

國立交通大學

電子工程學系 電子研究所碩士班

碩 士 論 文

**IEEE 802.16e OFDMA 多輸入輸出通道估測技術之探討與數位
訊號處理器實現**



**Study in IEEE 802.16e OFDMA MIMO Channel Estimation
Techniques and Associated Digital Signal Processor
Implementation**

研 究 生：余光中

指 導 教 授：林大衛 博士

中 華 民 國 九 十 七 年 十 二 月



IEEE 802.16e OFDMA 多輸入輸出通道估測技術之
探討與數位訊號處理器實現

Study in IEEE 802.16e OFDMA MIMO Channel Estimation
Techniques and Associated Digital Signal Processor
Implementation

研究生: 余光中

Student: Kuang-Chung Yu

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



A Thesis

Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical and Computer Engineering
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of
Master of Science
in
Electronics Engineering
December 2008
Hsinchu, Taiwan, Republic of China

中華民國九十七年十二月



IEEE 802.16e OFDMA 多輸入輸出通道估測技術之探討

與數位訊號處理器實現

研究生:余光中

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

正交分頻多重進接(OFDMA)技術近來在行動環境中廣受注目且已經應用在許多數位通訊應用中。如果利用多通道傳輸(MIMO)更可提高其傳輸效率以及抵抗通道衰減的能力。本篇論文介紹 IEEE 802.16e 正交分頻多工存取(OFDMA)裡，多通道傳輸通道估計的問題、演算法、分析、以及實作方面的議題。

在通道估測當中，我們實做了兩個方法並去比較其效能。第一個是線性內插，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，之後我們分別在頻域以及時域上使用線性內插法來得到資料載波上的平率響應。第二個方法是 Wiener filter，由於此方法需要知道導訊載波以及資料載波通道的互相關，因此第一步是先用線性內插求得資料載波上的頻率響應，之後利用此結果去估測我們需要的互相關。最後我們利用其互相關以及導訊載波頻率響應的自相關求得 Wiener filter 在各個導訊載波的權重，以求得資料載波上更精確的頻率響應。其中 Wiener filter 的準確度和我們統計的通道範圍有關，因此對於不同的通道範圍我們也做了模擬。

在模擬當中，我們先在 AWGN 通道上驗證我們的模擬模型，然後再放置於多重路徑的 SUI-2 和 SUI-3 通道上模擬。

為了增進程式在數位處理器上的實行效率，我們將原始的浮點運算 C 程式修改為定點運算的程式版本。並做模擬以觀察其效能。

在本篇論文中，我們首先簡介 IEEE802.16e OFDMA 上行以及下行多通道傳輸的標準機制，以及各種通道估測的技術。接著是模擬多通道傳輸上使用不同通道估測技術的方式，最後介紹 DSP 的實現環境以及定點數的模擬結果。



Study in IEEE 802.16e OFDMA MIMO Channel Estimation Techniques and Associated Digital Signal Processor Implementation

Student: Kuang-Chung Yu

Advisor: Dr. David W. Lin

Department of Electronics Engineering
& Institute of Electronics
National Chiao Tung University

Abstract

OFDMA (orthogonal frequency division multiple access) technique has drawn much interest recently in the mobile transmission environment and been successfully applied to a wide variety of digital communications applications over the past several years. If we applied MIMO (multiple input and multiple output) technique it can enhance the performance of the transmission and the capability of the resistance the channel fading. In this thesis, we introduce the MIMO channel estimation problems, algorithms, analyses and implementation issues for IEEE 802.16e OFDMA PHY system.

In the channel estimation, we have implemented two methods to compare the performance. The first is linear interpolation. First we use LS estimator to estimate the channel response on pilot subcarriers, and then we use linear interpolation in time domain and frequency domain separately to get the frequency response on data subcarriers. The second method is Wiener filter. Because of using this method we need to know the cross-correlation of pilot and data subcarriers' channel responses, the first step is using the linear interpolation to get the frequency responses on data subcarriers and then using this result to estimate the cross-correlation we want. Finally we use the cross-correlation and the autocorrelation of pilot subcarrier channel responses to get the weight of pilot subcarriers, and get the more accurate channel response on data subcarriers. Meanwhile, the accuracy of Wiener filter is related to the rang of channel responses we use to average, so we also simulate in different channel range.

In the simulation, we verify our simulation model on AWGN channel and then do the

simulation on SUI-2 and SUI-3 multipath channel.

In order to increase the efficiency on DSP, we rewrite the floating-point C program to fixed-point version, and do the simulation to see the performance.

In the thesis, we first introduce the standard of the IEEE 802.16e OFDMA MIMO transmission and variant channel estimation methods. Then we simulate MIMO transmission using different channel estimation techniques. Finally we introduce the DSP implementation environment and fixed-point simulation result.



誌謝

本篇論文的順利完成，首先誠摯地感謝我的指導老師林大衛博士，感謝老師在這兩年多以來的細心指導，給予我在課業、研究上的幫助，使我學到了分析問題及解決問題的能力，同時，老師親切隨和的態度，也使我们能勇於發問，能夠勇於面對問題。在此，僅向老師致上最高的感謝之意。

另外要感謝的，是實驗室的洪崑健學長、吳俊榮學長及王海薇學姊。謝謝你們熱心地幫我解決了許多研究相關的問題。

感謝通訊電子與訊號處理實驗室(commmlab)，提供了充足的軟硬體資源，讓我在研究中不虞匱乏。感謝 94 級耀鈞、柏昇、依翎、順成四位學長的指導，以及 95 級的昀澤、婉清、佳楓等實驗室成員，讓我的研究生涯充滿歡樂又有所成長。

最後，要感謝的是我的家人，他們的支持讓我能夠心無旁騖的從事研究工作。謝謝所有幫助過我、陪我走過這一段歲月的師長、同儕與家人。謝謝！

誌於 2008.12 新竹交大

光中



Contents

1	Introduction	1
2	Introduction to IEEE802.16e OFDMA and MIMO Systems	3
2.1	Overview of OFDMA [4], [5]	3
2.1.1	Cyclic Prefix	4
2.1.2	Discrete-Time Baseband Equivalent System Model	5
2.2	Introduction to MIMO System	6
2.2.1	Transmit Diversity	7
2.2.2	Spatial Multiplexing	9
2.3	Basic OFDMA Symbol Structure in IEEE 802.16e	10
2.3.1	OFDMA Basic Terms	10
2.3.2	Frequency Domain Description	11
2.3.3	Primitive Parameters	11
2.3.4	Derived Parameters	12
2.3.5	Frame Structure	13
2.4	Uplink Transmission in IEEE 802.16e OFDMA	13

2.4.1	Data Mapping Rules	14
2.4.2	Carrier Allocations	15
2.4.3	Pilot Modulation	18
2.4.4	Data Modulation	19
2.5	Downlink Transmission in IEEE 802.16e OFDMA	19
2.5.1	Data Mapping Rules	20
2.5.2	Preamble Structure and Modulation	20
2.5.3	Subcarrier Allocations	22
2.5.4	Pilot Modulation	25
2.5.5	Data Modulation	25
2.6	Space-Time Coding in IEEE 802.16e OFDMA	25
2.6.1	STC Using Two Antennas	26
2.6.2	STC/FHDC Configurations	26
2.6.3	Uplink Using STC	27
2.6.4	STC Using Two Antennas in Downlink PUSC	27
3	Channel Estimation Techniques	29
3.1	Pilot-Symbol-Aided Channel Estimation [9]	29
3.1.1	The Least-Squares (LS) Estimator [10]	29
3.1.2	The LMMSE Estimator [11]	30
3.2	Two-Dimensional Channel Estimators	31
3.2.1	Linear Interpolation	31

3.2.2	2-D Wiener Filter [14]	32
4	Simulation of STC Uplink Channel Estimation	34
4.1	Linear Interpolation	34
4.2	Wiener Filtering	35
4.3	STC Decoding	38
4.4	Simulation Conditions	38
4.4.1	OFDMA Uplink System Parameters	38
4.4.2	Channel Models	39
4.5	Simulation Results	41
4.5.1	Simulation Flow	41
4.5.2	Validation of Simulation Model	42
4.5.3	Simulation Results and Analysis	43
5	Simulation of STC Downlink PUSC Channel Estimation	61
5.1	System Parameters and Channel Models	61
5.2	Linear Interpolation	61
5.3	Wiener Filtering	63
5.4	Simulation Study	65
5.4.1	Simulation Flow	65
5.4.2	Validation with AWGN Channel	65
5.4.3	Simulation Results	66

6	The DSP Hardware and Associated Software Development Environment	75
6.1	The TMS320C6416 DSP	75
6.1.1	TMS320C64x Features [21]	75
6.1.2	Central Processing Unit [21]	77
6.1.3	Memory Architecture [21]	83
6.2	The Code Composer Studio Development Tools [24], [25]	85
6.3	Code Optimization Methods [27]	87
6.3.1	Compiler Optimization Options [24], [25]	89
6.3.2	Using Ininsics	91
7	Fixed-Point DSP Implementation	92
7.1	Data Formats Considerations	92
7.2	Fixed-Point Simulation	93
7.3	DSP Computation Load	93
7.4	Program Code	106
8	Conclusion and Future Work	110
8.1	Conclusion	110
8.2	Potential Future Work	111
	Bibliography	113

List of Figures

2.1	Discrete-time model of the baseband OFDMA system (from[4]).	4
2.2	Time structure of OFDMA symbol (from [6]).	5
2.3	Discrete-time baseband equivalent of an OFDMA system with M users (from [5]).	6
2.4	Schematic block diagram of Alamouti's transmit diversity (from [8]).	7
2.5	Example of the data region which defines the OFDMA allocation (from [6]).	11
2.6	OFDMA frequency description (from [6]).	11
2.7	Example of an OFDMA frame (with only mandatory zone) in TDD mode (from [7]).	13
2.8	Example of mapping OFDMA slots to subchannels and symbols in the uplink (from [7]).	15
2.9	Structure of an uplink tile (from [6]).	15
2.10	PRBS generator for pilot modulation (from [6] and [7]).	18
2.11	QPSK, 16-QAM, and 64-QAM constellations (from [6]).	19
2.12	Example of mapping OFDMA slots to subchannels and symbols in the downlink in PUSC mode (from [7]).	21
2.13	Downlink transmission basic structure (from [6]).	21

2.14	Cluster structure (from [7]).	23
2.15	Illustration of STC (from [7]).	26
2.16	Mapping of data subcarriers in STTD mode (from [7]).	28
2.17	Cluster structure for STC PUSC using two antennas (from [7]).	28
4.1	Linear interpolation in STTD mode at antenna 0.	35
4.2	Wiener filtering in STTD mode at Antenna 0.	36
4.3	STTD transmission. (a) Neighboring channel responses are the same. (b) Responses are different.	39
4.4	Block diagram of the simulated system.	42
4.5	The SER curve for uncoded QPSK resulting from simulation matches the theoretical one.	44
4.6	MSE performance for uncoded QPSK resulting with linear interpolation, an- tenna 0.	45
4.7	SER performance for uncoded QPSK resulting from linear interpolation. . .	45
4.8	MSE performance of Wiener filtering channel estimation for uncoded QPSK, antenna 0. Autocorrelation and cross-correlation are obtained by averaging over one subchannel.	50
4.9	MSE performance of Wiener filtering channel estimation for uncoded QPSK, antenna 0. Autocorrelation and cross-correlation are obtained by averaging over ten subchannels.	51
4.10	Comparrision of SER performance with using Wiener filtering and linear in- terpolation channel estimation in STTD under QPSK modulation in AWGN. .	51

4.11	Comparrision of MSE performance with using Wiener filtering and linear interpolation channel estimation in STTD under QPSK modulation in AWGN.	52
4.12	MSE and SER performance for uncoded QPSK under Wiener filtering and linear interpolation channel estimations at different velocities in single-path Rayleigh fading channel with $\rho_{env} = 0$. (a) MSE. (b) SER.	54
4.13	MSE and SER performance for uncoded QPSK under Wiener filtering and linear interpolation channel estimation at different velocities in SUI-2 channel with channel correlation $\rho_{env} = 0$. (a) MSE. (b) SER.	55
4.14	MSE and SER performance for uncoded QPSK under Wiener filtering and linear interpolation channel estimation at different velocities in SUI-3 channel with channel correlation $\rho_{env} = 0$. (a) MSE. (b) SER.	56
4.15	Two different subchannel sets of MSE and SER performance for uncoded QPSK under Wiener filtering averaging over one subchannel at different velocities in SUI-2 channel with channel correlation $\rho_{env} = 0$. (a) MSE. (b) SER.	57
4.16	SER comparison between zero and nonzero antenna correlation ($\rho_{env} = 0.7$) in single-path Rayleigh fading.	58
4.17	SER comparison between zero and nonzero antenna correlation ($\rho_{env} = 0.5$) in SUI-2 channel.	59
4.18	SER comparison between zero and nonzero antenna correlation ($\rho_{env} = 0.4$) in SUI-3 channel.	60
5.1	Linear interpolation in STTD mode at antenna 0.	63
5.2	Wiener filtering in STTD mode at antenna 0.	64

5.3	Block diagram of the simulated system.	65
5.4	SER for uncoded QPSK resulting from simulation compared with theory. . .	66
5.5	MSE and SER performance for uncoded QPSK resulting from simulation with Wiener filtering and linear interpolation in AWGN channel. (a) MSE. (b) SER.	69
5.6	MSE and SER performance for uncoded QPSK resulting from simulation with Wiener filtering and linear interpolation at different velocities in single-path Rayleigh fading channel with $\rho_{env} = 0$. (a) MSE. (b) SER.	70
5.7	MSE and SER performance for uncoded QPSK resulting from simulation with Wiener filtering and linear interpolation at different velocities in SUI-2 channel with $\rho_{env} = 0$. (a) MSE. (b) SER.	71
5.8	MSE and SER performance for uncoded QPSK resulting from simulation with Wiener filtering and linear interpolation at different velocities in SUI-3 channel with $\rho_{env} = 0$. (a) MSE. (b) SER.	72
5.9	SER comparison of zero and nonzero antenna correlations ($\rho_{env} = 0.7$) in single-path Rayleigh fading.	73
5.10	SER comparison of zero and nonzero antenna correlations ($\rho_{env} = 0.5$) in SUI-2.	74
5.11	SER comparison of zero and nonzero antenna correlations ($\rho_{env} = 0.4$) in SUI-3.	74
6.1	The DSP on the Sundance board [21].	76
6.2	Block diagram of the TMS320C6416 DSP [21].	78
6.3	Pipeline phases of TMS320C6416 DSP [21].	79
6.4	TMS320C64x CPU data paths [21].	84

6.5	Code development flow for TI C6000 DSP [27].	88
7.1	fix point simulation flow.	93
7.2	Uplink channel estimation performance under fixed- and floating-point computation in AWGN. (a) MSE. (b) SER.	94
7.3	Uplink channel estimation performance under fixed- and floating-point computation in single-path Rayleigh fading. (a) MSE. (b) SER.	95
7.4	Uplink channel estimation performance under fixed- and floating-point computation in SUI-2 channel. (a) MSE. (b) SER.	96
7.5	Uplink channel estimation performance under fixed- and floating-point computation in SUI-3 channel. (a) MSE. (b) SER.	97
7.6	Uplink channel estimation performance under fixed- and floating-point computation in Vehicular A channel. (a) MSE. (b) SER.	98
7.7	MSE and SER under fixed- and floating-point computation in AWGN. (a) MSE. (b) SER.	99
7.8	Downlink channel estimation performance under fixed- and floating-point computation in single-path Rayleigh fading. (a) MSE. (b) SER.	100
7.9	Downlink channel estimation performance under fixed- and floating-point computations in SUI-2 channel. (a) MSE. (b) SER.	101
7.10	Downlink channel estimation performance under fixed- and floating-point computation in SUI-3 channel. (a) MSE. (b) SER.	102
7.11	Downlink channel estimation performance under fixed- and floating-point computation in Vehicular A channel (a) MSE. (b) SER.	103
7.12	Wiener filtering C code block diagram.	105

7.13	<i>FIXED.H</i>	106
7.14	<i>linear_interpolation</i>	107
7.15	Part of assembly code of function <i>linear_interpolaton</i>	108
7.16	Software pipelineing information of function <i>linear_interpolation</i>	109



List of Tables

2.1	OFDMA Uplink Subcarrier Allocations [6], [7]	16
2.2	OFDMA Downlink Subcarrier Allocation Under PUSC [6], [7]	23
4.1	α_i of Eight Data Subcarriers at Antenna 0.	37
4.2	OFDMA Uplink Parameters	40
4.3	Channel Profiles of SUI-2 and SUI-3 [17]	41
4.4	Power-Delay Profile of the ETSI Vehicular A Channel	42
4.5	Mean Delay and RMS Delay Spread	42
4.6	MSE of Eight Data Subcarriers at Antennas 0 and 1	44
5.1	OFDMA Downlink Parameters	62
6.1	Execution Stage Length Description for Each Instruction Type [21]	80
6.2	Functional Units and Operations Performed (Part 1 of 2) [21]	81
6.3	Functional Units and Operations Performed (Part 2 of 2) [21]	82
7.1	OFDMA Uplink DSP Load Under 1024-FFT with 10 Subchannel	105
7.2	OFDMA Downlink DSP Load Under 1024-FFT, Major Group 0 with STC	106

Chapter 1

Introduction

Orthogonal frequency division multiple access (OFDMA) has emerged as one of the prime multiple access schemes for broadband wireless networks (e.g., IEEE 802.16 Mobile WiMAX, IEEE 802.20 and 3G LTE). As a special case of multicarrier multiple access schemes, OFDMA exclusively assigns each subchannel to only one user, eliminating intra-cell interference [1]. In frequency selective channels, an intrinsic advantage of OFDMA is its capability to exploit the so-called multiuser diversity provided by multipath channels. Other advantages of OFDMA include finer granularity and better link budget [1]. OFDMA 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). OFDMA is used in the 2 to 11 GHz systems. The IEEE Standard 802.16-2004 is for broadband wireless access systems that provide a variety of wireless access services to fixed outdoor and indoor users. The 802.16e is designed to support terminal mobility, and currently it can serve terminals with a speed up to 120 km/h [2].

Multiple-antenna techniques can be used to increase diversity and improve the bit error rate (BER) performance of wireless systems, increase the cell range, increase the transmitted

data rate through spatial multiplexing, and/or reduce interference from other users. The WiMAX Forum has selected two different multiple antenna profiles for use on the downlink and uplink. One of them is based on the space-time code (STC) proposed by Alamouti for transmit diversity [3], and the other is a 2x2 spatial multiplexing scheme.

This thesis focuses on the channel estimation methods for IEEE802.16e WirelessMAN-OFDMA multi-input multi-output (MIMO) systems. We construct the multipath channel simulator to simulate the MIMO system in IEEE 802.16e and study the performance of different channel estimation methods. And we consider software implementation of the channel estimation using a digital signal processor (DSP).

The thesis is organized as follows. First, in chapter 2, we introduce the OFDMA specifications in IEEE 802.16e, especially its MIMO mode of operation. In chapter 3, various channel estimation techniques are introduced. In chapter 4, we discuss the performance of channel estimation in uplink transmission and in chapter 4, we show the performance of downlink channel estimation. It is seen that, due to modeling errors in parameter estimations, linear interpolation performs better than Wiener filtering. In chapter 6, we describe the implementation platform, which consists of a Texas Instruments' TMS320C6416 DSP on a Sundance compancy's Carrier board. In chapter 7, we present some DSP implementation issues and fixed-point simulation results. Finally, chapter 8 gives the conclusion and points out some potential future work.

Chapter 2

Introduction to IEEE802.16e OFDMA and MIMO Systems

We first introduce the basic concepts of the OFDMA and MIMO techniques for multicarrier modulation. The specifications of IEEE 802.16e are introduced afterwards.

2.1 Overview of OFDMA [4], [5]

Orthogonal frequency-division multiple-access (OFDMA) is a major multiple access scheme considered for future wireless systems. In an OFDMA system, several users simultaneously transmit their data by modulating mutually exclusive sets of orthogonal subcarriers. Thus each user's signal can be separated easily in the frequency domain. One typical structure is the subband OFDMA, which divides all available subcarriers into a number of subbands. Each user is allowed to use one or more available subbands for the data transmission. Pilot symbols are employed for the estimation of channel state information (CSI) within the subbands. Besides multiuser diversity, robustness to narrowband interference and capability of channel assignment are two other advantages of OFDMA. Figure 2.1 shows an OFDMA network in which active users simultaneously communicate with the base station (BS).

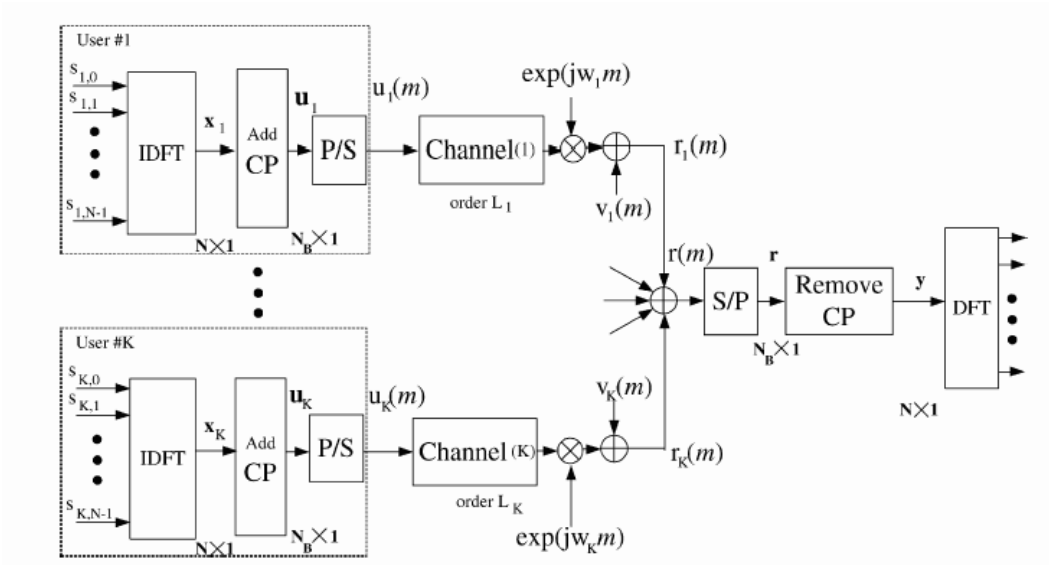


Figure 2.1: Discrete-time model of the baseband OFDMA system (from[4]).

2.1.1 Cyclic Prefix

Cyclic prefix (CP), or guard time, is used to overcome the intersymbol interference (ISI) and interchannel interference (ICI) problems. The multiuser channel is assumed to be substantially invariant within one OFDMA symbol duration. The symbol timing mismatch is assumed to be smaller than the CP duration. In this scenario, users do not interfere each other in the frequency domain.

A CP is a copy of the last part of the OFDMA symbol (see Fig. 2.2). It is used to collect multipath propagation effects of the last symbol so as to maintaining the orthogonality of the tones. However, the transmitter energy increases with the length of the guard time while the receiver energy remains the same (since the cyclic extension is discarded in the receiver), so there is a $10 \log(1 - T_g/(T_b + T_g))/\log(10)$ dB loss in E_b/N_0 .

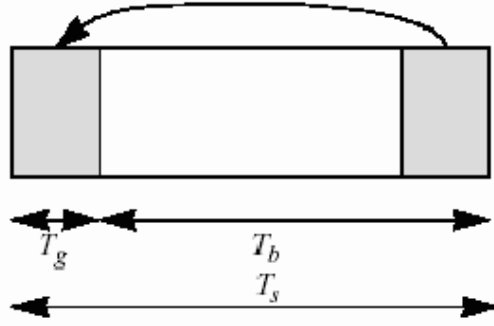


Figure 2.2: Time structure of OFDMA symbol (from [6]).

2.1.2 Discrete-Time Baseband Equivalent System Model

The material in this subsection is mainly taken from [5]. Consider an OFDMA system with M active users sharing a bandwidth of $B = \frac{1}{T}$ Hz (where T is the sampling period) as shown in Fig. 2.3. The system consists of K subcarriers, of which K_u are useful subcarriers (excluding guard bands and DC subcarrier). The users are allocated non-overlapping subcarriers in the spectrum depending on their needs.

The discrete time baseband channel consists of L multipath components and has the form

$$h(l) = \sum_{m=0}^{L-1} h_m \delta(l - l_m) \quad (2.1)$$

where h_m is a zero-mean complex Gaussian random variable with $E[h_i h_j^*] = 0$ for $i \neq j$. In frequency domain,

$$H = F \underline{h} \quad (2.2)$$

where $H = [H_0, H_1, \dots, H_{K-1}]^T$, $\underline{h} = [h_0, \dots, h_{L-1}, 0, \dots, 0]^T$ and F is K -point DFT matrix. The impulse response length l_{L-1} is upper bounded by the length of CP (L_{cp}).

The received signal in frequency domain is given by

$$\underline{Y}_n = \sum_{i=1}^M X_{i,n} \underline{H}_{i,n} + \underline{V}_n \quad (2.3)$$

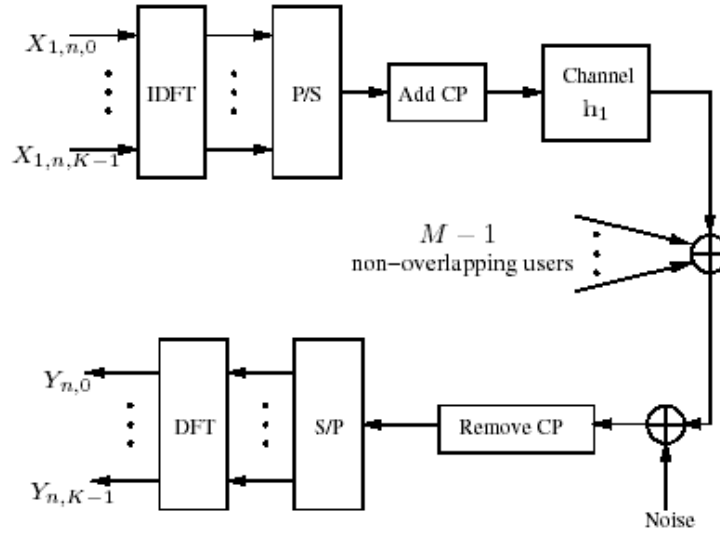


Figure 2.3: Discrete-time baseband equivalent of an OFDMA system with M users (from [5]).

where $X_{i,n} = \text{diag}(X_{i,n,0}, \dots, X_{i,n,K-1})$ is a $K \times K$ diagonal data matrix and $\underline{H}_{i,n}$ is the $K \times 1$ channel vector (2.2) corresponding to the i th user in n th symbol. The noise vector \underline{V}_n is distributed as $\mathcal{CN}(0, \sigma^2 I_K)$ where I_k is the K -dimensional identity matrix.

2.2 Introduction to MIMO System

In this section, we introduce two MIMO transmission mechanisms that are used in WiMAX. One is called transmit diversity and the other spatial multiplexing. The material in this section is mainly taken from [8].

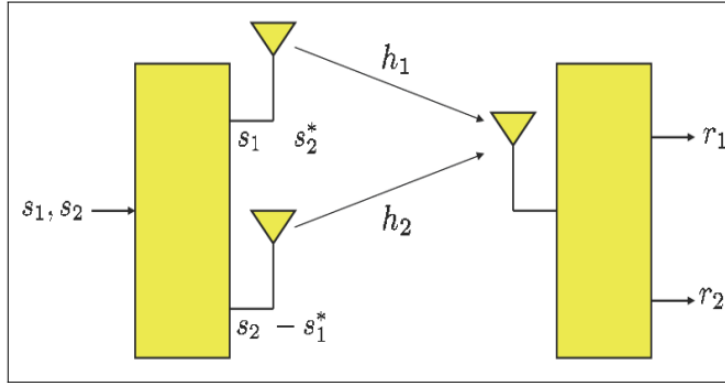


Figure 2.4: Schematic block diagram of Alamouti's transmit diversity (from [8]).

2.2.1 Transmit Diversity

One MIMO transmission technique employed in WiMAX is the space-time coding (STC) scheme proposed by Alamouti [3] for transmit diversity. (In the IEEE 802.16e-2005 specifications, this scheme is referred to as Matrix A.) This technique can be described as follows.

Let (s_1, s_2) represent a group of two consecutive symbols in the input data stream to be transmitted. During a first symbol period t_1 , transmit (Tx) antenna 1 transmits symbol s_1 and Tx antenna 2 transmits symbol s_2 . Next, during the second symbol period t_2 , Tx antenna 1 transmits symbol s_2^* and Tx antenna 2 transmits symbol $-s_1^*$. Denote the channel response (at the subcarrier frequency at hand) from Tx1 to the receiver (Rx) by h_1 and the channel response from Tx2 to the receiver by h_2 . The received signal samples corresponding to the symbol periods t_1 and t_2 can be written as:

$$r_1 = h_1 s_1 + h_2 s_2 + n_1, \quad (2.4)$$

$$r_2 = h_1 s_2^* + h_2 s_1^* + n_2, \quad (2.5)$$

where n_1 and n_2 are additive noise terms. The receiver computes the following signals to

estimate the symbols s_1 and s_2 :

$$x_1 = h_1^* r_1 - h_2 r_2^* = (|h_1|^2 + |h_2|^2) s_1 + h_1^* n_1 - h_2 n_2^*, \quad (2.6)$$

$$x_2 = h_2^* r_1 + h_1 r_2^* = (|h_1|^2 + |h_2|^2) s_2 + h_2^* n_1 + h_1 n_2^*. \quad (2.7)$$

These expressions clearly show that x_1 (resp. x_2) can be sent to a threshold detector to estimate symbol s_1 (resp. symbol s_2) without interference from the other symbol. Moreover, since the useful signal coefficient is the sum of the squared moduli of two independent fading channels, these estimations benefit from perfect second-order diversity, equivalent to that of Rx diversity under maximum-ratio combining (MRC). Alamouti's transmit diversity can also be combined with MRC when two antennas are used at the receiver. In this scheme, the received signal samples corresponding to the symbol periods t_1 and t_2 can be written as

$$r_{11} = h_{11} s_1 + h_{12} s_2 + n_{11}, \quad (2.8)$$

$$r_{12} = h_{11} s_2^* - h_{12} s_1^* + n_{12}, \quad (2.9)$$

for the first receive antenna, and

$$r_{21} = h_{21} s_1 + h_{22} s_2 + n_{21}, \quad (2.10)$$

$$r_{22} = h_{21} s_2^* - h_{22} s_1^* + n_{22}, \quad (2.11)$$

for the second receive antenna. In these expressions, h_{ji} designates the channel response from Tx i to Rx j , with $i, j = 1, 2$, and n_{ij} designates the noise on the corresponding channel. This MIMO scheme does not give any spatial multiplexing gain, but it has 4th-order diversity, which can be fully recovered by a simple receiver as follows. The optimum receiver estimates

the transmitted symbols s_1 and s_2 using

$$\begin{aligned} x_1 &= h_{11}^* r_{11} - h_{12} r_{12}^* + h_{21}^* r_{21} - h_{22} r_{22}^* \\ &= (|h_{11}|^2 + |h_{12}|^2 + |h_{21}|^2 + |h_{22}|^2) s_1 + h_{11}^* n_{11} - h_{12} n_{12}^* + h_{21}^* n_{21} - h_{22} n_{22}^*, \end{aligned} \quad (2.12)$$

$$\begin{aligned} x_2 &= h_{12}^* r_{11} + h_{11} r_{12}^* + h_{22}^* r_{21} + h_{21} r_{22}^* \\ &= (|h_{11}|^2 + |h_{12}|^2 + |h_{21}|^2 + |h_{22}|^2) s_2 + h_{12}^* n_{11} + h_{11} n_{12}^* + h_{22}^* n_{21} + h_{21} n_{22}^*. \end{aligned} \quad (2.13)$$

These equations clearly show that the receiver fully recovers the fourth-order diversity of the 2×2 system.

