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 802.16m 線性最小均方差通道估測之數位訊號處理器實現

Digital Signal Processor Implementation of LMMSE Channel Estimation for IEEE 802.16m

研 究 生：張智凱

指導教授：林大衛 博士

中 華 民 國 一 ０ ０ 年 八 月

IEEE 802.16m 線性最小均方差通道估測之數位訊號處理器實現

Digital Signal Processor Implementation of LMMSE Channel Estimation for IEEE 802.16m

研究生：張智凱　　　　Student: Chih-Kai Chang

指導教授：林大衛 博士　　　　Advisor: Dr. David W. Lin

國 立 交 通 大 學

電子工程學系　　電子研究所碩士班

碩士論文

A Thesis
Submitted to Department of Electronics Engineering & Institute of Electronics
College of Electrical and Computer Engineering
National Chiao Tung University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in
Electronics Engineering
August 2011
Hsinchu, Taiwan, Republic of China

中華民國一００年八月

IEEE 802.16m 線性最小均方差通道估測之數位訊號處理器實現

研究生：張智凱　　　　　　　　　　指導教授：林大衛 博士

國立交通大學

電子工程學系　電子研究所碩士班

摘要

本篇論文介紹 IEEE 802.16m 正交分頻多工存取(OFDMA)於上行傳輸以及下行傳輸的通道估測部分，並採用空間頻率區塊碼(SFBC)以對抗抗通道衰減，之後再將其引入 DSP 定點運算並觀察分析其效能。

本篇論文採用線性最小均方差的通道估測技術。首先我們先使用最小平方差的估測器估計在導訊(pilot)上的通道響應，再利用導訊上的通道響應使用線性內插法得到其餘導訊位置上的通道響應。之後估測延遲參數，再利用指數函數的功率延遲曲線求出相關函數，最後利用相關函數作線性最小均方差估測資料載波上的通道響應。

本篇論文可大致分為三個部份。第一部份我們先簡介 IEEE 802.16m 於上下行傳輸的相關規格制定。第二部份我們介紹最小均方差演算法並驗證其採用浮點運算於 AWGN 通道以及多重路徑 SUI 通道上的效能。第三部分我們介紹 DSP 晶片實現環境並修改浮點運算程式碼為定點運算，然後針對定點運算程式碼進行優化及加速的動作。最後再分析其在 AWGN 及多重路徑 SUI 通道上的效能。

Digital Signal Processor Implementation of LMMSE Channel Estimation for IEEE 802.16m

Student：Chih-Kai Chang　　　　Advisor：Dr. David W. Lin

Department of Electronics Engineering

Institute of Electronics

National Chiao Tung University

Abstract

In this thesis, we discuss about IEEE 802.16m OFDMA uplink and downlink channel estimation and adopt SFBC (special frequency block code) technique to resistant the channel fading. After that we use DSP fixed-point operation to analysis the performance.

We present LMMSE channel estimation method in this thesis. First, we use LS estimator on pilot subcarrier to get the channel response. And then we use linear interpolation method to get channel response on other pilot position. After that we estimate the delay parameters and find correlation function associate exponential PDP. Finally base on this correlation function, do LMMSE filtering to estimate the data subcarrier response.

This thesis can be separated into three parts. First we introduce about the standard of IEEE 802.16m OFDMA uplink and downlink transmission. Second, we introduce LMMSE algorithm and validate on AWGN and SUI multipath channel to analysis the performance on floating-point condition. Third we introduce DSP implementation environment and modify the floating-point C code to fixed-point. And then we against the fixed-point C code to accelerate and optimization. Finally we do the simulation on AWGN and SUI multipath channel to analysis the performance.

誌謝

經過層層的審核以及編排，本篇論文終於順利完成，在這邊首先要特別感謝我的指導教授林大衛博士不厭其煩的幫我修改文法以及編排方式，讓整篇論文能看起來更加的流暢完整。除了論文的部分外，林大衛老師這兩年來細心以及不厭其煩的指導令我獲益良多，其中又以老師嚴格自我要求的精神更深遠的影響了我。

此外，在這兩年的日子當中通訊電子與訊號處理實驗室的所有成員們，包含各位師長、同學、學長姐以及學弟妹們，感謝你們讓我在這兩年的時光充滿活力以及歡笑。在這邊感謝曲建全學長、林鴻志學長、蕭世璞學長、洪朝雄學長、蔡彰哲學長、邱頌恩學長、吳思賢學長和盧世榮學長對我在學術上的不吝指導與建議，謝謝你們幫我解決了許多學業方面的疑問。感謝 98 級威宇、曉盈、卓翰、俊言、強丹、兆軒、郁婷、琬瑜、怡茹、頌文、書緯、偵源、凱翔、復凱及 99 級學弟們等實驗室成員，平日和我一起念書、討論、玩耍，讓我的研究生涯擁有美好的回憶。

最後，必須感謝我的家人，我的父母親，讓我在落魄失意的時候能有溫暖的避風港。還有我高中時代的好友們，謝謝你們在任何時候給我的所有關心。有你們給予我的支持，使我在外地讀書時能無後顧之憂。感謝以上所有人的支持，也謝謝所有幫助過我、陪我走過這段歲月的人

張智凱

民國一百年八月　於新竹

Contents

List of Figures

List of Tables

Chapter 1

Introduction

ITU-R defined the criterion of the fourth-generation (4G) mobile communication standard IMT-Advanced formally in June 2003, which specifies that the minimum total data rate in a cell is to be 100 Mbps in the environment with high mobility and 1 Gbps in the static environment [1]. In this thesis, we consider the IEEE 802.16m system in time-division duplex (TDD) mode, where downlink (DL) and uplink (UL) transmissions are time multiplexed in each frame. Our study focuses on the digital signal processor (DSP) implementation of a linear minimum mean-square error (LMMSE) channel estimation technique for IEEE 802.16m. The technique was proposed in [2]. Below, we follow [3] to introduce the pilot-aided LMMSE channel estimation problem.

Pilot-aided channel estimation is widely employed in today's coherent wireless orthogonal frequency-division multiplexing (OFDM) systems. The subcarriers that carry pilot signals are usually dispersed in frequency and in time. The LMMSE technique is also known as Wiener filtering. Given some initial channel estimates at the pilot subcarriers, the LMMSE channel estimate at any subcarrier d is given by [4], [5]

$$\hat{h}_d = \mathbf{w}_d^H \mathbf{h}_p \tag{1.1}$$

with

$$\mathbf{w}_d = (\mathbf{R}_{pp} + \sigma_n^2 \mathbf{I})^{-1} \mathbf{r}_{dp} \tag{1.2}$$

where h_d is the desired channel estimate; $\mathbf{w}_d$ is the Wiener filter; superscript H denotes Hermitian transpose; $\mathbf{h}_p$ is the vector of given initial channel estimates; $\mathbf{R}_{pp} = E(\mathbf{h}_p \mathbf{h}_p^H)$, with E denoting expectation and $\mathbf{h}_p$ being the vector of true channel responses at pilot carriers; $\mathbf{r}_{dp} = E(\mathbf{h}_p \mathbf{h}_d^H)$ with $\mathbf{h}_d$ being the true channel response at subcarrier d; σ_n^2 is the variance of additive noise in $\mathbf{h}_p$, assumed white Gaussian (i.e., AWGN); and $\mathbf{I}$ denotes an identity matrix. A usual method to estimate pilot response is called least-square (LS) method, which divides the received signal at each pilot carrier by the known pilot value there to obtain the pilot response.

To carry out the LMMSE estimation, we should know $\mathbf{R}_{pp}$, $\mathbf{r}_{dp}$, and σ_n^2. The estimation of σ_n^2 can be achieved by measuring the received power at the null subcarriers but the values of $\mathbf{R}_{pp}$ and $\mathbf{r}_{dp}$ need more work. One aspect of the problem has to do with the fact that an accurate estimate requires averaging over sufficiently many samples. But when the channel is time-varying, there may not be enough data point within a coherence time to facilitate this, especially if the system only transmits a small number of pilots. Another aspect of the problem of estimating $\mathbf{r}_{dp}$ is that usually requires interpolation (which implies approximation) in addition to averaging. The problem in estimating $\mathbf{R}_{pp}$ and $\mathbf{r}_{dp}$ makes strict-sense LMMSE channel estimation impractical in many cases.

To sidestep the above problem, reference [3] proposed a method to estimate the power-delay profile (PDP). Taking the Fourier transform of the PDP give the correlation function in frequency domain as.

$$\mathbf{R}_{Hf}(f) \triangleq \mathcal{F}[P_h(\tau)]. \tag{1.3}$$

For simplicity, we can employ a simple model for the channel PDP. One such choice is the exponential model [6]–[8], for which the entire second-order channel statistics are defined by the mean delay τ_μ and the root-mean-square (RMS) delay spread τ_{rms}. Given τ_μ and τ_{rms} as well as σ_n^2, one can calculate $\mathbf{R}_{pp}$ and $\mathbf{r}_{dp}$ and then calculate the Wiener filter $\mathbf{w}_d$. The price paid is that the PDP model employed may be an oversimplification of the actual situation, which results in modeling error. But [3] shows that the exponential PDP can yield good performance for LMMSE channel estimation and is suitable for various pilot-transmitting OFDM signal structures.

1.1 Constitution and Thesis Organization

In this thesis, we use the method of above-mentioned for downlink (DL) and uplink (UL) channel estimation in IEEE 802.16m and implement on digital signal processor (DSP) platform. In chapter 2, we introduce the IEEE 802.16m OFDMA downlink and uplink specification,respectively. In chapter 3, we introduce the channel estimation methods. In chapters 4, we discuss the performance of channel estimation methods by floating point implementation for downlink and uplink. In chapter 5, we describe the DSP implementation platform. In chapter 6, we introduce fixed point implementation and optimization methods. In chapter 7, we discuss the performance of channel estimation methods by fixed point implementation for downlink and uplink. At last, we give the conclusion and discuss some potential future work in chapter 8.

Chapter 2

Introduction to IEEE 802.16m OFDMA

The IEEE 802.16m standard is based on orthogonal frequency division multiplexing (OFDM) and orthogonal frequency division multiple access (OFDMA). The IEEE 802.16m Task Group intends to complete the first version of the standard in 2011. In this chapter, we first introduce some basic concepts regarding OFDM and OFDMA. Then we give an overview of the ninth draft of the IEEE 802.16m standard dated Oct 6, 2010, which we have used as the basis of the present work. For the sake of simplicity, we only introduce the specifications that are most relevant to our study.

2.1 Overview of OFDMA [9], [10]

OFDMA is considered one most appropriate scheme for future wireless systems, including fourth-generation (4G) broadband wireless networks. In an OFDMA system, users simultaneously transmit their data by modulating mutually exclusive sets of orthogonal subcarriers, so that each user's signal can be separated in the frequency domain. One typical structure is subband OFDMA, where all available subcarriers are divided into a number of subbands and each user is allowed to use one or more available subbands for the data transmission.

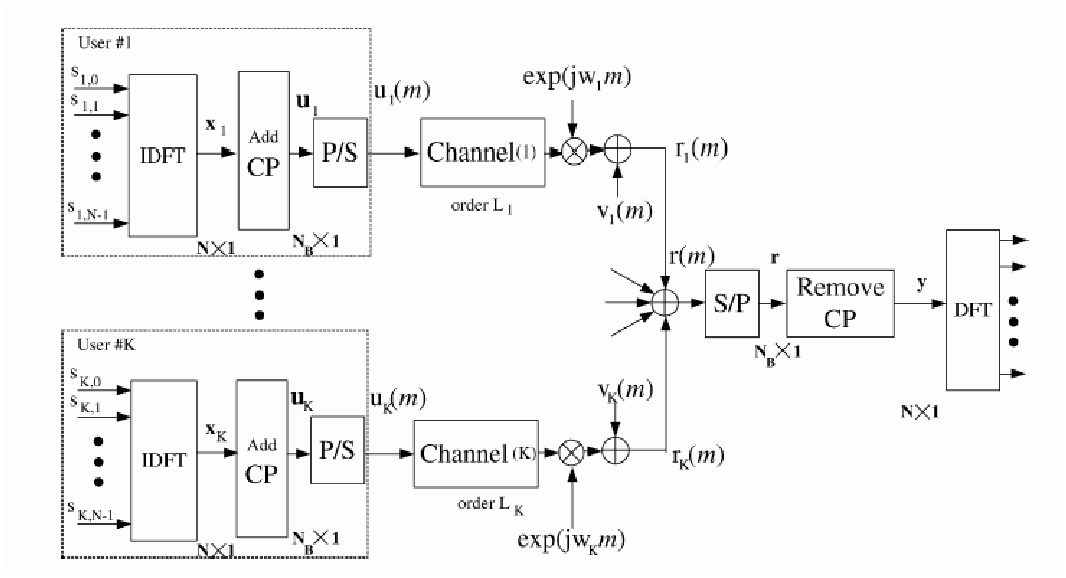


Figure 2.1: Discrete-time model of the baseband OFDMA system (from [9]).

Usually, pilot symbols are employed for the estimation of channel state information (CSI) within the subband. IEEE 802.16e and 802.16m are examples of such systems. Figure 2.1 shows an OFDMA system in which active users simultaneously communicate with the base station (BS).

2.1.1 Cyclic Prefix

Cyclic prefix (CP) or guard time is used in OFDM and OFDMA systems to overcome the intersymbol and interchannel intercarrier problems. The multiuser channel is usually assumed to be substantially invariant within one-block (or one-symbol) duration. The channel delay spread plus symbol timing mismatch is usually assumed to be smaller than the CP duration. In this condition, users do not interfere with each other in the frequency domain, when there is proper time and frequency synchronization.

A CP is a copy of the last part of the OFDMA symbol (see Fig. 2.2). A copy of the last T_g segment of the useful symbol period is used to collect multipaths while maintaining the

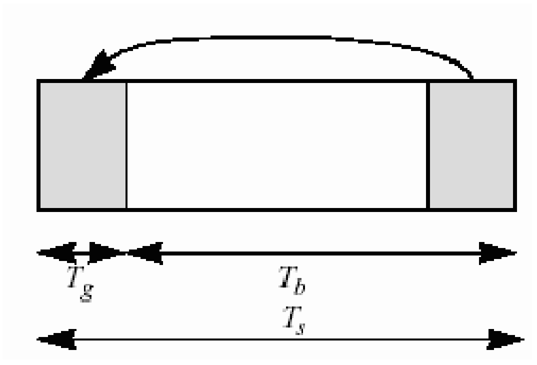


Figure 2.2: OFDMA symbol time structure (Fig. 428 in [11]).

orthogonality of subcarriers. However, the transmitter energy increases with the length of the guard time while the receiver energy remains the same, as the CP is discarded in the receiver. So there is a $10\log_{10}(1 - T_g/(T_b + T_g))$ loss in E_b/N_0.

2.1.2 Discrete-Time Baseband Equivalent System Model

The material in this subsection is mainly taken from [10]. 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 according to their needs.

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

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

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

$$\mathbf{H} = \mathbf{F}\mathbf{h} \tag{2.2}$$

where $\mathbf{H} = [H_0, H_1, ..., H_{K-1}]^T$ is the vector of channel frequency response, $\mathbf{h} = [h_0, ..., h_{L-1}, 0, ..., 0]^T$

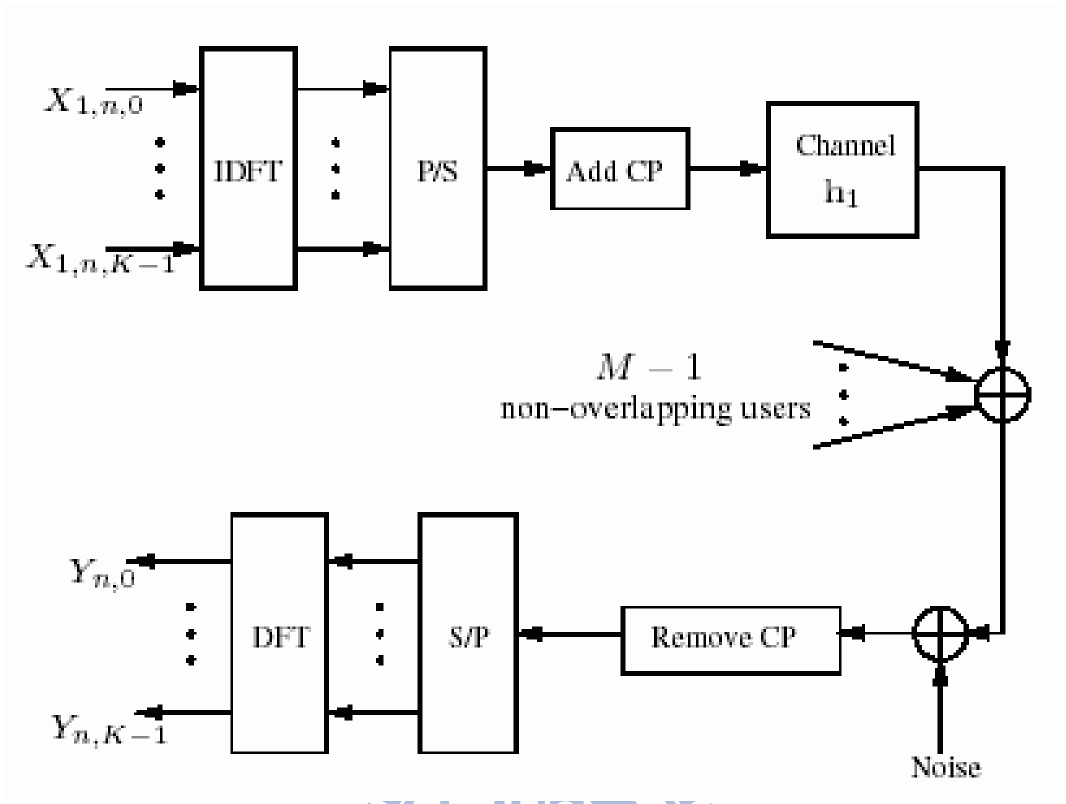


Figure 2.3: Discrete-time baseband equivalent of an OFDMA system with M users (from [10]).

and $\mathbf{F}$ is the K-point discrete fourier transform (DFT) matrix. The impulse response length l_{L-1} is upper bounded by the length of CP (L_{cp}). The received signal in the frequency domain for a certain OFDM symbol is given by

$$\mathbf{Y}_n = \sum_{i=1}^{M} \mathbf{X}_{i,n}\mathbf{H}_{i,n} + \mathbf{V}_n \tag{2.3}$$

where $\mathbf{X}_{i,n} = \text{diag}(X_{i,n,0}, ..., X_{i,n,K-1})$ is the $K \times K$ diagonal data matrix corresponding to ith user in the nth OFDM symbol, $\mathbf{H}_{i,n}$ is the associated $K \times 1$ channel vector structured similar to the $\mathbf{H}$ in (2.2), and $\mathbf{V}_n$ is a noise vector distributed as $\mathcal{CN}(0, \sigma^2 I_K)$.

2.2 Basic OFDMA Symbol Structure in IEEE 802.16m [11]

The Advanced Air Interface (AAI) defined by IEEE 802.16m is designed for nonline-of-sight (NLOS) operation in the licensed frequency bands below 6 GHz. It supports time-division-duplexing (TDD) and frequency-division-duplexing (FDD) duplex modes, including half FDD (H-FDD) mobile station (MS) operation. Unless otherwise specified, the frame structure attributes and baseband processing are common for all duplex modes.

The AAI uses OFDMA as the multiple access scheme in the downlink and the uplink. The material of this section is mainly taken from [11].

2.2.1 Source Basic Terms

We introduce some basic terms that appear in the OFDMA physical layer (PHY) of IEEE 802.16m. These definitions help us understand the concepts of subcarrier allocation and transmission in IEEE 802.16m OFDMA.

- Physical and logical resource units: A physical resource unit (PRU) is the basic physical unit for resource allocation. It comprises P_{sc} consecutive subcarriers by N_{sym} consecutive OFDMA symbols. P_{sc} is 18 subcarriers and N_{sym} is 6 OFDMA symbols for type-1 subframes, 7 for type-2 subframes, and 5 for type-3 subframes. Table 2.1 illustrates the PRU sizes for different subframe types. A logical resource unit (LRU) is the basic logical unit for distributed and localized resource allocations. A LRU is also $P_{sc} \cdot N_{sym}$ subcarriers for type-1, type-2, and type-3 subframes. The LRU includes the pilots that are used in a PRU. The effective number of subcarriers in an LRU depends on the number of allocated pilots.

- Distributed resource unit: A distributed resource unit (DRU) contains a group of sub-

Table 2.1: PRU Structure for Different Types of Subframes

Subframe Type	Number of Subcarriers	Number of Symbols
1	18	6
2	18	7
3	18	5

carriers which are spread across the distributed resource allocations within a frequency partition. The size of DRU equals the size of PRU, i.e., P_{sc} subcarriers by N_{sym} OFDMA symbols.

- Contiguous resource unit: A localized resource unit, also known as contiguous resource unit (CRU), contains a group of subcarriers which are contiguous across the localized resource allocations. The size of CRU equals the size of PRU, i.e., P_{sc} subcarriers by N_{sym} OFDMA symbols.

In the frequency, OFDMA symbol is made up of subcarriers, the number of which determines the DFT 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.

The purpose of the guard bands is to help enable proper bandlimiting.

2.2.2 Primitive and Derived 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. This value is given in Table 2.2 and 2.3 for each nominal bandwidth.
- G: This is the ratio of CP time to "useful" time, i.e., T_{cp}/T_s. The following values shall be supported: 1/16, 1/8, and 1/4.

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

- N_{FFT}: smallest power of two greater than N_{used}.
- Sampling frequency: $F_s = \lfloor n \cdot BW/8000 \rfloor \times 8000$.
- Subcarrier spacing: $\triangle f = F_s/N_{FFT}$.
- Useful symbol time: $T_b = 1/\triangle f$.
- CP time: $T_g = G \times T_b$.
- OFDMA symbol time: $T_s = T_b + T_g$.
- Sampling time: T_b/N_{FFT}.

2.2.3 Frame Structure

The basic frame structure of IEEE 802.16m's AAI is illustrated in Fig. 2.4. Each 20ms superframe is divided into four 5-ms radio frames. When using the same OFDMA parameters as in Tables. 2.2 and 2.3 with channel bandwidth of 5, 10, or 20 MHz, each 5-ms radio frame further consists of eight subframes for $G = 1/8$ and 1/16. With channel bandwidth of 8.75

Table 2.2: OFDMA Parameters (Table 794 in [11])

The nominal channel bandwidth, BW (MHz)			5	7	8.75	10	20
Sampling factor, n			28/25	8/7	8/7	28/25	28/25
Sampling frequency, F_s (MHz)			5.6	8	10	11.2	22.4
FFT size, N_{FFT}			512	1024	1024	1024	2048
Subcarrier spacing, Δf (kHz)			10.94	7.81	9.77	10.94	10.94
Useful symbol time, T_b (μs)			91.4	128	102.4	91.4	91.4
CP ratio, $G = 1/8$	OFDMA symbol time, T_s (μs)		102.857	144	115.2	102.857	102.857
	FDD	Number of OFDMA symbols per 5ms frame	48	34	43	48	48
		Idle time (μs)	62.857	104	46.40	62.857	62.857
	TDD	Number of OFDMA symbols per 5ms frame	47	33	42	47	47
		TTG + RTG (μs)	165.714	248	161.6	165.714	165.714
CP ratio, $G = 1/16$	OFDMA symbol time, T_s (μs)		97.143	136	108.8	97.143	97.143
	FDD	Number of OFDMA symbols per 5ms frame	51	36	45	51	51
		Idle time (μs)	45.71	104	104	45.71	45.71
	TDD	Number of OFDMA symbols per 5ms frame	50	35	44	50	50
		TTG + RTG (μs)	142.853	240	212.8	142.853	142.853
CP ratio, $G = 1/4$	OFDMA symbol time, T_s (μs)		114.286	160	128	114.286	114.286
	FDD	Number of OFDMA symbols per 5ms frame	43	31	39	43	43
		Idle time (μs)	85.694	40	8	85.694	85.694
	TDD	Number of OFDMA symbols per 5ms frame	42	30	37	42	42
		TTG + RTG (μs)	199.98	200	264	199.98	199.98

Table 2.3: Additional OFDMA Parameters (Table 795 in [11])

The nominal channel bandwidth, BW (MHz)		5	7	8.75	10	20
Number of guard sub-carriers	Left	40	80	80	80	160
	Right	39	79	79	79	159
Number of used sub-carriers		433	865	865	865	1729
Number of physical resource unit (18x6) in a type-1 AAI subframe.		24	48	48	48	96

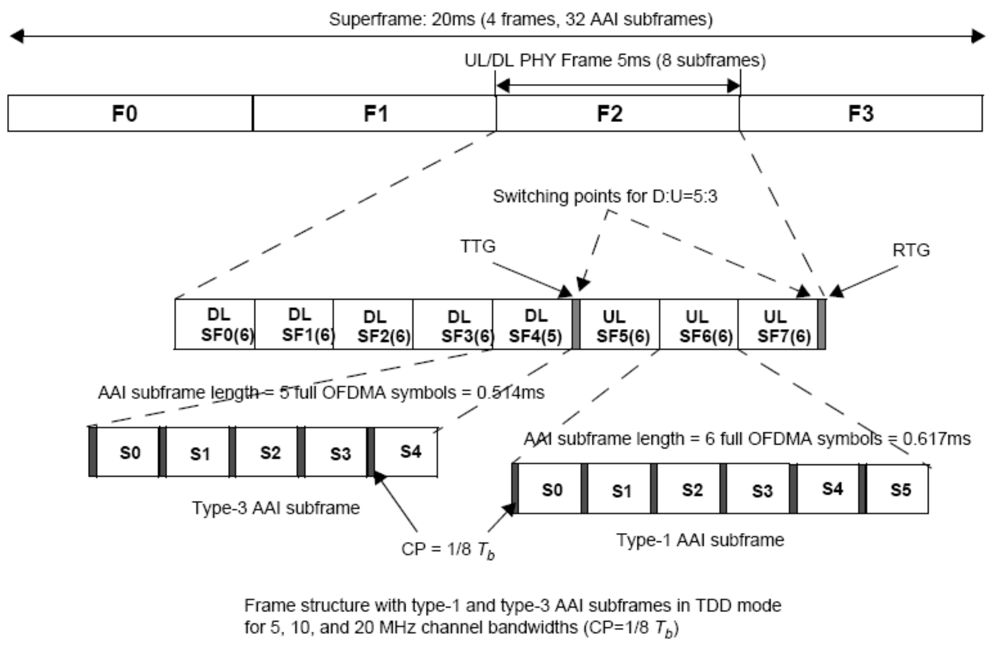

Superframe: 20ms (4 frames, 32 AAI subframes)
UL/DL PHY Frame 5ms (8 subframes)
F0
F1
F2
F3
Switching points for D:U=5:3
TTG
RTG
DL SF0(6)
DL SF1(6)
DL SF2(6)
DL SF3(6)
DL SF4(5)
UL SF5(6)
UL SF6(6)
UL SF7(6)
AAI subframe length = 5 full OFDMA symbols = 0.514ms
S0
S1
S2
S3
S4
Type-3 AAI subframe
CP = 1/8 T_b
AAI subframe length = 6 full OFDMA symbols = 0.617ms
S0
S1
S2
S3
S4
S5
Type-1 AAI subframe
Frame structure with type-1 and type-3 AAI subframes in TDD mode
for 5, 10, and 20 MHz channel bandwidths (CP=1/8 T_b)

Figure 2.4: Frame structure for 5, 10 and 20 MHz modes (Fig. 484 in [11]).

or 7 MHz, each 5-ms radio frame further consists of seven and six subframes, respectively, for $G = 1/8$ and $1/16$. In the case of $G = 1/4$, the number of subframes per frame is one less than that of other CP lengths for each bandwidth case. A subframe shall be assigned for either DL or UL transmission. There are four AAI subframe types:

- Type-1 subframe consists of six OFDMA symbols.
- Type-2 subframe consists of seven OFDMA symbols.
- Type-3 subframe consists of five OFDMA symbols.
- Type-4 subframe consists of nine OFDMA symbols. This type shall be applied only to UL subframe for the 8.75 MHz channel bandwidth when supporting the WirelessMAN-OFDMA (i.e., IEEE 802.16) frames.

The basic frame structure is applied to FDD and TDD duplexing schemes, including H-FDD MS operation. The number of switching points in each radio frame in TDD systems shall be two, where a switching point is defined as a change of directionality, i.e., from DL to UL or from UL to DL.

A data burst shall occupy either one AAI subframe (i.e., the default TTI transmission) or contiguous multiple AAI subframes (i.e., the long TTI transmission). Any 2 long TTI bursts allocated to an AMS shall not be partially overlapped, i.e. any two long TTI bursts in FDD shall either be over the same 4 subframes or without any overlap. The long TTI in FDD shall be 4 AAI subframes for both DL and UL. For DL (UL), the long TTI in TDD shall be all DL (UL) AAI subframes in a frame. A subframe can have default TTI allocations and long TTI allocations.

2.3 Downlink Transmission in IEEE 802.16m OFDMA [11]

Again this section is mainly taken from [11]. Each DL subframe is divided into 4 or fewer frequency partitions with each partition consisting of a set of physical resource units across the total number of OFDMA symbols available in the subframe. Each frequency partition can include contiguous (localized) and/or non-contiguous (distributed) PRUs. Each frequency partition can be used for different purposes such as fractional frequency reuse (FFR) or multicast and broadcast services (MBS). Fig. 2.5 illustrates the downlink physical structure in an example of two frequency partitions with frequency partition 2 including both contiguous and distributed resource allocations.

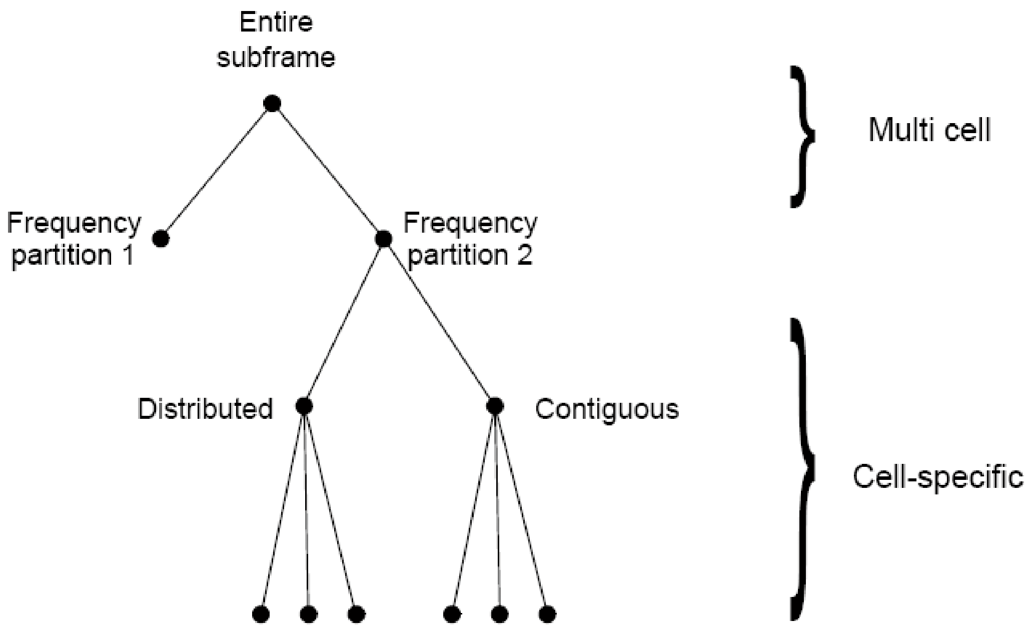


Figure 2.5: Example of downlink physical structure (Fig. 499 in [11]).

2.3.1 Subband Partitioning

The PRUs are first subdivided into subbands and minibands where a subband comprises N_1 adjacent PRUs and a miniband comprises N_2 adjacent PRUs, with $N_1 = 4$ and $N_2 = 1$. Subbands are suitable for frequency selective allocation as they provide a contiguous allocation of PRUs in frequency. Minibands are suitable for frequency diverse allocation and are permuted in frequency.

The number of subbands reserved is denoted by K_{SB}. The number of PRUs allocated to subbands is denoted by L_{SB}, where $L_{SB} = N_1 \cdot K_{SB}$, depending on system bandwidth. A 5, 4 or 3-bit field called Downlink Subband Allocation Count (DSAC) determines the value of K_{SB}. The remainder of the PRUs are allocated to minibands. The number of minibands in an allocation is denoted by K_{MB}. The number of PRUs allocated to minibands is denoted by L_{MB}, where $L_{MB} = N_2 \cdot K_{MB}$. The total number of PRUs is denoted N_{PRU} where

Table 2.4: Mapping Between DSAC and K_{SB} for 1024 FFT Size (Table 803 in [11])

DSAC	**Number of subbands allocated (*K_{SB}*)**	***DSAC***	**Number of subbands allocated (*K_{SB}*)**
0	0	8	8
1	1	9	9
2	2	10	10
3	3	11	NA.
4	4	12	NA.
5	5	13	NA.
6	6	14	NA.
7	7	15	NA.

$N_{PRU} = L_{SB} + L_{MB}$. The maximum number of subbands that can be formed is denoted N_{sub} where $N_{sub} = \lfloor N_{PRU}/N_1 \rfloor$. Table 2.4 show the mapping between DSAC and K_{SB} for the 1024FFT size. PRUs are partitioned and reordered into two groups called subband PRUs and miniband PRUs and denoted PRU_{SB} and PRU_{MB}, respectively. The set PRU_{SB} is numbered from 0 to $L_{SB} - 1$,and the set PRU_{MB} is numbered from 0 to $L_{MB} - 1$. The mapping of PRUs to PRU_{SB} is defined as

$$PRU_{SB}[j] = PRU[i], \quad j = 0, 1, ..., L_{SB} - 1, \tag{2.4}$$

where

$$\begin{aligned} i = N_1 \cdot \{ \lceil \frac{N_{sub}}{N_{sub} - K_{SB}} \rceil \cdot \lfloor \frac{j + L_{MB}}{N_1} \rfloor + \lfloor \lfloor \frac{j + L_{MB}}{N_1} \rfloor \cdot \frac{GCD(N_{sub}, \lceil \frac{N_{sub}}{N_{sub} - K_{SB}} \rceil)}{N_{sub}} \rfloor \} \mod \{N_{sub}\} \\ + \{j + L_{MB}\} \mod \{N_1\} \end{aligned} \tag{2.5}$$

with $GCD(x, y)$ being the greatest common divisor of x and y, and the mapping of PRUs to PRU_{MB} as

$$PRU_{MB}[k] = PRU[i], \quad k = 0, 1, ..., L_{MB} - 1, \tag{2.6}$$

where

$$i = \begin{cases} N_1 \cdot \{\lceil \frac{N_{sub}}{N_{sub}-K_{SB}} \cdot \rceil \cdot \lfloor \frac{k}{N_1} \rfloor + \lfloor \lfloor \frac{k}{N_1} \rfloor \cdot \frac{GCD(N_{sub}, \lceil \frac{N_{sub}}{N_{sub}-K_{SB}} \rceil)}{N_{sub}} \rfloor\} \mod \{N_{sub}\} & \\ +\{k\}) \mod \{N_1\}, & K_{SB} > 0 \\ , i = k, & K_{SB} = 0. \end{cases} \tag{2.7}$$

Fig. 2.6 illustrates the PRU to PRU_{SB} and PRU_{MB} mappings for a 10 MHz bandwidth with K_{SB} equal to 7.

2.3.2 Miniband Permutation

The miniband permutation maps the PRU_{MB}s to Permuted PRU_{MB}s ($PPRU_{MB}$s) to ensure that frequency diverse PRUs are allocated to each frequency partition. Fig. 2.7 shows an example. The mapping from PRU_{MB} to $PPRU_{MB}$ is given by:

$$PPRU_{MB}[j] = PRU_{MB}[i], \quad j = 0, 1, ..., L_{MB} - 1, \tag{2.8}$$

where

$$i = (q(j) \mod D) \cdot P + \lfloor \frac{q(j)}{D} \rfloor, \tag{2.9}$$

$$P = \min(K_{MB}, N_1/N_2), \tag{2.10}$$

$$r(j) = \max(j - ((K_{MB} \mod P) \cdot D), 0), \tag{2.11}$$

$$q(j) = j + \lfloor \frac{r(j)}{D-1} \rfloor, \tag{2.12}$$

$$D = \lfloor \frac{K_{MB}}{P} + 1 \rfloor. \tag{2.13}$$

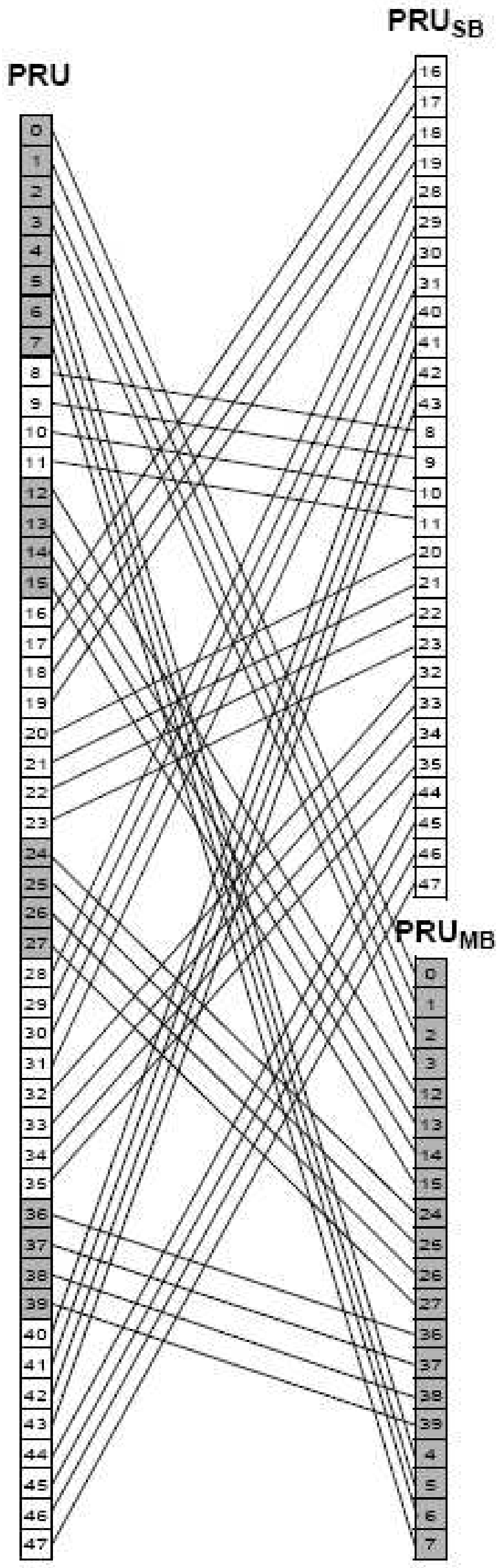


Figure 2.6: PRU to PRU_{SB} and PRU_{MB} mapping for BW = 10 MHz, and $K_{SB} = 7$ (Fig. 500 in [11]).

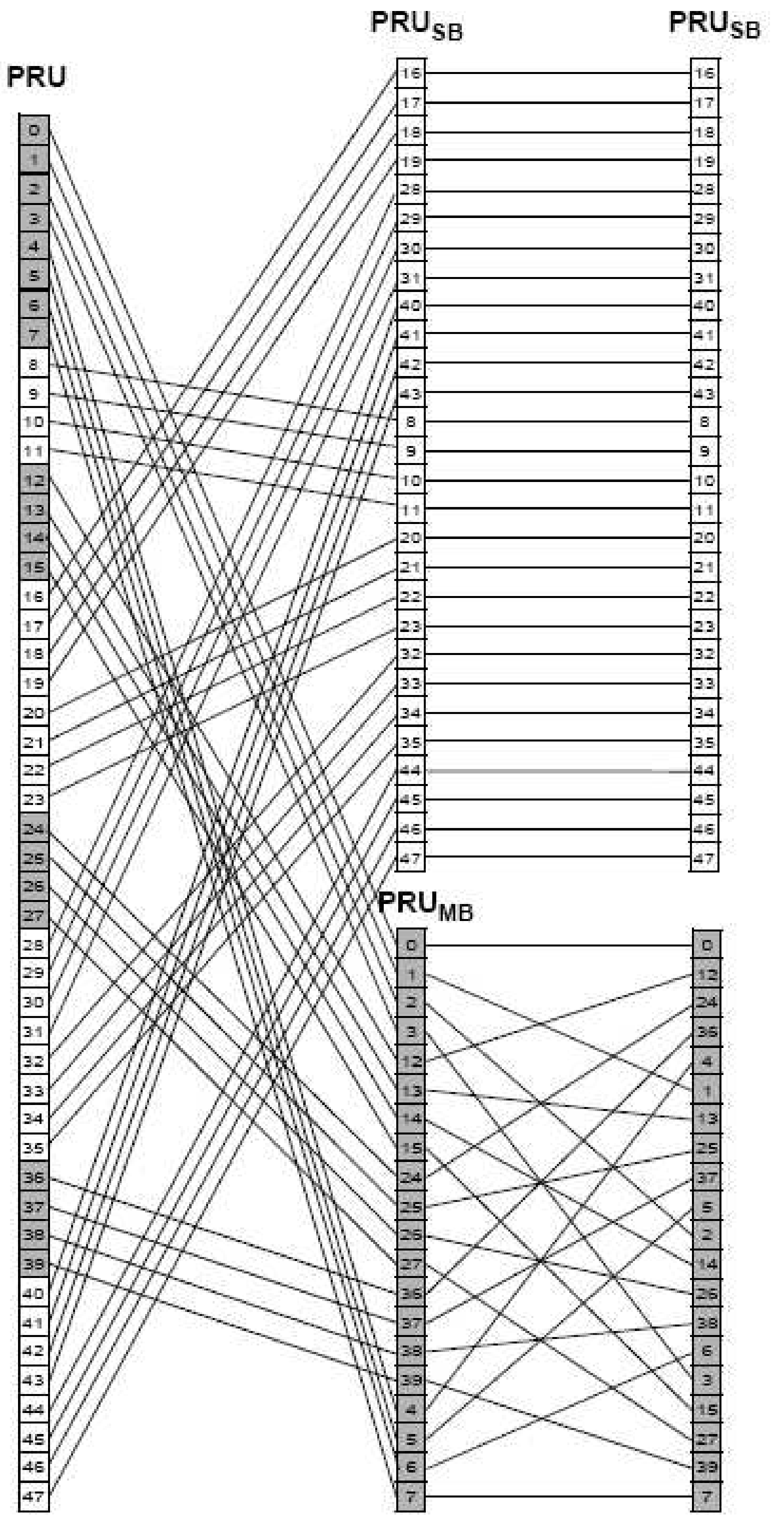


Figure 2.7: Mapping from PRUs to PRU_{SB} and PRU_{MB} mapping for BW = 10 MHz, and $K_{SB} = 7$ (Fig. 501 in [11]).

2.3.3 Frequency Partitioning

The PRU_{SB} and $PPRU_{MB}$ are allocated to one or more frequency partitions. The frequency partition configuration is transmitted in the SFH in a 4 or 3-bit called the Downlink Frequency Partition Configuration (DFPC) depending on system bandwidth. Frequency Partition Count (FPCT) defines the number of frequency partitions. Frequency Partition Size (FPSi) defines the number of PRUs allocated to FP_i. FPCT and FPSi are determined from DFPC as shown in Table 2.5. A 3, 2, or 1-bit parameter called the Downlink Frequency Partition Subband Count (DFPSC) defines the number of subbands allocated to FP_i, $i > 0$. Fig. 2.8 continues the examples in Figs. 2.6 and 2.7 and shows how PRU_{SB} and $PPRU_{MB}$ can be mapped to frequency partitions. The number of subbands in ith frequency partition is denoted by K_{SB,FP_i}. The number of minibands is denoted by K_{MB,FP_i}, which is determined by FPS and FPSC fields. The number of subband PRUs in each frequency

Table 2.5: Mapping Between DFPC and Frequency Partitioning for 1024 FFT Size (Table 806 in [11])

DFPC	**Freq. Partitioning (FP_0:FP_1:FP_2:FP_3)**	***FPCT***	***FPS_0***	***FPS_i (i>0)***
0	1 : 0 : 0 : 0	1	N_{PRU}	0
1	0 : 1 : 1 : 1	3	0	$FPS_1 = N_{PRU} - 2$*floor(N_{PRU}/3) FPS_2 = floor(N_{PRU}/3) FPS_3 = floor(N_{PRU}/3)
2	1 : 1 : 1 : 1	4	N_{PRU}-3*floor(N_{PRU}/4)	floor(N_{PRU}/4)
3	3 : 1 : 1 : 1	4	N_{PRU}-3*floor(N_{PRU}/6)	floor(N_{PRU}/6)
4	5 : 1 : 1 : 1	4	N_{PRU}-3*floor(N_{PRU}/8)	floor(N_{PRU}/8)
5	9 : 5 : 5 : 5	4	N_{PRU}-3*floor(N_{PRU}*5/24)	floor(N_{PRU}*5/24)
6	0 : 1 : 1 : 0	2	0	N_{PRU}/2 for i=1, 2 0 for i =3
7	1 : 1 : 1 : 0	3	N_{PRU}-2*floor(N_{PRU}/3)	floor(N_{PRU}/3) for i=1, 2 0 for i =3

partition is denoted by $L_{SB,FPi}$, which is given by $L_{SB,FP_i} = N_1 \cdot K_{SB,FPi}$. The number of miniband PRUs in each frequency partition is denoted by $L_{MB,FPi}$, which is given by $L_{MB,FP_i} = N_2 \cdot K_{MB,FPi}$. We have

$$K_{SB,FPi} = \begin{cases} K_{SB} - (FPCT-1)\cdot DFPSC, & i=0, FPCT=4, \\ DFPSC, & i>0, FPCT=4, \\ DFPSC, & i=0, FPCT=3, DFPC=1, \\ K_{SB} - (FPCT-1)\cdot DFPSC, & i=0, FPCT=3, DFPC\neq 1, \\ DFPSC, & i=1,2, FPCT=3, DFPC\neq 1, \\ DFPSC, & i=1,2, FPCT=2, \\ K_{SB}, & i=0, FPCT=1, \end{cases} \tag{2.14}$$

$$K_{MB,FPi} = (FPS_i - K_{SB,FPi}\cdot N_1)/N_2, \quad 0 \le i < FPCT. \tag{2.15}$$

The mapping of subband PRUs and miniband PRUs to the frequency partition is given by

$$PRU_{FPi}(j) = \begin{cases} PRU_{SB}(k_1), & 0 \le j < L_{SB,FPi}, \\ PPRU_{MB}(k_2), & L_{SB,FPi} \le j < (L_{SB,FPi} + L_{MB,FPi}), \end{cases} \tag{2.16}$$

where

$$k_1 = \sum_{m=0}^{i-1} L_{SB,FPm} + j \tag{2.17}$$

and

$$k_2 = \sum_{m=0}^{i-1} L_{SB,FPm} + j - L_{SB,FPi}. \tag{2.18}$$

2.3.4 Cell-Specific Resource Mapping

The content of this section is mainly taken from [11]. PRU_{FPi}s are mapped to LRUs. All further PRU and subcarrier permutation are constrained to the PRUs of a frequency partition.

2.3.4.1 CRU/DRU Allocation

The partition between CRUs and DRUs is done on a sector specific basis.Let $L_{SB-CRU,FPi}$ and $L_{MB-CRU,FPi}$ denote the number of allocated subband CRUs and miniband CRUs for

FP_i $(i \geq 0)$. The number of total allocated subband and miniband CRUs, in units of a subband (i.e. N_1 PRUs), for FP_i $(i \geq 0)$ is given by the downlink CRU allocation size, $DCAS_i$. The numbers of subband-based and miniband-based CRUs in FP_0 are given by $DCAS_{SB,0}$ and $DCAS_{MB,0}$, in units of a subband and miniband, respectively. When DFPC = 0, $DCAS_i$ must be equal to 0.

ForFP_0, the values of $DCAS_{SB,0}$ is explicitly signaled in the SFH as a 5, 4 or 3-bit field to indicate the number of subbands in unsigned-binary format, with $DCAS_{SB,0} \leq K_{SB,FP}$. A 5, 4 or 3-bit Downlink miniband-based CRU allocation size($DCAS_{MB,0}$) is sent in the SFH only for partition FP_0, depending on FFT size. The number of subband-based CRUs for FP_0 is given by

$$L_{SB-CRU,FP_0} = N_1 \cdot DCAS_{SB,0}. \tag{2.19}$$

For FP_i $(i > 0, FPCT \neq 2)$, the number of subband CRUs (L_{SB-CRU,FP_i}) and miniband CRUs (L_{MB-CRU,FP_i}) are derived as follows:

$$L_{SB-CRU,FP_i} = N_1 \cdot min\{DCAS_i, K_{SB,FP_i}\}, \tag{2.20}$$

$$L_{MB-CRU,FP_i} = \begin{cases} 0, & DCAS_i \leq K_{SB,FP_i}, \\ (DCAS_i - K_{SB,FP_i}) \cdot N_1, & DCAS_i > K_{SB,FP_i}. \end{cases} \tag{2.21}$$

When FPCT = 2, $DCAS_{SB,i}$ and $DCAS_{MB,i}$ for $i = 1$ and 2 are signaled using the $DCAS_{SB,0}$ and $DCAS_{MB,0}$ fields in the SFH. Since FP_0 and FP_3 are empty, $L_{SB-CRU,FP_0} = L_{MB-CRU,FP_0} = L_{DRU,FP_0}$ and $L_{SB-CRU,FP_3} = L_{MB-CRU,FP_3} = L_{DRU,FP_3}$. For $i = 1$ and 2, $L_{SB-CRU,FP_i} = N_1 \cdot DCAS_{SB,0}$ and L_{MB-CRU,FP_i} is obtained from $DCAS_{MB,0}$ using the mapping in Table 2.6. The total number of CRUs in frequency partition FP_i, for $0 \leq 1 < FPCT$, is denoted by L_{CRU,FP_i}, where

$$L_{CRU,FP_i} = L_{SB-CRU,FP_i} + L_{MB-CRU,FP_i}. \tag{2.22}$$

The number of DRUs in each frequency partition is denoted byL_{DRU,FP_i}, where

$$L_{DRU,FP_i} = FPS_i - L_{CRU,FPi} \tag{2.23}$$

Table 2.6: Mapping Between $DCAS_{MB,0}$ and Number of Miniband-Based CRUs for FP_0 for 1024 FFT Size (Table 809 in [11])

$DCAS_{MB,0}$	Number of miniband-based CRU for FP_0 (i.e. $L_{MB\text{-}CRU,\ FP0}$)	$DCAS_{MB,0}$	Number of miniband-based CRU for FP_0 (i.e. $L_{MB\text{-}CRU,\ FP0}$)
0	0	8	16
1	2	9	18
2	4	10	20
3	6	11	22
4	8	12	24
5	10	13	38
6	12	14	40
7	14	15	42

and FPS_i is the number of PRUs allocated to FP_i.

The mapping from PRU_{FP_i} *to* CRU_{FP_i} is given by:

$$CRU_{FP_i}[j] = \begin{cases} PRU_{FP_i}, & 0 \le j < L_{SB-CRU,FP_i}, \\ PRU_{FP_i}[k + L_{SB-CRU,FP_i}], & L_{SB-CRU,FP_i} \le j < L_{CRU,FP_i}, \end{cases} \tag{2.24}$$

where k $= s[j - L_{SB-CRU,FP_i}]$ with s[ ] being the CRU/DRU allocation sequence defined as

$$s[j] = \{PermSeq(j) + DL\_PermBase\} \mod (FPS_i - L_{SB-CRU,FP_i}). \tag{2.25}$$

We have $0 \le s[j] < FPS_i - L_{SB-CRU,FP_i}$ in(2.25), $PermSeq()$ is the permutation sequence of length $(FPS_i - L_{SB-CRU,FP_i})$ and is determined by $SEED = \{IDcell \cdot 343\} \bmod 1024$ and $DL\_PermBase$ is set to preamble $IDcell$. The mapping of PRU_{FP_i} *to* DRU_{FP_i} is given by

$$DRU_{FP_i}[j] = PRU_{FP_i}[k + L_{SB-CRU,FP_i}], \quad 0 \le j < L_{DRU,FP_i} \tag{2.26}$$

where $k = s[j + L_{CRU,FP_i} - L_{SB-CRU,FP_i}]$. Fig. 2.9 presents an example to illustrate the various steps of subband partitioning, miniband permutation, frequency partitioning, and

CRU/DRU allocation for the case of 10 MHz system bandwidth. For this example, $K_{SB} = DSAC = 7$, $FPCT = 4$, $FPS_i = 12(for i \geq 0)$, $DFPSC = 2$, $DCAS_{SB,0} = 1$, $DCAS_{MB,0} = 1$, $DCAS_i = 2$ and $IDcell = 0$.

2.3.4.2 Subcarrier Permutation

The downlink DRUs are used to form two-stream DLRUs by subcarrier permutation. The subcarrier permutation defined for the DL distributed resource allocations within a frequency partition spreads the subcarriers of the DRU across the whole distributed resource allocations. The granularity of the subcarrier permutation is equal to a pair of subcarriers.

After mapping all pilots, the remainders of the used subcarriers are used to define the distributed LRUs. To allocate the LRUs, the remaining subcarriers are paired into contiguous tone-pairs. Each LRU consists of a group of tone-pairs.

Let $L_{SC,l}$ denote the number of data subcarriers in lth OFDMA symbol within a PRU, i.e., $L_{SC,l} = P_{SC} - N_l$, where N_l denotes the number of pilot subcarriers in the lth OFDMA symbol within a PRU. Let $L_{SP,l}$ denote the number of data subcarrier-pairs in the lth OFDMA symbol within a PRU and is equal to $L_{SC,l}/2$. The DL subcarrier permutation is performed as follows. For each lth OFDMA symbol in the subframe:

1. Allocate the N_l pilots within each DRU as described in Section 2.3.4.3. Denote the data subcarriers of $DRU_{FPi}[j]$ in the lth OFDMA symbol as

$$SC_{DRU,j,l}^{FPi}[k], \quad 0 \leq j < L_{DRU,FPi}, \qquad 0 \leq k < L_{SC,l}. \tag{2.27}$$

2. Renumber the $L_{DRU,FPi} \cdot L_{SC,l}$ data subcarriers of the DRUs in order, from 0 to $L_{DRU,FPi} \cdot L_{SC,l} - 1$. Group these contiguous and logically renumbered subcarriers into $L_{DRU,FPi} \cdot L_{SP,l}$ pairs and renumber them from 0 to $L_{DRU,FPi} \cdot L_{SP,l} - 1$. The

renumbered subcarrier pairs in the lth OFDMA symbol are denoted as

$$RSP_{FPi,l}[u] = \{SC_{DRUj,l}^{FPi}[2v], SC_{DRUj,l}^{FPi}[2v+1]\}, \quad 0 \le u < L_{DRU,FPi}L_{SP,l}, \tag{2.28}$$

where $\quad j = \lfloor u/L_{SP,l} \rfloor \quad$ and $v = \{u\} \mod (L_{SP,l})$.

3. Apply the subcarrier permutation formula to map $RSP_{FPi,l}$ into the sth distributed LRU, s=0,1,...,$L_{DRU,FPi}$-1, where the subcarrier permutation formula is given by

$$SP_{LRUs,l}^{FPi}[m] = RSP_{FPi,l}[k], \quad 0 \le m < L_{SP,l}, \tag{2.29}$$

with

$$k = L_{DRU,FPi} \cdot f(m,s,l) + g(PermSeq(), s, m, l, t). \tag{2.30}$$

In the above,

1. $SC_{LRUs,l}^{FPi}[m]$ is the mth subcarrier pair in the lth OFDMA symbol in the sth distributed LRU of the tth subframe;
2. m is the subcarrier pair index, 0 to $L_{SP,l} - 1$;
3. l is the OFDMA symbol index, 0 to $N_{sym} - 1$;
4. s is the distributed LRU index, 0 to $L_{DRU,FPi} - 1$;
5. t is the subframe index with respect to the frame;
6. $PermSeq()$ is the permutation sequence of length $L_{DRU,FPi}$ and is determined by $SEED = \{IDcell \times 343\} \mod 2^{10}$;
7. $g(PermSeq(), s, m, l, t)$ is a function with value from the set [0,$L_{DRU,FPi}$-1], which is defined according to

$$g(PermSeq(), s, m, l, t) = \{PermSeq[\{f(m,s,l) + s + l\} \mod \{L_{DRU,FPi}\}] + DL\_PermBase\} \mod L_{DRU,FPi}. \tag{2.31}$$

where $DL\_PermBase$ is set to preamble IDcell, and $f(m,s,l) = (m + 13 * (s + l))$mod $L_{SP,l}$.

2.3.4.3 Random Sequence Generation

The permutation sequence generation algorithm with 10-bit $SEED$ $(S_{n-10}, S_{n-9}, ..., S_{n-1})$ shall generate a permutation sequence of size M according to the following process:

- Initialization:
  1. Initialize the variables of the first order polynomial equation with the 10-bit seed, SEED. Set $d_1 = \lfloor SEED/2^5 \rfloor + 619$ and $d_2 = SEED \mod 2^5$.
  2. Initialize an array A with size M to contents $0, 1, \ldots, M-1$ (i.e., $A[i] = i$), for $0 \le i < M$.
  3. Initialize the counter i to $M - 1$.
  4. Initialize x to -1.
- Repeat the following steps if $i > 0$:
  1. Initialize the counter j to 0.
  2. Loop as follows:
     (a) Increment x by i.
     (b) Calculate the output variable of $y = \{(d_1 \cdot x + d_2) \mod 1031\} \mod M$. If $y \ge i$, set y=y mod(i+1)
     (c) Swap $A[i]$ and $A[y]$
     (d) Decrement i by 1.
- $PermSeq(i) = A[i]$, where $0 \le i < M$.

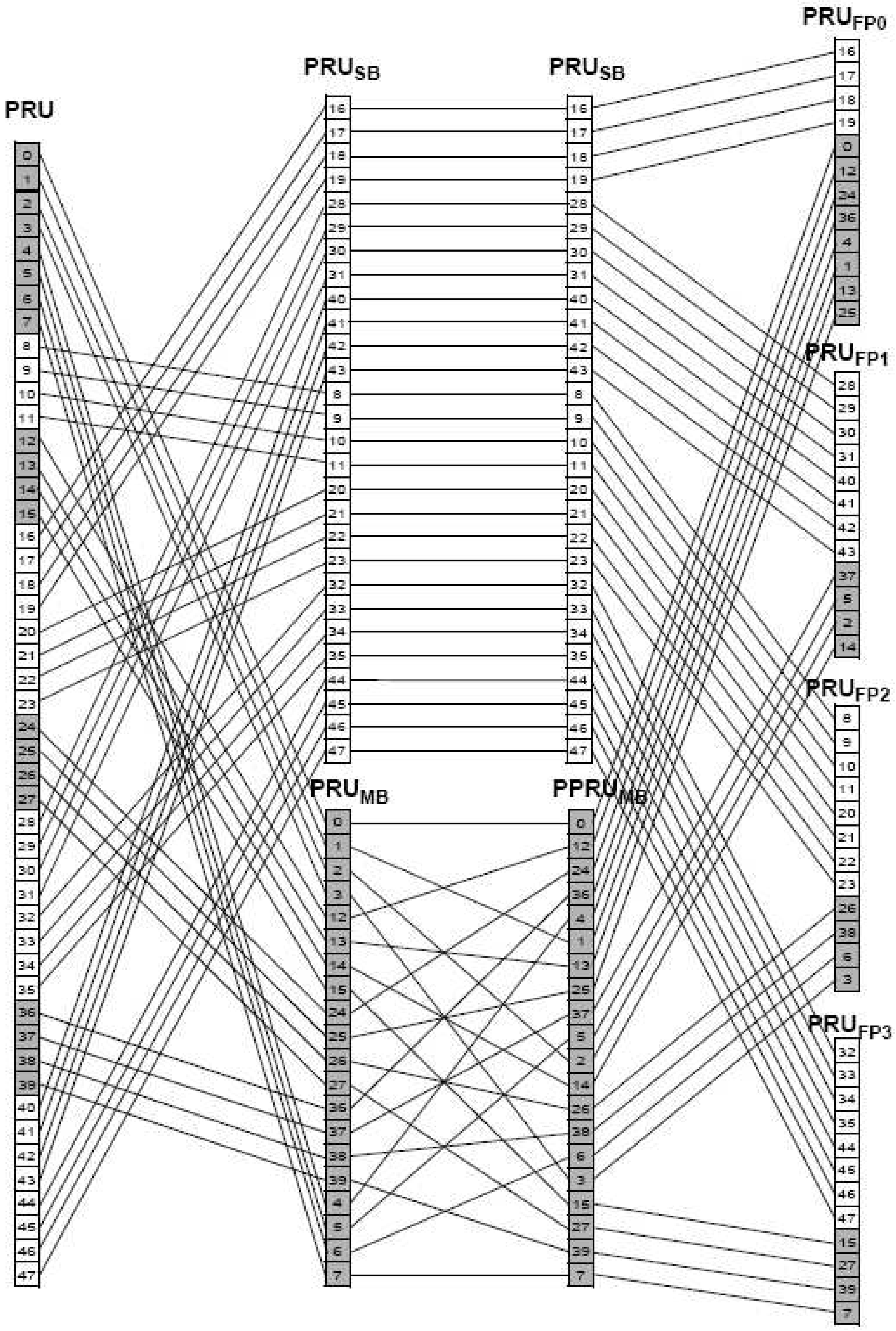


Figure 2.8: Frequency partitioning for BW = 10 MHz, K_{SB} = 7, FPCT = 4, $FPS_0 = FPS_i$ = 12, and DFPSC = 1 (Fig. 502 in [11]).

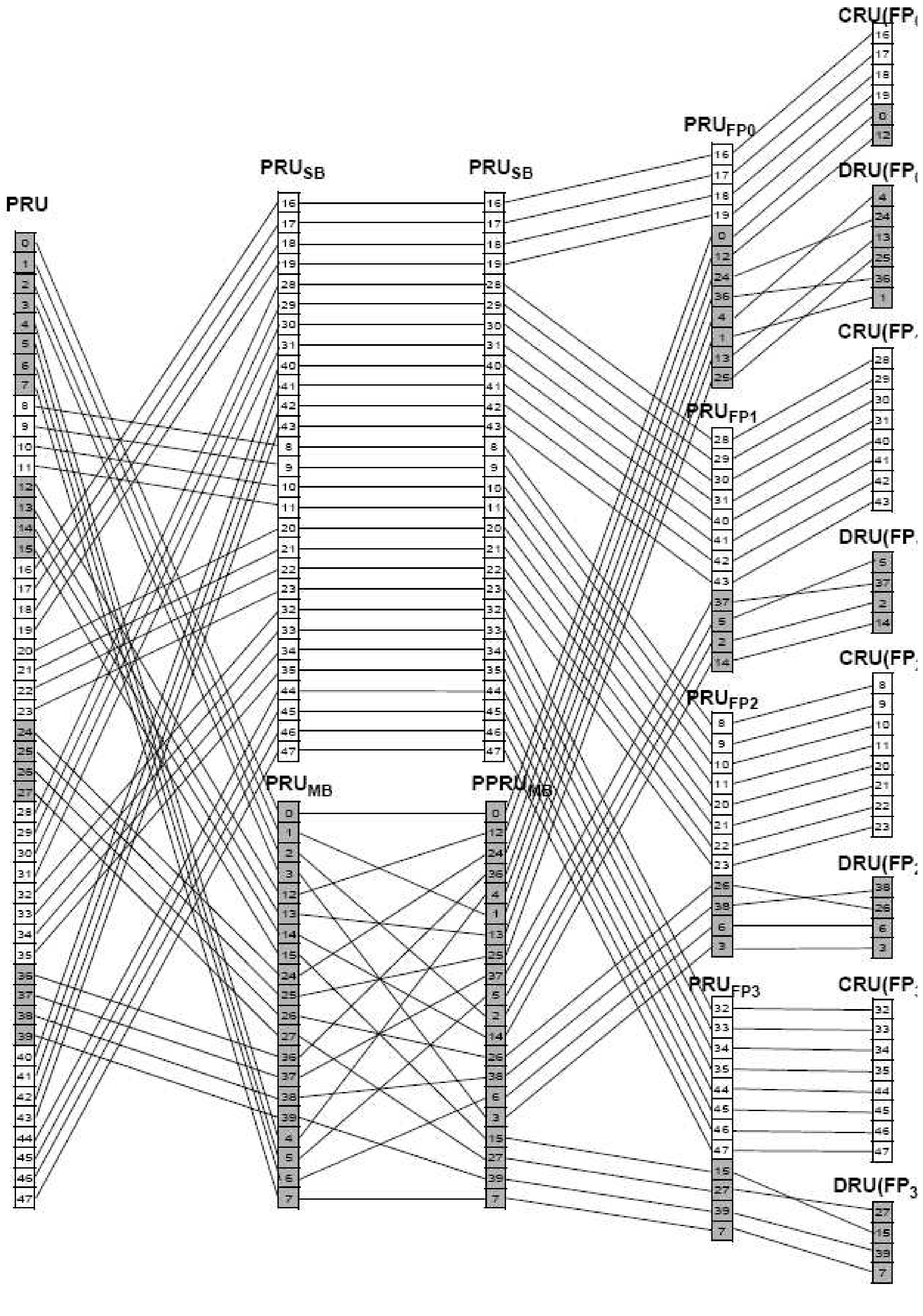


Figure 2.9: Frequency partition for BW=10MHz, $K_{SB} = 7$, $FPCT = 4$, $FPCT = 4$, $FPS_0 = FPS_i = 12$, $DFPSC = 2$, $DCAS_{SB,0} = 1$, $DCAS_{MB,0} = 1$, $DCAS_i = 2$ and $IDcell = 0$ (Fig. 503 in [11]).

