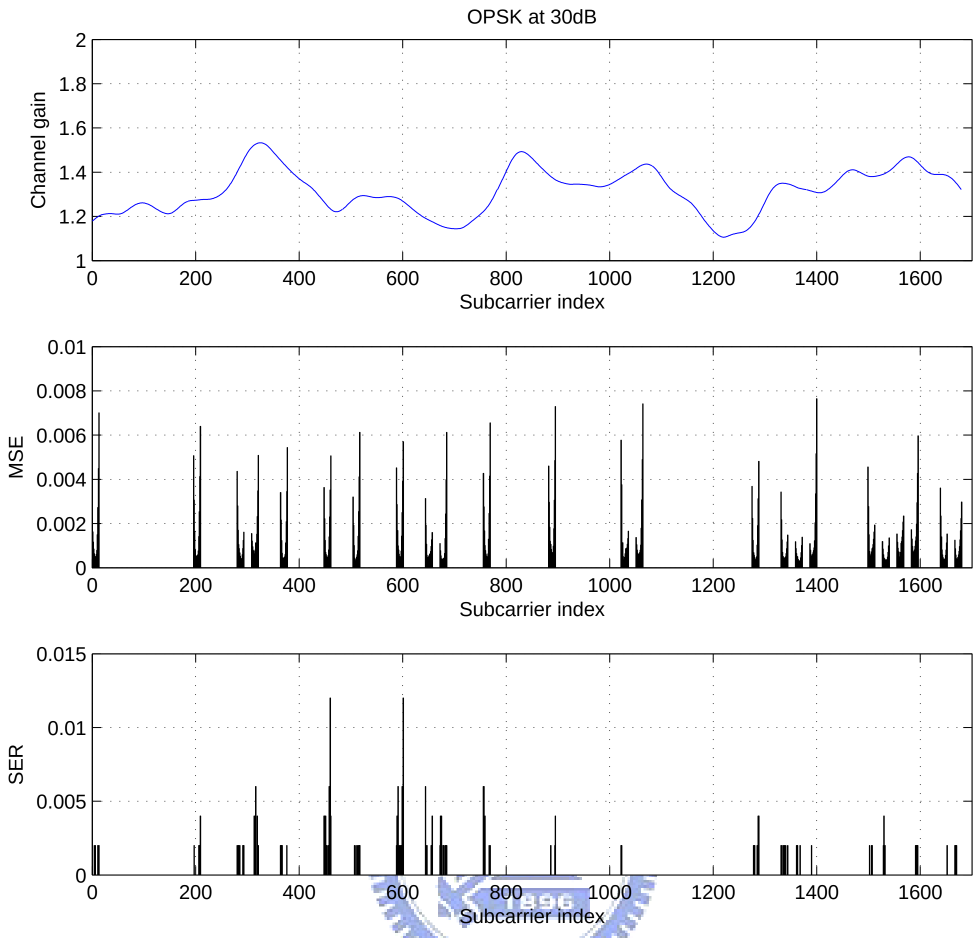


Figure 6.6: MSE and SER spread over used subcarriers in QPSK at 30 dB.

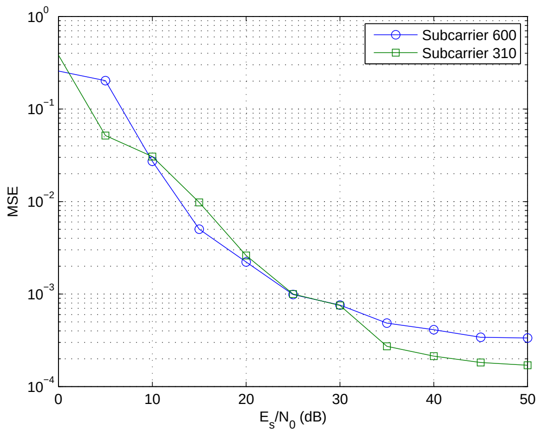
6.3.2 Two-Dimensional Channel Estimators

6.3.2.1 Time-Domain Interpolation

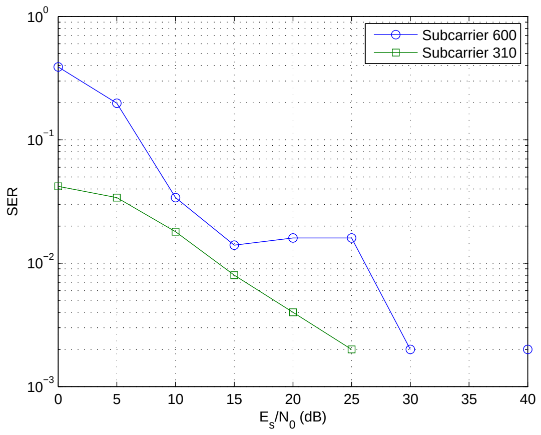
We apply interpolation along both the frequency and the time axes. For simplicity, we just combine the frequency-interpolated $H_{freq}(k)$ with the time-interpolated $H_{time}(k)$ by

$$H_{new}(k) = \frac{H_{freq}(k) + H_{time}(k)}{2}. \tag{6.2}$$

The interpolation on $H_{time}(k)$ is done on the real part and the imaginary part separately.

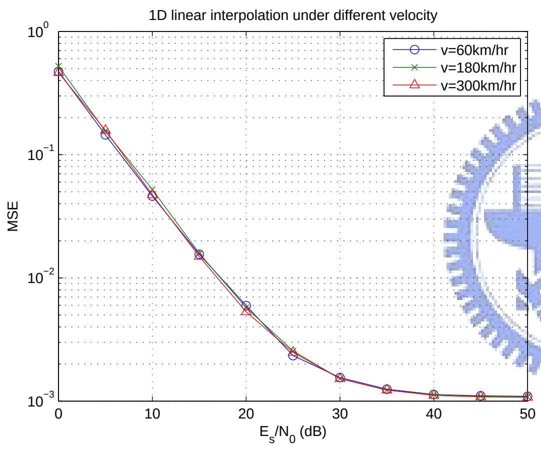


(a)

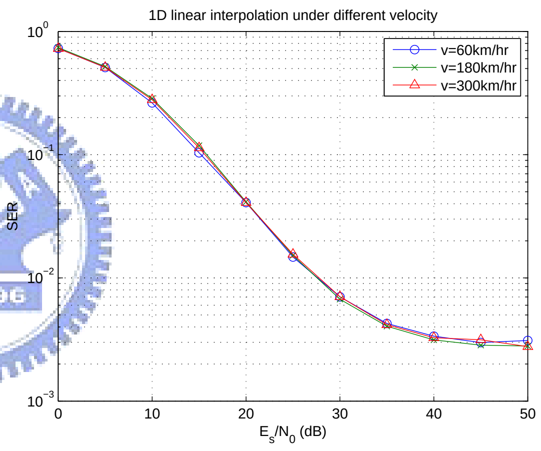


(b)

Figure 6.7: Performance at worst and best subcarriers. (a) MSE. (b) SER.