2.2.2 Spatial Multiplexing

The second MIMO technique employed in WiMAX is the 2×2 spatial multiplexing (using the so-called matrix $B = (s_1, s_2)^T$). This technique does not offer any diversity gain from the Tx side. But it offers a diversity gain of 2 on the receiver side when detected using maximum-likelihood (ML) detection.

To describe the technique, we omit the time and frequency dimensions, leaving only the space dimension. The symbols transmitted by Tx1 and Tx2 in parallel are denoted s_1 and s_2 , respectively. Denoting by h_{ji} the channel response from Tx i to Rx j ($i, j = 1, 2$), the signals received by the two Rx antennas are given by

$$r_1 = h_{11}s_1 + h_{12}s_2 + n_1, \quad (2.14)$$

$$r_2 = h_{21}s_1 + h_{22}s_2 + n_2, \quad (2.15)$$

which can be written in matrix form as

$$\begin{bmatrix} r_1 \\ r_2 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix} + \begin{bmatrix} n_1 \\ n_2 \end{bmatrix}. \quad (2.16)$$

The ML detector makes an exhaustive search over all possible values of the transmitted

symbols and decides in favor of (s_1, s_2) which minimizes the Euclidean distance

$$D(s_1, s_2) = |r_1 - h_{11}s_1 - h_{12}s_2|^2 + |r_2 - h_{21}s_1 - h_{22}s_2|^2. \quad (2.17)$$

2.3 Basic OFDMA Symbol Structure in IEEE 802.16e

The WirelessMAN-OFDMA PHY is based on OFDM modulation and is designed for nonline-of-sight (NLOS) operation in frequency bands below 11 GHz. For licensed bands, channel bandwidths allowed shall be limited to the regulatory provisioned bandwidth divided by any power of 2 no less than 1.0 MHz. The material in this section is mainly taken from [5] and [6].

2.3.1 OFDMA Basic Terms

We introduce some basic terms in OFDMA PHY. These definitions help us understand the concepts in subcarrier allocation and transmission of IEEE 802.16e OFDMA.

- Slot: A slot in 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 (DL) PUSC (Partial Usage of SubChannels), one slot is one subchannel by two OFDMA symbols. For uplink (UL), one slot is one subchannel by three OFDMA symbols.
- Data region: In OFDMA, a data region is a two-dimensional allocation of a group of contiguous subchannels in a group of contiguous OFDMA symbols. All the allocations refer to logical subchannels. A two-dimensional allocation may be visualized as a rectangle, such as the 4×3 rectangle shown in Fig. 2.5.
- Segment: A segment is a subdivision of the set of available OFDMA subchannels (that may include all available subchannels). One segment is used for deploying a single instance of the MAC.

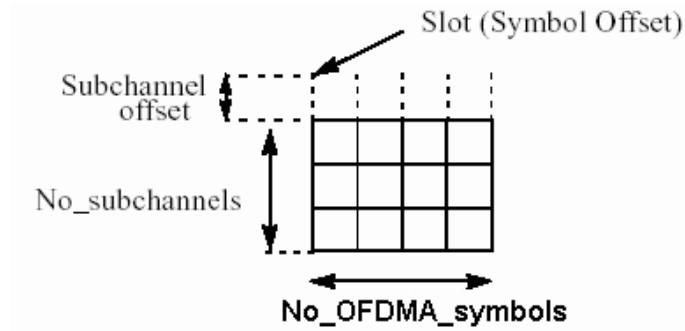


Figure 2.5: Example of the data region which defines the OFDMA allocation (from [6]).

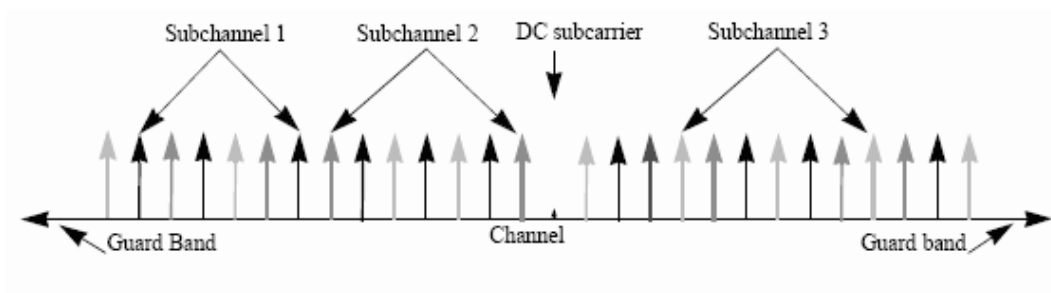


Figure 2.6: OFDMA frequency description (from [6]).

2.3.2 Frequency Domain Description

An OFDMA symbol (see Fig. 2.6) is made up of subcarriers, the number of which determines the FFT size used. There are several subcarrier types:

- Data subcarriers: for data transmission.
- Pilot subcarriers: for various estimation purposes.
- Null subcarriers: no transmission at all, for guard bands and DC subcarrier.

2.3.3 Primitive Parameters

Four primitive parameters characterize the OFDMA symbols:

- BW : the nominal channel bandwidth.
- N_{used} : number of used subcarriers (which includes the DC subcarrier).
- n : sampling factor. This parameter, in conjunction with BW and N_{used} , determines the subcarrier spacing and the useful symbol time. For channel bandwidths that are a multiple of 1.75 MHz, $n = 8/7$, else for channel bandwidths that are a multiple of any of 1.25, 1.5, 2 or 2.75 MHz, $n = 28/25$, else for channel bandwidths not otherwise specified, $n = 8/7$.
- G : the ratio of CP time to “useful” time, i.e., T_{cp}/T_s . The following values shall be supported: 1/32, 1/16, 1/8, and 1/4.

2.3.4 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: $F_s = \text{floor}(n \cdot BW / 8000) \times 8000$.
- Subcarrier spacing: $\Delta f = F_s / N_{FFT}$.
- Useful symbol time: $T_b = 1 / \Delta f$.
- CP time: $T_g = G \times T_b$.
- OFDMA symbol time: $T_s = T_b + T_g$.
- Sampling time: T_b / N_{FFT} .

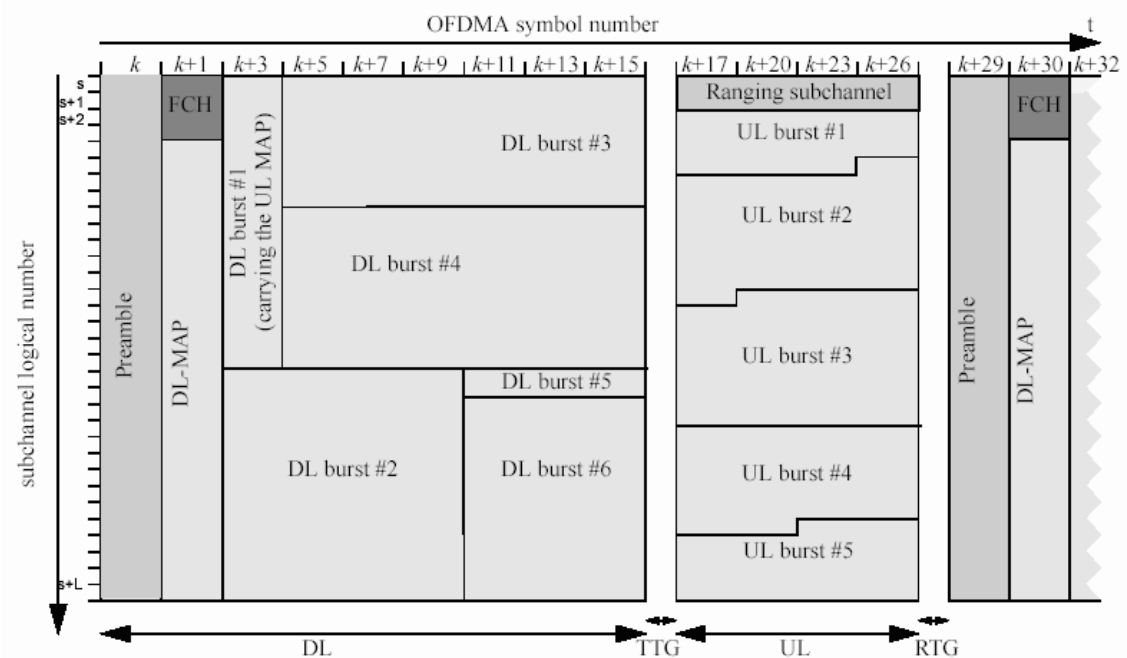


Figure 2.7: Example of an OFDMA frame (with only mandatory zone) in TDD mode (from [7]).

2.3.5 Frame Structure

When implementing a time-division duplex (TDD) system, the frame structure is built from base station (BS) and subscriber station (SS) transmissions. Each frame in the DL transmission begins with a preamble followed by a DL transmission period and a UL transmission period. In each frame, the TTG and RTG shall be inserted between the downlink and uplink and at the end of each frame, respectively, to allow the BS to turn around. Fig. 2.7 shows an example of an OFDMA frame with only mandatory zone in TDD mode.

2.4 Uplink Transmission in IEEE 802.16e OFDMA

The material in this section is mainly taken from [6] and [7].

2.4.1 Data Mapping Rules

The UL mapping consists of two stages. In the first stage, the OFDMA slots allocated to each burst are selected. In the second stage, the allocated slots are mapped.

Stage 1: Allocate OFDMA slots to bursts. A UL allocation is created by selecting an integer number of contiguous slots according to the ordering of steps 1 to 3. This results in the general burst structure shown by the gray area in Fig. 2.8.

- 1) Segment the data into blocks sized to fit into one OFDMA slot.
- 2) Each slot shall span one or more subchannels in the subchannel axis and one or more OFDMA symbols in the time axis (see Fig. 2.8 for an example). 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 symbol index is increased. When the edge of the UL zone is reached, continue the mapping from the lowest numbered OFDMA symbol in the next available subchannel.
- 4)

Stage 2: Map OFDMA slots within the UL allocation.

- 1) Map the slots such that the lowest numbered slot occupies the lowest numbered subchannel in the lowest numbered OFDMA symbol.
- 2) Continue the mapping such that the subchannel index is increased. When the last subchannel is reached, continue the mapping from the lowest numbered subchannel in the next OFDMA symbol that belongs to the UL allocation. The resulting order is shown by the arrows in Fig. 2.8.

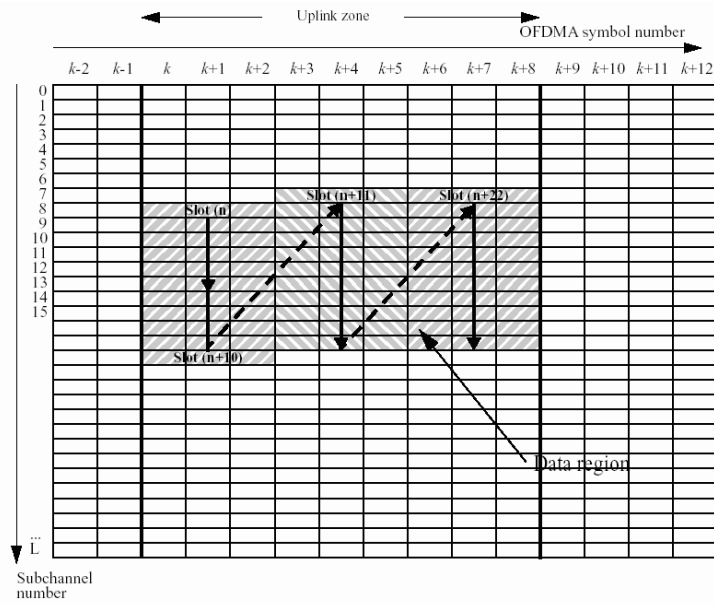


Figure 2.8: Example of mapping OFDMA slots to subchannels and symbols in the uplink (from [7]).

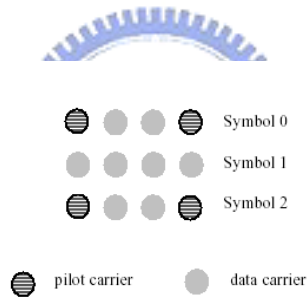


Figure 2.9: Structure of an uplink tile (from [6]).

Fig. 2.8 illustrates the order of OFDMA slots mapping to subchannels and OFDMA symbols.

2.4.2 Carrier Allocations

Consider the 1024-FFT PUSC permutation for example. Under it, the uplink supports 35 subchannels. Each transmission uses 48 data carriers as the minimal block of processing. Each new transmission for the uplink commences with the parameters as given in Table 2.1.

Table 2.1: OFDMA Uplink Subcarrier Allocations [6], [7]

Parameter	Value	Notes
Number of DC subcarriers	1	Index 512 (counting from 0)
N_{used}	841	Number of all subcarriers used within a symbol
Guard subcarriers: Left, Right	92,91	
TilePermutation		Used to allocate tiles to subchannels 11, 19, 12, 32, 33, 9, 30, 7, 4, 2, 13, 8, 17, 23, 27, 5, 15, 34, 22, 14, 21, 1, 0, 24, 3, 26, 29, 31, 20, 25, 16, 10, 6, 28, 18
$N_{subchannels}$	35	
$N_{subcarriers}$	24	
N_{tiles}	210	
Number of subcarriers per tile	4	Number of all subcarriers within a tile
Tiles per subchannel	6	

A slot in the uplink is composed of three OFDMA symbols and one subchannel. Within each slot, there are 24 data subcarriers and 12 pilot subcarriers. The subchannel is constructed from six uplink tiles, each having four successive active subcarriers with the configuration as illustrated in Fig. 2.9.

The usable subcarriers in the allocated frequency band shall be divided into N_{tiles} physical tiles with parameters from Table 2.1. The allocation of physical tiles to logical tiles in subchannels is performed according to:

$$Tiles(s, n) = N_{subchannels} \cdot n + (Pt[(s + n) \bmod N_{subchannels}] + UL_PermBase) \bmod N_{subchannels}$$

where:

- $Tiles(s, n)$ is the physical tile index in the FFT with tiles being ordered consecutively from the most negative to the most positive used subcarrier (0 is the starting tile index),

- n is the tile index 0..5 in a subchannel,
- Pt is the tile permutation,
- s is the subchannel number in the range $0..N_{subchannels} - 1$,
- $UL_PermBase$ is an integer value in the range 0..69, which is assigned by a management entity, and
- $N_{subchannels}$ is the number of subchannels for the FFT size given in Table 2.1.

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

- 1) After allocating the pilot carriers within each tile, indexing of the data subcarriers within each slot is performed starting from the first symbol at the lowest indexed subcarrier of the lowest indexed tile and continuing in an ascending manner through the subcarriers in the same symbol, then going to the next symbol at the lowest indexed data subcarrier, and so on. Data subcarriers shall be indexed from 0 to 47.
- 2) The mapping of data onto the subcarriers will follow the equation below. This equation calculates the subcarrier index (as assigned in item 1) to which the data constellation point is to be mapped:

$$Subcarrier(n, s) = (n + 13 \cdot s) \text{ mod } N_{subcarriers}$$

where:

- $Subcarrier(n, s)$ is the permuted subcarrier index corresponding to data subcarrier n is subchannel s ,
- n is a running index between 0 and 47, indicating the data constellation point,

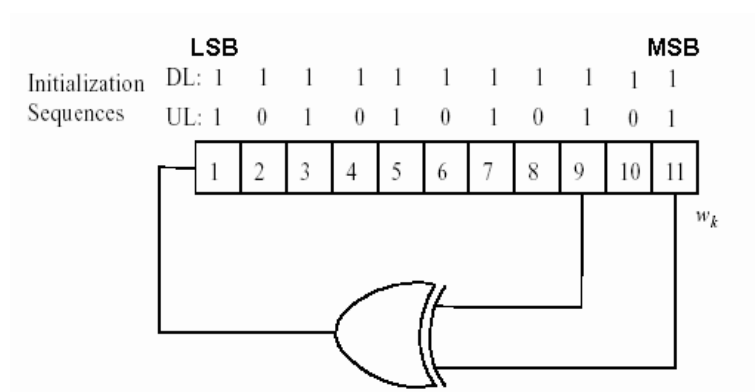


Figure 2.10: PRBS generator for pilot modulation (from [6] and [7]).

- s is the subchannel number, and
- $N_{subcarriers}$ is the number of subcarriers per slot.

2.4.3 Pilot Modulation

The PRBS (pseudo-random binary sequence) generator depicted in Fig. 2.10 is used to produce a sequence, w_k . The value of the pilot modulation, on subcarrier k , shall be derived from w_k .

For the mandatory tile structure in the uplink, pilot subcarriers shall be inserted into each data burst in order to constitute the symbol and they shall be modulated according to their subcarrier location within the OFDMA symbol. The pilot subcarriers shall be modulated according to

$$\Re\{c_k\} = 2\left(\frac{1}{2} - w_k\right), \quad \Im\{c_k\} = 0. \quad (2.18)$$

In all permutations except UL PUSC, downlink TUSC1, and the DL and UL STC permutations/modes, each pilot shall be transmitted with a boosting of 2.5 dB over the average non-boosted power of each data tone. That is, these pilot subcarriers shall be modulated

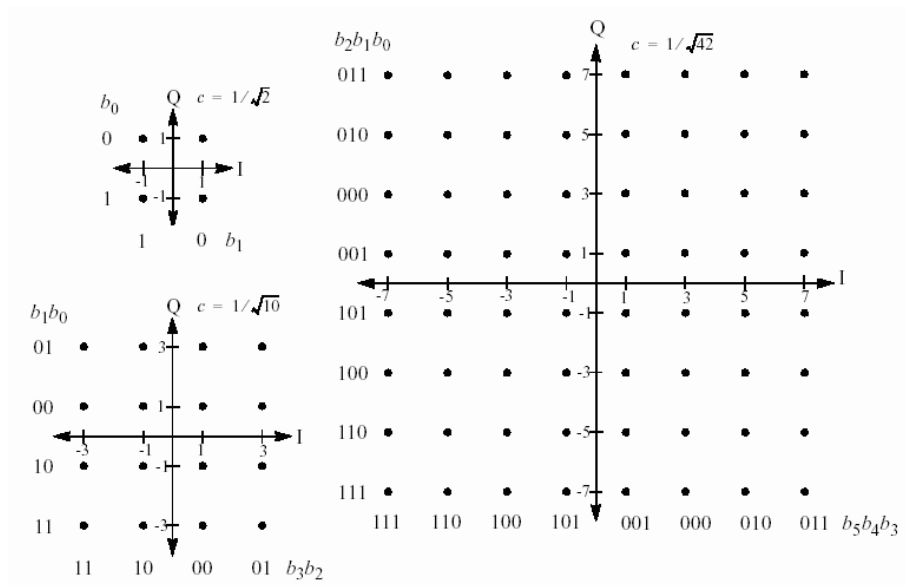
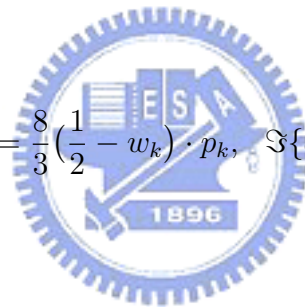


Figure 2.11: QPSK, 16-QAM, and 64-QAM constellations (from [6]).

according to

$$\Re\{c_k\} = \frac{8}{3} \left(\frac{1}{2} - w_k \right) \cdot p_k, \quad \Im\{c_k\} = 0. \quad (2.19)$$



2.4.4 Data Modulation

The employed constellations are as shown in Fig. 2.11. 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.

2.5 Downlink Transmission in IEEE 802.16e OFDMA

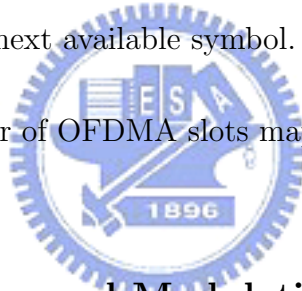
This section briefly introduces the specifications of IEEE 802.16e OFDMA PUSC downlink transmission. The material is mainly taken from [6] and [7].

2.5.1 Data Mapping Rules

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 of OFDMA slots mapping to subchannels and OFDMA symbols.



2.5.2 Preamble Structure and Modulation

Fig. 2.13 shows a downlink transmission period. The first symbol of the downlink transmission is the preamble. There are three types of preamble carrier-sets, which are defined below. The subcarriers in the preamble are modulated using a boosted BPSK modulation with a specific pseudo-noise (PN) code. The PN series modulating the pilots in the preamble can be found in [6, pp. 553–562].

The preamble carrier-sets are defined as

$$PreambleCarrierSet_n = n + 3 \cdot k, \quad (2.20)$$

where:

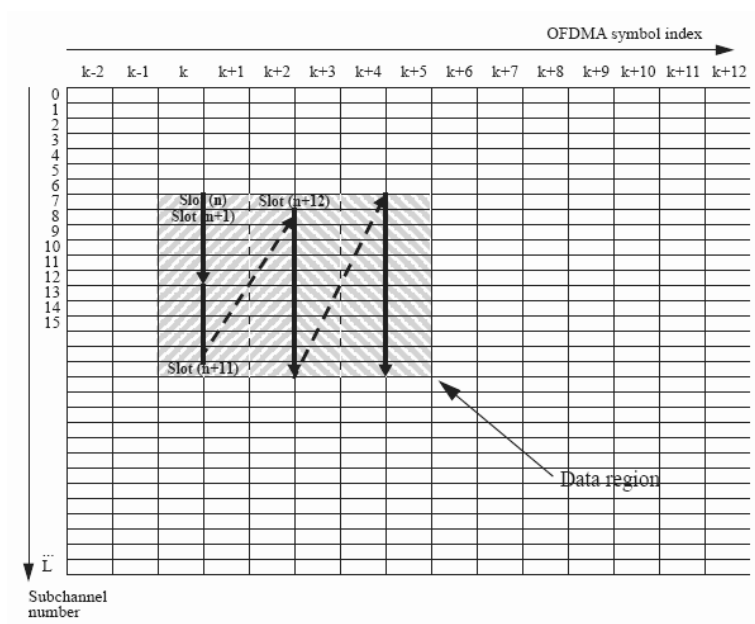


Figure 2.12: Example of mapping OFDMA slots to subchannels and symbols in the downlink in PUSC mode (from [7]).

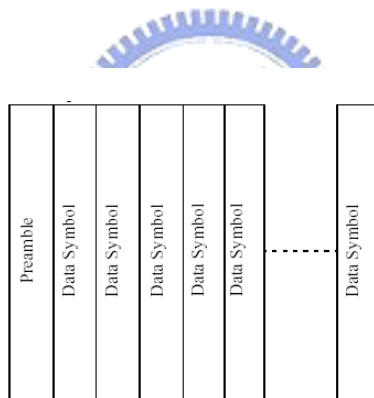


Figure 2.13: Downlink transmission basic structure (from [6]).

- $PreambleCarrierSet_n$ specifies all subcarriers allocated to the specific preamble,
- n is the index of the preamble carrier-set indexed $0 \leq 1 \leq 2$ and
- k is a running index, $0 \leq k \leq 283$.

Each segment uses one type of preamble out of the three sets in the following manner: For the preamble symbol, there are 172 guard band subcarriers on the left side and the right side of the spectrum. Segment i uses preamble carrier-set i , where $i = 0, 1, 2$. The DC subcarrier is not modulated at all and the appropriate PN is discarded. That is, the DC subcarrier is always zeroed.

The pilots in downlink preamble shall be modulated as

$$\begin{aligned}\Re\{PreamblePilotsModulated\} &= 4 \cdot \sqrt{2} \cdot \left(\frac{1}{2} - w_k\right), \\ \Im\{PreamblePilotsModulated\} &= 0.\end{aligned}\tag{2.21}$$

2.5.3 Subcarrier Allocations

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. The pilot tones are allocated first; what remains are data subcarriers, which are divided into subchannels that are used exclusively for data. Pilots and data carriers are allocated within each cluster.

Figure 2.14 shows the cluster structure with subcarriers from left to right in order of increasing subcarrier index. For the purpose of determining PUSC pilot location, even and odd symbols are counted from the beginning of the current zone. The first symbol in the zone is even. The preamble is not counted as part of the first zone. Table 2.2 summarizes the parameters of the OFDMA PUSC symbol structure.

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

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

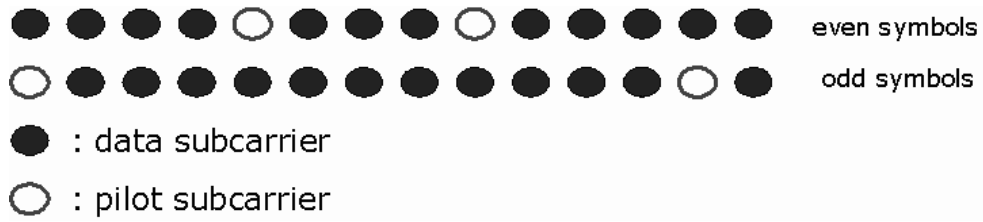


Figure 2.14: Cluster structure (from [7]).

Table 2.2: OFDMA Downlink Subcarrier Allocation Under PUSC [6], [7]

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

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

$$\begin{aligned}
 & \textit{LogicalCluster} \\
 & = \begin{cases} \textit{RenumberingSequence}(\textit{PhysicalCluster}), & \text{first DL zone,} \\ \textit{RenumberingSequence}((\textit{PhysicalCluster} + \\ \quad 13 \cdot \textit{DL_PermBase}) \bmod N_{\textit{clusters}}), & \text{otherwise.} \end{cases}
 \end{aligned}$$

3) Divids the clusters into six major groups. Group 0 includes clusters 0–11, group 1 clusters 12–19, group 2 clusters 20–31, group 3 clusters 32–39, group 4 clusters 40–51 and group 5 clusters 52–59. These groups may be allocated to segments. If a segment is being used, then at least one group shall be allocated to it. (By default group 0 is allocated to segment 0, group 2 to segment 1, and group 4 to segment 2.)

4) Allocate subcarriers to subchannel in each major group separately for each OFDMA symbol by first allocating the pilot subcarriers within each cluster and then taking all remaining data subcarriers within the symbol. The exact partitioning into subchannels is according to the equation below, called a permutation formula:

$$\textit{subcarrier}(k, s) = N_{\textit{subchannels}} \cdot n_k + \{p_s[n_k \bmod N_{\textit{subchannels}}] + \textit{DL_PermBase}\} \bmod N_{\textit{subchannels}}$$

where:

- $\textit{subcarrier}(k, s)$ is the subcarrier index of subcarrier k in subchannel s ,
- s is the index number of a subchannel, from the set $[0..N_{\textit{subchannels}} - 1]$,
- $n_k = (k + 13 \cdot s) \bmod N_{\textit{subcarriers}}$, where k is the subcarrier-in-subchannel index from the set $[0..N_{\textit{subcarriers}} - 1]$,
- $N_{\textit{subchannels}}$ 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, and
- $DL_PermBase$ is an integer from 0 to 31.

2.5.4 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 to produce a sequence, w_k .

Each pilot shall be transmitted with a boosting of 2.5 dB over the average non-boosted power of each data tone. That is, the pilot subcarriers shall be modulated according to

$$\Re\{c_k\} = \frac{8}{3} \left(\frac{1}{2} - w_k \right), \quad \Im\{c_k\} = 0. \quad (2.22)$$

2.5.5 Data Modulation

Downlink transmission also employs the modulations shown in Fig. 2.11. gray-mapped QPSK and Gray-mapped 16QAM shall be supported, whereas the support of 64QAM (also Gray-mapped) is optional.

2.6 Space-Time Coding in IEEE 802.16e OFDMA

This section briefly introduces the space-time coding of IEEE 802.16e. The material is mainly taken from [6] and [7].

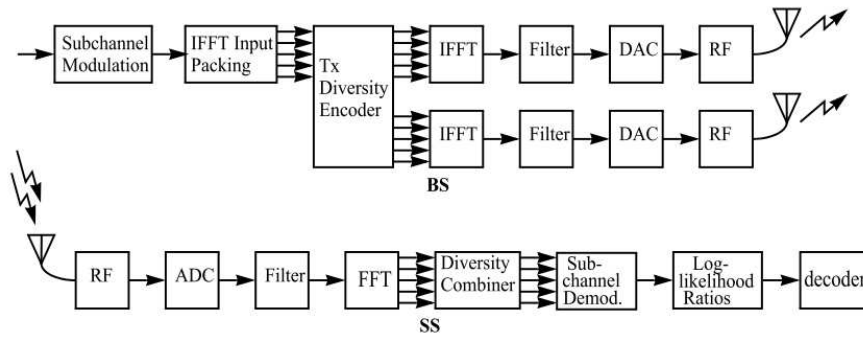


Figure 2.15: Illustration of STC (from [7]).

2.6.1 STC Using Two Antennas

STC (in some cases also termed STTD) or FHDC may be used on the DL to provide higher order (space) Tx diversity. Consider using two Tx antennas on the BS side and one reception antenna on the SS side. This scheme requires multiple-input single-output channel estimation. Decoding is very similar to maximum ratio combining.

Figure 2.15 shows Tx diversity insertion into the OFDMA chain. Each Tx antenna has its own OFDMA chain, but they have the same local oscillator for synchronization purposes. The two antennas transmit two different OFDMA data symbols in the same time. Time domain (space-time) or frequency domain (space-frequency) repetition is used.