2.3.4.4 Test Case for Subcarrier Permutation

In this subsection, we provide an example of the subcarrier permutation. It is also used in the later channel estimation simulation.

- NPRU = 48
- DSAC = 7
- $K_{SB} = 7$
- $K_{MB} = 20$
- $L_{SB} = 28$
- $N_{sub} = 12$
- DFPC = 2
- FP0 : FP1 : FP2 : FP3 = 1:1:1:1
- FPS = 12
- ID_Cell = 2
- SEED = 686
- DL_PermBase = 2

Fig. 2.10 shows the original positions of LRUs permutation before subcarrier permutation. And we give the number for each pair (every two subcarriers as a pair) as shown in Fig. 2.11. Figs. 2.12 to 2.15 show the subcarrier permutation for each DRU. In these figures, each number pair X_Y indicate the DRU number (X) and the subcarrier index (Y).

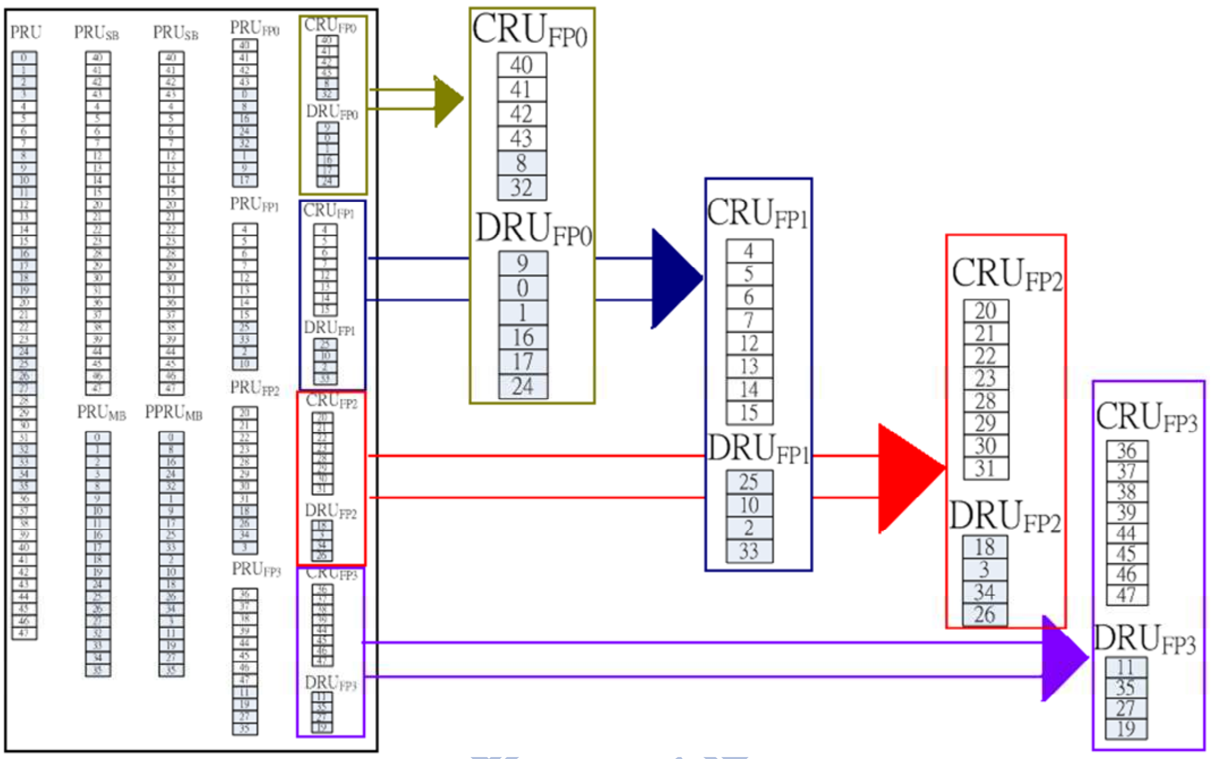


Figure 2.10: LRU positions before subcarrier permutation.

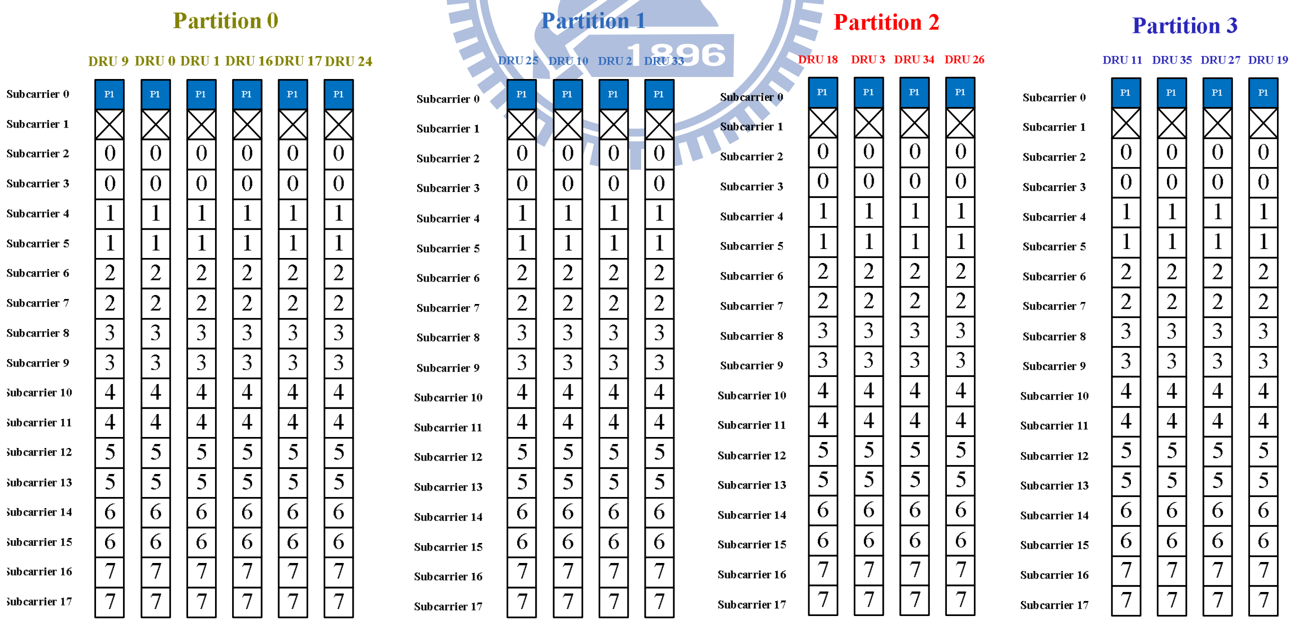


Figure 2.11: Subcarrier pairs before subcarrier permutation for each frequency partition.

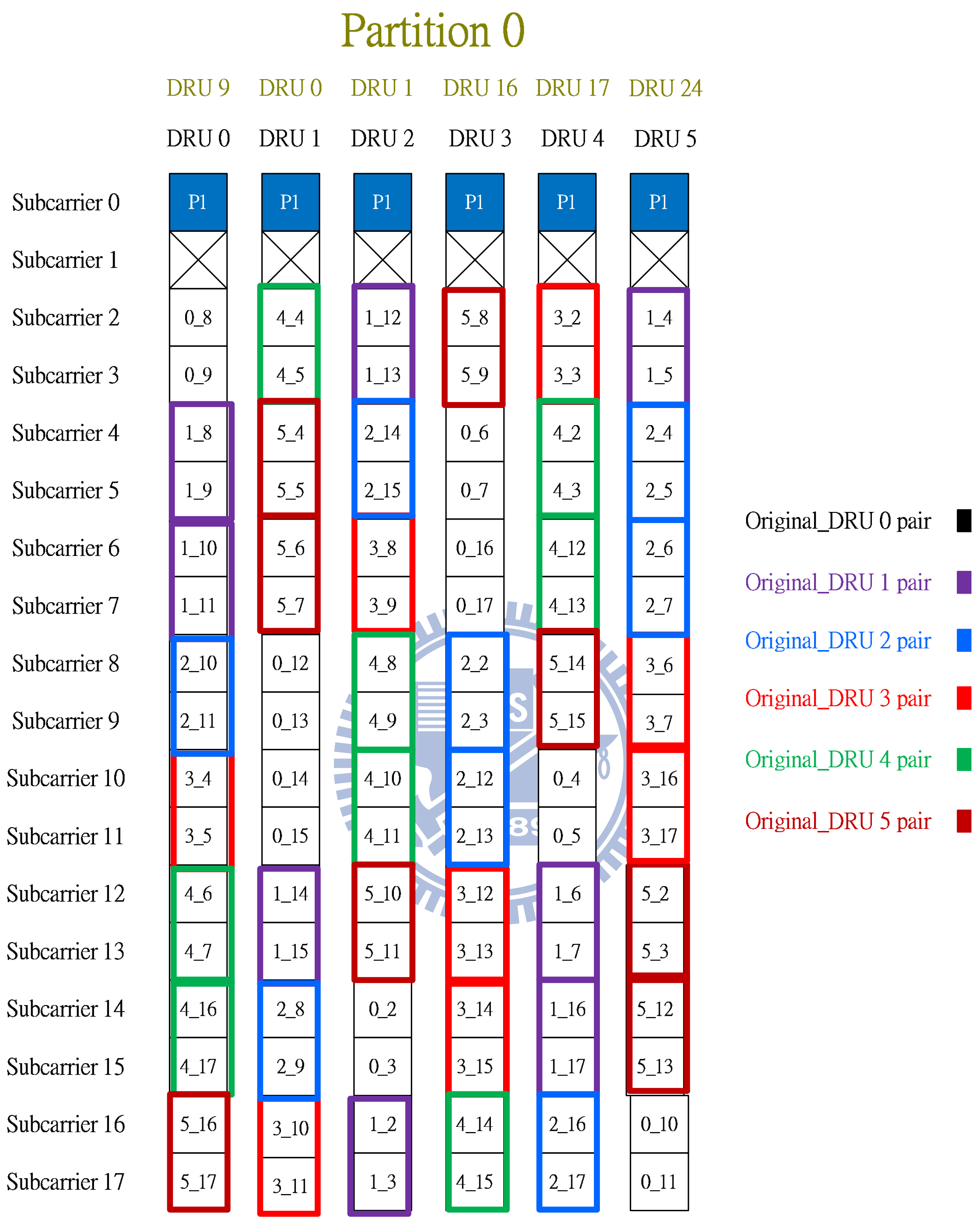


Figure 2.12: Subcarrier permutation for FP0.

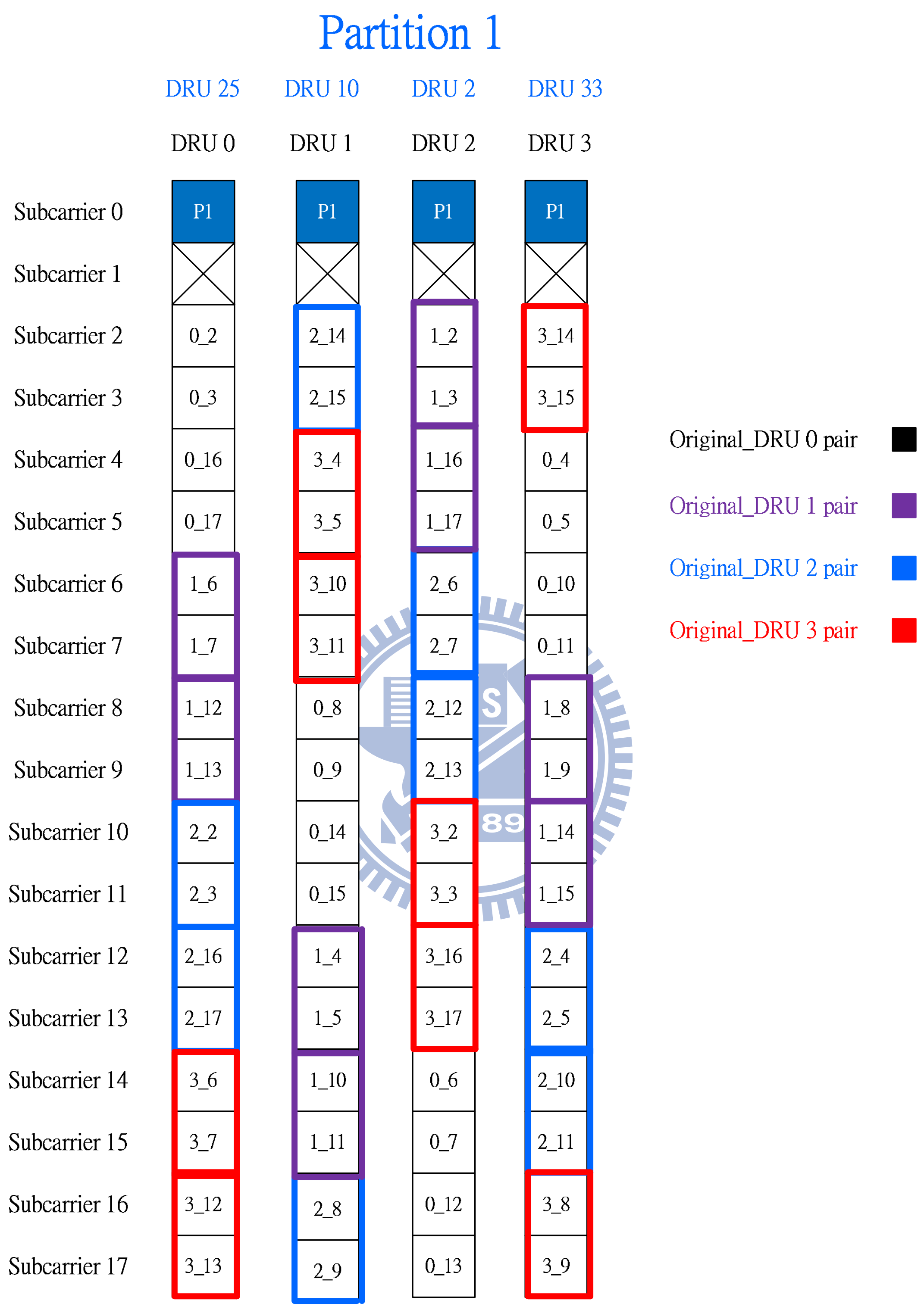


Figure 2.13: Subcarrier permutation for FP1.

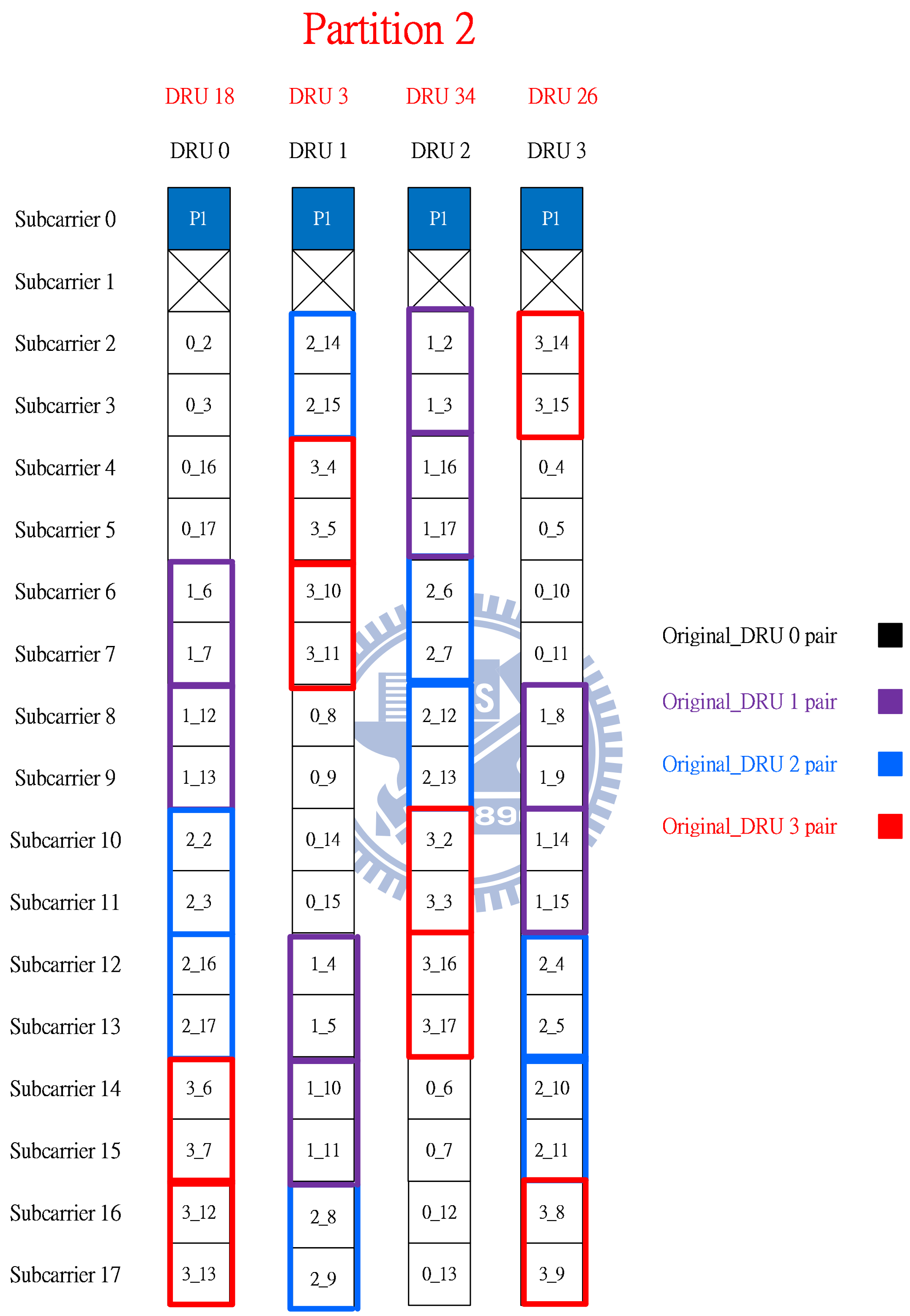


Figure 2.14: Subcarrier permutation for FP2.

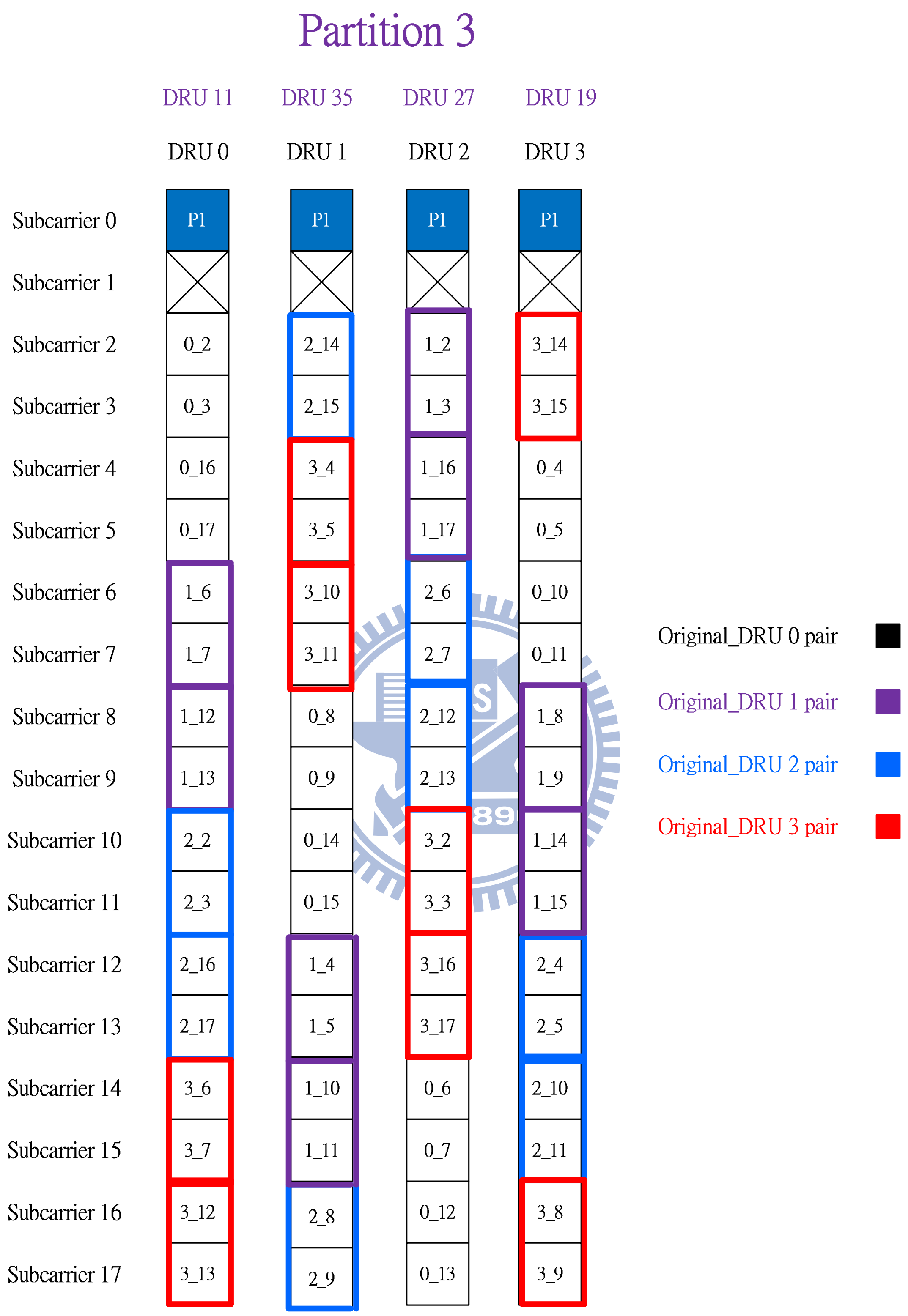


Figure 2.15: Subcarrier permutation for FP3.

2.3.5 Pilot Structure

2.3.5.1 Pilot patterns

Pilot patterns are specified within a PRU. Base pilot patterns used for DL data transmission with one data stream in dedicated and common pilot scenarios are shown in Figure 2.16, with the subcarrier index increasing from top to bottom and the OFDM symbol index increasing from left to right. Figs. 2.16(a) and 2.16(b) show the pilot locations for stream sets 0 and 1, respectively.

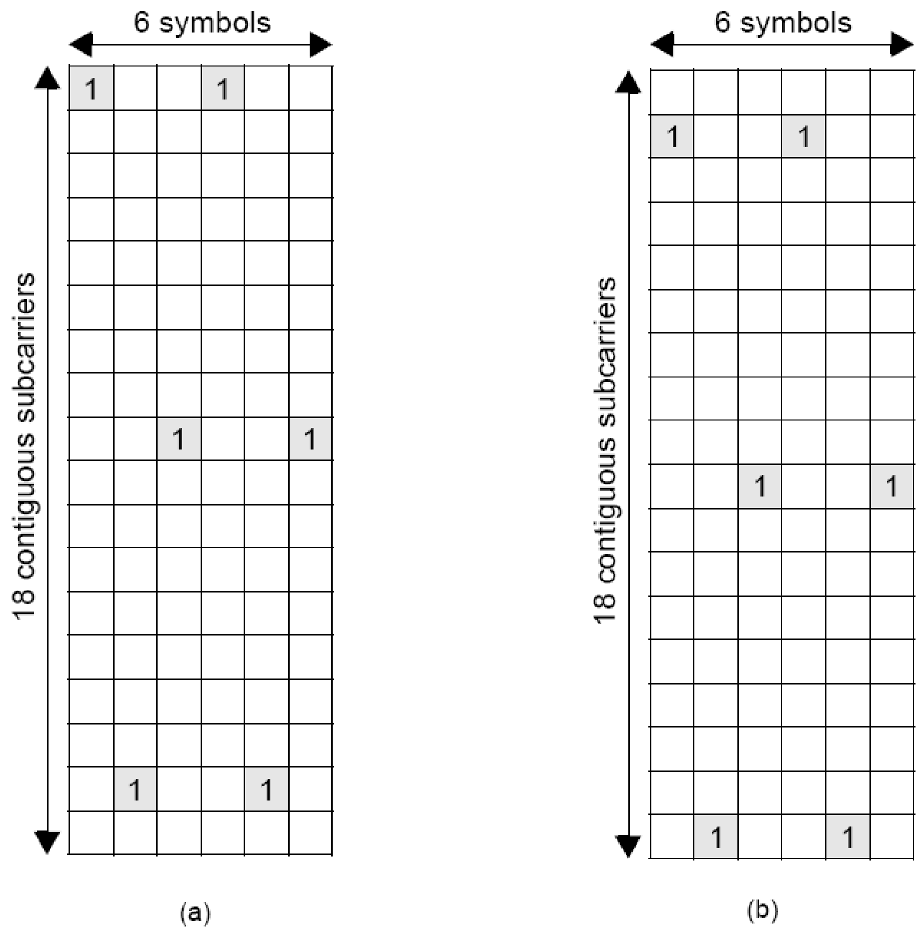


Figure 2.16: Pilot patterns used for 1 DL data streams (Fig. 505 in [11]).

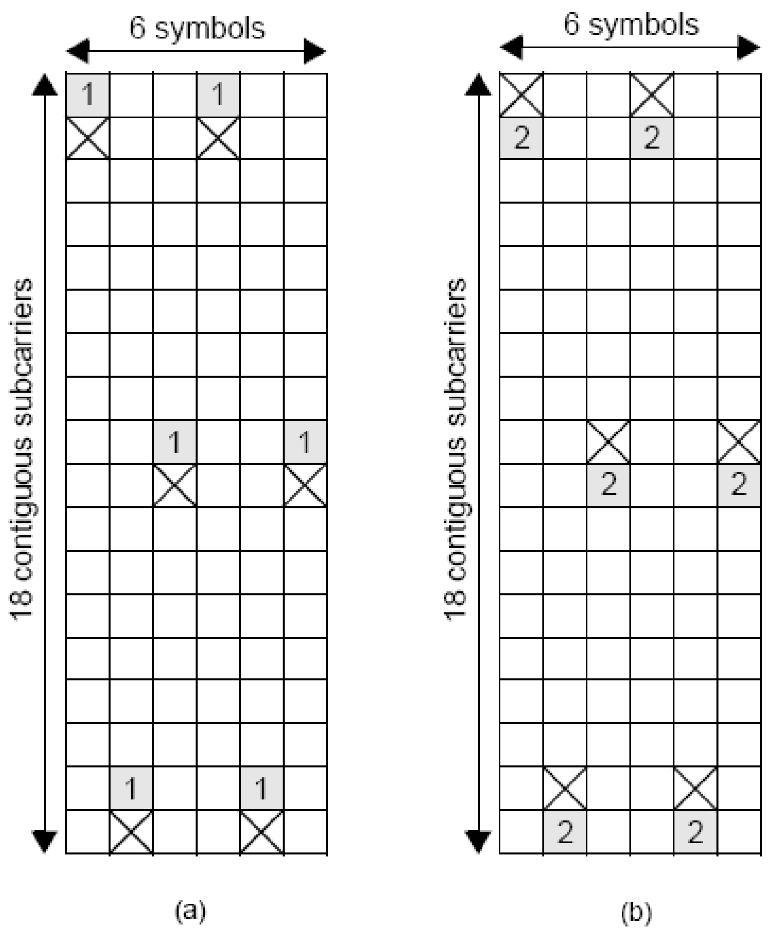


Figure 2.17: Pilot patterns used for 2 DL data streams (Fig. 506 in [11]).

The base pilot patterns used for two DL data streams in dedicated and common pilot scenarios are shown in Figure 2.17, where the subcarriers are indexed similarly to Figs. 2.16. Figs. 2.17(a) and 2.17(b) show the pilot locations for pilot streams 1 and 2 in a PRU, respectively. The number on a pilot subcarrier indicates the pilot stream the pilot subcarrier corresponds to. The subcarriers marked as “X” are null sub-carriers, on which no pilot or data is transmitted. In this thesis, we used the pilot structure for 2 DL data streams.

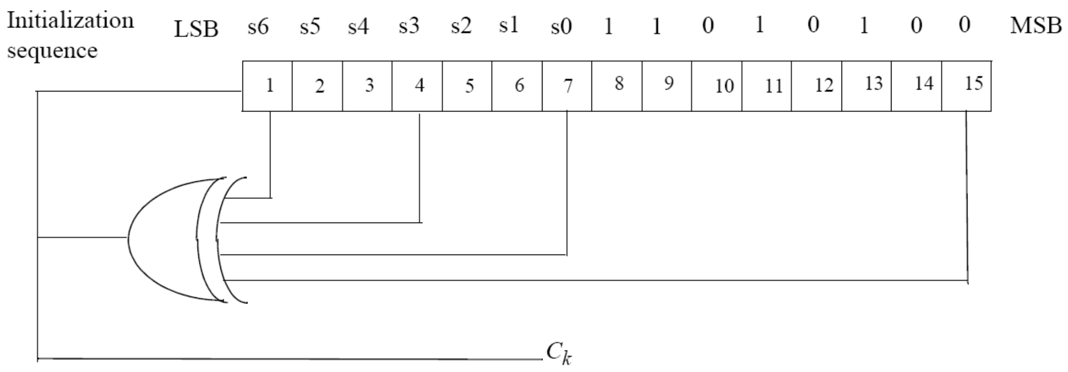


Figure 2.18: PRBS generator for pilot modulation (Figure 584 in [11]).

2.3.5.2 Pilot Modulation

Pilot subcarriers are inserted into each data burst in order to constitute the symbol. The PRBS (pseudo-random binary sequence) generator depicted in Fig. 2.18 is used to produce a sequence w_k. Each pilot is transmitted with a boosting of 2.5 dB over the average non-boosted power of each data tone. The pilot subcarriers is modulated according to

$$\Re\{c_k\} = \frac{8}{3}(\frac{1}{2} - w_k), \quad \Im\{c_k\} = 0. \tag{2.32}$$

2.3.6 Downlink MIMO Architecture and Data Processing

The architecture of downlink MIMO at the transmitter side is shown in Figure 2.19. The MIMO encoder block maps L MIMO layer ($L \geq 1$) onto M_t MIMO streams ($M_t \geq L$), which are fed to the precoder block. A MIMO layer is an information path fed to the MIMO encoder as an input. A MIMO layer represents one channel coding block. For the spatial multiplexing modes in single-user MIMO (SU-MIMO),“rank” is defined as the number of MIMO streams to be used for the user allocated to the Resource Unit (RU). For SU-MIMO, only one user is scheduled in one RU, and only one channel coding block exists at the input

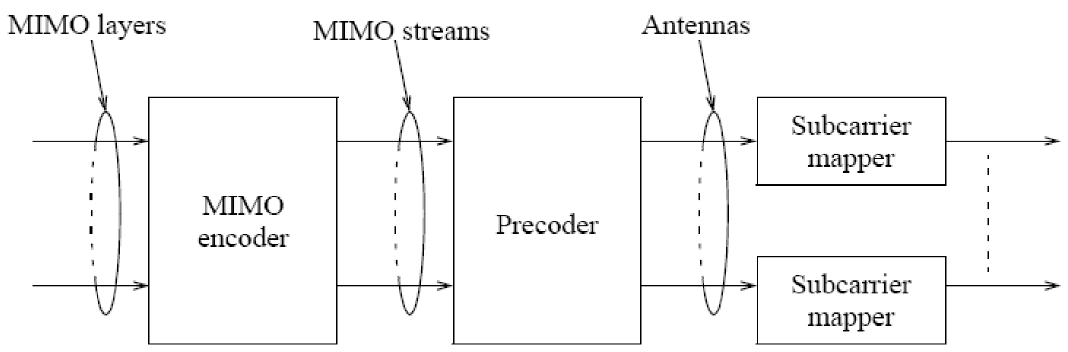


Figure 2.19: DL MIMO (Figure 547 in [11]).

of the MIMO encoder (vertical MIMO encoding at transmit side). For MU-MIMO, multiple users can be scheduled in one RU, and multiple channel coding blocks exist at the input of the MIMO encoder. The existence of multiple channel coding blocks at the input of the MIMO encoder can be caused by either using horizontal encoding or by using vertical encoding in several MIMO layers or by using a combination of vertical and horizontal encoding in several MIMO layers at the transmit side. Using multiple MIMO layers is called multi-layer encoding.

2.3.7 MIMO Layer to MIMO Stream Mapping

MIMO layer to MIMO stream mapping is performed by the MIMO encoder. The MIMO encoder is a batch processor that operates on M input symbols at a time. The input to the MIMO encoder is represent by an $M \times 1$ vector as

$$\mathbf{s} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_M \end{bmatrix} \tag{2.33}$$

where s_i is the ith input symbol within a batch. In case of MU-MIMO transmissions, the M symbols belong to different MSs. Two consecutive symbols may belong to a single MIMO layer. One MS shall have at most one MIMO layer.

MIMO layer to MIMO stream mapping of the input symbols is done in the space dimension first. The output of the MIMO encoder is an $M_t \times N_F$ MIMO STC matrix as

$$\mathbf{X} = \mathbf{S}(\mathbf{s}) \tag{2.34}$$

where

- M_t is the number of MIMO streams,
- N_F is the number of subcarriers occupied by one MIMO block,
- $\mathbf{X}$ is the output of the MIMO encoder,
- $\mathbf{s}$ is the input MIMO layer vector,
- $\mathbf{S}()$ is a function that maps an input MIMO layer vector to an STC matrix, and
- $\mathbf{S}(\mathbf{s})$ is an STC matrix.

The STC matrix $\mathbf{X}$ can be expressed as

$$\mathbf{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,N_F} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,N_F} \\ \cdots & \cdots & \cdots & \cdots \\ x_{M_t,1} & x_{M_t,2} & \cdots & x_{M_t,N_F}. \end{bmatrix}. \tag{2.35}$$

The four MIMO encoder formats (MEF) are space frequency block code (SFBC), vertical encoding (VE), multi-layer encoding (ME), and conjugate data repetition (CDR). For SU-MIMO transmissions, the STC rate is defined as

$$R = \frac{M}{N_F}. \tag{2.36}$$

For MU-MIMO transmissions, the STC rate per user (R) is equal to 1 or 2.

2.3.7.1 SFBC Encoding

The input to the MIMO encoder is represented by a 2×1 vector

$$\mathbf{s} = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}. \tag{2.37}$$

The MIMO encoder generates the SFBC matrix

$$\mathbf{X} = \begin{bmatrix} s_1 & -s_2^* \\ s_2 & s_1^* \end{bmatrix}. \tag{2.38}$$

where $\mathbf{X}$ is a 2×2 matrix. The SFBC matrix $\mathbf{X}$, occupies two consecutive subcarriers.

2.3.7.2 Vertical Encoding

The input and the output of MIMO encoder is represented by an $M \times 1$ vector

$$\mathbf{x} = \mathbf{s} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_M \end{bmatrix} \tag{2.39}$$

where s_i is the ith input symbol within a batch.

For vertical encoding, s_i, $i = 1, \ldots, M$, belong to the same MIMO layer. The encoder is an identity operation.

2.3.7.3 Multi-layer Encoding

The input and the output of MIMO encoder is represented by an $M \times 1$ vector

$$\mathbf{x} = \mathbf{s} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_M \end{bmatrix} \tag{2.40}$$

where s_i is the ith input symbol within a batch.

For multi-layer encoding, s_i, $i = 1, \ldots, M$, belong to different MIMO layers, where two consecutive symbols may belong to a single MIMO layer. Multi-layer encoding is only used for MU-MIMO mode. The encoder is an identity operation.

2.3.7.4 CDR Encoding

The input to the MIMO encoder is represented by a 1×1 vector

$$\mathbf{s} = s_1. \tag{2.41}$$

The MIMO encoder generates the CDR matrix

$$\mathbf{X} = \begin{bmatrix} s_1 & s_1^* \end{bmatrix}. \tag{2.42}$$

The CDR matrix **X** occupies two consecutive subcarriers.

2.3.7.5 Signal Reception for SFBC Encoding

In this these, we use the SFBC to MIMO stream mapping. Figure 2.20 shows the 2×1 SFBC structure in our system, where s_1 and s_2 are transmission signals, P_1 and P_2 are pilots, and i indicate the channel response for antenna i at subcarrier j. We encode and decode for a pair of subcarriers and use the least-square method to decode the received signal as

$$\mathbf{r} = \mathbf{H}\mathbf{s} + \mathbf{n},\ \mathbf{s} = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix}, \tag{2.43}$$

$$\begin{bmatrix} r_1 \\ r_2^* \end{bmatrix} = \begin{bmatrix} h_{11} & h_{21} \\ h_{22}^* & -h_{12}^* \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} + n, \tag{2.44}$$

$$\begin{bmatrix} \hat{s}_1 \\ \hat{s}_2 \end{bmatrix} = \frac{1}{h_{11}h_{12}^* + h_{21}h_{22}^*} \begin{bmatrix} r_1 h_{12}^* + r_2^* h_{21} \\ r_1 h_{22}^* - r_2^* h_{11} \end{bmatrix}. \tag{2.45}$$

For simplicity we use linear model, where r_i is received signal correspond to vector **r**, and **H** is channel response matrix, and s_i is transmission signal corresponding to vector **s**, where $\hat{s}_i$ is the estimation of s_i.

2.4 Uplink Transmission in IEEE 802.16m OFDMA [11]

In this section we introduce the UL specification of IEEE 802.16m. The material is mainly taken from [11]. In UL transmission, the permutation methods and the pilot structure are

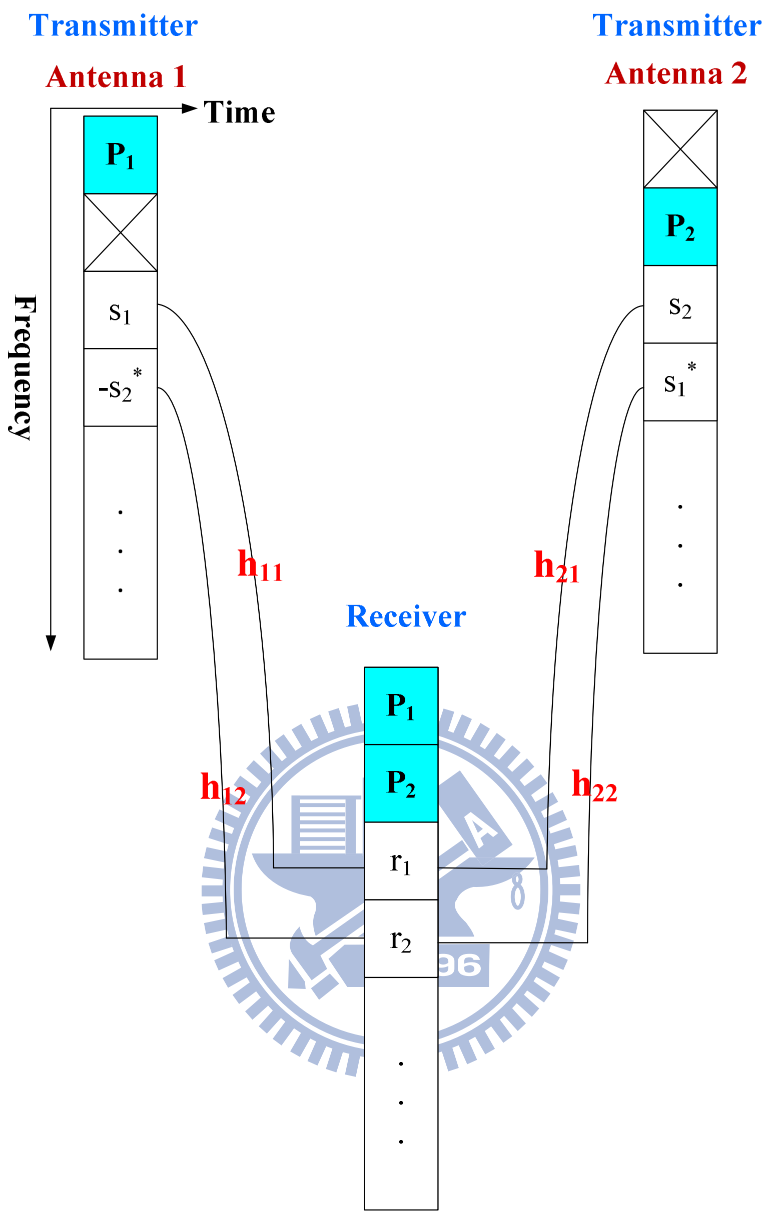


Figure 2.20: SFBC structure.

the same as in DL except DRU permutation which uses so called tile permutation. But in this thesis, we assume the UL data to be large and continuous, and hence we only use CRUs in the simulation and do not consider DRUs. So we do not consider the tile permutation.

By our choice only some terminology ha be modified. The overall signal organization story very similar to that for DL. Below is an itemized list of the modifications:

- Subband partitioning: replace Downlink Subband Allocation Count (DSAC) by Uplink Subband Allocation Count (USAC).
- Miniband partitioning: no change.
- Frequency partitioning: replace Downlink Frequency Partitioning Subband Count (DF-PSC) by Uplink Frequency Partitioning Subband Count (UFPSC).
- CRU/DRU allocation: replace Downlink CRU Allocation Size (DCAS) to Uplink CRU Allocation Size (UCAS).
- MIMO layer to MIMO stream mapping: no change.

Chapter 3

Channel Estimation Techniques

In this chapter, we briefly review three channel estimation method, which are the least-square technique (LS), linear interpolation and linear minimum mean-square error (LMMSE) estimation. We use the LS technique to estimate the channel responses at the pilots, use linear interpolation in the time domain to estimate the channel responses in some other symbols and use LMMSE estimation in the frequency domain to estimate the frequency response at various other nonpilot subcarriers. These techniques are introduced separately below, followed by how they are employed in the context of IEEE 802.16m.

3.1 Basic Channel Estimation Methods

3.1.1 Least-Squares (LS) Estimator

Based on the a priori known data, we can estimate the channel responses at pilot carriers roughly by the least-squares (LS) technique. An LS estimator minimizes the squared error [12]

$$||\mathbf{y} - \mathbf{H}_{LS}\mathbf{x}||^2 \tag{3.1}$$

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

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

Therefore, (3.1) can be rewritten as

$$[y(m) - x(m)h_{LS}(m)]^2, \text{ for all } m = m_i. \tag{3.3}$$

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

$$h_{LS}(m) = \frac{y(m)}{x(m)} = \frac{x(m)h(m) + n(m)}{x(m)} = h(m) + \frac{n(m)}{x(m)} \tag{3.4}$$

where $n(m)$ is the complex white Gaussian noise on subcarrier m. We may collect $h_{LS}(m)$ into $\mathbf{h}_{p,LS}$, an $N_p \times 1$ vector as

$$\begin{aligned} \mathbf{h}_{p,LS} = & \ [h_{p,LS}(0)\ h_{p,LS}(1) \cdots h_{p,LS}(N_p - 1)]^T \\ = & \ [\tfrac{y_p(0)}{x_p(0)}, \tfrac{y_p(1)}{x_p(1)}, \cdots, \tfrac{y_p(N_p-1)}{x_p(N_p-1)}]^T. \end{aligned} \tag{3.5}$$

where $h_{p,LS}(i)$ means the channel response uses LS method for estimation on ith pilot, and $y_p(i)$ means received signal on ith pilot and $x_p(i)$ means transmission signal on ith pilot. The LS estimator is a simplest channel estimator one can think of.

3.1.2 Linear Interpolation

After obtaining the channel response estimate at the pilot subcarriers, we use linear interpolation to obtain the responses at some other subcarriers. Exactly where apply it will be discussed later. Linear interpolation is a commonly considered scheme due to its low complexity. It does the interpolation between two known data. We use the channel estimate at two pilot subcarriers obtained by the LS estimator to estimate the channel frequency response information at the pilot subcarriers between them.

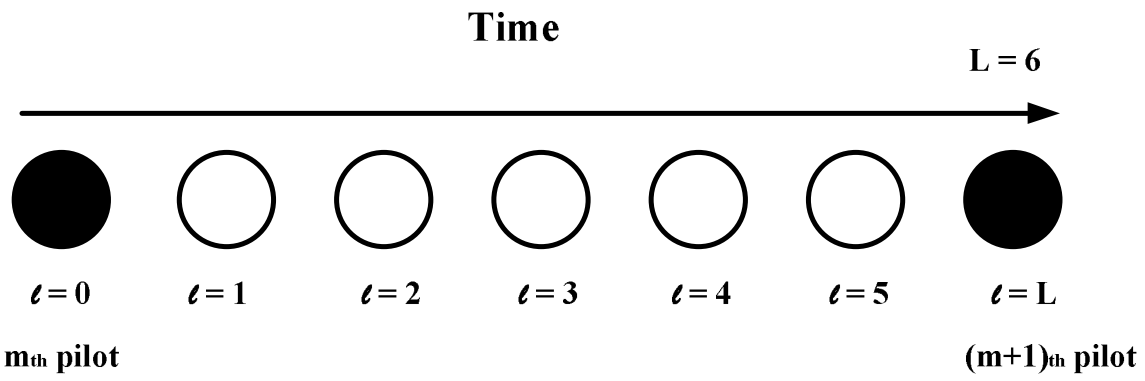


Figure 3.1: Linear interpolation.

Linear interpolation may be done in frequency or in time. We only perform it in time. The channel estimate at the other pilot subcarrier for time index k, $mL < k < (m+1)L$, using linear interpolation is given by [13]

$$h_e(k) = h_e(mL+l) = [h_p(m+1) - h_p(m)]\frac{l}{L} + h_p(m) \tag{3.6}$$

where $h_p(k)$, $k = 0, 1, \cdots, N_p$, are the channel frequency responses at pilot subcarriers, L is the pilot subcarriers spacing, and $0 < l < L$.

3.1.3 LMMSE Channel Estimation

We have given a brief introduction to the method in chapter 1. Now we describe it in more detail. The material in this section is mainly taken from [2], [3].

3.1.3.1 Channel Modeling for Channel Estimation

Consider a discrete-time equivalent lowpass channel impulse response

$$h(n) = \sum_{l=0}^{L-1} \alpha_l \delta(n-l) \tag{3.7}$$

where n and l are integers in units of the sampling period T_s and α_l is the complex gain of path l. The mean delay and the RMS delay spread are given by, respectively,

$$\tau_\mu = \frac{\sum_{l=0}^{L-1} E(|\alpha_l|^2)l}{\sum_{l=0}^{L-1} E(|\alpha_l|^2)} \tag{3.8}$$

and

$$\tau_{rms} = \sqrt{\frac{\sum_{l=0}^{L-1} E(|\alpha_l|^2)(l-\tau_\mu)^2}{\sum_{l=0}^{L-1} E(|\alpha_l|^2)}}. \tag{3.9}$$

One question here is how the expectation $E(|\alpha_l|^2)$ should be defined. As our purpose is channel estimation, suppose one channel estimation is performed for K OFDM symbols. Then the expectation should be an average that is taken over these symbol, note that we may let $k = 1$. In addition, we assume that the channel estimator input contains no carrier frequency error, but the PDP can have a nonzero initial delay τ_0, although conventional definition of the PDP usually zero out the initial path delays.

Fourier transforming the PDP gives the corresponding frequency autocorrelation function. For an exponential PDP with initial delay τ_0, we have

$$\frac{R_f(k)}{R_f(0)} = \frac{e^{-j2\pi\tau_0 k/N}}{1 + j2\pi\tau_{rms}k/N} \tag{3.10}$$

where $\tau_0 = \tau_\mu - \tau_{rms}$ and N is the DFT size used in the multicarrier system. For a uniform PDP of width T with a initial delay τ_0,

$$\frac{R_f(k)}{R_f(0)} = \frac{e^{-j2\pi\tau_0 k/N}\sin(\pi Tk/N)}{\pi Tk/N} \tag{3.11}$$

where $\tau_\mu = \tau_0 + T/2$ and $T = \sqrt{12}\tau_{rms}$.

3.1.3.2 Estimation of Channel Delay Parameters

The frequency response of the channel in (3.7) is given by

$$H(f) = \sum_{l=0}^{L-1} \alpha_l e^{-j2\pi lf/N} \tag{3.12}$$

where the division by N in the exponent normalizes the period of $H(f)$ in f to N. If we advance the channel response by τ (arbitrary) time units, then the frequency response becomes

$$H_a(f) = e^{j2\pi\tau f/N}H(f) = \sum_{l=0}^{L-1}\alpha_l e^{-j2\pi(l-\tau)f/N}. \tag{3.13}$$

Differentiating $H_a(f)$ with respect to f, we get

$$\frac{dH_a(f)}{df} = \frac{-j2\pi}{N}\sum_{l=0}^{L-1}\alpha_l(l-\tau)e^{-j2\pi(l-\tau)f/N}. \tag{3.14}$$

Applying Parsevals theorem, we get

$$J(\tau) \triangleq \left\langle |\frac{dH_a(f)}{df}|^2 \right\rangle = \frac{4\pi^2}{N^2}\sum_{l=0}^{L-1}|\alpha_l|^2(l-\tau)^2 \tag{3.15}$$

where $< \cdot >$ denotes frequency averaging. Hence

$$\overline{J}(\tau) \triangleq E(\left\langle |\frac{dH_a(f)}{df}|^2 \right\rangle) = \frac{4\pi^2}{N^2}\sum_{l=0}^{L-1}E((|\alpha_l|^2)(l-\tau)^2). \tag{3.16}$$

The above equations show that $\overline{J}(\tau)$ is minimized when $\tau = \tau_\mu$. In addition,

$$\tau_{rms}^2 = \frac{N^2 \min \overline{J}(\tau)}{4\pi^2\sum_{l=0}^{L-1}E(|\alpha_l|^2)}. \tag{3.17}$$

For pilot-transmitting OFDM systems, the below given a way to find τ_μ and τ_{rms} from the frequency domain channel estimate.

Consider a system where one out of every F_s subcarriers is a pilot. We can approximate $dH_a(f)/df$ by the first-order difference, $[H_a(f+F_s)-H_a(f)]/F_s$ and substitute it into (3.16). Then, we obtain

$$\overline{J}(\tau) \approx \frac{1}{F_s^2}E\left\langle |e^{j\phi}H(f+F_s) - H(f)|^2 \right\rangle_p \tag{3.18}$$

where $\phi = 2\pi\tau F_s/N$, f takes values only over pilot frequencies, and $< \cdot >_p$ denotes averaging over pilot subcarriers. Then we modify the approximation by taking circular differencing over f rather than linear differencing. Therefore, we approximate $\overline{J}(\tau)$ by

$$\overline{J}(\tau) \approx \frac{1}{F_s^2}E|\left\langle |e^{j\phi}H\ ((f+F_s)\%N)\ - H(f)|^2 \right\rangle_p \tag{3.19}$$

where % denotes modulo operation, and we have assumed that $(f+F_s)\%N$ is a pilot subcarrier. Now let R_i be the (instantaneous) frequency-domain autocorrelation of the channel response:

$$R_i = \Big\langle H((f+iF_s)\ \%N)H^*(f)\Big\rangle_p. \tag{3.20}$$

Then from (3.19) we have

$$\overline{J}(\tau) \approx \frac{2}{F_s^2}[E(R_0 - \Re\{e^{j\phi}E(R_1)\})]. \tag{3.21}$$

Then (3.21) gives an approximation of $\overline{J}(\tau)$ defined in (3.16). According to (3.21), τ_μ and τ_{rms} can be estimated in the following way:

1. estimate the channel responses at the pilot subcarriers,
2. estimate R_i $(i=0,1)$,
3. estimate $\overline{J}(\tau)$,
4. find the value of τ that minimizes $\overline{J}(\tau)$, and
5. substitute the result into (3.17) to estimate τ_{rms}^2.

Step 1 can be achieved using the LS method. For step 2, R_0 and R_1 can be estimated via

$$R_0 = \Big\langle |H(f)|^2\Big\rangle_p - \sigma_n^2, \quad R_1 = \Big\langle H((f+F_s)\ \%N)H^*(f)\Big\rangle_p, \tag{3.22}$$

where we assume that σ_n^2 has been estimated in some way, such as from the received power in the null subcarriers. Thus, for step 3, $\overline{J}(\tau)$ can be estimated using

$$J_{Av}(\tau) \triangleq \frac{2}{F_s^2}\Big[Av(R_0) - R\{e^{j\phi}Av(R_1)\}\Big] \tag{3.23}$$

where Av denotes time averaging, i.e., averaging over OFDM symbols say k. For step 4, we may estimate the mean delay as

$$\tau_\mu \triangleq \arg\min J_{Av}(\tau) = -\frac{N\angle Av(R_1)}{2\pi F_s}, \tag{3.24}$$

which also yields min $J_{Av}(\tau) = 2[Av(R_0) - |Av(R_1)|]/F_s^2$. Finally, for step 5, in view of (3.17) and that $R_0 =< |H(f)|^2 >_p$, we may estimate τ_{rms} as

$$\tau_{rms} = \frac{N}{2\pi F_s}\sqrt{2\left[1 - \frac{|Av(R_1)|}{Av(R_0)}\right]}. \tag{3.25}$$

3.1.3.3 LMMSE Filtering

To complete LMMSE channel estimation, the above estimates of delay parameters, namely, τ_μ and τ_{rms}, can be substituted into proper places in (3.10) or (3.11) depending on the choice of PDP model. Then the resulting autocorrelation function of channel frequency response can be used in LMMSE channel estimation as outlined in chapter 1.

3.2 Application to IEEE 802.16m

3.2.1 LMMSE Channel Estimation for IEEE 802.16m

As mentioned previously, We employ the technique of [2], [3]. The LMMSE channel estimation for IEEE 802.16m proceeds as follows.

1. Do LS channel estimation at pilot subcarriers of all used PRUs.

2. Linearly interpolate in time to acquire two additional channel estimates per PRU per symbol, as shown by the arrows in Fig. 3.2.

3. Estimate R_0 and R_1 as explained below.

4. Estimate τ_μ and τ_{rms} as discussed below.

5. Find the autocorrelation function associated with the exponential PDP as

$$\frac{R_f(k)}{R_f(0)} = \frac{e^{-j2\pi\tau_0 k/N}}{1 + j2\pi\tau_{rms}k/N}, \tag{3.26}$$

   where $\tau_0 = \tau_\mu - \tau_{rms}$ and N is the DFT size used in the multicarrier system.

6. Based on the above autocorrelation function, do LMMSE filtering to estimate the data subcarrier responses as

$$\mathbf{w}_d = (\mathbf{R}_{pp} + \sigma_n^2\mathbf{I})^{-1}\mathbf{r}_{dp}, \tag{3.27}$$

$$h_d = \mathbf{w}_d^h \mathbf{h}_p. \tag{3.28}$$

Let $H(f)$ denote the resulting channel estimate at pilot subcarrier f from step 1. Let σ_n^2 be the variance of additive white gaussian noise (AWGN) at the pilot locations. After linear interpolation from two pilots in step 2, the noise variance at the interpolated locations has to be modify (see Sections. 3.2.3 and 3.2.4). For step 3, R_0 and R_1 can be estimated via

$$R_0 = \left\langle |H(f)|^2 \right\rangle_p - \sigma_n^2, \quad R_1 = \left\langle H((f + F_s)\ \%N)H^*(f) \right\rangle_p, \tag{3.29}$$

where $\left\langle |H(f)|^2 \right\rangle_p$ is the averaged magnitude-squares of estimation pilot subcarrier channel responses, σ_n^2 is the estimated noise variance at pilot subcarriers, and $F_s = 8$ (there is one pilot subcarrier every 8 subcarriers in IEEE 802.16m)). According to the signal structure of 16m, σ_n^2 here should be calculate as discussed in Sections. 3.2.3 and 3.2.4. For step 4, we may estimate the mean delay as

$$\tau_\mu = -\frac{N\angle Av(R_1)}{2\pi F_s}. \tag{3.30}$$

And we may estimate the RMS delay spread as

$$\tau_{rms} = \frac{N}{2\pi F_s}\sqrt{2\left[1 - \frac{|Av(R_1)|}{Av(R_0)}\right]}, \tag{3.31}$$

in our present work, we do delay estimation based on only one OFDM symbol for simplicity and for better performance in a time-varying channel. Hence we estimate R_0 and R_1 for each OFDM symbol separately.

3.2.2 Linear Interpolation in Time

Before LMMSE channel estimation, we first estimate the pilot responses for each symbol. However, more because each OFDMA symbol contains only one pilot in each PRU, we need

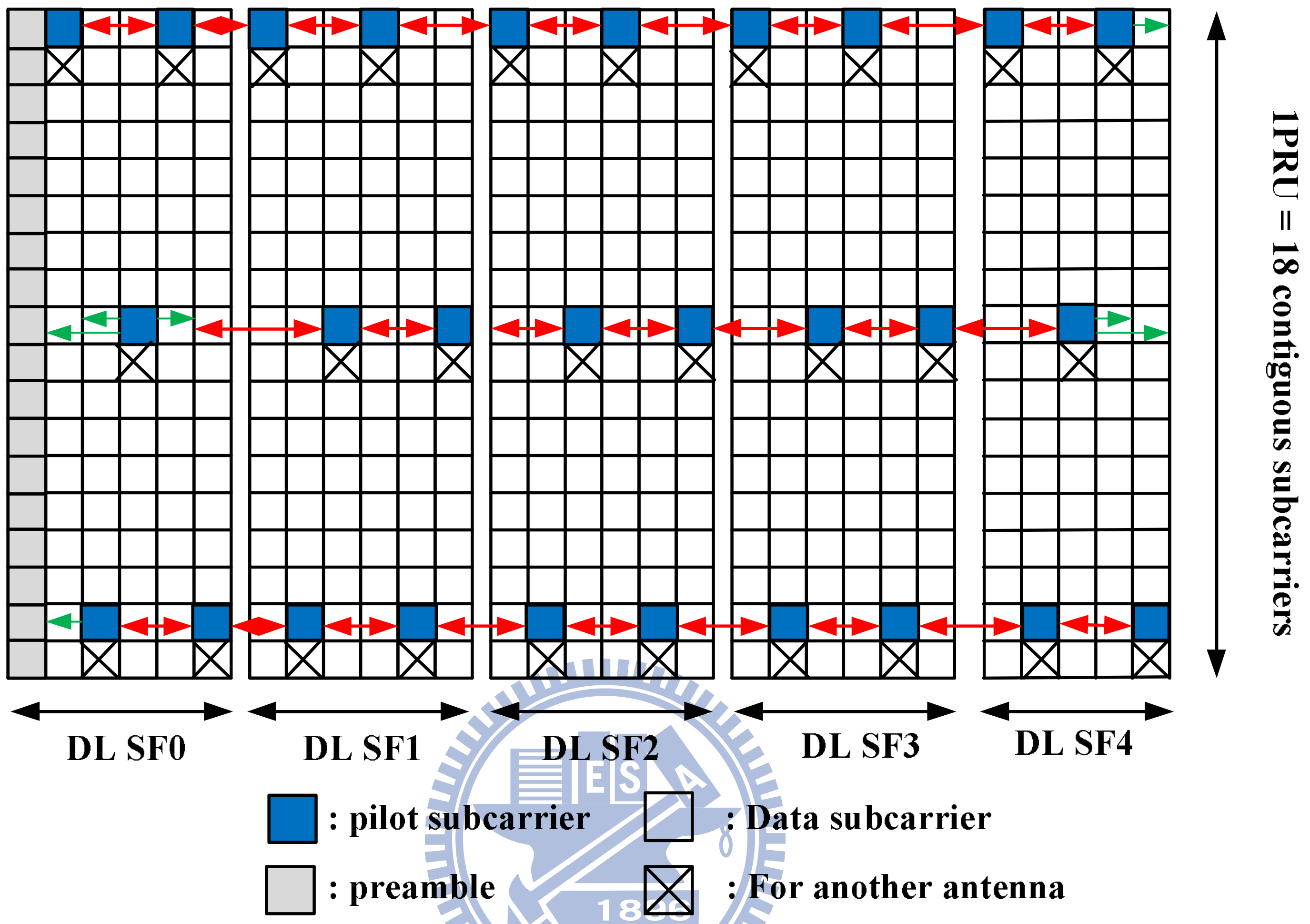


Figure 3.2: Linear interpolation in time domain for 16m downlink signal.

to generate (equivalent) pilot response to facilitate the LMMSE operation in the frequency domain. Fig. 3.2 shows the DL signal structure of IEEE 802.16m. We use linear interpolation in the time domain to generate the additional pilot response required, as indicate by the arrows in Fig. 3.2.

The detailed steps for each OFDM symbol are as follows.

1) First to third data symbols:

- Estimate the channel response at each pilot location using the LS technique.

- Copy the pilot response estimate in the nearest symbols to the same subcarrier location in the present symbol to be the equivalent pilot response estimate mentioned abpve.

2) Fourth to $(n-3)$th data symbols:

- Estimate the channel response at each pilot location by using the LS technique.
- Use linear interpolation in time to estimate the channel response at each pilot location of a nearest previous and a nearest next OFDM symbols.

3) $(n-2)$th to the last nth data symbols:

- Estimate the channel response at each pilot location using the LS technique.
- Use the estimated channel responses at pilot locations of the nearest previous symbols as the equivalent pilot response estimate.

This method gives our system a three-symbol time latency. For example, if we want to get the channel response of the first symbol, we must wait until we receive the fourth data symbol. But in the last three symbols, this latency is made up.