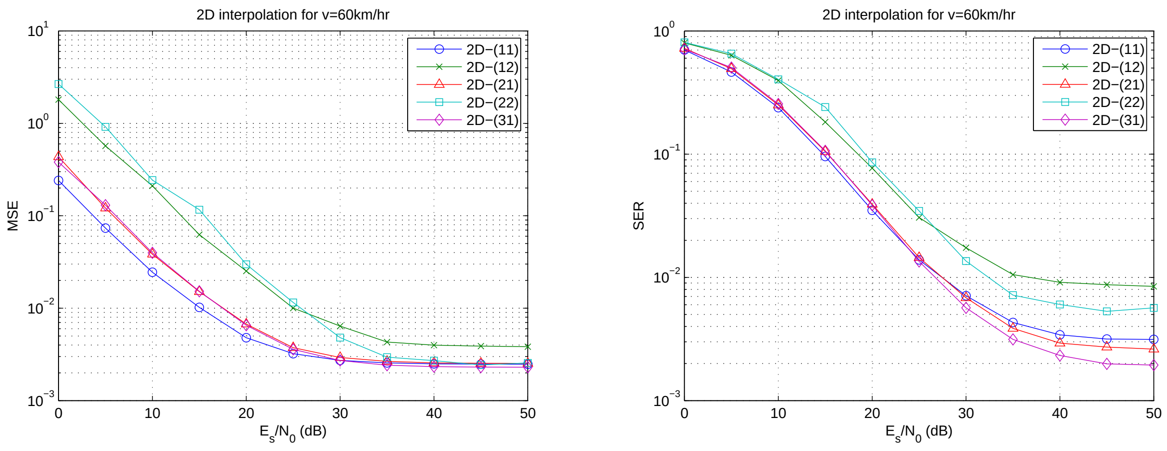
(a)



(b)

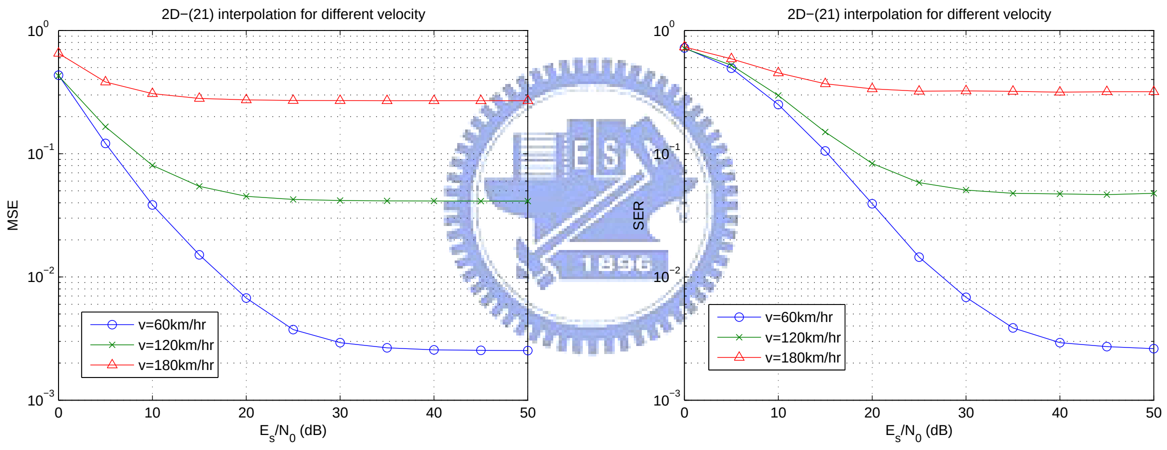
Figure 6.8: The performance of 1D linear interpolation for different velocity with 16QAM. (a) MSE. (b) SER.

The results are shown in Fig. 6.9 for different kinds of interpolation order. Similarly, the results under different velocities are as shown in Fig. 6.10. Here the first number in the parentheses denotes the interpolation order along the frequency axis and the second denotes that along the time axis. For example, “2D-(21)” means that second-order interpolation is done along the frequency axis and linear interpolation is done along the time axis. From



(a) (b)

Figure 6.9: Performance of 2D interpolation for v = 60 km/h with 16QAM. (a) MSE. (b) SER.



(a) (b)

Figure 6.10: Performance of 2D-(21) interpolation for different velocities with 16QAM. (a) MSE. (b) SER.

these figures, we may conclude that the time variations of channel characteristics are hard to predict via simple interpolation.

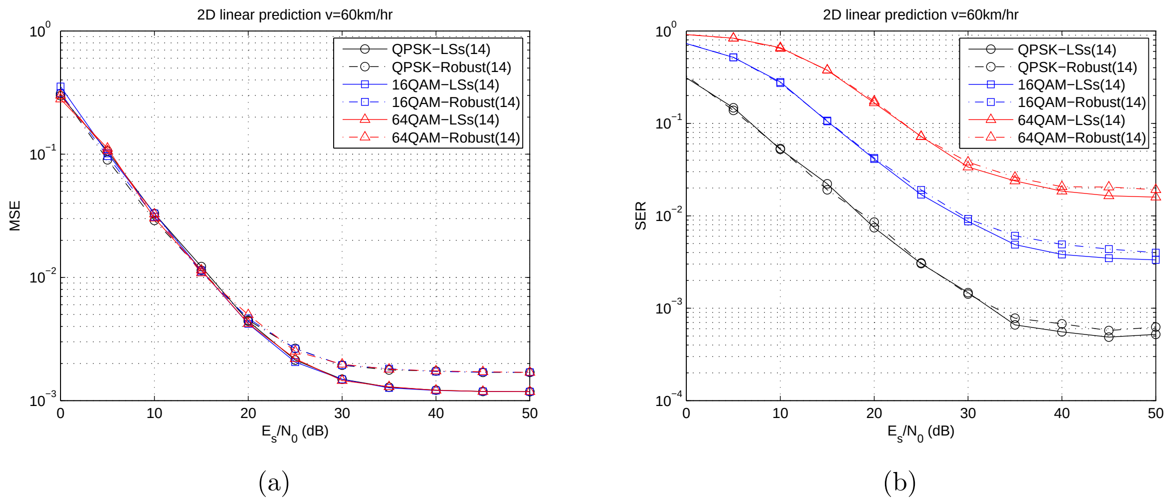


Figure 6.11: Performance of linear prediction using different merit functions. (a) MSE. (b) SER.

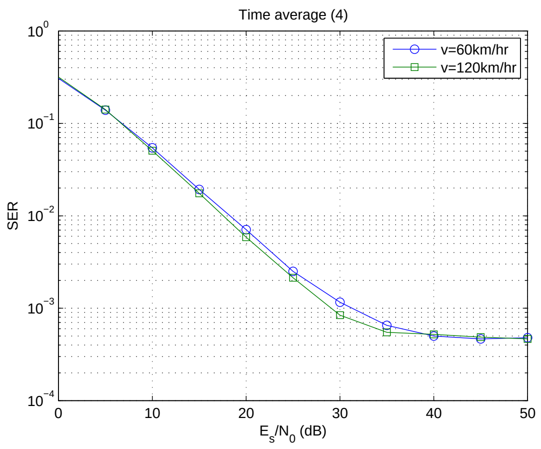
6.3.2.2 Linear Prediction

Instead of using interpolation in the time domain, we now consider linear prediction based on recent symbols. We use (3.31) and (3.36) as the merit functions. In fact, the former is the linear least squares method denoted "LSs" in Fig. 6.11 and the latter is more robust against noise, denoted "Robust" in Fig. 6.11.

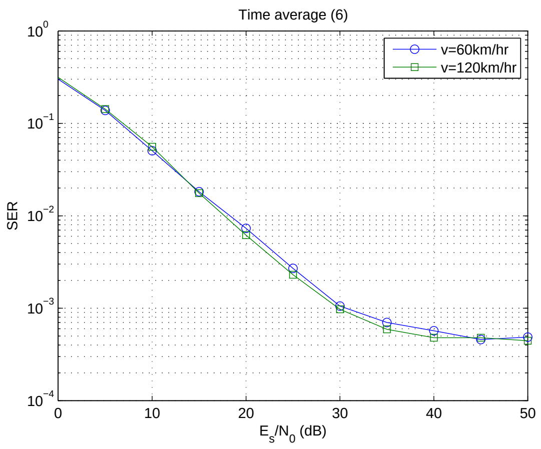
In Fig. 6.11, the first number in each parenthesis means the interpolation order along the frequency axis while the second number means how many recent symbols we use in the prediction. Intuitively, in high SNR, the noise effects are small and the least squares should have better performance. This point corresponds to our simulation results.

6.3.3 Time Averaging

As mentioned in chapter 3, within the coherent time, the channel can be regarded as nearly time-invariant. We can mitigate noise effects by averaging the estimated channel responses over several OFDMA symbols based on the maximum Doppler spread.