2.6.2 STC/FHDC Configurations

Two transmission formats are allowed for the two-antenna configuration, each having its own capacity-diversity tradeoffs. The following matrices define the transmission formats with the row index indicating the antenna number and column index indicating the OFDMA symbol. The entries define the transmission from a subchannel used for this transmission configuration (the same operation is repeated for all subchannels used in this format). Transmission format

A uses matrix A (space time coding rate = 1):

$$A = \begin{bmatrix} S_1 & -(S_2)^* \\ S_2 & (S_1)^* \end{bmatrix}, \quad (2.23)$$

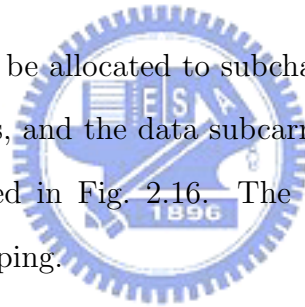
whereas transmission format B uses matrix B (space time coding rate = 2):

$$B = \begin{bmatrix} S_1 \\ S_2 \end{bmatrix}. \quad (2.24)$$

2.6.3 Uplink Using STC

A user supporting transmission using STC configuration in the UL shall use a modified UL tile. The 2-Tx diversity data (STTD mode) or 2-Tx spatial multiplexing (SM mode) data can be mapped onto each subcarrier. The mandatory tile shall be modified to accommodate these configurations.

In STTD mode, the tiles shall be allocated to subchannels. The pilots in each tile shall be split between the two antennas, and the data subcarriers shall be encoded in pairs after constellation mapping, as depicted in Fig. 2.16. The data subcarriers transmitted from antenna 0 follow the original mapping.



2.6.4 STC Using Two Antennas in Downlink PUSC

In PUSC, the data allocation to cluster is changed to accommodate two antennas transmission with the same estimation capabilities, in which each cluster shall be transmitted twice from each antenna. Figure 2.14 is replaced by Figure 2.17 in the definition of PUSC permutation when STC is enabled. The pilot locations change in period of 4 symbols.

Symbols are counted from the beginning of the current zone. The first symbol in the zone is even. STC encoding is done on each pair of symbols $2n, 2n + 1$ ($n = 0, 1, \dots$).

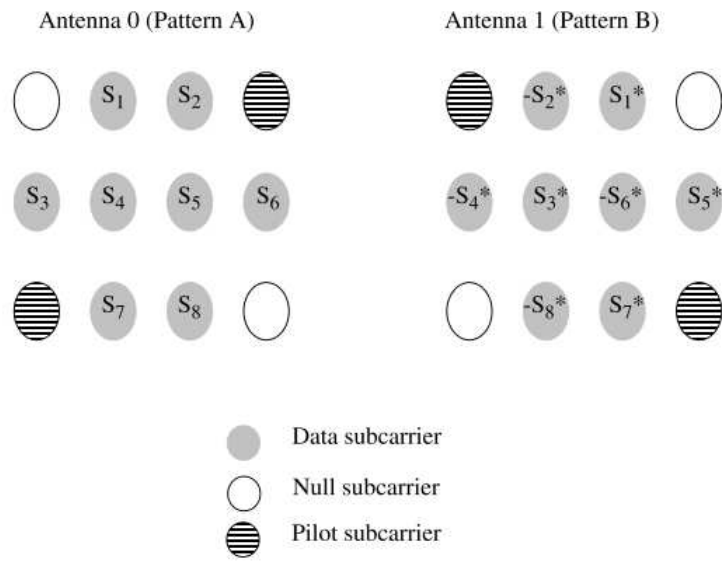


Figure 2.16: Mapping of data subcarriers in STTD mode (from [7]).

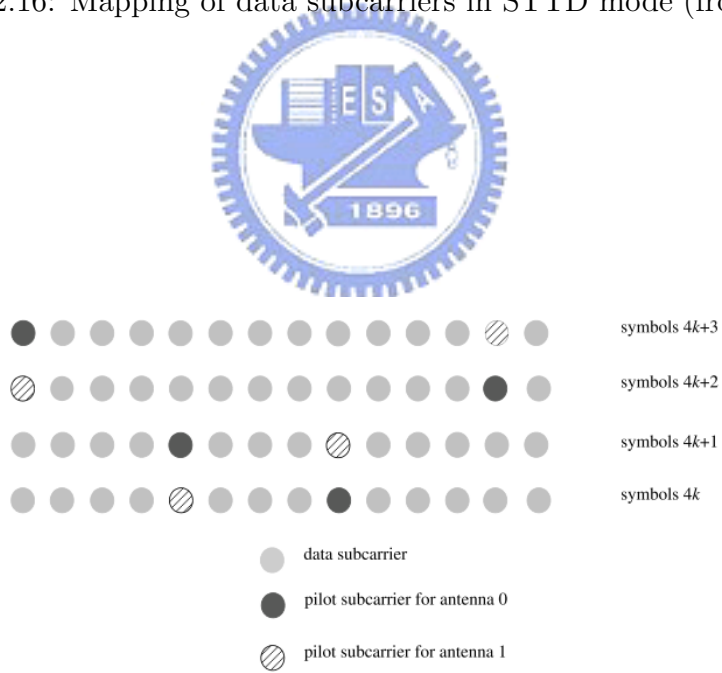


Figure 2.17: Cluster structure for STC PUSC using two antennas (from [7]).

Chapter 3

Channel Estimation Techniques

In this chapter, we discuss some channel estimation methods.

3.1 Pilot-Symbol-Aided Channel Estimation [9]

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 this type of schemes, pilot information has to be transmitted regularly to mitigate error propagation.

3.1.1 The Least-Squares (LS) Estimator [10]

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 :

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

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 $N \times N$ diagonal matrix whose diagonal values are 0 except at pilot locations m_i where $i = 0, \dots, N_p - 1$:

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

Therefore, (3.1) can be rewritten as

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

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

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

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

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

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 desirable.

3.1.2 The LMMSE Estimator [11]

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

[11]. The linear minimum mean-square error channel estimator tries to minimize the mean squared error between the actual and the estimated channels, the latter obtained by a linear transformation applied to $\hat{\mathbf{H}}_{\mathbf{p},\text{LS}}$. The mathematical representation of the LMMSE estimator on pilot signals is

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

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

$$\mathbf{R}_{H_p H_p,LS} = E\{\mathbf{H}_p \mathbf{H}_{p,LS}^H\}, \quad (3.7)$$

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

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

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 [12].

3.2 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.

3.2.1 Linear Interpolation

After obtaining the channel response estimate at the pilot subcarriers, one may use interpolation to obtain the response at the rest of the subcarriers. Linear interpolation is a commonly

considered scheme due to its low complexity. It does the interpolation between two known data. That is, we use the channel information at two pilots obtained by the LS estimator to estimate the channel frequency response information at the data subcarriers between them.

The channel estimates at data subcarrier k , $mL < k < (m+1)L$, using linear interpolation is given by [13]

$$H_e(k) = H_e(m+l) = (H_p(m+1) - H_p(m))\frac{l}{L} + H_p(m) \quad (3.10)$$

where $H_p(k)$, $k = 0, 1, \dots, N_p$, are the channel frequency responses at pilot subcarriers, L is the pilot subcarriers spacing, and $0 < l < L$. In two-dimensional channel estimation, to suit the tile structure, we first interpolate in the time domain and then in the frequency domain.

3.2.2 2-D Wiener Filter [14]

The Wiener filter is the optimum (in the sense of minimum mean-squared error) linear filter or smoother or predictor, if the noise is additive.

Assume we have transmitted pilot signal vector \mathbf{p} and received pilot signal vector $\hat{\mathbf{p}}$ containing noise. We want to find an estimate \hat{h} of the channel response h as a linear combination of $\hat{\mathbf{p}}$. That means we want to find \mathbf{w} that makes $J(\mathbf{w})$ minimum, where \mathbf{w} and $J(\mathbf{w})$ are defined as

$$\hat{h} = \mathbf{w}^T \hat{\mathbf{p}}, \quad J(\mathbf{w}) = E \left[|h - \hat{h}|^2 \right]. \quad (3.11)$$

By applying the orthogonal projection theorem, we get

$$\mathbf{w} = \underline{\theta}^T \Phi^{-1} \quad (3.12)$$

where $\underline{\theta}$ is the cross-covariance vector between \mathbf{p} and h , and Φ is the auto-covariance matrix between pilots.

Let k and l be the subcarrier number and the OFDM symbol number, respectively. The correlation values may be assumed as given by [15]

$$E [h_{k,l}\hat{p}_{k',l'}^*] = r_f(k - k')r_t(l - l') \quad (3.13)$$

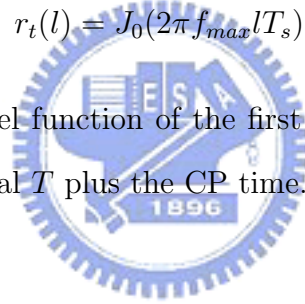
where $r_t(l)$ and $r_f(k)$ are the correlation functions in time and frequency, respectively. For an exponentially decaying multipath power delay profile,

$$r_f(k) = \frac{1}{1 + j2\pi\tau_{rms}k/T} \quad (3.14)$$

where $1/T$ is the subcarrier spacing, which is the inverse of the FFT interval T . For a time-fading signal with a maximum Doppler frequency f_{max} and a Clarke-Gans spectrum, the time correlation function $r_t(l)$ is given by

$$r_t(l) = J_0(2\pi f_{max}lT_s) \quad (3.15)$$

where J_0 is the zeroth order Bessel function of the first kind and T_s is the OFDM symbol duration, which is the FFT interval T plus the CP time.



Chapter 4

Simulation of STC Uplink Channel Estimation

In this chapter we will simulate two different channel estimation methods for the in IEEE 802.16e OFDMA uplink system. One is linear interpolation and the other is Wiener filter. We evaluate the performance of both methods mainly by observing the mean square error (MSE) and the symbol error rate (SER).

4.1 Linear Interpolation

As described in chapter 2, the uplink transmission uses a tile structure to transmit pilot and data information. In the STC mode, one tile only contains two pilots. So we use tile $(N - 1)$ and tile $(N + 1)$ to interpolate the channel response of tile N , as shown in Fig. 4.1, to yield enough references for linear interpolation in frequency. Within three successive three tiles, we first estimate the channel response at each pilot position. Then we interpolate for the frequency response at each data subcarrier from the estimated pilot response in time domain. Lastly, we get the frequency response of the whole tile by interpolating the frequency response in the frequency domain.

The detailed steps for channel estimation are as follows:

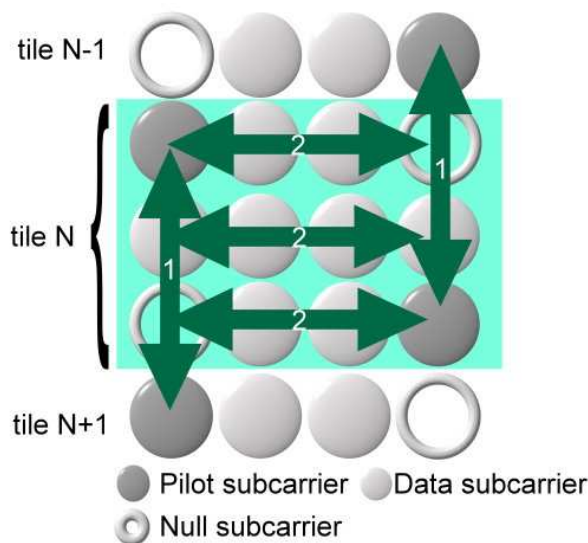


Figure 4.1: Linear interpolation in STTD mode at antenna 0.

- Estimate the channel response at each pilot location by using the LS technique.
- Use linear interpolation in the time dimension to get some data subcarrier responses (marked 1 in Fig. 4.1).
- Estimate the channel responses of the remaining subcarriers in a tile by frequency domain interpolation (marked 2 in Fig. 4.1).

4.2 Wiener Filtering

For two-dimensional Wiener filtering, we also choose three contiguous tiles to do channel estimation as depicted in Fig. 4.2. To do Wiener filtering, we have to know two parameters: autocorrelation of channel responses Φ at pilots and cross-correlation $\underline{\theta}$ of channel response at data subcarriers and pilots. In the uplink, one subchannel contains six tiles. Thus we use the average over six tiles in the same subchannel to calculate Φ . That means if pilot subcarrier P_k 's channel response is p_k and the estimate is \hat{p}_k , where $\hat{p}_k = p_k + n_k$ with n_k

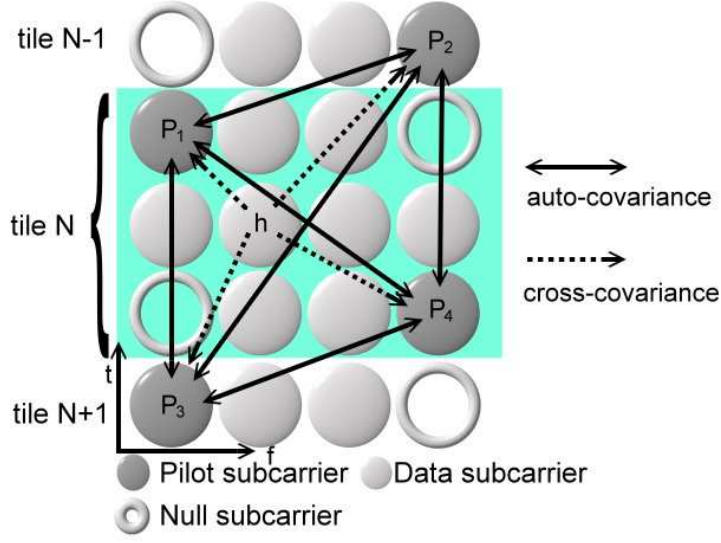


Figure 4.2: Wiener filtering in STTD mode at Antenna 0.

being additive white Gaussian noise (AWGN), then we estimate Φ by

$$\Phi = E \begin{bmatrix} \hat{p}_1 \hat{p}_1^* & \hat{p}_1 \hat{p}_2^* & \hat{p}_1 \hat{p}_3^* & \hat{p}_1 \hat{p}_4^* \\ \hat{p}_2 \hat{p}_1^* & \hat{p}_2 \hat{p}_2^* & \hat{p}_2 \hat{p}_3^* & \hat{p}_2 \hat{p}_4^* \\ \hat{p}_3 \hat{p}_1^* & \hat{p}_3 \hat{p}_2^* & \hat{p}_3 \hat{p}_3^* & \hat{p}_3 \hat{p}_4^* \\ \hat{p}_4 \hat{p}_1^* & \hat{p}_4 \hat{p}_2^* & \hat{p}_4 \hat{p}_3^* & \hat{p}_4 \hat{p}_4^* \end{bmatrix} \approx \frac{1}{6} \sum_{\text{one subchannel}} \begin{bmatrix} \hat{p}_1 \hat{p}_1^* & \hat{p}_1 \hat{p}_2^* & \hat{p}_1 \hat{p}_3^* & \hat{p}_1 \hat{p}_4^* \\ \hat{p}_2 \hat{p}_1^* & \hat{p}_2 \hat{p}_2^* & \hat{p}_2 \hat{p}_3^* & \hat{p}_2 \hat{p}_4^* \\ \hat{p}_3 \hat{p}_1^* & \hat{p}_3 \hat{p}_2^* & \hat{p}_3 \hat{p}_3^* & \hat{p}_3 \hat{p}_4^* \\ \hat{p}_4 \hat{p}_1^* & \hat{p}_4 \hat{p}_2^* & \hat{p}_4 \hat{p}_3^* & \hat{p}_4 \hat{p}_4^* \end{bmatrix}. \quad (4.1)$$

To calculate $\underline{\theta}$, we first assume the channel response at data subcarrier h_k is the 2-D linear interpolation of four nearest pilot channel responses [16] as

$$h_k = \sum_{i=0}^3 \alpha_i p_i \quad (4.2)$$

where α_i are the linear interpolation weights. There are eight data subcarriers in a tile. We list the α_i of different data subcarrier in Table 4.1.

Since we can only get \hat{p}_k , so we use \hat{p}_k in place of p_k , yielding

Table 4.1: α_i of Eight Data Subcarriers at Antenna 0.

Carrier	α_1	α_2	α_3	α_4
1	2/3	2/9	0	1/9
2	1/3	4/9	0	2/9
3	2/3	0	1/3	0
4	4/9	1/9	2/9	2/9
5	2/9	2/9	1/9	4/9
6	0	1/3	0	2/3
7	2/9	0	4/9	1/3
8	1/9	0	2/9	2/3

$$\begin{aligned}
 E \left[\left(\sum_{i=0}^3 \alpha_i \hat{p}_i \right) \hat{p}_k^* \right] &= E \left[\left(\sum_{i=0}^3 \alpha_i p_i + \sum_{i=0}^3 \alpha_i p_i n_i \right) p_k^* + n_k^* \right] \\
 &= E \left[\left(\sum_{i=0}^3 \alpha_i p_i \right) p_k^* \right] + \alpha_k \sigma_0^2
 \end{aligned} \tag{4.3}$$

where n_i denotes the noise at pilot i and σ_0^2 is the variance of the white Gaussian noise. Here

$$E [h_j \hat{p}_k^*] = E \left[\left(\sum_{i=0}^3 \alpha_i \hat{p}_i \right) \hat{p}_k^* \right] - \alpha_k \sigma_0^2. \tag{4.4}$$

As a result, the cross-correlation vector $\underline{\theta}_k$ is given by

$$\begin{aligned}
 \underline{\theta}_k &= E \left[h_k \hat{p}_1^* \quad h_k \hat{p}_2^* \quad h_k \hat{p}_3^* \quad h_k \hat{p}_4^* \right] \\
 &= E \left[\left(\sum_{i=0}^3 \alpha_i \hat{p}_i \right) \hat{p}_1^* \quad \left(\sum_{i=0}^3 \alpha_i \hat{p}_i \right) \hat{p}_2^* \quad \left(\sum_{i=0}^3 \alpha_i \hat{p}_i \right) \hat{p}_3^* \quad \left(\sum_{i=0}^3 \alpha_i \hat{p}_i \right) \hat{p}_4^* \right] \\
 &\quad - \left[\alpha_0 \sigma_0^2 \quad \alpha_1 \sigma_0^2 \quad \alpha_2 \sigma_0^2 \quad \alpha_3 \sigma_0^2 \right].
 \end{aligned} \tag{4.5}$$

Using the average over the six tiles of one subchannel to approximate the expectation operation, we have

$$\begin{aligned}
 \underline{\theta}_k &\approx \frac{1}{6} \left(\sum_{\text{one subchannel}} \left[\left(\sum_{i=0}^3 \alpha_i \hat{p}_i \right) \hat{p}_1^* \quad \left(\sum_{i=0}^3 \alpha_i \hat{p}_i \right) \hat{p}_2^* \quad \left(\sum_{i=0}^3 \alpha_i \hat{p}_i \right) \hat{p}_3^* \quad \left(\sum_{i=0}^3 \alpha_i \hat{p}_i \right) \hat{p}_4^* \right] \right) \\
 &\quad - \left[\alpha_0 \sigma_0^2 \quad \alpha_1 \sigma_0^2 \quad \alpha_2 \sigma_0^2 \quad \alpha_3 \sigma_0^2 \right]
 \end{aligned} \tag{4.6}$$

where we can estimate σ_0^2 by the power of the subcarriers in the guard band and those that have a null value.

4.3 STC Decoding

After we have estimated the channel response, we can decode the STC, where the decoding method has been introduced before. If the channel responses are the same in neighboring subchannels as shown in Figure 4.3(a), then the received signals r_1 and r_2 are given by in, absence of noise,

$$r_1 = S_1 h_0 - S_2^* h_1, \quad r_2 = S_2 h_0 + S_1^* h_1. \quad (4.7)$$

Then we can decode the received signal as

$$\begin{aligned} S_1 &= (r_1 h_0^* + r_2^* h_1) / (|h_0|^2 + |h_1|^2), \\ S_2 &= (r_2 h_0^* - r_1^* h_1) / (|h_0|^2 + |h_1|^2). \end{aligned} \quad (4.8)$$

But in a real channel, the neighboring channel responses may not be the same, especially when in high mobile speed, as shown in Figure 4.3(b),

$$r_1 = S_1 h_0 - S_2^* h_2, \quad r_2 = S_2 h_1 + S_1^* h_3. \quad (4.9)$$

So we decode by calculating

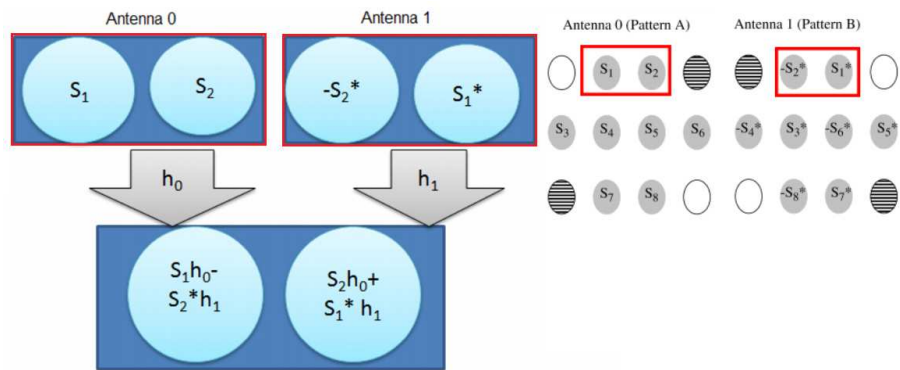
$$\begin{aligned} S_1 &= (r_1 h_1^* + r_2^* h_2) / (h_0 h_1^* + h_2 h_3^*), \\ S_1 &= (r_2 h_0^* - r_1^* h_3) / (h_2 h_3 + h_0^* h_1). \end{aligned} \quad (4.10)$$

4.4 Simulation Conditions

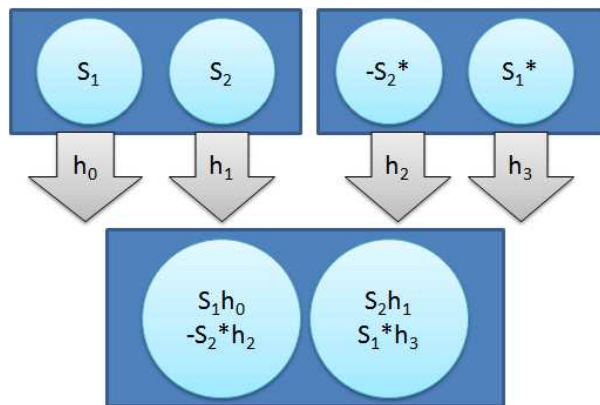
This section gives the system parameters and introduces the channel models used in our simulation work.

4.4.1 OFDMA Uplink System Parameters

In chapter 2, we introduced the primitive and the derived parameters of the system. The system parameters used in our simulation are listed in Table 4.2.



(a)



(b)

Figure 4.3: STTD transmission. (a) Neighboring channel responses are the same. (b) Responses are different.

4.4.2 Channel Models

We consider the following channel models: AWGN, single-path Rayleigh, SUI, and ETSI Vehicular A.

Erceg *et al.* [17] published a total of 6 different radio channel models for type G2 (i.e., LOS and NLOS) MMDS BWA systems in three terrain categories. The three types in suburban area are:

- A: hilly terrain, heavy tree,

Table 4.2: OFDMA Uplink Parameters

Parameters	Values
Bandwidth	10 MHz
Carrier frequency	3.5 GHz
N_{FFT}	1024
N_{used}	841
Sampling factor n	28/25
G	1/8
Sampling frequency	11.2 MHz
Subcarrier spacing	10.94 kHz
Useful symbol time	91.43 μ s
CP time	11.43 μ s
OFDMA symbol time	102.86 μ s
Sampling time	89.29 ns

- C: flat terrain, light tree, and
- B: between A and C.

The correspondence with the so-called SUI channels is as follows:

- C: SUI-1, SUI-2,
- B: SUI-3, SUI-4, and
- A: SUI-5, SUI-6.



In the above, SUI-1 and SUI-2 are Ricean multipath channels, whereas the other four are Rayleigh multipath channels [18]. The Rayleigh channels are more hostile and exhibit a greater RMS delay spread. And the SUI-2 represents a worst case link for terrain type C. We employ the SUI-2 and SUI-3 models in our simulation, but we use Rayleigh fading to model all the paths in these channels. The channel characteristics are as shown in Table 4.3.

We also employ the ETSI Vehicular A model [19]. The model's power delay profile is as shown in Table 4.4. The mean delay and the RMS delay are shown in Table 4.5.

Table 4.3: Channel Profiles of SUI-2 and SUI-3 [17]

SUI – 2 Channel				
	Tap 1	Tap 2	Tap 3	Units
Delay	0	0.4	1.1	μs
Power (omni ant.)	0	-12	-15	dB
90% K-fact. (omni)	2	0	0	
75% K-fact. (omni)	11	0	0	
Power (30° ant.)	0	-18	-27	dB
90% K-fact. (30°)	8	0	0	
75% K-fact. (30°)	36	0	0	
Doppler	0.2	0.15	0.25	Hz
Antenna Correlation:		$\rho_{ENV} = 0.5$		Terrain Type: C
Gain Reduction Factor:		GRF = 2 dB		Omni antenna: $\tau_{RMS} = 0.202 \mu\text{s}$,
Normalization Factor:		$F_{omni} = -0.3930 \text{ dB}$, $F_{30^\circ} = -0.0768 \text{ dB}$		overall K: K = 1.6 (90%); K = 5.1 (75%)
				30° antenna: $\tau_{RMS} = 0.069 \mu\text{s}$,
				overall K: K = 6.9 (90%); K = 21.8 (75%)

SUI – 3 Channel				
	Tap 1	Tap 2	Tap 3	Units
Delay	0	0.4	0.9	μs
Power (omni ant.)	0	-5	-10	dB
90% K-fact. (omni)	1	0	0	
75% K-fact. (omni)	7	0	0	
Power (30° ant.)	0	-11	-22	dB
90% K-fact. (30°)	3	0	0	
75% K-fact. (30°)	19	0	0	
Doppler	0.4	0.3	0.5	Hz
Antenna Correlation:		$\rho_{ENV} = 0.4$		Terrain Type: B
Gain Reduction Factor:		GRF = 3 dB		Omni antenna: $\tau_{RMS} = 0.264 \mu\text{s}$,
Normalization Factor:		$F_{omni} = -1.5113 \text{ dB}$, $F_{30^\circ} = -0.3573 \text{ dB}$		overall K: K = 0.5 (90%); K = 1.6 (75%)
				30° antenna: $\tau_{RMS} = 0.123 \mu\text{s}$,
				overall K: K = 2.2 (90%); K = 7.0 (75%)

4.5 Simulation Results

4.5.1 Simulation Flow

Figure 5.3 illustrates our simulated system. We assume perfect synchronization. After channel estimation, we calculate the MSE between the real channel and the estimated one,

Table 4.4: Power-Delay Profile of the ETSI Vehicular A Channel

Tap	Relative Delay (μs)	Average Power (dB)
1	0	0
2	0.31	-1.0
3	0.71	-9.0
4	1.09	-10.0
5	1.73	-15.0
6	2.51	-20.0

Table 4.5: Mean Delay and RMS Delay Spread

Channel	Mean Delay (μs)	RMS Delay Spread (μs)
SUI-2	0.0027 (0.0302 samples)	0.0428 (0.4793 samples)
SUI-3	0.0413 (0.4626 samples)	0.1318 (1.4762 samples)
Vehicular A	0.1325 (1.4840 samples)	0.1821 (2.0395 samples)

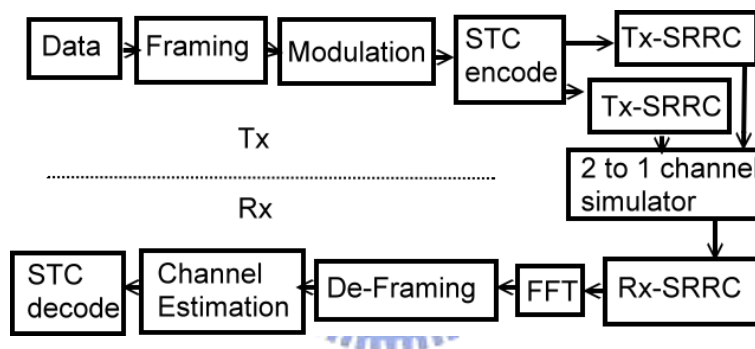


Figure 4.4: Block diagram of the simulated system.

where the average is taken over the subcarriers. The symbol error rate (SER) can also be obtained after demapping.

4.5.2 Validation of Simulation Model

Before considering multipath channels, we do simulation with an AWGN channel to validate the simulation model. We validate the model by comparing theoretical SER curves and the SER curves resulting from simulations, where we use C/C++ programming language and

TI's Code Composer Studio (CCS).

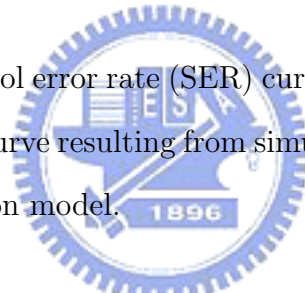
For an even number of bits per symbol, the SER of rectangular QAM is given by

$$P_s = 4 \left(1 - \frac{1}{\sqrt{M}} \right) Q \left(\sqrt{\frac{3}{M-1} \frac{E_s}{N_0}} \right) \quad (4.11)$$

where

- M = number of symbols in modulation constellation; for example, $M = 4$ for QPSK, $M = 16$ for 16QAM and $M = 64$ for 64QAM,
- E_s = average symbol energy,
- N_0 = noise power spectral density (W/Hz), and
- $Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty e^{-t^2/2} dt$, $x \geq 0$.

In Figure 4.5, the theoretical symbol error rate (SER) curve versus E_s/N_0 for uncoded QPSK is plotted together with the SER curve resulting from simulation under no channel estimation error. This validates the simulation model.



4.5.3 Simulation Results and Analysis

For verification of the simulation results, note that the theoretical MSE for linear interpolation in AWGN is given by

$$\begin{aligned} MSE &= E[|h - \hat{h}|^2] \\ &= E[|h - \sum_{i=0}^3 \alpha_i(p_i + n_i)|^2] \\ &= E[|\sum_{i=0}^3 \alpha_i n_i|^2] \\ &= \sum_{i=0}^3 \alpha_i^2 \sigma_0^2. \end{aligned} \quad (4.12)$$

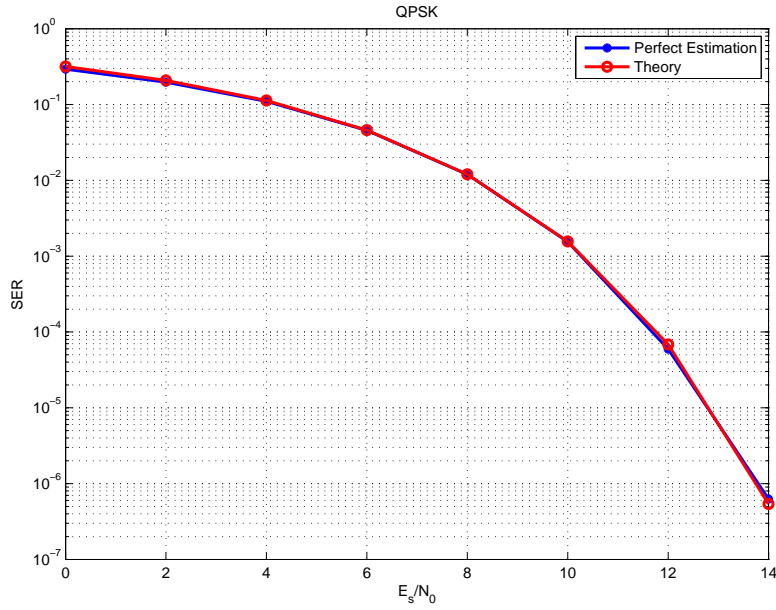


Figure 4.5: The SER curve for uncoded QPSK resulting from simulation matches the theoretical one.