3.2.3 Noise Variance Variation in DL Channel Estimation

Due to the LS channel estimation and time-domain linear interpolation, the value of σ_n^2 used in the LMMSE channel estimation varies with frequency and with time. First, because the pilots are power boosted by a factor of 16/9, the variance of estimation error is 9/16 times the AWGN variance. In addition, the value has to be multiplied by a scale factor according to the coefficient of the linear interpolation at equivalent pilot location, which can be classified into three cases for the downlink as shown in Fig. 3.3. In one case, there is one subcarrier channel response interpolated using two adjacent pilot responses. The error variance at the subcarrier is $1/2 \times \sigma_n^2$. In another case, there are two temporally consecutive subcarrier responses interpolated using one previous and one succeeding pilot responses in time. The error variance at these two subcarrier locations is $5/9 \times \sigma_n^2$. In the last case, there are four temporally consecutive subcarrier response interpolation using one previous and one succeeding pilot response int time. The error variance σ_n^2 at the first and the 4th subcarrier location is $17/25 \times \sigma_n^2$, and that at the 2nd and the 3rd subcarrier locations is $13/25 \times \sigma_n^2$. In summary, the average pilot subcarrier estimation error variance for each of the 28 data symbols of the DL sunframe is:

- 1st symbol: $(1 + 1 + 1)/3 \times 9/16 \times \sigma_n^2$,
- 2nd symbol: $(1 + 1 + 5/9)/3 \times 9/16 \times \sigma_n^2$,

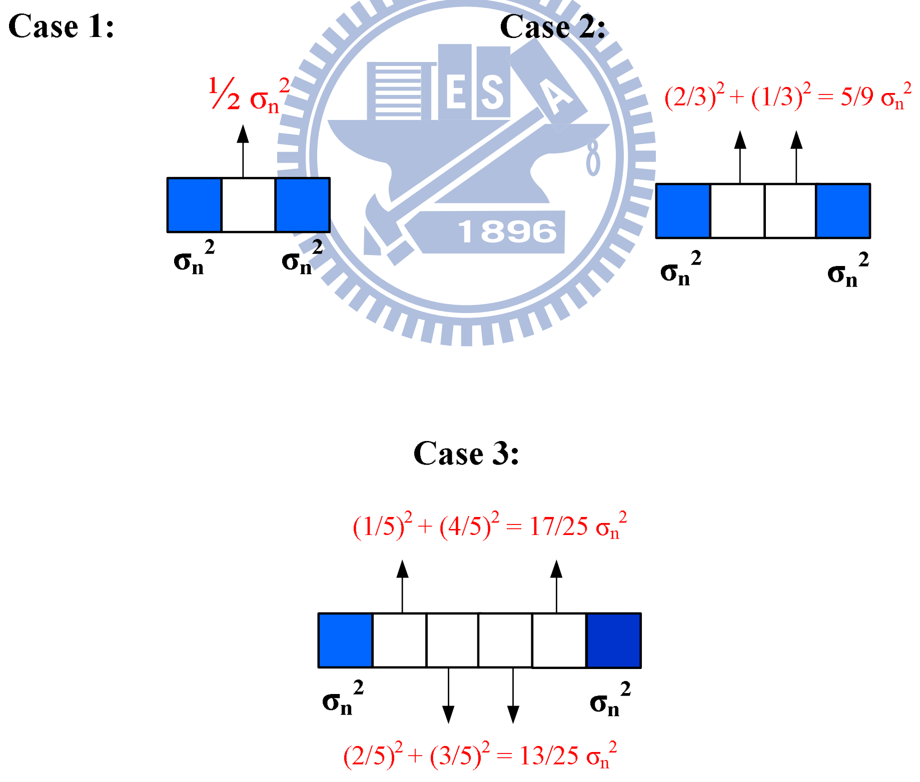


Figure 3.3: Noise variance variation for different ways of interpolation.

- 3rd symbol: $(1+5/9+5/9)/3 \times 9/16 \times \sigma_n^2$,
- 4th symbol: $(1+1+5/9)/3 \times 9/16 \times \sigma_n^2$,
- 5th symbol: $(1+1/2+13/25)/3 \times 9/16 \times \sigma_n^2$,
- 6th symbol: $(5/9+1/2+1)/3 \times 9/16 \times \sigma_n^2$,
- 7th symbol: $(1+5/9+5/9)/3 \times 9/16 \times \sigma_n^2$,
- 8th to 25th symbols: $(1+5/9+5/9)/3 \times 9/16 \times \sigma_n^2$,
- 26th symbol: $(1+5/9+5/9)/3 \times 9/16 \times \sigma_n^2$,
- 27th symbol: $(1+1+5/9)/3 \times 9/16 \times \sigma_n^2$, and
- 28th symbol: $(1+1+1)/3 \times 9/16 \times \sigma_n^2$.

3.2.4 Noise Variance Variation in UL Channel Estimation

The variation in estimation error variance for UL pilot responses can be calculated in a similar way as in the DL, but simpler because there is only one kind of time-domain interpolation. In summary, the average noise variance for each of the 18 data symbols of the UL subframe is as follows.

- 1st symbol: $(1+1+1)/3 \times 9/16 \times \sigma_n^2$,
- 2nd symbol: $(1+1+5/9)/3 \times 9/16 \times \sigma_n^2$,
- 3rd to 16th symbols: $(1+5/9+5/9)/3 \times 9/16 \times \sigma_n^2$,
- 17th symbol: $(1+1+5/9)/3 \times 9/16 \times \sigma_n^2$, and
- 18th symbol: $(1+1+1)/3 \times 9/16 \times \sigma_n^2$.

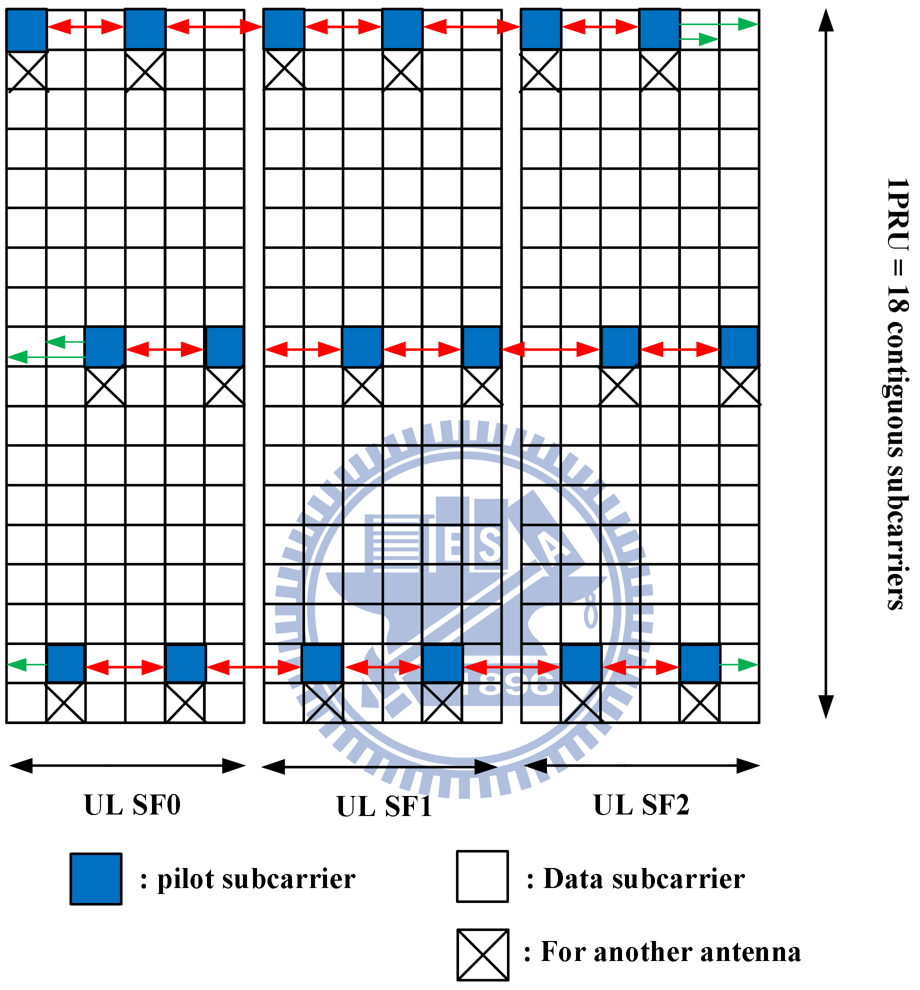


Figure 3.4: Linear interpolation in time domain for 16m uplink signal.

Chapter 4

Floating-Point Simulation

In this chapter, we perform floating-point simulation for IEEE 802.16m channel estimation using the LMMSE method. In preparation for the DSP implementation, the simulation employs a C/C++ platform. We evaluate the performance via mean-square-error (MSE) and symbol error rate (SER).

4.1 System Parameters and Channel Model

Table 4.1 shows the parameters used in our simulation work. For the downlink, TDD frame length = 5 ms, DL subframe size = 1 preamble + 28 OFDM symbols, and 48 PRUs are used in transmission. And for the Uplink, TDD frame length = 5 ms, UL subframe size = 18 OFDM symbols, and 48 PRUs are used in transmission. In addition to AWGN, we use SUI-1 to SUI-6 for simulation. Their channel profiles are as shown in Tables 4.2 to 4.4. Erceg *et al.* [14] 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,
- C: flat terrain, light tree, and

Table 4.1: OFDMA Downlink and Uplink Parameters

Parameters	Values
Bandwidth	10 MHz
Central frequency	3.5 GHz
N_{used}	865
Sampling factor n	28/25
G	1/8
N_{FFT}	1024
Sampling frequency	11.2 MHz
Subcarrier spacing	10.94 kHz
Useful symbol time	91.43 μs
CP time	11.43 μs
OFDMA symbol time	102.86 μs
Sampling time	44.65 ns

- B: between A and C.

The correspondence with the so-called SUI channels is:

- 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. 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 SUI-1 to SUI-6 model in our simulation, but we use Rayleigh fading to model all the paths in these channels.

Table 4.2: Profiles of SUI-1 and SUI-2 Channels [14]

SUI – 1 Channel

	Tap 1	**Tap 2**	**Tap 3**	**Units**
Delay	0	0.4	0.9	μs
Power (omni ant.)	0	-15	-20	dB
90% K-fact. (omni)	4	0	0	
75% K-fact. (omni)	20	0	0	
Power (30° ant.)	0	-21	-32	dB
90% K-fact. (30°)	16	0	0	
75% K-fact. (30°)	72	0	0	
Doppler	0.4	0.3	0.5	Hz

Antenna Correlation: $\rho_{ENV} = 0.7$ **Gain Reduction Factor:** GRF = 0 dB **Normalization Factor:** $F_{omni} = -0.1771$ dB, $F_{30^\circ} = -0.0371$ dB	**Terrain Type:** C **Omni antenna:** $\tau_{RMS} = 0.111$ μs, overall K: K = 3.3 (90%); K = 10.4 (75%) **30° antenna:** $\tau_{RMS} = 0.042$ μs, overall K: K = 14.0 (90%); K = 44.2 (75%)

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: $\rho_{ENV} = 0.5$ **Gain Reduction Factor:** GRF = 2 dB **Normalization Factor:** $F_{omni} = -0.3930$ dB, $F_{30^\circ} = -0.0768$ dB	**Terrain Type:** C **Omni antenna:** $\tau_{RMS} = 0.202$ μs, overall K: K = 1.6 (90%); K = 5.1 (75%) **30° antenna:** $\tau_{RMS} = 0.069$ μs, overall K: K = 6.9 (90%); K = 21.8 (75%)

Table 4.3: Profiles of SUI-3 and SUI-4 Channels [14]

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: $\rho_{ENV} = 0.4$ **Gain Reduction Factor:** GRF = 3 dB **Normalization Factor:** $F_{omni} = -1.5113$ dB, $F_{30°} = -0.3573$ dB		**Terrain Type:** B **Omni antenna:** $\tau_{RMS} = 0.264$ μs, overall K: K = 0.5 (90%); K = 1.6 (75%) **30° antenna:** $\tau_{RMS} = 0.123$ μs, overall K: K = 2.2 (90%); K = 7.0 (75%)		

SUI – 4 Channel				
	Tap 1	**Tap 2**	**Tap 3**	**Units**
Delay	0	1.5	4	μs
Power (omni ant.)	0	-4	-8	dB
90% K-fact. (omni)	0	0	0	
75% K-fact. (omni)	1	0	0	
Power (30° ant.)	0	-10	-20	dB
90% K-fact. (30°)	1	0	0	
75% K-fact. (30°)	5	0	0	
Doppler	0.2	0.15	0.25	Hz
Antenna Correlation: $\rho_{ENV} = 0.3$ **Gain Reduction Factor:** GRF = 4 dB **Normalization Factor:** $F_{omni} = -1.9218$ dB, $F_{30°} = -0.4532$ dB		**Terrain Type:** B **Omni antenna:** $\tau_{RMS} = 1.257$ μs overall K: K = 0.2 (90%); K = 0.6 (75%) **30° antenna:** $\tau_{RMS} = 0.563$ μs overall K: K = 1.0 (90%); K = 3.2 (75%)		

Table 4.4: Profiles of SUI-5 and SUI-6 Channels [14]

SUI – 5 Channel				
	Tap 1	**Tap 2**	**Tap 3**	**Units**
Delay	0	4	10	μs
Power (omni ant.)	0	-5	-10	dB
90% K-fact. (omni)	0	0	0	
75% K-fact. (omni)	0	0	0	
50% K-fact (omni)	2	0	0	
Power (30° ant.)	0	-11	-22	dB
90% K-fact. (30°)	0	0	0	
75% K-fact. (30°)	2	0	0	
50% K-fact. (30°)	7	0	0	
Doppler	2	1.5	2.5	Hz
Antenna Correlation: $\rho_{ENV} = 0.3$ **Gain Reduction Factor:** GRF = 4 dB **Normalization Factor:** $F_{omni} = -1.5113$ dB, $F_{30^\circ} = -0.3573$ dB		**Terrain Type:** A **Omni antenna:** $\tau_{RMS} = 2.842$ μs overall K: K = 0.1 (90%); K = 0.3 (75%); K = 1.0 (50%) **30° antenna:** $\tau_{RMS} = 1.276$ μs overall K: K = 0.4 (90%); K = 1.3 (75%); K = 4.2 (50%)		

SUI – 6 Channel				
	Tap 1	**Tap 2**	**Tap 3**	**Units**
Delay	0	14	20	μs
Power (omni ant.)	0	-10	-14	dB
90% K-fact. (omni)	0	0	0	
75% K-fact. (omni)	0	0	0	
50% K-fact. (omni)	1	0	0	
Power (30° ant.)	0	-16	-26	dB
90% K-fact. (30°)	0	0	0	
75% K-fact. (30°)	2	0	0	
50% K-fact. (30°)	5	0	0	
Doppler	0.4	0.3	0.5	Hz
Antenna Correlation: $\rho_{ENV} = 0.3$ **Gain Reduction Factor:** GRF = 4 dB **Normalization Factor:** $F_{omni} = -0.5683$ dB, $F_{30^\circ} = -0.1184$ dB		**Terrain Type:** A **Omni antenna:** $\tau_{RMS} = 5.240$ μs overall K: K = 0.1 (90%); K = 0.3 (75%); K = 1.0 (50%) **30° antenna:** $\tau_{RMS} = 2.370$ μs overall K: K = 0.4 (90%); K = 1.3 (75%); K = 4.2 (50%)		

4.2 DL Simulation Results

4.2.1 Simulation Flow

Figure 4.1 illustrates the block diagram of our simulated system. We assume perfect synchronization and omit it in our simulation. We also omit error control coding and decoding. After channel estimation, we calculate the channel MSE between the true channel and the estimated one, where the average is taken over all the subcarriers. The symbol error rate (SER) can also be obtained after demapping. For this section, we only focus on channel estimation part for simulation and Fig. 4.2 shows the details of the LMMSE channel estimation process.

We verify the correctness of the program code by simulation AWGN channel transmission. Both SISO and SFBC MIMO transmission are consider. We run the 10^5 symbols in each simulation to obtain the numerical results. Figs. 4.3 shows the DL channel estimation performance in SISO transmission for AWGN channel. We see that the simulation results are in agreement with the theory. We also see that LMMSE channel estimation performs better than 2D linear interpolation (using linear interpolation in time and frequency to get channel response) by about 2 dB in MSE. Here, E_s/N_0 means the average transmitted signal power divided by No $= \sigma_n^2$. Figure. 4.4 shows the performance in SFBC DL transmission over AWGN channel.

4.2.2 SISO Transmission Results

We conduct simulation with the six SUI channels to examine the channel estimation performance. To check with the simulation curve in [2], our simulation has same results to the [2]. As expected, If the velocity becomes higher, the channel varies faster in time. Therefore, the performance with a higher velocity is worse. For SUI-1 and SUI-2, the delay spreads are

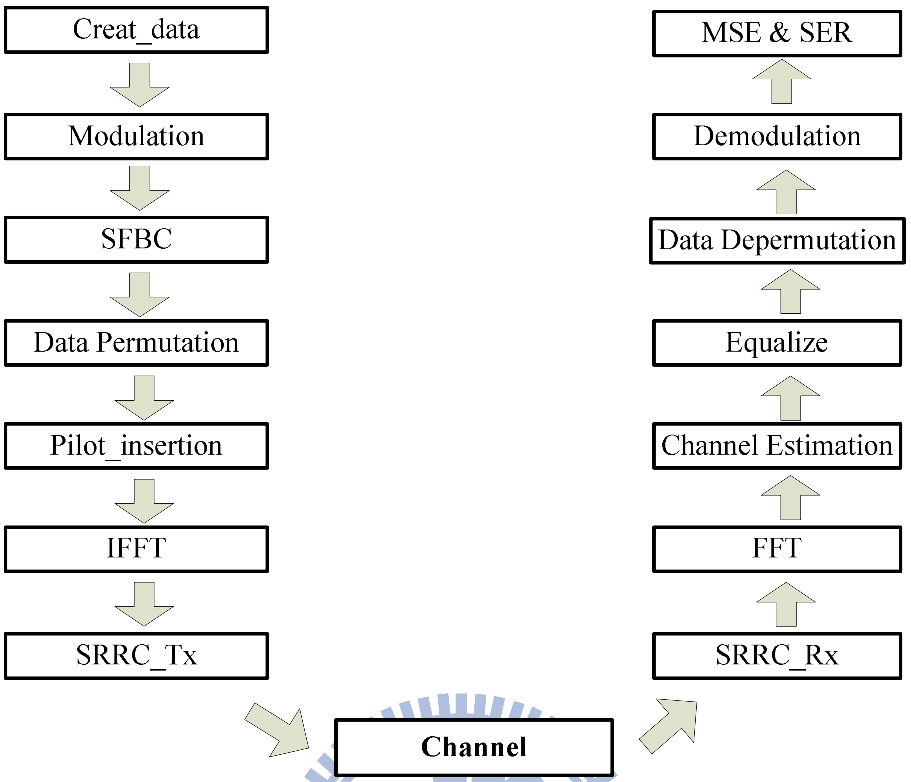


Figure 4.1: Simulation blok diagram for the transmission system.

smaller than other channels, so the performance is the best of all channels. For SUI-3 and SUI-4, the delay spreads are moderate among all channels and the performance is also in the middle. For SUI-5, the delay spread is close to the CP length, so the performance is worse than other channels except SUI-6. For SUI-6, because the delay spread is greater than the CP length, the performance is the worst of all channels and the MSE performance of channel estimation different velocities become hardly distinguishable. Please refer to appendix A for the numerical results.

4.2.3 SFBC Transmission Results

SFBC is a technique that uses multiple antennas to achieve better diversity effect. In our simulation, we use two transmit antennas and one receive antenna. We estimate the two

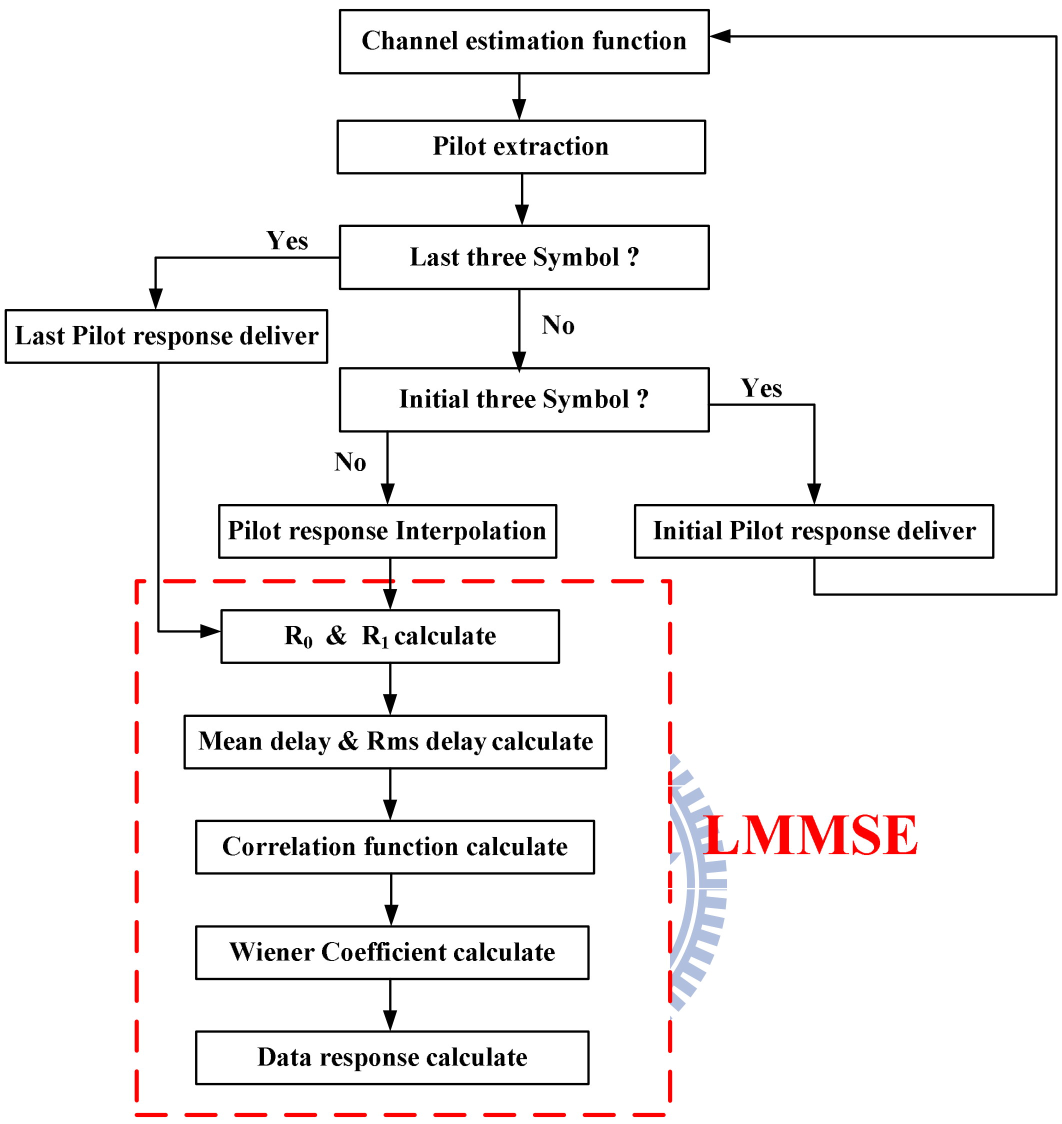

Channel estimation function
Pilot extraction
Last three Symbol ?
Yes
Last Pilot response deliver
No
Initial three Symbol ?
Yes
No
Pilot response Interpolation
Initial Pilot response deliver
R_0 & R_1 calculate
Mean delay & Rms delay calculate
Correlation function calculate
Wiener Coefficient calculate
Data response calculate
LMMSE

Figure 4.2: Simulation blok diagram for the channel estimation.

channel responses separately. So the MSE performance is not different by using SFBC as compared to SISO, but the SER performance is better. To check with our results, it is also agreement to with theory. Appendix B contains the numerical results.

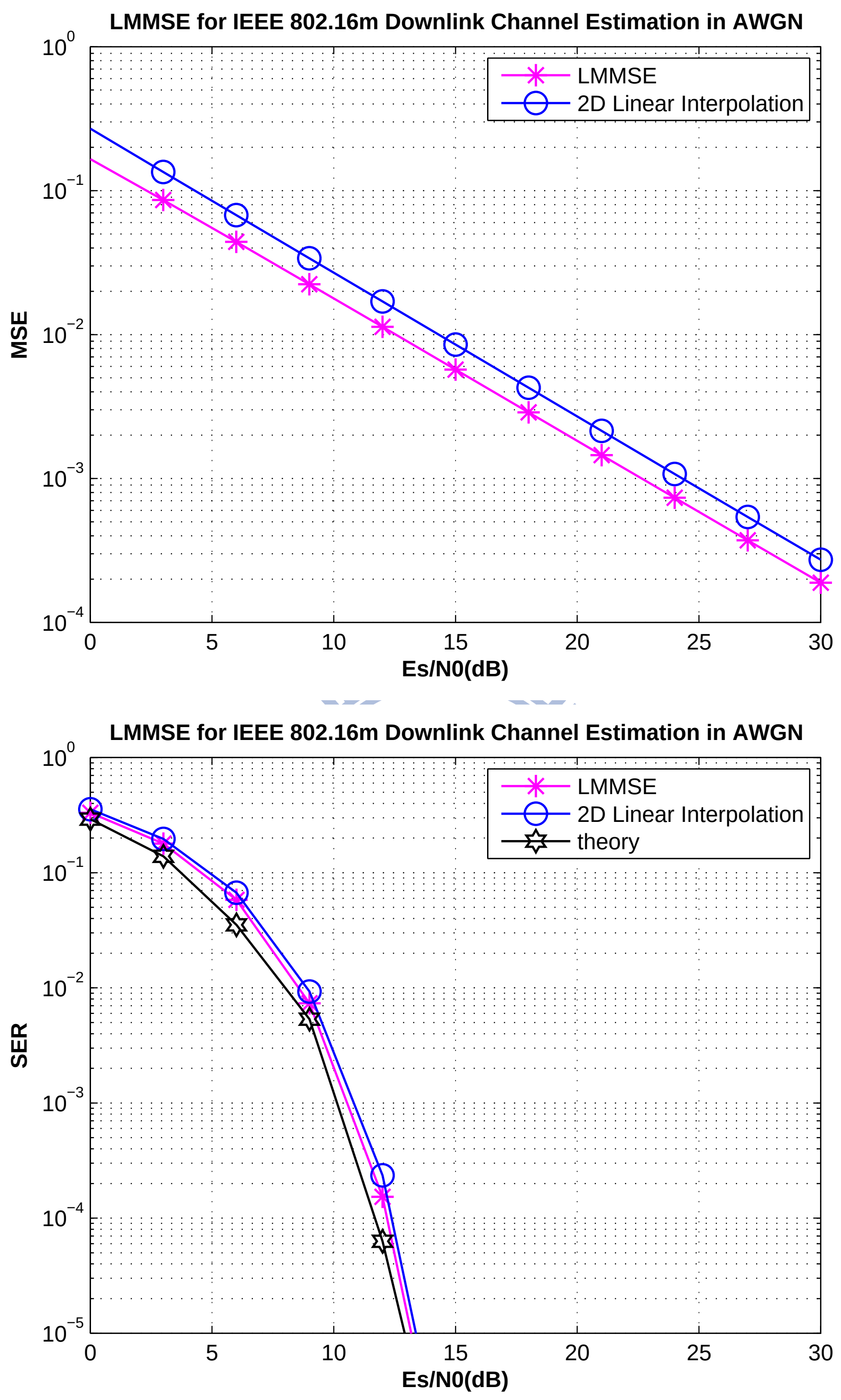
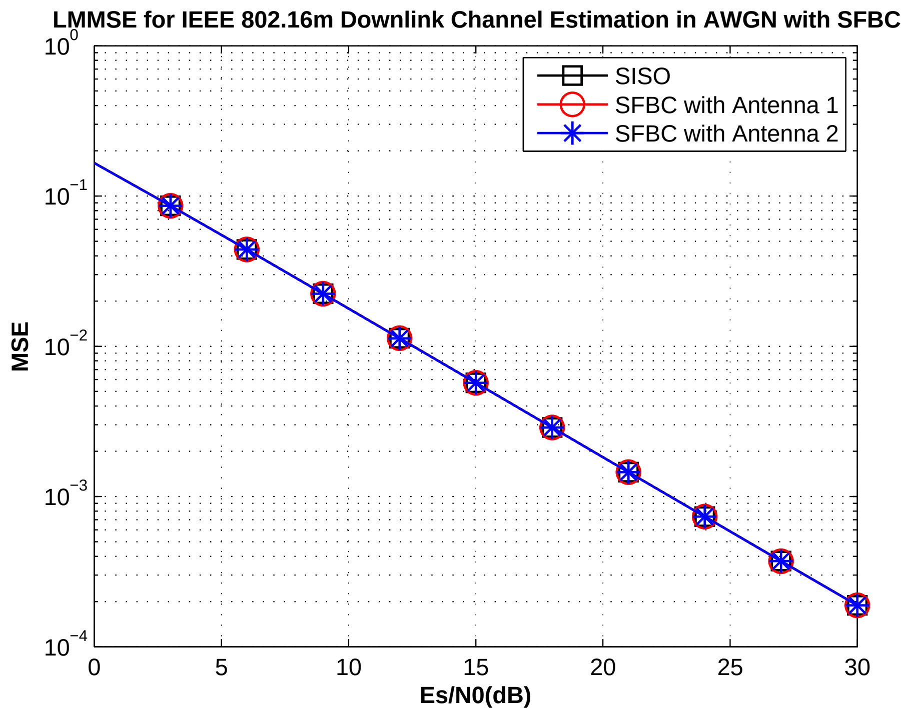


Figure 4.3: Channel estimation MSE and SER for QPSK in AWGN for IEEE 802.16m SISO downlink.



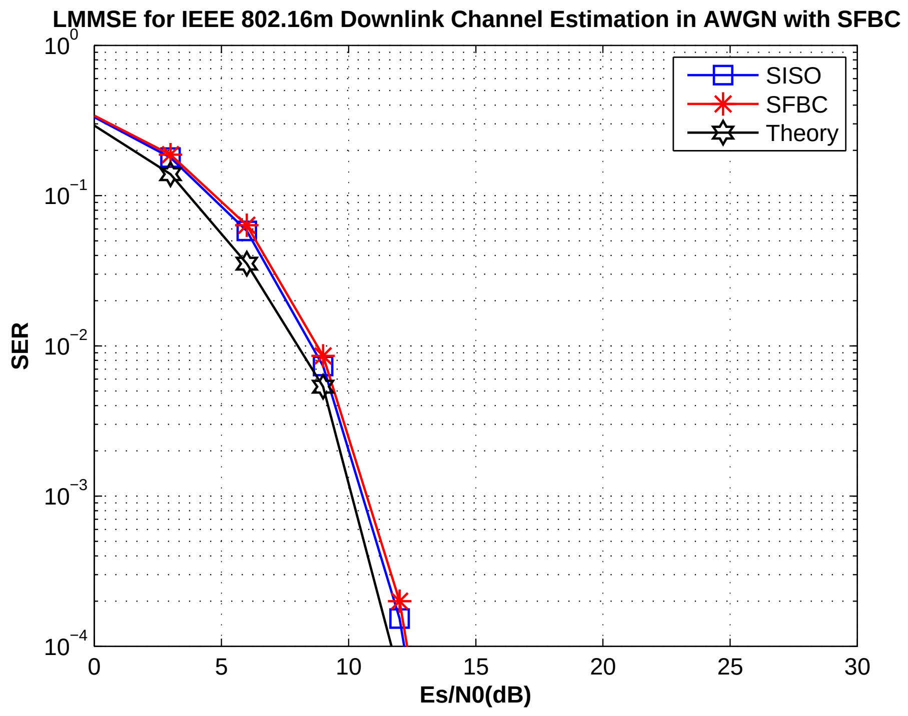


Figure 4.4: Channel estimation MSE and SER for QPSK with SFBC in AWGN channel for IEEE 802.16m downlink.

4.3 UL Simulation Results

4.3.1 Validation with AWGN Channel

We verify the correctness of the program code by simulation AWGN channel transmission. Both SISO and SFBC MIMO transmission are consider. We run 10^5 symbols in each simulation to obtain the numerical results. Figs. 4.5 and Figs. 4.6 shows the UL channel estimation performance in SISO transmission for AWGN channel. To check with our results, it is also agreement to with theory.

4.3.2 SISO Transmission Result

Figs. 4.7 and Figs. 4.8 show the MSE and SER for each frequency partition (FP). For downlink system, we use 48 PRUs for average to do the LMMSE channel estimation, but in uplink, we only consider the CRUs for each frequency partition. For FP0 which uses 6 PRUs for average to calculate LMMSE channel estimation. For FP1, FP2 and FP3 which use 8 PRUs for average. However the average numbers are different for each frequency partition, but we can not consider the performance of FP0 would worse than other frequency partitions, we can only confirm the standard deviation of mean delay for FP0 would higher than other frequency partitions. Table 4.5 show the standard deviation for each frequency at $v = 3$. From this table we can see the standard deviation for FP0 would higher than other frequency partitions. Please refer to appendix C for the numerical results.

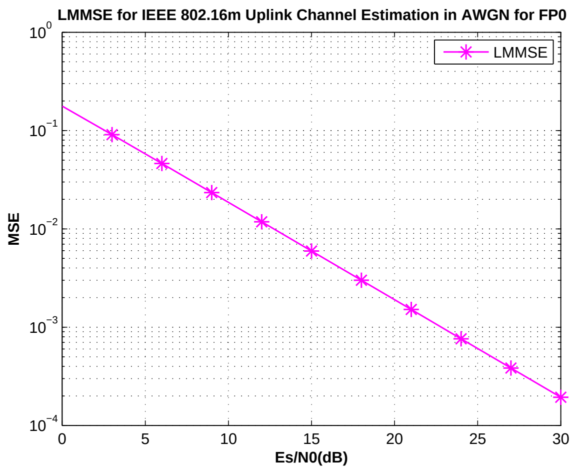
4.3.3 SFBC Transmission Result

In our simulation, we use two transmit antennas and one receive antenna. We estimate the two channel responses separately. So the MSE performance is not different by using SFBC as compared to SISO, but the SER performance is better. Appendix D contains the numerical

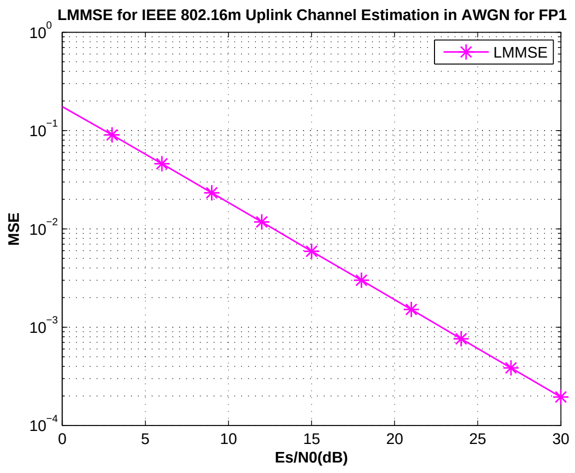
Table 4.5: Standard Deviation of Mean Delay for Each Frequency Partition at Different SNR in SUI-1 (v=3 km/h)

	FP0	FP1	FP2	FP3
SNR=0	0.013033777	0.012460021	0.01252457	0.011758789
SNR=2	0.012034557	0.01139748	0.011134512	0.010194757
SNR=4	0.011115335	0.010731387	0.010529548	0.00996801
SNR=6	0.008169569	0.007884321	0.008050248	0.007225824
SNR=8	0.006136158	0.006134259	0.005914159	0.004900918
SNR=10	0.005518237	0.004877316	0.005310798	0.004903142
SNR=12	0.005462989	0.003978956	0.004087563	0.004519001
SNR=14	0.003900397	0.003194372	0.00317707	0.00294827
SNR=16	0.002587206	0.00170759	0.00171771	0.00158806
SNR=18	0.001402389	0.001267293	0.001057377	0.001222587
SNR=20	0.001271882	0.00123742	0.000991301	0.001067567

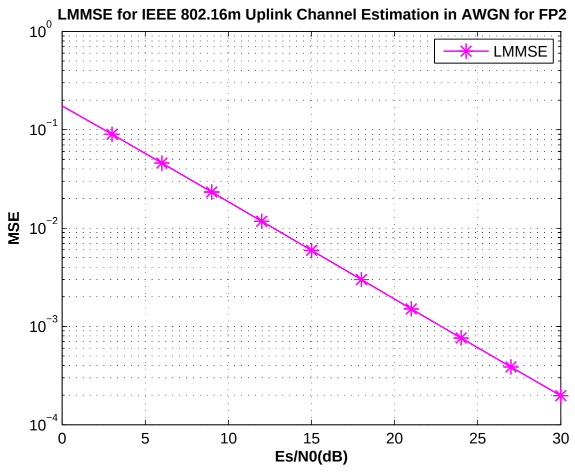
results.



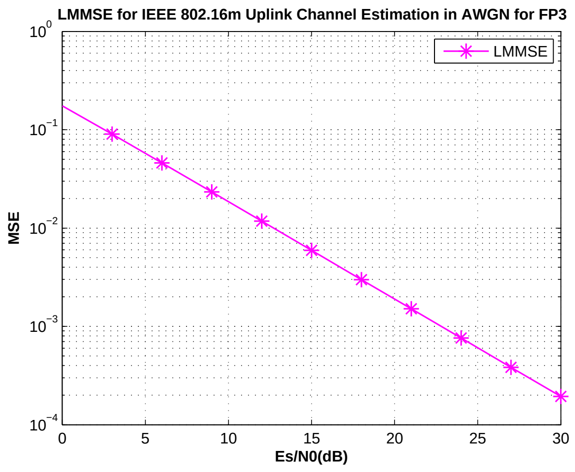
(a)



(b)

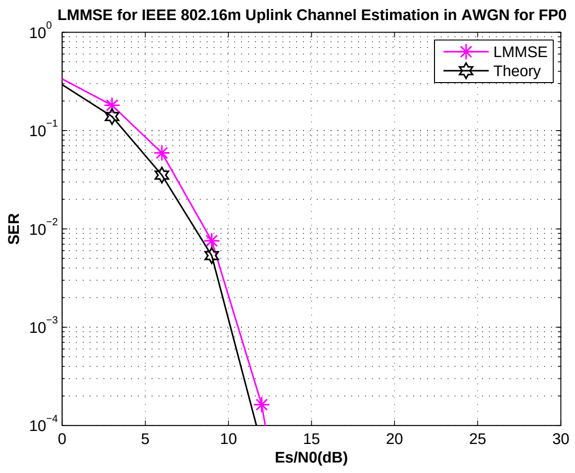


(c)

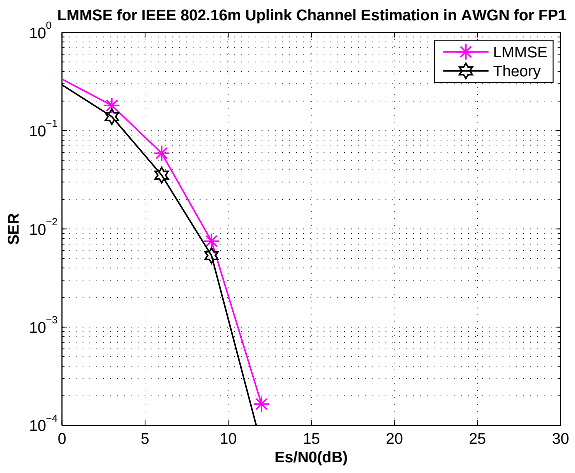


(d)

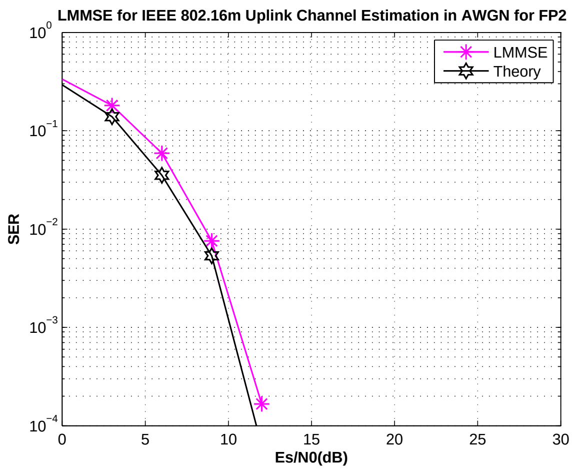
Figure 4.5: Channel estimation MSE use QPSK modulation for different frequency partitions in AWGN channel for IEEE 802.16m uplink.



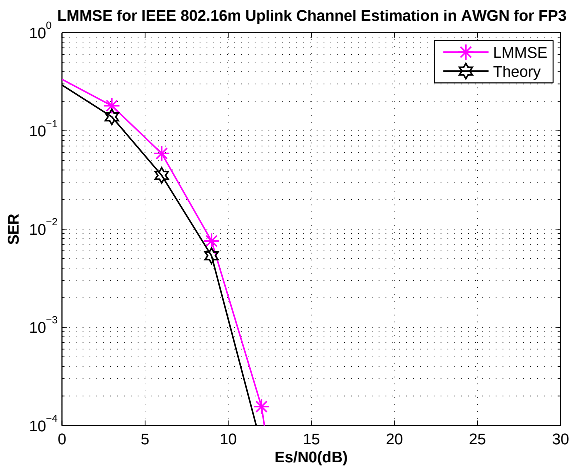
(a)



(b)

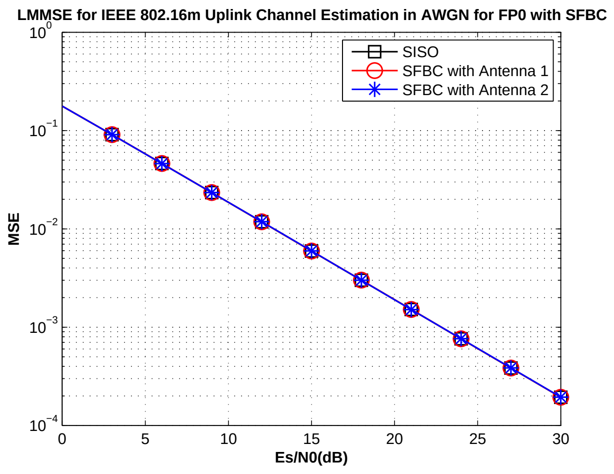


(c)

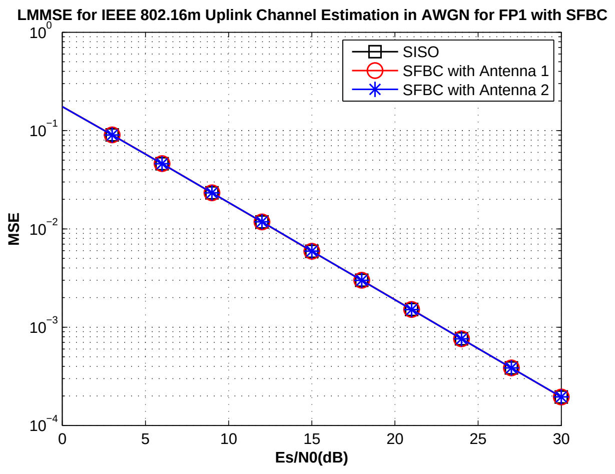


(d)

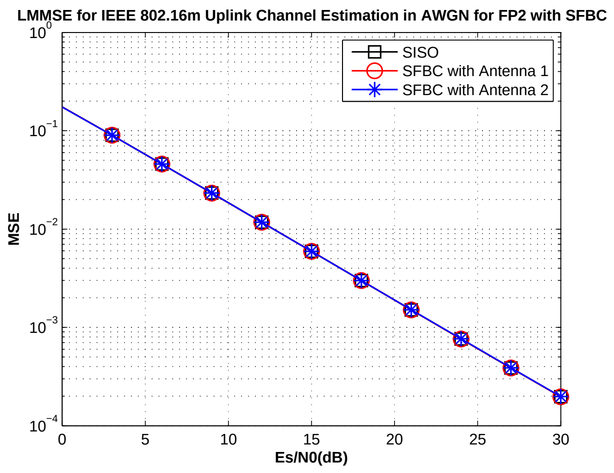
Figure 4.6: Channel estimation SER use QPSK modulation for different frequency partitions in AWGN channel for IEEE 802.16m uplink.



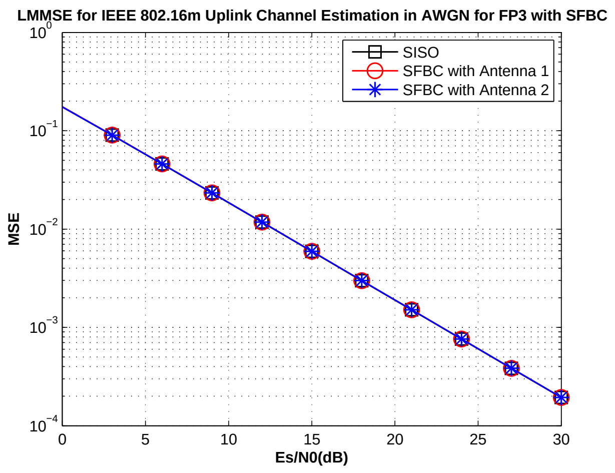
(a)



(b)

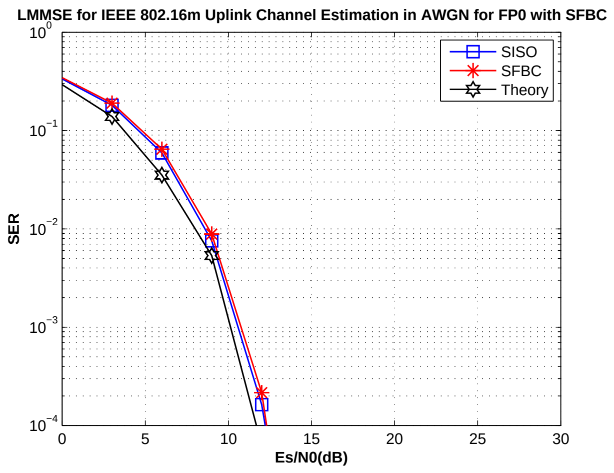


(c)

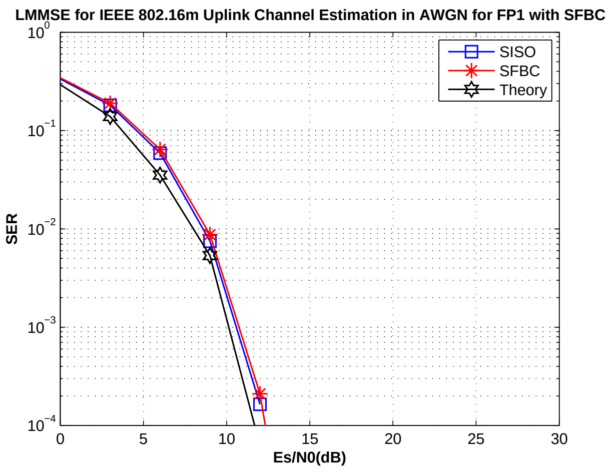


(d)

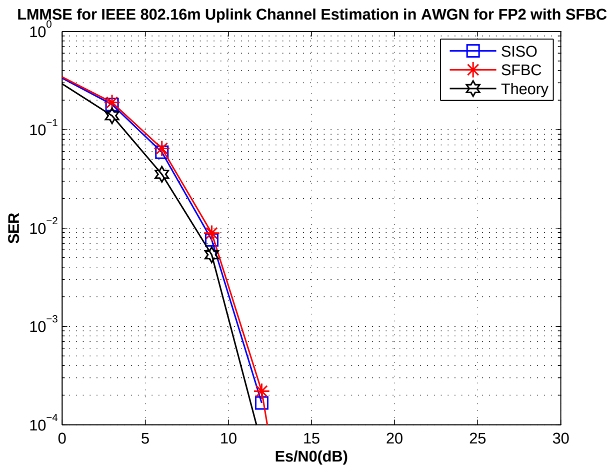
Figure 4.7: Channel estimation MSE use QPSK modulation with SFBC for different frequency partitions in AWGN channel for IEEE 802.16m uplink.



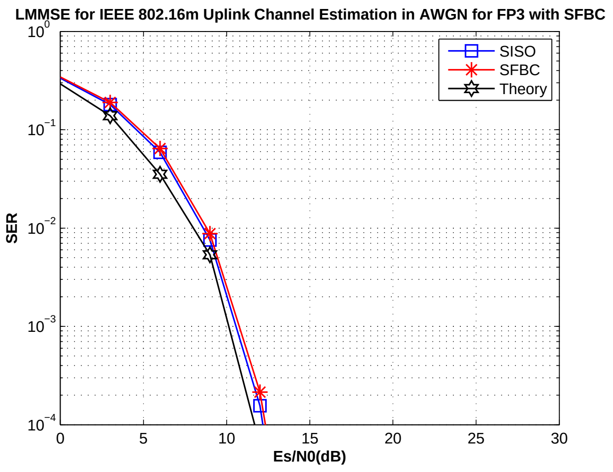
(a)



(b)



(c)



(d)

Figure 4.8: Channel estimation SER use QPSK modulation with SFBC for different frequency partitions in AWGN channel for IEEE 802.16m uplink.

Chapter 5

DSP Hardware and Associated Software Development Environment

DSP implementation of the LMMSE channel estimation method for IEEE 802.16m is the major objective of this thesis research. For this we use Code Composer Studio (CCS) and adopt DSP chip which type is TMS320C6416 for simulation. TMS320C6416 is made by Texas Instruments (TI) and we would introduce it and it's development environment in this chapter.

5.1 The TMS320C6416 DSP [15]

5.1.1 TMS320C64x Features

The TMS320C64x DSPs are the highest-performance fixed-point DSP generation of the TMS320C6000 DSP devices, with a performance of up to 1 billion instructions per second (MIPS) and an efficient C compiler. The TMS320C64x device is based on the second-generation high-performance, very-long-instruction-word (VLIW) architecture developed by Texas Instruments (TI). The C6416 device has two high-performance embedded coprocessors, Viterbi Decoder Coprocessor (VCP) and Turbo Decoder Coprocessor (TCP) that significantly speed up channel-decoding operations on-chip. But our implementation does not make use of VCP and TCP.

The C64x core CPU consists of 64 general-purpose 32-bits registers and 8 function units, where the 8 functional units contain 2 multipliers and 6 arithmetic units. C6000 features:

- Executes up to eight instructions per cycle and allows designers to develop effective RISC-like code with fast development time.
- Instruction packing packs the code tightly reduce code size, program fetches, and power consumption.
- Conditional execution of all instructions, which reduces costly branching and increases parallelism for higher sustained performance.
- Efficient C compiler and assembly optimizer effecting fast development and improved parallelization.
- 8/16/32-bit data support.
- 40-bit arithmetic options add extra precision for applications requiring it.
- Supports saturation and normalization in arithmetic operations.
- Field manipulation and instruction extract, set, clear, and bit counting support common operations found in control and data manipulation applications.

The additional features of C64x include:

- 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.

- Supports for non-aligned 32-bit (word) and 64-bit (double word) memory accesses.
- Special communication-specific (Galoris field) instructions have been added to address common operations in error-correcting codes.
- Bit count and rotate hardware extends support for bit-level algorithms.

5.1.2 Central Processing Unit

A block diagram of the C6416 DSP is shown in Fig. 5.1. 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.
- 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.

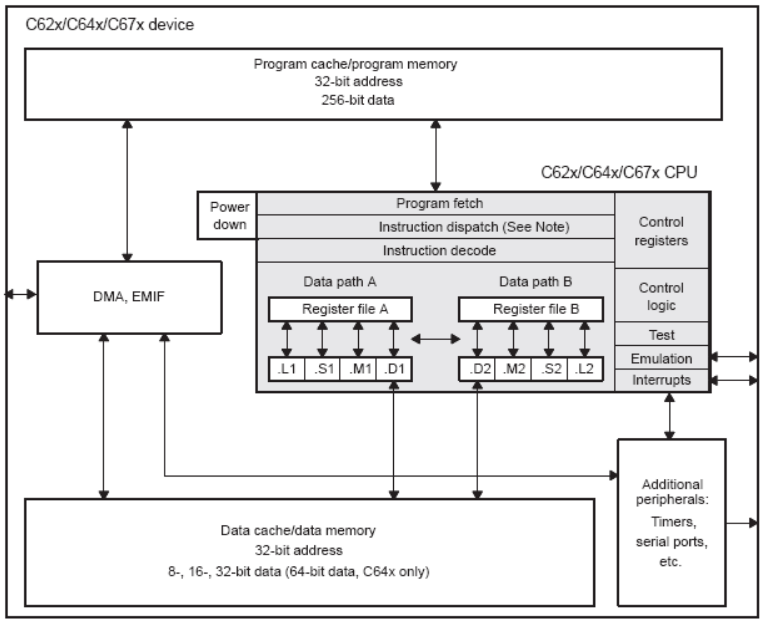


Figure 5.1: Block diagram of the TMS320C6416 DSP [15].

5.1.3 Functional Units

The eight functional units in the C6000 data paths can be divided into two groups of four; each of them in one data path is almost identical to the corresponding one in the other data path. The functional units are described in Table 5.1.

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.

The data line in the CPU supports 32-bit, 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 (shown in Fig. 5.2). 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.

5.1.4 Memory Architecture

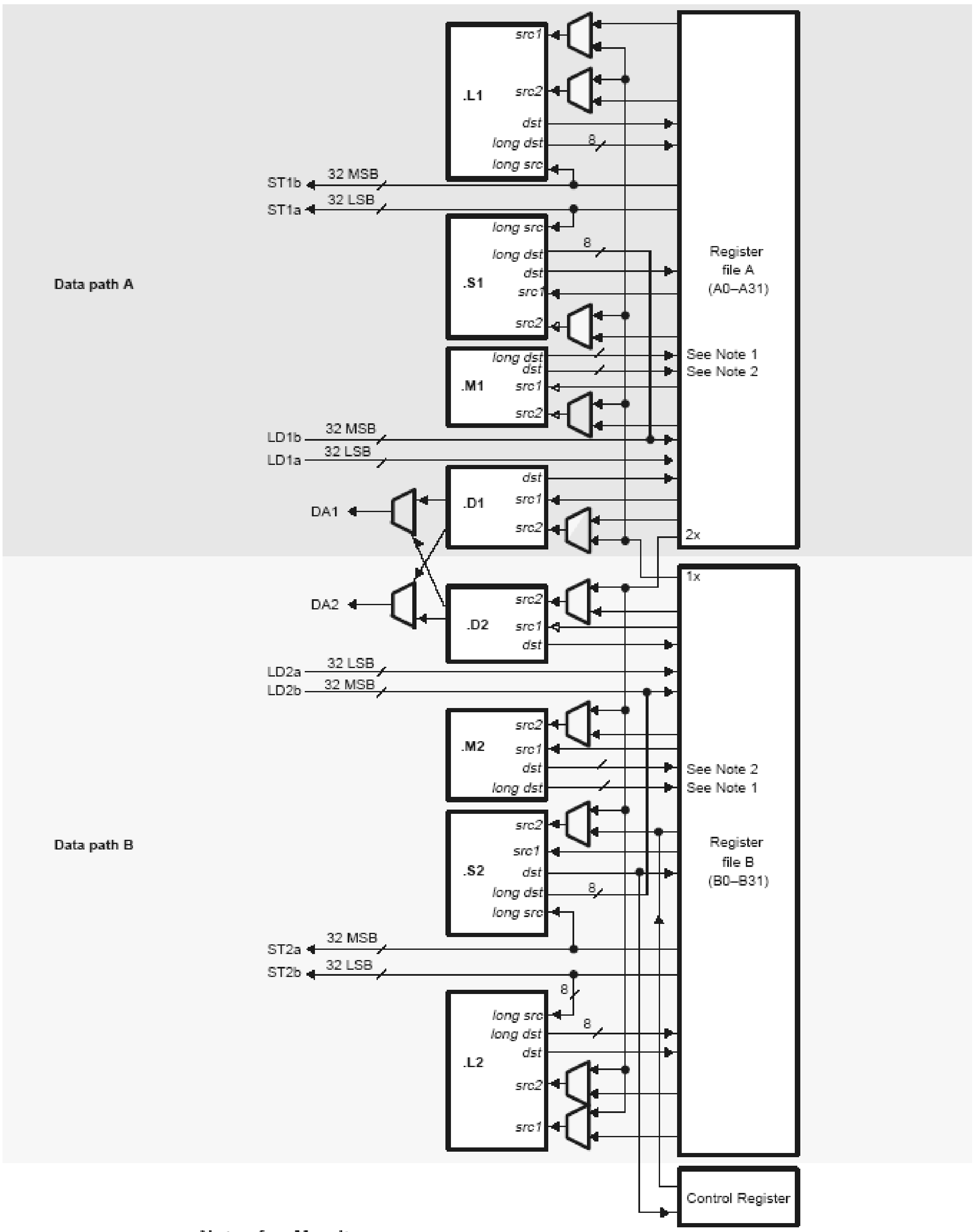
The C64x has a 32-bit, byte-addressable address space. Internal (on-chip) memory is organized into separate data and program spaces. When off-chip memory is used, these spaces are unified on most devices to a single memory space via the external memory interface (EMIF).

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 1 MB.
- Program cache.
- 32-bit external memory interface supports SDRAM, SBSRAM, SRAM, and other asynchronous memories.
- Two-level caches [16]. Level 1 cache is split into program (L1P) and data (L1D) cache. Each L1 cache is 16 KB. Level 2 memory is configurable and can be split into L2 SRAM (addressable on-chip memory) and L2 cache for caching external memory locations. The size of L2 is 1 MB. External memory can be many MB large. The access time depends on the memory technology used but is typically around 100 to 133 MHz. In our system, the external memory usable by the DSP is a 32 MB SDRAM. Table 5.2 shows the cache characteristics of L1 and L2 caches.

Table 5.1: Functional Units and Operations Performed [15]

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 $\rightarrow$ SP, INT $\rightarrow$ DP, INT $\rightarrow$ 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 $\rightarrow$ DP conversion 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



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

Figure 5.2: TMS320C64x CPU data paths [15].

Table 5.2: Cache Characteristics [16]

L1P Characteristics

Characteristic	C621x/C671x DSP	C64x DSP
Organization	Direct-mapped	Direct-mapped
Protocol	Read Allocate	Read Allocate
CPU access time	1 cycle	1 cycle
Capacity	4 Kbytes	16 Kbytes
Line size	64 bytes	32 bytes
Single miss stall	5 cycles	8 cycles
Miss pipelining	No	Yes

L1D Characteristics

Characteristic	C621x/C671x DSP	C64x DSP
Organization	2-way set-associative	2-way set-associative
Protocol	Read Allocate, Write-back	Read Allocate, Write-back
CPU access time	1 cycle	1 cycle
Capacity	4 Kbytes	16 Kbytes
Line size	32 bytes	64 bytes
Single read miss stall (L2 SRAM)	4 cycles	6 cycles
Single read miss stall (L2 Cache)	4 cycles	8 cycles
Miss pipelining	No	Yes
Multiple consecutive misses (L2 SRAM)	4 cycles	4 + 2 × M cycles
Multiple consecutive misses (L2 Cache)	4 cycles	6 + 2 × M cycles
Write miss	Passed through 4 × 32-bit write buffer. Only stalls when full.	Passed through 4 × 64-bit write buffer. Only stalls when full.

L2 Cache Characteristics

Characteristic	C621x/C671x DSP	C64x DSP
Organization	1-, 2-, 3-, or 4-way set-associative (depending on selected cache capacity)	4-way set-associative
Protocol	Read and write allocate Write-back	Read and write allocate Write-back
Capacity	16/32/48/64 Kbytes	32/64/128/256 Kbytes
Line size	128 bytes	128 bytes
Replacement strategy	Least recently used	Least recently used

5.2 Code Development Tools [17], [18]

We now introduce the software development environment used in our work. TI supports a useful GUI development tool set to DSP users for developing and debugging their codes: the 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 to suit different performance needs.
- 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 [19]. The routines included in the DSP library are organized as follows:

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

However TI offer these routines for optimize, but we did no use anyone in our C code.

5.2.1 Code Development Flow

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. 5.3. The tutorial section of the Programmer's Guide [20] focuses on phases 1 and 2, and the Guide also instructs the programmer about the tuning stage of phase 3. It is important to give the compiler enough information to fully maximize its potential. An benefical feature of this compiler is that it 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. 5.3.

- 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 [20] to improve the C code. Use the C6000 profiling tools to check its performance. If the code is still not as efficient as we would like it to be, proceed to phase 3.
- Extract the time-critical areas from the C code and rewrite the code in linear assembly. We can use the assembly optimizer to optimize the code.

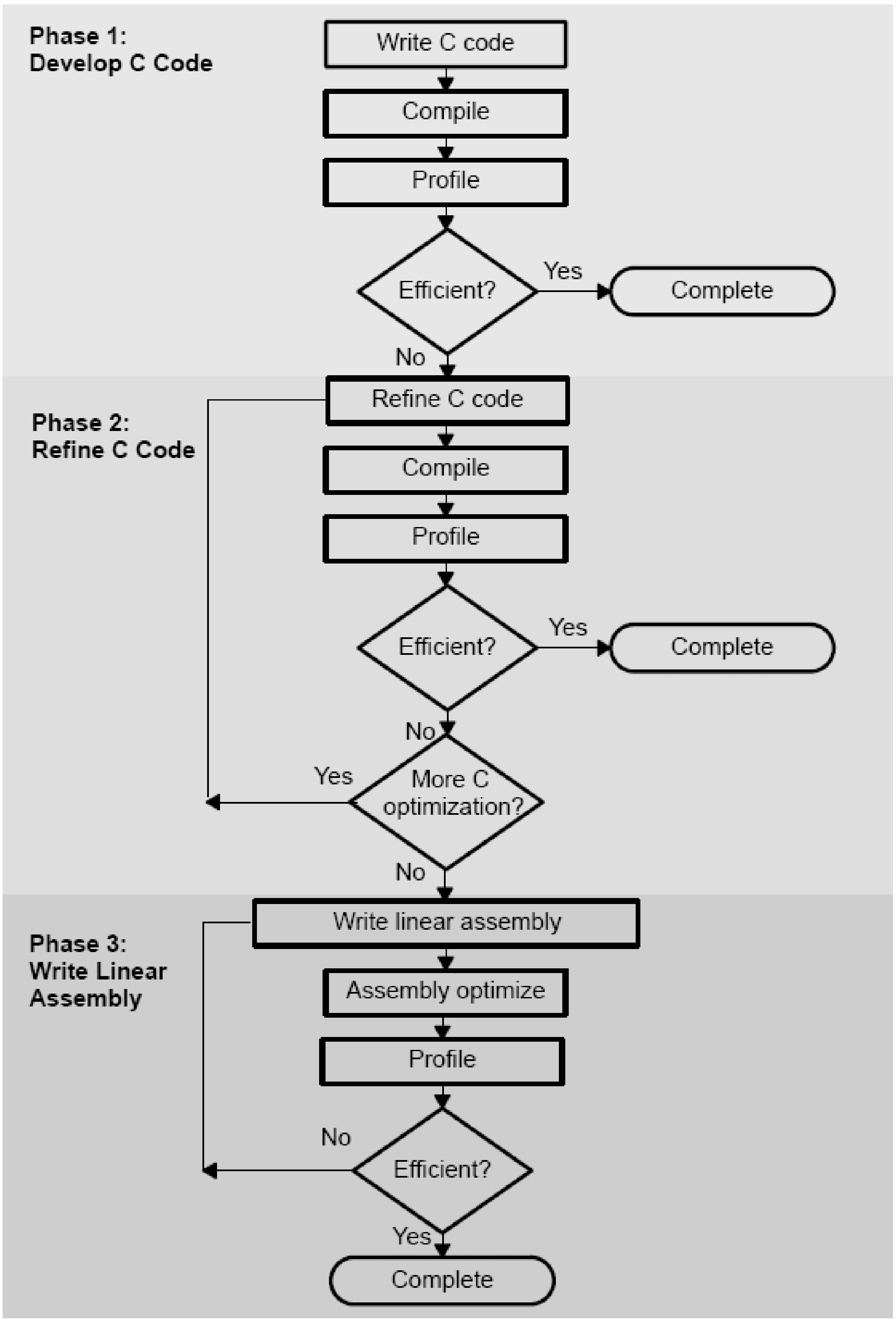


Figure 5.3: Code development flow for TI C6000 DSP [20].

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.