(a)



(b)

Figure 6.12: Performance of time-averaging. (a) SER for averaging 4 symbols (b) SER for averaging 6 symbols

Note that although the envelope of each path varies more slowly than the phase. The phase of each path is more unpredictable over time. That is, we can not just average the real part and imaginary part of $H(k)$ over several OFDMA symbols. Therefore, we amend (3.42) to

$$H_{avg}(k) = \frac{1}{4} \times \Bigg\{ H_0^{interp}(k) + ||H_{-1}^{interp}(k)||\measuredangle H_0^{interp}(k) + ||H_{-2}^{interp}(k)||\measuredangle H_0^{interp}(k) + ||H_{-3}^{interp}(k)||\measuredangle H_0^{interp}(k) \Bigg\} \tag{6.3}$$

where $H_n^{interp}(k)$ is the interpolated channel response at the previous nth symbol time. In other words, we average the amplitude at each subcarrier k but force their phases equal to that of $H_0^{interp}(k)$. Fig. 6.12(a) shows the results of averaging latest 4 symbols under 3 different velocities. And Fig. 6.12(b) shows the simulation results of averaging latest 6 symbols. Because the maximum Doppler shift is not severe in each case, so the performance is almost the same.

6.4 OFDMA DL FUSC Channel Estimation Simulation

This section shows simulation results of the OFDMA downlink FUSC system. The system parameters are as shown in Table 6.1.

6.4.1 One-Dimensional Channel Estimators

6.4.1.1 Polynomial and Rational Function Interpolation Results

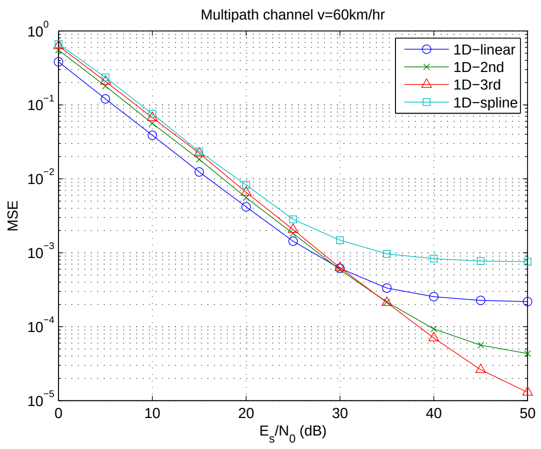
Fig. 6.13 shows the simulation results of polynomial interpolation. To verify its consistency on all used subcarriers, Fig. 6.14 shows the MSE and SER distribution situation over the 1703 used subcarriers. It shows a fact that the lower channel gain will cause worst performance in accordance with our knowledge. The worst subcarrier index is about 500 and best subcarrier is about 1100. Fig. 6.15 indicates that the simulation results does not differ much at worst and best subcarriers

As for the rational function interpolation, the results are shown in Fig. 6.16. Here, we let $\mu = 0$ and $\nu = 1$ in (3.17). It can be seen that the performance of rational function interpolation is much worse than that of polynomial interpolation. Also it can be seen that the curve of the channel gain in Fig. 6.14 is not suitable for rational interpolation.

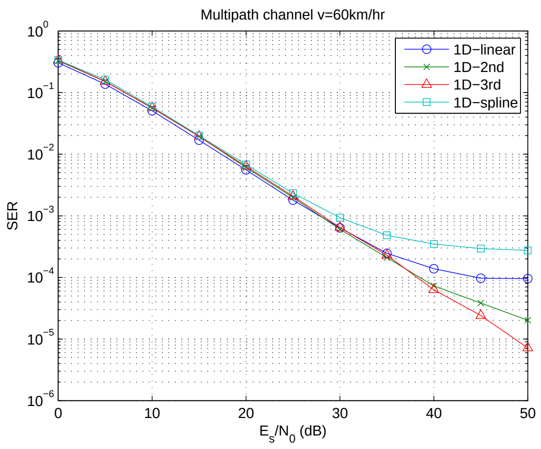
6.4.2 Adaptive Channel Estimators

In the discussion of OFDMA downlink PUSC simulation, we have shown that it is hard to model the channel time variations via interpolation or prediction. In the following simulation, we apply normalized least mean square (NLMS) algorithm to track time variations.

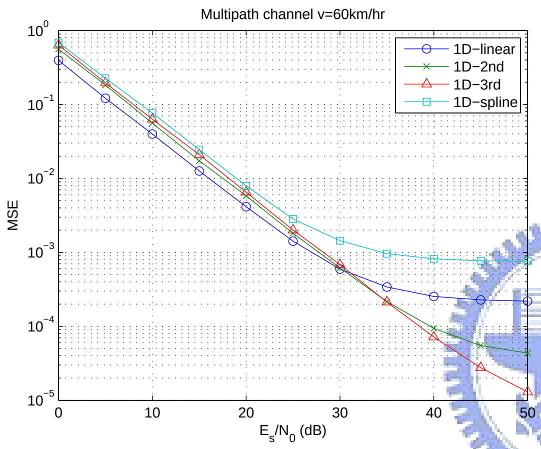
We transmit a block starting with a preamble and followed by 10 to 20 OFDMA symbols. First we interpolate the channel response $H_{preamble}(k)$, obtained from the preamble. Then



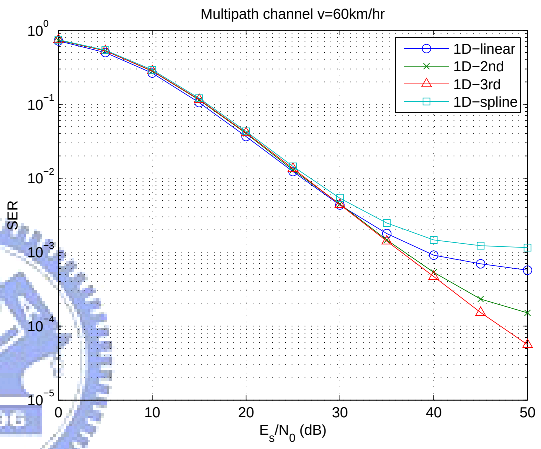
(a)



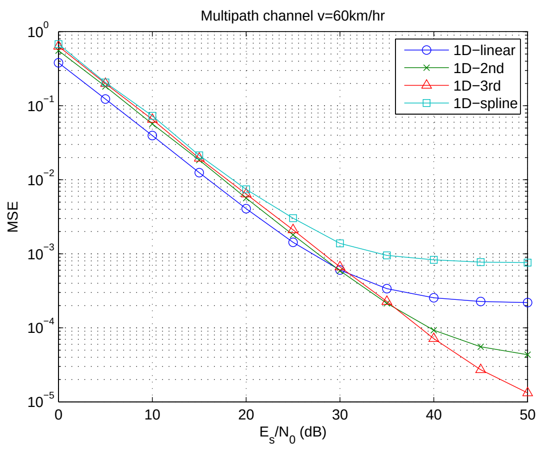
(b)



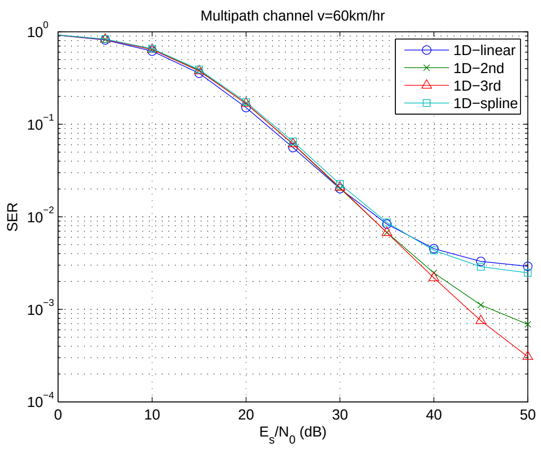
(c)



(d)



(e)



(f)

Figure 6.13: OFDMA DL FUSC MSE and SER for v = 60 km/h in each modulation scheme. (a) MSE for QPSK. (b) SER for QPSK. (c) MSE for 16QAM. (d) SER for 16QAM. (e) MSE for 64QAM. (f) SER for 64QAM.