Table 4.6: MSE of Eight Data Subcarriers at Antennas 0 and 1

Carrier Index, Antenna 0	1,8	2,7	3,6	4,5
Carrier Index, Antenna 1	2,7	1,8	4,5	3,6
MSE	$\frac{41}{81}\sigma_0^2$	$\frac{29}{81}\sigma_0^2$	$\frac{5}{9}\sigma_0^2$	$\frac{25}{81}\sigma_0^2$

Note also that, the MSE is different at different data subcarriers. The MSE at the eight data subcarriers in a tile are listed in Table 4.6.

The theoretical MSE and the simulation result are shown in Figure 4.6.

If we regard MSE as noise, then the equivalent SNR would be $SNR/(1 + MSE)$. So we can get theoretical SER under AWGN channel for QPSK as

$$SER = 2Q \left(\sqrt{\frac{E_s}{1.4321N_0}} \right). \quad (4.13)$$

The result is as shown in Figure 4.7. Since STC using two data subcarriers to decode. The influence of MSE on two neighbor subcarriers would be average.

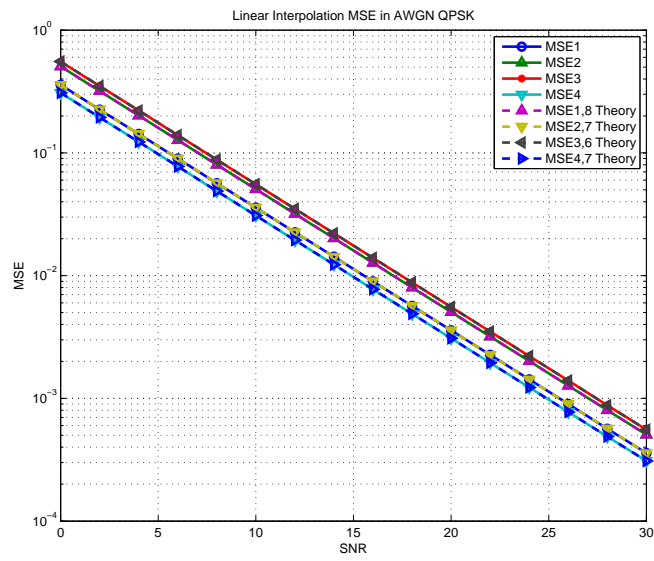


Figure 4.6: MSE performance for uncoded QPSK resulting with linear interpolation, antenna 0.

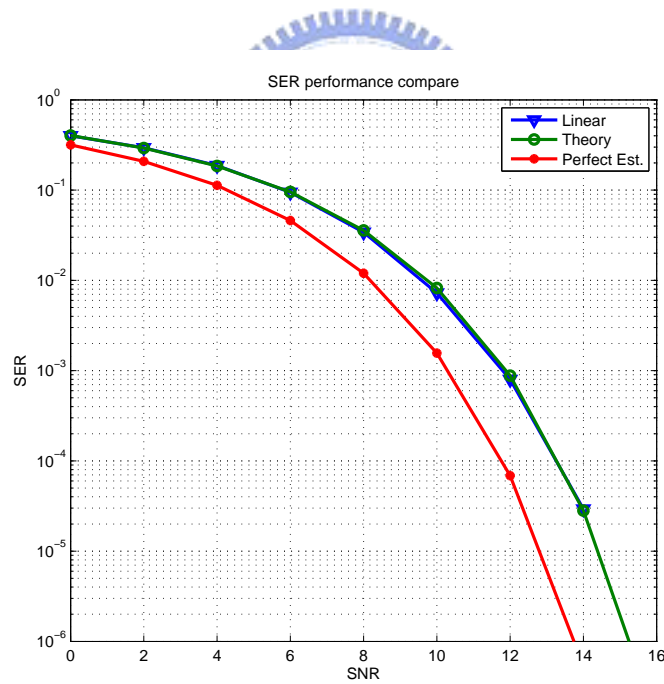


Figure 4.7: SER performance for uncoded QPSK resulting from linear interpolation.

Now we derive the theoretical MSE for Wiener filtering in AWGN channel. Suppose we know the autocorrelation Φ and the cross-correlation $\underline{\theta}$. Since the channel response $h = 1$ in AWGN and the noise power is σ_0^2 , we get

$$\Phi = \begin{bmatrix} 1 + \sigma_0^2 & 1 & 1 & 1 \\ 1 & 1 + \sigma_0^2 & 1 & 1 \\ 1 & 1 & 1 + \sigma_0^2 & 1 \\ 1 & 1 & 1 & 1 + \sigma_0^2 \end{bmatrix}, \quad (4.14)$$

$$\underline{\theta} = [1 \ 1 \ 1 \ 1]. \quad (4.15)$$

The received pilot vector \mathbf{p} , containing noise, is given by

$$\mathbf{p}^T = [1 + n_0 \ 1 + n_1 \ 1 + n_2 \ 1 + n_3]. \quad (4.16)$$

Hence the estimation is given by

$$\begin{aligned} \hat{h} &= [1 \ 1 \ 1 \ 1] \begin{bmatrix} 1 + \sigma_0^2 & 1 & 1 & 1 \\ 1 & 1 + \sigma_0^2 & 1 & 1 \\ 1 & 1 & 1 + \sigma_0^2 & 1 \\ 1 & 1 & 1 & 1 + \sigma_0^2 \end{bmatrix}^{-1} \begin{bmatrix} 1 + n_0 \\ 1 + n_1 \\ 1 + n_2 \\ 1 + n_3 \end{bmatrix} \\ &= \frac{4 + n_0 + n_1 + n_2 + n_3}{4 + \sigma_0^2}. \end{aligned} \quad (4.17)$$

The theoretical MSE in AWGN channel is thus

$$\begin{aligned} MSE &= E [|h - \hat{h}|^2] \\ &= E \left[\left| 1 - \frac{4 + n_0 + n_1 + n_2 + n_3}{4 + \sigma_0^2} \right|^2 \right] \\ &= \frac{\sigma_0^4 + 4\sigma_0^2}{(4 + \sigma_0^2)^2}. \end{aligned} \quad (4.18)$$

But actually we do not know the autocorrelation and the cross-correlation. In our calculation, we sum the six tiles (one subchannel) instead to estimate the autocorrelation and, further, use linear interpolation to approximate the channel response at the data subcarrier

locations to estimate the cross-correlation over the six tiles. These methods cause errors. The true cross-correlation is

$$\underline{\theta} = E \left[\begin{array}{cccc} hp_1^* & hp_2^* & hp_3^* & hp_4^* \end{array} \right]. \quad (4.19)$$

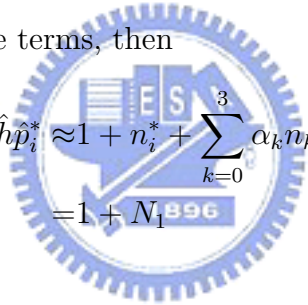
We use \hat{h} in place of h , where \hat{h} is given by

$$\hat{h} = \sum_{i=0}^3 \alpha_i \hat{p}_i = \sum_{i=0}^3 \alpha_i (1 + n_i). \quad (4.20)$$

The estimated cross-correlation is then equal to, for one tile,

$$\begin{aligned} \hat{h} \hat{p}_i^* &= \left(\sum_{k=0}^3 \alpha_k (1 + n_k) \right) (1 + n_i)^* \\ &= 1 + n_i^* + \sum_{k=0}^3 \alpha_k n_k + \sum_{k=0}^3 \alpha_k n_k n_i^*. \end{aligned} \quad (4.21)$$

If we ignore the second-order noise terms, then

$$\begin{aligned} \hat{h} \hat{p}_i^* &\approx 1 + n_i^* + \sum_{k=0}^3 \alpha_k n_k \\ &= 1 + N_1 \end{aligned} \quad (4.22)$$


where

$$\begin{aligned} N_1 &\sim N \left(0, \frac{(1 + \alpha_i)^2 \sigma_0^2 + \left(\sum_{\substack{k=0 \\ k \neq i}}^3 \alpha_k^2 \right) \sigma_0^2}{2} \right) \\ &+ jN \left(0, \frac{(1 - \alpha_i)^2 \sigma_0^2 + \left(\sum_{\substack{k=0 \\ k \neq i}}^3 \alpha_k^2 \right) \sigma_0^2}{2} \right). \end{aligned} \quad (4.23)$$

We add up all the estimates for the six tiles to estimate $\underline{\theta}$, resulting in

$$\underline{\theta}' = \underline{\theta} + \left[\begin{array}{cccc} \delta_0 & \delta_1 & \delta_2 & \delta_3 \end{array} \right] \quad (4.24)$$

where

$$\begin{aligned}
\underline{\theta} &= [6 \ 6 \ 6 \ 6], \\
\delta_i &\sim N \left(0, 3(1 + \alpha_i)^2 \sigma_0^2 + 3 \left(\sum_{\substack{k=0, \\ k \neq i}}^3 \alpha_k^2 \right) \sigma_0^2 \right) \\
&\quad + jN \left(0, 3(1 - \alpha_i)^2 \sigma_0^2 + 3 \left(\sum_{\substack{k=0, \\ k \neq i}}^3 \alpha_k^2 \right) \sigma_0^2 \right). \tag{4.25}
\end{aligned}$$

Similarly, if we ignore the second-order noise terms in the estimation of the autocorrelation matrix Φ , then the estimated quantity containing noise is given by

$$\Phi' = \Phi + \Delta \tag{4.26}$$

where

$$\begin{aligned}
\Phi &= \begin{bmatrix} 6 + 6\sigma_0^2 & 6 & 6 & 6 \\ 6 & 6 + 6\sigma_0^2 & 6 & 6 \\ 6 & 6 & 6 + 6\sigma_0^2 & 6 \\ 6 & 6 & 6 & 6 + 6\sigma_0^2 \end{bmatrix}, \\
\Delta &= \begin{bmatrix} n_{00} & n_{01} & n_{02} & n_{03} \\ n_{10} & n_{11} & n_{12} & n_{13} \\ n_{20} & n_{21} & n_{22} & n_{23} \\ n_{30} & n_{31} & n_{32} & n_{33} \end{bmatrix}, \tag{4.27}
\end{aligned}$$

with

$$n_{ij} = \sum_{6 \text{ tiles}} (n_i + n_j^*). \tag{4.28}$$

The channel estimate by Wiener filtering using estimated autocorrelation and cross-correlation is given by

$$\begin{aligned}
\hat{h} &= \underline{\theta}' \Phi'^{-1} \begin{bmatrix} \hat{p}_1 \\ \hat{p}_2 \\ \hat{p}_3 \\ \hat{p}_4 \end{bmatrix} \\
&\approx \underline{\theta} \Phi^{-1} \begin{bmatrix} \hat{p}_1 \\ \hat{p}_2 \\ \hat{p}_3 \\ \hat{p}_4 \end{bmatrix} + [\delta_0 \quad \delta_1 \quad \delta_2 \quad \delta_3] \Phi^{-1} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} - \theta \Phi^{-1} \Delta \Phi^{-1} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \\
&= \frac{4 + \sum_{i=0}^3 n_i}{4 + \sigma_0^2} + \frac{\frac{1}{6} \sum_{6tiles} (\sum_{i=0}^3 n_i^* + 4 \sum_{i=0}^3 \alpha_i n_i)}{4 + \sigma_0^2} - \frac{\frac{4}{6} \sum_{6tiles} \sum_{i=0}^3 (n_i + n_i^*)}{(4 + \sigma_0^2)^2} \\
&\approx \frac{4 + N_2 + N_3}{4 + \sigma_0^2} + \frac{N_4}{(4 + \sigma_0^2)^2} \tag{4.29}
\end{aligned}$$

where

$$\begin{aligned}
N_2 &\sim N \left(0, \sum_{i=0}^3 \frac{(1 + \frac{1}{6} + \frac{4}{6} \alpha_i)^2 \sigma_0^2}{2} \right) + jN \left(0, \sum_{i=0}^3 \frac{(1 - \frac{1}{6} + \frac{4}{6} \alpha_i)^2 \sigma_0^2}{2} \right), \\
N_3 &\sim N \left(0, \frac{5}{72} \sum_{i=0}^3 (1 + 4\alpha_i)^2 \sigma_0^2 \right) + jN \left(0, \frac{5}{72} \sum_{i=0}^3 (1 - 4\alpha_i)^2 \sigma_0^2 \right), \\
N_4 &\sim N \left(0, \frac{32}{3} \sigma_0^2 \right). \tag{4.30}
\end{aligned}$$

And the MSE would be

$$\begin{aligned}
MSE &= E \left[|h - \hat{h}|^2 \right] \\
&= \frac{\sigma_0^4 + \text{Var}(N_2) + \text{Var}(N_3)}{(4 + \sigma_0^2)^2} + \frac{(N_4)}{(4 + \sigma_0^2)^4} \\
&= \frac{\sigma_0^4 + \frac{1}{2} \sum_{i=0}^3 \left[(1 + \frac{1}{6} + \frac{4}{6} \alpha_i)^2 + (1 - \frac{1}{6} + \frac{4}{6} \alpha_i)^2 \right] \sigma_0^2}{(4 + \sigma_0^2)^2} \\
&\quad + \frac{\frac{5}{72} \sum_{i=0}^3 \left[(1 + 4\alpha_i)^2 + (1 - 4\alpha_i)^2 \right] \sigma_0^2}{(4 + \sigma_0^2)^2} \\
&\quad + \frac{\frac{32}{3} \sigma_0^2}{(4 + \sigma_0^2)^4}. \tag{4.31}
\end{aligned}$$

The theoretical MSE and the simulation result are shown in Figure 4.8. In the simulation, we use the average over guard band subcarriers (subcarriers 0 through 89 and 933 through

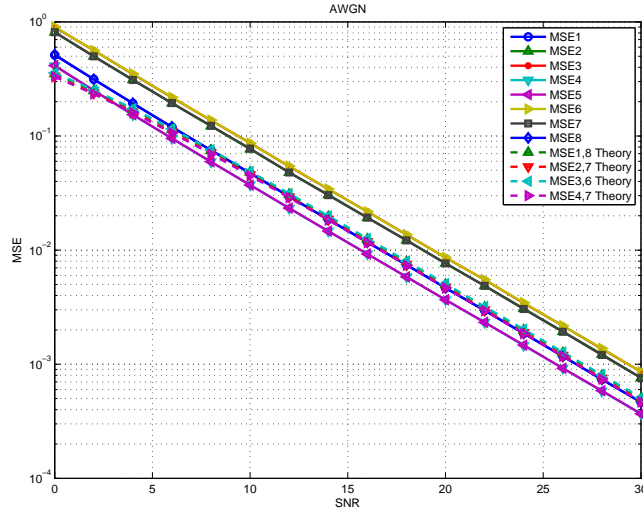


Figure 4.8: MSE performance of Wiener filtering channel estimation for uncoded QPSK, antenna 0. Autocorrelation and cross-correlation are obtained by averaging over one sub-channel.

1023) to estimate the noise power. In the following simulations, each data point is an average over simulation of 420000 tiles and each symbol containing ten subchannel (average over three subchannels use 378000 tiles and nine subchannel instead).

From the simulation result, we can see that the performance of Wiener filtering is worse than linear interpolation if only six tiles are used to estimate the autocorrelation and the cross-correlation. The reason should be due to noise-induced model mismatch as the autocorrelation and the cross-correlation are both calculated from noisy signal. If we use ten subchannels to estimate the autocorrelation and the cross-correlation, then we can get better performance. And the performance is much closer to the theory under known autocorrelation and cross-correlation. Figure 4.9 shows the MSE simulation result where ten subchannels to estimate the autocorrelation and the cross-correlation.

Fig 4.10 and 4.11 shows the SER and MSE performance under channel estimation by linear interpolation and that by Wiener filtering with averages taken over one, three, five

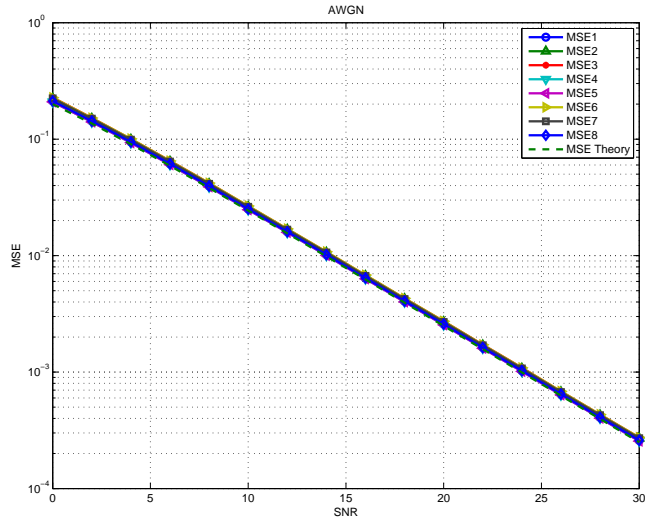


Figure 4.9: MSE performance of Wiener filtering channel estimation for uncoded QPSK, antenna 0. Autocorrelation and cross-correlation are obtained by averaging over ten subchannels.

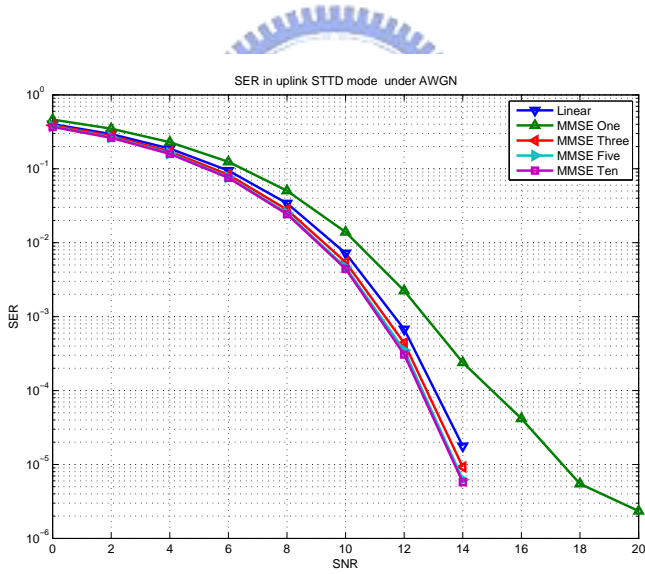


Figure 4.10: Comparison of SER performance with using Wiener filtering and linear interpolation channel estimation in STTD under QPSK modulation in AWGN.

and ten subchannels, separately, in AWGN channel. We can see that if we use only one subchannel to average, the performance of Wiener filter is worse than linear interpolation

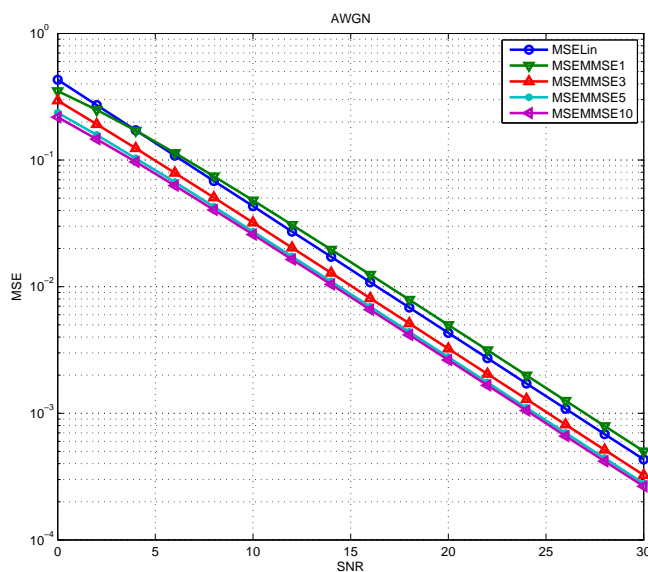


Figure 4.11: Comparison of MSE performance with using Wiener filtering and linear interpolation channel estimation in STTD under QPSK modulation in AWGN.

and if we choose more subchannel to average, the performance is better. The performance of using five subchannel to average is close to using ten subchannels.

In SUI channels, the antenna correlation ρ_{env} is defined as follows: The baseband signals are modeled as two complex random processes $X(t)$ and $Y(t)$ with an envelope correlation coefficient of

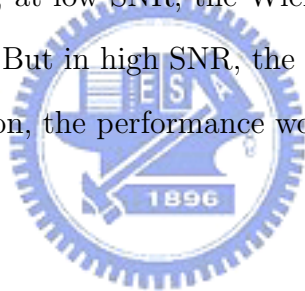
$$\rho_{env} = \left| \frac{E \{ (X - E \{X\}) (Y - E \{Y\})^* \}}{\sqrt{E \{ |X - E \{X\}|^2 \} E \{ |Y - E \{Y\}|^2 \}}} \right|. \quad (4.32)$$

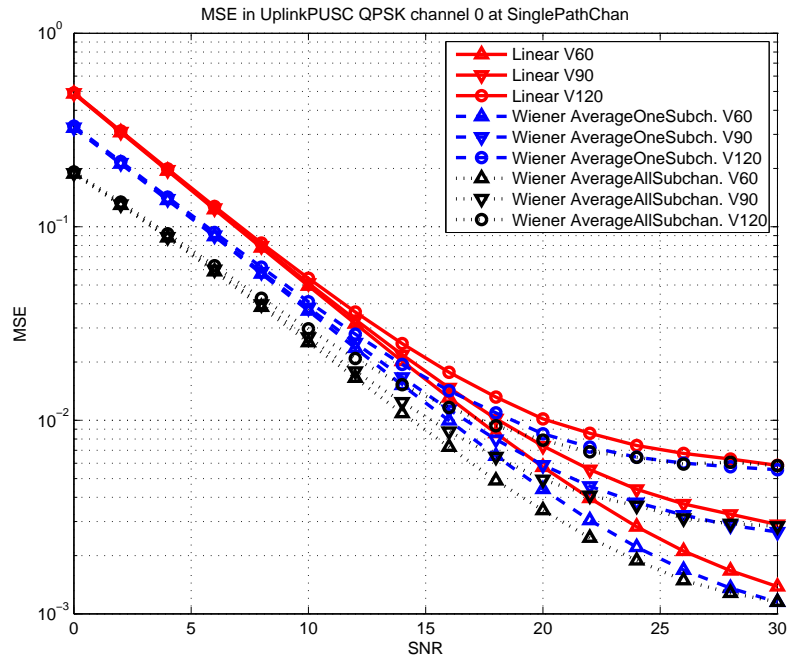
In our simulation, we consider to two different cases, one with correlation equal to zero and the other with nonzero antenna correlation. We can see that in 2-Tx transmission with zero correlation, the slope of SER is nearly equal to -2 , meaning a diversity order of 2. The presence of antenna correlation will decrease the performance.

Fig. 4.12 shows the STTD transmission performance with channel estimation by linear interpolation and that by Wiener filtering under single-path Rayleigh fading at several dif-

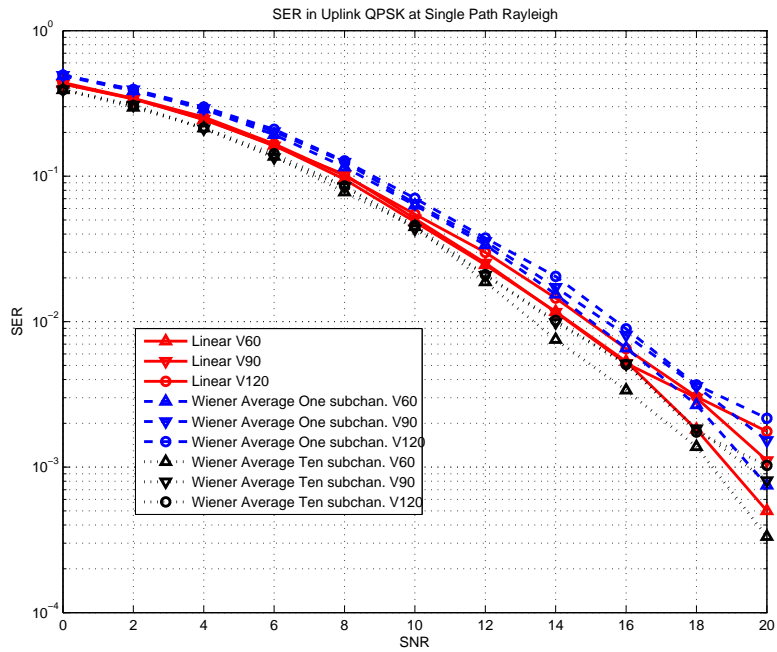
ferent velocities, where the antenna correlation is equal to zero. Figs. 4.13 and 4.14 are under SUI2 and SUI3 respectively. In OFDMA, the tile allocation in frequency domain is not contiguous, if choose different subchannel to average to get correlation, the performance of Wiener filtering might be different. In Fig. 4.15 we simulate two different subchannel sets, each set containing ten subchannels, and using Wiener filtering with correlation average over one subchannel. In the simulation, we see no difference at SER and MSE in two different sets of subchannel. Thus we can ignore the influence of different subchannel.

In Figs. 4.16, 4.17, and 4.18, we compare the SER with zero and nonzero antenna correlations. From the simulation, we can see that, since the power delay profile does not exceed the CP length, the MSEs for different power delay profiles have little difference. We also notice that at high SNR, the MSE saturates because of channel fading. Comparing linear interpolation and Wiener filtering, at low SNR, the Wiener filter has better performance if the samples averaged are enough. But in high SNR, the performance is almost the same. If there is nonzero antenna correlation, the performance would degrade.



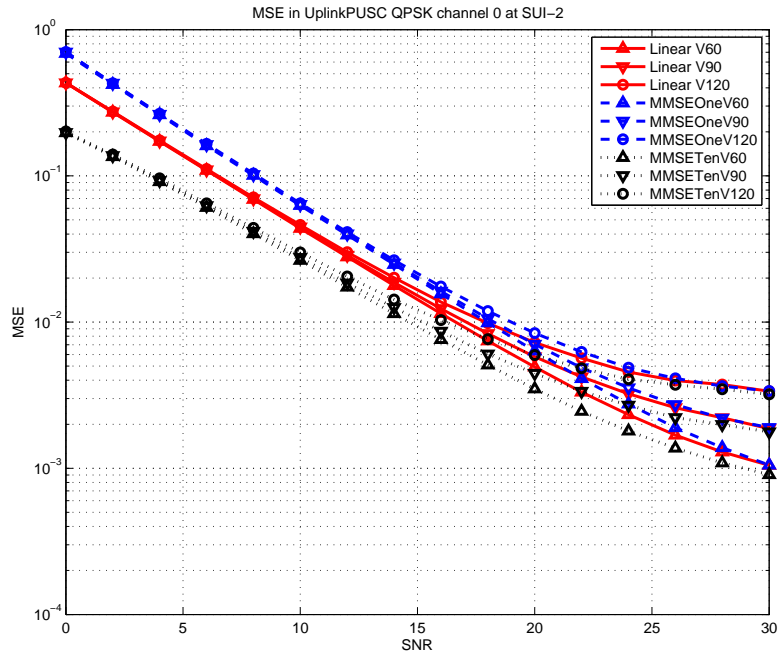


(a)

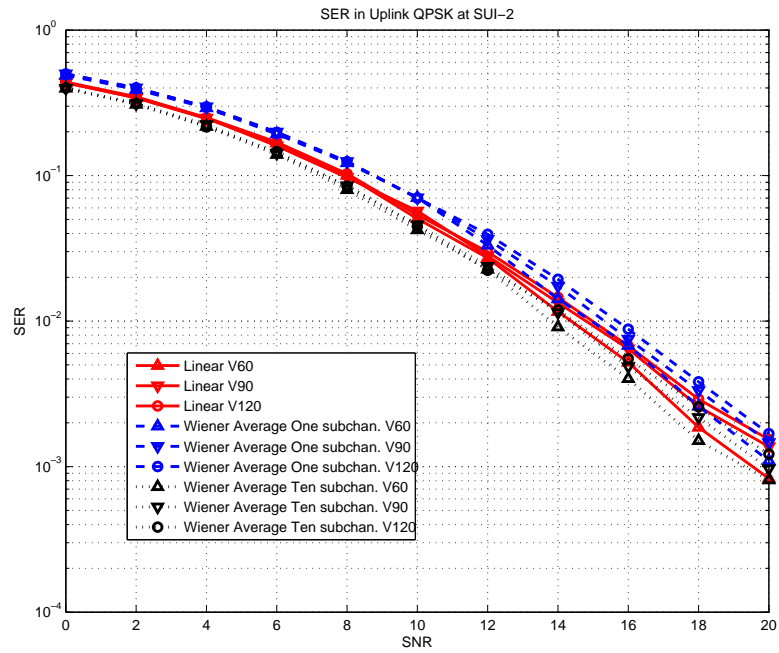


(b)

Figure 4.12: MSE and SER performance for uncoded QPSK under Wiener filtering and linear interpolation channel estimations at different velocities in single-path Rayleigh fading channel with $\rho_{env} = 0$. (a) MSE. (b) SER.