5.2.2 Compiler Optimization Options

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. Our code do not use up the L2 memory, and 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 -o*n* 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.

5.2.3 Using Intrinsics

The C6000 compiler provides intrinsics, which are special functions that map directly to C64x instructions, to optimize the C code performance. All instructions that are not easily expressed in C code are supported as intrinsics. Intrinsics are specified with a leading underscore (_) and are accessed by calling them as we call a function. A table of TMS320C6000 C/C++ compiler intrinsics can be found in [20]. In this thesis, we do not use any intrinsic in our program, maybe which can develop in the future.

Chapter 6

Fixed-Point Implementation and Optimization Methods

In this chapter, we discuss some technique issue concerning our fixed-point implementation and some optimization methods to reduce the run time. Before the above, we first introduce the basic concepts of fixed-point and floating-point arithmetic. What is their difference? What are the advantages and disadvantage of fixed-point calculation? After these, we propose some techniques regarding C code implementation and optimization.

6.1 Fixed-Point Concepts

6.1.1 Fixed-Point and Floating-Point

IEEE 754 is a standard for floating-point arithmetic, which describes the formats for floating-point numbers. There are two of them, namely, single precision and double precision. The formats are shown in Figure 6.1. The floating-point value is given by

$$floating-point\ value = (-1)^s \times (1 + fraction) \times 2^{Exp\ -\ bias}. \tag{6.1}$$

From this equation, we see that the translation for floating-point numbers is not very

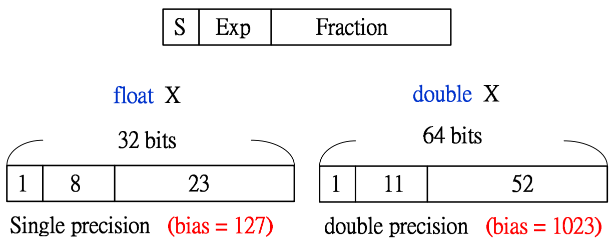


Figure 6.1: Formats for floating-point numbers.

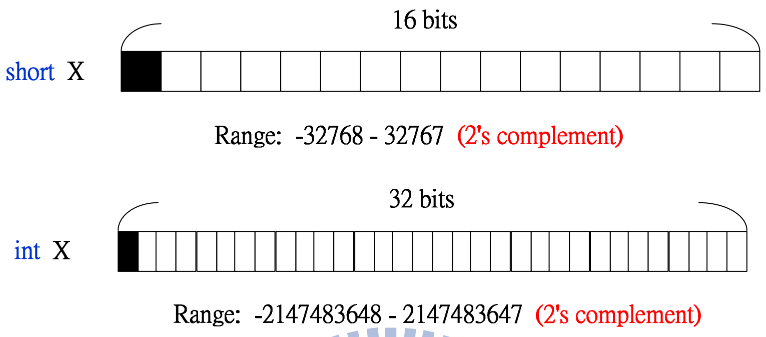

16 bits
short X
Range: -32768 - 32767 (2's complement)
32 bits
int X
Range: -2147483648 - 2147483647 (2's complement)

Figure 6.2: Formats for fixed-point numbers.

directly, which makes the associated arithmetic some what complex. But with this format, we can define a wide range of values.

The fixed-point formats, are shown in Figure 6.2, where the black fields indicate the sign bit. We can see that they are simpler and more direct than the floating point ones, and so hardware implementation can be cheaper. But the worst defect is dynamic range. The range of fixed-point values are usually more limited ones. Therefore, if an application requires a large dynamic range of numerical values, one should use floating-point arithmetic. Otherwise, one should use fixed-point arithmetic. In our implementation, we choose to use fixed-point implementation.

6.1.2 Fixed-Point Translation and Rounding

In this section, we discuss how to translate the floating-point values to fixed-point. There are three basic types of fixed-point length on the TI DSP chip. *short*, *integer* and *long*. *short* type and the *integer* types are 16 bits and 32 bits (both including signed bit) in length on TMS320C6416 DSP. The *long* type has 40 bits, but it requires more computational time and memory space, so we do not use this type in our implementation. Unless necessary, we use the *short* type, which can decrease the loading for the DSP chip and enhance the speed of the implementation.

To avoid excessive loss of precision (including overflow), we may use suitable scaling and rounding when representing a value in fixed point. For example, to represent the number 0.33 to a precision of $1/2 \times 2-10$, we may compute $0.33 \times 2^{10} + 0.5 = 338.42$ and truncate at the decimal point to obtain.

$$Fixed(\ 0.33 \times 2^{10} + 0.5) = 338. \tag{6.2}$$

Note that the addition of 0.5 before truncation is for rounding. If we do not add this 0.5, the average error can be bigger. In the above example, the result would have been 337 if we did not add the 0.5. In C, we can use right shift of bits to accomplish division by integer power of 2. For example, to calculate 100/16, we can use the instruction $100 \ >> \ 4$. To effect rounding at the last bit position, we can use $100 + 2^3 \ >> \ 4$.

6.1.3 Q and S Expressions

There are two common ways to represent fixed-point formats. One is Q expression the other is S expression. Table 6.1 shows the decimal ranges of these expressions. In this thesis, we use the Q expression. For convenience, we can use the *macro* instruction in C code to define some expressions, which can help us know the scaling factors of various quantities without a

Table 6.1: Q and S Expressiona [21]

Q	S	Decimal Range	Q	S	Decimal Range
Q15	S0.15	-1≦X≦0.9999695	Q7	S8.7	-256≦X≦255.9921875
Q14	S1.14	-2≦X≦1.9999390	Q6	S9.6	-512≦X≦511.9804375
Q13	S2.13	-4≦X≦3.9998779	Q5	S10.5	-1024≦X≦1023.96875
Q12	S3.12	-8≦X≦7.9997559	Q4	S11.4	-2048≦X≦2047.9375
Q11	S4.11	-16≦X≦15.9995117	Q3	S12.3	-4096≦X≦4095.875
Q10	S5.10	-32≦X≦31.9990234	Q2	S13.2	-8192≦X≦8191.75
Q9	S6.9	-64≦X≦63.9980469	Q1	S14.1	-16384≦X≦16383.5
Q8	S7.8	-128≦X≦127.9960938	Q0	S15.0	-32768≦X≦32767

lot of notes. At the last of this subsection, we give an example to illustrate the fixed-point arithmetic. Figure 6.3 shows the example. In this example, there are two different scaling factors in the original representation of X and Y, when we add them together, the one with a lower scaling factor needs to be down-scaled. This avoids overflow of the final result, but also sacrifice some precision.

6.2 Nonlinear Function Implementation

In our implementation, we will need to carry out the computation of some function that are implemented the math library of C as subroutines. These subroutines may take many CPU cycles to execute. We can use some technique to avoid calling these subroutines and save cycles. We discuss the techniques below. Specially, the functions that we are interested in are sqrt(), sin(), cos(), and arctan().

Ex. $Z = X + Y = 0.5 + 3.1 = 3.6$

$$X = 0.5 = 0100\ 0000\ 0000\ 0000 \xrightarrow{Q15} X_q = (short)\,X \times 2^{15} = 0.5 \times 32768 = 16384$$

$$Y = 3.1 = 0110\ 0011\ 0011\ 0011 \xrightarrow{Q13} Y_q = (short)\,X \times 2^{13} = 3.1 \times 8192 = 25395$$

$$temp = 16384 >> 2 = 4096$$

$$Z_q = (\text{int})(X_q + temp) = 4096 + 25395 = 29491$$

$$Z = Z_q \times 2^{-13} = 3.59997559 \approx 3.6 \ (\text{assume } Q_z = 13)$$

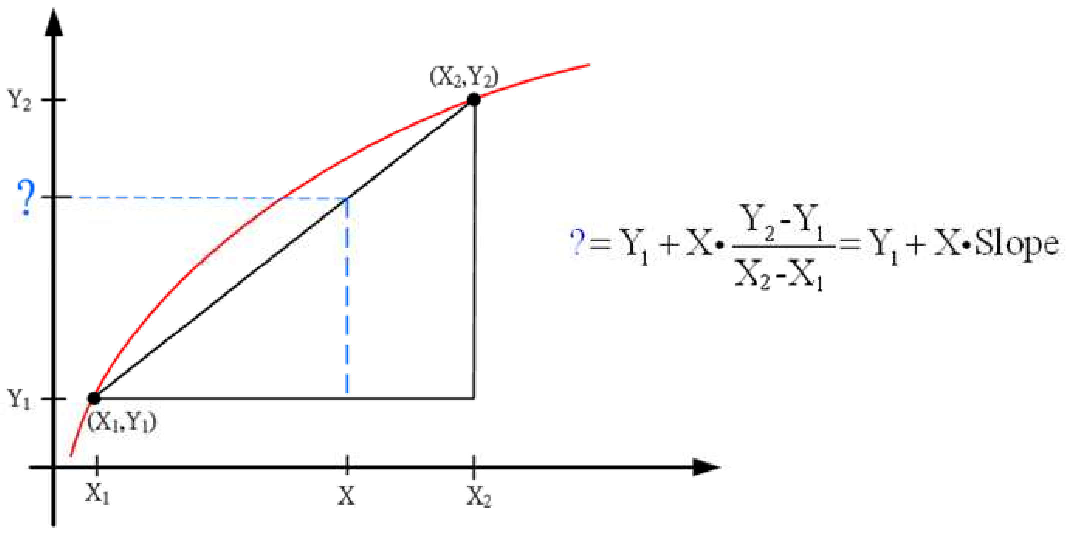
Figure 6.3: Example of fixed-point arithmetic.

6.2.1 Table Look-Up Method

A simplest method to implement a mathematical function is to use a look-up table. First, we build a table which contain the desired number of output values for the function. Then, we only need to enter the proper input index to get the answer from the table. But table size determines the needed memory space. If we want to have more output precision, then a larger table needs to be limit. Therefore, we have to know what range of input and output values that we need. When know there ranges, we can prevent wasting of unnecessary table space. Table look-up offer a simple method to replace calling library, but its precision depends on the table size. If the memory size is very limited, then it may not have enough space to store a table with the desired precision.

6.2.2 Mixed Method [21]

Mixed method is a trade off between table look-up and computation, which is based on table look-up but calculates the output value further by linear interpolation as shown in Figure 6.4. First, we have to build two tables of reduced size compared to the full-precision table, one is to record the function outputs and the other to record the slopes between pairs of input



Y_2
?
Y_1
(X_2,Y_2)
(X_1,Y_1)
X_1
X
X_2
$? = Y_1 + X \cdot \frac{Y_2 - Y_1}{X_2 - X_1} = Y_1 + X \cdot \text{Slope}$

Figure 6.4: Mixed method.

values. See Figure 6.4. If we want to know the corresponding output of X, we can get the answer by calculating the distance between X and X_1 and the slope between X_1 and X_2. Adopting this technique, we can use a little computation to exchange for more precision.

6.2.3 Implementation of Various Function

6.2.3.1 Sqrt()

We adopt the mixed method on square root function implementation. Because we know the input range is 0 to 4 by earlier calculation, so we build two tables. One has 401 entries for the corresponding output values at an input interval of 0.01 precision. Another one is the slope value table which has 400 entries. Figure 6.5 shows the comparison between two methods. From this figure we can realize if we only use table look-up, then we can lose much precision. However, while the mixed method is more precise than table look-up, the output values at small input values are less accurate. So we build another two tables for the input range between 0 to 0.01, which have 101 entries (the output table) and 100 entries (slope table) each.

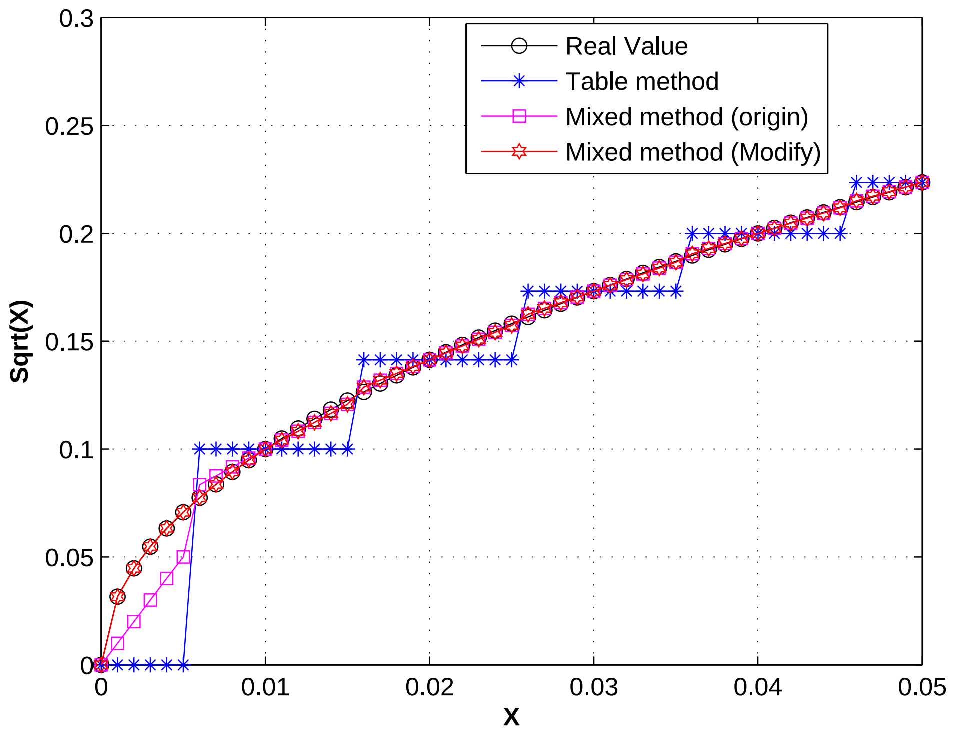


Figure 6.5: Mixed method and table look-up comparison for sqrt().

6.2.3.2 Sin() and Cos()

For sin() and cos(), we only build tables for sin and use the properties of trigonometric function to get cos() values. Because sin() is a symmetric function, we only create tables for input angles from 0 to $\pi/2$. We let the output table and the slope table have 158 and 157 entries each, at an interval of 0.01 precision. Figure 6.6 shows a part of the sin() curve.

6.2.3.3 Arctan()

In theory, the input range of arctangent starts from negative infinity and goes to positive infinity. It is impossible to create a infinite range table. So we have to use some tips to solve

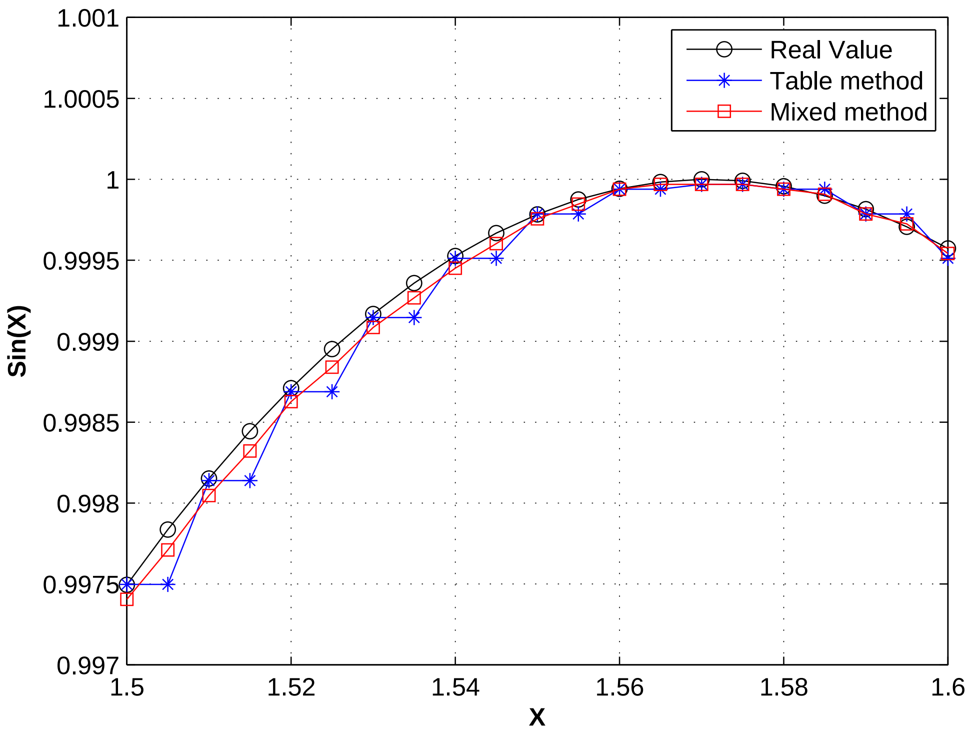


Figure 6.6: Mixed method and table look-up comparison for sin().

this problem. First, we note that the actual function as

$$\arctan(y/x) = \pi/2 - \arctan(x/y). \tag{6.3}$$

Because of the symmetric property , we only create output table values from 0 to 1, another one which can be achieved by (6.3). We let the output table and the slope table have 201 and 200 entries each, at an interval of 0.005 precision. Figure 6.7 shows a part of arctangent curve, which is very close to the real value.

6.2.3.4 MSE Performance of Two Methods

For verification of the performance of the two methods, we use 10000 random data for test. Table 6.2 shows the mean square error for nonlinear function implementation. From this

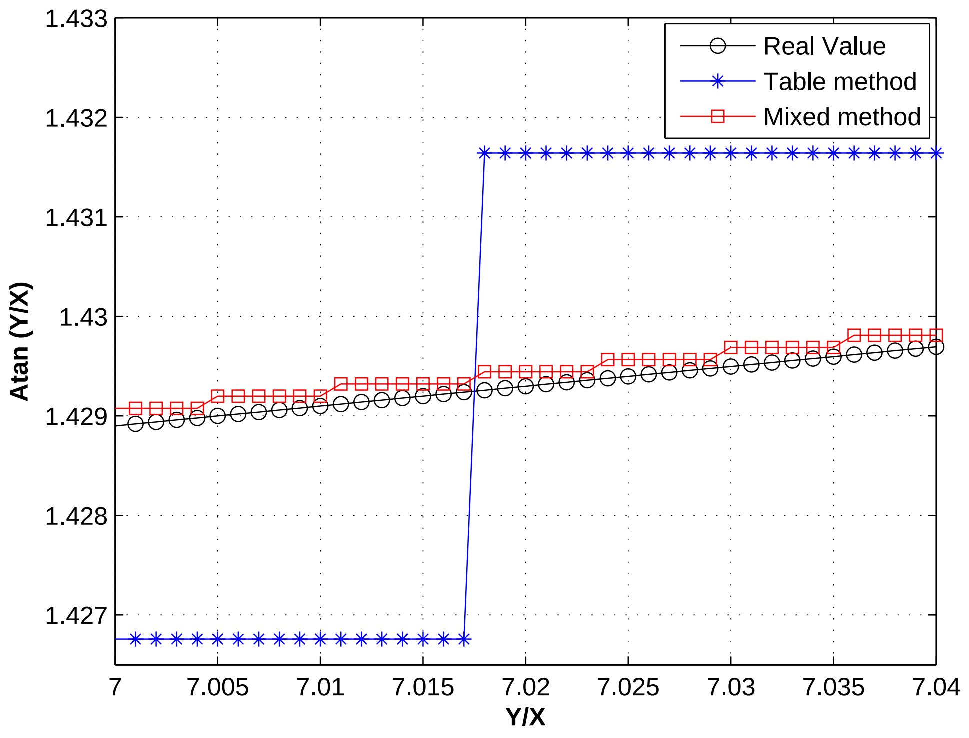


Figure 6.7: Mixed method and table look-up comparison for arctan.

table we can see that the mixed method has better performance than the pure table look-up method.

6.2.4 Summary

We summarize the way for computation of function values as follows.

1. Find the range of input values.

2. Declare the input as 16-bit fixed-point number if possible.

3. Scale the input property.

4. Use the mixed method to compute the function values.

Table 6.2: MSE for Nonlinear Function Implementation

Function name \ Method	Table	Mixed
sqrt()	3.131×10^{-6}	9.124×10^{-9}
sin()	3.509×10^{-6}	4.227×10^{-7}
arctan()	1.979×10^{-6}	7.451×10^{-9}

6.3 Q-Scaling Flow Chart

Figure 6.8 shows the flow chart of the channel estimation procedure together with the scaling factors of key variables. The channel estimation method has been introduced in chapter 4. The scaling factor have been designed based on theoretical analysis and they are determined so that a highest possible computational accuracy can be maintained without overflow.

6.4 DIV_fixed Function Analysis

For fixed-point DSP implementation, we should try to avoid divisions. This is because the DSP does not have a hardware divide but call a library subroutine to carry out the division, which cost quite a few CPU cycles. If one wants to calculate x/y for some y many times, it is advisable to calculate the reciprocal of the divisor and record it first. Them we can calculate x/y for any x. This can reduce a large amount of cycles. In this work, we use divisions in several place which can reference to Fig.6.8: 1 division to calculate Rmstemp, 1 division for arctangent function to calculate angle, 17 divisions to calculate the correlation function, 4 divisions for Levinson-Durbin parameter calculation and 6 divisions for each data response calculation, totalling to $1 + 1 + 17 + 4 + 6 \times 15$(data subcarrier number) $= 113$ divisions in theory for each OFDMA symbols.

But sometimes to calculate the reciprocal of the divisor would incur two problems. First

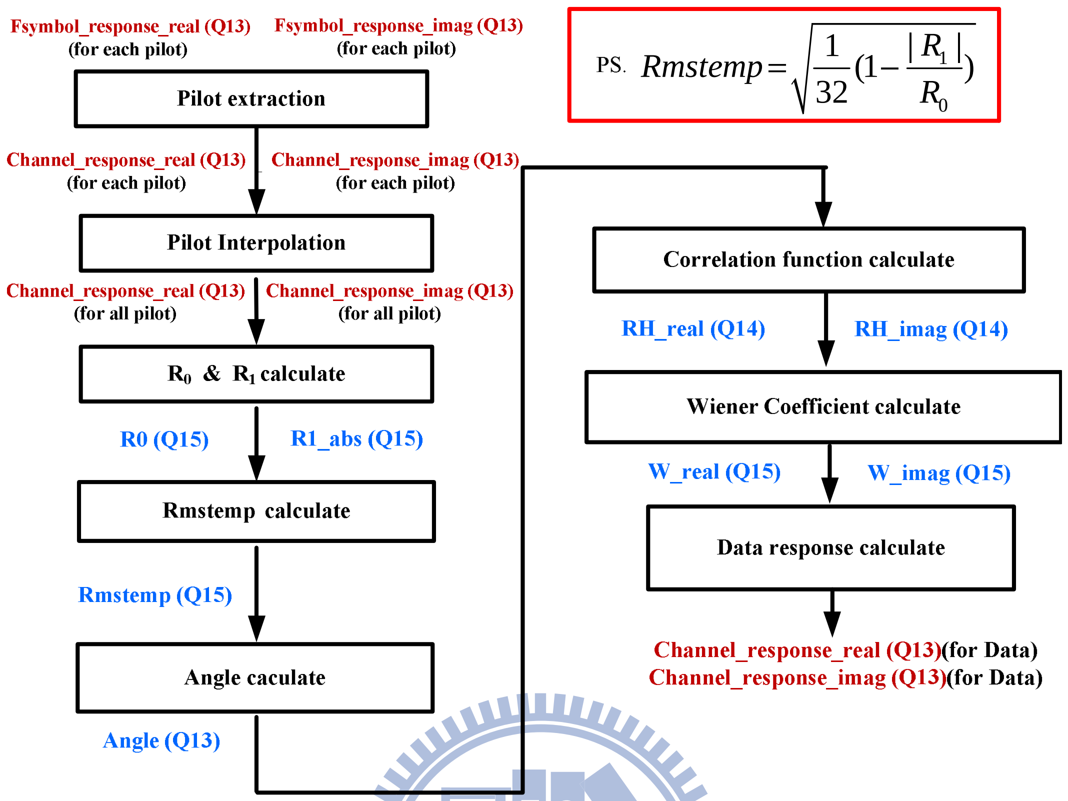

Fsymbol_response_real (Q13)
(for each pilot)
Fsymbol_response_imag (Q13)
(for each pilot)
PS. $Rmstemp = \sqrt{\frac{1}{32}(1 - \frac{|R_1|}{R_0})}$
Pilot extraction
Channel_response_real (Q13)
(for each pilot)
Channel_response_imag (Q13)
(for each pilot)
Pilot Interpolation
Channel_response_real (Q13)
(for all pilot)
Channel_response_imag (Q13)
(for all pilot)
R0 & R1 calculate
R0 (Q15)
R1_abs (Q15)
Rmstemp calculate
Rmstemp (Q15)
Angle caculate
Angle (Q13)
Correlation function calculate
RH_real (Q14)
RH_imag (Q14)
Wiener Coefficient calculate
W_real (Q15)
W_imag (Q15)
Data response calculate
Channel_response_real (Q13)(for Data)
Channel_response_imag (Q13)(for Data)

Figure 6.8: Q-scaling flow chart for LMMSE channel estimation.

is about overflow. For example, suppose we want to calculate $1/y$ and scale to Q13 and y uses Q14 scaling. Then

$$k = \frac{1}{y} \Rightarrow (Q13)(k) = \frac{(Q27)1}{(Q14)y} = \frac{2^{27}}{Y},$$
$$\text{range of } Y : 1 \text{ to } 2^{15} \text{ (assume } Y \text{ is positive)},$$
$$\text{range of } K : 2^{12} \text{ to } 2^{27}.$$

As for range of K, we have to use integer type for record K. If we want to reuse that, for example $Z/Y = Z \times K$ (however Z is short or integer type), we have to readjust the K scaling to avoid overflow. The second problem is about precision. For example, if we want to calculate $1/y$ and y is scaling to Q28, we can only get Q4 (assume y is positive) precision for result of $1/y$ at most. To avoid overflow and promote the precision of $1/y$, we design a

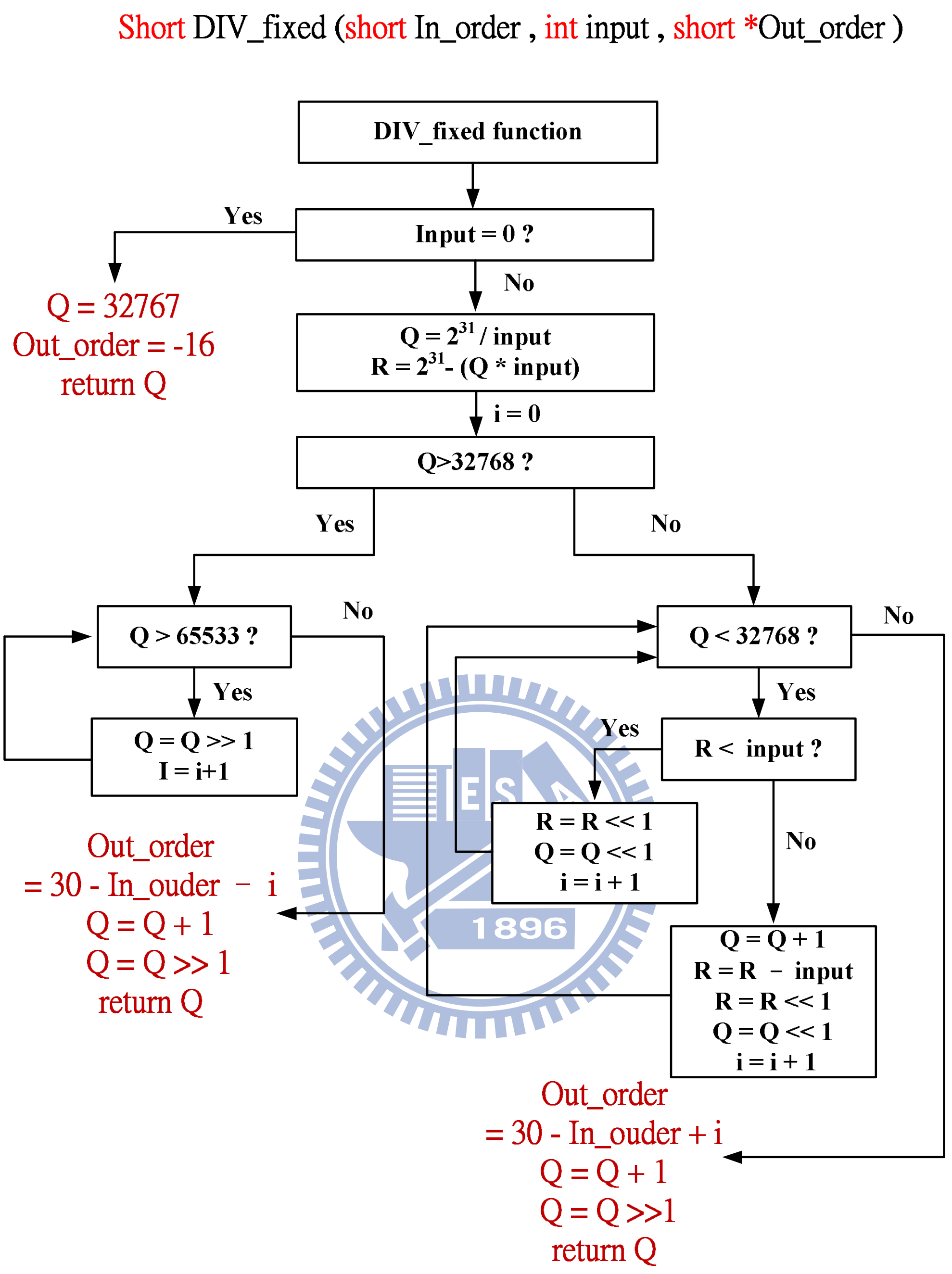


Figure 6.9: DIV_fixed structure

subfunction which named DIV_fixed. Figure 6.9 shows the structure of that. Left branch can solve overflow problem and right branch can promote the precision of $1/y$.

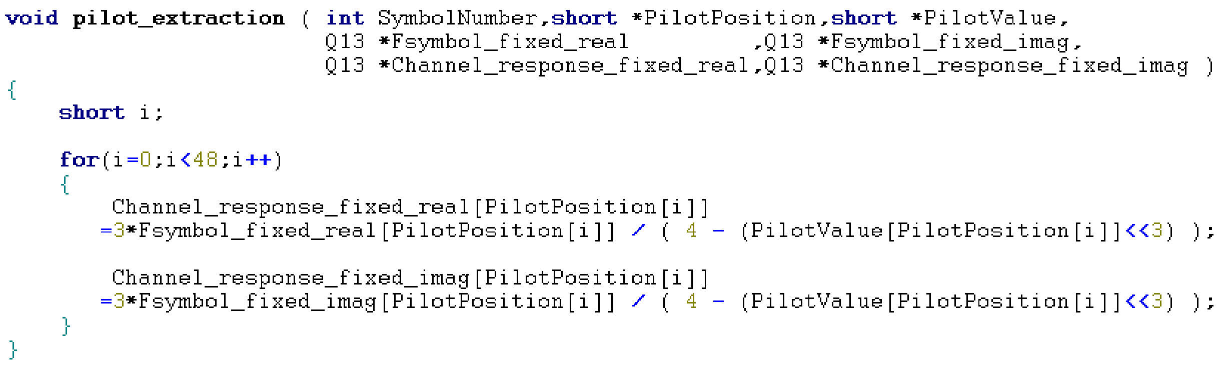
6.5 Implementation and Optimization

In this section we discuss the implementation and optimization of the channel estimation method whose flow chart has been given in Figure 6.8. Because DL and UL transmission have the same implementation and optimization methods, we only simulate with 28 symbols and run eleven SNR values from 0 to 30 dB in downlink SISO channel estimation.

6.5.1 Channel Estimation at pilot Subcarrier

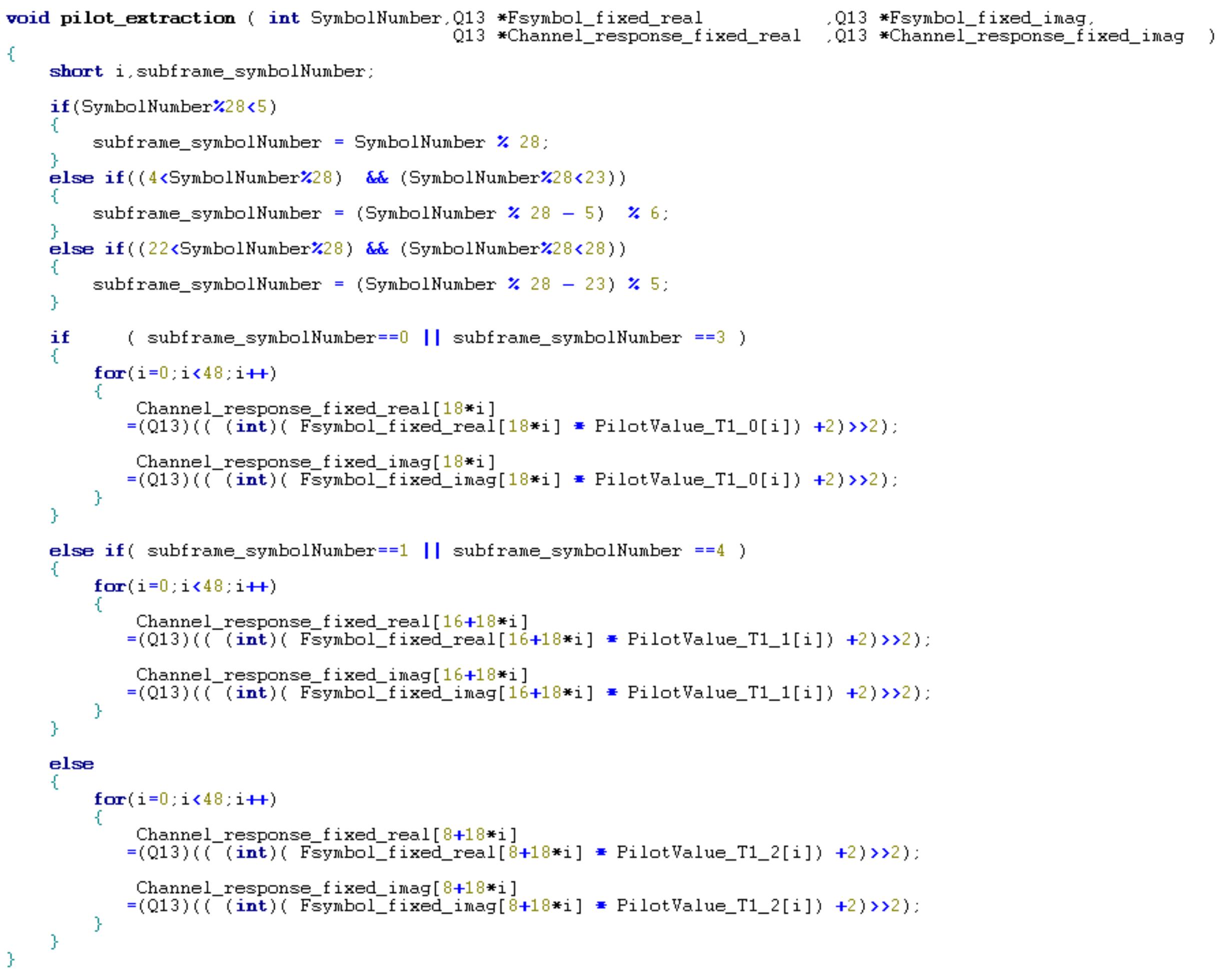
When we have received the signal, we first use the LS method to estimate the channel response at each pilots. Figure 6.10 shows the original version of pilot extraction function. In Figure 6.10 we observe that there are two parts that can be improved. One is about indexing. PilotPosition is an array which indicates the pilot positions of all the symbols, but using an array would result in some CPU cycles to obtain the desired values therein, so we try to avoid it. Because these pilot positions has regularity in 16m, we replace PilotPosition with the actual values, which is the first modification. The second modification is to replace division by multiplication. Figure 6.11 shows the final version of the same code section and Table 6.3 shows the clock cycles for this function, where Excl.Total cycle means total number of cycles for all executions excluding function calls. The CCS can provide two different cycle data for reference, one with function aspect, and the other with loop aspect. In the former one can see the cycles for the function in the code, and in the latter, the loops we have broken the code from one loop in the original version into three loops in the modified version, but the total cycles are now less than the original version.

To evaluate the efficiency, note that there are two multipliers on C6416 chips. If the two multipliers can work in parallel without stall, the required cycles are Symbols $\times$ SNR $\times$ PRU $\times$ multiplications $= 28 \times 11 \times 48 \times 1 = 14784$, so the efficiency is $14784/63509 = 23.3\%$. The performance is less than ideal, maybe which can improve in the future.



```c
void pilot_extraction ( int SymbolNumber,short *PilotPosition,short *PilotValue,
                        Q13 *Fsymbol_fixed_real         ,Q13 *Fsymbol_fixed_imag,
                        Q13 *Channel_response_fixed_real,Q13 *Channel_response_fixed_imag )
{
    short i;

    for(i=0;i<48;i++)
    {
        Channel_response_fixed_real[PilotPosition[i]]
       =3*Fsymbol_fixed_real[PilotPosition[i]] / ( 4 - (PilotValue[PilotPosition[i]]<<3) );

        Channel_response_fixed_imag[PilotPosition[i]]
       =3*Fsymbol_fixed_imag[PilotPosition[i]] / ( 4 - (PilotValue[PilotPosition[i]]<<3) );
    }
}
```

Figure 6.10: C code for pilot subcarrier channel estimation before modification.



```c
void pilot_extraction ( int SymbolNumber,Q13 *Fsymbol_fixed_real          ,Q13 *Fsymbol_fixed_imag,
                                         Q13 *Channel_response_fixed_real  ,Q13 *Channel_response_fixed_imag   )
{
    short i,subframe_symbolNumber;

    if(SymbolNumber%28<5)
    {
        subframe_symbolNumber = SymbolNumber % 28;
    }
    else if((4<SymbolNumber%28)  && (SymbolNumber%28<23))
    {
        subframe_symbolNumber = (SymbolNumber % 28 - 5)  % 6;
    }
    else if((22<SymbolNumber%28) && (SymbolNumber%28<28))
    {
        subframe_symbolNumber = (SymbolNumber % 28 - 23) % 5;
    }

    if     ( subframe_symbolNumber==0 || subframe_symbolNumber ==3 )
    {
        for(i=0;i<48;i++)
        {
         Channel_response_fixed_real[18*i]
        =(Q13)(( (int)( Fsymbol_fixed_real[18*i] * PilotValue_T1_0[i]) +2)>>2);

         Channel_response_fixed_imag[18*i]
        =(Q13)(( (int)( Fsymbol_fixed_imag[18*i] * PilotValue_T1_0[i]) +2)>>2);
        }
    }

    else if( subframe_symbolNumber==1 || subframe_symbolNumber ==4 )
    {
        for(i=0;i<48;i++)
        {
         Channel_response_fixed_real[16+18*i]
        =(Q13)(( (int)( Fsymbol_fixed_real[16+18*i] * PilotValue_T1_1[i]) +2)>>2);

         Channel_response_fixed_imag[16+18*i]
        =(Q13)(( (int)( Fsymbol_fixed_imag[16+18*i] * PilotValue_T1_1[i]) +2)>>2);
        }
    }

    else
    {
        for(i=0;i<48;i++)
        {
         Channel_response_fixed_real[8+18*i]
        =(Q13)(( (int)( Fsymbol_fixed_real[8+18*i] * PilotValue_T1_2[i]) +2)>>2);

         Channel_response_fixed_imag[8+18*i]
        =(Q13)(( (int)( Fsymbol_fixed_imag[8+18*i] * PilotValue_T1_2[i]) +2)>>2);
        }
    }
}
```

Figure 6.11: C code for pilot subcarrier channel estimation after modification.

Table 6.3: Pilot Subcarrier Channel Estimation Clock Cycles Comparison

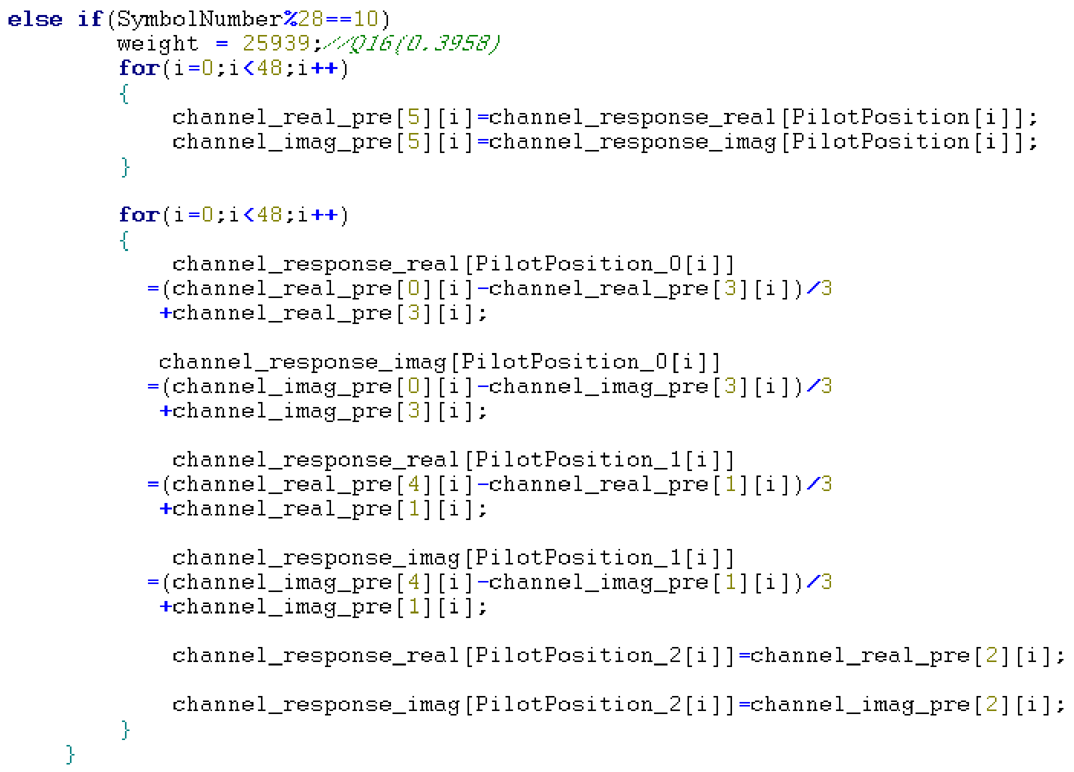
	Symbol Type	Excl. Total cycle	
Original	Function	579032	
	Loop	396311	
First modification	Function	465605	
	Loop (1)	91234	258246
	Loop (2)	90470	
	Loop (3)	76542	
Second modification	Function	102462	
	Loop (1)	22385	63509
	Loop (2)	22982	
	Loop (3)	18142	

Table 6.4: Pilot Interpolation Clock Cycles Comparison

	Symbol Type	Excl. Total cycle
Original	Loop	485433
Modified	Loop	113784

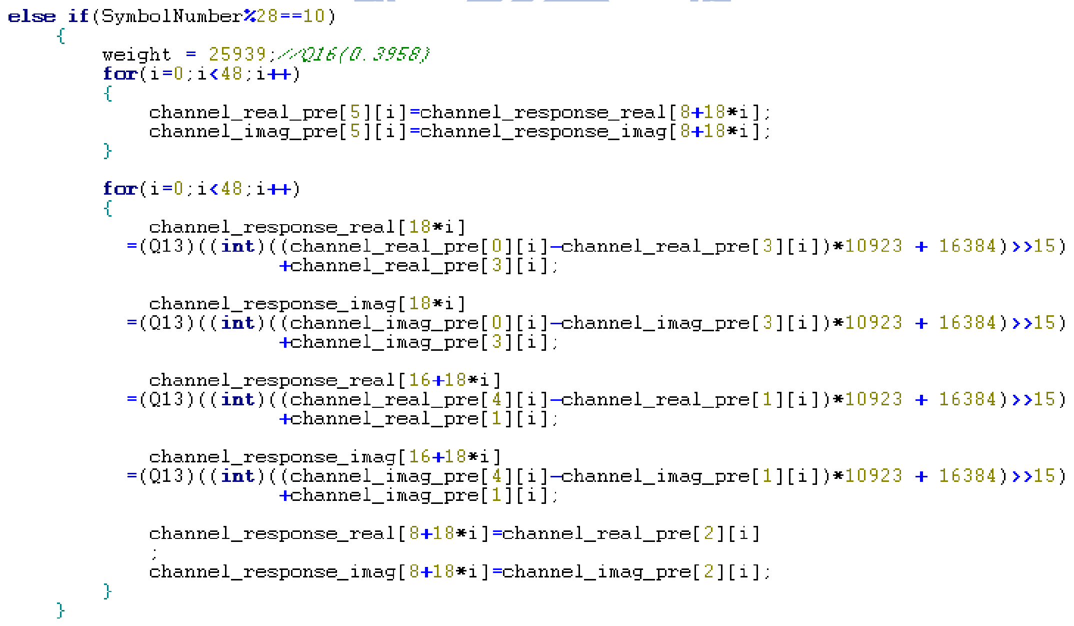
6.5.2 Time-Domain Interpolation of Pilot Channel Responses

After we get channel response for each pilot, we should do linear interpolation in time to get the channel responses as shown in Figure 3.2. To optimize this function, we can replace the constant divisions with multiplications. Figures 6.12 and 6.13 show a part of the original version and the modified version interpolation code, respectively. Table 6.4 shows the resulting cycle. To calculate efficiency, if the multipliers can work in parallel without stall, the require cycles are Symbols $\times$ SNR $\times$ PRU $\times$ multiplications $= 28 \times 11 \times 48 \times 4 = 59136$. Efficiency is $59136/113784 = 52\%$. The efficiency has exceeded fifty percent, the performance looks better.



```
else if(SymbolNumber%28==10)
         weight = 25939;//Q16(0.3958)
         for(i=0;i<48;i++)
         {
              channel_real_pre[5][i]=channel_response_real[PilotPosition[i]];
              channel_imag_pre[5][i]=channel_response_imag[PilotPosition[i]];
         }

         for(i=0;i<48;i++)
         {
              channel_response_real[PilotPosition_0[i]]
           =(channel_real_pre[0][i]-channel_real_pre[3][i])/3
            +channel_real_pre[3][i];

             channel_response_imag[PilotPosition_0[i]]
           =(channel_imag_pre[0][i]-channel_imag_pre[3][i])/3
            +channel_imag_pre[3][i];

              channel_response_real[PilotPosition_1[i]]
           =(channel_real_pre[4][i]-channel_real_pre[1][i])/3
            +channel_real_pre[1][i];

              channel_response_imag[PilotPosition_1[i]]
           =(channel_imag_pre[4][i]-channel_imag_pre[1][i])/3
            +channel_imag_pre[1][i];

              channel_response_real[PilotPosition_2[i]]=channel_real_pre[2][i];

              channel_response_imag[PilotPosition_2[i]]=channel_imag_pre[2][i];
         }
     }
```

Figure 6.12: C code for pilot channel response interpolation before modification.



```
else if(SymbolNumber%28==10)
     {
          weight = 25939;//Q16(0.3958)
          for(i=0;i<48;i++)
          {
               channel_real_pre[5][i]=channel_response_real[8+18*i];
               channel_imag_pre[5][i]=channel_response_imag[8+18*i];
          }

          for(i=0;i<48;i++)
          {
               channel_response_real[18*i]
            =(Q13)((int)((channel_real_pre[0][i]-channel_real_pre[3][i])*10923 + 16384)>>15)
                         +channel_real_pre[3][i];

               channel_response_imag[18*i]
            =(Q13)((int)((channel_imag_pre[0][i]-channel_imag_pre[3][i])*10923 + 16384)>>15)
                         +channel_imag_pre[3][i];

               channel_response_real[16+18*i]
            =(Q13)((int)((channel_real_pre[4][i]-channel_real_pre[1][i])*10923 + 16384)>>15)
                         +channel_real_pre[1][i];

               channel_response_imag[16+18*i]
            =(Q13)((int)((channel_imag_pre[4][i]-channel_imag_pre[1][i])*10923 + 16384)>>15)
                         +channel_imag_pre[1][i];

               channel_response_real[8+18*i]=channel_real_pre[2][i]
               ;
               channel_response_imag[8+18*i]=channel_imag_pre[2][i];
          }
     }
```

Figure 6.13: C code for pilot channel response interpolation after modification.

6.5.3 Calculation of R_0 and R_1

After we calculate channel response for all pilots, we do LMMSE channel estimation for the data positions. First we should calculate the autocorrelation values R_0 and R_1. Because R_0 and R_1 are used in subsequent computation, their precision is very important. So we let them be Q15 to have maximum fractional bits allowed in 16-bit fixed-point format.

6.5.4 Rmstemp and Angle Calculation

We use the R_0 and R_1 to calculate mean delay and RMS delay spread which are combination by Rmstemp and the angle of R_1. Rmstemp is given by

$$\text{Rmstemp} = \sqrt{\frac{1}{32} \times (1 - \frac{|R_1|}{R_0})}$$

which is a reduced parameter and which has positive proportional with RMS delay spread. The fixed-point implementation methods to calculate Rmstemp and angle has been discussed in sections 6.2.3.1 and 6.2.3.3, respectively.

6.5.5 Correlation Function Calculate

When we have the Rmstemp and the angle, we can use them to calculate the correlation function in the frequency domain. But in fixed-point implementation, because of numerical errors, some results may violate the physical meaning. The correlation function is given by

$$RH\_real[i] = R_0 \times \frac{\cos(i \times (\frac{\angle R_1}{8} + Rmstemp)) + i \times Rmstemp \times \sin(i \times (\frac{\angle R_1}{8} + Rmstemp))}{1 + (i \times Rmstemp)^2},$$
$$RH\_imag[i] = R_0 \times \frac{\sin(i \times (\frac{\angle R_1}{8} + Rmstemp)) - i \times Rmstemp \times \cos(i \times (\frac{\angle R_1}{8} + Rmstemp))}{1 + (i \times Rmstemp)^2},$$

and Figure 6.14 illustrates its shapes in time and frequency. From the left plot, we see that the amplitude of $RH[0] = R_0$ is its maximum value in the frequency domain. So we set

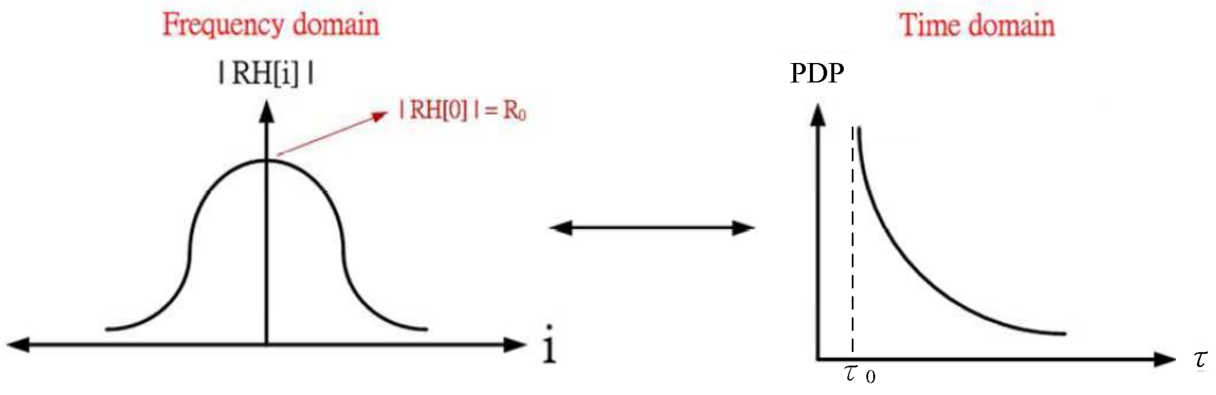


Figure 6.14: Correlation function shapes in the time domain and the frequency domain.

a limit as follows: if $|RH[i]|^2 > |RH[i-1]|^2$, then let $RH\_real[i] = RH\_real[i-1]$ and $RH\_imag[i] = RH\_imag[i-1]$, for $1 \leq i \leq 17$.

6.5.6 Wiener Coefficients Calculation

We use the correlation function to calculate the autocorrelation $\mathbf{R}_{pp}$ and the crosscorrelation $\mathbf{r}_{dp}$. As for the Wiener filter coefficients, we need to calculate the inverse matrix of $\mathbf{R}_{pp}$. We consider three methods: Gauss elimination, the direct formula method, and the Levinson-Durbin method. Recall that the Wiener filter coefficients as given by $\mathbf{w}_d = (\mathbf{R}_{pp} + \sigma_n^2\mathbf{I})^{-1}\mathbf{r}_{dp}$, where by the 16m signal structure, $(\mathbf{R}_{pp} + \sigma_n^2\mathbf{I})$ is a 3×3 matrix given by

$$\mathbf{R} \triangleq \mathbf{R}_{pp} + \sigma_n^2\mathbf{I} = \begin{bmatrix} RH[0]+\sigma_n^2 & RH[8] & RH[16] \\ RH[8]^* & RH[0]+\sigma_n^2 & RH[8] \\ RH[16]^* & RH[8]^* & RH[0]+\sigma_n^2 \end{bmatrix} = \begin{bmatrix} R_0 & R_8 & R_{16} \\ R_8^* & R_0 & R_8 \\ R_{16}^* & R_8^* & R_0 \end{bmatrix}. \quad (6.4)$$

6.5.6.1 Gauss Elimination

Figure 6.15 shows the process of Gauss elimination. Because **R** is a covariance matrix, it inverse has a Hermitian structure. For this reason, we only need to do the elimination six times to get the whole inverse matrix. But in this process, we can not control the numerical ranges of the values, so it is hard to give these values a proper scaling. After we complete

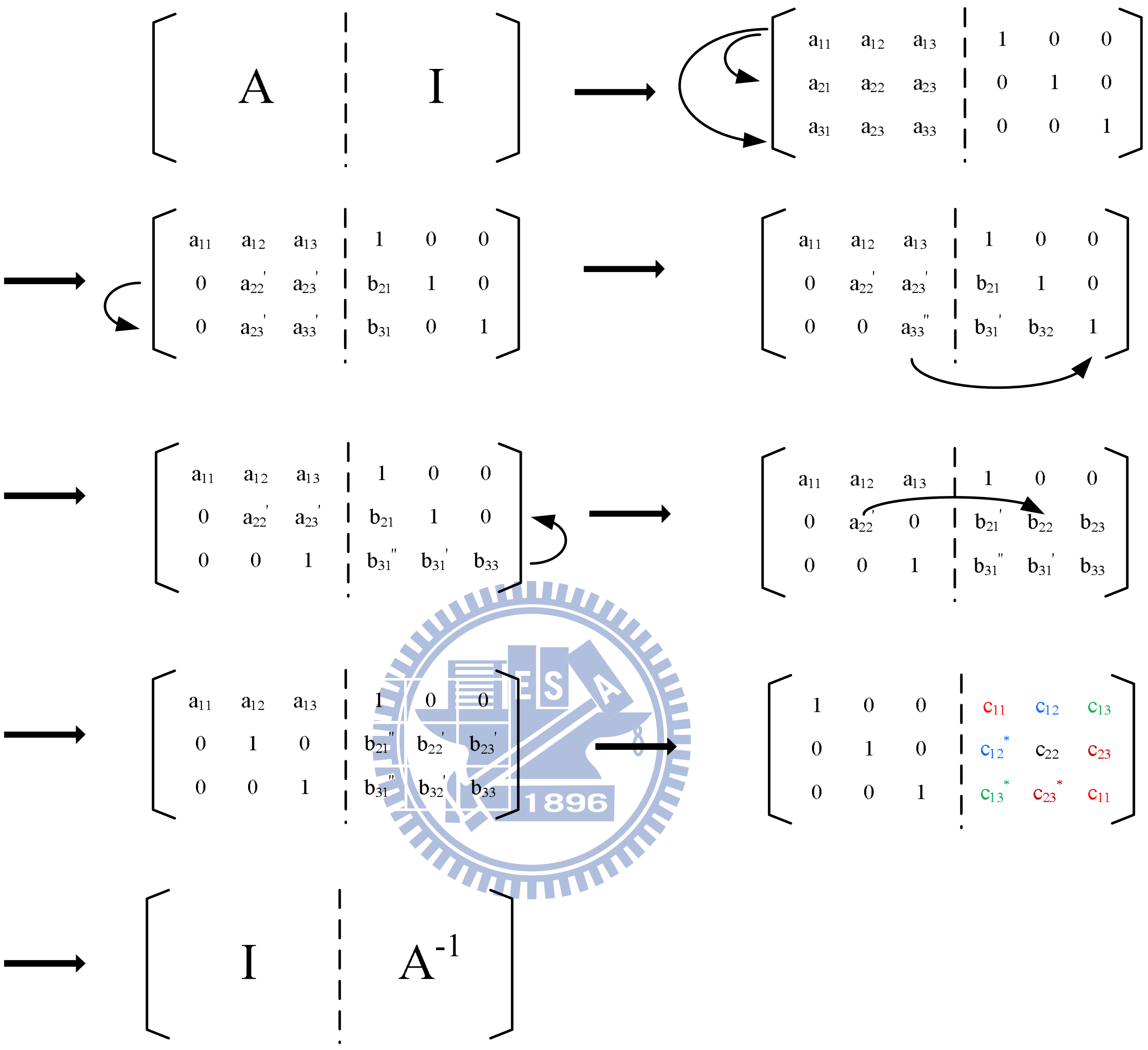

A
I
a_{11} a_{12} a_{13} 1 0 0
a_{21} a_{22} a_{23} 0 1 0
a_{31} a_{23} a_{33} 0 0 1
a_{22}' a_{23}' b_{21}
a_{23}' a_{33}' b_{31}
a_{33}'' b_{31}' b_{32}
b_{31}'' b_{31}' b_{33}
b_{22}' b_{23}
b_{21}'' b_{22}' b_{23}'
b_{31}'' b_{32}' b_{33}
c_{11} c_{12} c_{13}
c_{12}^* c_{22} c_{23}
c_{13}^* c_{23}^* c_{11}
I
A^{-1}

Figure 6.15: Gauss elimination process.

the process, we may lose a large amount of precision in the process. So we also consider other methods to have better control over the scaling of values.

6.5.6.2 Formula Method

The formula method is useful in computation of the inverse of small matrix. It is given by

$$\mathbf{R}^{-1} = \frac{1}{|det|}\begin{bmatrix} \begin{vmatrix} R_0 & R_8 \\ R_8^* & R_0 \end{vmatrix} & \begin{vmatrix} R_8^* & R_0 \\ R_8 & R_{16} \end{vmatrix} & \begin{vmatrix} R_8 & R_{16} \\ R_0 & R_8 \end{vmatrix} \\ \begin{vmatrix} R_{16}^* & R_0 \\ R_8^* & R_8 \end{vmatrix} & \begin{vmatrix} R_0 & R_{16} \\ R_{16}^* & R_0 \end{vmatrix} & \begin{vmatrix} R_8^* & R_8 \\ R_0 & R_{16} \end{vmatrix} \\ \begin{vmatrix} R_8^* & R_0 \\ R_{16}^* & R_8^* \end{vmatrix} & \begin{vmatrix} R_{16}^* & R_8^* \\ R_0 & R_8 \end{vmatrix} & \begin{vmatrix} R_0 & R_8 \\ R_8^* & R_0 \end{vmatrix} \end{bmatrix}. \quad (6.5)$$

The key issue lies in the multiplication by R_0, R_8 and R_{16}. To avoid overflow, we should downscale R_0, R_8 and R_{16} to Q9 (from the original Q14), which means significant loss in precision as result, we do not use this method in the final implementation.

6.5.6.3 Levinson-Durbin Method [22]

The Levinson-Durbin method is a common method for Toeplitz-shaped covariance matrix inverse calculation. Its advantages are a lower computational complexity than the Gauss elimination method in the case of large matrix and simple stability check [22]. We use *short* type and Q15 scaling for all parameters shown in Figs. 6.16 except $w_{1,0}$, $w_{1,1}$, P_0, P_1 and P_2. *Short* type using can accelerate the operation speed and Q15 scaling can maximize the precision. For $w_{1,0}$, $w_{1,1}$, P_0, P_1 and P_2, we let them to Q14 scaling analysis. We modify the formula in [22, pp. 377–378] to accommodate complex values and the process is shown in Figure 6.16. We divide the process into two functions, the upper function responsible to calculate k_1, k_2, $a_{2,1}$, P_0, P_1 and P_2. Another function $Levinson - Durbin$ calculates parameters remains. Figure 6.17 shows the implementation functions for Levinson-Durbin method. Because we do fixed-point implementation, we should set some limits to prevent overflow. Figure 6.18 shows the restrictions on Levinson-Durbin computation for fixed-point implementation.

$$Rw = p \quad R = \begin{pmatrix} r(0) & r(1) & r(2) \\ r^*(1) & r(0) & r(1) \\ r^*(2) & r^*(1) & r(0) \end{pmatrix} \quad w = \begin{pmatrix} w_{2,0} \\ w_{2,1} \\ w_{2,2} \end{pmatrix} \quad p = \begin{pmatrix} p(0) \\ p(1) \\ p(2) \end{pmatrix}$$

$$\boxed{\begin{gathered} P_0 = r(0) \\ c_0 = \frac{p(0)}{P_0} \\ w_{0,0} = c_0 \\ k_1 = \frac{r^*(1)}{P_0} \\ a_{1,1} = k_1 \\ P_1 = (1 - |k_1|^2) \times P_0 \\ c_1 = \frac{p(1) - a_{1,1} \times p(0)}{P_1} \\ w_{1,0} = w_{0,0} - a_{1,1}^* \times c_1 \\ w_{1,1} = c_1 \end{gathered}} \quad \boxed{\begin{gathered} k_2 = \frac{r^*(2) - a_{1,1} \times r^*(1)}{P_1} \\ a_{2,1} = a_{1,1} - k_1 \times a_{1,1}^* \\ a_{2,2} = k_2 \\ P_2 = (1 - |k_2|^2) \times P_1 \\ c_2 = \frac{p(2) - (a_{2,2} \times p(0)) - (a_{2,1} \times p(1))}{P_2} \\ w_{2,0} = w_{1,0} - a_{2,2}^* \times c_2 \\ w_{2,1} = w_{1,1} - a_{2,1}^* \times c_2 \\ w_{2,2} = c_2 \end{gathered}}$$

Figure 6.16: Basic Levinson-Durbin process for complex numbers.

$$P_0 = r(0)$$
$$k_1 = \frac{r^*(1)}{P_0}$$
$$a_{1,1} = k_1$$
$$P_1 = (1 - |k_1|^2) \times P_0$$
$$k_2 = \frac{r^*(2) - a_{1,1} \times r^*(1)}{P_1}$$
$$a_{2,1} = a_{1,1} - k_1 \times a_{1,1}^*$$
$$a_{2,2} = k_2$$
$$P_2 = (1 - |k_2|^2) \times P_1$$

```
$Levinson\_Durbin(short * p, short * P, short * k, short * W)$
{
        $c_0 = \frac{p(0)}{P_0}$
        $w_{0,0} = c_0$
        $c_1 = \frac{p(1) - a_{1,1} \times p(0)}{P_1}$
        $w_{1,0} = w_{0,0} - a_{1,1}^* \times c_1$
        $w_{1,1} = c_1$
        $c_2 = \frac{p(2) - a_{2,2} \times p(0) - a_{2,1} \times p(1)}{P_2}$
        $w_{2,0} = w_{1,0} - a_{2,2}^* \times c_2$    $w_{2,1} = w_{1,1} - a_{2,1}^* \times c_2$    $w_{2,2} = c_2$
    }
}
```

Figure 6.17: Implementation functions for Levinson-Durbin method.

```
if ( $\mathrm{Re}\{p(1)-a_{11}\times p(0)\} > P_1$ )
{
    $\mathrm{Re}\{c_1\} = 32767;(Q15)$
    $\mathrm{Im}\{c_1\} = 0;$
}
else if ( $\mathrm{Im}\{p(1)-a_{11}\times p(0)\} > P_1$ )
{
    $\mathrm{Im}\{c_1\} = 32767;$
    $\mathrm{Re}\{c_1\} = 0;$
}
else
{
    $c_1 = \dfrac{p(1)-a_{11}\times p(0)}{P_1}$
}
```

```
if ( $\mathrm{Re}\{r^*(2)-a_{11}\times r^*(1)\} > P_1$ )
{
    $\mathrm{Re}\{k_2\} = 32767;(Q15)$
    $\mathrm{Im}\{k_2\} = 0;$
}
else if ( $\mathrm{Im}\{r^*(2)-a_{11}\times r^*(1)\} > P_1$ )
{
    $\mathrm{Im}\{k_2\} = 32767;$
    $\mathrm{Re}\{k_2\} = 0;$
}
else
{
    $k_2 = \dfrac{r^*(2)-a_{11}\times r^*(1)}{P_1}$
}
```

```
if ( $\mathrm{Re}\{a_{11}-k_1\times a_{11}^*\} > 32767$ )
{
    $\mathrm{Re}\{a_{21}\} = 32767;(Q15)$
    $\mathrm{Im}\{a_{21}\} = 0;$
}
else if ( $\mathrm{Im}\{a_{11}-k_1\times a_{11}^*\} > 32767$ )
{
    $\mathrm{Im}\{a_{21}\} = 32767;$
    $\mathrm{Re}\{a_{21}\} = 0;$
}
else
{
    $a_{21} = a_{11}-k_1\times a_{11}^*$
}
```

```
if ( $\mathrm{Re}\{p(2)-(a_{22}\times p(0))-(a_{21}\times p(1))\} > P_2$ )
{
    $\mathrm{Re}\{c_2\} = 32767;(Q15)$
    $\mathrm{Im}\{c_2\} = 0;$
}
else if ( $\mathrm{Im}\{p(2)-(a_{22}\times p(0))-(a_{21}\times p(1))\} > P_2$ )
{
    $\mathrm{Im}\{c_2\} = 32767;$
    $\mathrm{Re}\{c_2\} = 0;$
}
else
{
    $c_2 = \dfrac{p(2)-(a_{22}\times p(0))-(a_{21}\times p(1))}{P_2}$
}
```

```
if ( $P_1 == 0$ )
{
    $R^\dagger = \dfrac{1}{9r(0)}\begin{pmatrix} 1 & k_1^* & (k_1^*)^2 \\ k_1 & \|k_1\|^2 & \|k_1\|^2\times k_1^* \\ (k_1)^2 & \|k_1\|^2\times k_1 & \|k_1\|^4 \end{pmatrix}$
    $W = R^\dagger\times p$
}
else
{
    $P_1 = (1-|k_1|^2)\times P_0$
}
```

Figure 6.18: Necessary precautions to maintain stability on fixed-point implementation for Levinson-Durbin method.