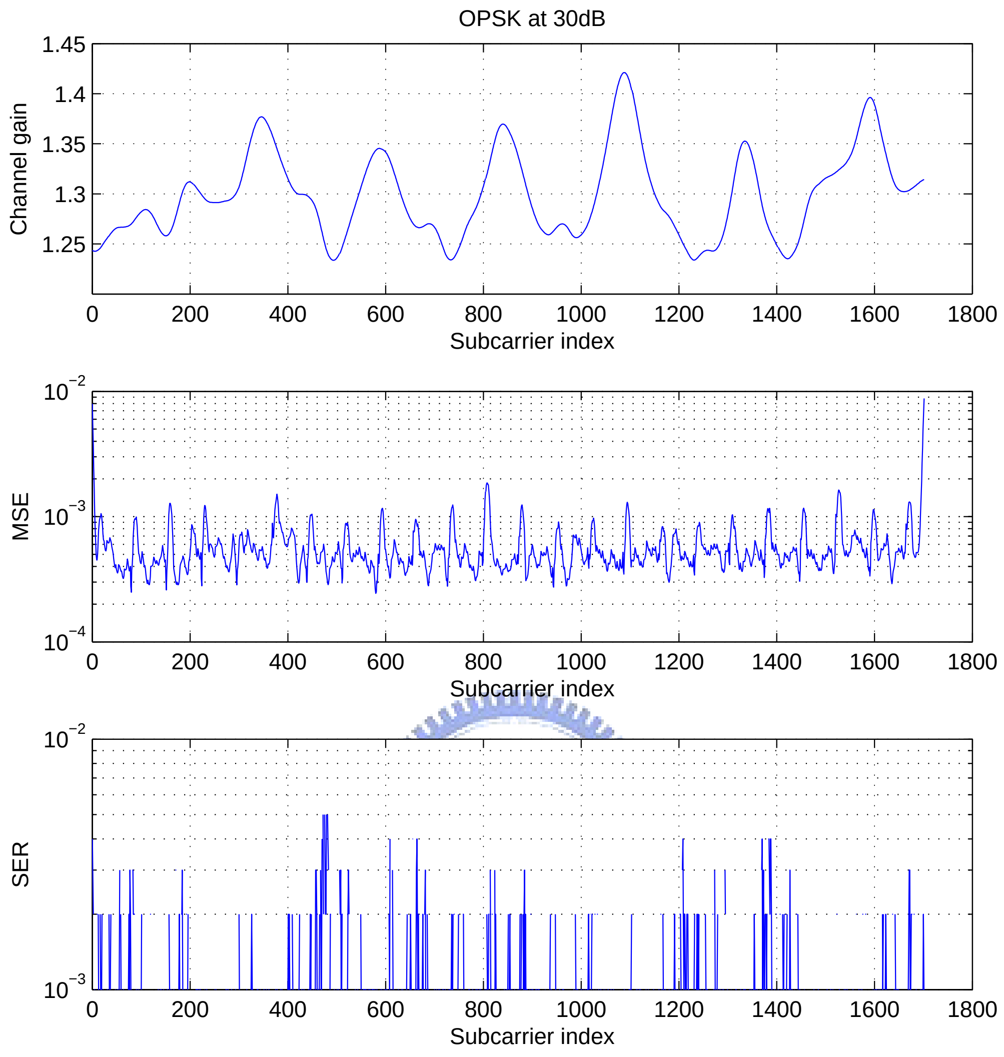


Figure 6.14: MSE and SER spread over used subcarriers in QPSK at 30 dB.

we use the interpolated $H_{preamble}(k)$ to initialize NLMS algorithm. The detailed algorithms are given in (3.46) and (3.49).

Figs. 6.17 and 6.18 show the simulation results of NLMS based on equalization and channel estimation respectively. We may conclude that the NLMS tracking ability on time variations is very limited.

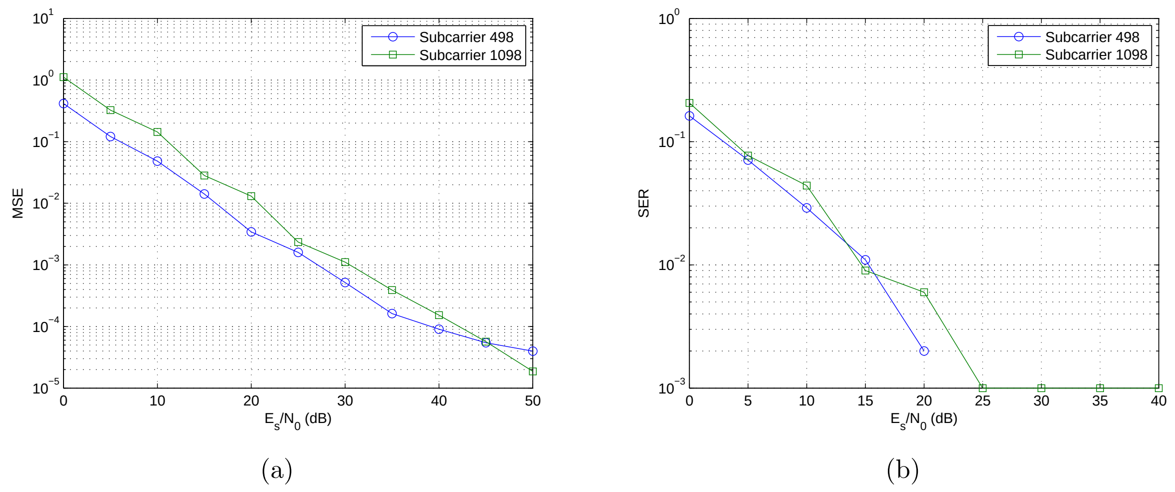


Figure 6.15: Performance at worst and best subcarriers. (a) MSE. (b) SER.

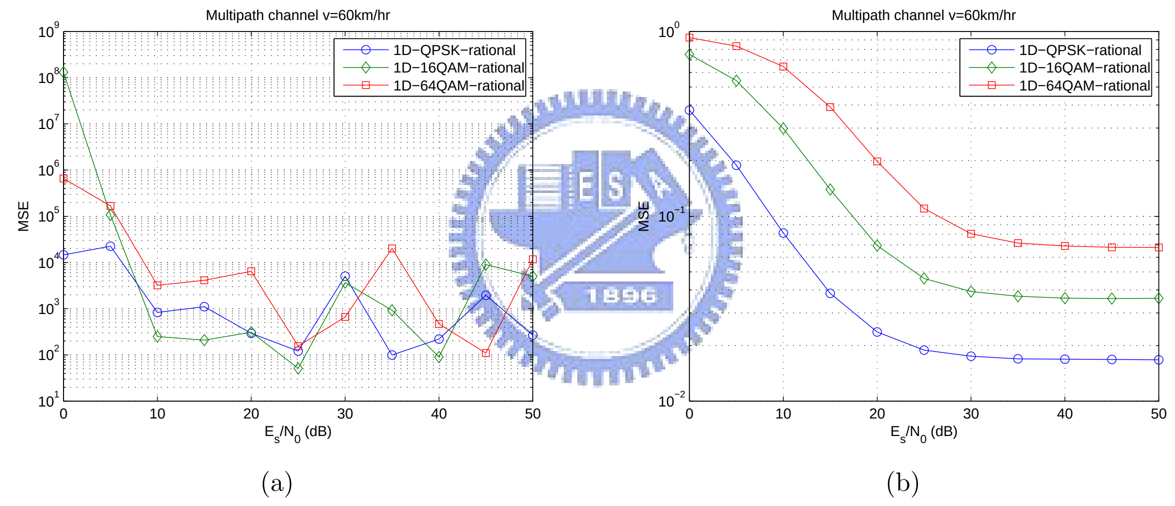
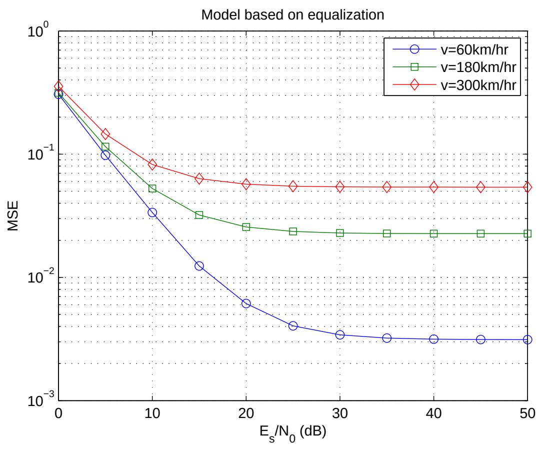