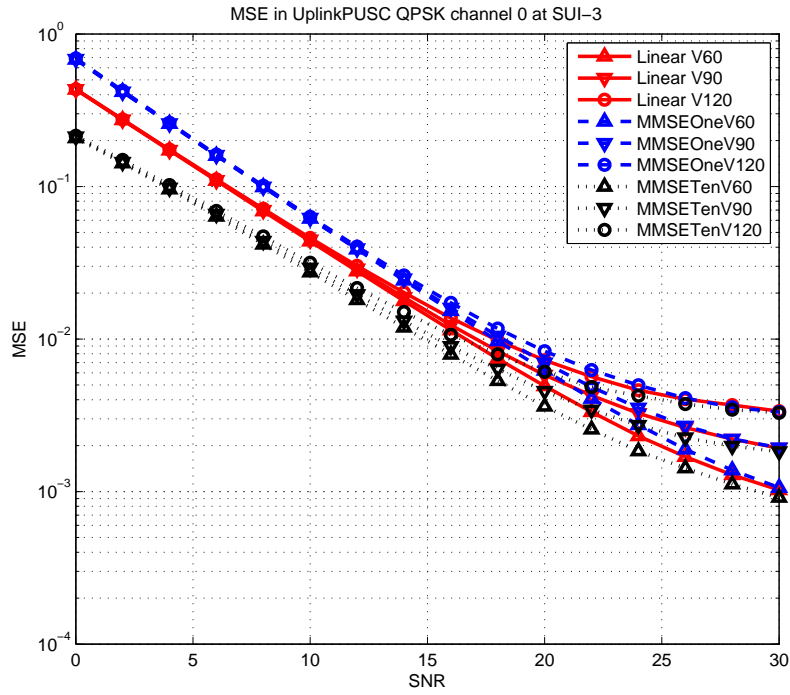


(a)

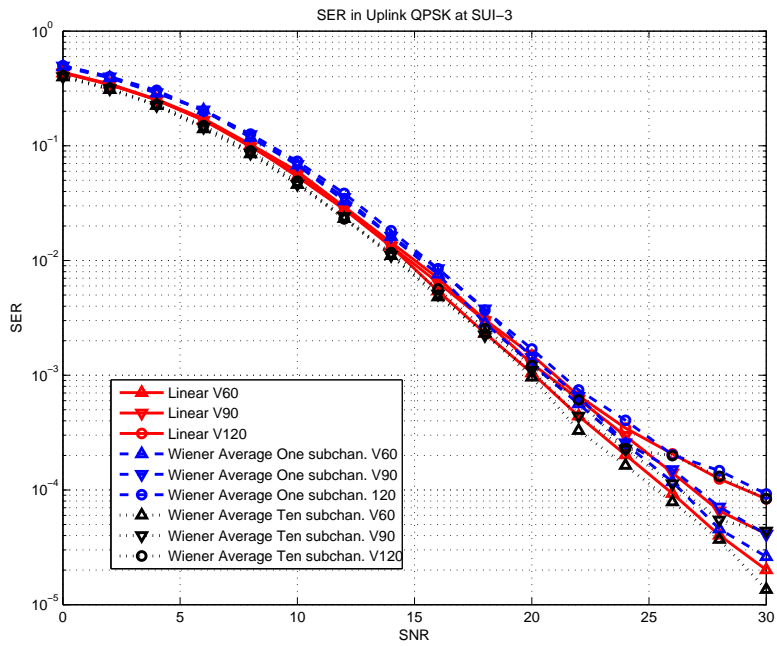


(b)

Figure 4.13: MSE and SER performance for uncoded QPSK under Wiener filtering and linear interpolation channel estimation at different velocities in SUI-2 channel with channel correlation $\rho_{env} = 0$. (a) MSE. (b) SER.

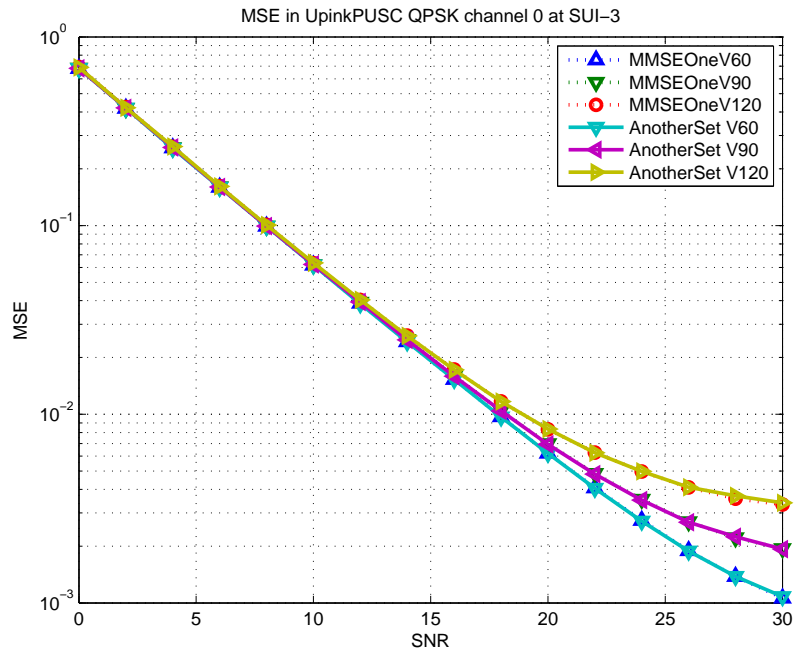


(a)

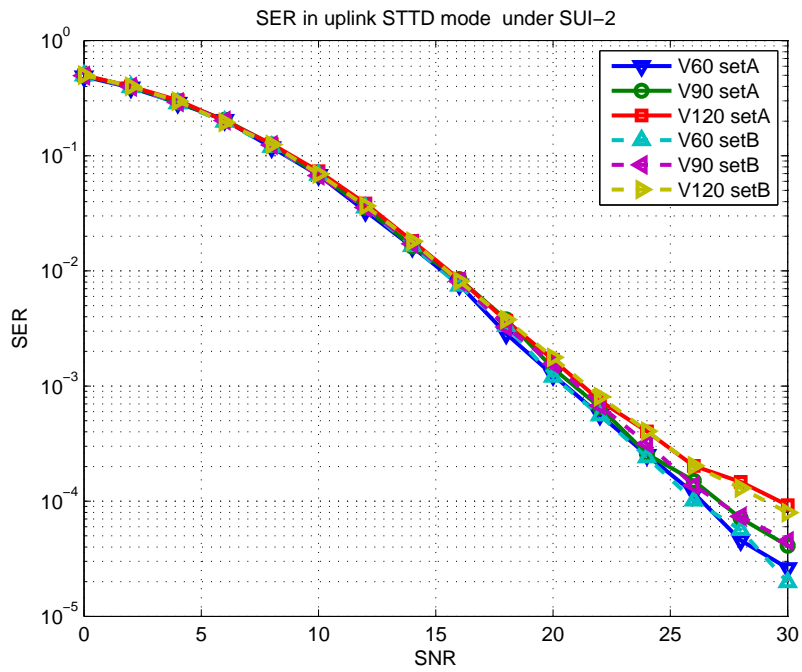


(b)

Figure 4.14: MSE and SER performance for uncoded QPSK under Wiener filtering and linear interpolation channel estimation at different velocities in SUI-3 channel with channel correlation $\rho_{env} = 0$. (a) MSE. (b) SER. 56



(a)



(b)

Figure 4.15: Two different subchannel sets of MSE and SER performance for uncoded QPSK under Wiener filtering averaging over one subchannel at different velocities in SUI-2 channel with channel correlation $\rho_{env} = 0$. (a) MSE. (b) SER.

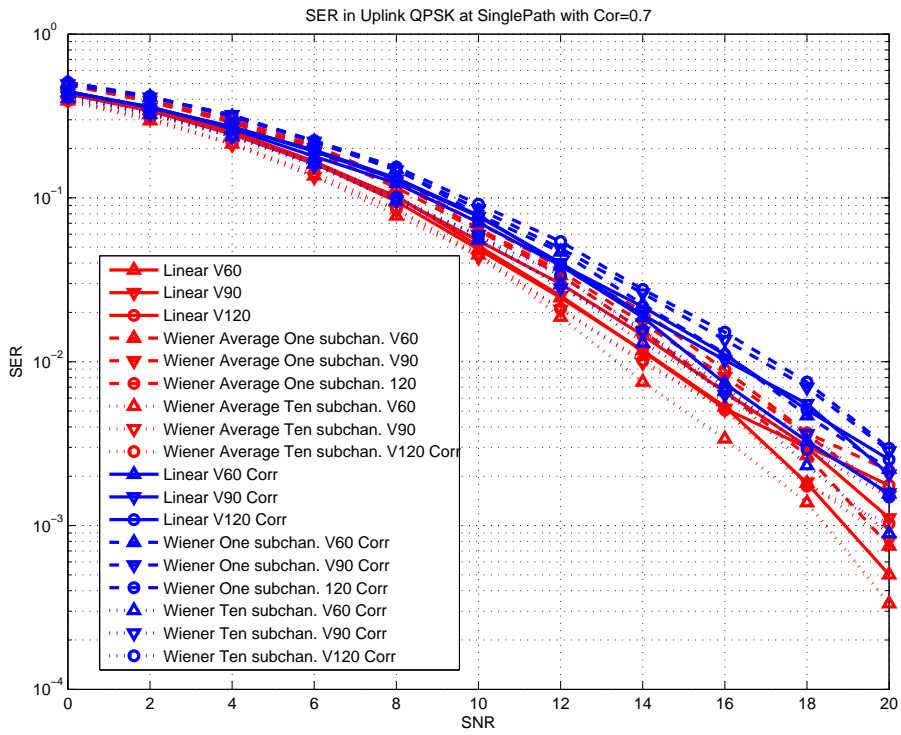


Figure 4.16: SER comparison between zero and nonzero antenna correlation ($\rho_{env} = 0.7$) in single-path Rayleigh fading.

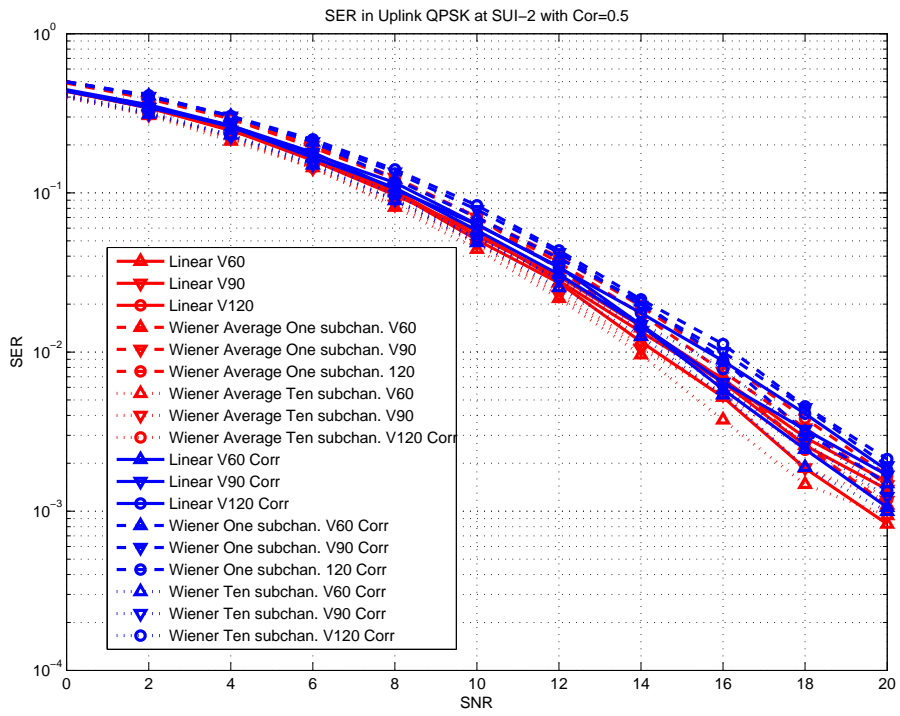


Figure 4.17: SER comparison between zero and nonzero antenna correlation ($\rho_{env} = 0.5$) in SUI-2 channel.

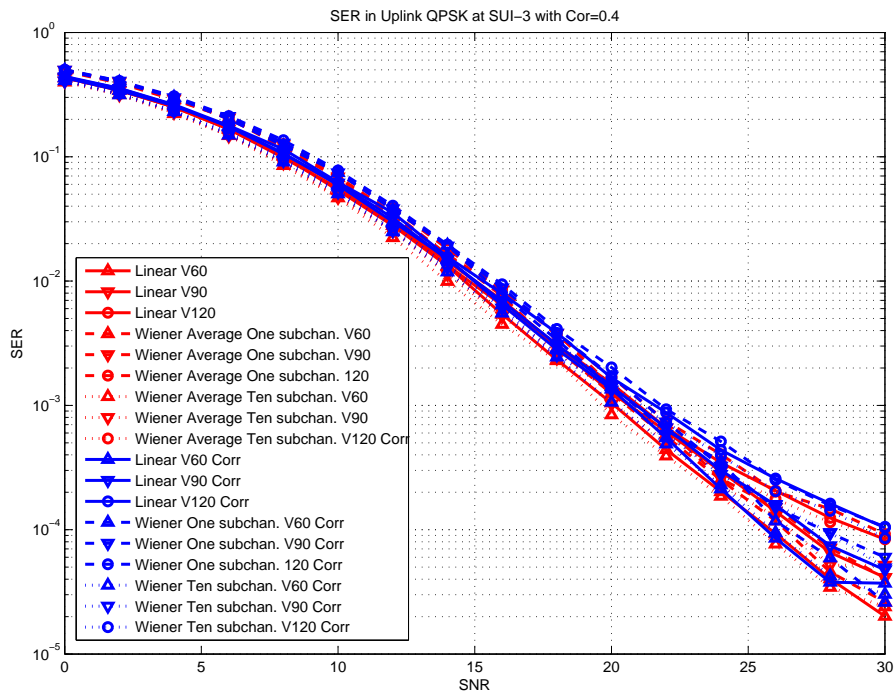


Figure 4.18: SER comparison between zero and nonzero antenna correlation ($\rho_{env} = 0.4$) in SUI-3 channel.

Chapter 5

Simulation of STC Downlink PUSC Channel Estimation

In this chapter we will simulate two different channel estimation methods for the in IEEE 802.16e OFDMA downlink PUSC system. One is linear interpolation and the other is Wiener filtering as introduced before. We evaluate the performance of both methods mainly by observing the mean square error (MSE) and the symbol error rate (SER).

5.1 System Parameters and Channel Models

Table 5.1 gives the primitive and derived parameters used in our simulation work. In addition to AWGN, we use SUI-2 and SUI-3 to do simulation. Their profiles are already introduced in Table 4.3.

5.2 Linear Interpolation

Similar to uplink, the number of pilots contained in one cluster is not enough for us to interpolate for the channel response. We use cluster $(N-1)$ and cluster $(N+1)$ to interpolate the channel response of cluster N , as shown in Fig. 5.1. Within three successive clusters, we first estimate the channel response at each pilot position. Then we interpolate for the

Table 5.1: OFDMA Downlink Parameters

Parameters	Values
Bandwidth	10 MHz
Carrier frequency	3.5 GHz
N_{FFT}	1024
N_{used}	841
Sampling factor n	28/25
G	1/8
Sampling frequency	11.2 MHz
Subcarrier spacing	10.94 kHz
Useful symbol time	91.43 μ s
CP time	11.43 μ s
OFDMA symbol time	102.86 μ s
Sampling time	89.29 ns

frequency response at each intervening data subcarrier from the estimated pilot responses in the time domain. Lastly, we get the frequency responses of the remaining data subcarriers in the cluster by interpolation and extrapolations in the frequency domain, as in the case of uplink.

The detailed steps are as follows:

- Estimate the channel response at each pilot location by using the LS technique.
- Use linear interpolation to estimate the data subcarrier responses between the pilots in the time dimension (marked 1 in Fig. 5.1).
- Estimate the channel responses at the remaining data subcarriers in a cluster by frequency domain interpolation (marked 2 in Fig. 5.1).
- Extrapolate for the channel responses at the rightmost data subcarriers in the cluster.

As shown in Fig. 5.1, all the data subcarrier responses are estimated by interpolation using the four nearest pilot subcarriers except for the rightmost data subcarriers where extrapola-

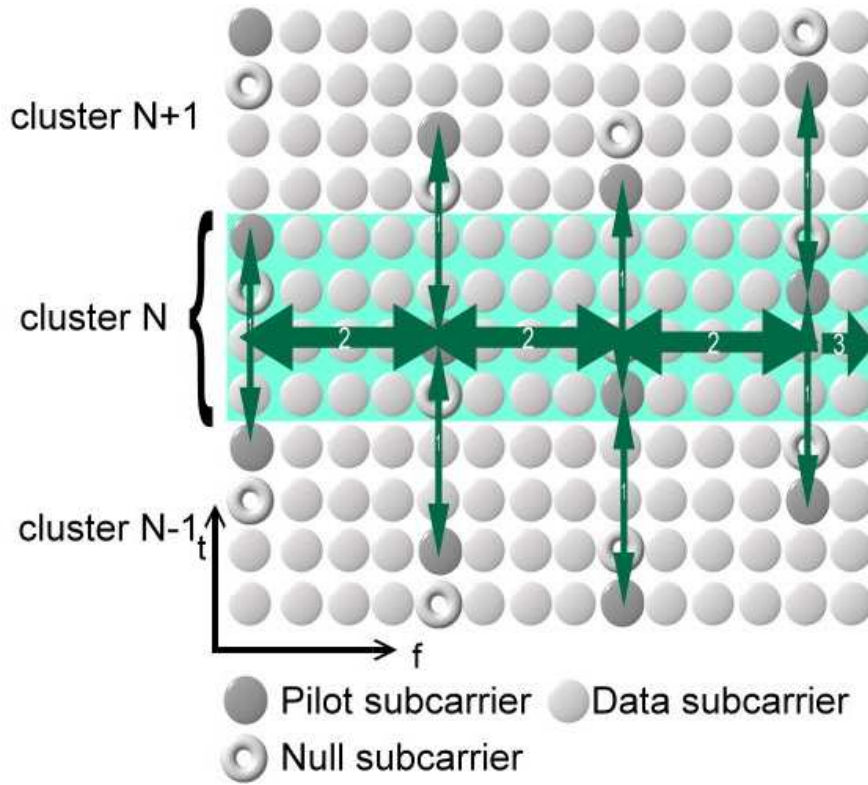


Figure 5.1: Linear interpolation in STTD mode at antenna 0.



tion is used.

5.3 Wiener Filtering

As mentioned before, in Wiener filtering, we need to know the autocorrelation between pilots and cross-correlation between data subcarriers and pilots. To calculate the autocorrelation, we suppose that a major group or a whole OFDMA symbol employ STC encoding. Then we can average over the major group or the entire symbol to estimate the autocorrelation.

We use the four pilots in the cluster to do two-dimensional Wiener filtering, as shown in Figure 5.2. In the case of the autocorrelation, if we want a more accurate estimate, we can average over three temporally contiguous clusters. To calculate the cross-correlation,

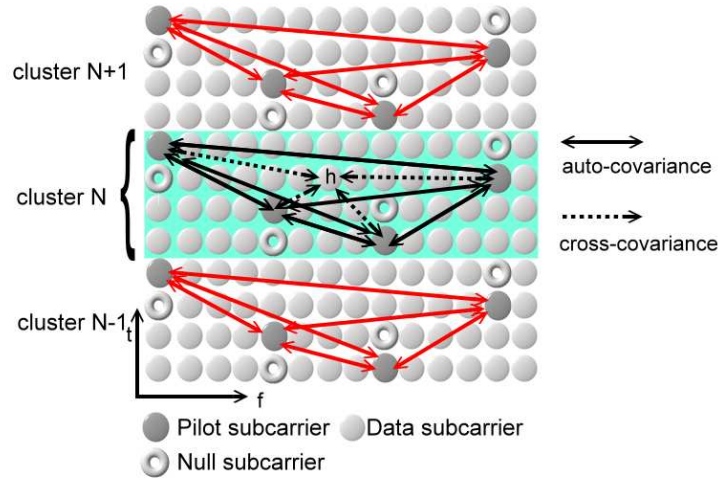


Figure 5.2: Wiener filtering in STTD mode at antenna 0.

we linearly interpolate for the estimated responses at the pilot locations and average their sample cross-correlations with the pilot channel response estimates over the clusters in the frequency domain.

The detailed steps are as follows:

- Estimate the channel responses using linear interpolation.
- Estimate the noise power by averaging over the guard band subcarriers.
- Use the estimated channel responses and add the noise power correction term to calculate cross-correlation.
- Calculate the autocorrelation of the four pilots in the cluster.
- Use the Wiener filtering formula to estimate the channel responses.

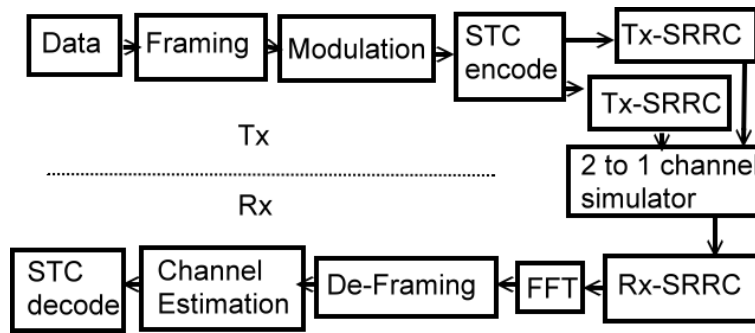


Figure 5.3: Block diagram of the simulated system.

5.4 Simulation Study

5.4.1 Simulation Flow

Figure 5.3 illustrates the block diagrams of our simulated system. We also assume perfect synchronization and omit it in our simulation. After channel estimation, as we do in uplink transmission, we calculate the channel MSE between the real channel and the estimated one, where the average is taken over the subcarriers. The symbol error rate (SER) can also be obtained after demapping. The used channel models are as the same as described in Chapter 4.

5.4.2 Validation with AWGN Channel

Before considering multipath channels, we do simulation with an AWGN channel to validate the simulation model. We validate this model by comparing the theoretical SER and the SER resulting from simulation.

In Figure 5.4, the theoretical SER curve versus SNR for uncoded QPSK is plotted together with that resulting from the simulation. The simulation is obtained under no channel estimation error. This validates the simulation model.

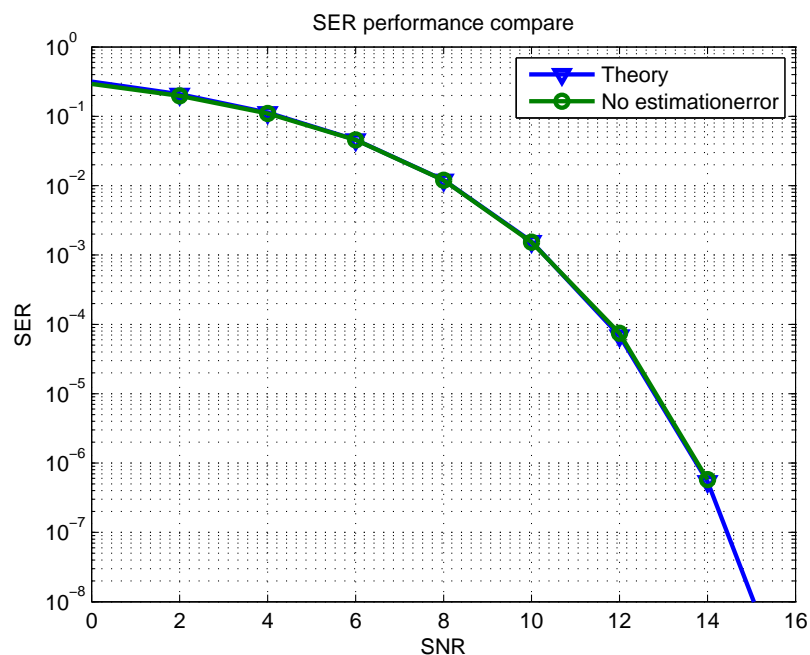


Figure 5.4: SER for uncoded QPSK resulting from simulation compared with theory.

5.4.3 Simulation Results

We know that the accuracy of autocorrelation estimates would affect the Wiener filter performance. The autocorrelation is obtained from the average of the sample correlation over a number of clusters. That means, in AWGN channel, if more clusters are used in the average, then the accuracy of the autocorrelation estimates is better. So in the following simulation, we show three different choices of number of clusters to average for the autocorrelation. One is averaging over one major group (in the simulation we choose major group 0 which contains 12 clusters) in frequency domain. The second one is averaging over one major group in the frequency domain over three contiguous clusters in the time domain (a total of 36 clusters). The third is averaging over all subchannels (60 clusters in one OFDM symbol). The cross-correlation in the first method is averaging over one major group in frequency domain as the autocorrelation. It is worth noting that the cross-correlation in the second

method is obtained from averaging only over the frequency domain in one major group as in the first method since we only interpolate the frequency response in the middle cluster. The cross-correlation in the third method is averaging over all subchannels. . Figure 5.5 show the MSE and SER simulation result.

From the simulation, we find that the autocorrelation obtained from average over time and frequency domains together with the cross-correlation obtained only from frequency domain has poor performance, even though the autocorrelation is more accurate. In fact, if we use the theoretical cross-correlation value (the theoretical cross-correlation value is 1 in AWGN channel) together with the autocorrelation obtained from simulation, the performance is bad, too. The reason should be that the number of pilots we use is not close to infinity. Since the pilot signals contain noise, if we only use a finite number of pilot signals, the autocorrelation and the cross-correlation are not independent in statistic. And if we choose the same set of pilots to calculate cross-correlation, the statistical dependency causes no influence. Besides this, the ill-conditioning of matrix inversion is another reason causing the performance degradation. Use $SNR = 10$ for example. That means if the pilot power equals to 1, then the noise power is 0.1. The autocorrelation would be

$$\Phi = \begin{bmatrix} 1.1 & 1 & 1 & 1 \\ 1 & 1.1 & 1 & 1 \\ 1 & 1 & 1.1 & 1 \\ 1 & 1 & 1 & 1.1 \end{bmatrix}, \quad (5.1)$$

and the eigenvalues are 0.1, 0.1, 0.1 and 4.1. The condition number is

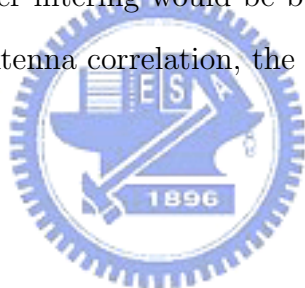
$$\kappa(\Phi) = 41 \quad (5.2)$$

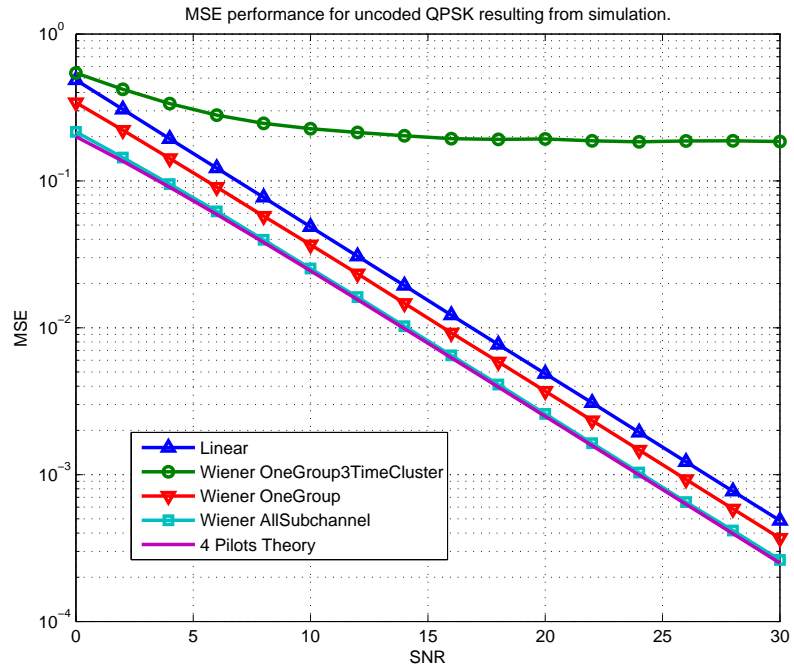
Since the samples we choose are not enough, the noise power does not concentrate on the diagonal terms and causes matrix ill-conditioning.

Figure 5.6 presents the simulation results in the STTD with linear interpolation and Wiener filtering channel estimations and compares them with perfect estimation under single

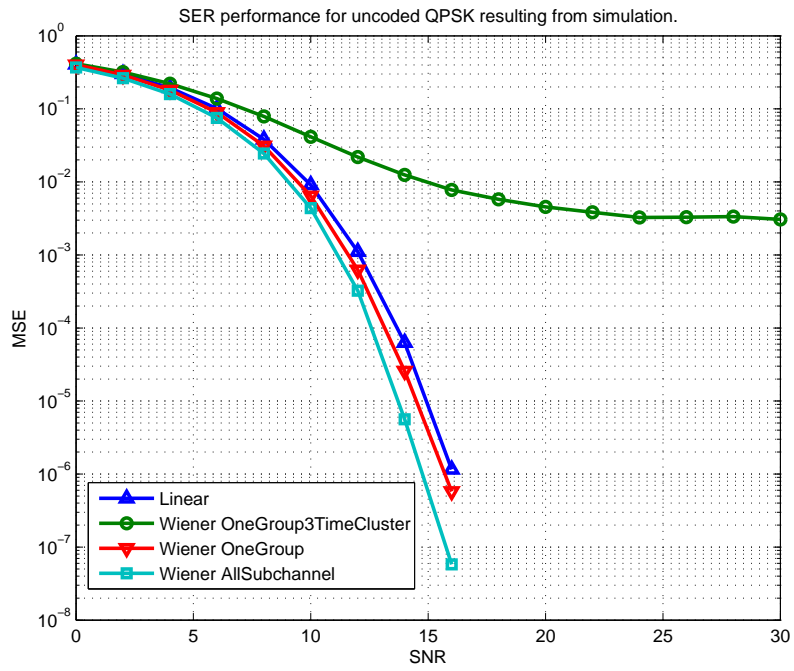
path Rayleigh fading channel in different velocities with zero antenna correlation. Figures 5.7 and 5.8 are under SUI2 and SUI3. We can see that, in low SNR, Wiener filter has better performance than linear interpolation, but in high SNR, the performance is a little worse than linear interpolation. The reason should be that the pilots we used in the Wiener filtering are not the nearest with the data subcarrier that we want to estimate but in the same cluster instead. That means the channel response we estimated is the linear combination of the responses of the four pilots in the same cluster. Since the estimation using linear interpolation is the linear combination of the nearest pilots, in high SNR, the performance is a little better than Wiener filtering.

In Figs. 5.9, 5.10, and 5.11, we compare the SER under zero and nonzero antenna correlation. From the simulation, we see that as in the uplink, if we choose proper samples to average, the performance of Wiener filtering would be better than the linear interpolation. And if the channel has nonzero antenna correlation, the performance would degrade.