6.5.6.4 Lower Bound on Noise Variance

Because of numerical errors in fixed-point calculation, when the SNR is high, too small noise variance in the covariance matrix may cause it to become singular and hamper its inversion. So we set a lower bound to noise variance therein. Assume the original noise variance is σ_1^2. Let there be a compensation noise variance is σ_2^2. Then the total noise variance is $\sigma_n^2 = \sigma_1^2 + \sigma_2^2$. The covariance matrix becomes

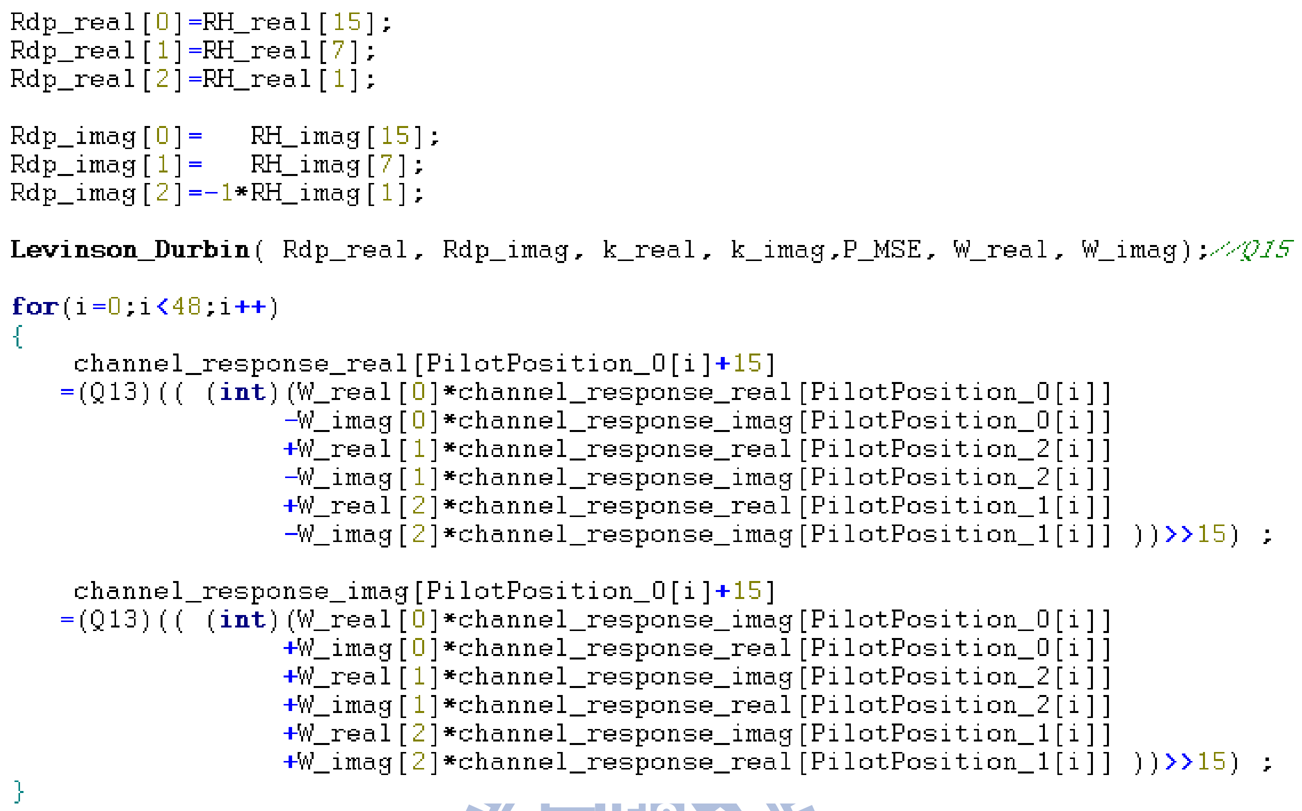
$$\begin{aligned}\mathbf{R} &= \begin{bmatrix} RH[0]+\sigma_1^2+\sigma_2^2 & RH[8] & RH[16] \\ RH[8]^* & RH[0]+\sigma_1^2+\sigma_2^2 & RH[8] \\ RH[16]^* & RH[8]^* & RH[0]+\sigma_1^2+\sigma_2^2 \end{bmatrix} \\ &= \begin{bmatrix} RH[0]+\sigma_n^2 & RH[8] & RH[16] \\ RH[8]^* & RH[0]+\sigma_n^2 & RH[8] \\ RH[16]^* & RH[8]^* & RH[0]+\sigma_n^2 \end{bmatrix}.\end{aligned}$$

This way of lower-boundary the noise variance, however, leads to an error floor in the channel estimation performance. In our experiments, for the downlink system, a lower bound of $\sigma_n^2 = (Q16)7000$ ($SNR \approx 9.714$) dB would get the better performance. For uplink system, a lower bound of $\sigma_n^2 = (Q16)9000$ ($SNR \approx 8.622$) dB is better.

At last we offer a possible solution for error floor. We can maintain the original noise variance in matrix **R** but compensate a little noise variance on c_0 and k_1, where c_0 and k_1 are parameters in Levinson-Durbin method and which has discussed in Section 6.5.6.3. By reducing the amount of noise variance compensation mights be possible to let error floor slightly, but we do not validate the performance are there on this way, maybe which can implement in future work.

6.5.7 Data Channel Response Calculate

After we get the Wiener filter coefficients, we use them to calculate the data subcarrier channel responses. For optimization, Figure 6.19 shows the original code. We can employ the method used in Section 6.5.1 to replace the PilotPosition. Another optimization is



```
Rdp_real[0]=RH_real[15];
Rdp_real[1]=RH_real[7];
Rdp_real[2]=RH_real[1];

Rdp_imag[0]=   RH_imag[15];
Rdp_imag[1]=   RH_imag[7];
Rdp_imag[2]=-1*RH_imag[1];

Levinson_Durbin( Rdp_real, Rdp_imag, k_real, k_imag,P_MSE, W_real, W_imag);//Q15

for(i=0;i<48;i++)
{
    channel_response_real[PilotPosition_0[i]+15]
   =(Q13)(( (int)(W_real[0]*channel_response_real[PilotPosition_0[i]]
                 -W_imag[0]*channel_response_imag[PilotPosition_0[i]]
                 +W_real[1]*channel_response_real[PilotPosition_2[i]]
                 -W_imag[1]*channel_response_imag[PilotPosition_2[i]]
                 +W_real[2]*channel_response_real[PilotPosition_1[i]]
                 -W_imag[2]*channel_response_imag[PilotPosition_1[i]] ))>>15) ;

    channel_response_imag[PilotPosition_0[i]+15]
   =(Q13)(( (int)(W_real[0]*channel_response_imag[PilotPosition_0[i]]
                 +W_imag[0]*channel_response_real[PilotPosition_0[i]]
                 +W_real[1]*channel_response_imag[PilotPosition_2[i]]
                 +W_imag[1]*channel_response_real[PilotPosition_2[i]]
                 +W_real[2]*channel_response_imag[PilotPosition_1[i]]
                 +W_imag[2]*channel_response_real[PilotPosition_1[i]] ))>>15) ;
}
```

Figure 6.19: C code for data subcarrier response calculation before modification.

replacing the pointer. Because we save the Wiener filter coefficients in an array, when we call them for multiplications, we have to use an array pointer to obtain their values, which result in multilevel data loading and cost large amount of cycles. So we first place the Wiener filter coefficients into temporary registers as shown in Figure 6.20 and use the temporary registers for multiplication. Table 6.5 shows the cycles and efficiency evaluation of all the modifications for data subcarrier response calculation, the field names Rounding means we consider the rounding effect in our program, we can see it would increase additional cycles. The average efficiency for fifteen data subcarrier can be calculated by : Symbols $\times$ SNR $\times$ PRU $\times$ multiplications $= 28 \times 11 \times 48 \times 6 = 88704$. Efficiency is $88704/143896.8 = 61.64\%$. We can see good performance from the result.

```
Rdp_real[0]=RH_real[15];
Rdp_real[1]=RH_real[7];
Rdp_real[2]=RH_real[1];

Rdp_imag[0]=   RH_imag[15];
Rdp_imag[1]=   RH_imag[7];
Rdp_imag[2]=-1*RH_imag[1];

Levinson_Durbin( Rdp_real,Rdp_imag,k_real,k_imag,P_MSE,W_real,W_imag);

temp0_real = W_real[0];
temp0_imag = W_imag[0];
temp1_real = W_real[1];
temp1_imag = W_imag[1];
temp2_real = W_real[2];
temp2_imag = W_imag[2];

for(i=0;i<48;i++)
{
    channel_response_real[18*i+15]
  =(Q13)(( (int)(temp0_real*channel_response_real[18*i]
                -temp0_imag*channel_response_imag[18*i]
                +temp1_real*channel_response_real[8+18*i]
                -temp1_imag*channel_response_imag[8+18*i]
                +temp2_real*channel_response_real[16+18*i]
                -temp2_imag*channel_response_imag[16+18*i] )+16384)>>15) ;


    channel_response_imag[18*i+15]
   =(Q13)(( (int)(temp0_real*channel_response_imag[18*i]
                 +temp0_imag*channel_response_real[18*i]
                 +temp1_real*channel_response_imag[8+18*i]
                 +temp1_imag*channel_response_real[8+18*i]
                 +temp2_real*channel_response_imag[16+18*i]
                 +temp2_imag*channel_response_real[16+18*i] )+16384)>>15) ;
}
```

Figure 6.20: C code for data subcarrier response calculation after modification.

Table 6.5: Data Subcarrier Channel Response Calculation Clock Cycles Comparison

Method Carrier index (0~17)	Original	PilotPosition replacement	Pointer replacement	Rounding
0				
1	425236	165841	166629	166265
2	426580	167244	165088	172172
3	426662	166937	164780	166529
4	426580	168685	163758	163720
5	428428	152192	112763	132963
6	425093	103918	106876	135828
7	423856	152460	107986	142604
8				
9	423707	152460	113960	135828
10	426844	152460	113960	136064
11	424082	152460	113960	135828
12	425275	152460	106876	135828
13	426272	152460	105028	132992
14	424580	149819	104798	132913
15	423620	149720	104296	133090
16				
17	424202	152460	113960	135828
Average	425401.133	152771.733	124314.533	143896.8
Efficiency	20.85%	58.06%	71.35%	61.64%

6.5.8 Summary

According to the above discussion, we reduce the total clock cycle time and code size successfully. Finally, DSP C6000 compiler compile our C code and packages to a out file, which size is 273 KB. Table 6.6 shows the clock cycles for all C code functions. Incl.Total cycle means total number of cycles for all executions including function calls. Subfunction *channel_estimation_fixed* includes all the functions described in Figure 6.8. And subfunction *interpolation* includes all the function described in Figure 6.8 except pilot subcarrier channel estimation. Subfunction *main* deals with SNR and symbol number setting. *ParameterTable* record the pilot value. Because of we do not write all the C code as functions, so we just approximate the proportion for total cycles. Table 6.7 shows the proportion for total cycles without considering TI library.

The original C code without any optimization needs 13340267 clock cycles for calculation, after our optimized methods as describes in this chapter, which just need 9587041 cycles. We reduced to approximate 72% cycles of the original program. We try to find the required time per OFDMA symbol, the required cycles are total cycles counts $\times$ clock cycle time $\div$ (SNR $\times$ Symbols) $= 9587041 \times 10^{-9} \div (11 \times 28) = 31.13(\mu s)$, and OFDMA symbol period are (FFT size + CP length $\div$ sampling frequency $= 1152 \div (11.2 \times 10^{6}) = 102.86(\mu s)$. In the other word, we only use 30% symbol period for channel estimation, which remains 70% symbol period for the other use, just like channel coding, synchronization, etc.

Table 6.6: Downlink Channel Estimation Clock Cycle Table from the Function Aspect

Symbol Name	Symbol Type	Access Count	cycle.Total: Incl. Total	cycle.Total: Excl. Total	
Levinson_Durbin	function	8932	5056491	3592638	37.47%
interpolation	function	308	9115806	3364697	35.10%
_divi <TI Library>	function	56634	1206137	1206137	12.58%
memcpy <TI Library>	function	11764	707276	707276	7.38%
Sin_Fix	function	5202	315037	315037	3.29%
pilot_extraction	function	308	104926	97842	1.02%
Cos_Fix	function	2601	241925	84410	0.88%
_divu <TI Library>	function	2396	53733	53733	0.56%
main	function	1	26976923	41298	0.43%
Atan_Fix	function	305	40096	31169	0.33%
channel_estimation_fixed	function	341	9262628	29917	0.31%
DIV_fixed	function	272	32249	23817	0.25%
_remi <TI Library>	function	1232	20798	20798	0.22%
Sqrt_Fix	function	350	13696	13696	0.14%
SqrtTable	function	1	2556	2556	0.03%
AtanTable	function	1	1159	1159	0.01%
SinTable	function	1	861	861	0.01%
Total required cycles				9587041	

Table 6.7: Downlink Channel Estimation Clock Cycle Proportional Distributed

Function name	Percentage
Pilot subcarrier channel estimation	1.69%
Time domain interpolation of pilot channel response	4.66%
Calculation of R_0 and R_1	3.22%
Rmstemp and angle calculation	1.11%
Correlation function calculate	7.19%
Wiener coefficient calculation	29.04%
Data channel response calculate	46.64%
Remains	6.45

Chapter 7

Fixed-Point Simulation Results

7.1 DL Simulation Results

7.1.1 Validation with AWGN Channel

We verify the correctness of the program code by simulation AWGN channel transmission. Both SISO and SFBC MIMO transmission are consider. We run 10^5 symbols in each simulation to obtain the numerical results. Figs. 7.1 shows the DL channel estimation performance in SISO transmission for AWGN channel. In our results, the MSE performance for fixed-point implementation is very close to floating-point result. Although we set a lower bound to noise variance, the error floor phenomenon is not obviously for MSE in AWGN channel.

7.1.2 SISO Transmission Results

We conduct simulation with the six SUI channels to examine the channel estimation performance. Figure 7.2 and Figure 7.3 shows the simulation results in SUI-2 channel and SUI-5 channel, respectively. In our result, because of the precise problem, the MSE performance of channel estimation different velocities become hardly distinguishably. When SNR small, the MSE curve for fixed-point is very close to floating-point, but if the SNR value is bigger than 12 dB, because we set a lower bound to avoid singular, the error floor would occurrence.

Another numerical results, please refer to Appendix E.

7.1.3 SFBC Transmission Results

SFBC is a technique that uses multiple antennas to achieve better diversity effect. Figure 7.5 and Figure 7.6 shows the simulation results in SUI-2 channel and SUI-5 channel, respectively. In our simulation, we use two transmit antennas and one receive antenna. We estimate the two channel responses separately. So the MSE performance is not different by using SFBC as compared to SISO, but the SER performance is better. To check with our results, the fixed-point MSE curve is very close to floating point and which has similar SER performance to floating-point in AWGN channel. Another numerical results, please refer to Appendix F.

7.2 UL Simulation Results

7.2.1 Validation with AWGN Channel

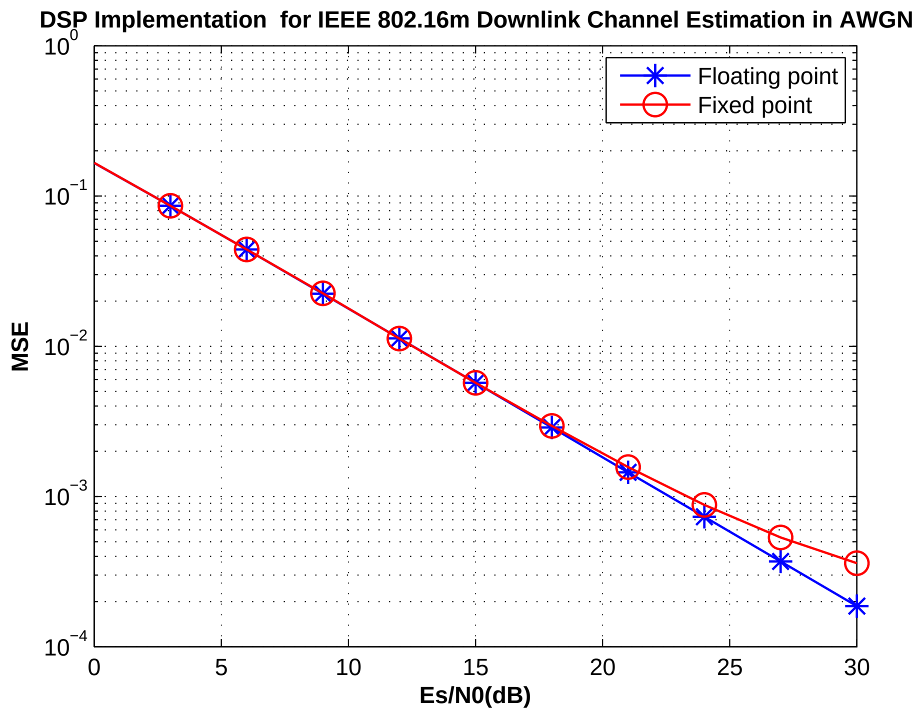
We verify the correctness of the program code by simulation AWGN channel transmission. Both SISO and SFBC MIMO transmission are consider. We run 10^5 symbols in each simulation to obtain the numerical results. Figure 7.7 and Figure 7.8 shows the UL channel estimation performance for each frequency partition (FP). For downlink system, we use 48 PRUs for average to do the LMMSE channel estimation, but in uplink, we only consider the CRUs for each frequency partition. For FP0 which uses 6 PRUs for average to calculate LMMSE channel estimation. For FP1, FP2 and FP3 which use 8 PRUs for average. However the average numbers are different for each frequency partition, but we can not consider the performance of FP0 would worse than other frequency partitions, we can only confirm the standard deviation of mean delay for FP0 would higher than other frequency partitions. To check with our results, fixed-point performance is very close to floating-point.

7.2.2 SISO Transmission Result

Figures 7.9 to 7.12 shows the simulation results in SUI-2 and SUI-5 channel, respectively. In our uplink results, however we set different noise variance lower bound (8.622 dB) with downlink (9.714 dB), the error floor would appear at 12 dB similarly. Another numerical results, please refer to Appendix G.

7.2.3 SFBC Transmission Result

Figures 7.15 to 7.18 shows the simulation results in SUI-2 and SUI-5 channel, respectively. In our simulation, we use two transmit antennas and one receive antenna. We estimate the two channel responses separately. So the MSE performance is not different by using SFBC as compared to SISO, but the SER performance is better. Another numerical results, please refer to Appendix H.



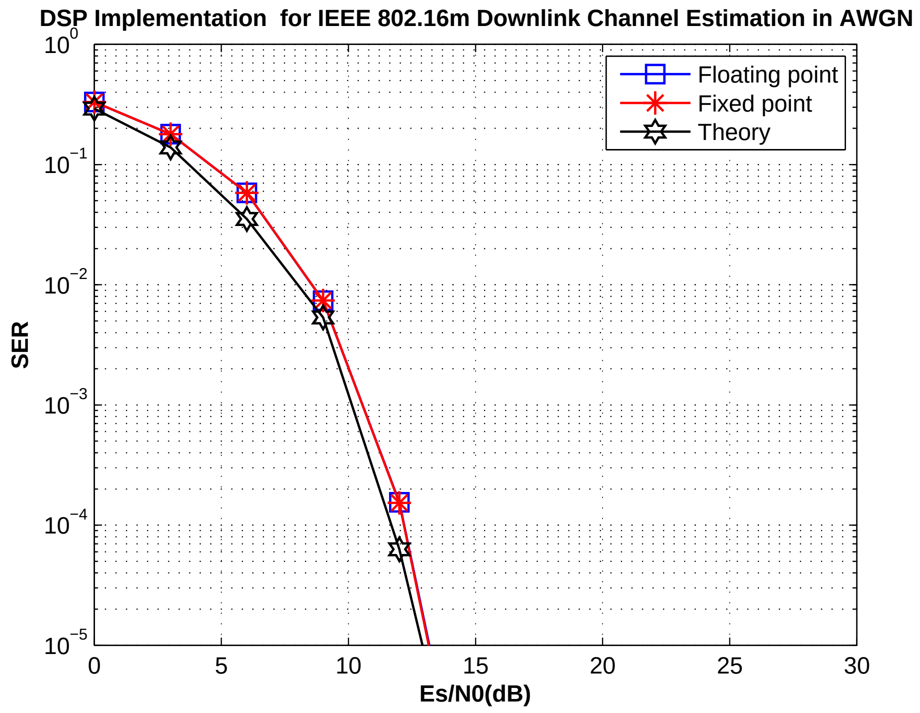
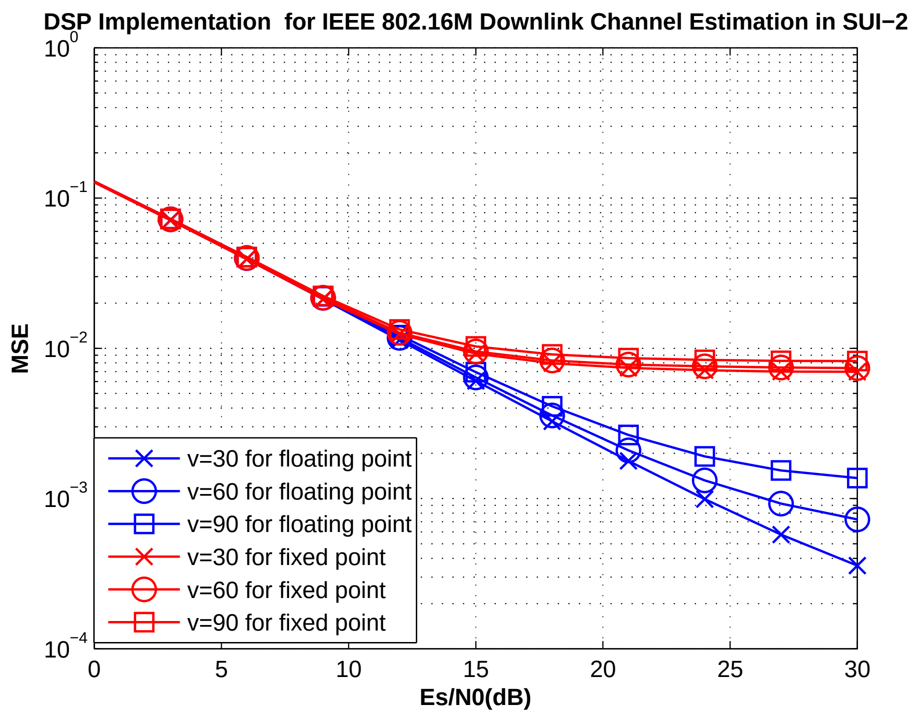


Figure 7.1: Fixed-point channel estimation MSE and SER for QPSK in AWGN for IEEE 802.16m downlink.



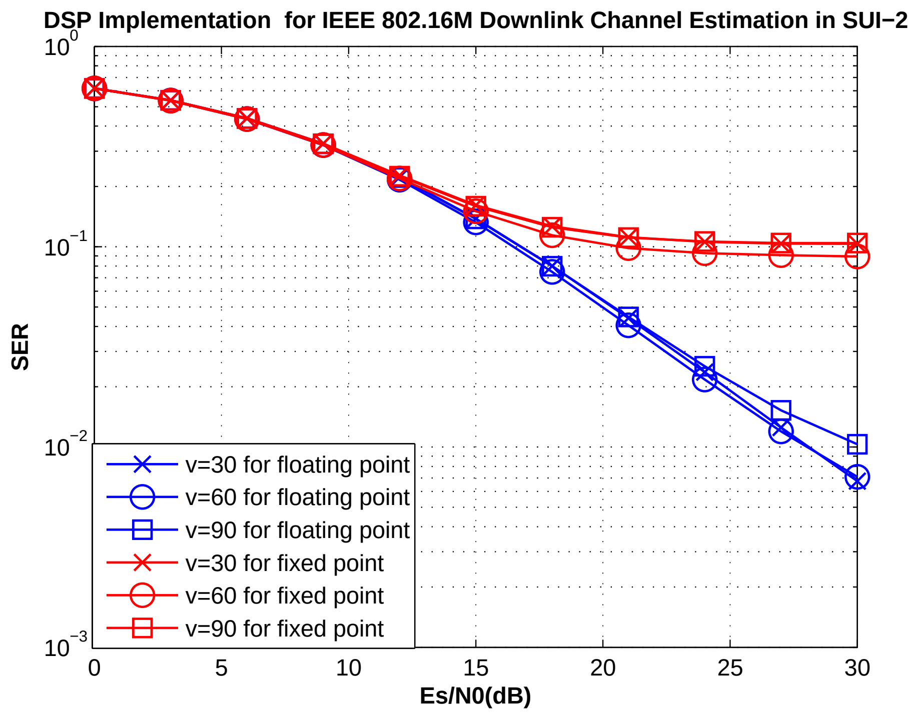
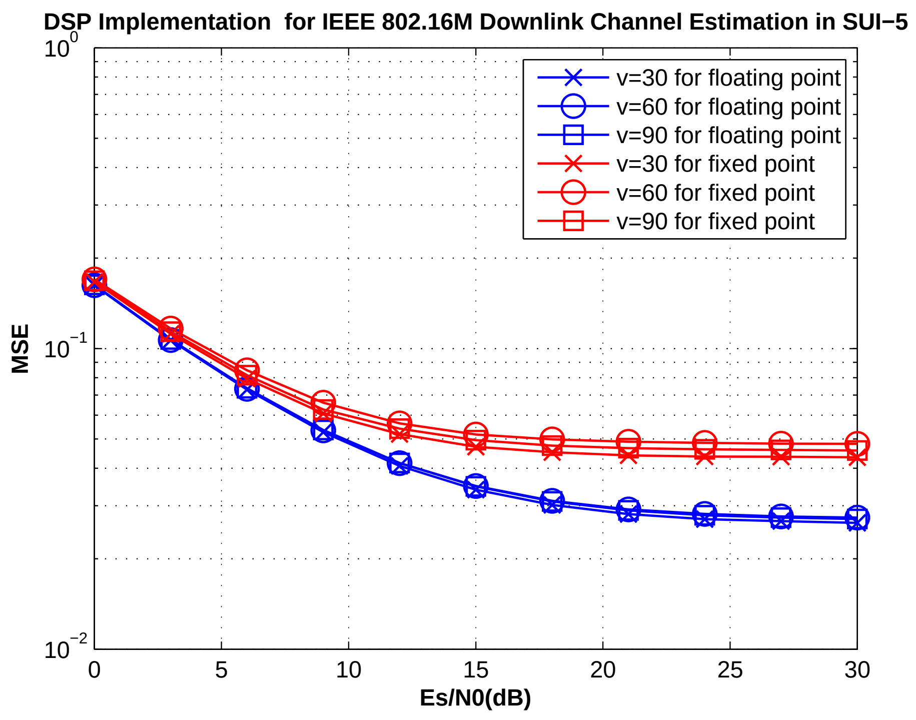


Figure 7.2: Fixed-point channel estimation MSE and SER for QPSK at different velocities in SUI-2 channel for IEEE 802.16m downlink, where the speed v is km/h.



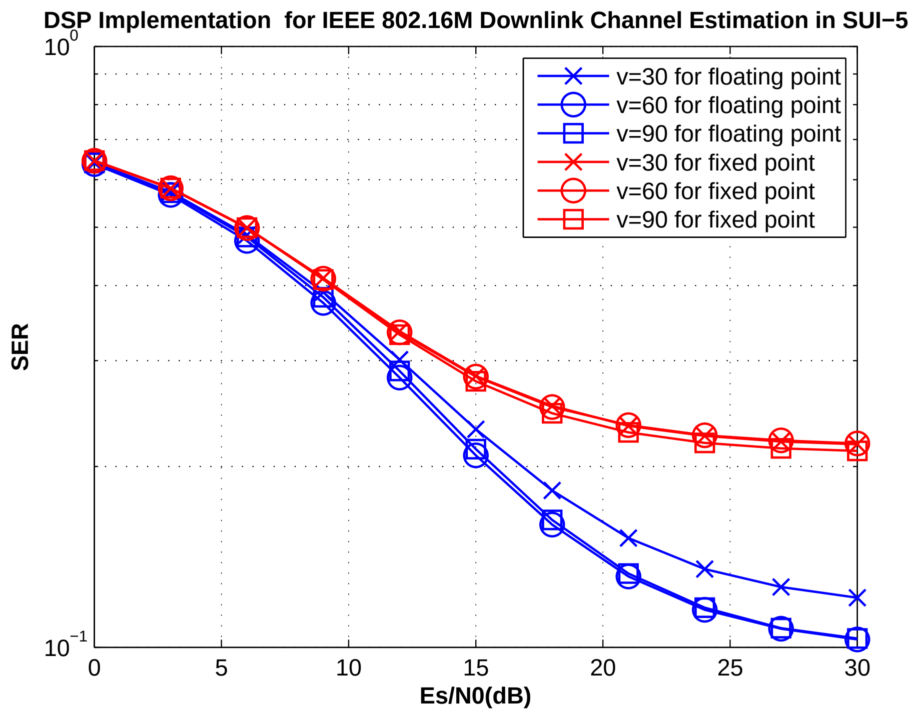
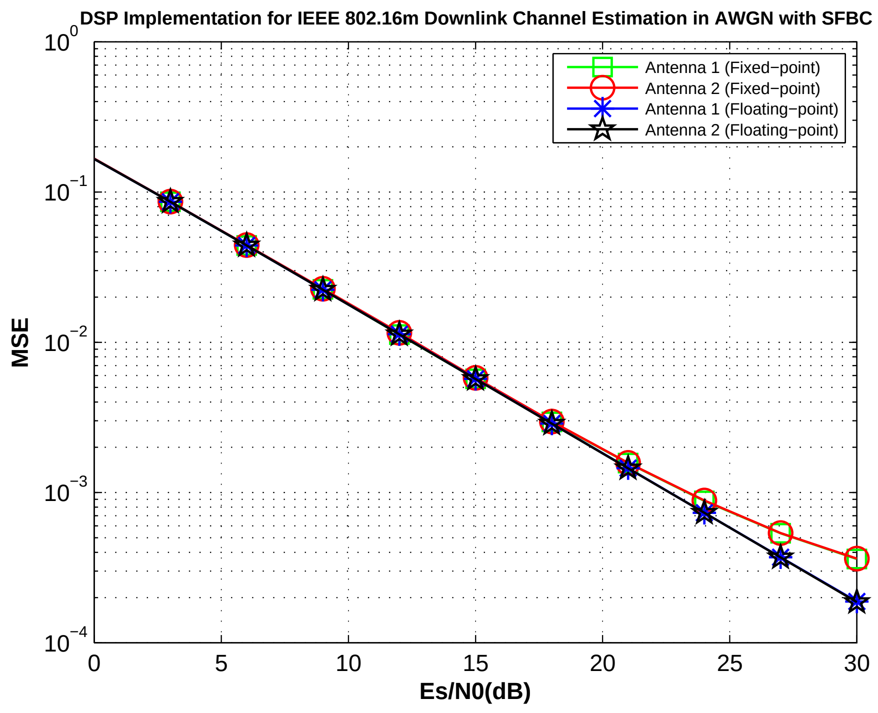


Figure 7.3: Fixed-point channel estimation MSE and SER for QPSK at different velocities in SUI-5 channel for IEEE 802.16m downlink, where the speed v is km/h.



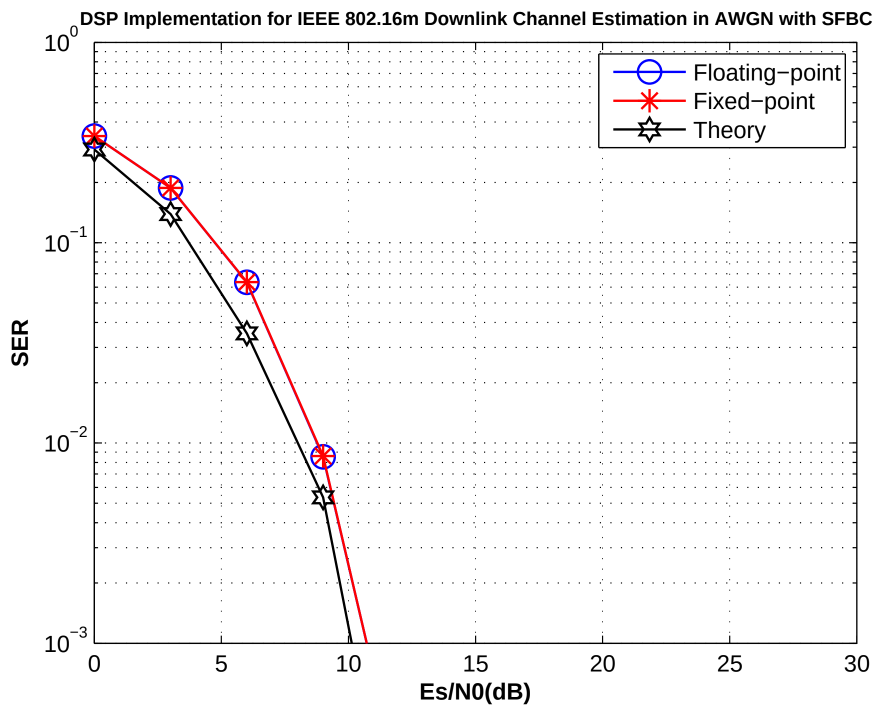
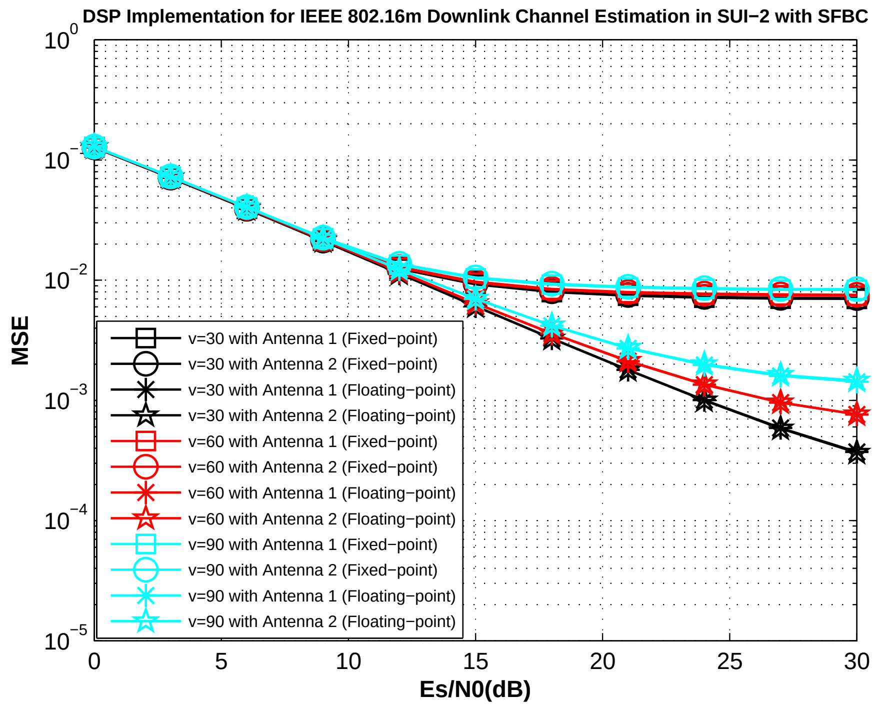


Figure 7.4: Fixed-point channel estimation MSE and SER for QPSK with SFBC in AWGN channel for IEEE 802.16m downlink.



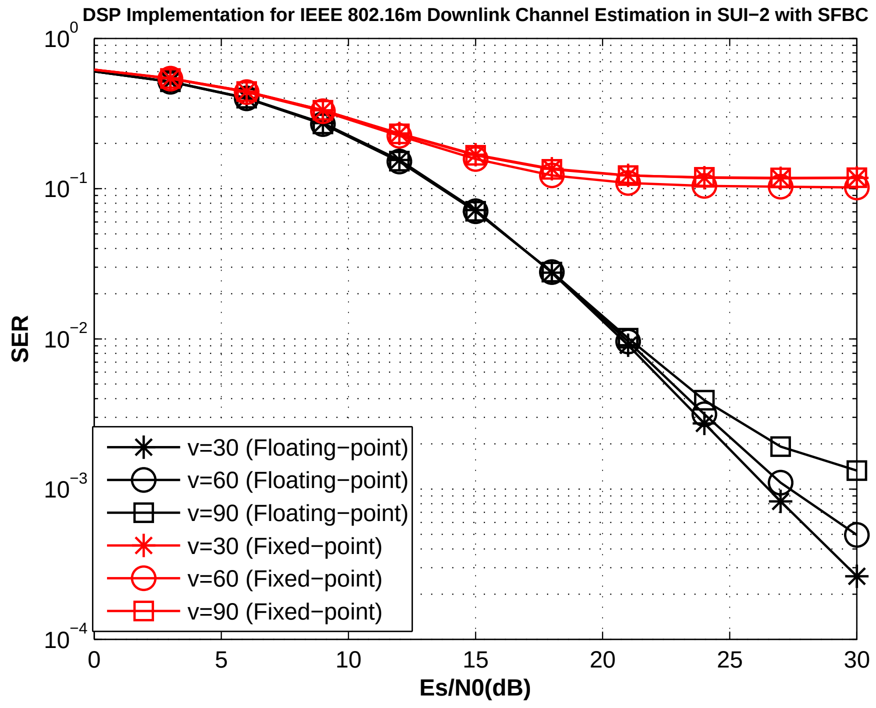
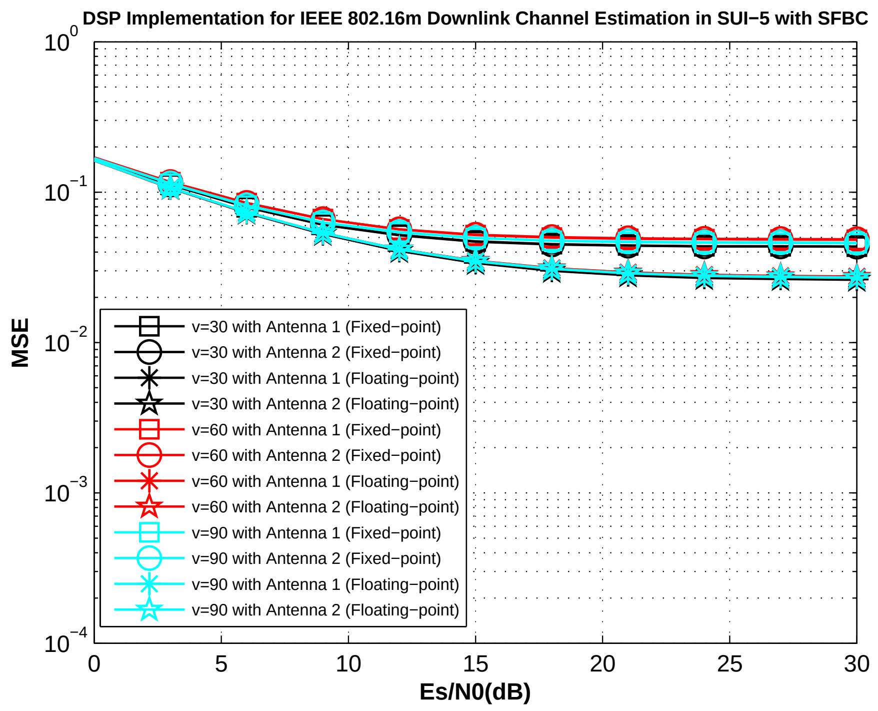


Figure 7.5: Fixed-point channel estimation MSE and SER for QPSK with SFBC at different velocities in SUI-2 channel for IEEE 802.16m downlink, where the speed v is km/h.



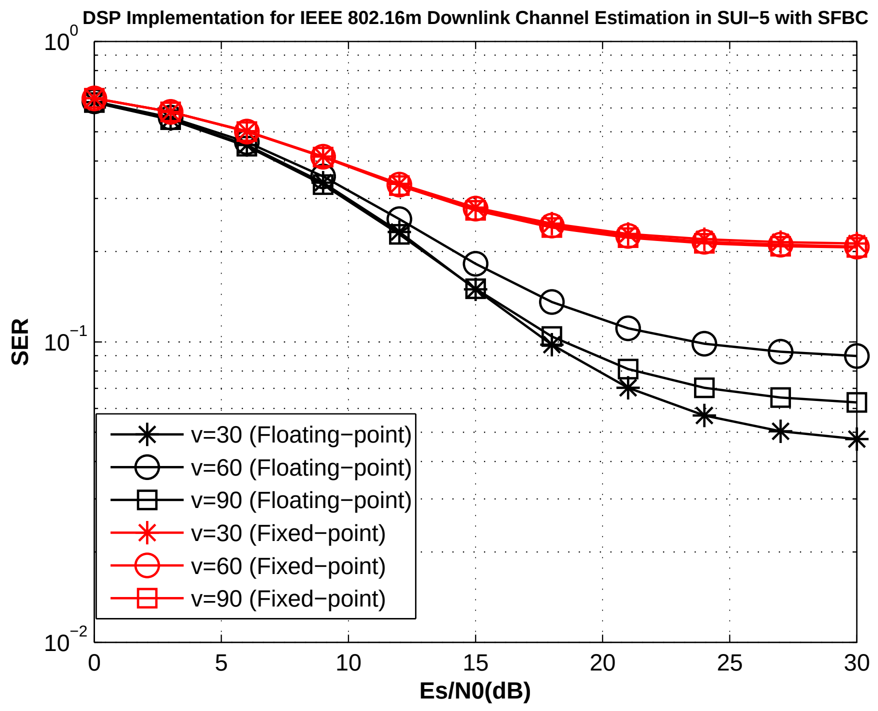
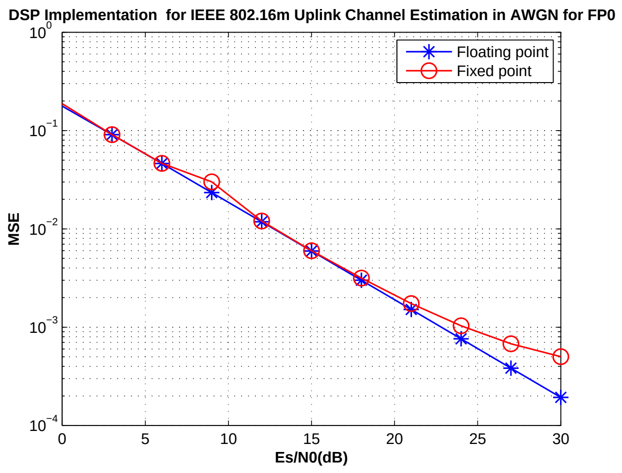
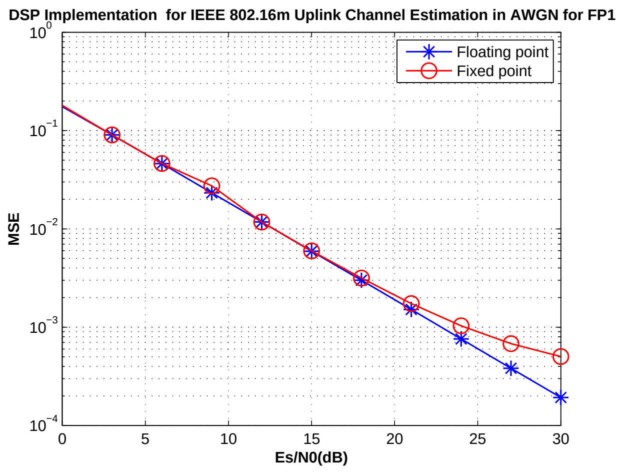


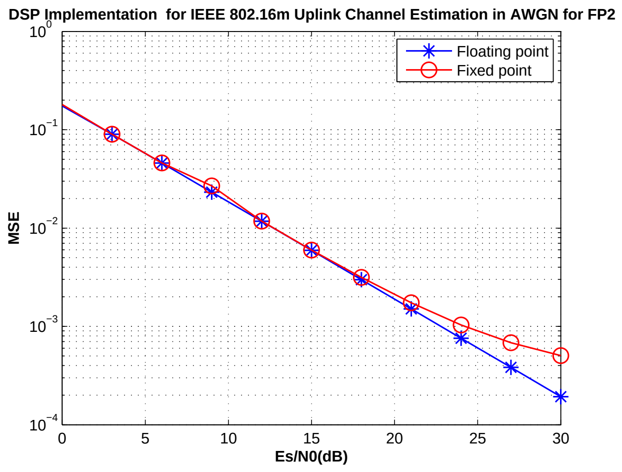
Figure 7.6: Fixed-point channel estimation MSE and SER for QPSK with SFBC at different velocities in SUI-5 channel for IEEE 802.16m downlink, where the speed v is km/h.



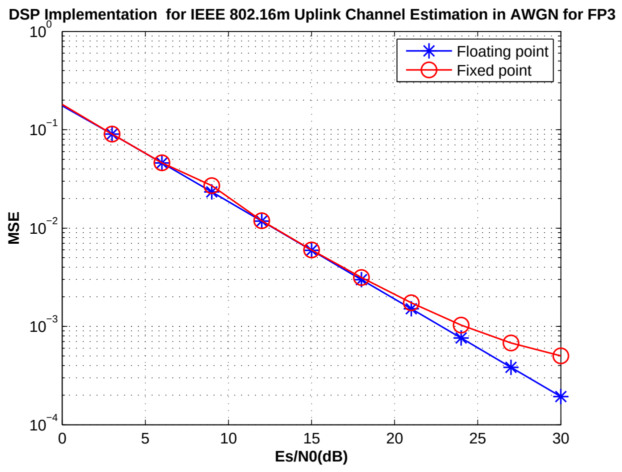
(a)



(b)

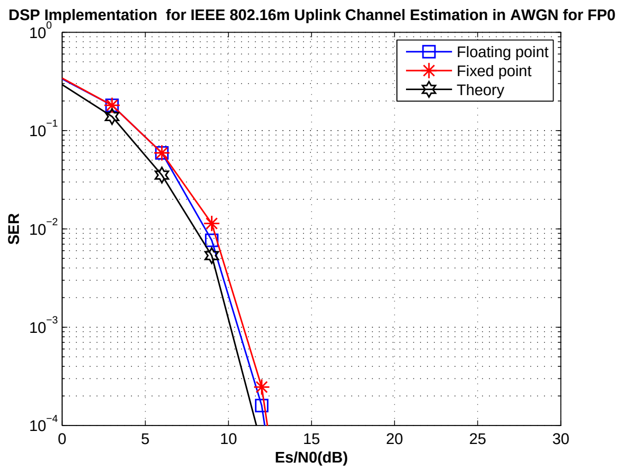


(c)

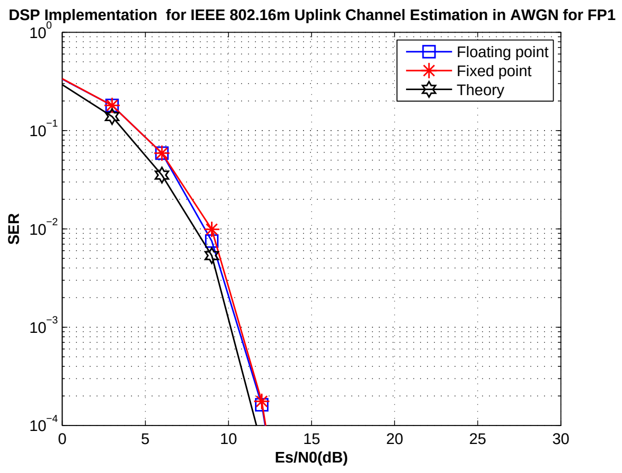


(d)

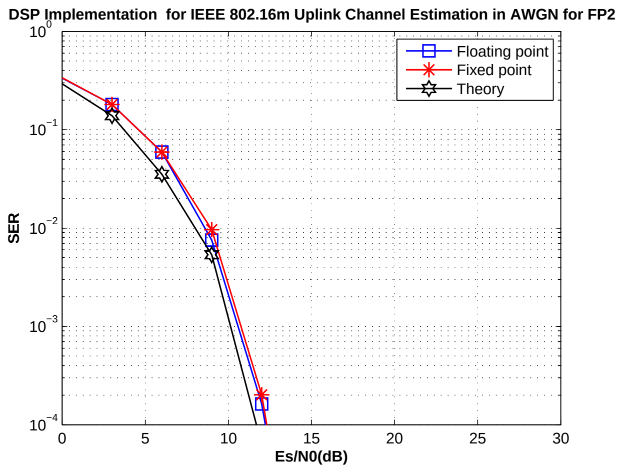
Figure 7.7: Fixed-point channel estimation MSE use QPSK modulation for different frequency partitions at different velocities in AWGN channel for IEEE 802.16m uplink.



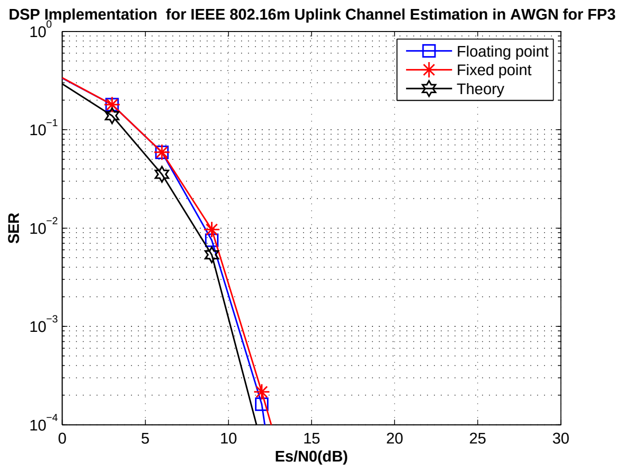
(a)



(b)



(c)



(d)

Figure 7.8: Fixed-point channel estimation SER use QPSK modulation for different frequency partitions at different velocities in AWGN channel for IEEE 802.16m uplink.

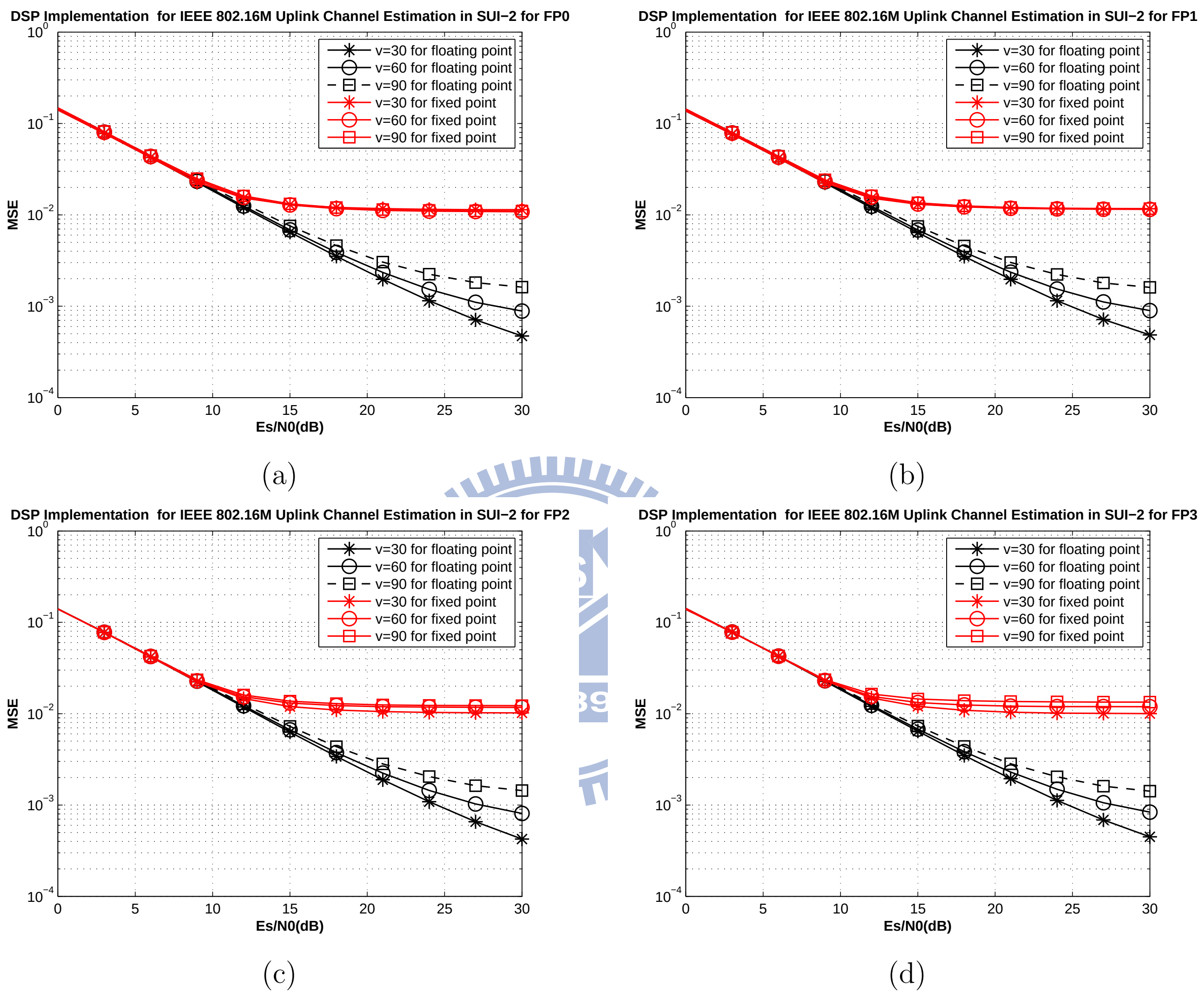
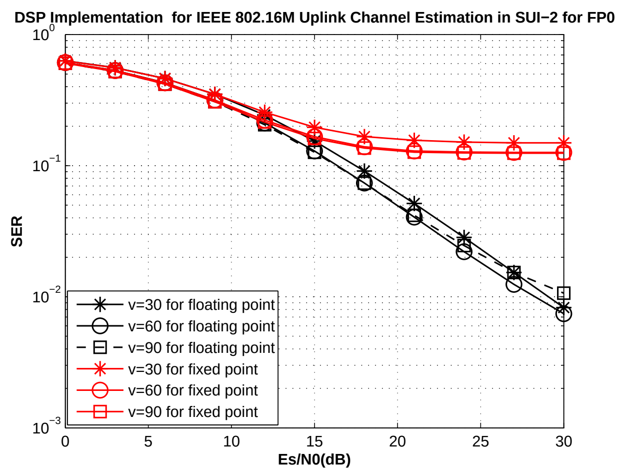
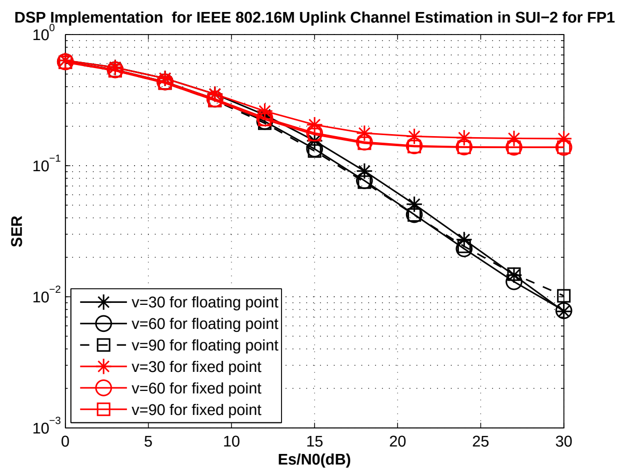


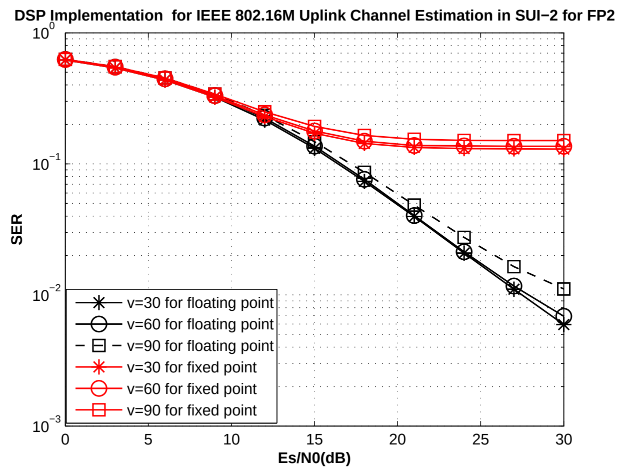
Figure 7.9: Fixed-point channel estimation MSE use QPSK modulation for different frequency partitions at different velocities in SUI-2 channel for IEEE 802.16m uplink, where the speed v is km/h.



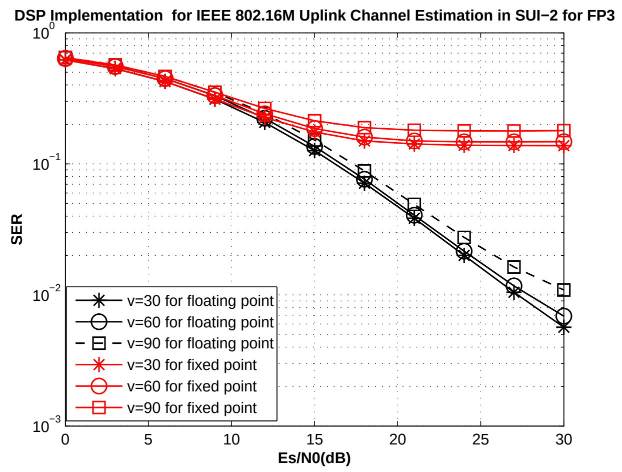
(a)



(b)

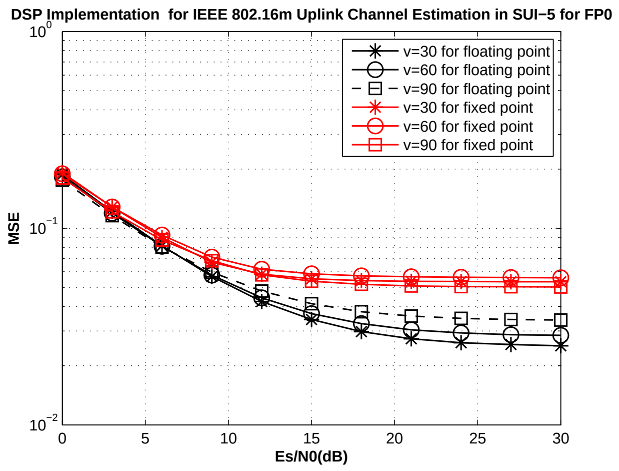


(c)

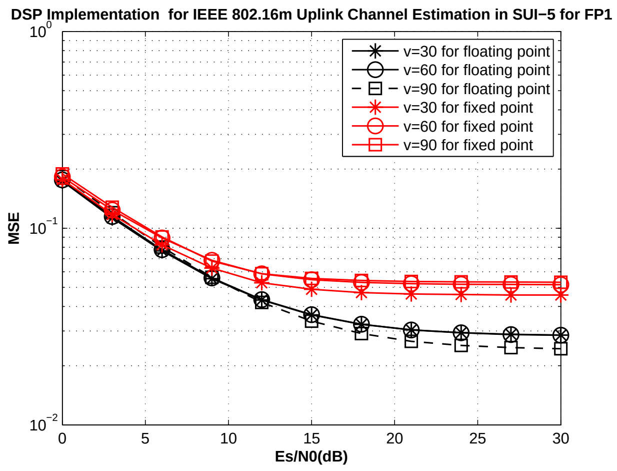


(d)

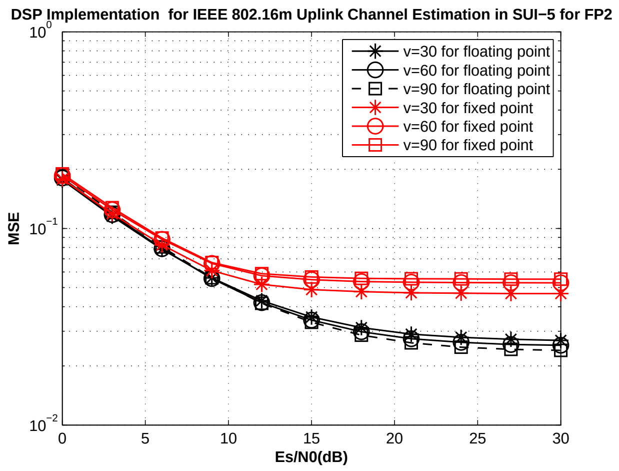
Figure 7.10: Fixed-point channel estimation SER use QPSK modulation for different frequency partitions at different velocities in SUI-2 channel for IEEE 802.16m uplink, where the speed v is km/h.