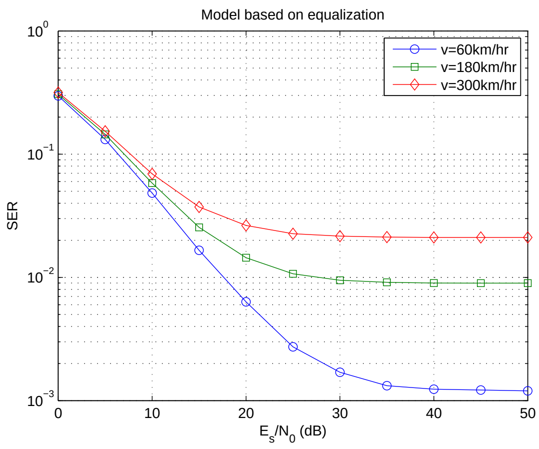
Figure 6.16: Performance of rational function interpolation. (a) MSE. (b) SER.

6.4.3 The Maximum Likelihood Channel Estimator

As described before, in MLE, the channel impulse response is viewed as a deterministic but unknown vector and no information on the channel statistics or the operating SNR is required. The MSE and SER based on MLE are as illustrated in Fig. 6.19. The solid lines in these figures denote results from 64-bit (double-precision) floating-point computation while

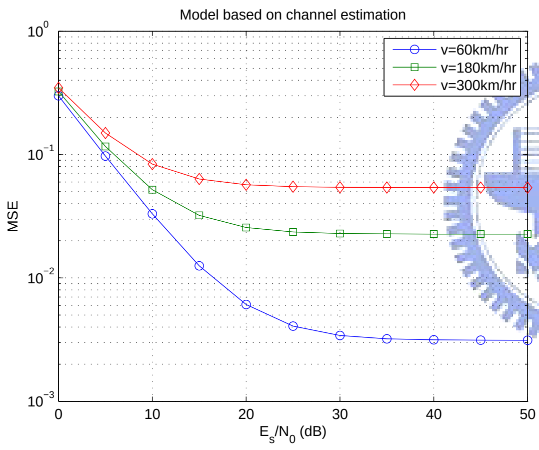


(a)

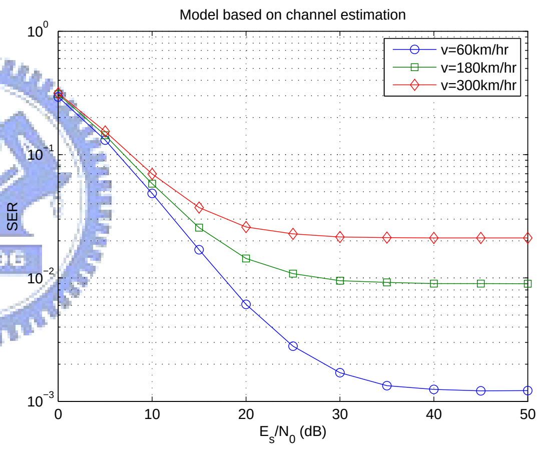


(b)

Figure 6.17: NLMS performance based on equalization. (a) MSE for QPSK. (b) SER for QPSK.



(a)



(b)

Figure 6.18: NLMS performance based on channel estimation. (a) MSE for QPSK. (b) SER for QPSK.

the dashed lines denote results from 32-bit (single-precision) floating point computation. It seems that the performance of MLE depends heavily on computational precision.

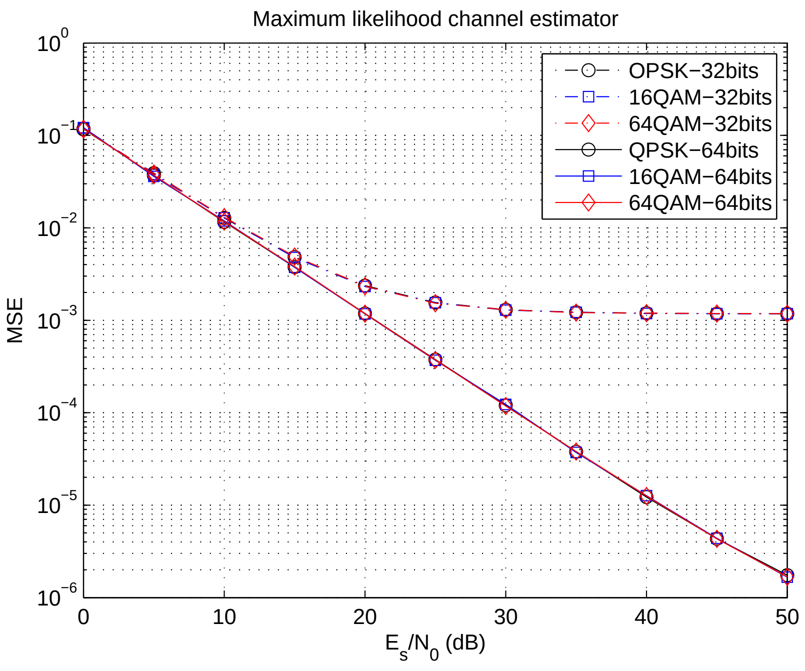
Let us review the mathematical expression of MLE in (3.27). In addition to many element-by-element multiplications, there needs a matrix inversion in this equation. Usually, matrix inversions have serious accuracy problems due to error propagation. Even with the use of some techniques of pivoting, matrix inversion is still a time-consuming and precision-dependent work. Furthermore, the matrix to be inverted is close to being singular when the cyclic prefix is lengthened or the pilot locations are not carefully selected.

Fig. 6.20 illustrates the eigenvalue spread of matrix $\mathbf{D}$ in (3.27). It shows that the eigenvalue spread increases exponentially as the CP is lengthened. In our simulation program, the available CP length is 32 or 64. The CP length greater than or equal to 128 will cause singularity problem.

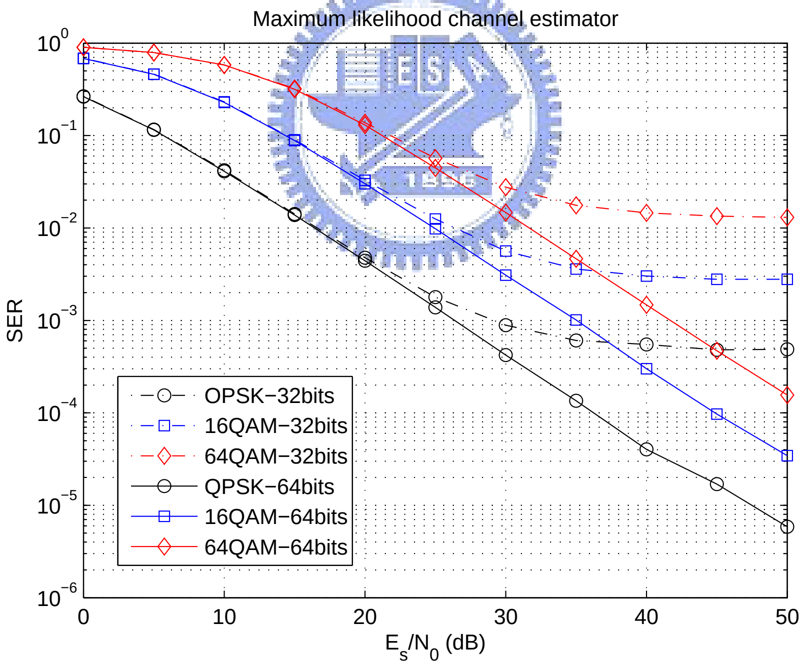
6.4.4 Joint Channel Estimation and Symbol Detection

The simulation flow of joint channel estimation and symbol detection (JCESD) has been shown in Fig. 3.5. Fig. 6.21 shows the MSE and SER of JCESD (with one time iteration) compared with MLE. It can be seen that the performance of JCESD is slightly better than that of MLE. This is because that in addition to pilots, we also use the channel response at data subcarriers. Especially at higher SNR, the data subcarriers are less susceptible to noise and thus contribute more reliable channel response estimation.