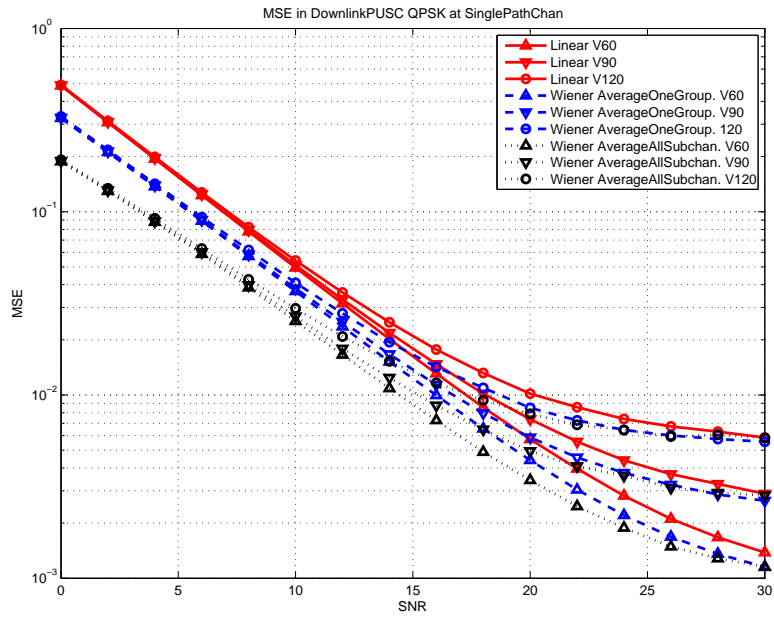


(a)

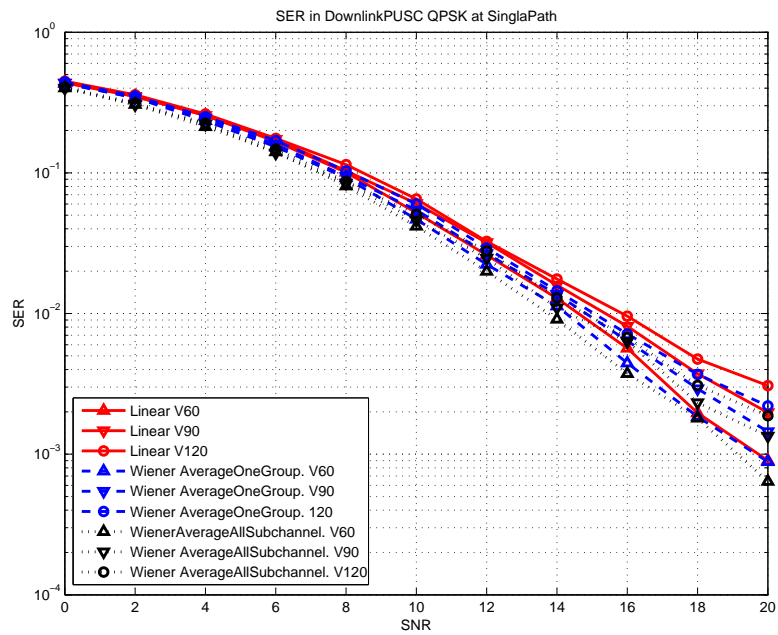


(b)

Figure 5.5: MSE and SER performance for uncoded QPSK resulting from simulation with Wiener filtering and linear interpolation in AWGN channel. (a) MSE. (b) SER.

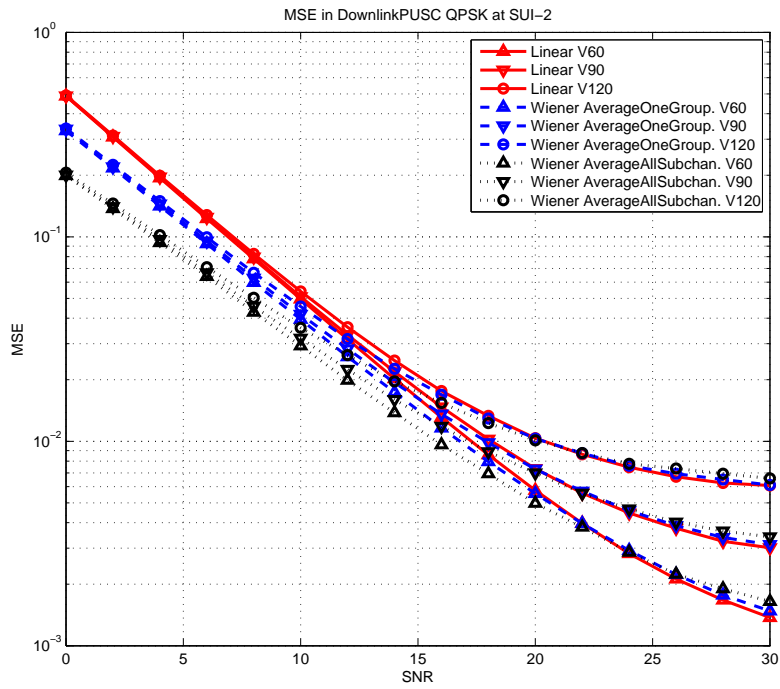


(a)

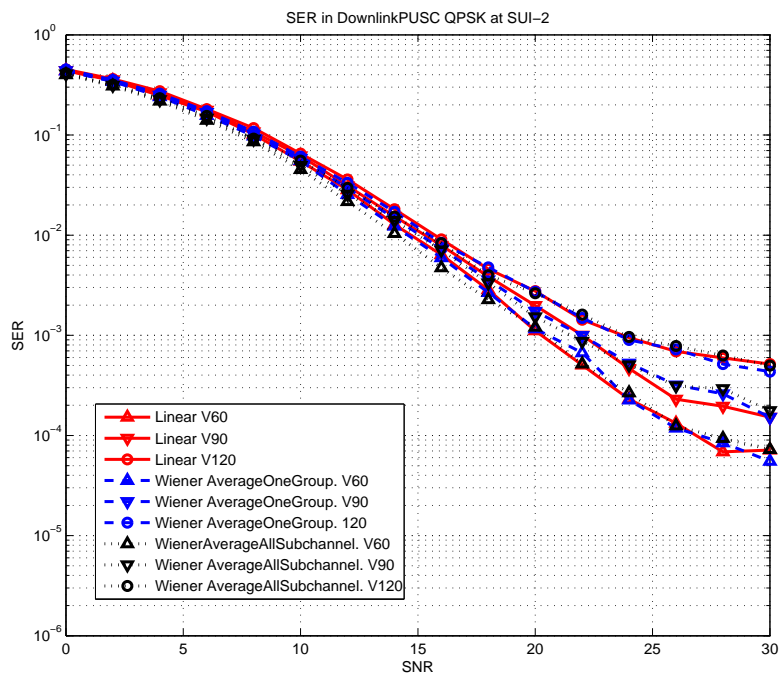


(b)

Figure 5.6: MSE and SER performance for uncoded QPSK resulting from simulation with Wiener filtering and linear interpolation at different velocities in single-path Rayleigh fading channel with $\rho_{env} = 0$. (a) MSE. (b) SER.

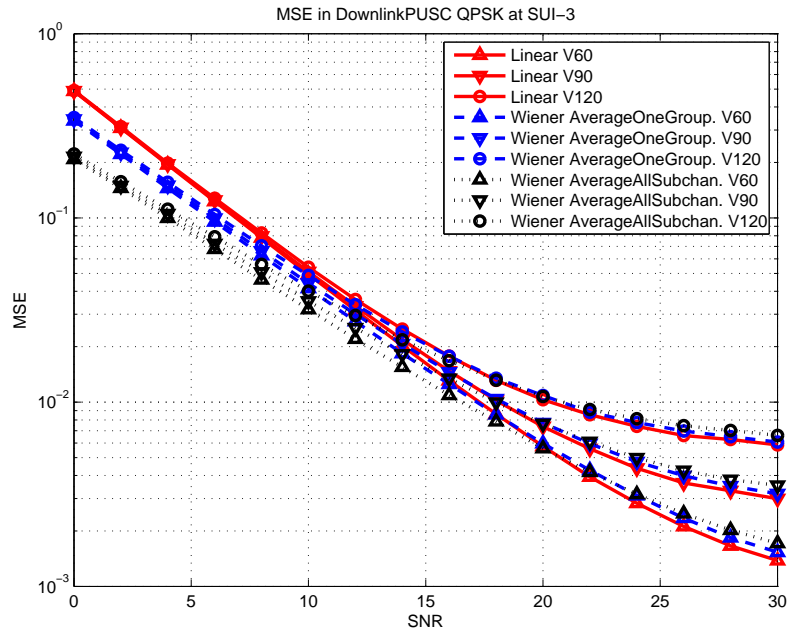


(a)

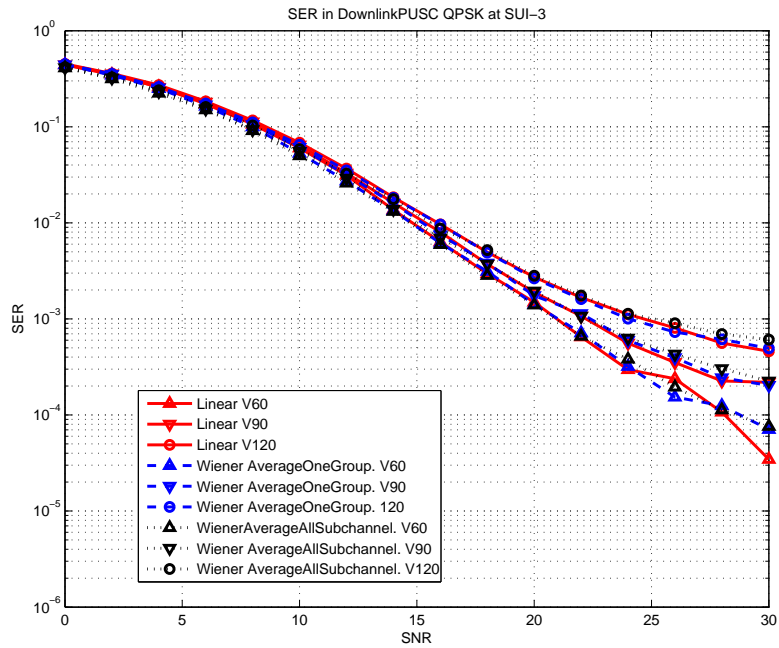


(b)

Figure 5.7: MSE and SER performance for uncoded QPSK resulting from simulation with Wiener filtering and linear interpolation at different velocities in SUI-2 channel with $\rho_{env} = 0$. (a) MSE. (b) SER.



(a)



(b)

Figure 5.8: MSE and SER performance for uncoded QPSK resulting from simulation with Wiener filtering and linear interpolation at different velocities in SUI-3 channel with $\rho_{env} = 0$. (a) MSE. (b) SER.

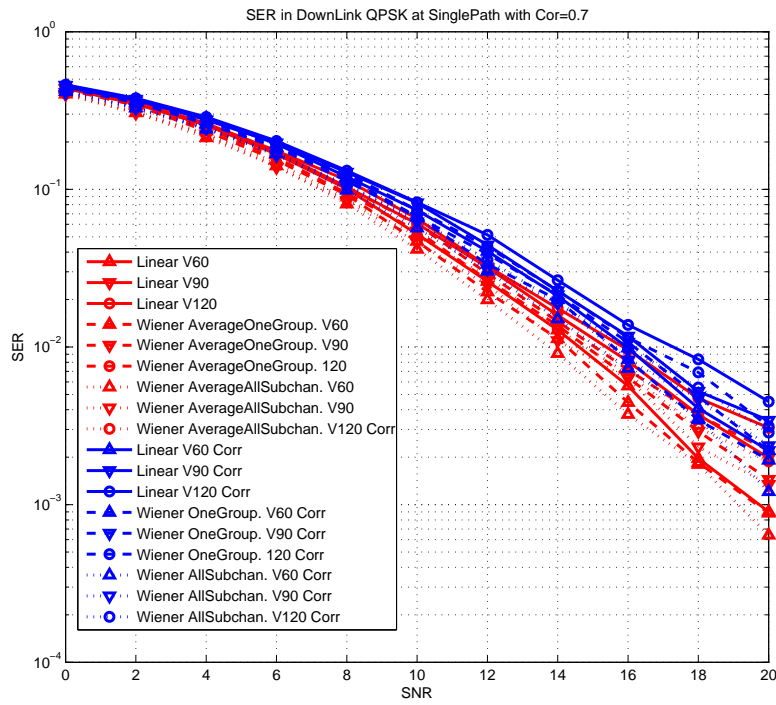


Figure 5.9: SER comparison of zero and nonzero antenna correlations ($\rho_{env} = 0.7$) in single-path Rayleigh fading.

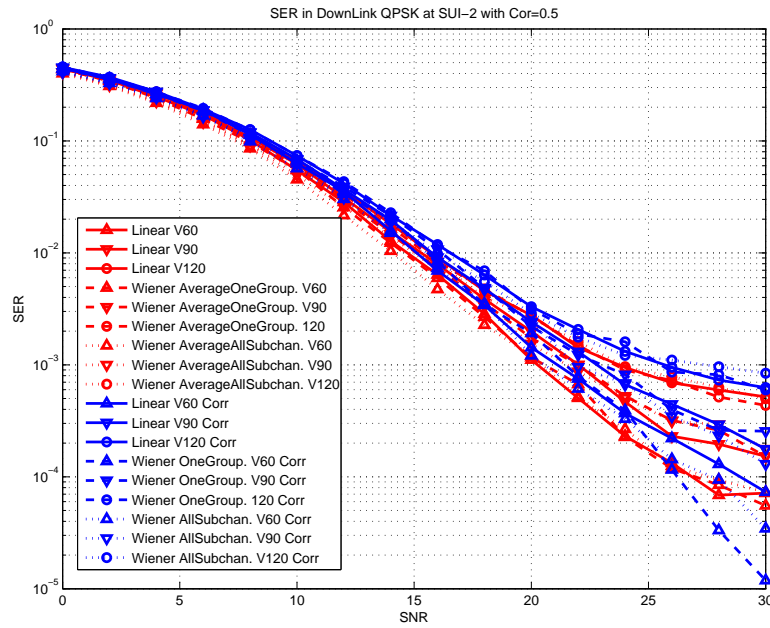


Figure 5.10: SER comparison of zero and nonzero antenna correlations ($\rho_{env} = 0.5$) in SUI-2.

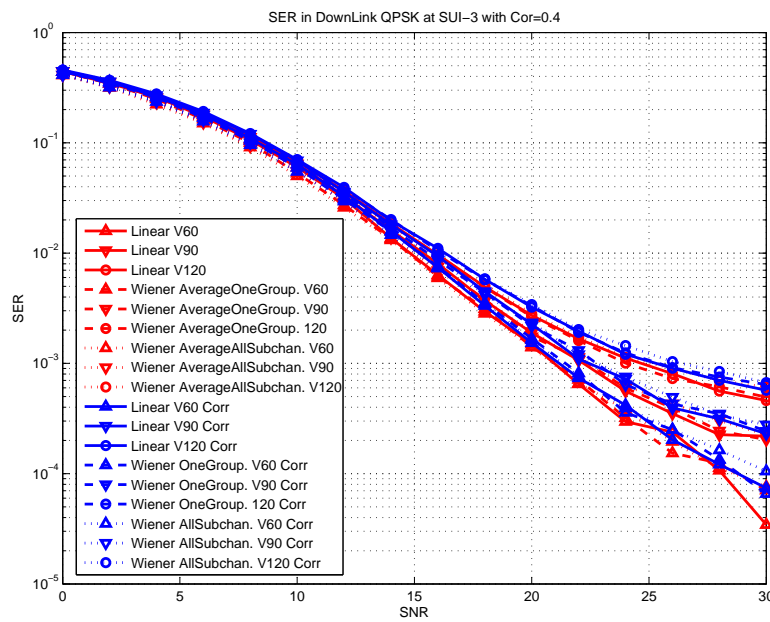


Figure 5.11: SER comparison of zero and nonzero antenna correlations ($\rho_{env} = 0.4$) in SUI-3.

Chapter 6

The DSP Hardware and Associated Software Development Environment

DSP implementation is the final goal of our work. The DSP system used is Sundance's PC plug-in board that houses TMS320C6416 DSP made by Texas Instruments (see Fig. 6.1). In this chapter, we introduce the architecture of the DSP chip and the software development environment.



6.1 The TMS320C6416 DSP

6.1.1 TMS320C64x Features [21]

The TMS320C64x DSP that we employ is the highest-performance fixed-point DSP generation of the TMS320C6000 DSP devices, with a performance of up to 1000 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 can significantly speed up channel-decoding operations on-chip. But they do not apply to the work reported in this thesis.

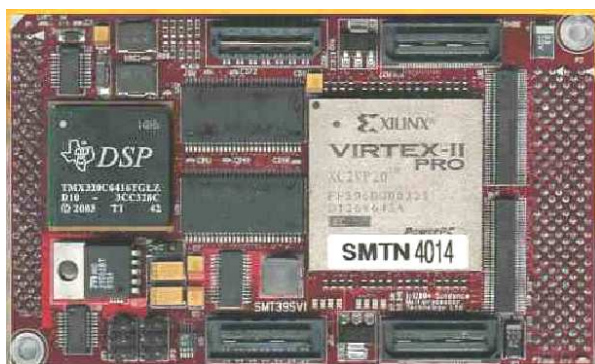


Figure 6.1: The DSP on the Sundance board [21].

The C64x core CPU consists of 64 general-purpose 32-bit registers and 8 function units. These 8 functional units contain 2 multipliers and 6 arithmetic units. Below are some C6000 features:

- Advanced VLIW executes up to eight instructions per cycle and 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 and reduces code size, program fetches, and power consumption.
- Conditional execution of all instructions reduces costly branching and increases parallelism for higher sustained performance.
- Efficient code execution on independent functional units, including efficient C compiler on DSP benchmark suite. and 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 the following:

- 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 addressing common operations in error-correcting codes.
- Bit count and rotate hardware extends support for bit-level algorithms.



6.1.2 Central Processing Unit [21]

The block diagram of the C6416 DSP is shown in the Fig. 6.2. The C64x CPU, shaded in the figure, contains:

- Program fetch unit.
- Instruction dispatch unit.
- Instruction decode unit.
- Two data paths, each with four functional units.

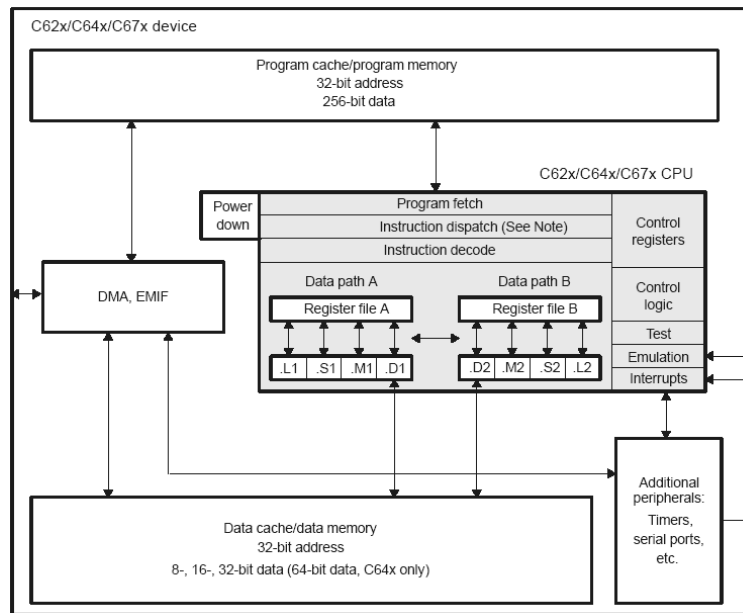


Figure 6.2: Block diagram of the TMS320C6416 DSP [21].

- 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.

6.1.2.1 Pipeline Structure

The TMS320C64x DSP pipeline provides flexibility to simplify programming and improve performance. The pipeline can dispatch eight parallel instructions every cycle. The pipeline

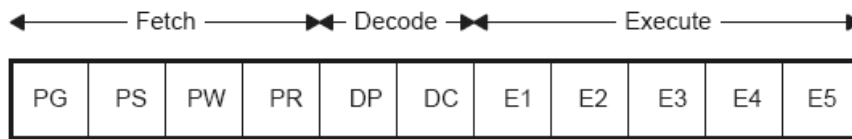


Figure 6.3: Pipeline phases of TMS320C6416 DSP [21].

phases are divided into three stages as shown in Fig. 6.3.

- Fetch has 4 phases:
 - 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 fetch 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: Execute 1.
 - E2: Execute 2.
 - E3: Execute 3.
 - E4: Execute 4.
 - E5: Execute 5.

Table 6.1: Execution Stage Length Description for Each Instruction Type [21]

		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‡

The pipeline operation of the C62x/C64x instructions can be categorized into seven instruction types. Six of them are shown in Table 6.1, 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, results are available from other instructions finishing execution during the same CPU cycle in which the multiply is in a delay slot.

Table 6.2: Functional Units and Operations Performed (Part 1 of 2) [21]

Functional Unit	Fixed-Point Operations	Floating-Point Operations
.L unit (.L1, .L2)	32/40-bit arithmetic and compare operations 32-bit logical operations Leftmost 1 or 0 counting for 32 bits Normalization count for 32 and 40 bits Byte shifts Data packing/unpacking 5-bit constant generation Dual 16-bit arithmetic operations Quad 8-bit arithmetic operations Dual 16-bit min/max operations Quad 8-bit min/max operations	Arithmetic operations DP → SP, INT → DP, INT → SP conversion 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	Compare Reciprocal and reciprocal square-root operations Absolute value operations SP → DP conversion operations



6.1.2.2 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 Tables 6.2 and 6.3.

Besides being able to perform 32-bit operations, the C64x also contains many 8-bit and 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.

Table 6.3: Functional Units and Operations Performed (Part 2 of 2) [21]

Functional Unit	Fixed-Point Operations	Floating-Point 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	32 X 32-bit fixed-point multiply operations Floating-point multiply operations
.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	Load doubleword with 5-bit constant offset

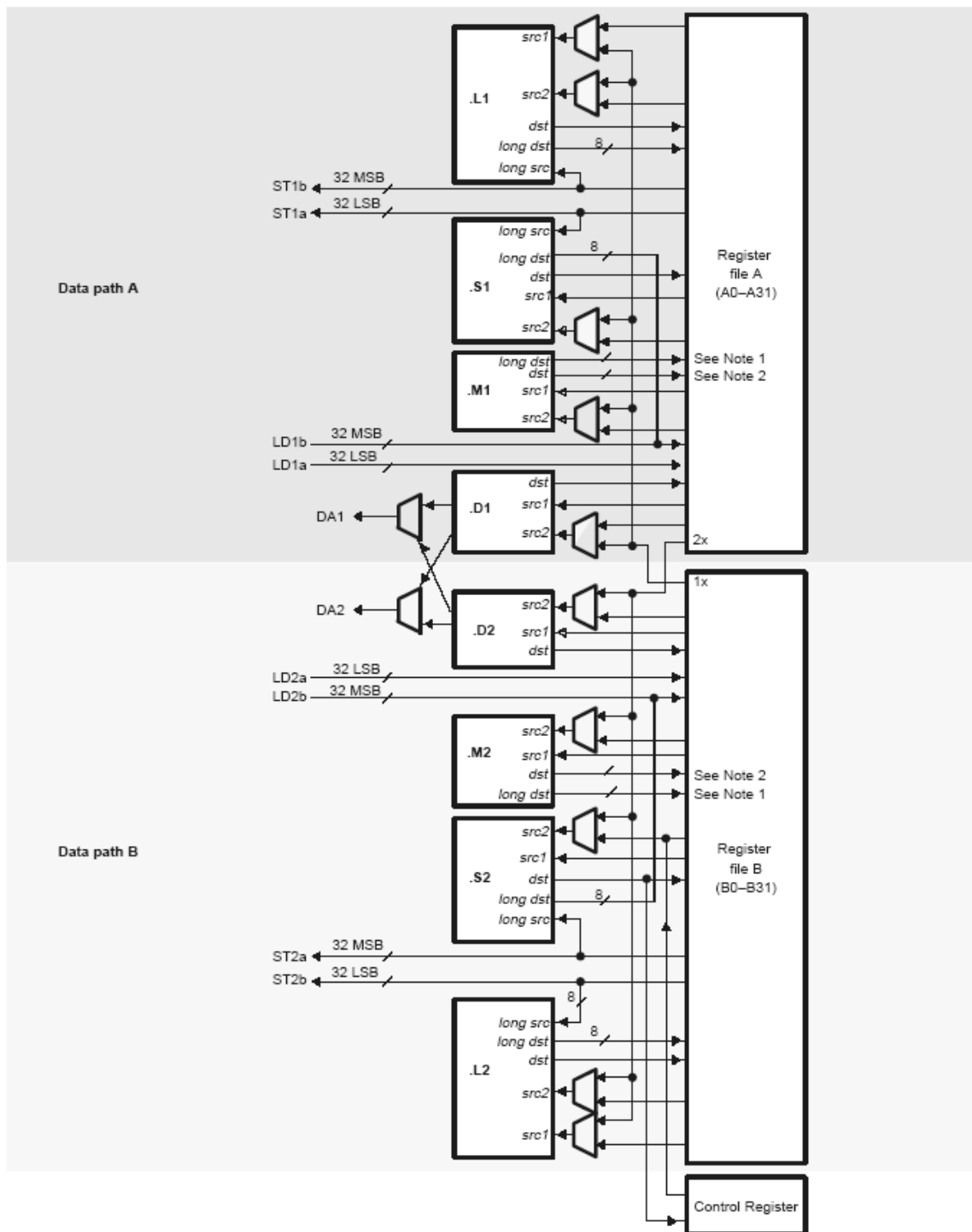
The data line in the CPU supports 32-bit operands, long (40-bit) and double word (64-bit) operands. Each functional unit has its own 32-bit write port into a general-purpose register file (listed in Fig. 6.4). All units ending in 1 (for example, .L1) write to register file A, and all units ending in 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.

6.1.3 Memory Architecture [21]

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 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 7 Mbits.
- Program cache.
- 32-bit external memory interface supports SDRAM, SBSRAM, SRAM, and other asynchronous memories.
- Two-level caches [22]. Level 1 cache is split into program (L1P) and data (L1D) caches. 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



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

Figure 6.4: TMS320C64x CPU data paths [21].

locations. The size of L2 is 1 MB. The access time of external memory 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.

6.2 The Code Composer Studio Development Tools [24], [25]

We now introduce the software environment used in our work. 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 [26]. The routines included in the DSP library are organized as follows:

- Adaptive filtering.
- Correlation.
- FFT.
- Filtering and convolution.
- Math.

- Matrix functions.
- Miscellaneous.

6.3 Code Optimization Methods [27]

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. 6.5. The tutorial section of the Programmer's Guide [27] 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. 6.5.

- 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 [27] to improve the C code. Use the C6000 profiling tools

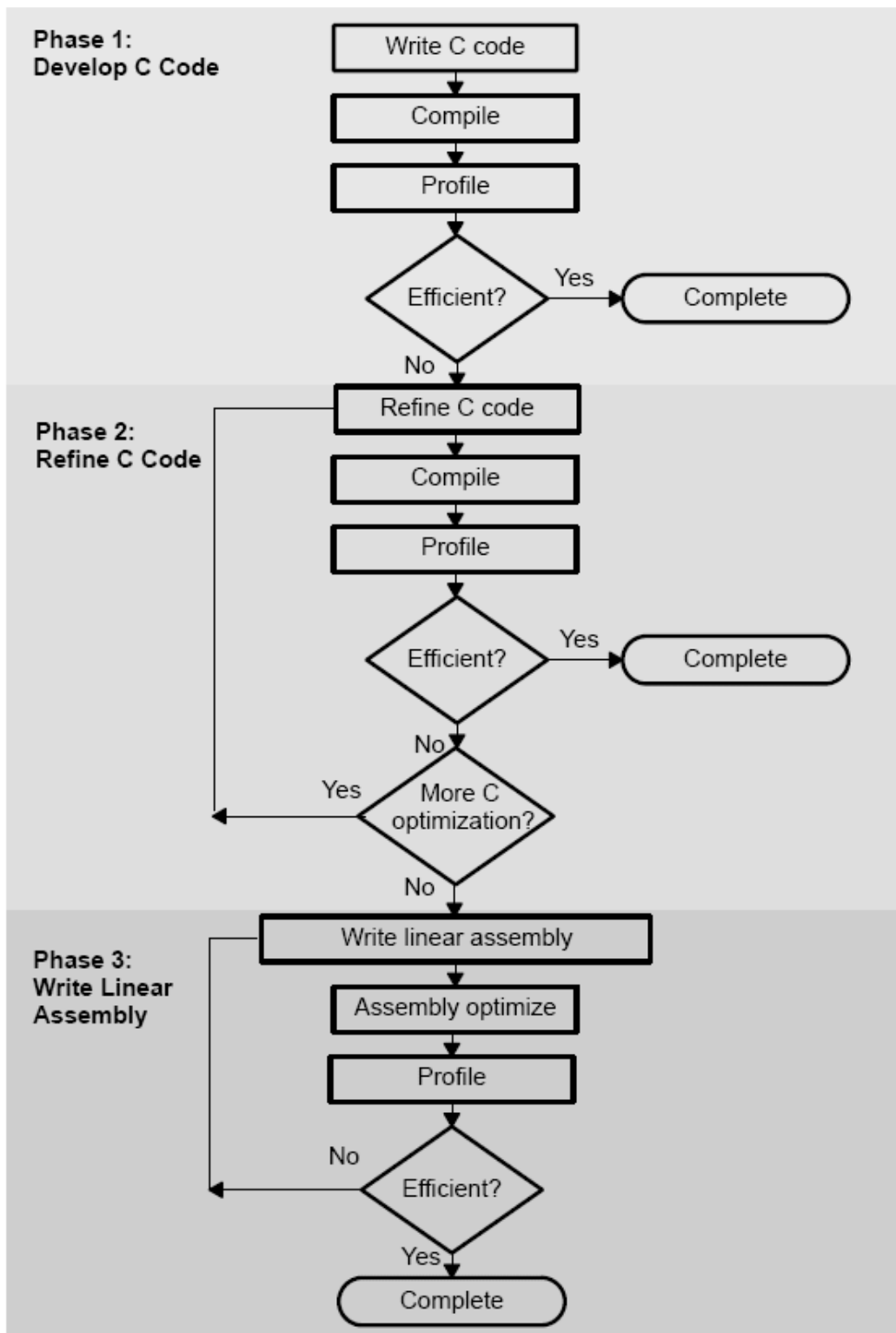


Figure 6.5: Code development flow for TI C6000 DSP [27].

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.

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.

6.3.1 Compiler Optimization Options [24], [25]

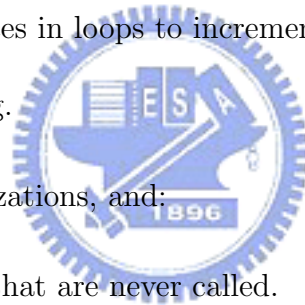
The compiler supports several options to optimize the code. The compiler options can be used to optimize code size or execution performance. Our primary concern in this work is the execution performance. Hence we do not care very much about the code size. The easiest way to invoke optimization is to 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. In our study, we use `-o3`, `pm` and Speed Most Critical (no instruction) as compiler condition number

6.3.2 Using Intrinsic

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 [27].