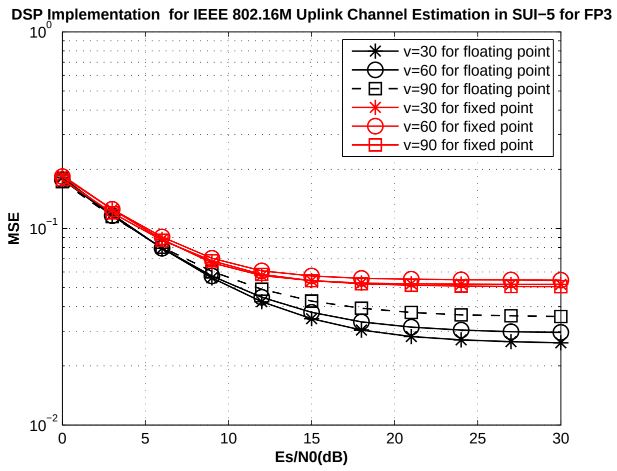
(a)



(b)

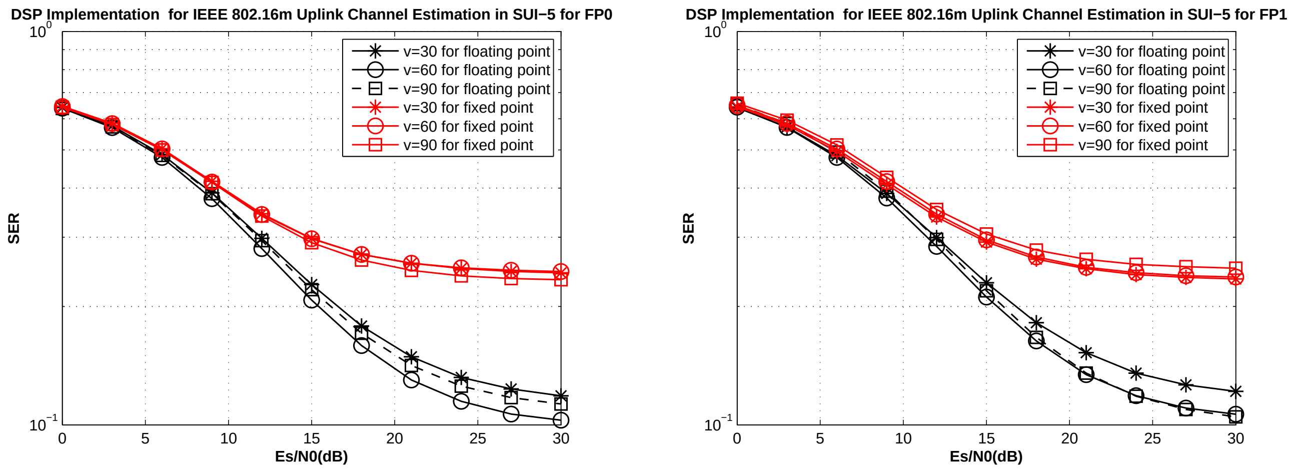


(c)

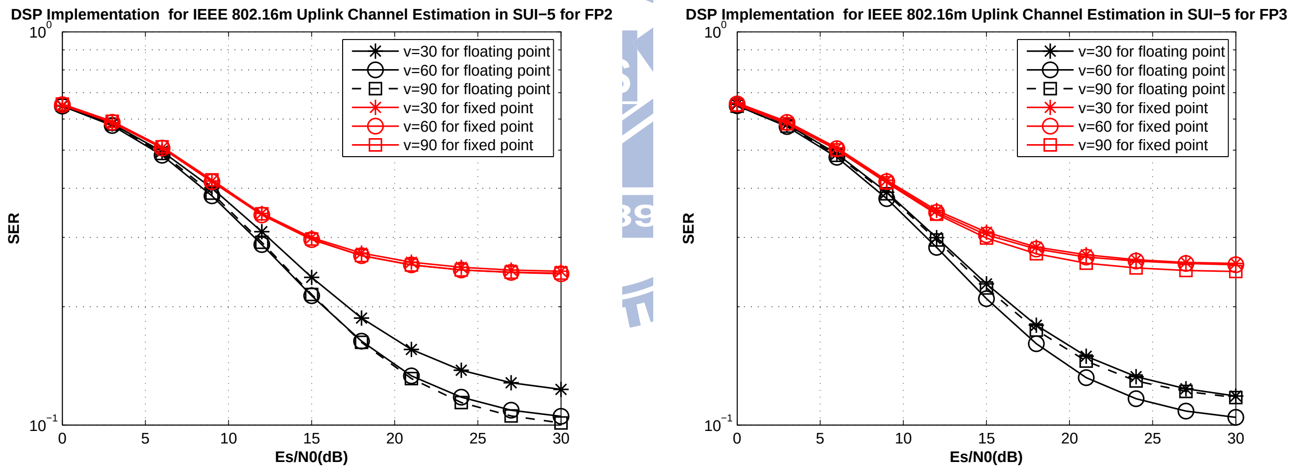


(d)

Figure 7.11: Fixed-point channel estimation MSE use QPSK modulation for different frequency partitions at different velocities in SUI-5 channel for IEEE 802.16m uplink, where the speed v is km/h.

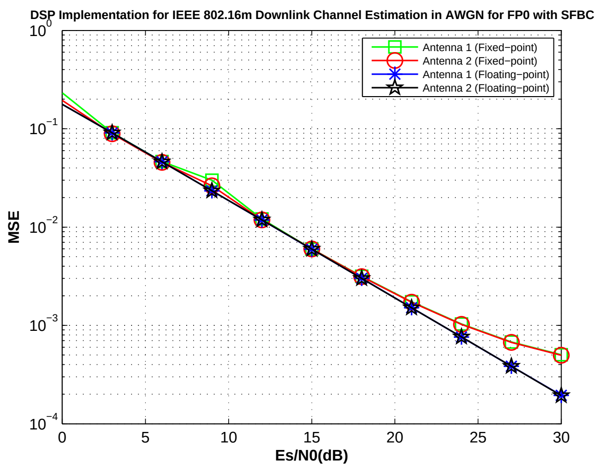


(a) (b)

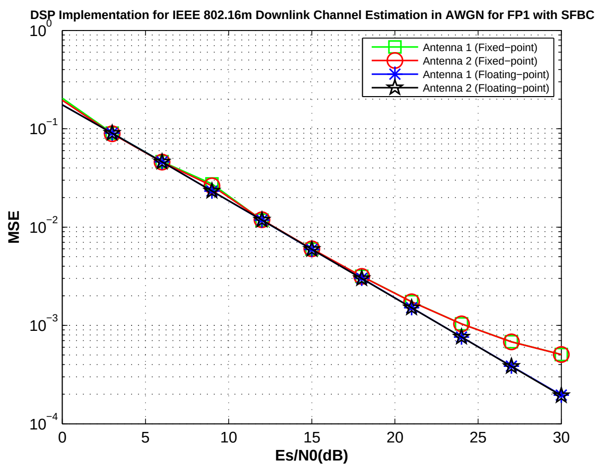


(c) (d)

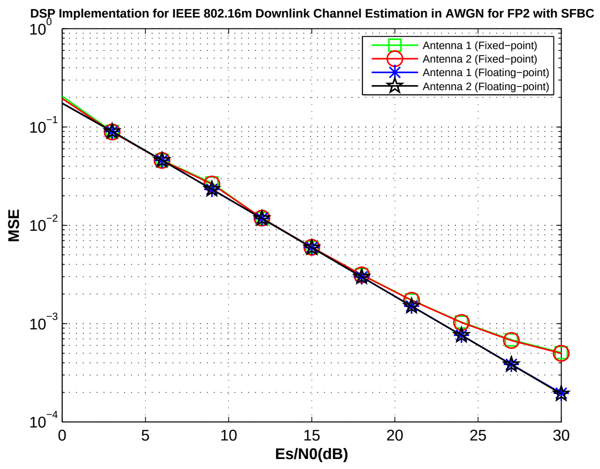
Figure 7.12: Fixed-point channel estimation SER use QPSK modulation for different frequency partitions at different velocities in SUI-5 channel for IEEE 802.16m uplink, where the speed v is km/h.



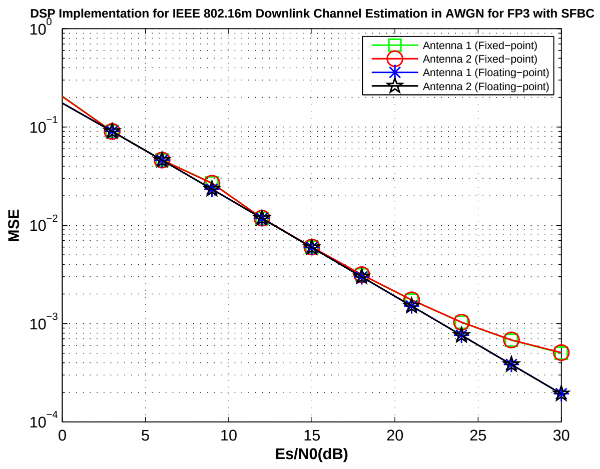
(a)



(b)

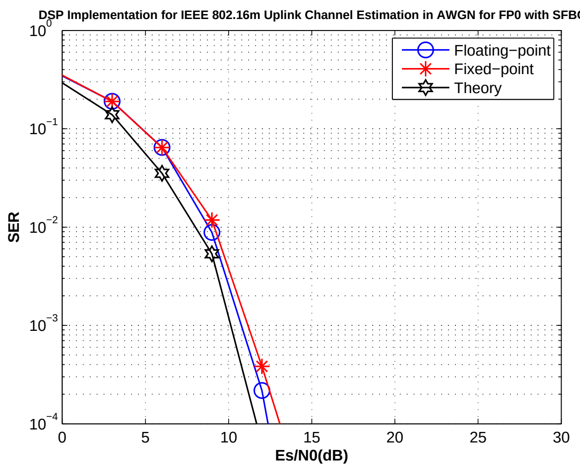


(c)

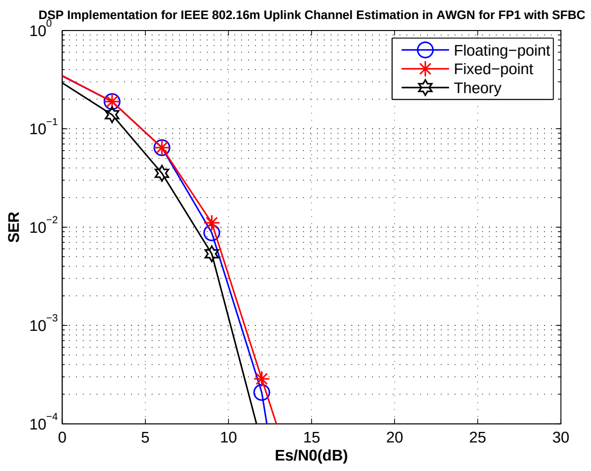


(d)

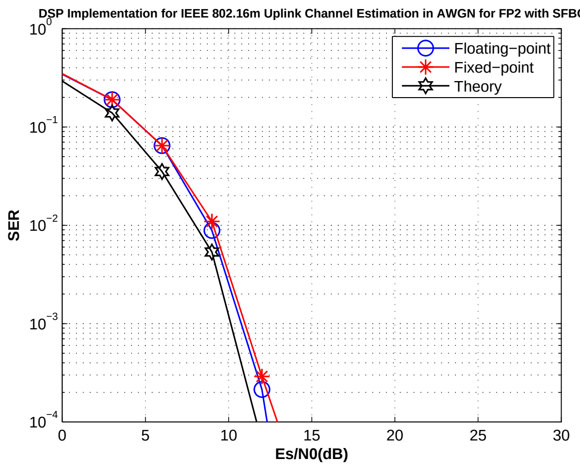
Figure 7.13: Fixed-point channel estimation MSE use QPSK modulation with SFBC for different frequency partitions at different velocities in AWGN Fixed-point channel for IEEE 802.16m uplink.



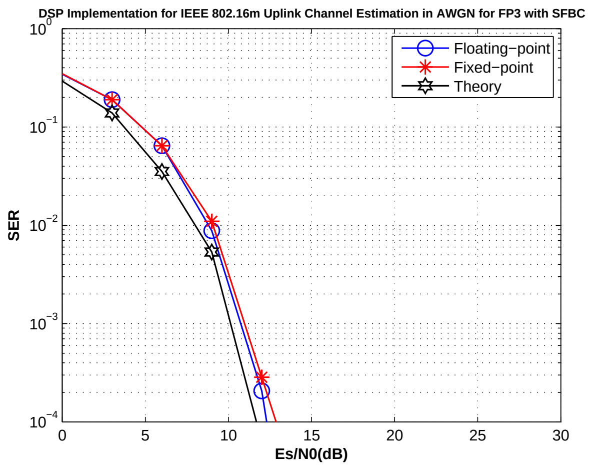
(a)



(b)

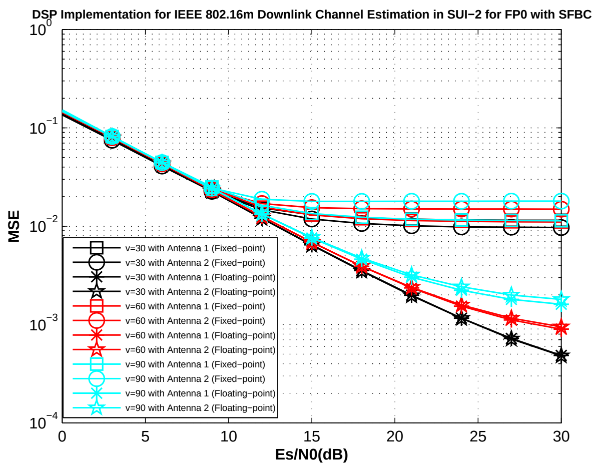


(c)

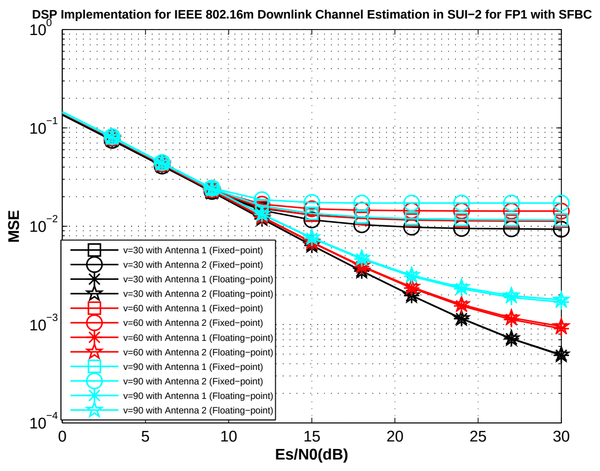


(d)

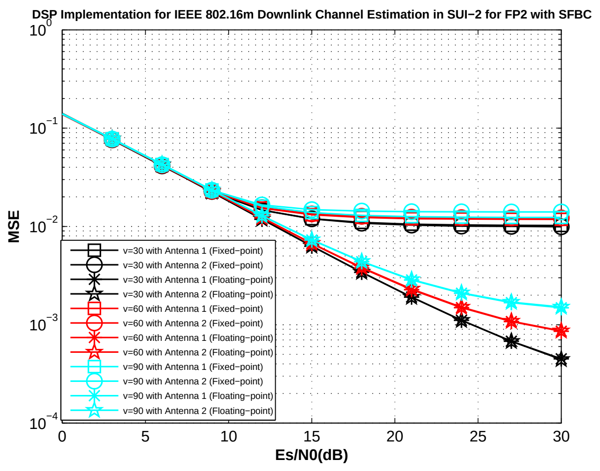
Figure 7.14: Fixed-point channel estimation SER use QPSK modulation with SFBC for different frequency partitions at different velocities in AWGN Fixed-point channel for IEEE 802.16m uplink.



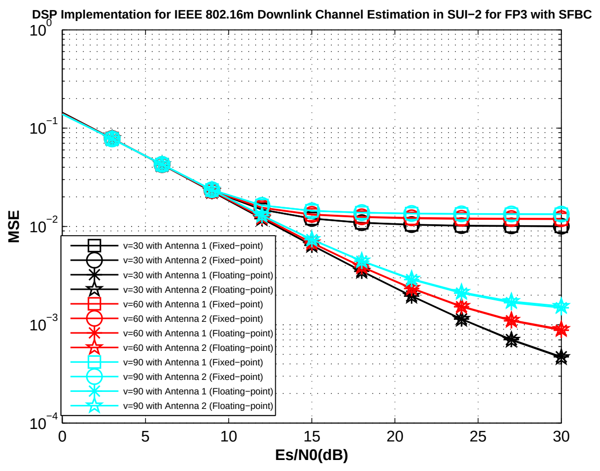
(a)



(b)

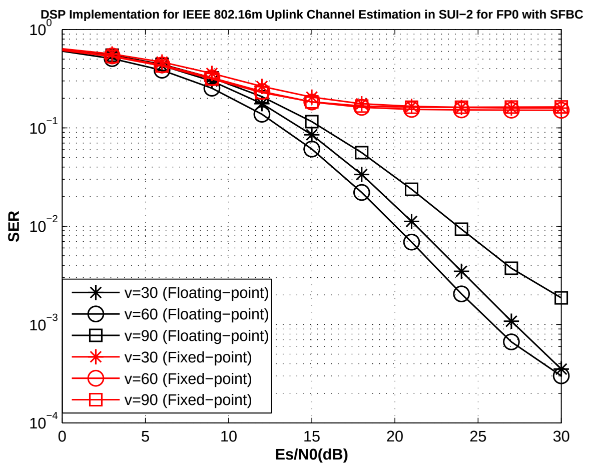


(c)

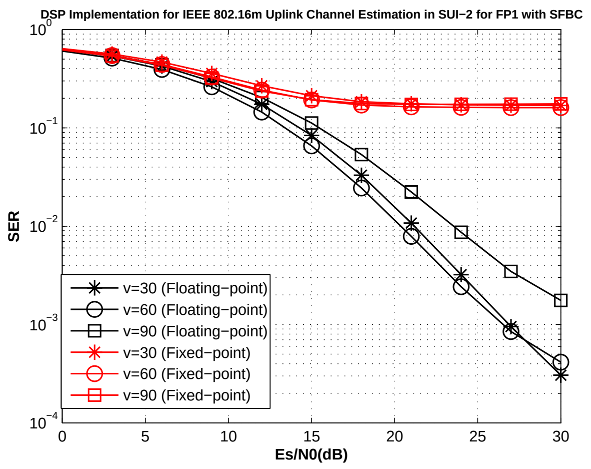


(d)

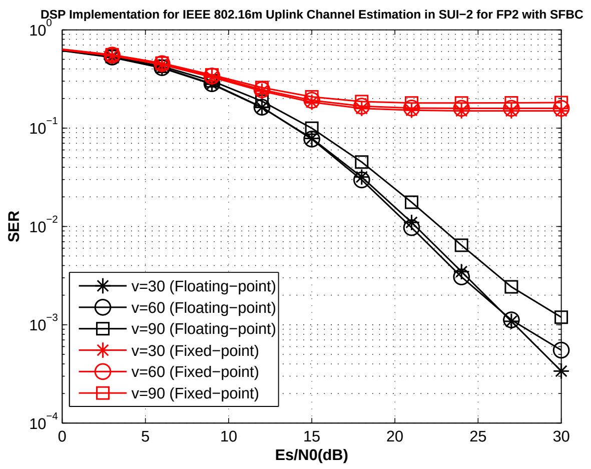
Figure 7.15: Fixed-point channel estimation MSE use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-2 Fixed-point channel for IEEE 802.16m uplink, where the speed v is km/h.



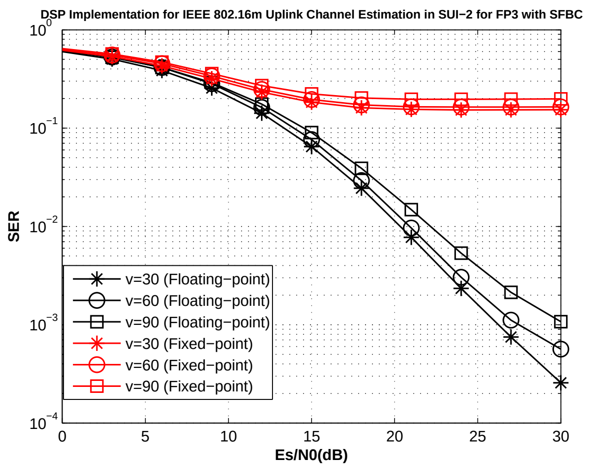
(a)



(b)

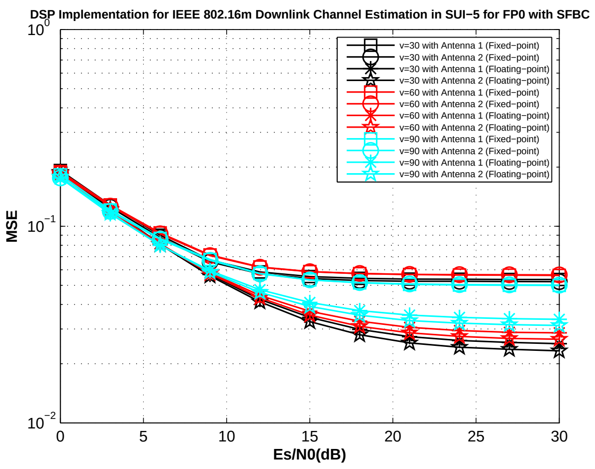


(c)

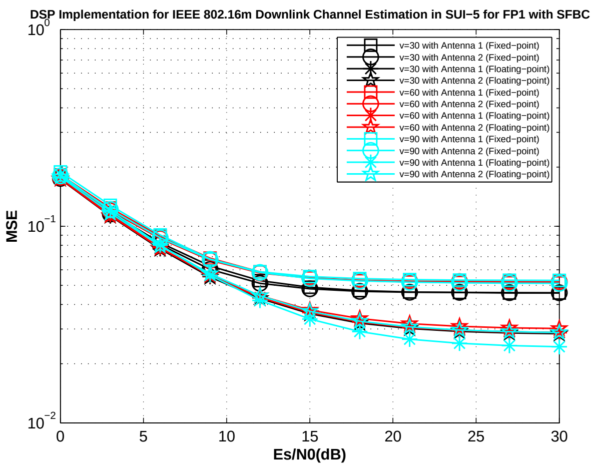


(d)

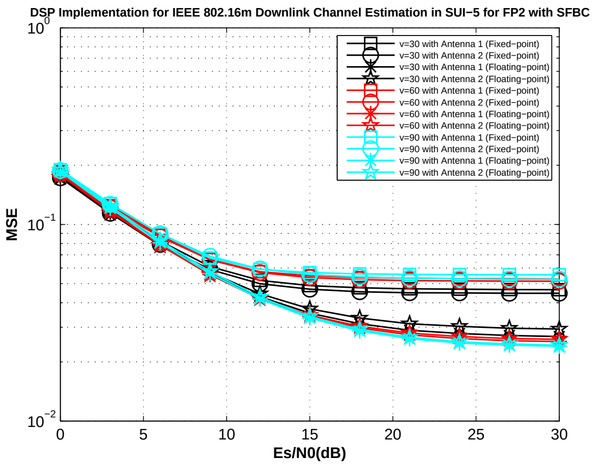
Figure 7.16: Fixed-point channel estimation SER use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-2 Fixed-point channel for IEEE 802.16m uplink, where the speed v is km/h.



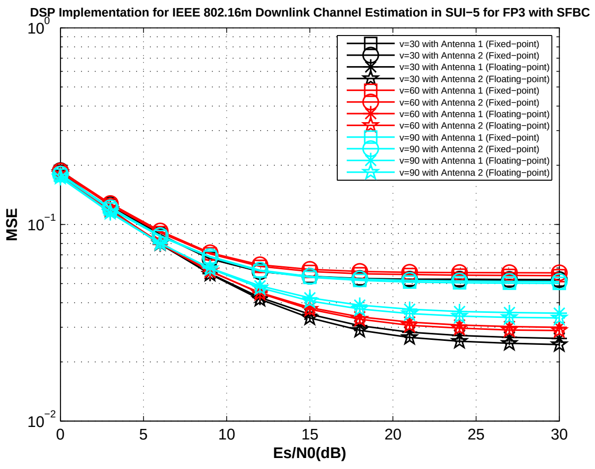
(a)



(b)

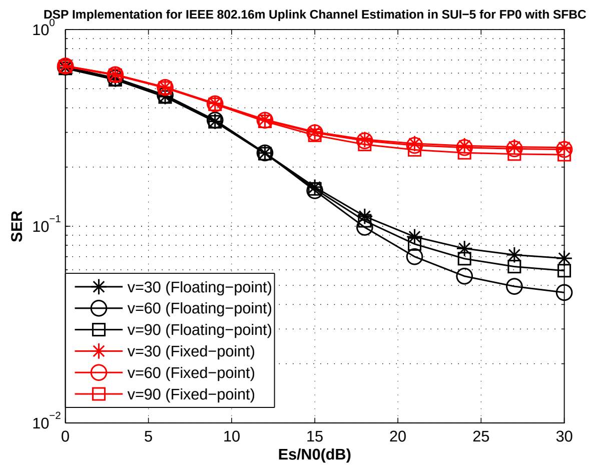


(c)

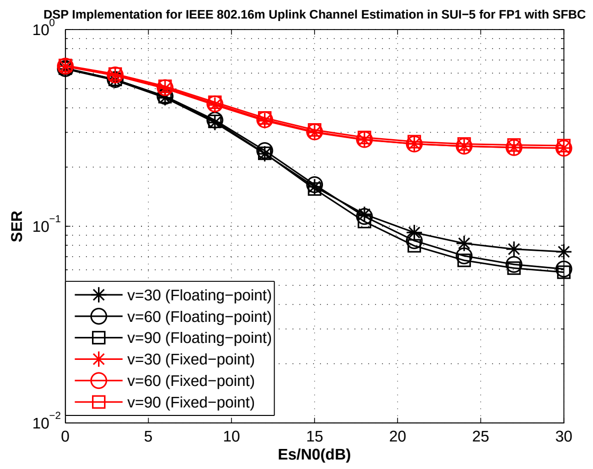


(d)

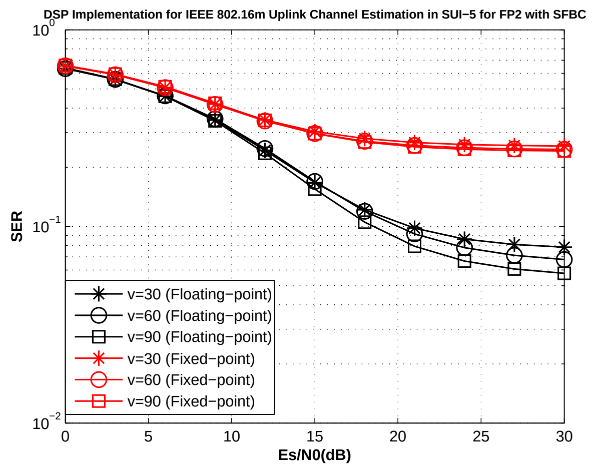
Figure 7.17: Fixed-point channel estimation MSE use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-5 Fixed-point channel for IEEE 802.16m uplink, where the speed v is km/h.



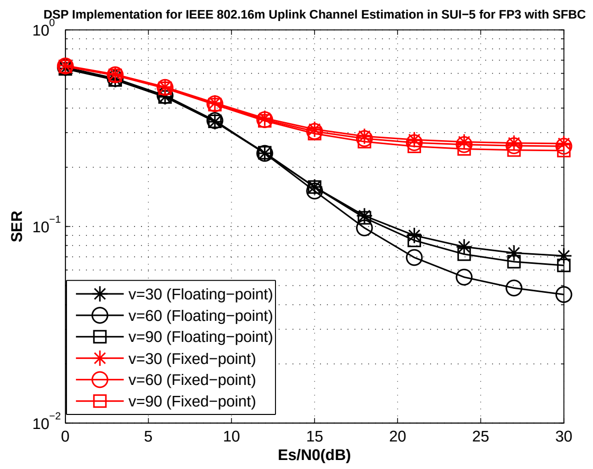
(a)



(b)



(c)



(d)

Figure 7.18: Fixed-point channel estimation SER use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-5 Fixed-point channel for IEEE 802.16m uplink, where the speed v is km/h.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this thesis, first half of that we discussed about the LMMSE channel estimation method for OFDMA downlink and uplink for IEEE 802.16m. We summarize the process to following steps:

- First, use least-square method on pilot position to get each pilot response.
- Second, do linear interpolation in time to get all the pilot response and related equivalent to true pilot position and there referred to as pilot positions..
- Third, use pilot response to calculate mean delay and rms delay.
- Fourth, use above two parameters to calculate correlation function
- Fifth, use correlation function to calculate Wiener filter coefficient.
- Sixth, calculate data response by Wiener filter calculation.

and we extended to multiple antennas with SFBC technique, we validation with C code simulation and compared the result with different channel types.

The last half of thesis we discussed about fixed-point implementation on DSP chip. We integrated the contents to following items:

- We analysis the advantages and defects of fixed-point and floating-point calculation.
- Discussed the importance of scaling factor and propose two different scaling expression for reference.
- The knack of nonlinear function implementation and summarized four steps for fixed-point implementation pre-work.
- We designed a division subfunction to prevent overflow operation and maintain the precision.
- Along the channel estimation process to analysis and optimized the C code.

8.2 Potential Future Work

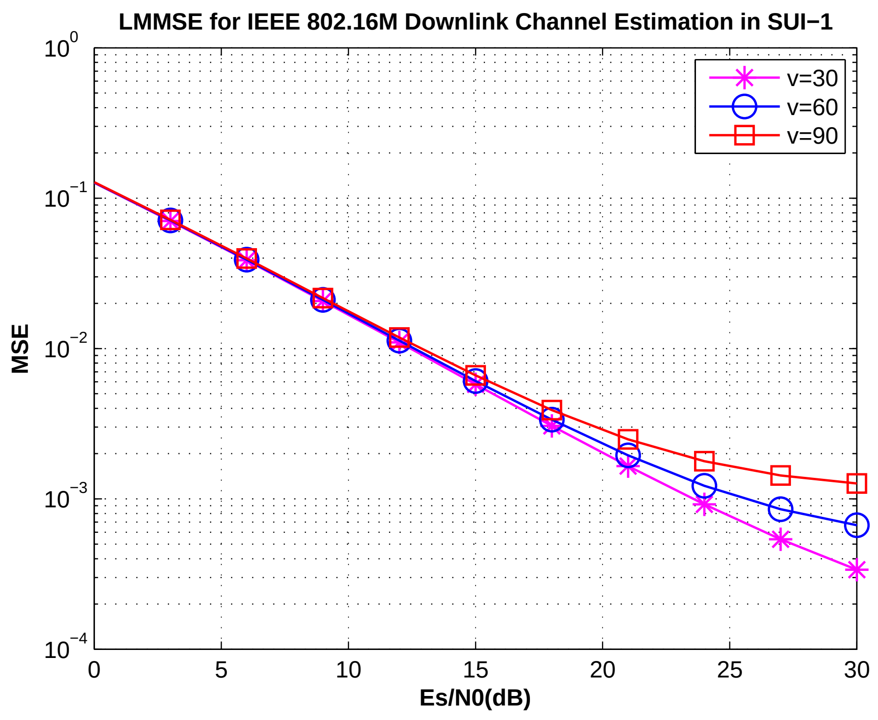
There are several possible extensions for our research:

- Enhance performance in the long delay spread channels.
- Consider the DRU part for transmission in uplink system.
- Solve the error flow of fixed-point implementation.
- Try to simulate high velocity like 240 km/h.
- Use DSP intrinsic function to accelerate C code.
- We do not consider the influence of intercarrier interference in this thesis. The simulation can be involved in the future.

Appendix A

Additional Downlink SISO Simulation Result for Floating-Point

In this appendix, we place additional downlink SISO numerical result used floating-point. Figs. A.1 to A.6 shows the SER and MSE for each SUI channel.



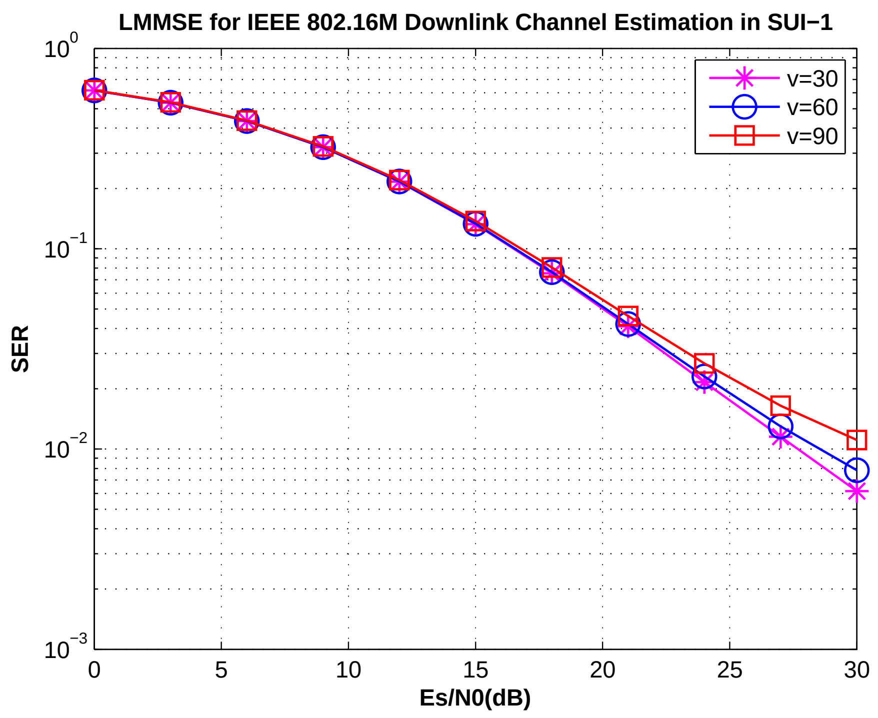
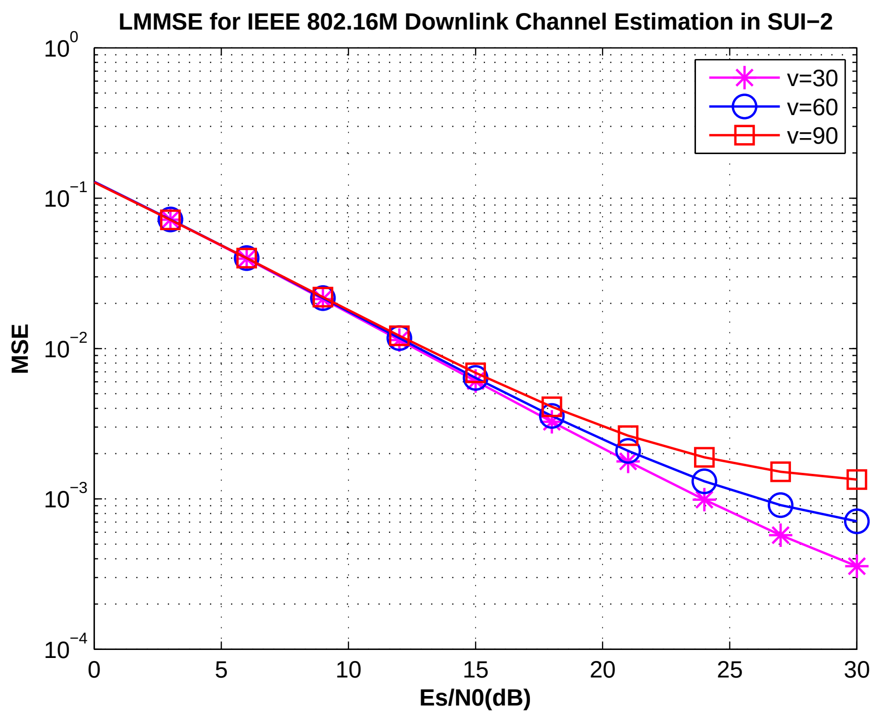


Figure A.1: Channel estimation MSE and SER for QPSK at different velocities in SUI-1 channel for IEEE 802.16m downlink, where the speed v is km/h.



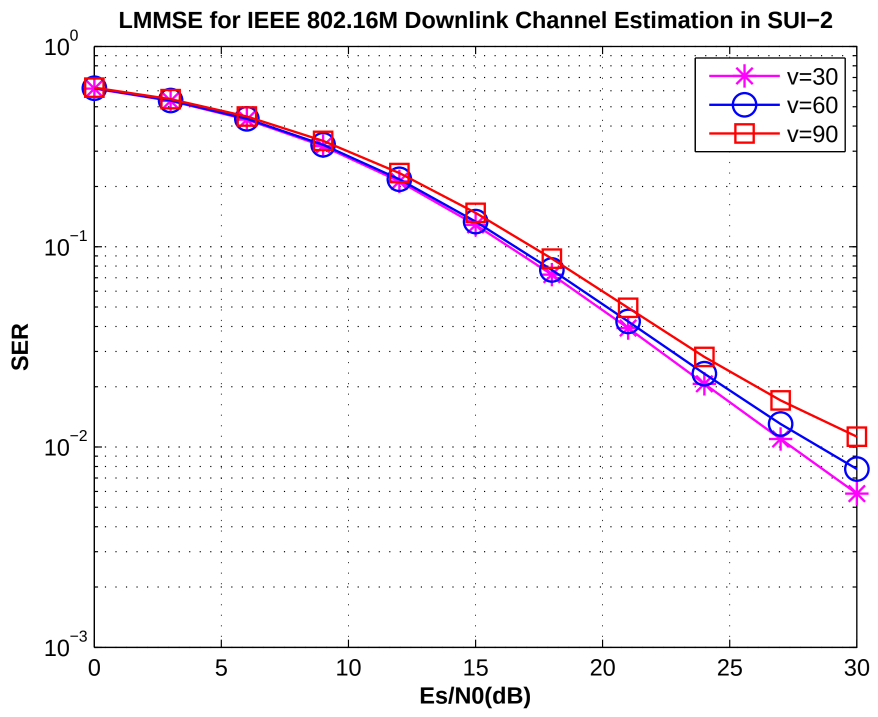
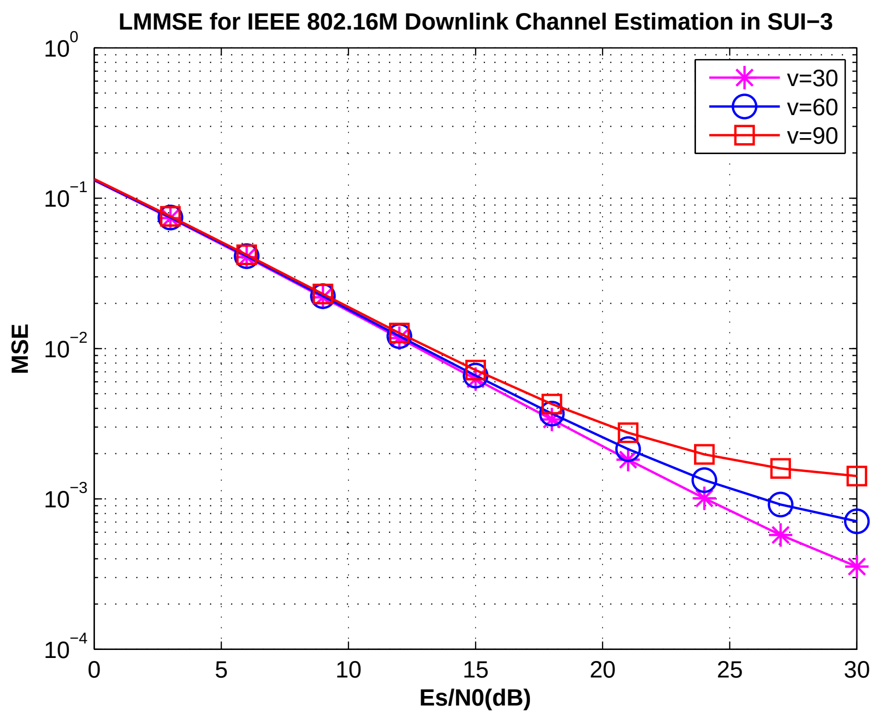


Figure A.2: Channel estimation MSE and SER for QPSK at different velocities in SUI-2 channel for IEEE 802.16m downlink, where the speed v is km/h.



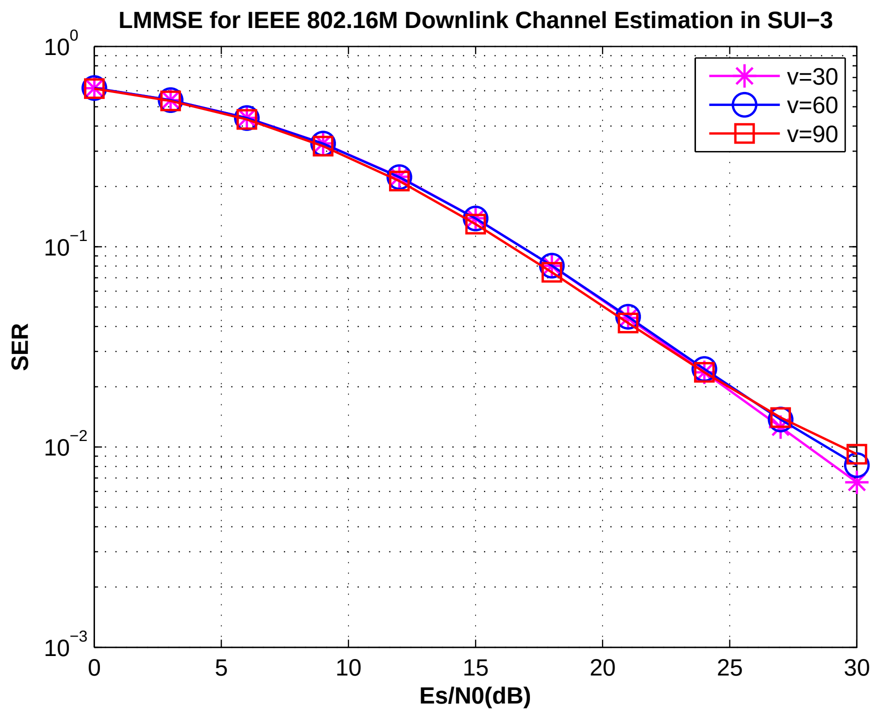
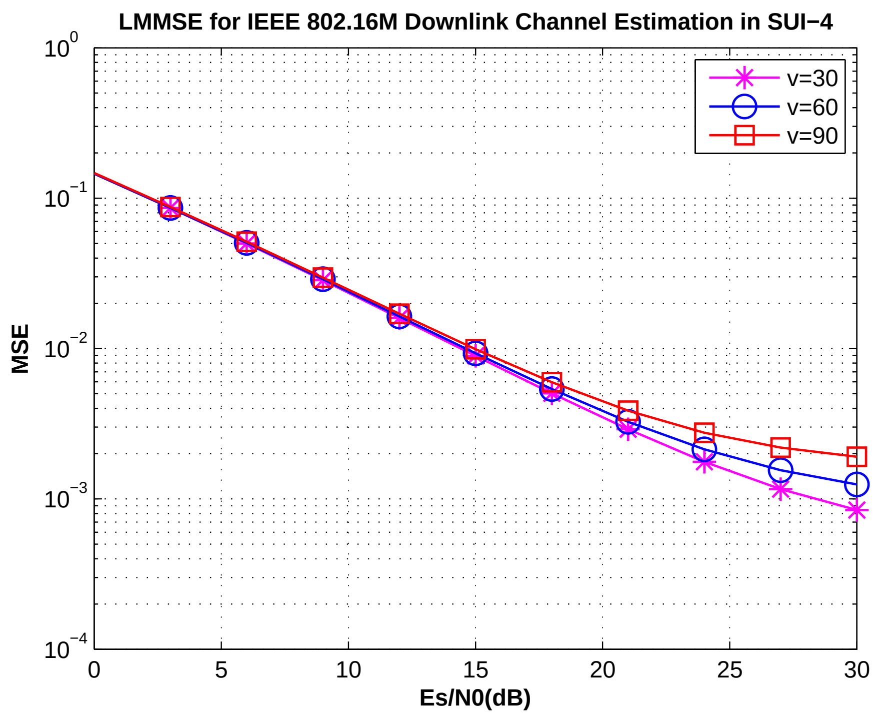


Figure A.3: Channel estimation MSE and SER for QPSK at different velocities in SUI-3 channel for IEEE 802.16m downlink, where the speed v is km/h.



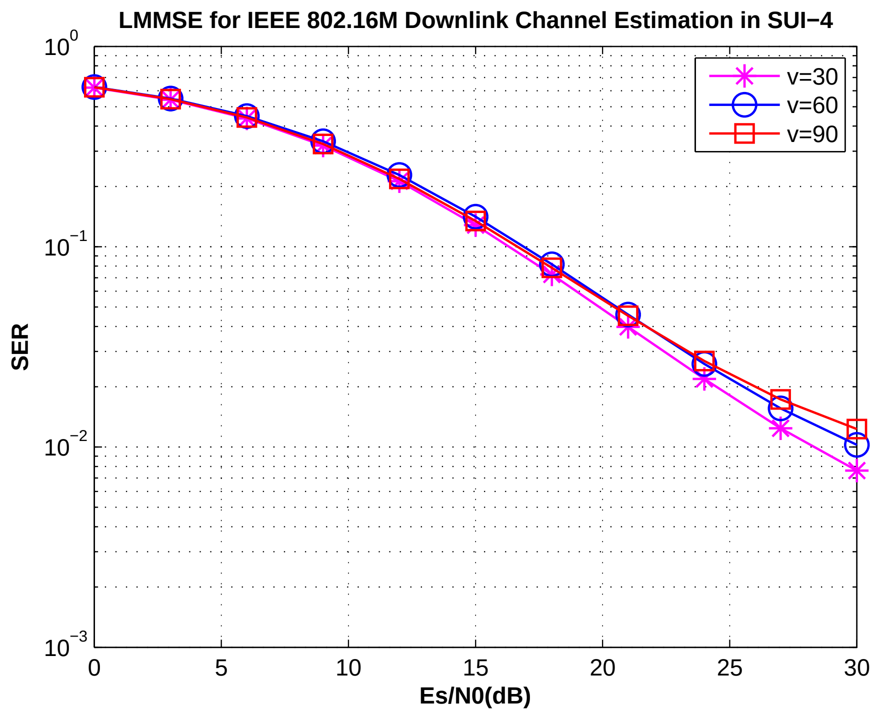
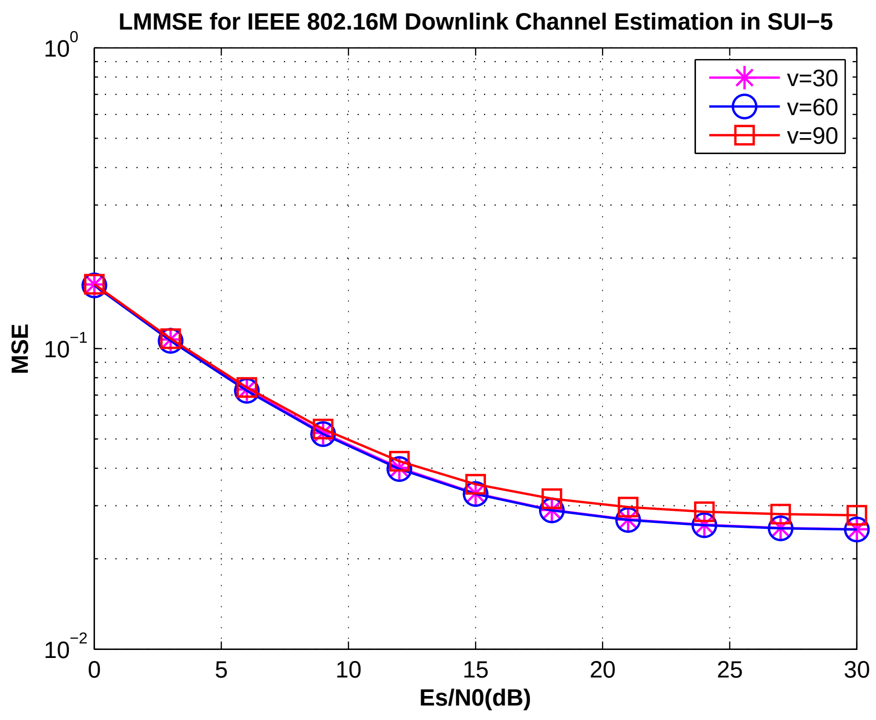


Figure A.4: Channel estimation MSE and SER for QPSK at different velocities in SUI-4 channel for IEEE 802.16m downlink, where the speed v is km/h.



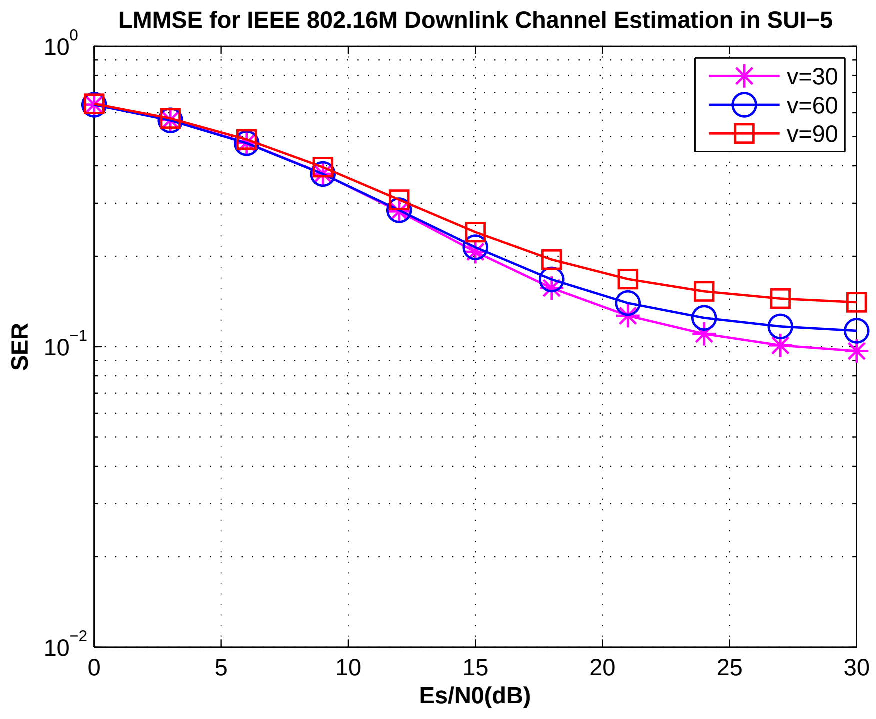
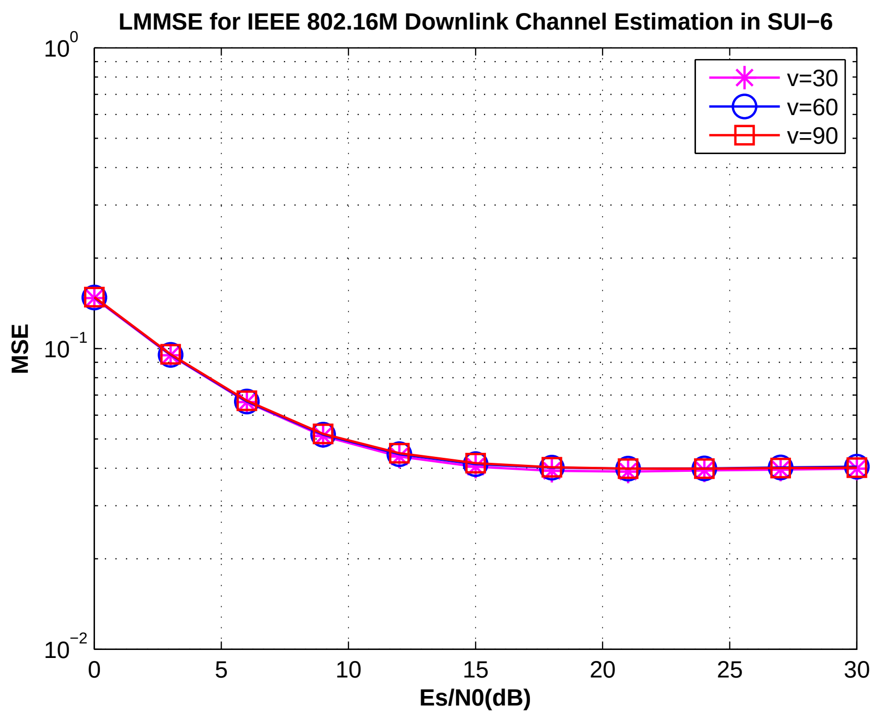


Figure A.5: Channel estimation MSE and SER for QPSK at different velocities in SUI-5 channel for IEEE 802.16m downlink, where the speed v is km/h.



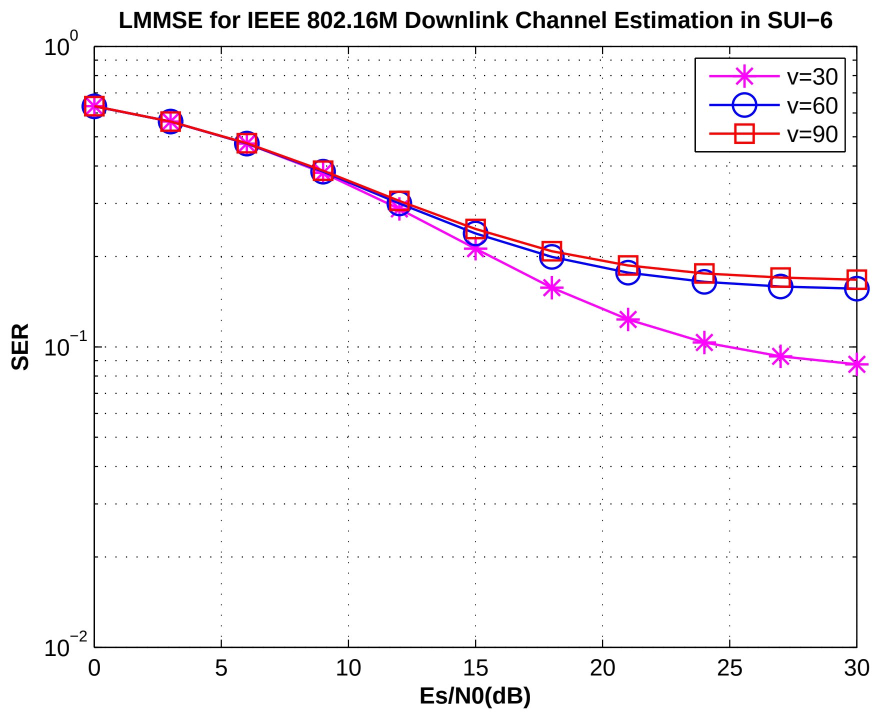
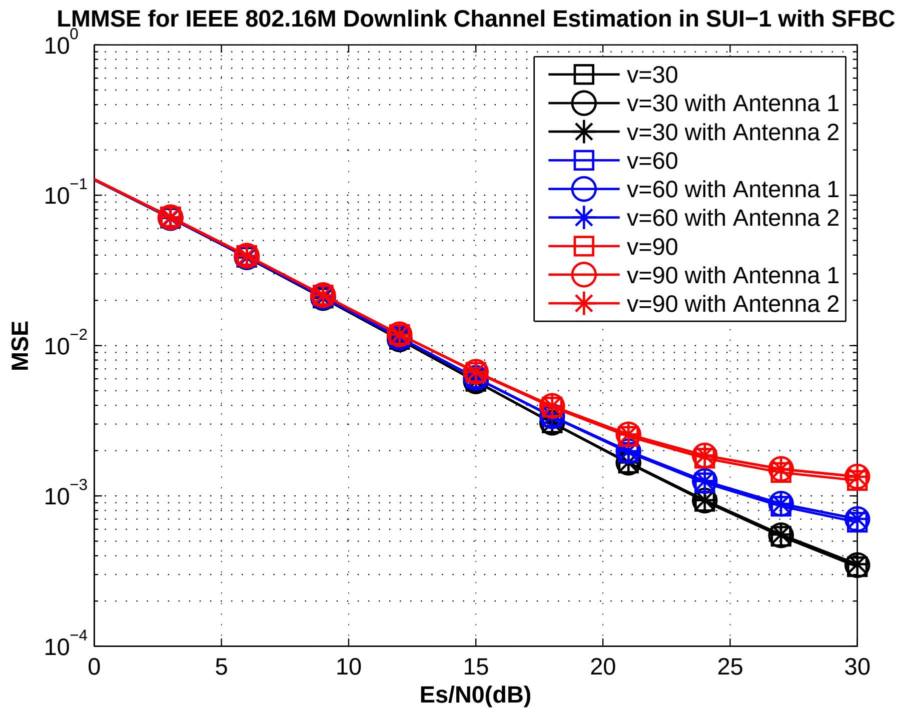


Figure A.6: Channel estimation MSE and SER for QPSK at different velocities in SUI-6 channel for IEEE 802.16m downlink, where the speed v is km/h.

Appendix B

Additional Downlink SFBC MIMO Simulation Result for Floating-Point

In this appendix, we place additional downlink SFBC MIMO numerical result used floating-point. Figs. B.1 to B.6 shows the SER and MSE for each SUI channel.



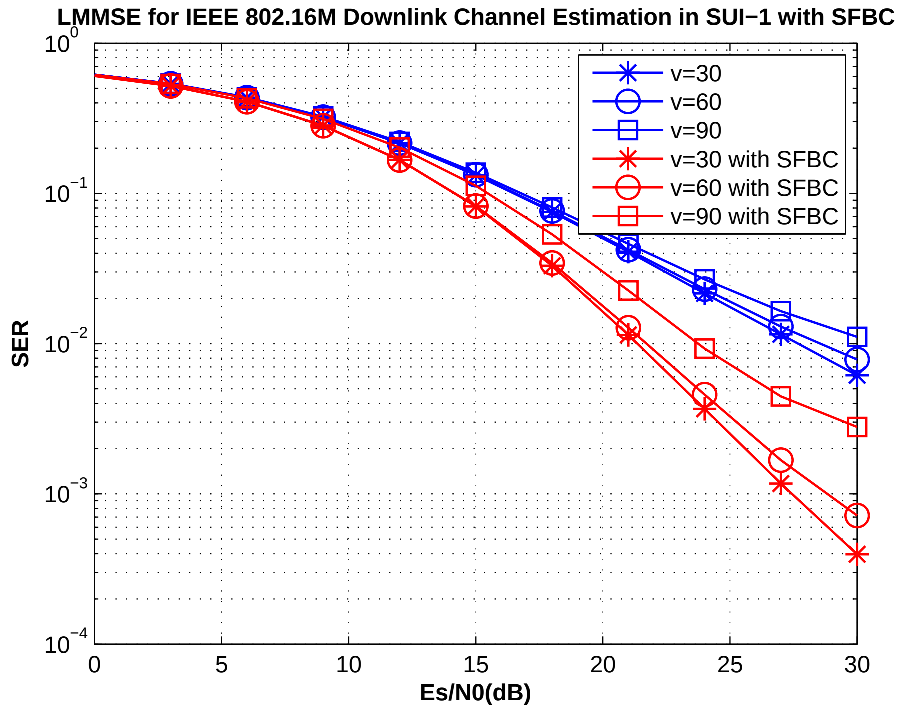
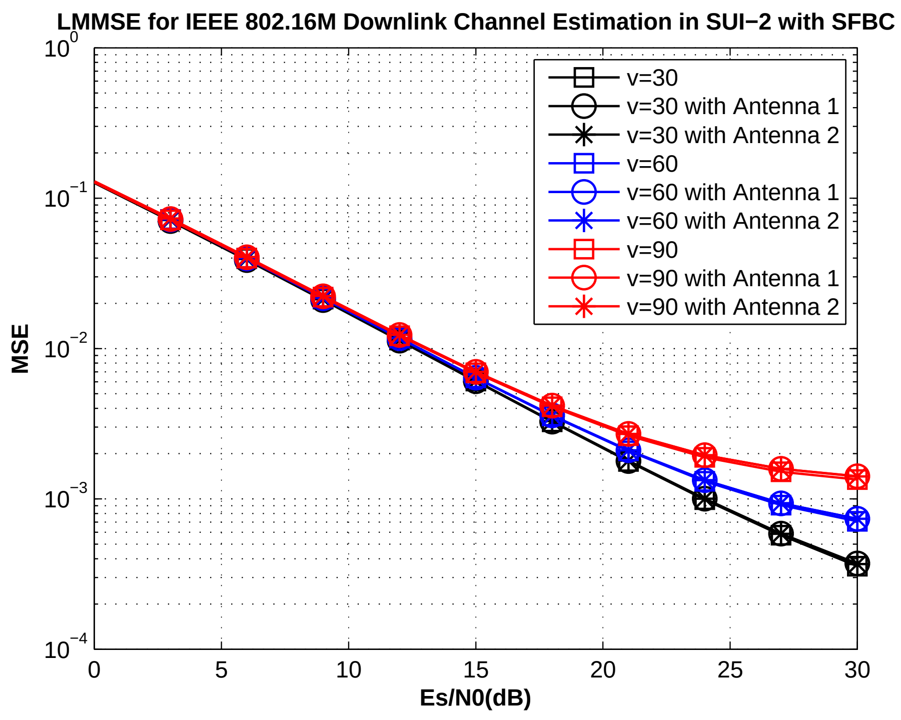


Figure B.1: Channel estimation MSE and SER for QPSK with SFBC at different velocities in SUI-1 channel for IEEE 802.16m downlink, where the speed v is km/h.



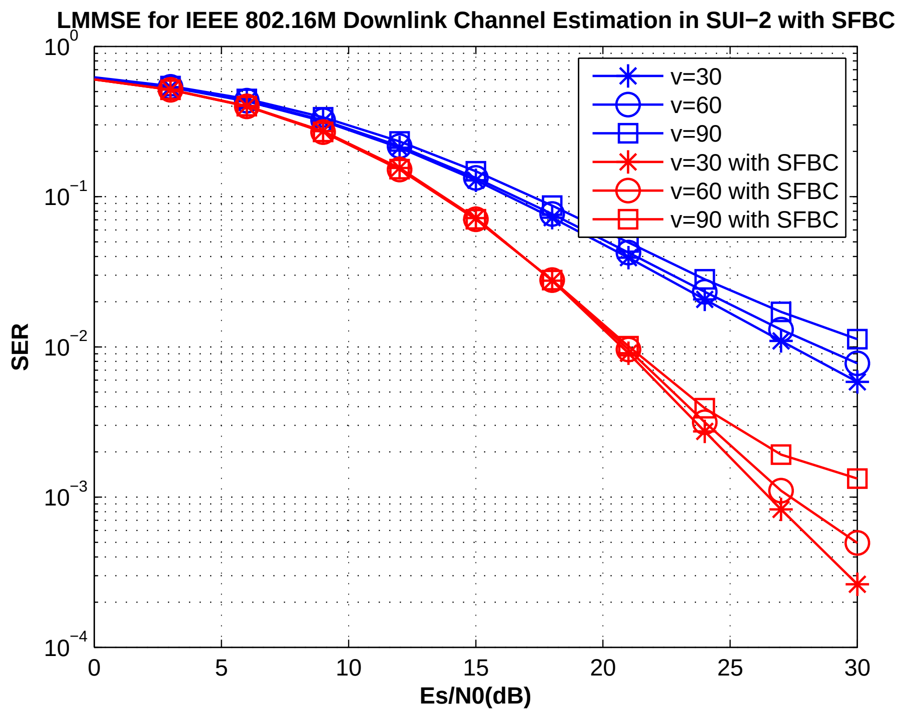
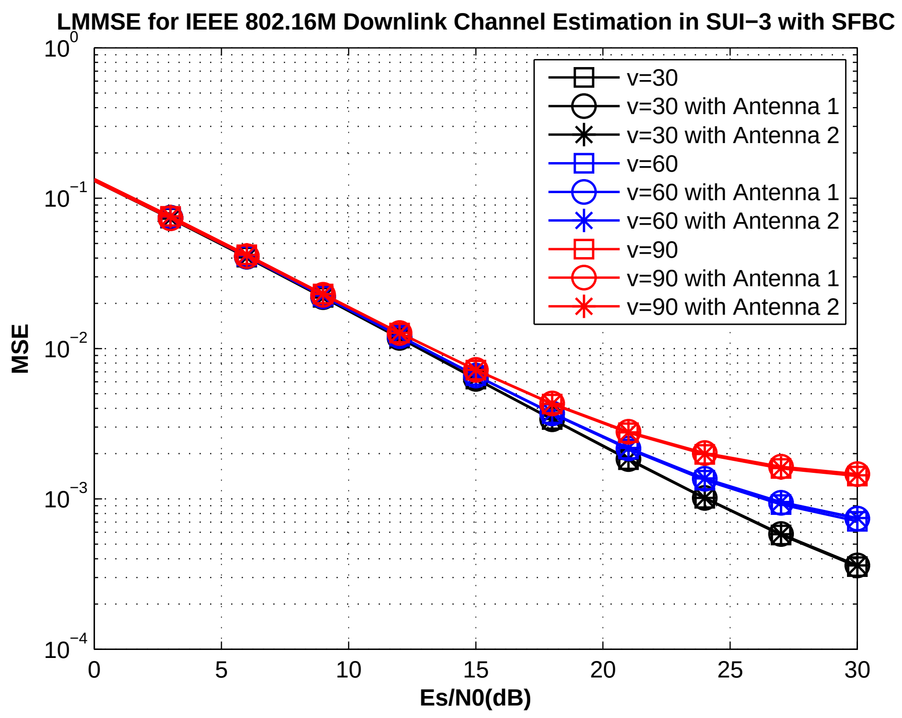


Figure B.2: Channel estimation MSE and SER for QPSK with SFBC at different velocities in SUI-2 channel for IEEE 802.16m downlink, where the speed v is km/h.