As for the MSE discrepancy between QPSK, 16QAM and 64QAM, this results from different power level of data symbols. Fig. 6.21(a) only shows results on average. In fact, QPSK data symbols have equal power, i.e., noise has equal impact on these symbols. On the contrary, the power levels of 16QAM or 64QAM data symbols differ. Hence, some data symbols with smaller power will yield less reliable channel response estimation. The simulation corresponds to our expectation. JCESD improves QPSK much, then 16QAM, and 64QAM least. A main difference between MLE and JCESD is that JCESD utilize both data and pilot subcarriers while MLE only use pilot subcarriers. Hence, JCESD has better performance than MLE.



(a)



(b)

Figure 6.19: Maximum likelihood channel estimators for different computational precision. (a) MSE. (b) SER.

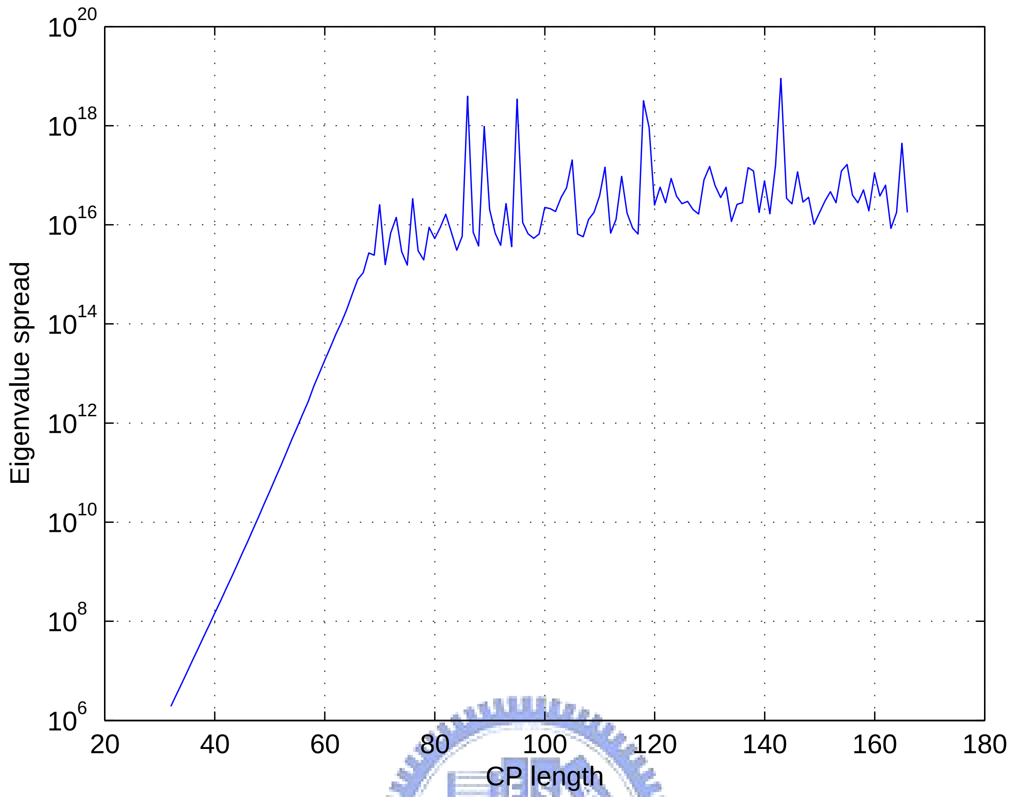
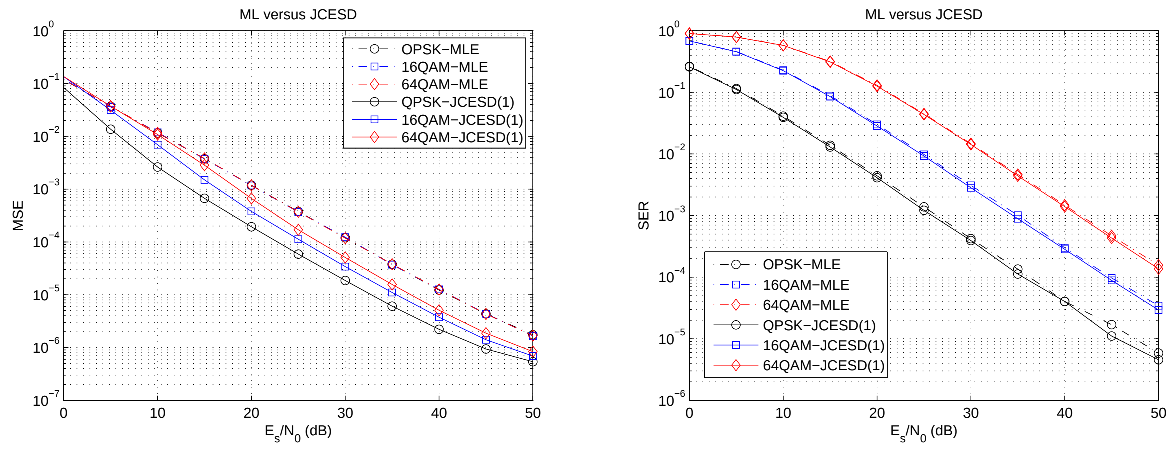


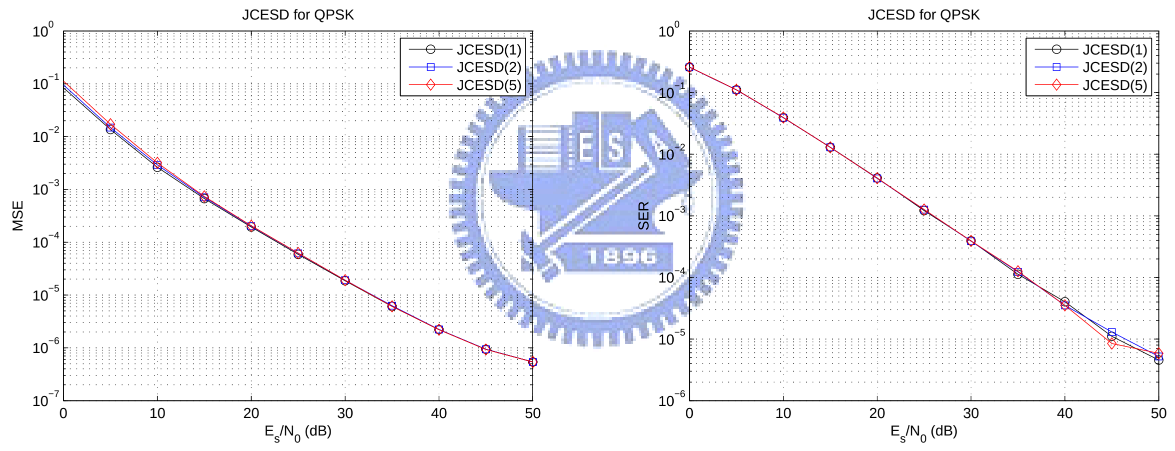
Figure 6.20: Eigenvalue spread of the matrix $\mathbf{D}$ in (3.27).

Fig. 6.22 shows results for different number of JCESD iterations. It shows that one time iteration is enough and two or more iterations do not improve further. In fact, more iterations will cause worst performance at lower SNR. This is because data symbols are easily smeared by noise at low SNR. This effect can be worse due to hard decision and hence contribute to unreliable channel estimation. Thus the performance at low SNR may be worse after JCESD.



(a) (b)

Figure 6.21: Maximum likelihood versus joint channel estimation and symbol detection. (a) MSE. (b) SER.



(a) (b)

Figure 6.22: Performance of JCESD with different number of iterations. (a) MSE. (b) SER.