Chapter 7

Fixed-Point DSP Implementation

7.1 Data Formats Considerations

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 consider implementation using in 16-bit fixed-point computations. Compared with 32-bit computation, it has better efficiency and negligible accuracy loss in many applications. Although fixed-point operation has less accuracy, it does have much shorter execution time. However, we find that the fixed-point format is only suitable in linear interpolation, not in Wiener filtering, because the Wiener filtering needs complex matrix inversion which needs very high accuracy.

In our simulation, we use the format Q2.13, which means a 16-bit fixed-point number with one sign bit, 2 integer bits, and then 13 fractional bits to the right of the dot. Here we only focus on the channel estimation function. Therefore, we only translate the input to channel estimation into fixed-point format for simplicity. The simulation flow is shown in

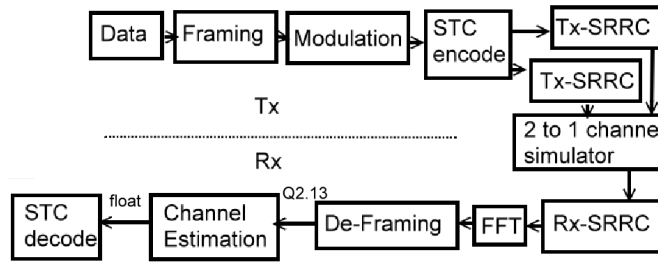


Figure 7.1: fix point simulation flow.

Fig. 7.1.

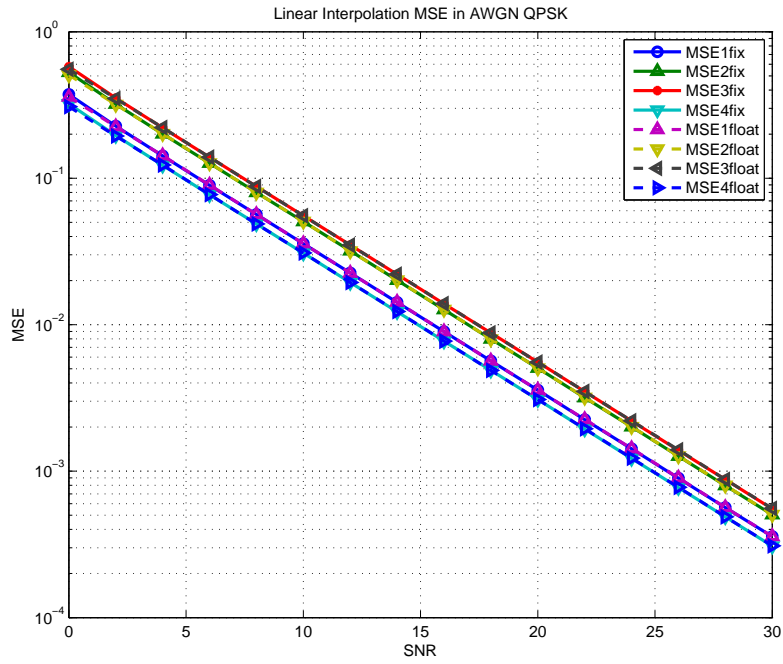
7.2 Fixed-Point Simulation

We only adopt fixed-point computation in the linear interpolation method, since Wiener filtering needs to calculate matrix inverse that needs high dynamic range.

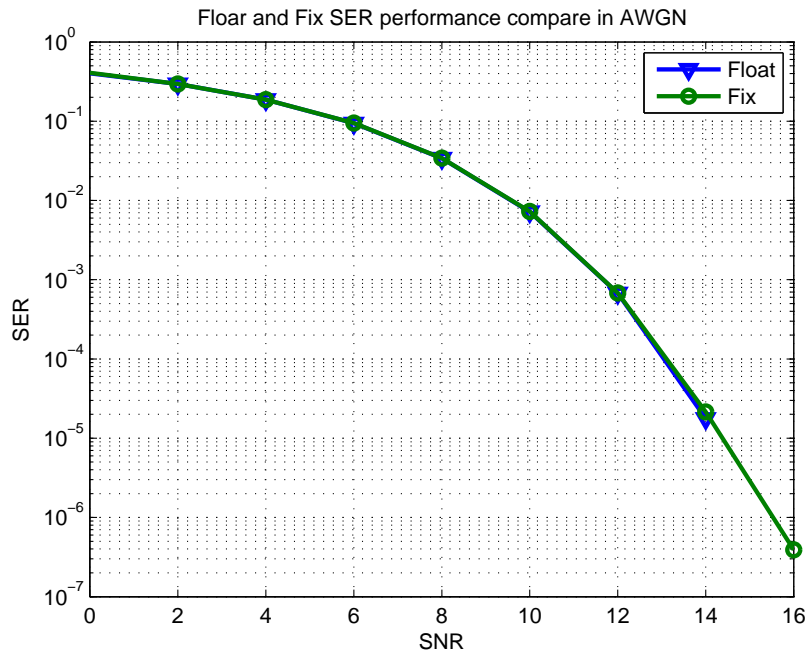
The accuracy results of linear interpolation channel estimation in uplink with different velocities in different channels are given in Figures 7.2 to 7.6. The simulation results under fixed-point and floating-point computations are shown in Figures 7.7 to 7.11. With fixed-point computation, we can see that the performance is almost the same the floating-point computation. In low SNR the performance of fixed-point computation has a little degradation. We think the reason is when the noise power is high, it is easy to cause fixed-point data overflow.

7.3 DSP Computation Load

We run the fixed-point C program under CCS to see the DSP cycle count performance. Although Wiener filtering is not suitable for fixed-point computation, we still run it in fixed-point format to see the computational load. We also compare the result with the uplink tile

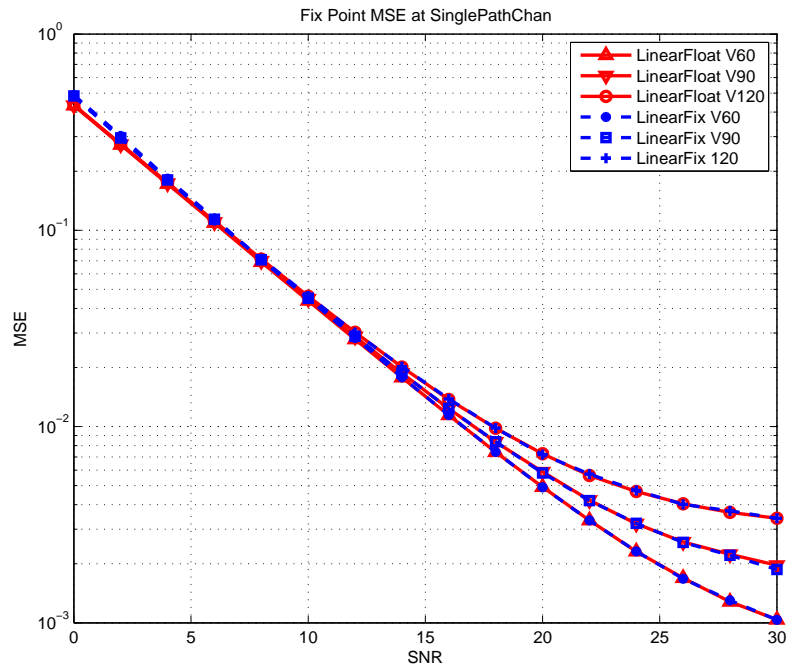


(a)

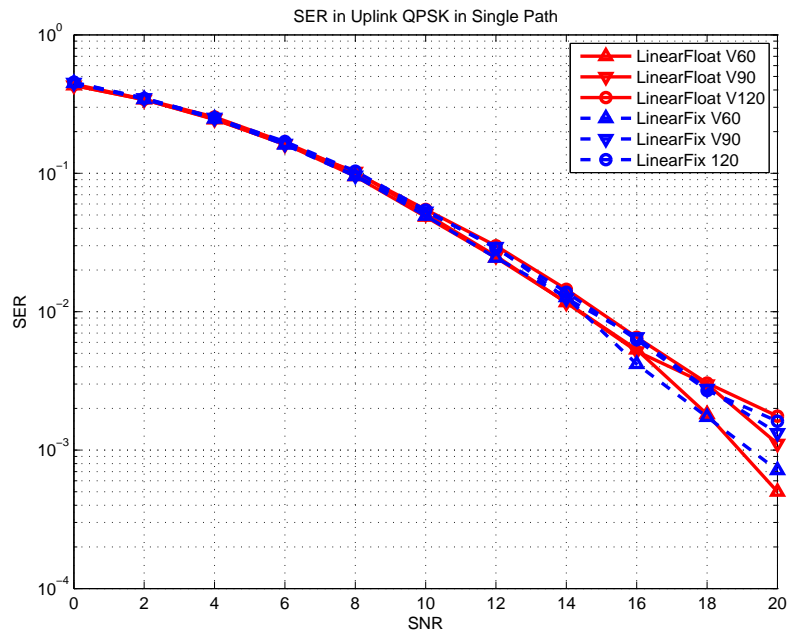


(b)

Figure 7.2: Uplink channel estimation performance under fixed- and floating-point computation in AWGN. (a) MSE. (b) SER.

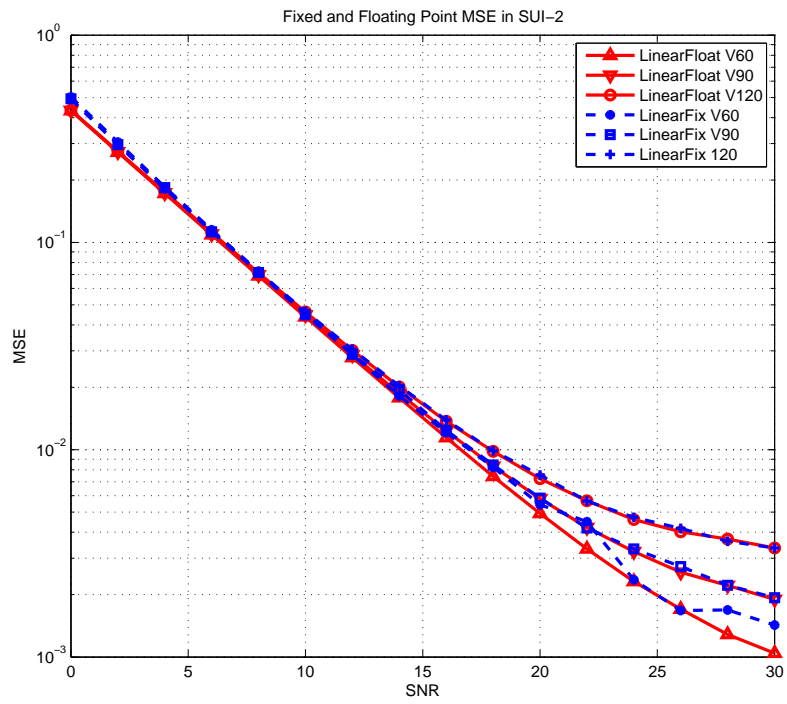


(a)

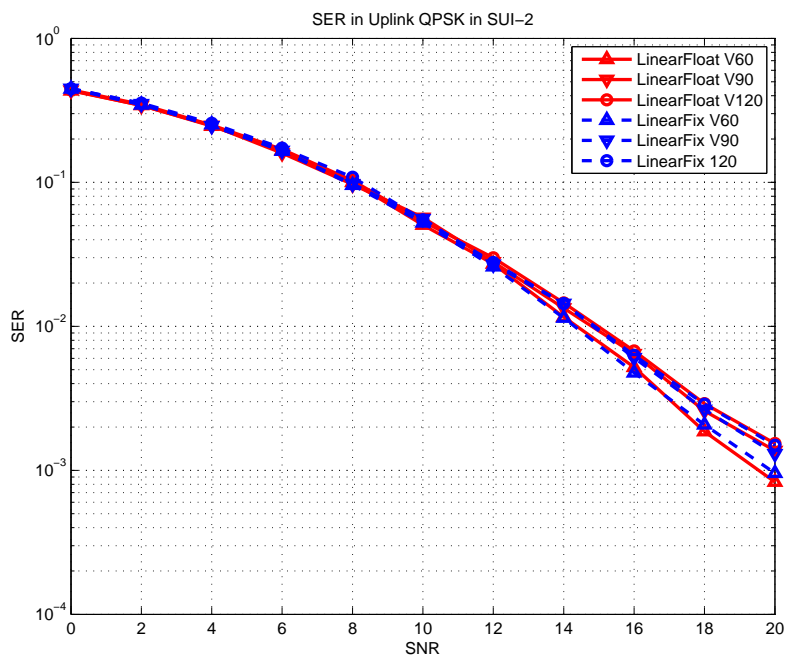


(b)

Figure 7.3: Uplink channel estimation performance under fixed- and floating-point computation in single-path Rayleigh fading. (a) MSE. (b) SER.

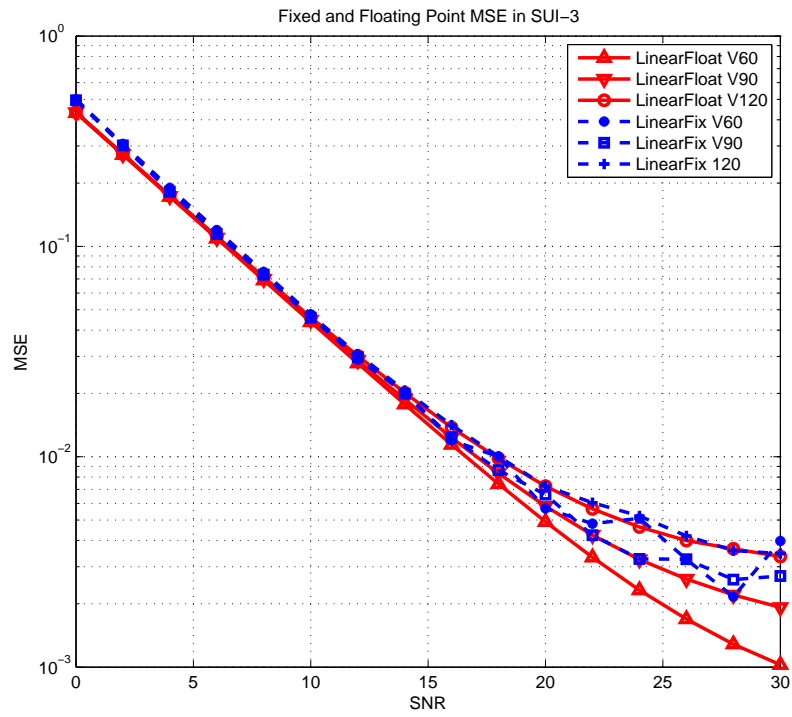


(a)

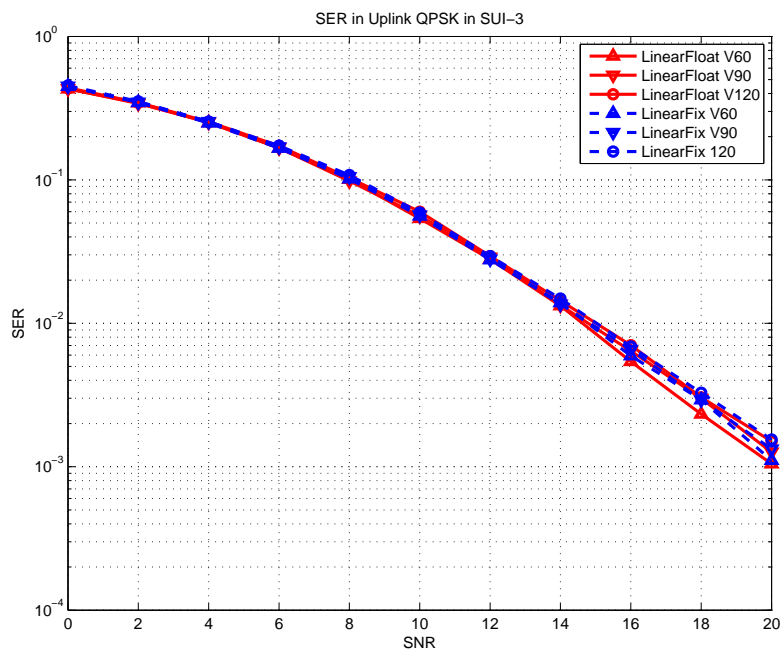


(b)

Figure 7.4: Uplink channel estimation performance under fixed- and floating-point computation in SUI-2 channel. (a) MSE. (b) SER.

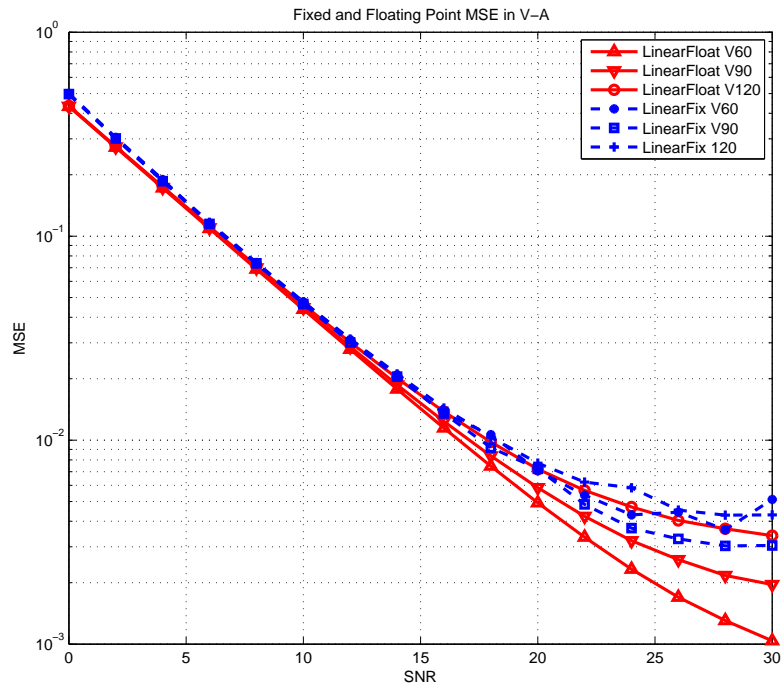


(a)

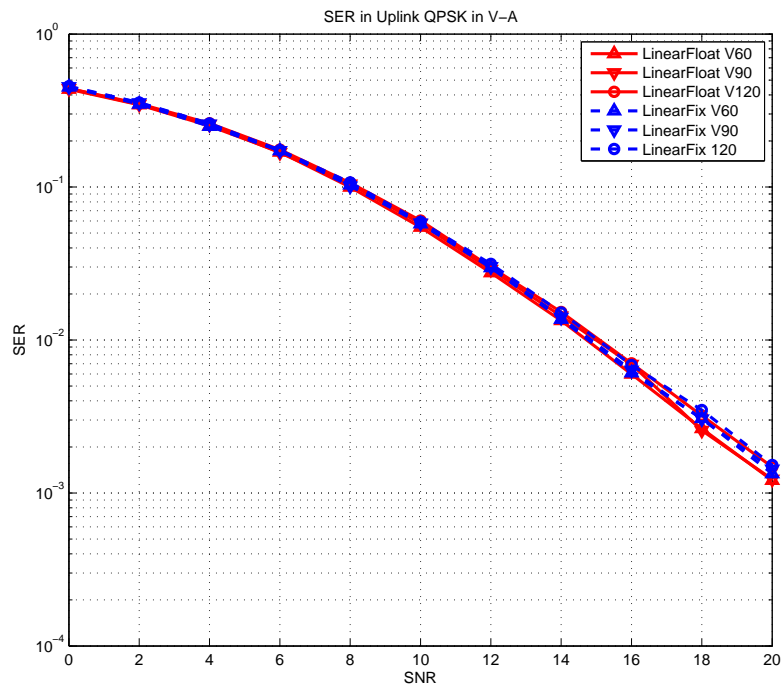


(b)

Figure 7.5: Uplink channel estimation performance under fixed- and floating-point computation in SUI-3 channel. (a) MSE. (b) SER.

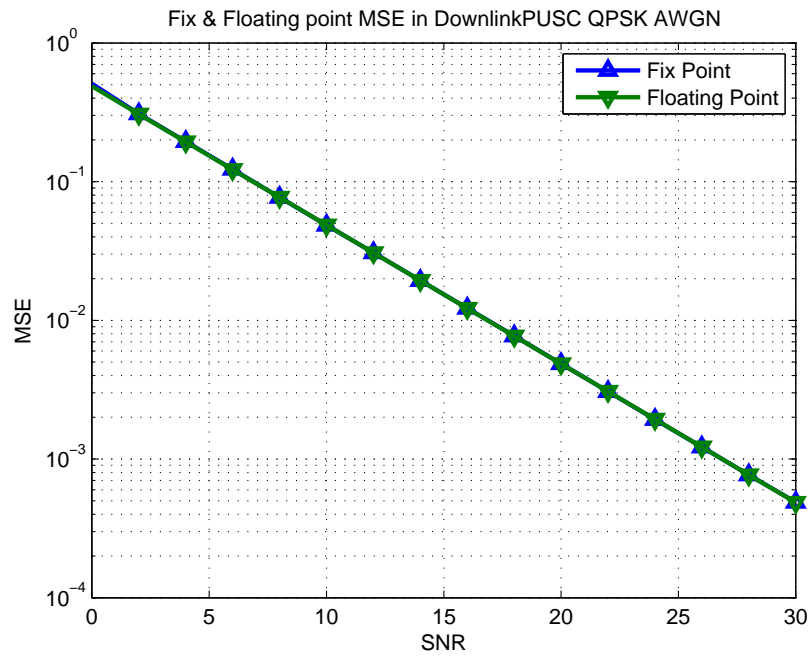


(a)

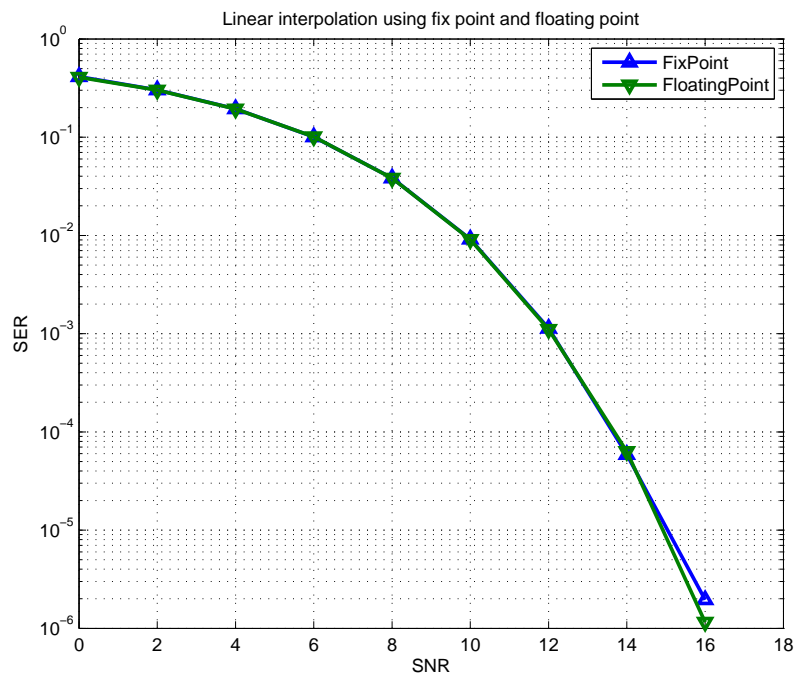


(b)

Figure 7.6: Uplink channel estimation performance under fixed- and floating-point computation in Vehicular A channel. (a) MSE. (b) SER.

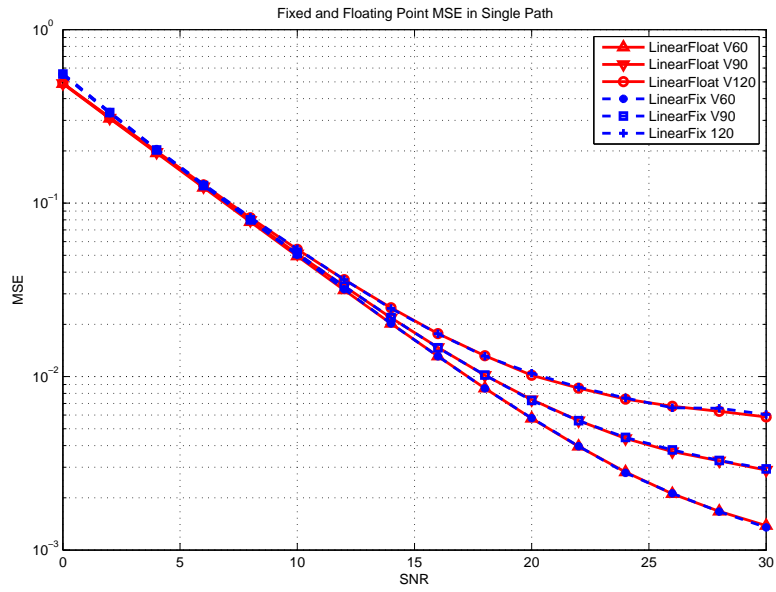


(a)

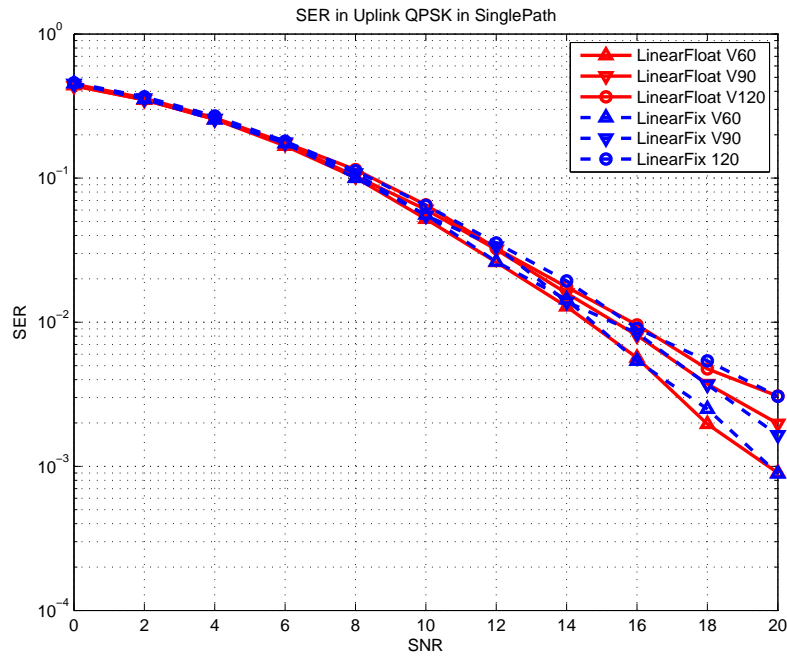


(b)

Figure 7.7: MSE and SER under fixed- and floating-point computation in AWGN. (a) MSE. (b) SER.

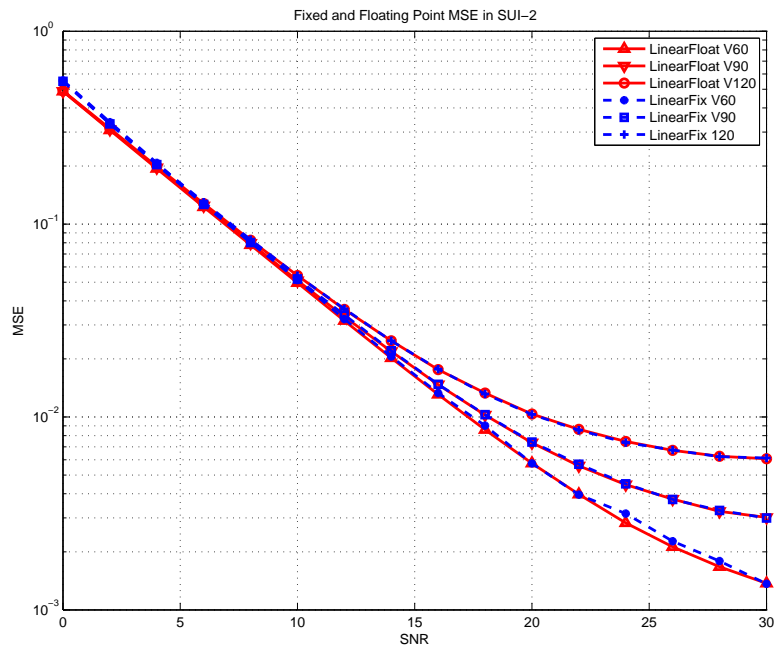


(a)

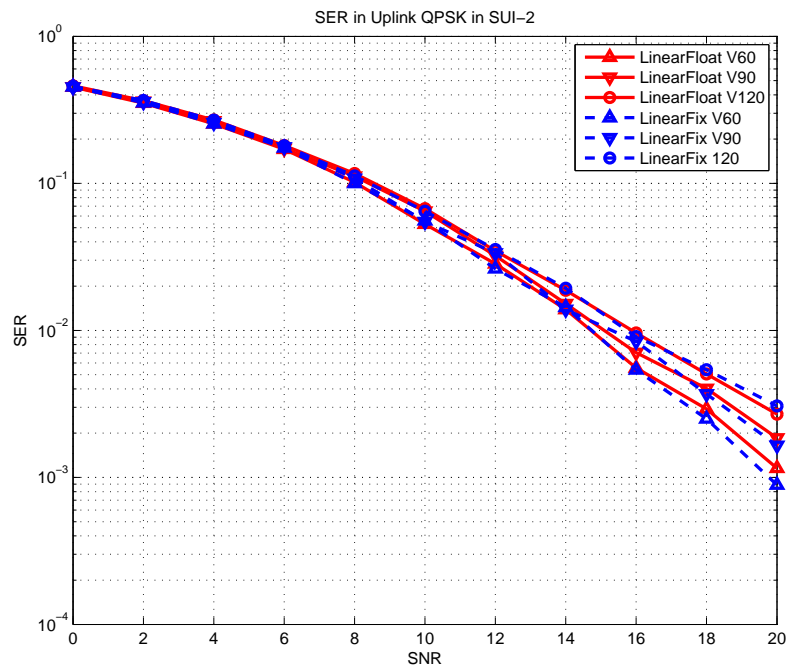


(b)

Figure 7.8: Downlink channel estimation performance under fixed- and floating-point computation in single-path Rayleigh fading. (a) MSE. (b) SER.