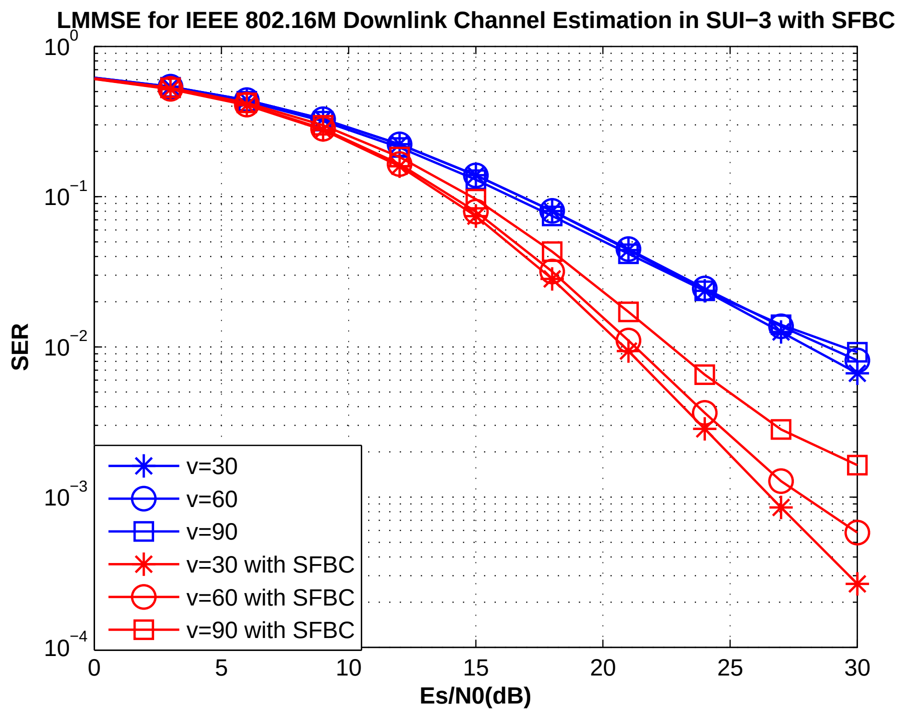
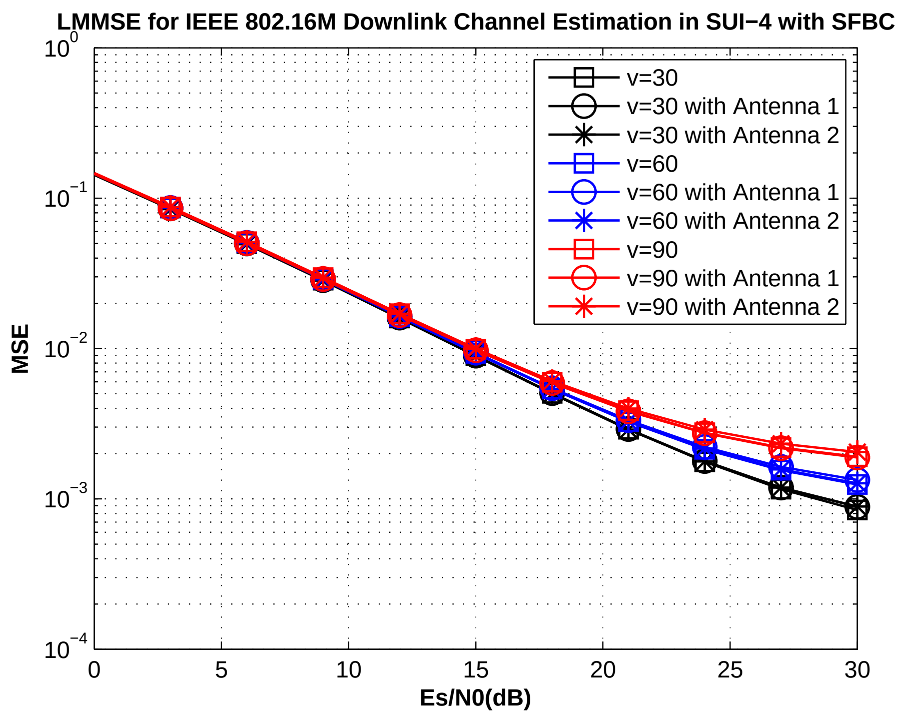


Figure B.3: Channel estimation MSE and SER for QPSK with SFBC at different velocities in SUI-3 channel for IEEE 802.16m downlink, where the speed v is km/h.



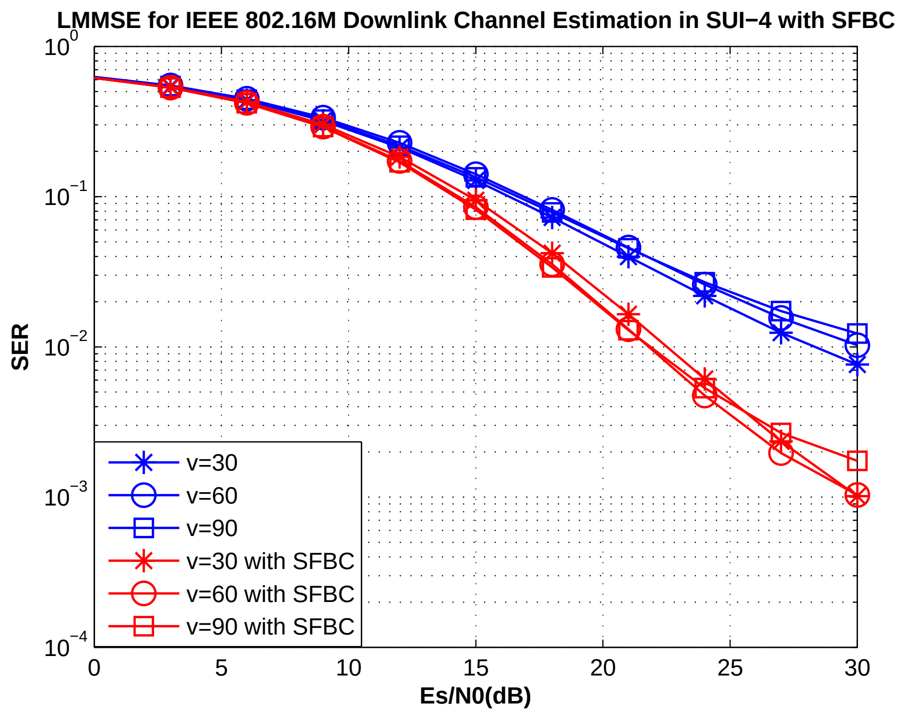
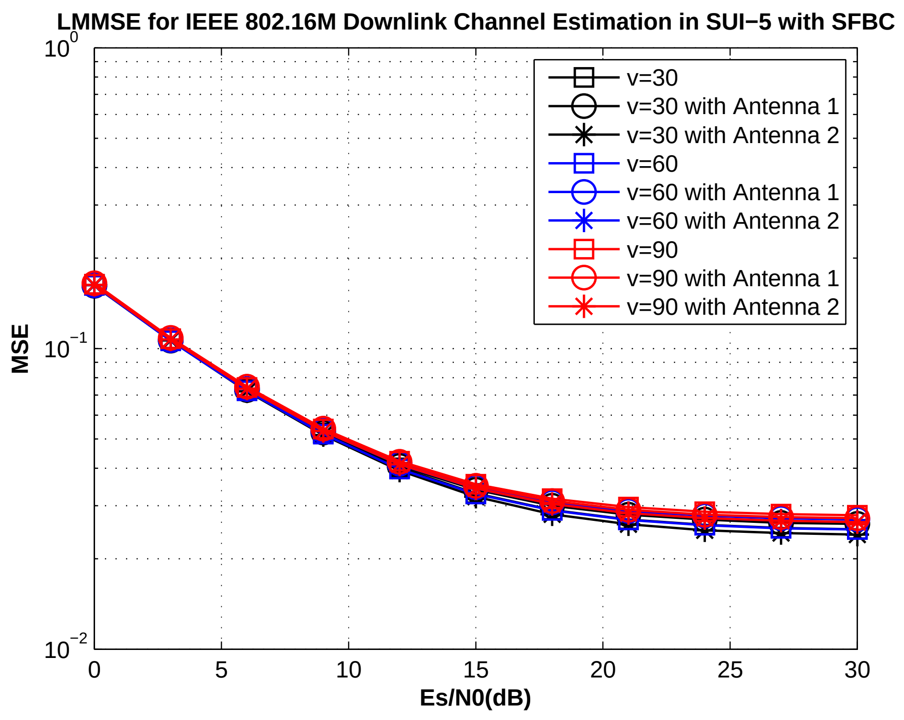


Figure B.4: Channel estimation MSE and SER for QPSK with SFBC at different velocities in SUI-4 channel for IEEE 802.16m downlink, where the speed v is km/h.



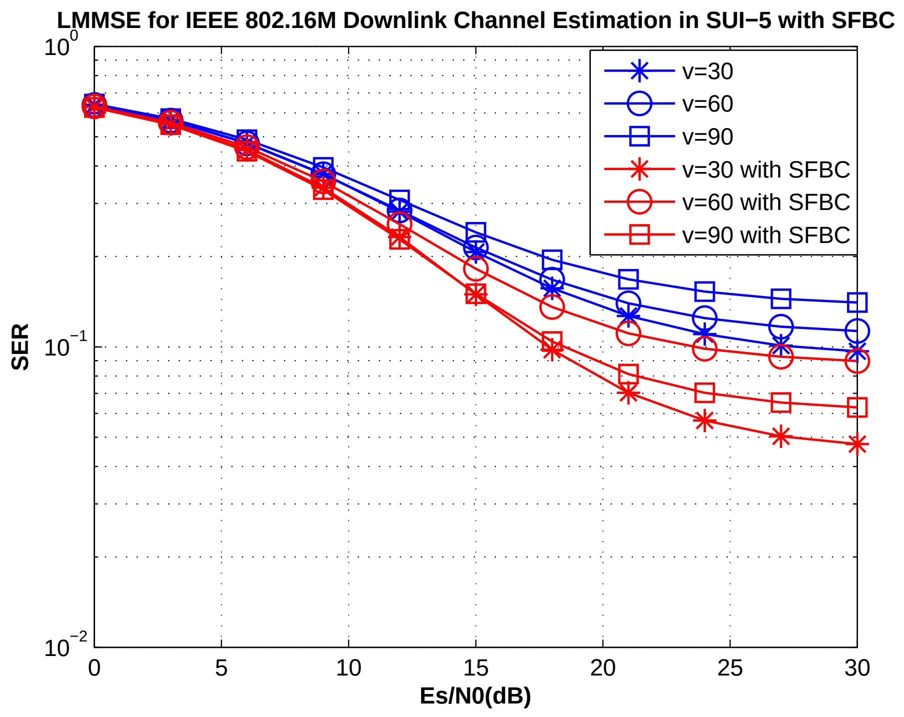
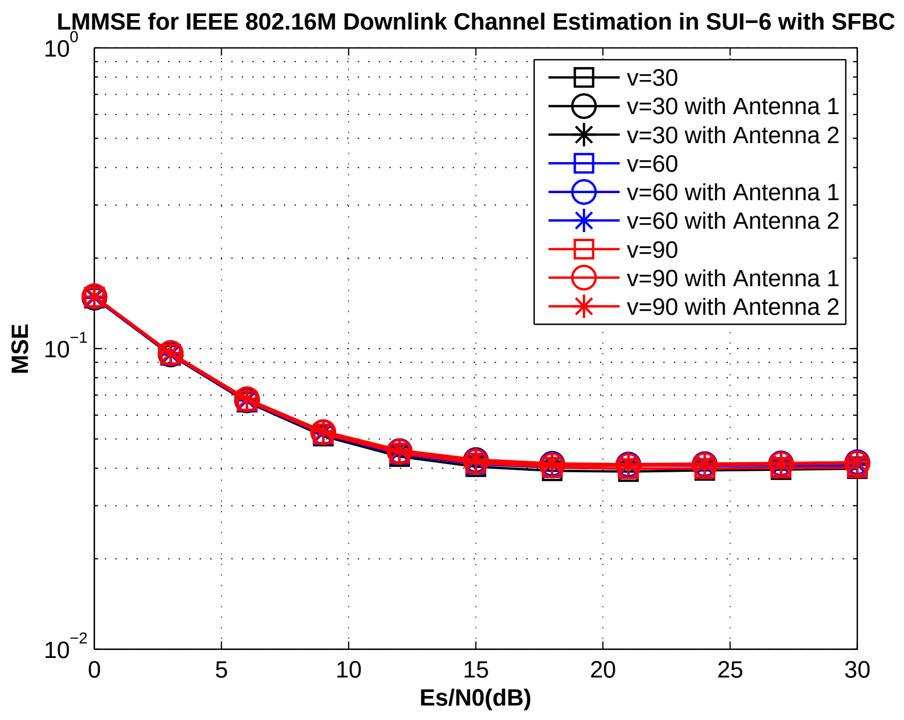


Figure B.5: Channel estimation MSE and SER for QPSK with SFBC at different velocities in SUI-5 channel for IEEE 802.16m downlink, where the speed v is km/h.



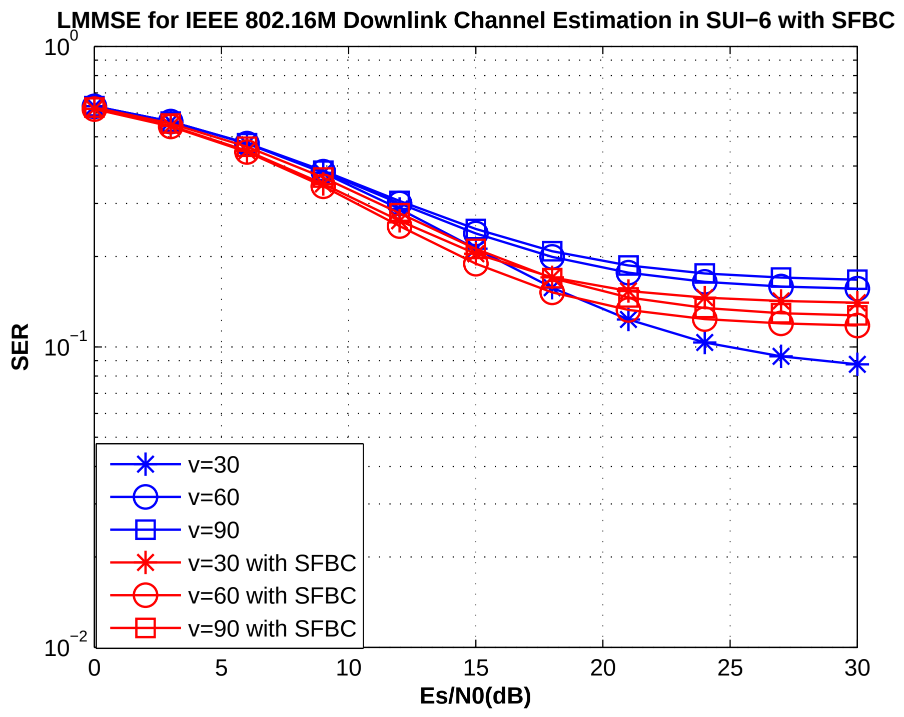
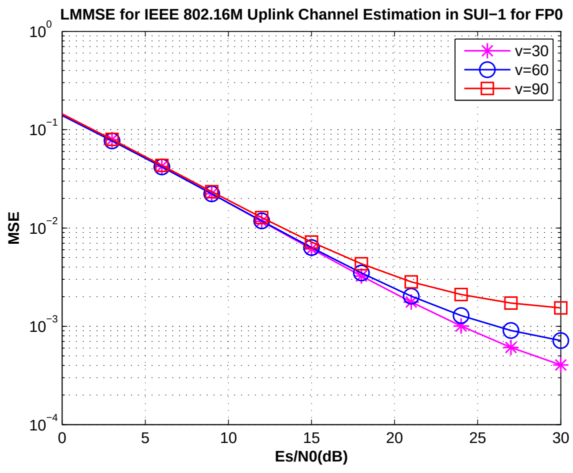


Figure B.6: Channel estimation MSE and SER for QPSK with SFBC at different velocities in SUI-6 channel for IEEE 802.16m downlink, where the speed v is km/h.

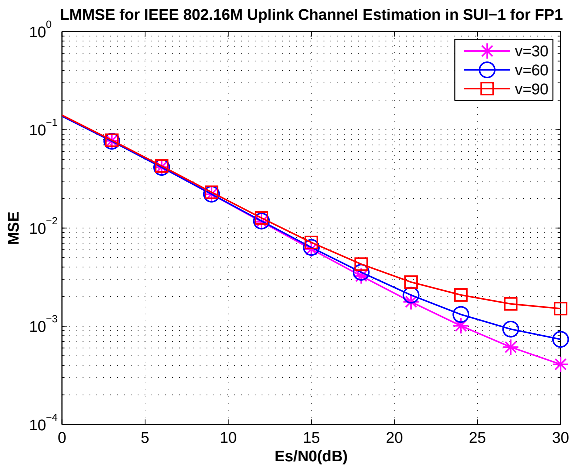
Appendix C

Additional Uplink SISO Simulation Result for Floating-Point

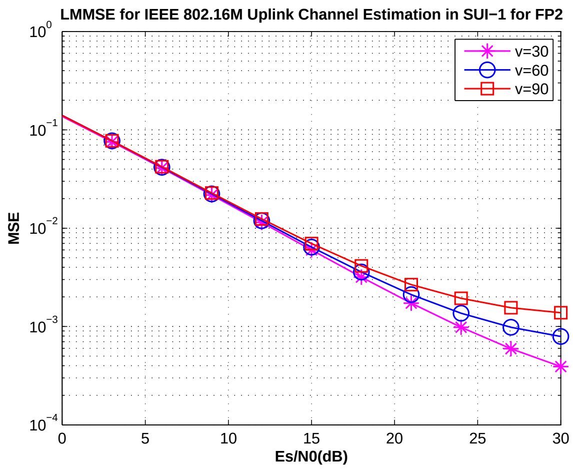
In this appendix, we place additional uplink SISO numerical result used floating-point. Figs. C.1 to C.12 shows the SER and MSE for each SUI channel.



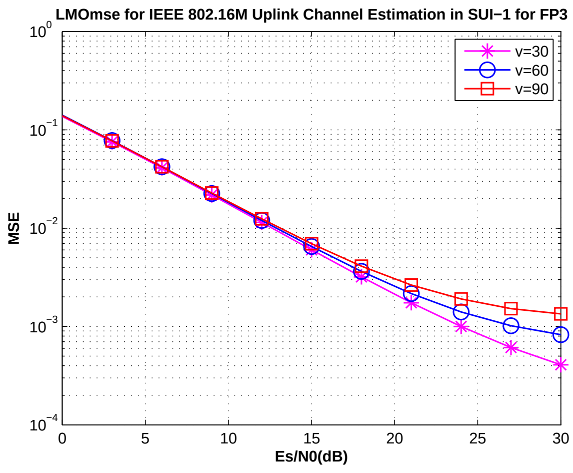
(a)



(b)

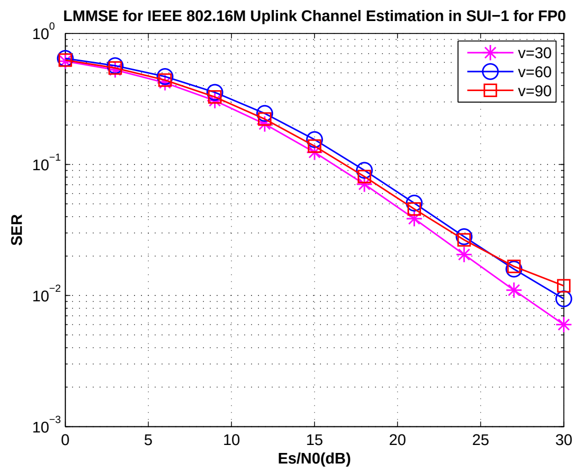


(c)

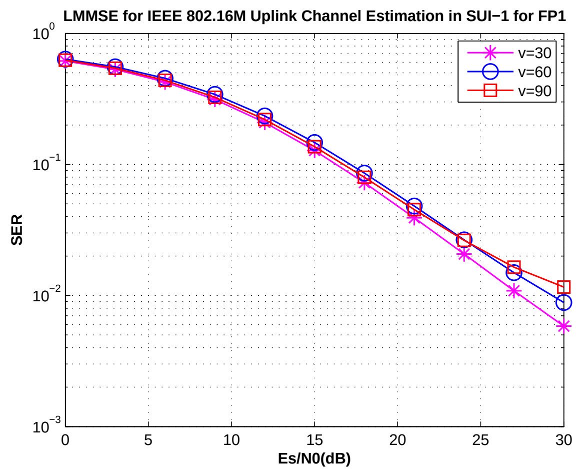


(d)

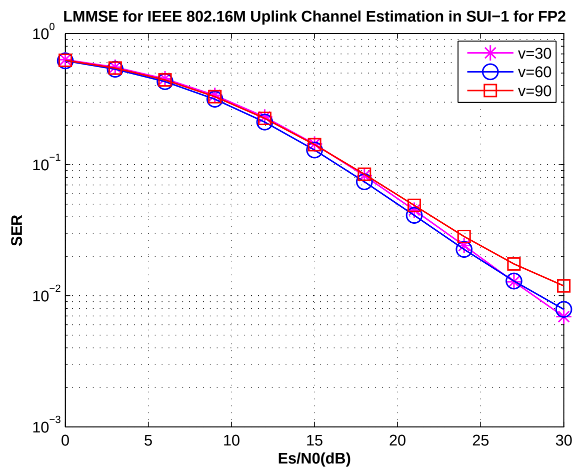
Figure C.1: Channel estimation MSE use QPSK modulation for different frequency partitions at different velocities in SUI-1 channel for IEEE 802.16m uplink, where the speed v is km/h.



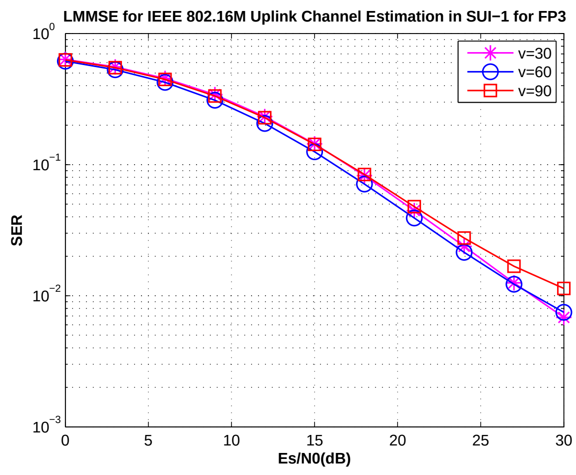
(a)



(b)

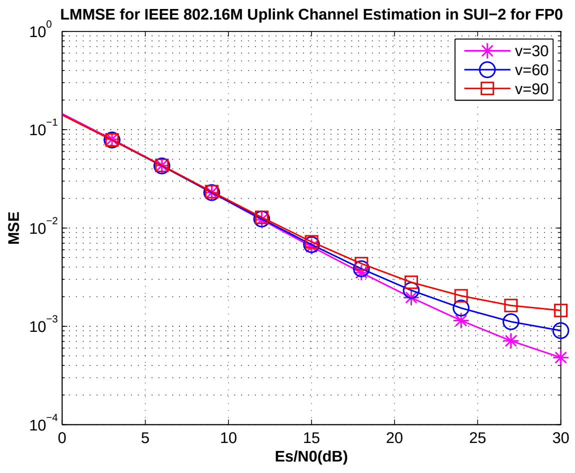


(c)

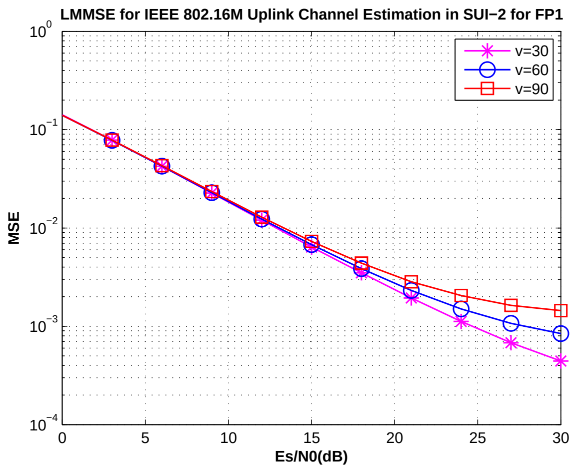


(d)

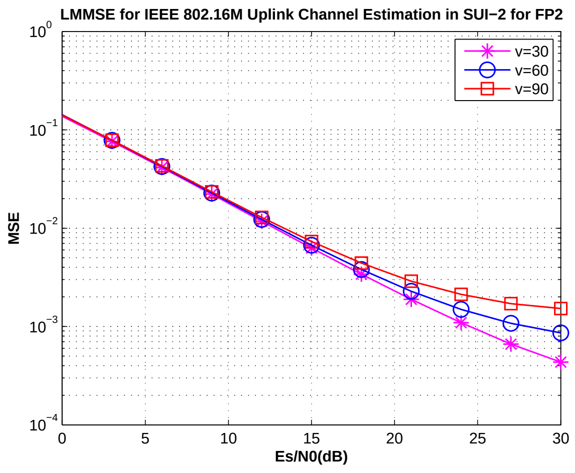
Figure C.2: Channel estimation SER use QPSK modulation for different frequency partitions at different velocities in SUI-1 channel for IEEE 802.16m uplink, where the speed v is km/h.



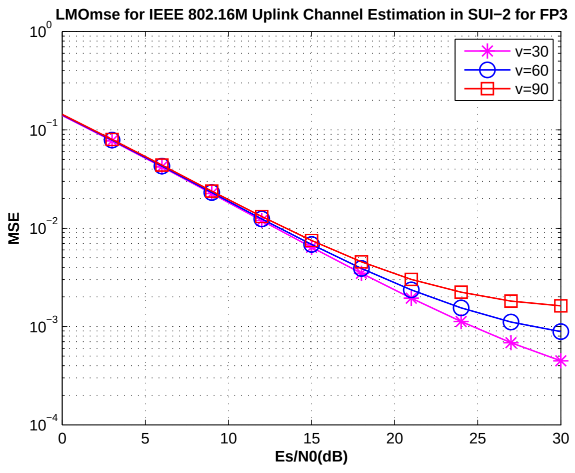
(a)



(b)

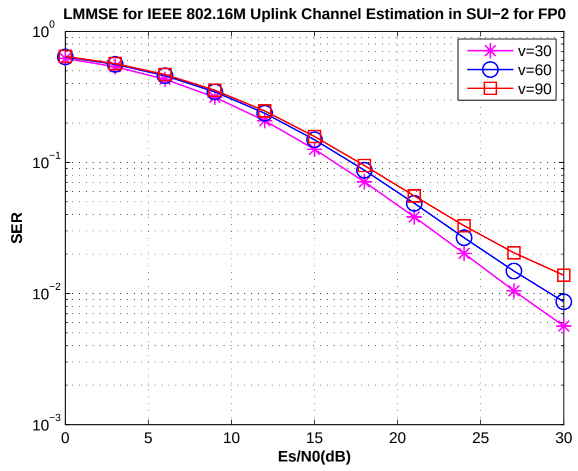


(c)

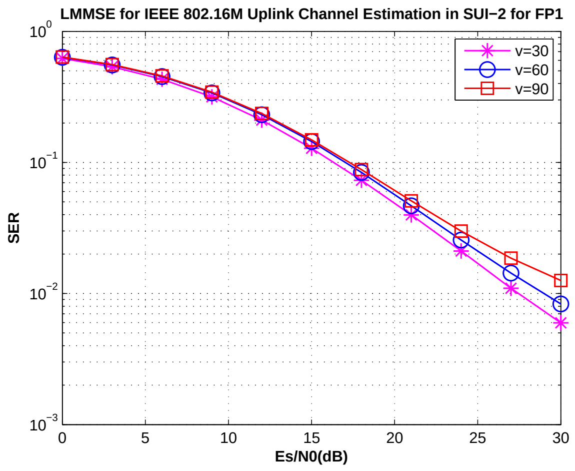


(d)

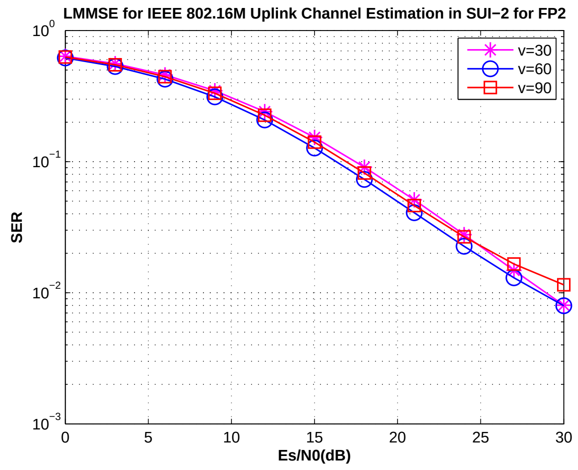
Figure C.3: Channel estimation MSE use QPSK modulation for different frequency partitions at different velocities in SUI-2 channel for IEEE 802.16m uplink, where the speed v is km/h.



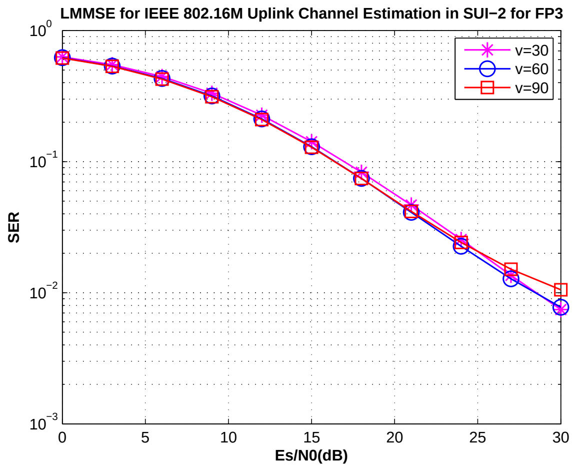
(a)



(b)

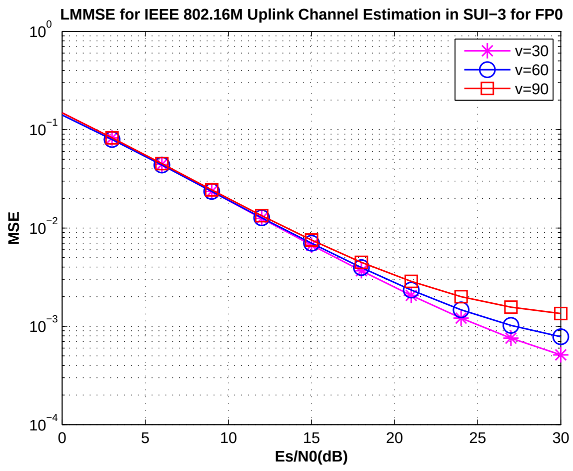


(c)

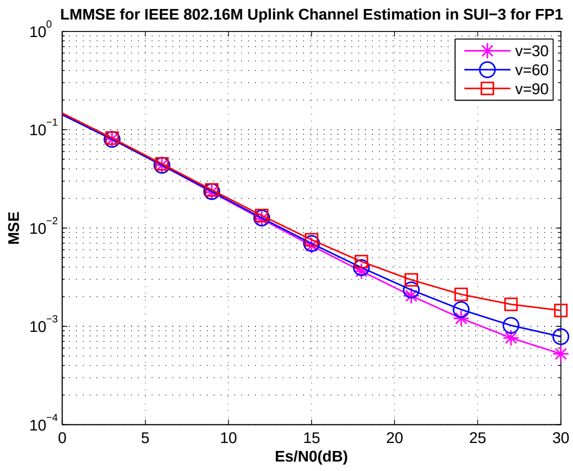


(d)

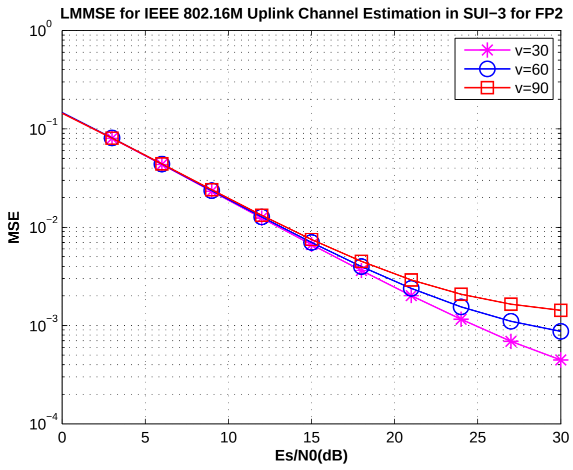
Figure C.4: Channel estimation SER use QPSK modulation for different frequency partitions at different velocities in SUI-2 channel for IEEE 802.16m uplink, where the speed v is km/h.



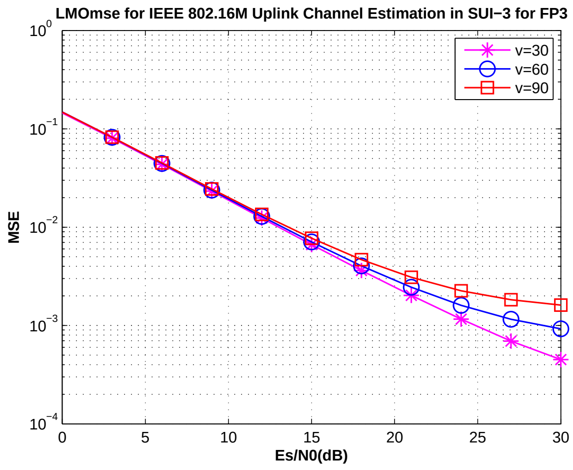
(a)



(b)

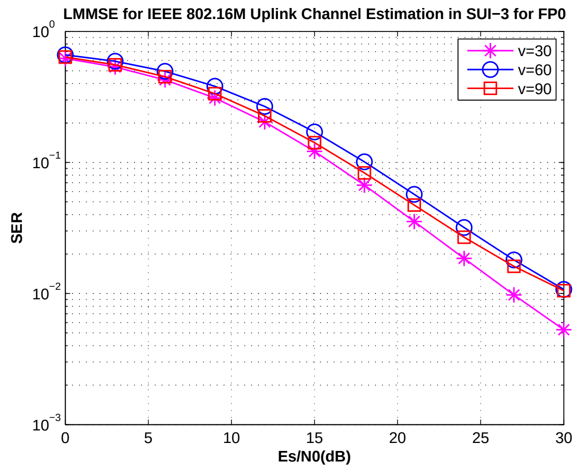


(c)

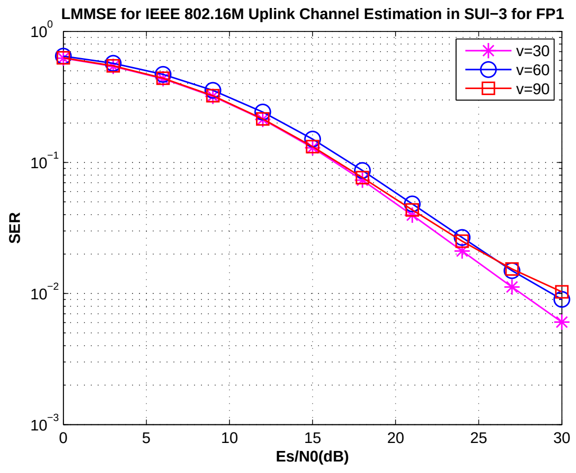


(d)

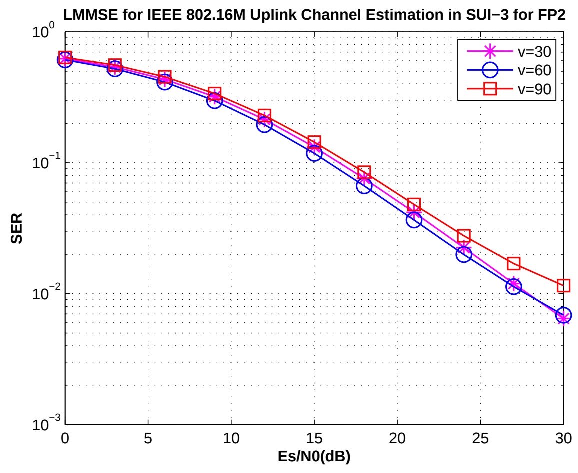
Figure C.5: Channel estimation MSE use QPSK modulation for different frequency partitions at different velocities in SUI-3 channel for IEEE 802.16m uplink, where the speed v is km/h.



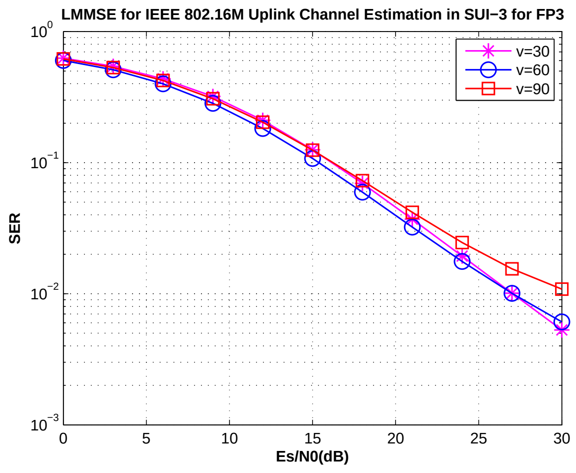
(a)



(b)

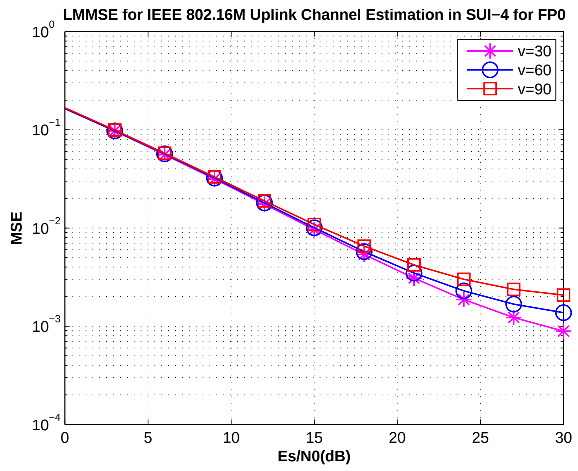


(c)

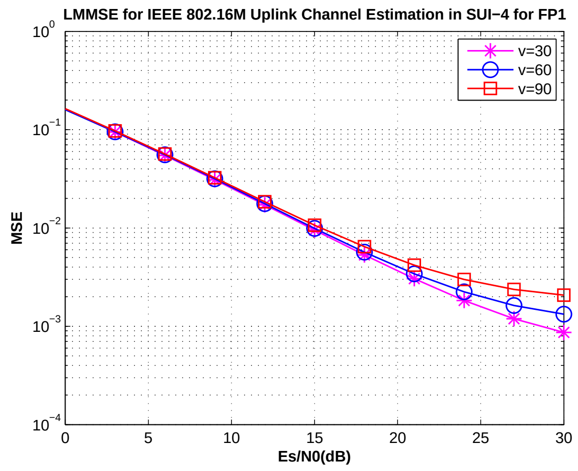


(d)

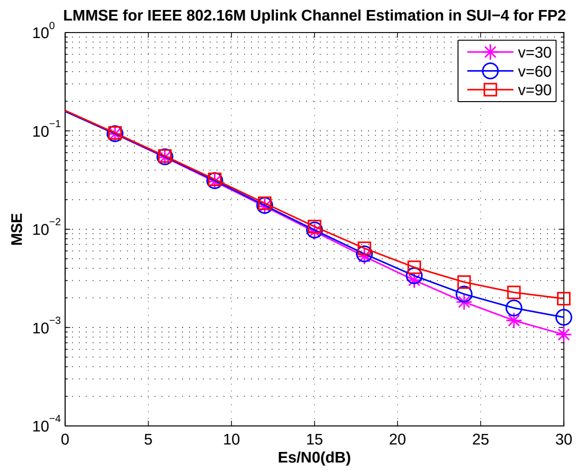
Figure C.6: Channel estimation SER use QPSK modulation for different frequency partitions at different velocities in SUI-3 channel for IEEE 802.16m uplink, where the speed v is km/h.



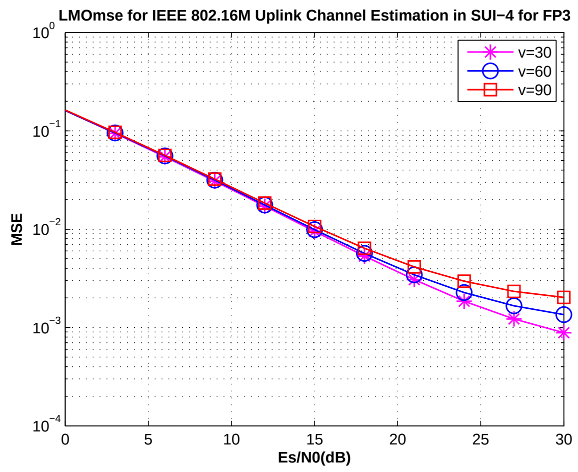
(a)



(b)

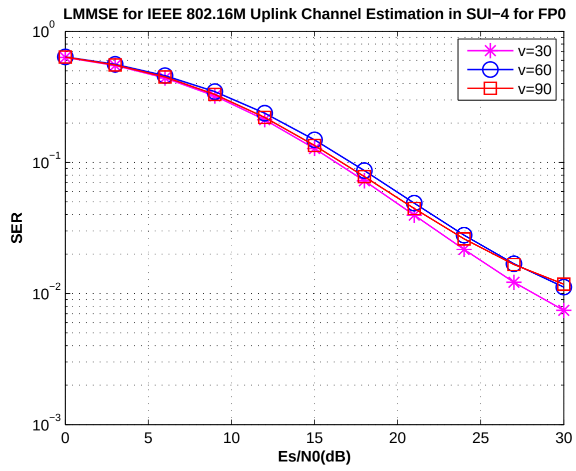


(c)

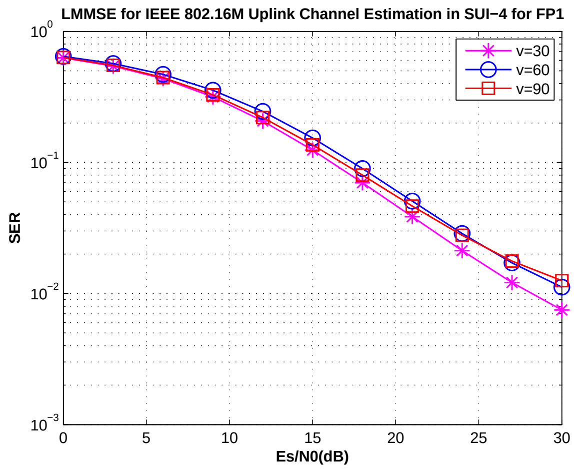


(d)

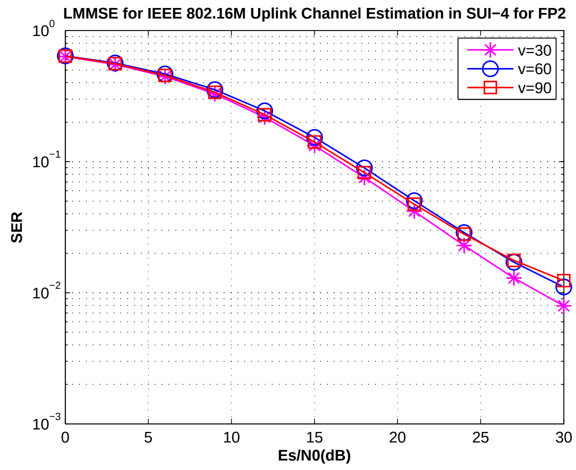
Figure C.7: Channel estimation MSE use QPSK modulation for different frequency partitions at different velocities in SUI-4 channel for IEEE 802.16m uplink, where the speed v is km/h.



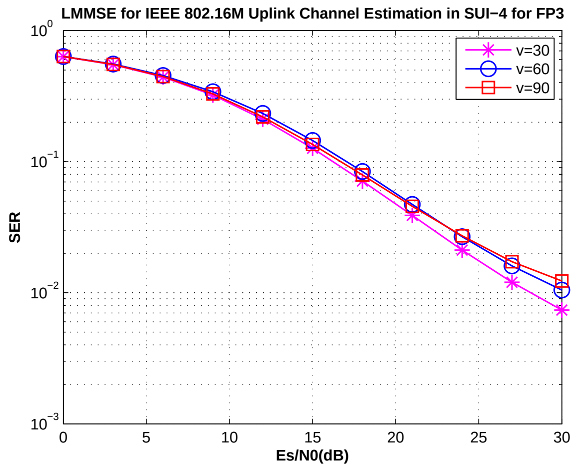
(a)



(b)

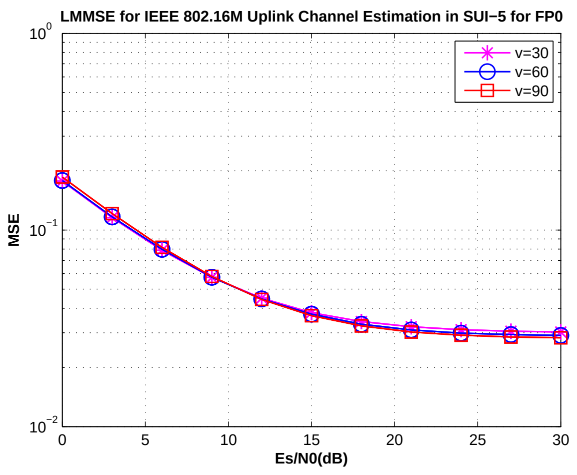


(c)

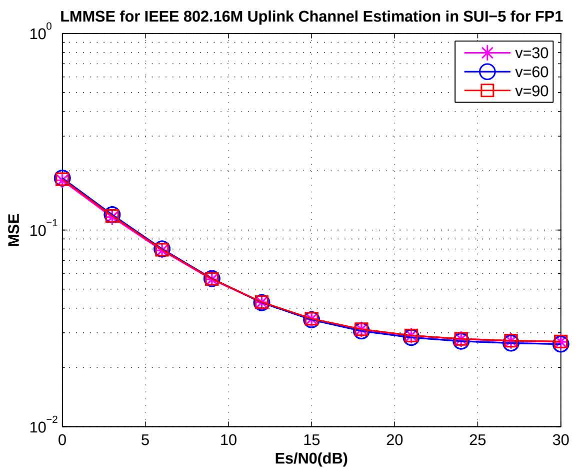


(d)

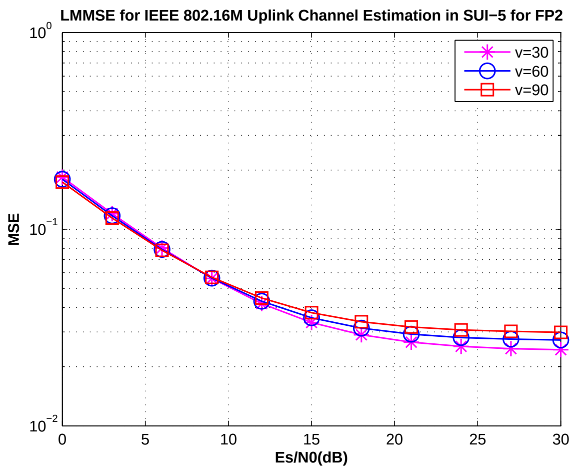
Figure C.8: Channel estimation SER use QPSK modulation for different frequency partitions at different velocities in SUI-4 channel for IEEE 802.16m uplink, where the speed v is km/h.



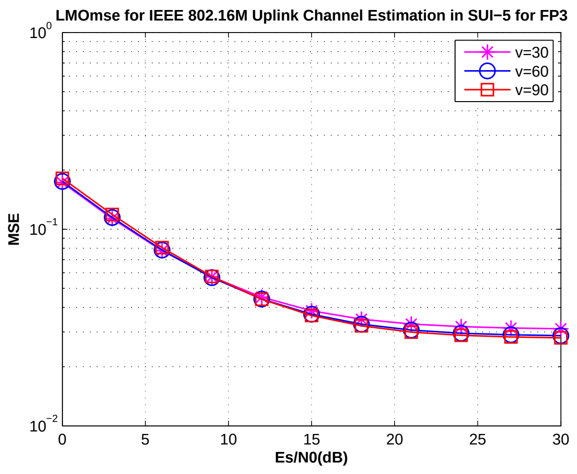
(a)



(b)

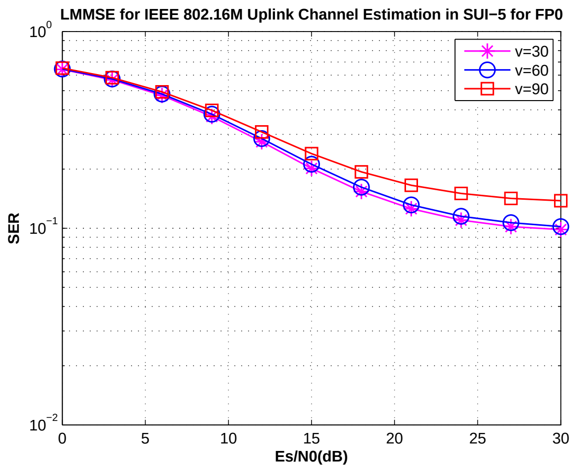


(c)

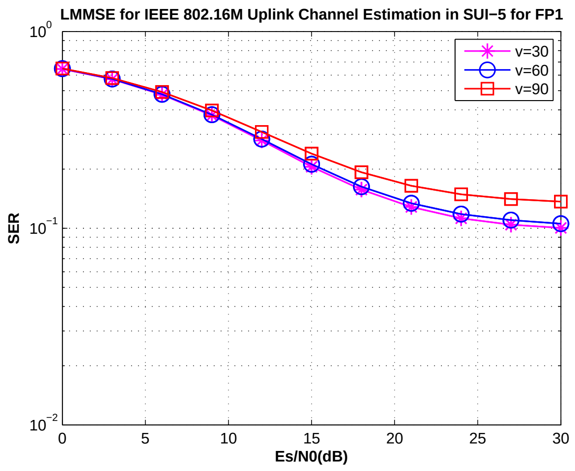


(d)

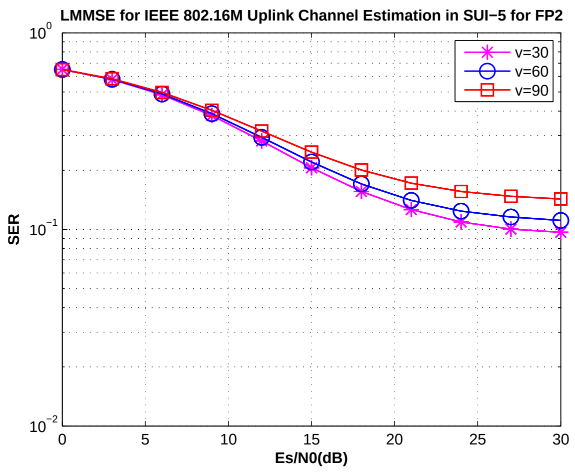
Figure C.9: Channel estimation MSE use QPSK modulation for different frequency partitions at different velocities in SUI-5 channel for IEEE 802.16m uplink, where the speed v is km/h.



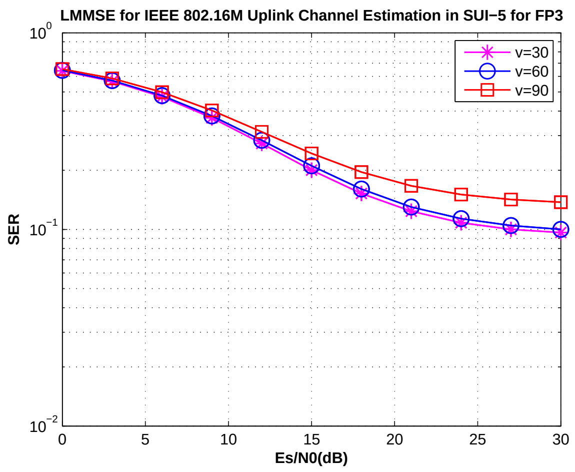
(a)



(b)

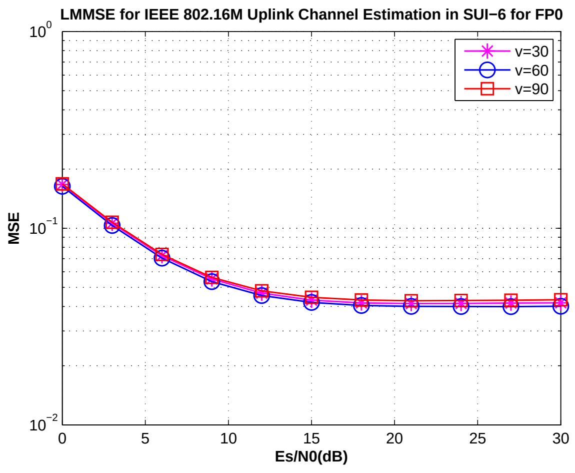


(c)

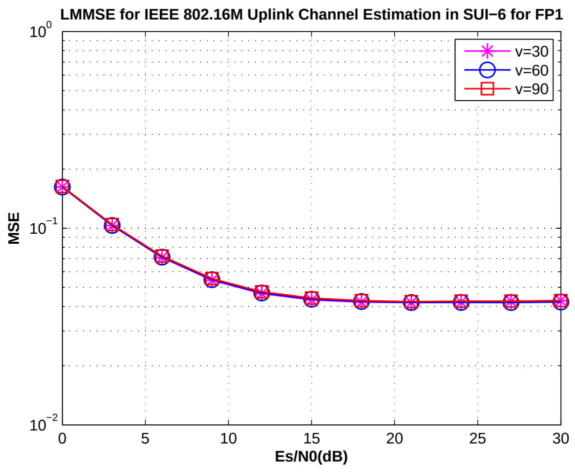


(d)

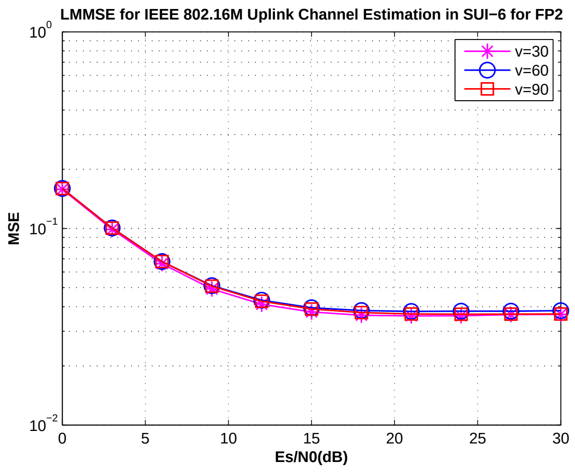
Figure C.10: Channel estimation SER use QPSK modulation for different frequency partitions at different velocities in SUI-5 channel for IEEE 802.16m uplink, where the speed v is km/h.



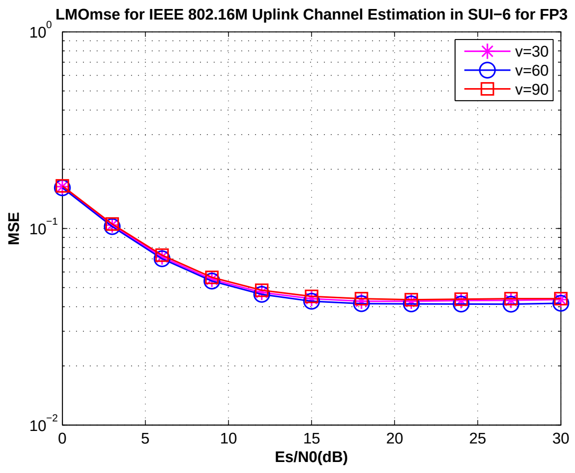
(a)



(b)

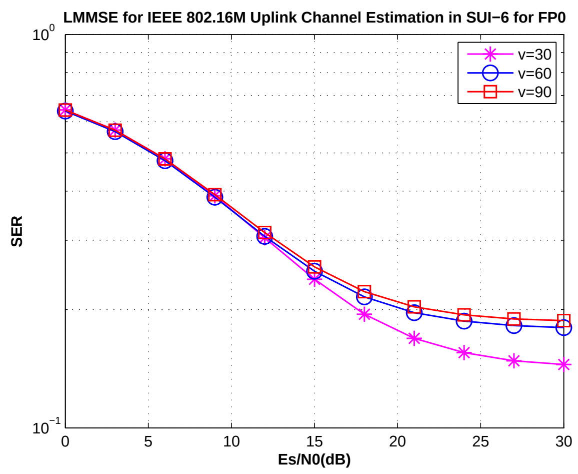


(c)

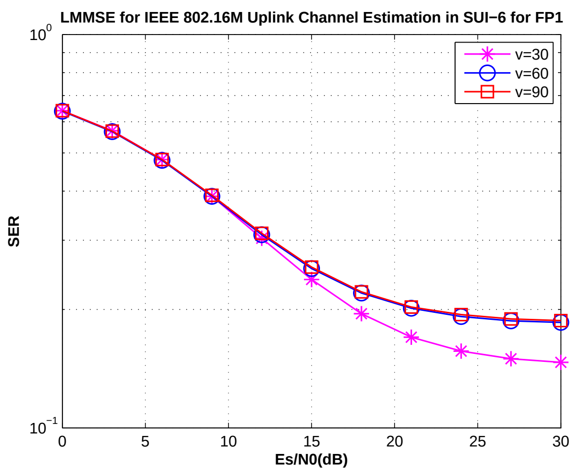


(d)

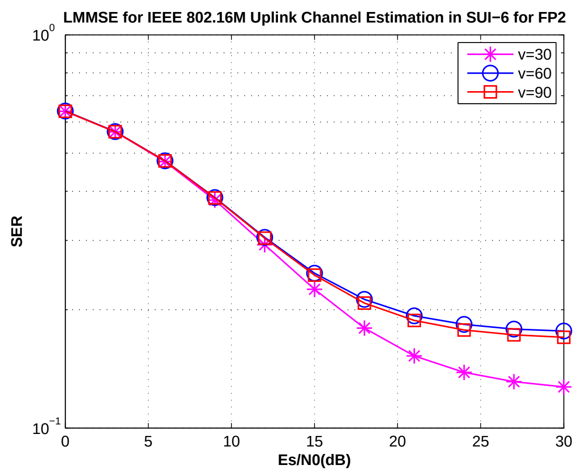
Figure C.11: Channel estimation MSE use QPSK modulation for different frequency partitions at different velocities in SUI-6 channel for IEEE 802.16m uplink, where the speed v is km/h.



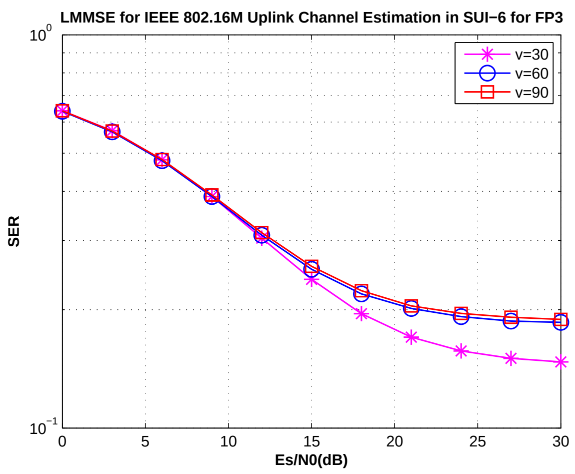
(a)



(b)



(c)



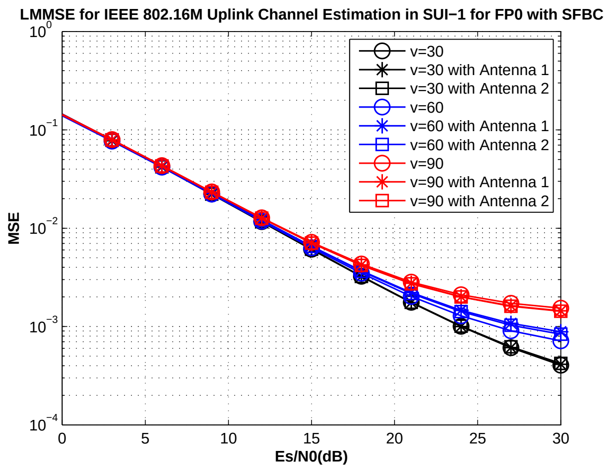
(d)

Figure C.12: Channel estimation SER use QPSK modulation for different frequency partitions at different velocities in SUI-6 channel for IEEE 802.16m uplink, where the speed v is km/h.

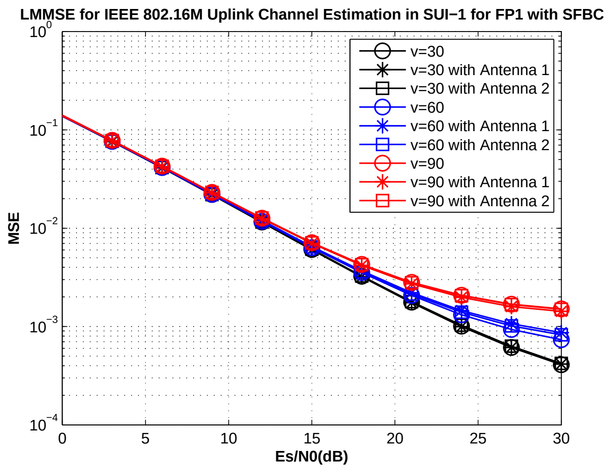
Appendix D

Additional Uplink SFBC MIMO Simulation Result for Floating-Point

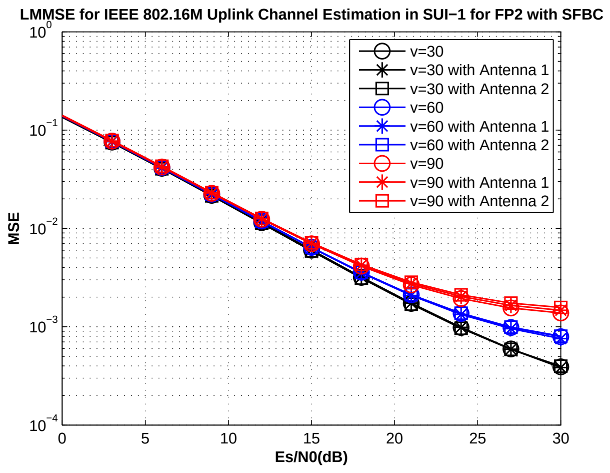
In this appendix, we place additional uplink SFBC MIMO numerical result used floating-point. Figs. D.1 to D.12 shows the SER and MSE for each SUI channel.