6.5 Overall Performance Comparison

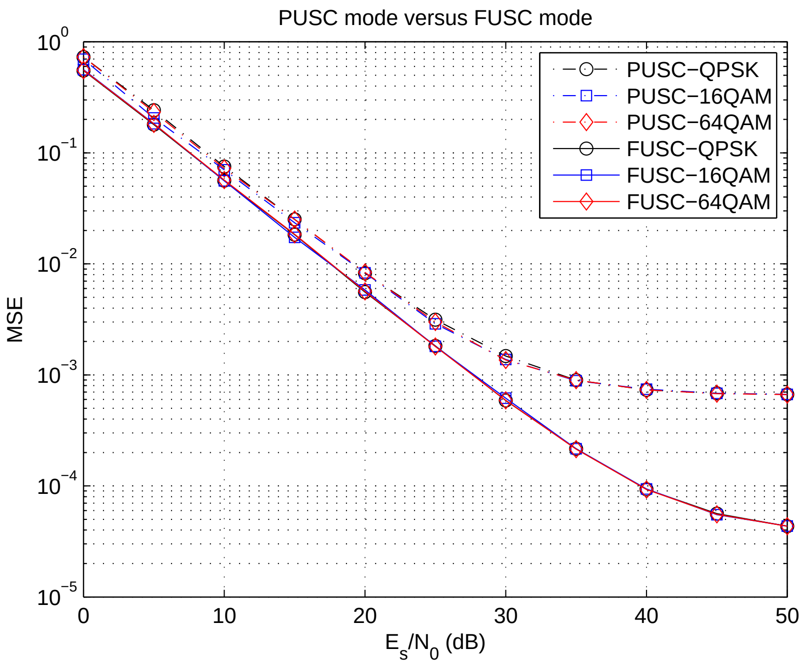
6.5.1 PUSC Mode Versus FUSC Mode

Intuitively, FUSC mode uses more pilots than PUSC mode and should have better performance. This point corresponds to our simulation results (1D second-order interpolation

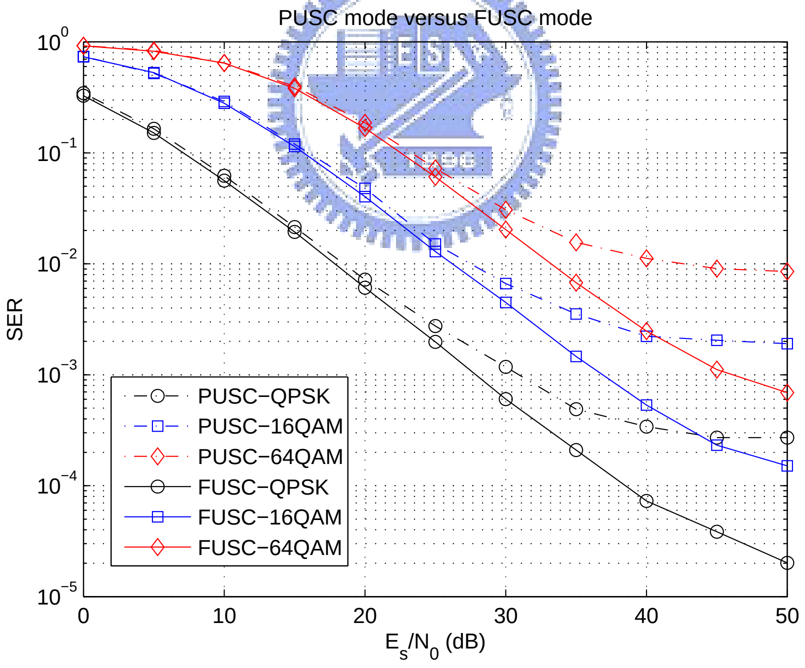
)shown in Fig. 6.23. Another reason is that when we do second-order interpolation, we need three estimated pilots and each cluster in PUSC mode contains only two pilots. It means that the remaining one pilot must be referred to another cluster which may be at a distance in frequency bands (because clusters in PUSC mode are distributive in available frequency bands).

6.5.2 Comparison Between Channel Estimation Techniques

We have shown many simulation results of many kinds of channel estimation techniques. Here we give an overall comparison in Fig. 6.24. The JCESD has the best performance and is followed by, MLE, 1D interpolation, 2D least squares, 2D interpolation and time averaging.

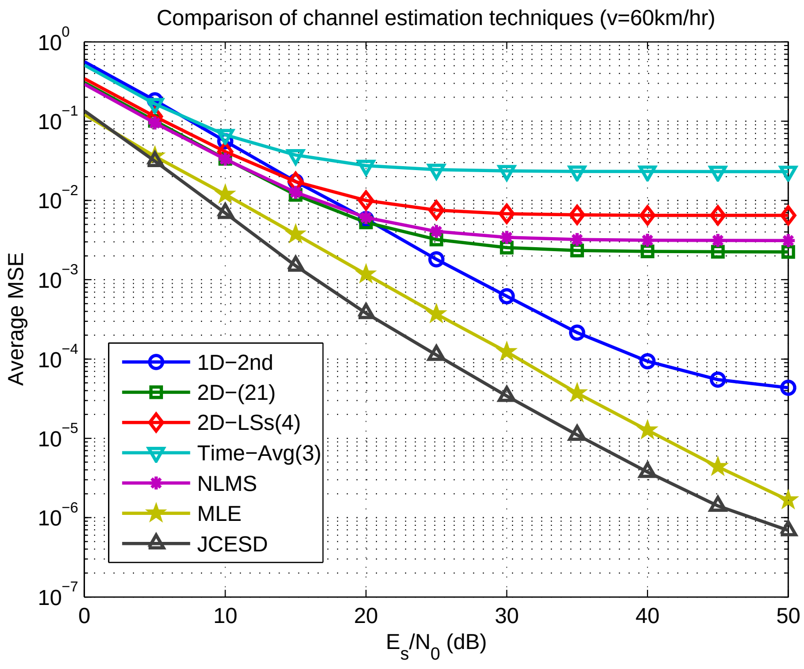


(a)

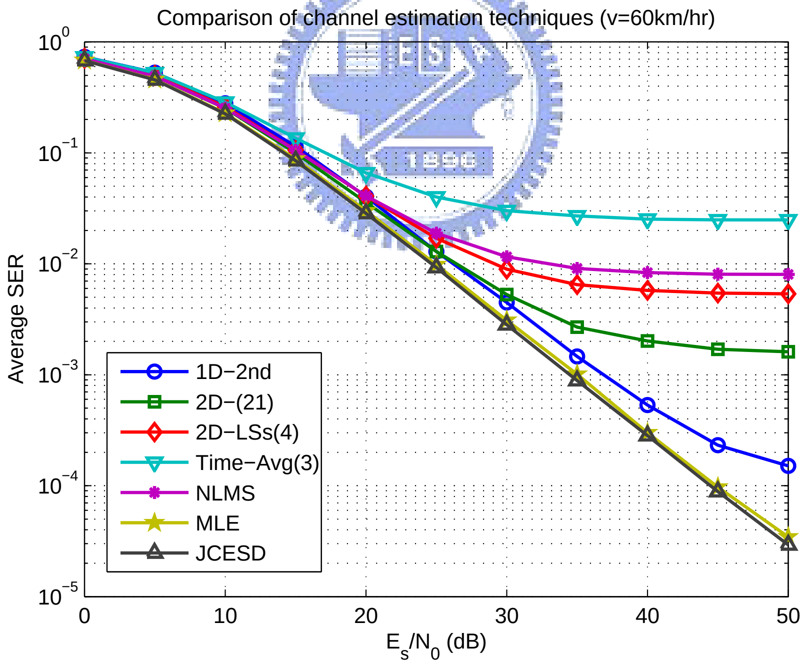


(b)

Figure 6.23: PUSC mode versus FUSC mode (1D-2nd interpolation). (a) MSE. (b) SER.