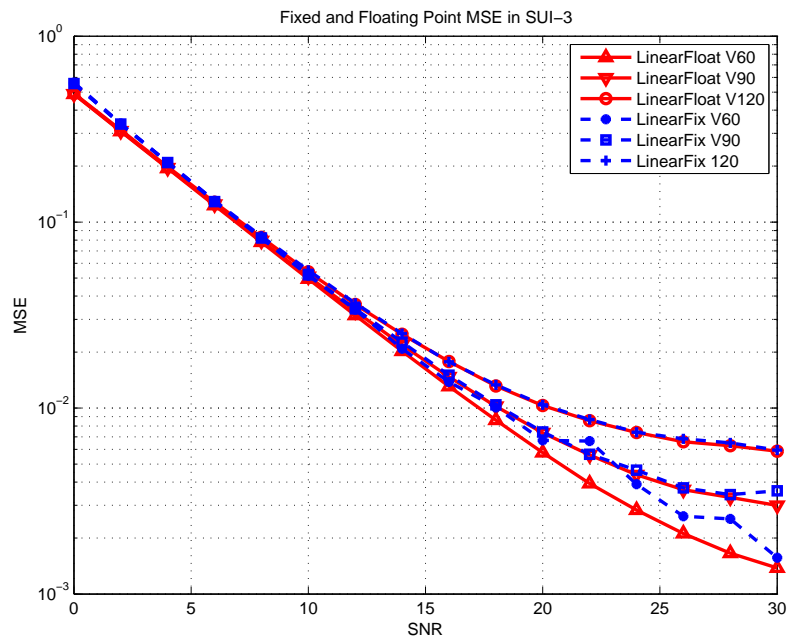


(a)

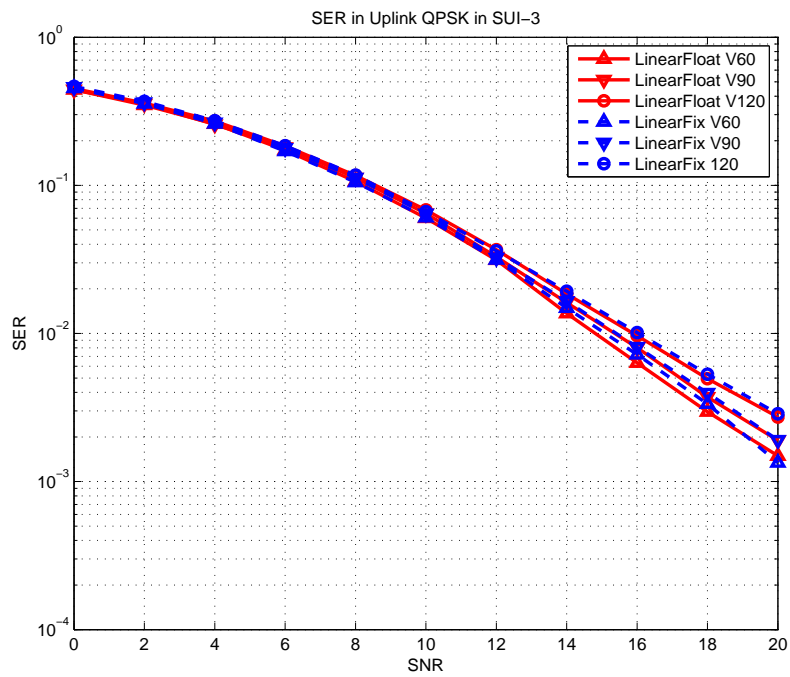


(b)

Figure 7.9: Downlink channel estimation performance under fixed- and floating-point computations in SUI-2 channel. (a) MSE. (b) SER.

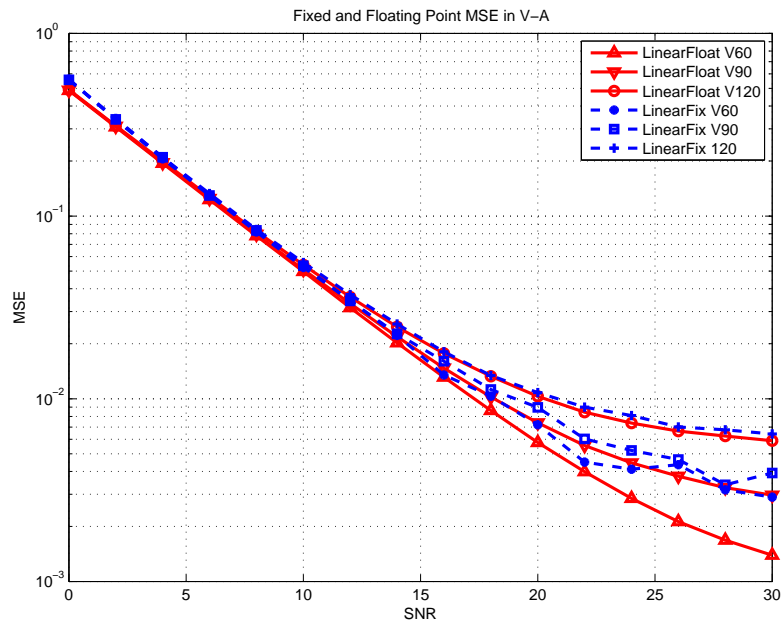


(a)

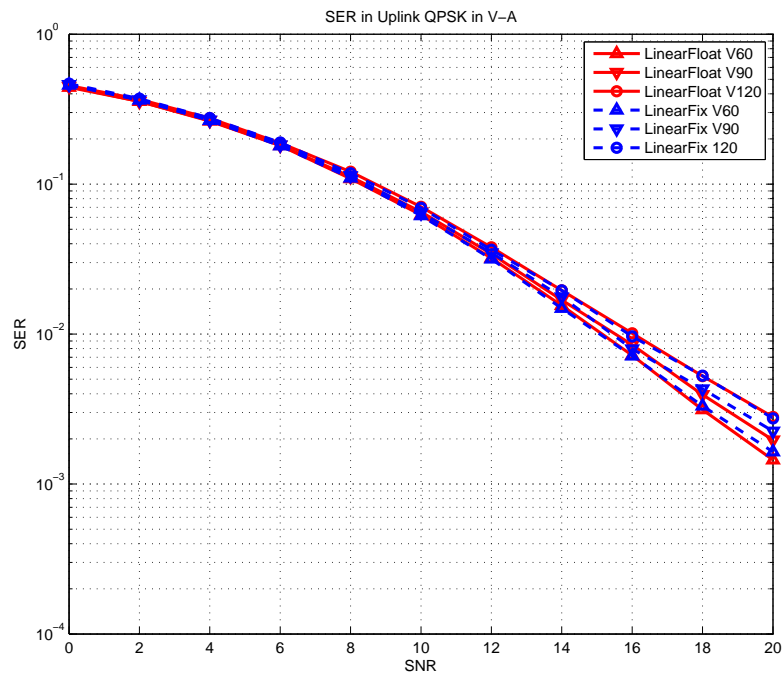


(b)

Figure 7.10: Downlink channel estimation performance under fixed- and floating-point computation in SUI-3 channel. (a) MSE. (b) SER.



(a)



(b)

Figure 7.11: Downlink channel estimation performance under fixed- and floating-point computation in Vehicular A channel (a) MSE. (b) SER.

structure without STC encoding. Table 7.1 shows the CCS simulation result in uplink.

Our DSP can execute 2 multiplications and 6 additions in one cycle. In the uplink channel estimation without using STC, the ideal case is one tile using ten additions (we ignore the multiplications and shifts because they can be executed with additions at the same time). Under STC, we need fifteen additions per tile, and we need to estimate two channels. So in the ideal case, the cycle count of linear interpolation under STC is three (*i.e.*, $\frac{15}{10} \times 2$) times that without using STC. In the simulation result, we can see the cycle count using STC 3.3 times without using STC, which is close to our expectation.

In the simulation, using linear interpolation needs 15450 cycles to complete the estimation job when executing on CCS. Since a tile spans 3 symbols, the cycle count averages to 5150 per symbol. The DSP we use, C6416T, has a 1 GHz processor clock with 32 MB DRAM. As the symbol time is 102.86 μsec , it amounts to approximately 0.050 of DSP computation load. In Wiener filter simulation, we do filtering with every subchannel separately. That means in ten subchannels with two antennas, we calculate complex matrix inverse 20 times, and each time needs 2754 cycles. In complex variable, the multiplication of three variables needs 16 multiplications (8 to calculate real part and 8 for imaginary part) and multiplication of four complex variable needs 20 multiplications. If we use the formula to solve the 4×4 matrix inverse, the determinant needs 24 terms of four variables multiplication, the adjoint matrix needs 16×6 terms of three variables multiplication, and need 16 multiplication to multiply the inverse of the determinant with the adjoint matrix. That means, one matrix inversion total needs about 2032 multiplications. Our DSP can execute 2 multiplications and 6 additions in one cycle. In the ideal case, the DSP needs about 1016 cycles. But in realistic it cost 2754 cycles, so the compile efficiency is about 0.36.

We also need to calculate the correlation with every data subcarrier. They both cause large computation load. In the simulation, the Wiener filtering requires 7.5 times more

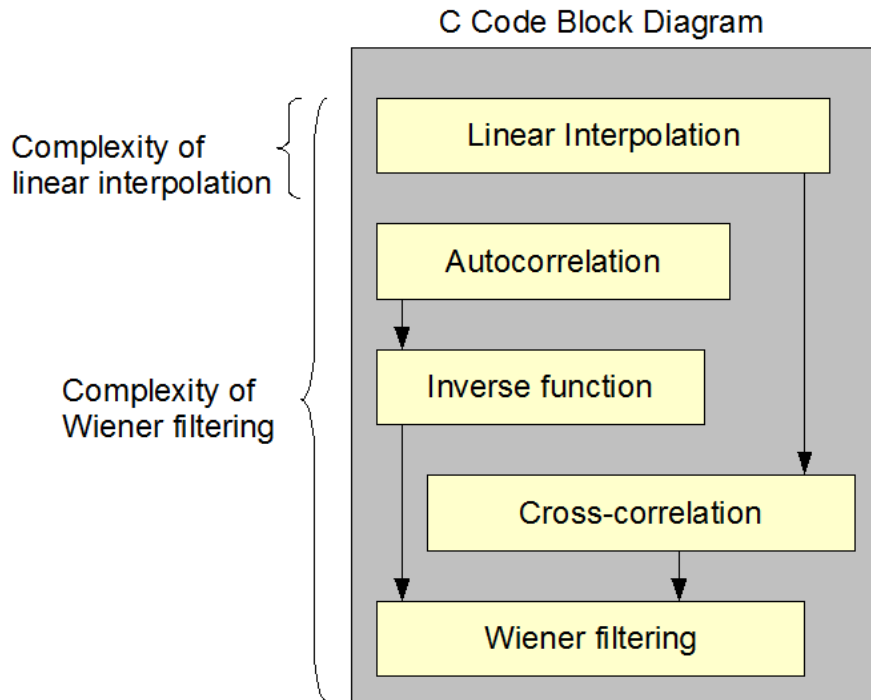


Figure 7.12: Wiener filtering C code block diagram.

Table 7.1: OFDMA Uplink DSP Load Under 1024-FFT with 10 Subchannel

Condition	Cycle count	DSP load factor for channel estimation
Linear interpolation (not using STC) [29]	4576	0.015
Linear interpolation (using STC)	15450	0.050
Wiener filtering (using STC)	131776	0.427

load compared to linear interpolation. Figure 7.12 shows the Wiener filtering C code block diagram. Since the Wiener filtering needs to use the result of linear interpolation, in the C code, the complexity of Wiener filtering contains that of linear interpolation.

Table 7.2 shows the DSP load in downlink channel estimation, in major group 0. There are 4 symbols in one STC downlink cluster. So using linear interpolation requires 0.041 of the DSP's computation power. We also compare linear interpolation and Wiener filtering.

Table 7.2: OFDMA Downlink DSP Load Under 1024-FFT, Major Group 0 with STC

Condition	Cycle count	DSP load factor for channel estimation
Linear interpolation	16903	0.041
Wiener filtering	88222	0.210

```

#define Q1_14 short
#define Q2_13 short
// Though same type (short), using different types to declare is much more clear
// and avoid some mistakes

#define ftoQ1_14(A) (short) (A*16384) // float to Q1.14 (2^14=16384)
#define ftoQ2_13(A) (short) (A*8192) // float to Q2.13 (2^13=8192)
#define ftoQ3_12(A) (short) (A*4096) // float to Q3.12 (2^12=4096)

#define Q1_14tof(A) (((float) A)/16384) // Q1.14 to float
#define Q2_13tof(A) (((float) A)/8192) // Q2.13 to float
#define Q3_12tof(A) (((float) A)/4096) // Q3.12 to float

```

Figure 7.13: *FIXED.H*.

We can see that for major group 0, it needs 5.2 times the computation of that in linear interpolation.

7.4 Program Code



Fig. 7.13 shows the header file *FIXED.H* which we use to transform floating-point data into fixed-point. Function *linear_interpolation* is the main function of channel estimation by linear interpolation in uplink. After receiving the first OFDMA symbol that defines a tile, it estimates the channel responses at the pilot subcarriers by using the LS technique, and buffers it, until it receives the next two tiles when it does linear interpolation and clears the buffer. The original code is shown in Fig. 7.14. Part of the corresponding assembly code of function *channel_estimation* is listed in Figures 7.15, where we can see the usage of registers in the DSP. Software pipelining information of the function is illustrated in Figure. 7.16 which shows the information of optimization by DSP compiler .

```

17 void interpolation(int N_pilot, int *pilot,
18                 Q2_13 *inv_pilot_mod,
19                 Q2_13 **tileRe,
20                 Q2_13 **tileIm,
21                 Q2_13 **respRe0,
22                 Q2_13 **respIm0,
23                 Q2_13 **respRe1,
24                 Q2_13 **respIm1,
25                 Q2_13 buffRe[][2][420],
26                 Q2_13 buffIm[][2][420])
27 {
28
29     Q2_13 dRe,dIm;
30     Q1_14 spacing;
31
32     int i=0;
33     int j=0;
34     spacing=ftoQ1_14(0.333333333);
35     for(i=0;i<N_pilot;i=i+2){
36         //Antenna0
37         dRe=(buffRe[1][1][i]-buffRe[0][1][i])*spacing>>14;
38         dIm=(buffIm[1][1][i]-buffIm[0][1][i])*spacing>>14;
39         respRe0[0][pilot[i]]=buffRe[0][1][i]+dRe;
40         respIm0[0][pilot[i]]=buffIm[0][1][i]+dIm;
41         respRe0[1][pilot[i]]=respRe0[0][pilot[i]]+dRe;
42         respIm0[1][pilot[i]]=respIm0[0][pilot[i]]+dIm;
43         respRe0[2][pilot[i]]=buffRe[1][1][i];
44         respIm0[2][pilot[i]]=buffIm[1][1][i];
45
46         dRe=(buffRe[2][0][i+1]-buffRe[1][0][i+1])*spacing>>14;
47         dIm=(buffIm[2][0][i+1]-buffIm[1][0][i+1])*spacing>>14;
48         respRe0[0][pilot[i+1]]=buffRe[1][0][i+1];
49         respIm0[0][pilot[i+1]]=buffIm[1][0][i+1];
50         respRe0[1][pilot[i+1]]=buffRe[1][0][i+1]+dRe;
51         respIm0[1][pilot[i+1]]=buffIm[1][0][i+1]+dIm;
52         respRe0[2][pilot[i+1]]=respRe0[1][pilot[i+1]]+dRe;
53         respIm0[2][pilot[i+1]]=respIm0[1][pilot[i+1]]+dIm;
54
55         for (j=0;j<3;j++)
56         {
57             dRe=(respRe0[j][pilot[i+1]]-respRe0[j][pilot[i]])*spacing>>14;
58             dIm=(respIm0[j][pilot[i+1]]-respIm0[j][pilot[i]])*spacing>>14;
59             respRe0[j][pilot[i+1]]=respRe0[j][pilot[i]]+dRe;
60             respIm0[j][pilot[i+1]]=respIm0[j][pilot[i]]+dIm;
61             respRe0[j][pilot[i+2]]=respRe0[j][pilot[i+1]]+dRe;
62             respIm0[j][pilot[i+2]]=respIm0[j][pilot[i+1]]+dIm;
63         }
64
65         //Antenna1
66         dRe=(buffRe[2][0][i]-buffRe[1][0][i])*spacing>>14;
67         dIm=(buffIm[2][0][i]-buffIm[1][0][i])*spacing>>14;
68         respRe1[0][pilot[i]]=buffRe[1][0][i];
69         respIm1[0][pilot[i]]=buffIm[1][0][i];
70         respRe1[1][pilot[i]]=buffRe[1][0][i]+dRe;
71         respIm1[1][pilot[i]]=buffIm[1][0][i]+dIm;
72         respRe1[2][pilot[i]]=respRe1[1][pilot[i]]+dRe;
73         respIm1[2][pilot[i]]=respIm1[1][pilot[i]]+dIm;
74
75         dRe=(buffRe[1][1][i+1]-buffRe[0][1][i+1])*spacing>>14;
76         dIm=(buffIm[1][1][i+1]-buffIm[0][1][i+1])*spacing>>14;
77         respRe1[0][pilot[i+1]]=buffRe[0][1][i+1]+dRe;
78         respIm1[0][pilot[i+1]]=buffIm[0][1][i+1]+dIm;
79         respRe1[1][pilot[i+1]]=respRe1[0][pilot[i+1]]+dRe;
80         respIm1[1][pilot[i+1]]=respIm1[0][pilot[i+1]]+dIm;
81         respRe1[2][pilot[i+1]]=buffRe[1][1][i+1];
82         respIm1[2][pilot[i+1]]=buffIm[1][1][i+1];
83
84         for (j=0;j<3;j++)
85         {
86             dRe=(respRe1[j][pilot[i+1]]-respRe1[j][pilot[i]])*spacing>>14;
87             dIm=(respIm1[j][pilot[i+1]]-respIm1[j][pilot[i]])*spacing>>14;
88             respRe1[j][pilot[i+1]]=respRe1[j][pilot[i]]+dRe;
89             respIm1[j][pilot[i+1]]=respIm1[j][pilot[i]]+dIm;
90             respRe1[j][pilot[i+2]]=respRe1[j][pilot[i+1]]+2*dRe;
91             respIm1[j][pilot[i+2]]=respIm1[j][pilot[i+1]]+2*dIm;
92         }
93     }
94 }
95

```

Figure 7.14: *linear_interpolation*.


```

*****
* FUNCTION NAME: interpolation(int, int *, short *, short *, short *, short *, short *, short *, short *, short *) [2][420], short (*) [2]
*
* Regs Modified : A0,A3,A4,A5,A6,A7,A8,A9,B0,B2,B4,B5,B6,B7,B8,B9,A16,*
*               A17,A18,A19,A20,A21,A22,A23,A24,A31,B16,B17,B18,*
*               B19,B20,B21,B22,B23,B24,B25,B26,B27,B28,B29,B30,*
*               B31,*
* Regs Used : A0,A3,A4,A5,A6,A7,A8,A9,A10,A12,B0,B2,B3,B4,B5,B6,B7,*
*            B8,B9,B10,DP,SP,A16,A17,A18,A19,A20,A21,A22,A23,*
*            A24,A31,B16,B17,B18,B19,B20,B21,B22,B23,B24,B25,*
*            B26,B27,B28,B29,B30,B31,*
* Local Frame Size : 0 Args + 0 Auto + 0 Save = 0 byte
*****
*****
*
* Using -g (debug) with optimization (-o3) may disable key optimizations!
*
*****
interpolation_FiPiPsPPsN54PA2_A420_sPA2_A420_s:
**-----**
    .dwcfi cfa_offset, 0
    .dwcfi save_reg_to_reg, 228, 19
$CSDW$539 .dwtag DW_TAG_formal_parameter, DW_AT_name("N_pilot")
    .dwattr $CSDW$539, DW_AT_TI_symbol_name("N_pilot")
    .dwattr $CSDW$539, DW_AT_type(*$CSDW$T$10)
    .dwattr $CSDW$539, DW_AT_location[DW_OP_reg4]
$CSDW$540 .dwtag DW_TAG_formal_parameter, DW_AT_name("pilot")
    .dwattr $CSDW$540, DW_AT_TI_symbol_name("pilot")
    .dwattr $CSDW$540, DW_AT_type(*$CSDW$T$101)
    .dwattr $CSDW$540, DW_AT_location[DW_OP_reg20]
$CSDW$541 .dwtag DW_TAG_formal_parameter, DW_AT_name("inv_pilot_mod")
    .dwattr $CSDW$541, DW_AT_TI_symbol_name("inv_pilot_mod")
    .dwattr $CSDW$541, DW_AT_type(*$CSDW$T$103)
    .dwattr $CSDW$541, DW_AT_location[DW_OP_reg6]
$CSDW$542 .dwtag DW_TAG_formal_parameter, DW_AT_name("tileRe")
    .dwattr $CSDW$542, DW_AT_TI_symbol_name("tileRe")
    .dwattr $CSDW$542, DW_AT_type(*$CSDW$T$105)
    .dwattr $CSDW$542, DW_AT_location[DW_OP_reg22]
$CSDW$543 .dwtag DW_TAG_formal_parameter, DW_AT_name("tileIm")
    .dwattr $CSDW$543, DW_AT_TI_symbol_name("tileIm")
    .dwattr $CSDW$543, DW_AT_type(*$CSDW$T$105)

```

Figure 7.15: Part of assembly code of function *linear_interpolaton*.

```

10377 ;*-----*
10378 ;* SOFTWARE PIPELINE INFORMATION
10379 ;*
10380 ;* Loop source line      : 55
10381 ;* Loop opening brace source line : 56
10382 ;* Loop closing brace source line : 63
10383 ;* Known Minimum Trip Count   : 3
10384 ;* Known Maximum Trip Count   : 3
10385 ;* Known Max Trip Count Factor : 3
10386 ;* Loop Carried Dependency Bound(*) : 59
10387 ;* Unpartitioned Resource Bound : 8
10388 ;* Partitioned Resource Bound(*) : 9
10389 ;* Resource Partition:
10390 ;*           A-side B-side
10391 ;* .L units      0  0
10392 ;* .S units      1  2
10393 ;* .D units      8  8
10394 ;* .M units      0  2
10395 ;* .X cross paths    2  7
10396 ;* .T address paths    8  8
10397 ;* Long read paths    0  0
10398 ;* Long write paths   0  0
10399 ;* Logical ops (.LS)   0  2  (.L or .S unit)
10400 ;* Addition ops (.LSD) 6  13 (.L or .S or .D unit)
10401 ;* Bound(.L .S .LS)   1  2
10402 ;* Bound(.L .S .D .LS .LSD) 5  9*
10403 ;*
10404 ;* Searching for software pipeline schedule at ...
10405 ;*   ii = 59 Schedule found with 1 iterations in parallel
10406 ;* Done
10407 ;*
10408 ;* Collapsed epilog stages : 0
10409 ;* Collapsed prolog stages : 0
10410 ;*
10411 ;* Minimum safe trip count : 1
10412 ;*-----*
10413 ;*$C$L135: ; PIPED LOOP PROLOG
10414 ;**-----*
10415 ;*$C$L136: ; PIPED LOOP KERNEL

```

Figure 7.16: Software pipelining information of function *linear_interpolation*.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this thesis, we presented several channel estimation methods for OFDMA STC uplink and downlink. To do channel estimation, first we use LS estimator to estimate the channel frequency response on pilot subcarriers. Then we used two different methods to calculate the channel response of entire cluster or tile. One is linear interpolation and the other one is Wiener filtering. In the linear interpolation, we demonstrated the performance in different channel conditions. In Wiener filtering, we chose different sets of samples to calculate autocorrelation and cross-correlation that causes different performance in our simulation. If we chose proper sets of samples to average, then the performance would be better than the linear interpolation. After applying these methods on the MIMO system, we could get better performance than the single antenna.

For DSP implementation, we replaced all the operations into 16-bit fixed point operation and implement the MIMO structure of IEEE 802.16e. In the uplink linear interpolation, we could see that the DSP computation load of using STC need 3.3 times that without using STC. Since the Wiener filtering is not suitable for fixed-point format, but we also compared the computation load. By calculating DSP load, we could see the Wiener filtering need large

amount of computation than linear interpolation.

8.2 Potential Future Work

There are several possible extension for our research:

- Construct MMSE error model to discuss how to improve the performance.
- Try other kinds of techniques to estimate channel response that more suitable for MIMO system.
- Optimize the performance on DSP.
- In this thesis, we do not consider the influence of intercarrier interference (ICI). The ICI simulation can be involved in the future.
- Use model base method to calculate cross-correlation to improve Wiener filtering.

If we want to use model base method, the first thing we want to know is the maximum Doppler frequency of the mobile station. In [30] shows the method that using pilots to calculate velocity. Since we know the time domain correlation is

$$r_t(l) = J_0(2\pi f_{max}lT_s) \quad (8.1)$$

then we can calculate the pilot correlation in time domain, and find out which neighbor two pilots correlation cause zero crossing and the time cause zero crossing called T_0 . Since The smallest positive zero crossing point of the Bessel function $J_0(x)$ occurs at $x = 2.405$. An estimation for the user maximum Doppler frequency can be obtained by

$$f_{max} = \frac{2.405}{2\pi T_0}. \quad (8.2)$$

But its defect is that it needs long time buffer to get real time domain correlation. Besides time domain correlation, we also need to know frequency domain correlation. As mention in chpter 3, for an exponentially decaying multipath power delay profile, the frequency domain correlation is

$$r_f(k) = \frac{1}{1 + j2\pi\tau_{rms}k/T}. \quad (8.3)$$

In [31] it mention the meathod of calculating τ_{rms} by using pilots.

By utilizing above two methods, we might improve the Wiener filtering by using model base method.



Bibliography

- [1] Hongxiang Li and Hui Liu, “An analysis on uplink OFDMA optimality,” in *Proc. IEEE Veh. Technol. Conf.*, vol. 3, 2006, pp. 1339–1343.
- [2] Liangshan Ma and Dongyan Jia, “The competition and cooperation of WiMAX, WLAN and 3G, in ” *Inter. Conf. Applica. Sys., Mobile Tech.* Nov. 2005, pp. 1–5.
- [3] S. M. Alamouti, “A simple transmit diversity technique for wireless communications,” *IEEE. Selected Areas Commun.*, vol. 16, pp. 1451–1458, Oct. 1988.
- [4] Man-On Pun, Michele Morelli, and C.-C. Jay Kuo, “Maximum-likelihood synchronization and channel estimation for OFDMA uplink transmissions,” *IEEE Trans. Commun.*, vol. 54, no. 4, pp. 726–736, Apr. 2006.
- [5] Lior Eldar, M. R. Raghavendra, S. Bhashyam, Ron Bercovich, and K. Giridhar, “Parametric channel estimation for pseudo-random user-allocation in uplink OFDMA,” in *IEEE Int. Conf. Commun.*, vol. 7, 2006, pp. 3035–3039.
- [6] 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.
- [7] 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 Broad-*

band 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.

- [8] B. Muquet, E. Biglieri, A. Goldsmith, and H. Sari, “MIMO techniques for Mobile WiMAX systems,” SEQUANS Communications White Paper, September 2006.
- [9] O. Edfors, M. Sandell, J. J. van de Beek, D. Landstrom, and F. Sjöberg, “An introduction to orthogonal frequency-dicision multiplexing,” <http://courses.ece.uiuc.edu/ece459/spring02/ofdmtutorial.pdf>.
- [10] M.-H. Hsieh, “Synchronization and channel estimation techniques for OFDM systems,” Ph.D. dissection, Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C., May 1998.
- [11] 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 Veh. Technol. Conf.*, Apr. 1996, pp. 923–927.
- [12] 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.
- [13] 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.
- [14] P. Hoeher, S. Kaiser, and P. Robertson, “Two-dimensional pilot-symbol-aided channel estimation by Wiener filtering,” in *IEEE Int. Conf. Acoust. Speech Signal Process.*, Apr. 1997, pp. 1845–1848.

- [15] R. Nee and R. Prasad, *OFDM for Wireless Multimedia Communications*. Norwell, Mass.: Artech House, 2000.
- [16] F. Said and H. Aghvami, "Linear two dimensional pilot assisted channel estimation for OFDM systems," in *IEE Conf. Telecommunications*, Edinburgh, Scotland, Apr. 1998, pp. 32V-36.
- [17] V. Erceg *et al*, "Channel models for broadband fixed wireless systems," IEEE 802.16.3c-00/53.
- [18] E. Bartsch, I. Wassell, and M. Sellars, "Equalization requirement study for broadband MMDS wireless access systems," presented at *Int. Symp. Communications*, Tainan, Taiwan, R.O.C., 2001.
- [19] ETSI, "Selection procedure for the choice of radio transmission technologies of the UMTS," ETSI tech. rep. TR 101 112, V3.0.2, pp. 38–43, Apr. 1994.
- [20] P. Dent, G. E. Bottomley, and T. Croft, "Jakes fading model revisited," *Electron. Lett.*, vol. 29, no. 13, pp. 1162–1163, June 1993.
- [21] Texas Instruments, *TMS320C6000 CPU and Instruction Set*. Literature number SPRU189F, Oct. 2000.
- [22] Texas Instruments, *TMS320C6000 DSP Cache Users Guide*. Literature number SPRU656A, May. 2003.
- [23] 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.
- [24] Texas Instruments, *Code Composer Studio User's Guide*. Literature number SPRU328B, Feb. 2000.

- [25] Texas Instruments, *TMS320C6000 Code Composer Studio Getting Started Guide*. Literature number SPRU509D, Aug. 2003.
- [26] Texas Instruments, *TMS320C64x DSP Library Programmer's Reference*. Literature number SPRU565B, Oct. 2003.
- [27] Texas Instruments, *TMS320C6000 Programmer's Guide*. Literature number SPRU198G, Oct. 2002.
- [28] 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.
- [29] Yi Ling Wang, "Reserch in and DSP Implementation of Channel Estimation Techniques for IEEE 802.16e OFDMA Uplink and Downlink," M.S. thesis, Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C., June 2006.
- [30] H. Schober and F. Jondral, "Velocity estimation for OFDM based communication systems," in *Veh. Technol. Conf.*, vol.2, fall 2002, pp. 715–718.
- [31] Kun-Chien Hung, "Digital signal processing algorithms for communication receives: synchronizatoin, equalization, and channel estimation," Ph.D. dissertation, Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C., Oct. 2008.

作者簡歷

姓名：余光中 (Kuang-Chung Yu)

生日：1984 年 4 月 11 日

出生地：台北市

學歷：交通大學電子工程系學士(2002.9~2006.6)

交通大學電子研究所碩士(2006.9~2008.11)

研究領域：通訊系統及數位訊號處理

論文題目：IEEE 802.16e OFDMA 多輸入輸出通道估測技術之探討與數位訊號處理器實現

(Study in IEEE 802.16e OFDMA MIMO Channel Estimation Techniques and Associated Digital Signal Processor Implementation)