(a)



(b)

Figure 6.24: Comparison of channel estimation techniques. (a) MSE. (b) SER.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we presented several channel estimation techniques and applied them on OFDM uplink and OFDMA downlink. In OFDM uplink simulation, we used sample-spaced static multipath channels and in OFDMA downlink simulation, we adopted nonsample-spaced time-varying multipath channels. To do the channel estimation, first, we used LS estimator to estimated the channel frequency response on the pilot subcarriers. Second, interpolations or MLE were used to get a rough channel estimation. Third, we combined the rough estimation with some time domain information or did channel estimation and symbol detection iteratively.

In the case of OFDM uplink, it showed that with adroit manipulation of preambles and pilots, simple linear and second-order interpolation is good enough and their performance is acceptable.

In the case of OFDMA downlink, one-dimensional interpolation performed well and had no discrepancy under different velocities. In two-dimensional channel estimation, we wanted to use interpolation, linear prediction or NLMS to track the channel time variations. However, from our simulation, it seemed that intending to model the time variation is not a good idea. Maximum likelihood estimator was a good one because of its excellent performance. Nevertheless, it needed great computational precision and had matrix inversion problem, such like singularity and error propagation. Despite this, MLE was suitable for use with delicate manipulation. We could pre-compute the matrix with high precision and store it in the memory or a file. The joint channel estimation and symbol detection had the best performance but time-consuming. Just as its name suggested, we did channel estimation and symbol detection iteratively. But two or more times iteration did not give us any promising improvement. From our simulation, we suggested only doing JCESD one time.

As for the DSP implementation, we only implemented OFDMA uplink on TI's board. To achieve the real-time channel estimation, we replaced all operations into 16-bit fixed point operation. As shown in Table 5.4, the real-time goal was not yet completed and there was still much room to improvement.

7.2 Potential Future Work

The following work will emphasis on the DSP implementation and better estimation techniques. These tasks include the following:

- Implement OFDMA DL PUSC and OFDMA DL FUSC on DSP board.
- The interpolation can be accelerated by loading the pre-computed coefficient matrix from a file rather than direct computation.
- The matrix in MLE is constructed by adopting all pilots (e.g., 166 in OFDMA DL FUSC). The matrix can be shrinked while maintaining non-singularity by careful selection of pilots.
- The performance of joint channel estimation and symbol detection can be still boosted

by excluding those subcarriers whose channel gain are relatively low, that is, susceptible to noise. All we have to do is to observe the histogram like Fig. 6.14 and only use those better subcarriers.

- In this thesis, we do not consider the influence of intercarrier interference (ICI). The ICI simulation can be involved in the future.

Bibliography

[1] A. R. S. Bahai and B. R. Saltzberg, *Multi-carrier Digital Communications Theory and Applications of OFDM.* New York: Kluwer Academic, 1999.

[2] R. Prasad, *OFDM for Wireless Communications Systems.* Boston: Artech House, 2004.

[3] M. Engels, *Wireless OFDM System: How to Make Them Work?* Boston: Kluwer Academic Publishers, 2002.

[4] 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 24, 2004.

[5] IEEE Std 802.16e-2005 and IEEE Std 802.16-2004/Cor 1-2005, *IEEE Standard for Local and metropolitan area networks Part 16: Air Interface for Fixed and Mobile 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.

[6] O. Edfors, M. Sandell, J. J. van de Beek, D. Landstrom, and F. Sjoberg, "An introduction to orthogonal frequency-dicision multiplexing," http://courses.ece.uiuc.edu/ece459/spring02/ofdmtutorial.pdf.

[7] M.-H. Hsieh, "Synchronization and channel estimation techniques for OFDM systems," Ph.D. dissectation, Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C., May 1998.

[8] O. Edfors, M. Sandell, J. J. van de Beek, S. K. Wilson, and P. O. Börjesson, "OFDM channel estimation by singular value decomposition," in *IEEE 46th Vehicular Technology Conference*, Apr. 1996, pp. 923–927.

[9] C. K. Koc and G. Chen, "Authors' reply [Computational complexity of matrix inversion]," *IEEE Trans. Aerospace Electronic Systems,* vol. 30, no 4, p. 1115, Oct. 1994.

[10] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipe in C.* Cambridge Univ. Press, 2002.

[11] S. Coleri, M. Ergen, A. Puri, and A. Bahai, "Channel estimation techniques based on pilot arrangement in OFDM systems," *IEEE Trans. Broadcasting,* vol. 48, no. 3, pp. 223–229, Sep. 2002.

[12] M.-H. Hsieh and C.-H. Wei, "Channel estimation for OFDM systems based on comb-type pilot arrangement in frequency selective fading channels," *IEEE Trans. Consumer Electron.*, vol. 44, no. 1, pp. 217–225, Feb. 1998.

[13] S. G. Kang, Y. M. Ha, and E. K. Joo, "A comparative investigation on channel estimation algorithms for OFDM in mobile communications," *IEEE Trans. Broadcasting,* vol. 49, no. 2, pp. 142–149, June 2003.

[14] S. A. Dyer and J. S. Dyer, "Cubic-spline interpolation. 1," *IEEE Instrument. Measurement Mag.,* vol. 4, no. 1, pp. 44–46, Mar. 2001.

[15] S. A. Dyer and H. Xin, "Cubic-spline interpolation. 2," *IEEE Instrument. Measurement Mag.,* vol. 4, no. 2, pp. 34–36, June 2001.

[16] M. Morelli and U. Mengali, “A comparison of pilot-aided channel estimation methods for OFDM systems,” *IEEE Trans. Signal Processing*, vol. 49, pp. 3065–3073, Dec. 2001.

[17] T. S. Rappaport, *Wireless Communications Principles and Practice.* Upper Saddle River, New Jersey: Prentice Hall, 1996.

[18] B. Farhang-Boroujeny, *Adaptive Filters: Theory and Applications.* Wiley, 1998, pp. 139–199.

[19] A. Scherb, C. Zheng, C. Kühn, and K. D. Kammeyer, “Comparision of methods for iterative joint data detection and channel estimation,” *IEEE 59th Veh. Technol. Conf.*, vol. 1, May 2004, pp. 98–201.

[20] B. Han, X. Gao, X. You, J. Wang, and E. Costa, “An iterative joint channel estimation and symbol detection algorith, applied in OFDM system with high data to pilot power ratio”, *Conf. Rec., IEEE Int. Conf. Commun.*, vol. 3, May 2003, pp. 2076–2080.

[21] Innovative Integration, *Quixote User’s Manual.* Dec. 2003.

[22] Texas Instruments, *TMS320C6000 CPU and Instruction Set.* Literature number SPRU189F, Oct. 2000.

[23] Texas Instruments, *TMS320C6000 DSP Cache Users Guide.* Literature number SPRU656A, May. 2003.

[24] Yu-Sheng Chen, “DSP software implementation and integration of IEEE 802.16a TDD OFDMA downlink transceiver system,” M.S. thesis, Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C., June 2005.

[25] Texas Instruments, *Code Composer Studio User’s Guide.* Literature number SPRU328B, Feb. 2000.

[26] Texas Instruments, *TMS320C6000 Code Composer Studio Getting Started Guide.* Literature number SPRU509D, Aug. 2003.

[27] Texas Instruments, *TMS320C64x DSP Library Programmer's Reference.* Literature number SPRU565B, Oct. 2003.

[28] Texas Instruments, *TMS320C6000 Programmer's Guide.* Literature number SPRU198G, Oct. 2002.

[29] Ruu-Ching Chen, "Techniques for the DSP software implementation of IEEE 802.16a TDD OFDMA downlink pilot-symbol-aided channel estimation," M.S. thesis, Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C., June 2005.

[30] P. Dent, G. E. Bottomley and T. Croft, "Jakes fading model revisited," *Electronics Letters*, vol. 29, no. 13, pp. 1162-1163, June 1993.

作者簡歷

學生王治傑，民國七十一年八月出生於台灣桃園縣。民國九十三年六月畢業於國立清華大學電機工程學系，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。民國九十五年六月取得碩士學位，碩士論文題目為『IEEE 802.16e OFDM 上行及 OFDMA 下行通道估測技術與數位信號處理器實現之探討』。研究範圍與興趣包括：通訊系統、通道估測、數位信號處理等。