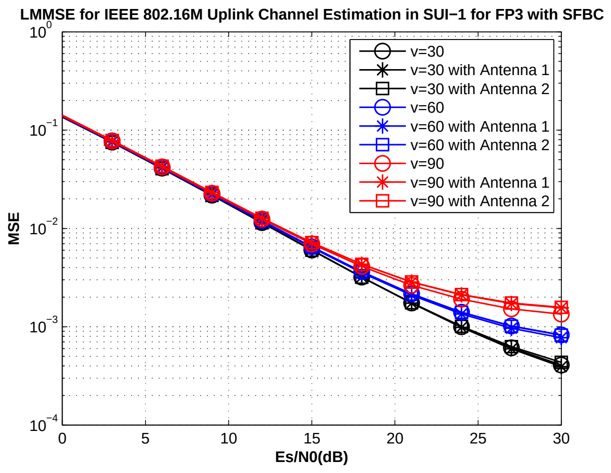
(a)



(b)



(c)



(d)

Figure D.1: Channel estimation MSE use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-1 channel for IEEE 802.16m uplink, where the speed v is km/h.

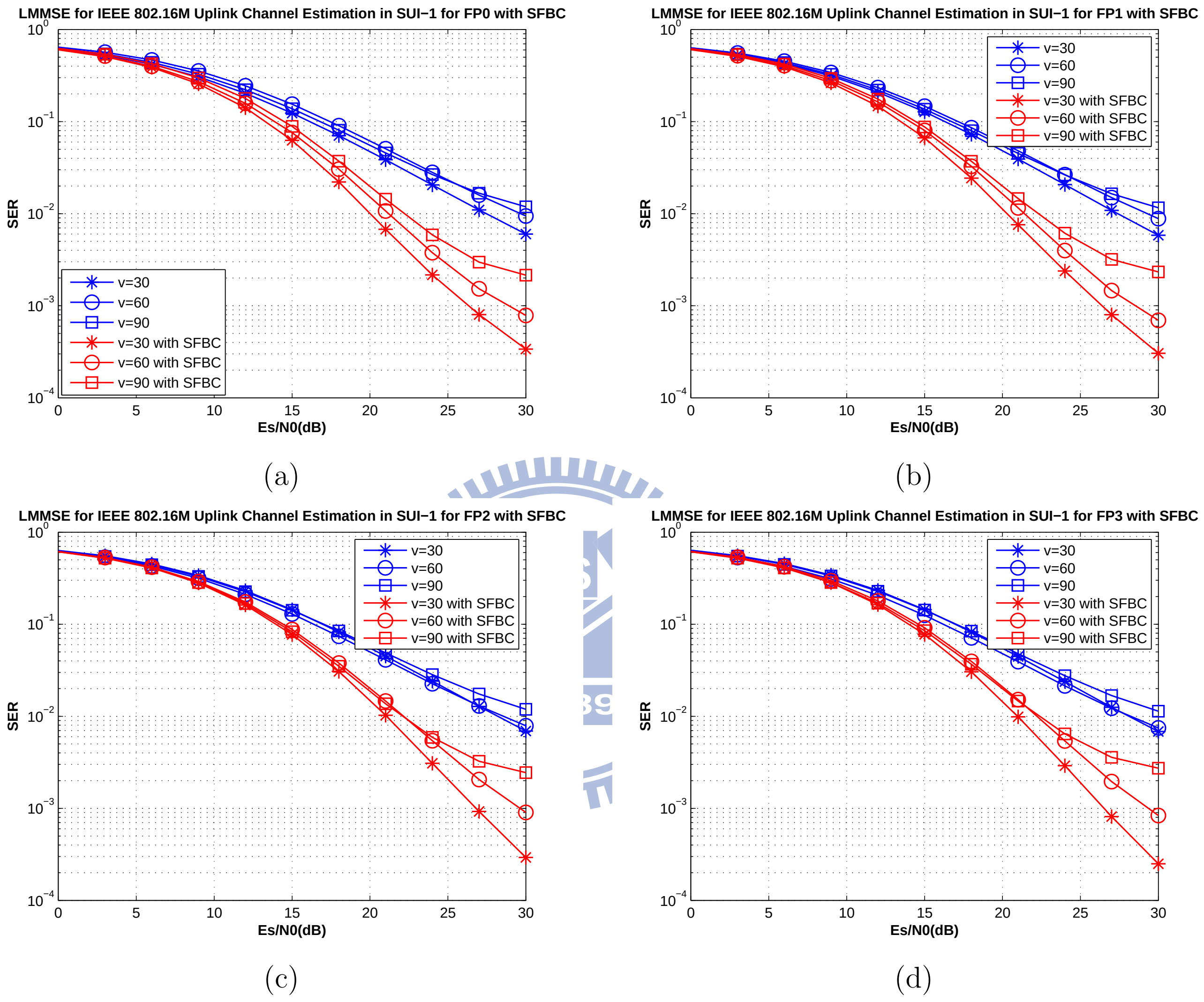
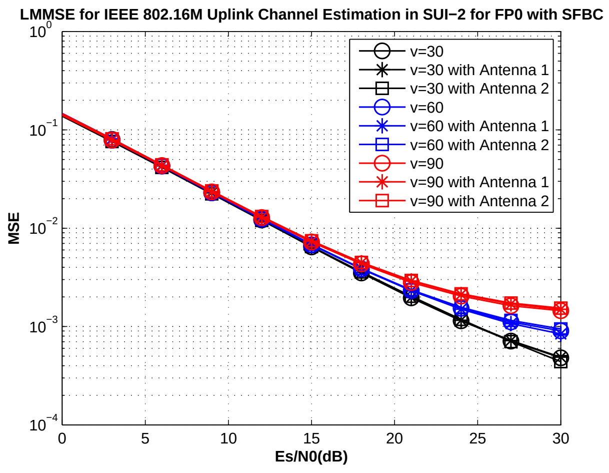
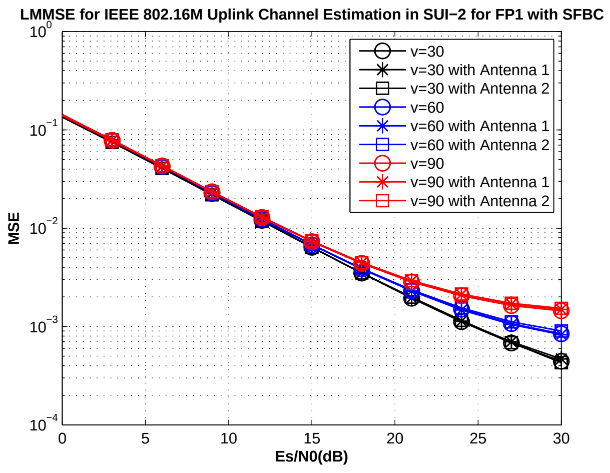


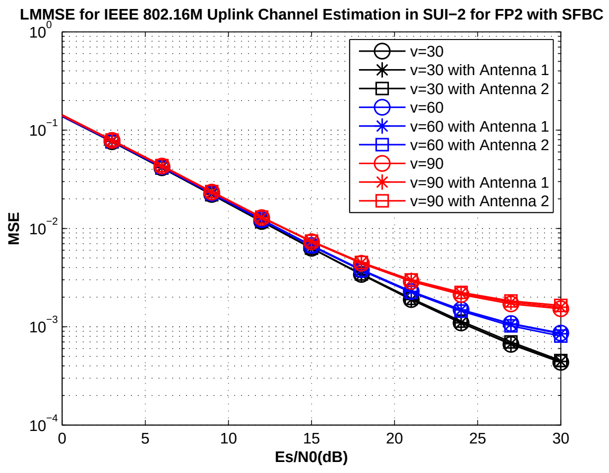
Figure D.2: Channel estimation SER use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-1 channel for IEEE 802.16m uplink, where the speed v is km/h.



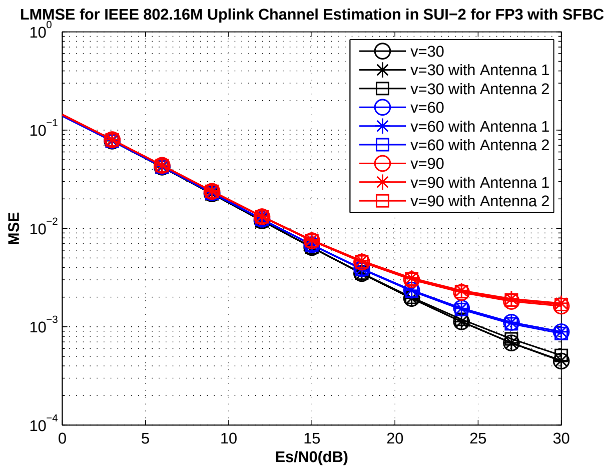
(a)



(b)

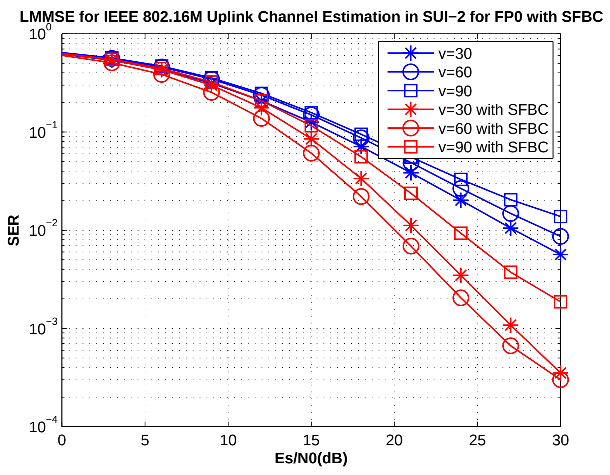


(c)

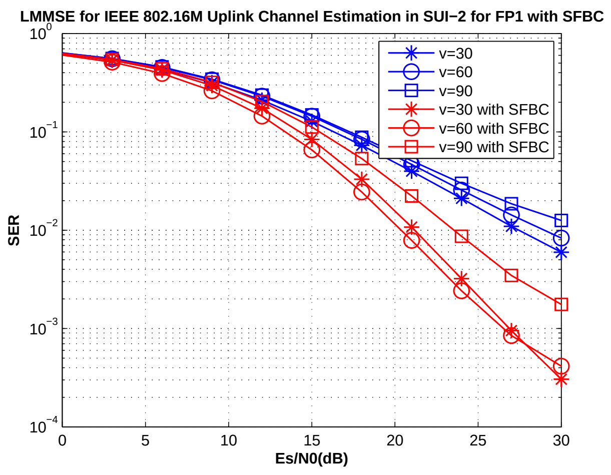


(d)

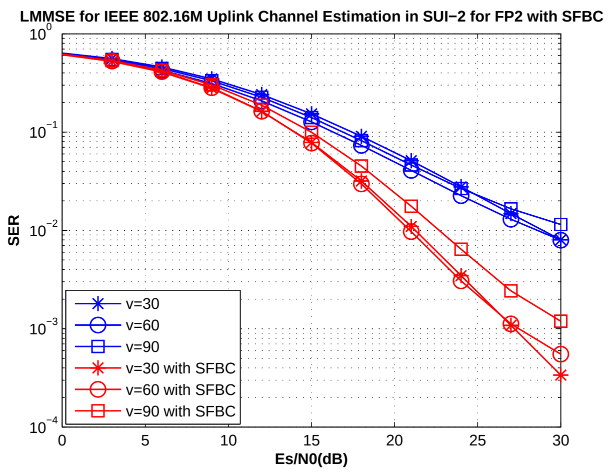
Figure D.3: Channel estimation MSE use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-2 channel for IEEE 802.16m uplink, where the speed v is km/h.



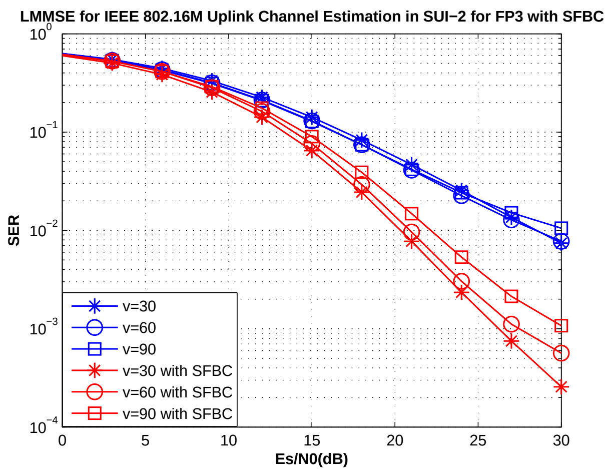
(a)



(b)

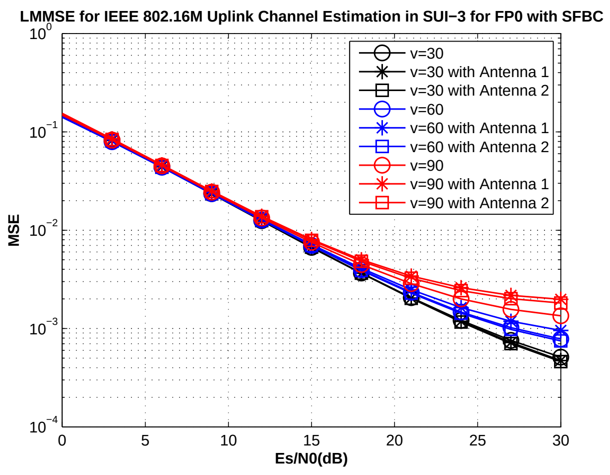


(c)

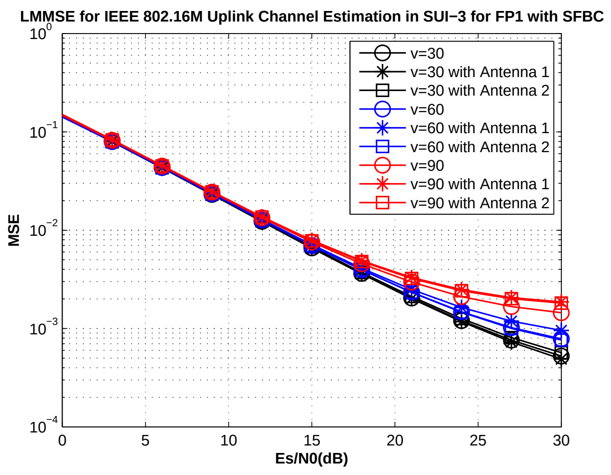


(d)

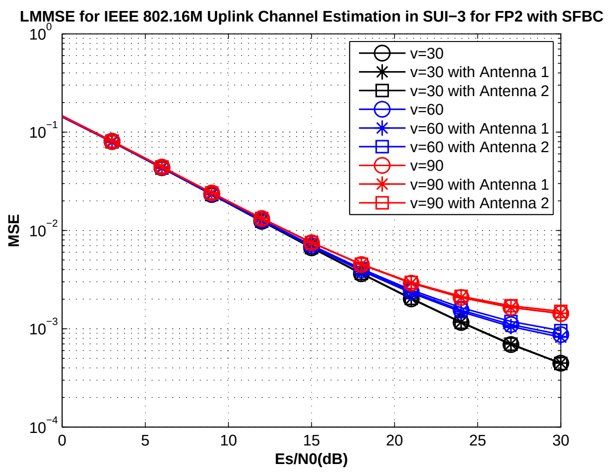
Figure D.4: Channel estimation SER use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-2 channel for IEEE 802.16m uplink, where the speed v is km/h.



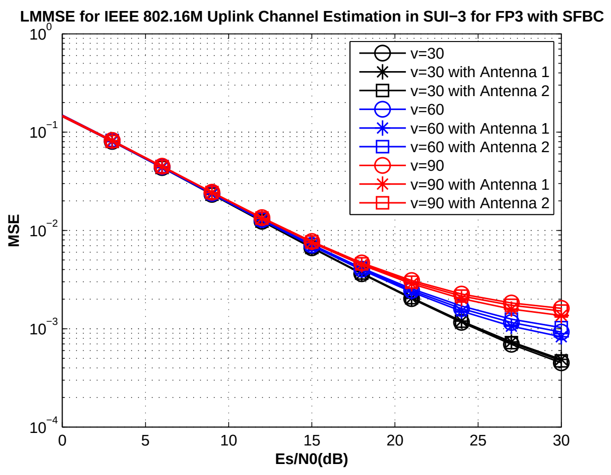
(a)



(b)

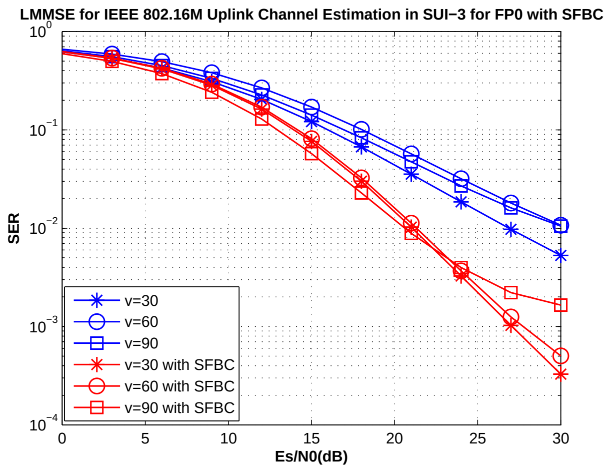


(c)

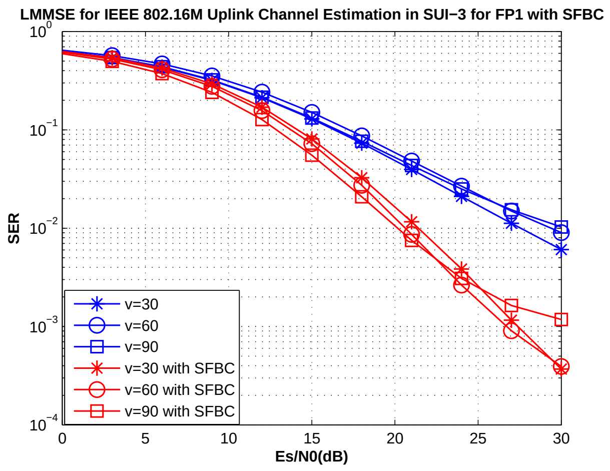


(d)

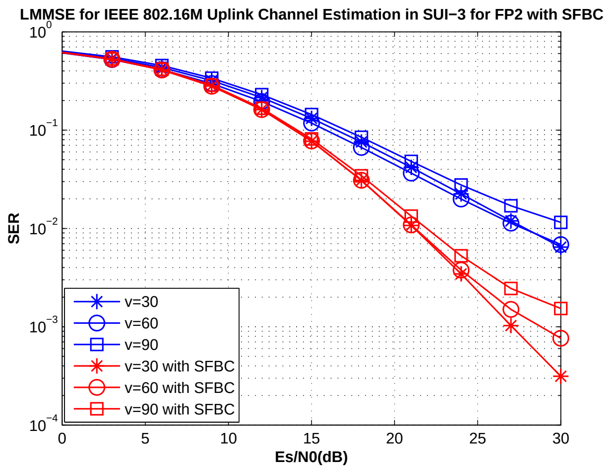
Figure D.5: Channel estimation MSE use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-3 channel for IEEE 802.16m uplink, where the speed v is km/h.



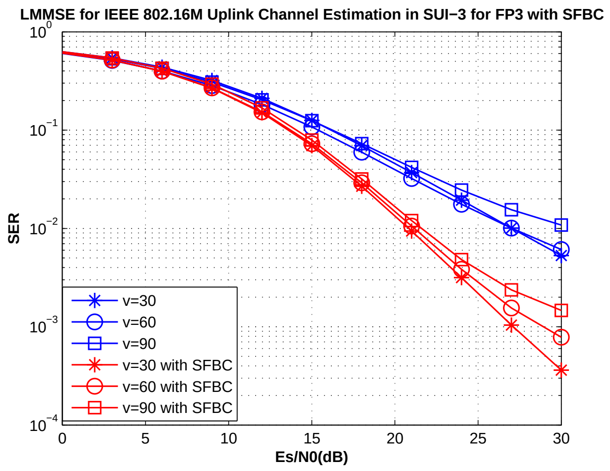
(a)



(b)

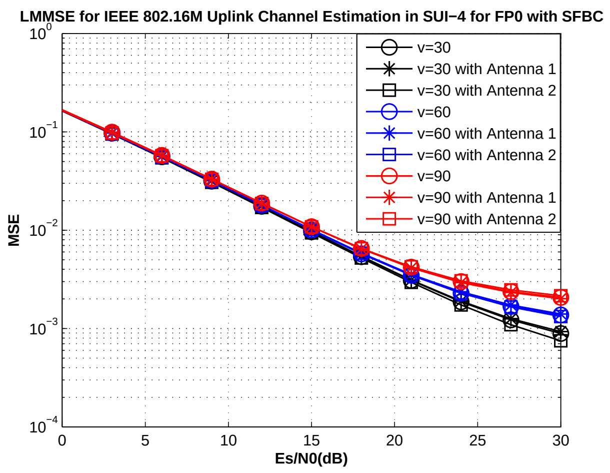


(c)

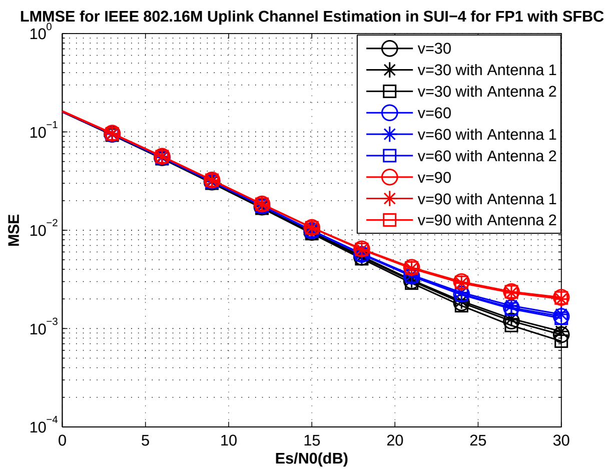


(d)

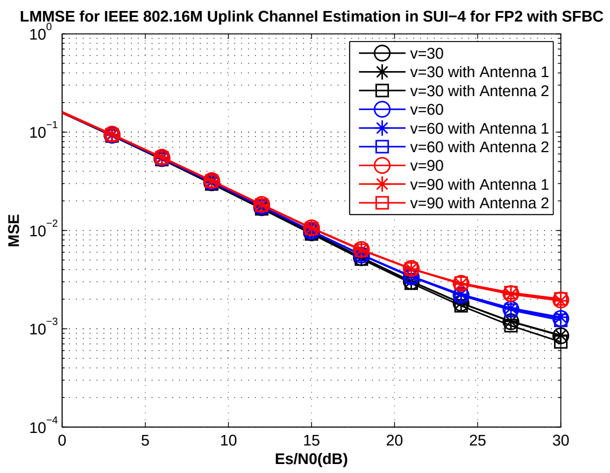
Figure D.6: Channel estimation SER use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-3 channel for IEEE 802.16m uplink, where the speed v is km/h.



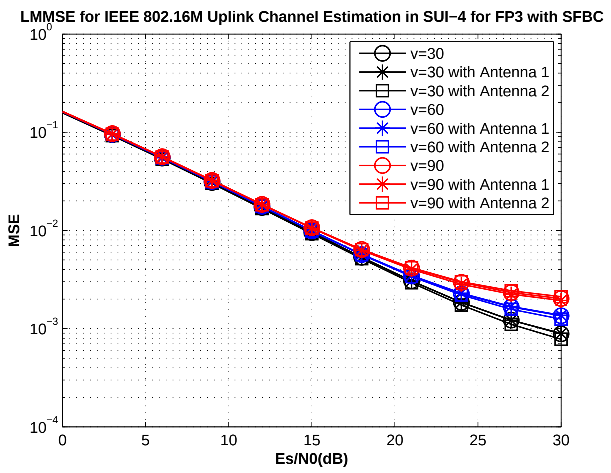
(a)



(b)

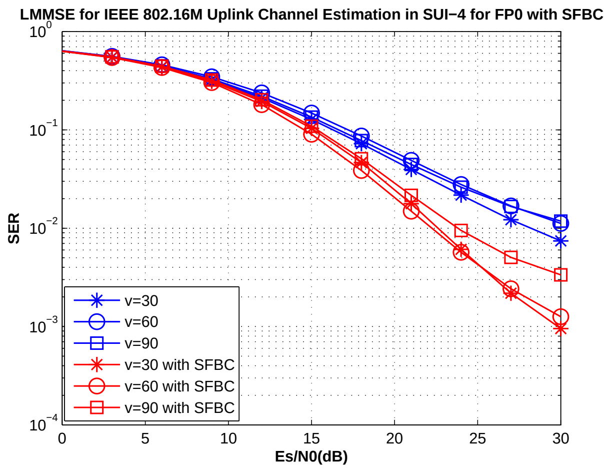


(c)

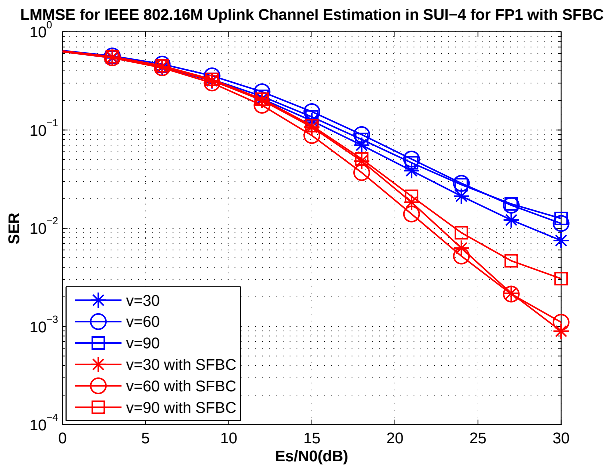


(d)

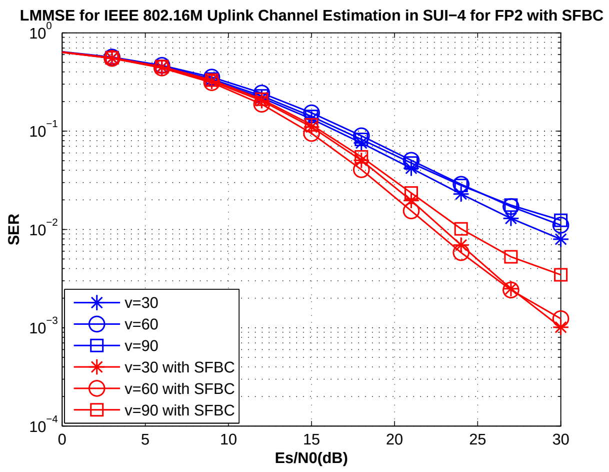
Figure D.7: Channel estimation MSE use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-4 channel for IEEE 802.16m uplink, where the speed v is km/h.



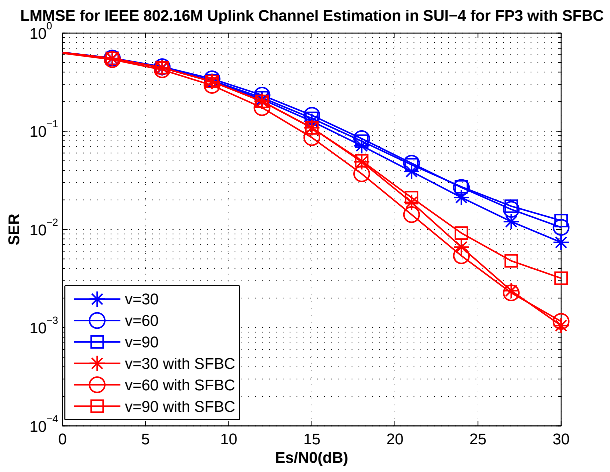
(a)



(b)

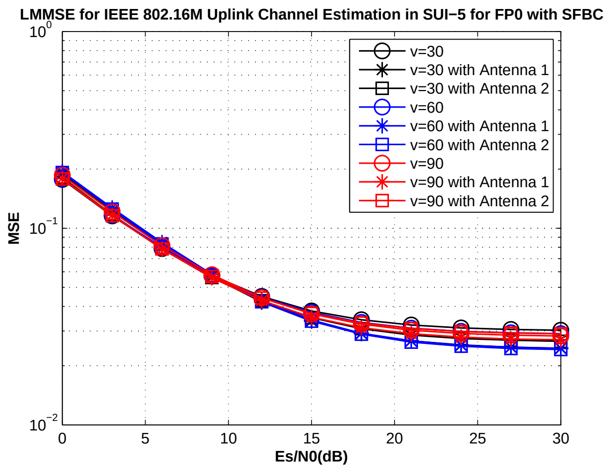


(c)

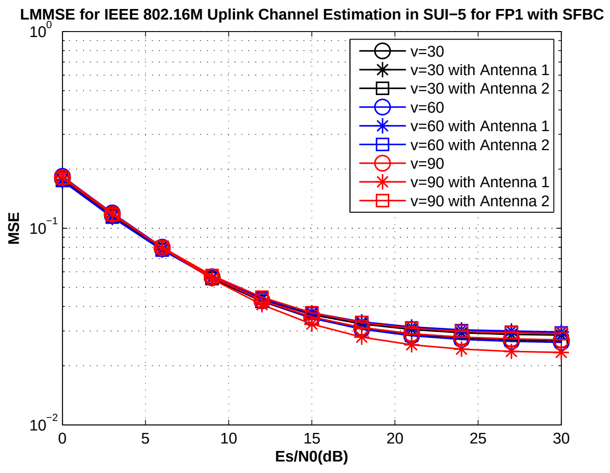


(d)

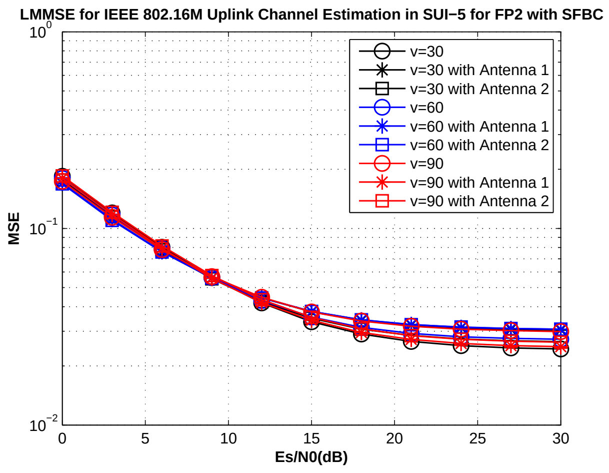
Figure D.8: Channel estimation SER use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-4 channel for IEEE 802.16m uplink, where the speed v is km/h.



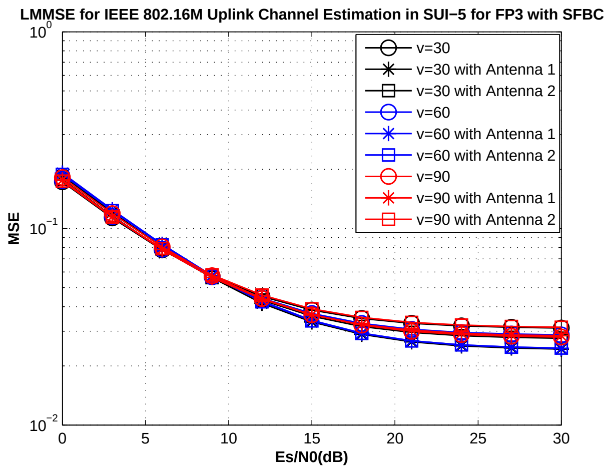
(a)



(b)

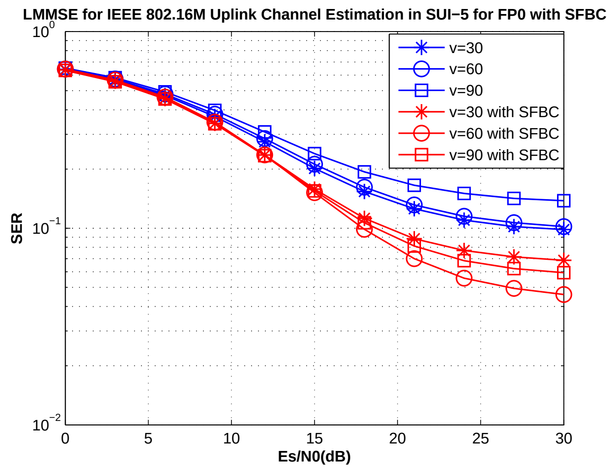


(c)

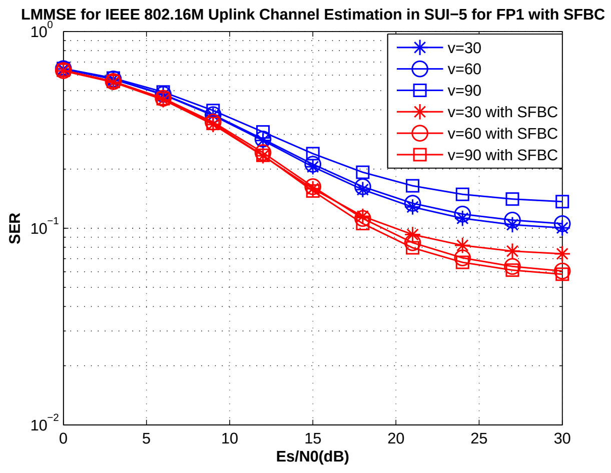


(d)

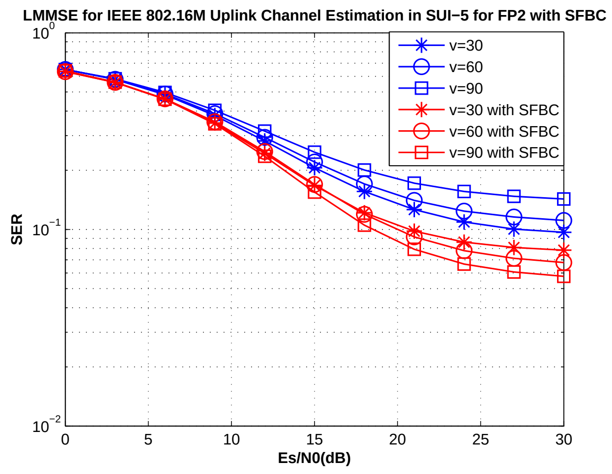
Figure D.9: Channel estimation MSE use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-5 channel for IEEE 802.16m uplink, where the speed v is km/h.



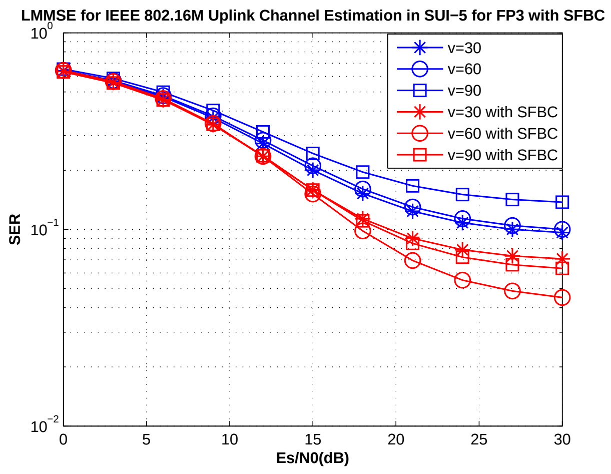
(a)



(b)

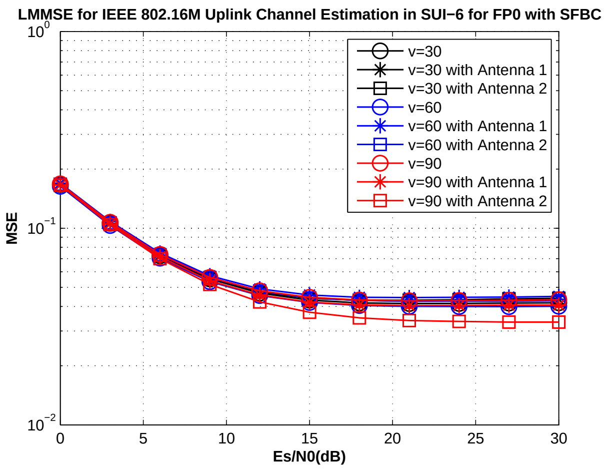


(c)

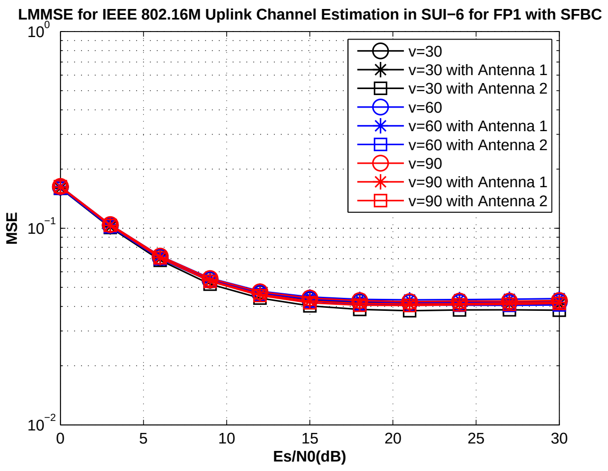


(d)

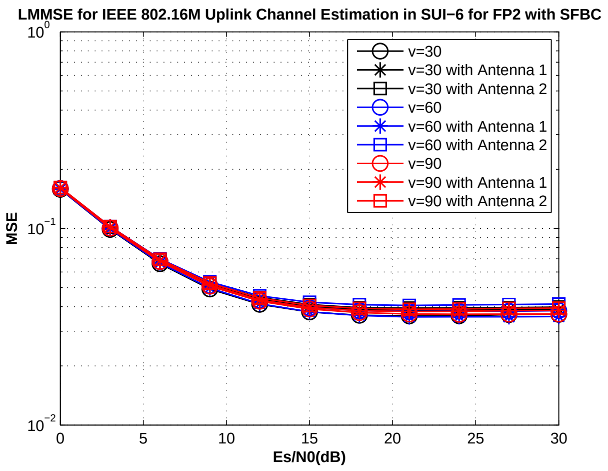
Figure D.10: Channel estimation SER use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-5 channel for IEEE 802.16m uplink, where the speed v is km/h.



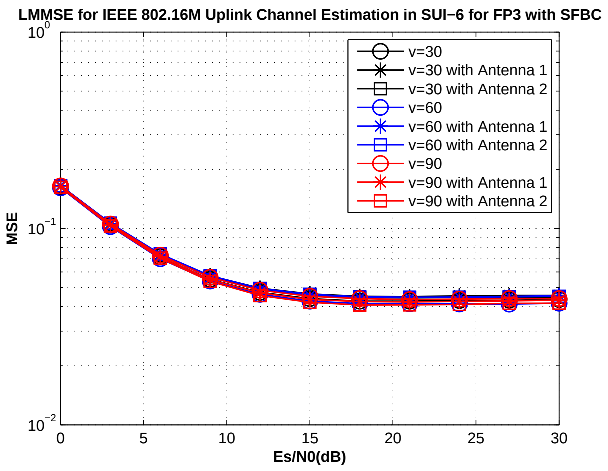
(a)



(b)

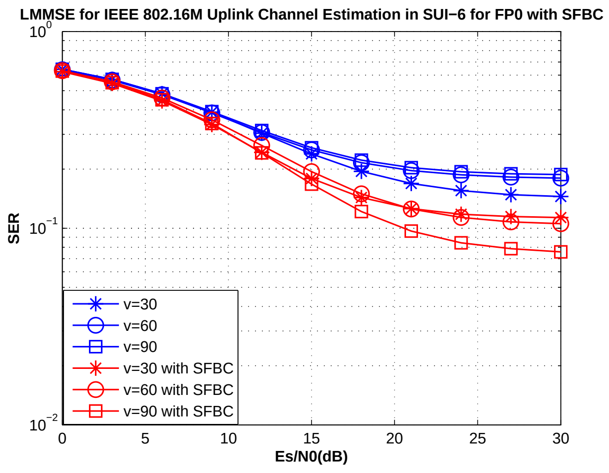


(c)

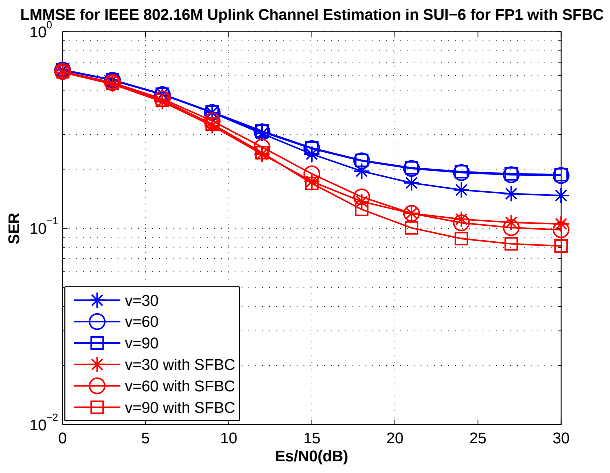


(d)

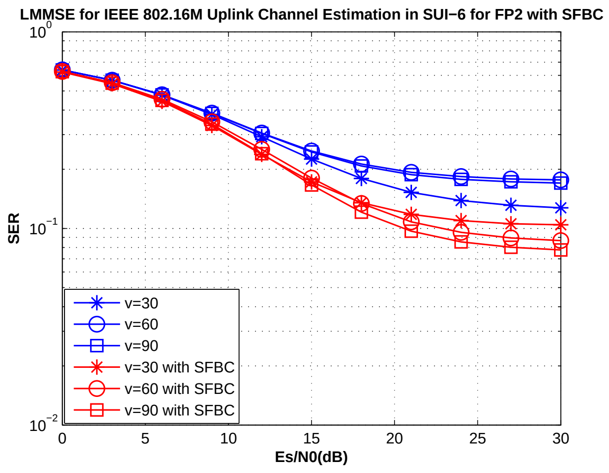
Figure D.11: Channel estimation MSE use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-6 channel for IEEE 802.16m uplink, where the speed v is km/h.



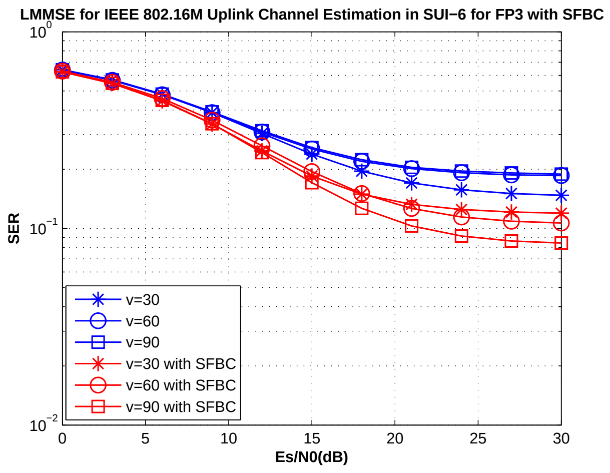
(a)



(b)



(c)



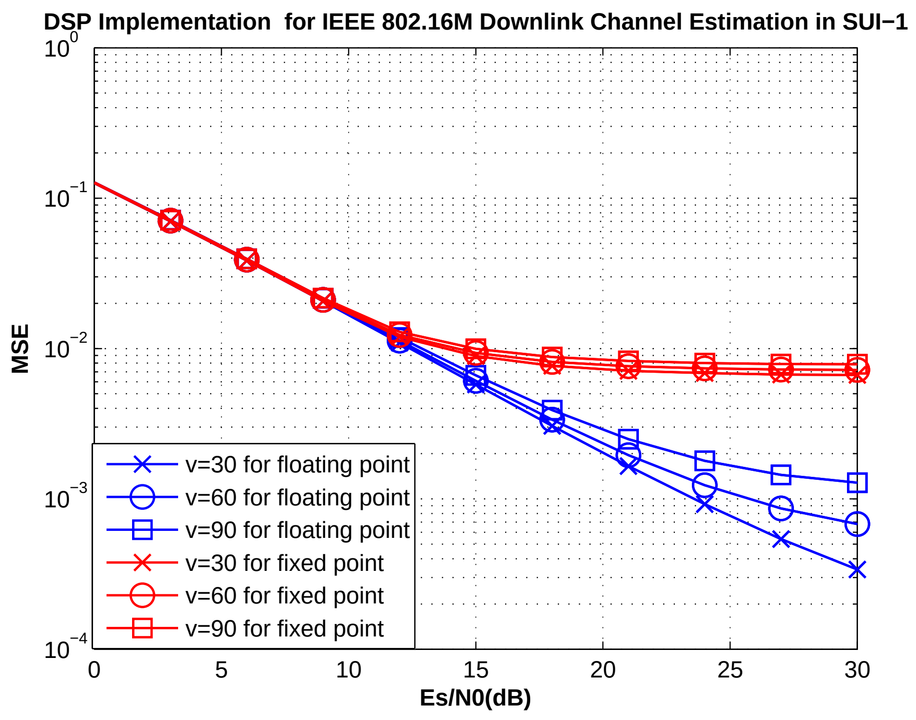
(d)

Figure D.12: Channel estimation SER use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-6 channel for IEEE 802.16m uplink, where the speed v is km/h.

Appendix E

Additional Downlink SISO Simulation Result for Fixed-Point

In this appendix, we place additional downlink SISO numerical result used fixed-point. Figs. E.1 to E.4 shows the SER and MSE for each SUI channel.



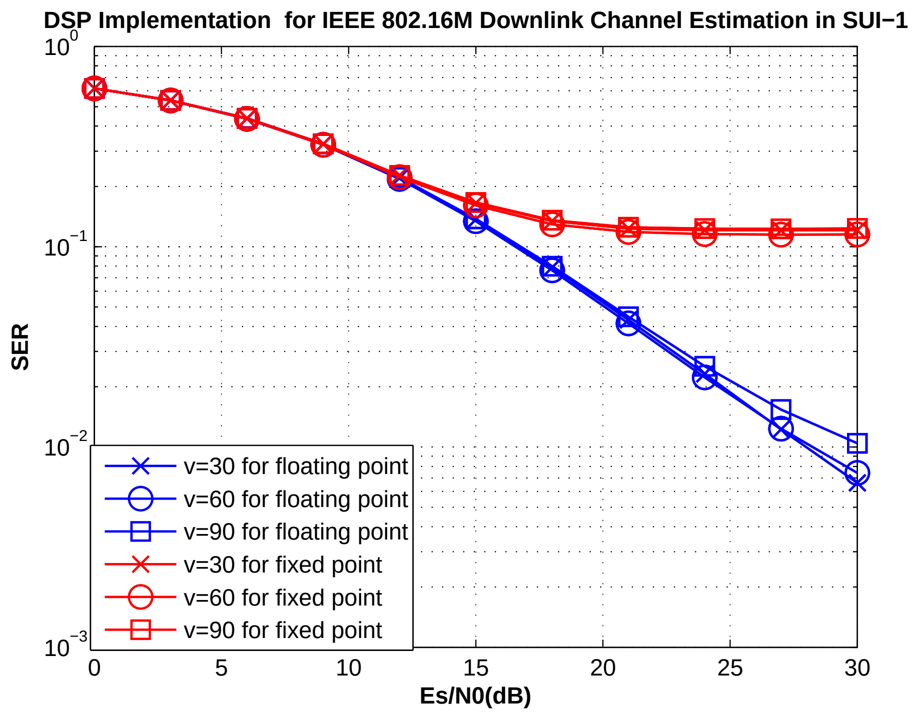
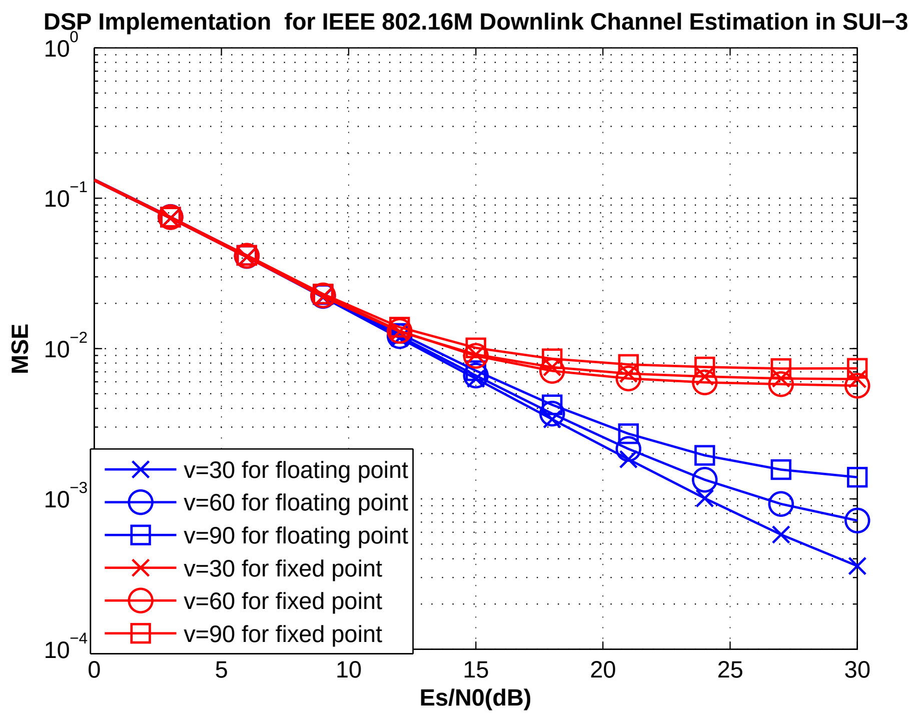


Figure E.1: Fixed-point channel estimation MSE and SER for QPSK at different velocities in SUI-1 channel for IEEE 802.16m downlink, where the speed v is km/h.



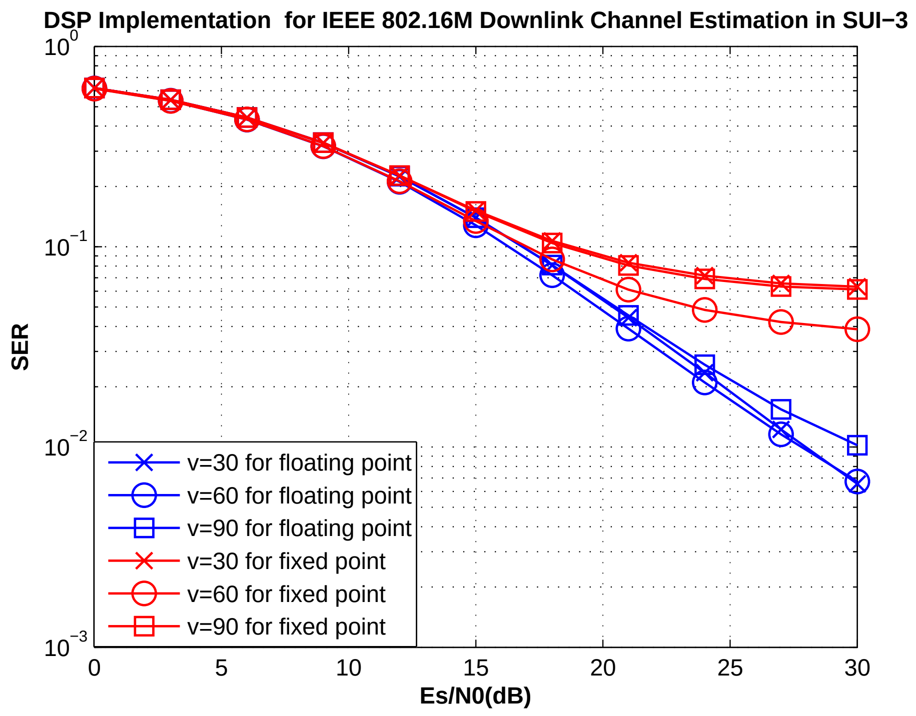
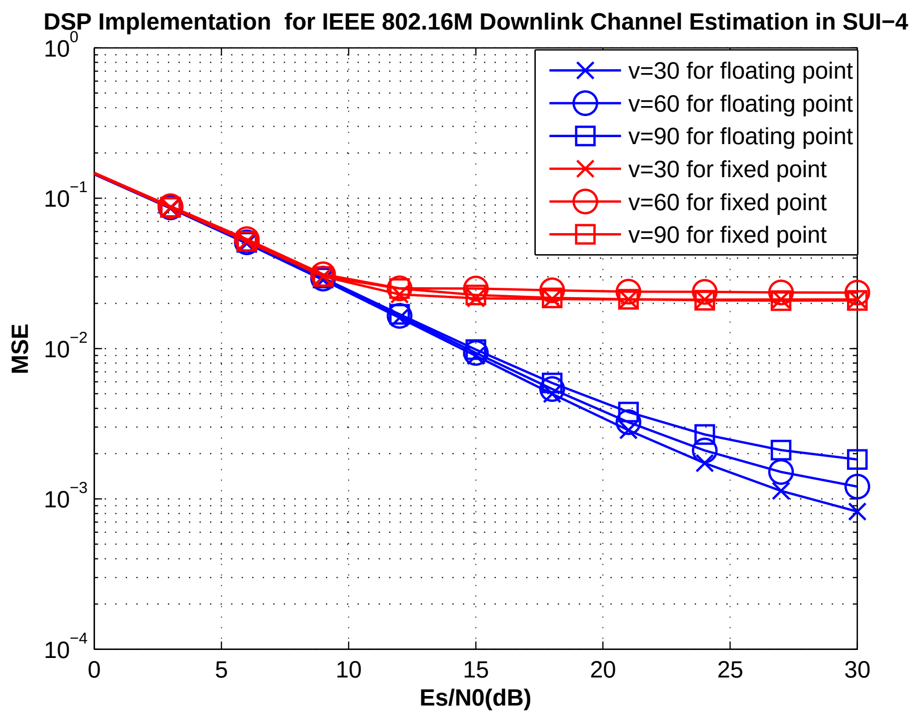


Figure E.2: Fixed-point channel estimation MSE and SER for QPSK at different velocities in SUI-3 channel for IEEE 802.16m downlink, where the speed v is km/h.



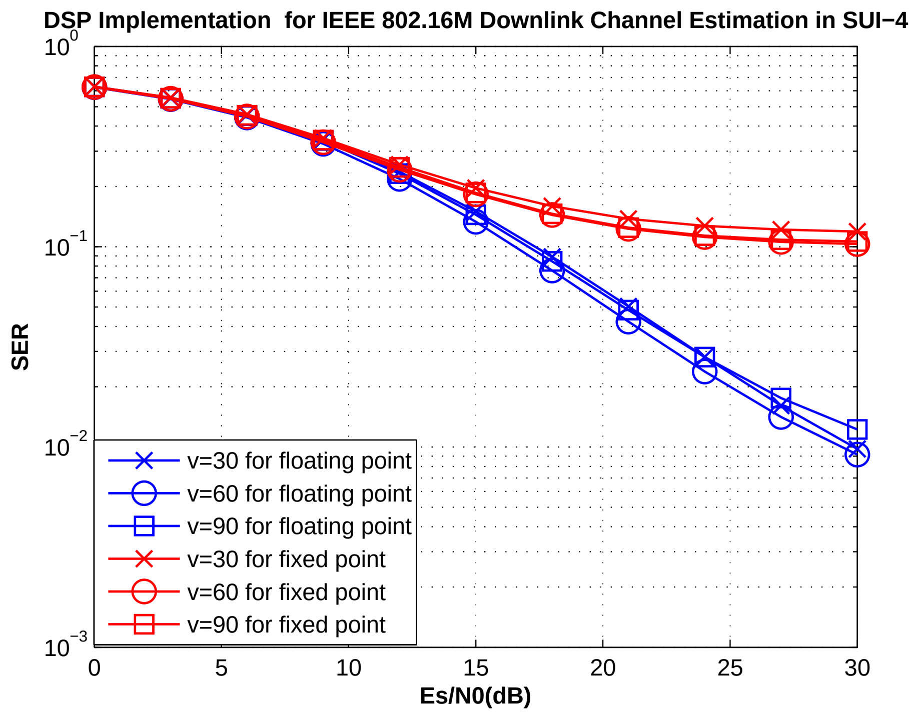
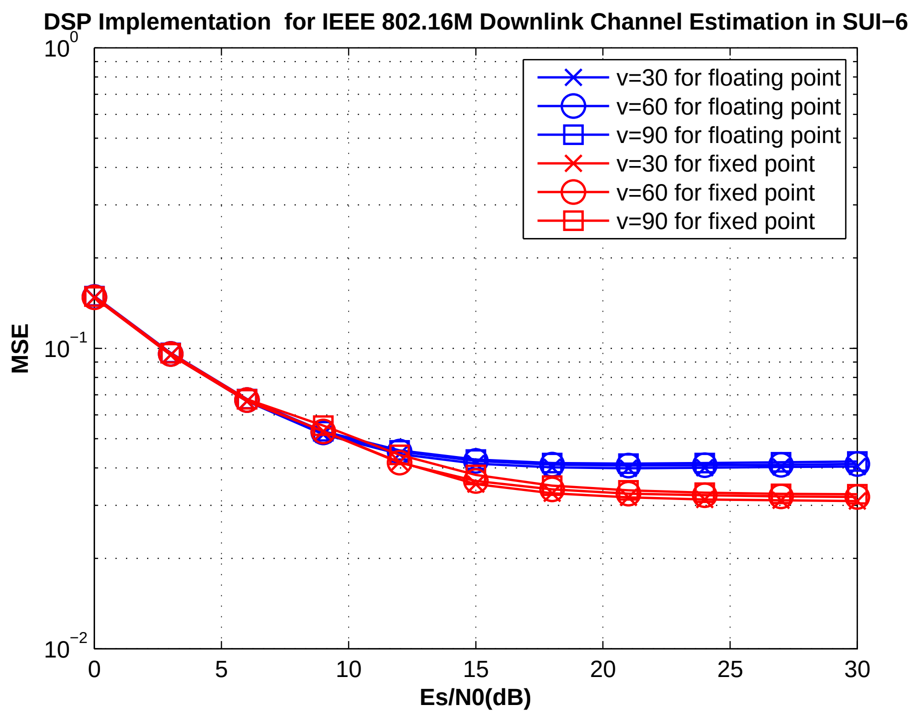


Figure E.3: Fixed-point channel estimation MSE and SER for QPSK at different velocities in SUI-4 channel for IEEE 802.16m downlink, where the speed v is km/h.



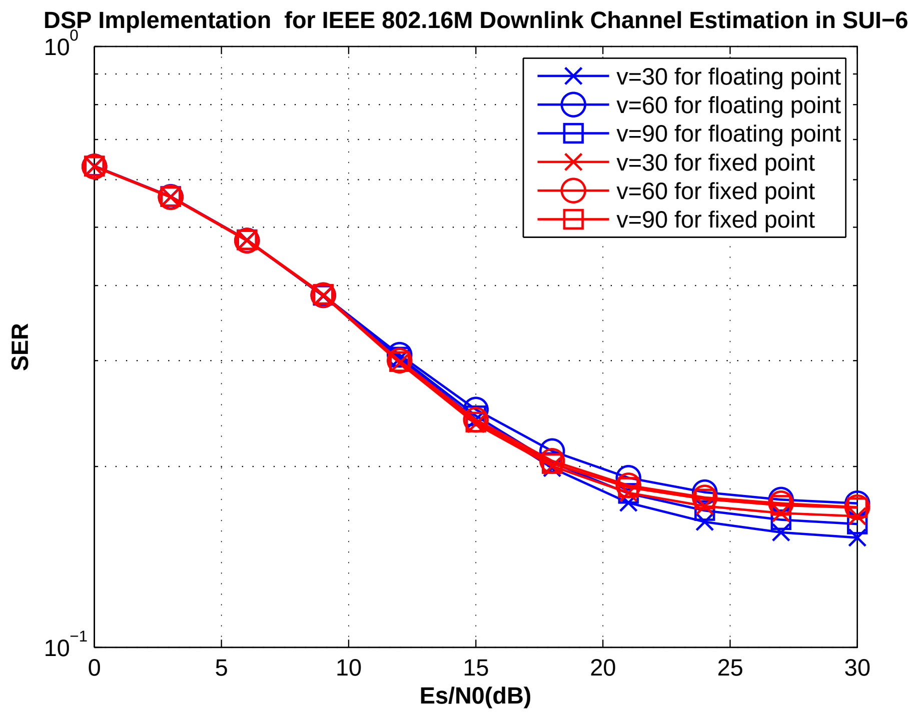
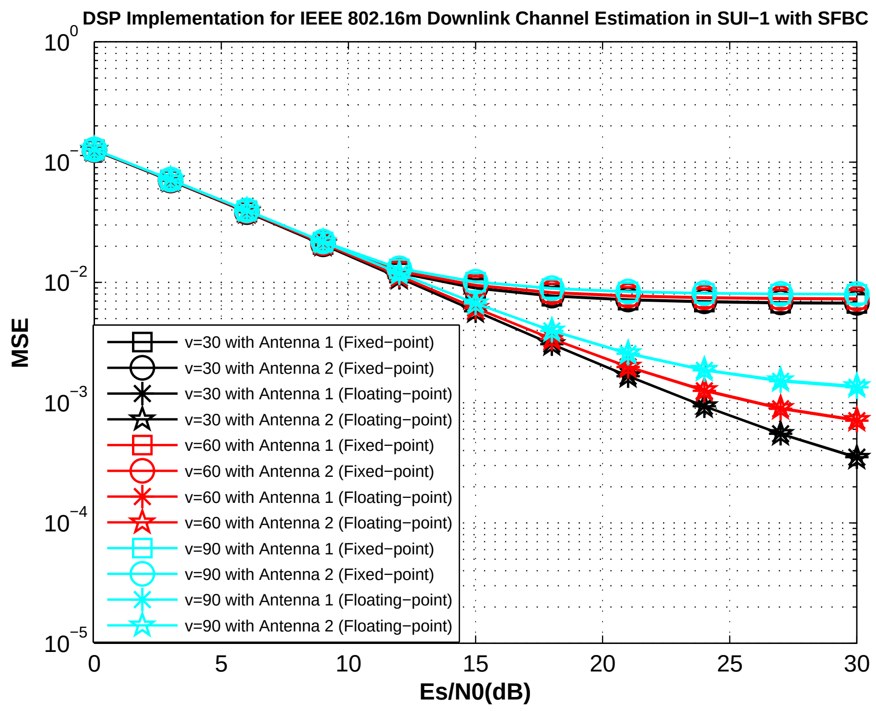


Figure E.4: Fixed-point channel estimation MSE and SER for QPSK at different velocities in SUI-6 channel for IEEE 802.16m downlink, where the speed v is km/h.

Appendix F

Additional Downlink SFBC MIMO Simulation Result for Fixed-Point

In this appendix, we place additional downlink SFBC MIMO numerical result used fixed-point. Figs. F.1 to F.4 shows the SER and MSE for each SUI channel.



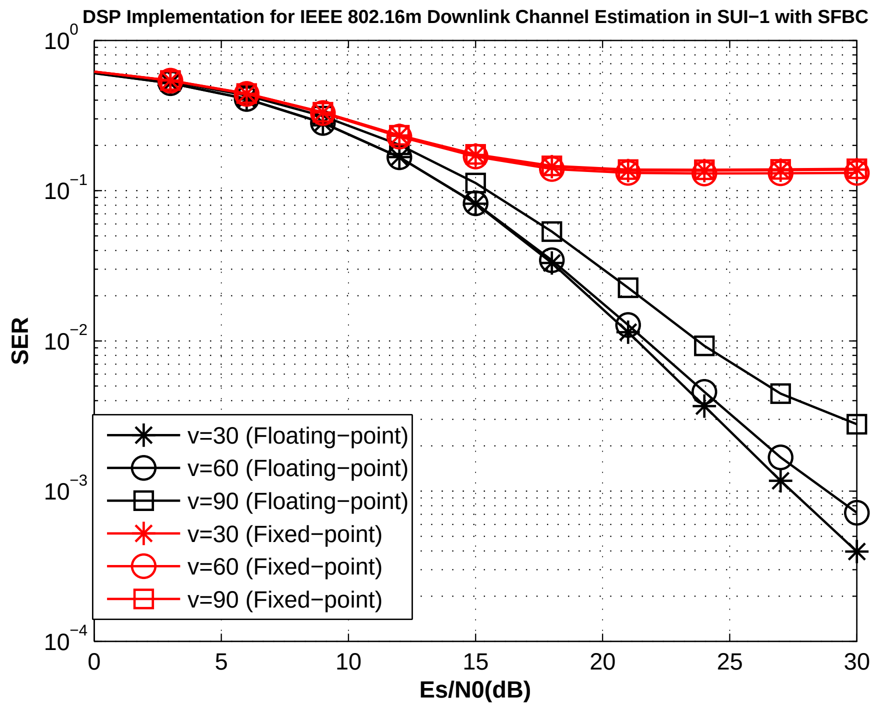
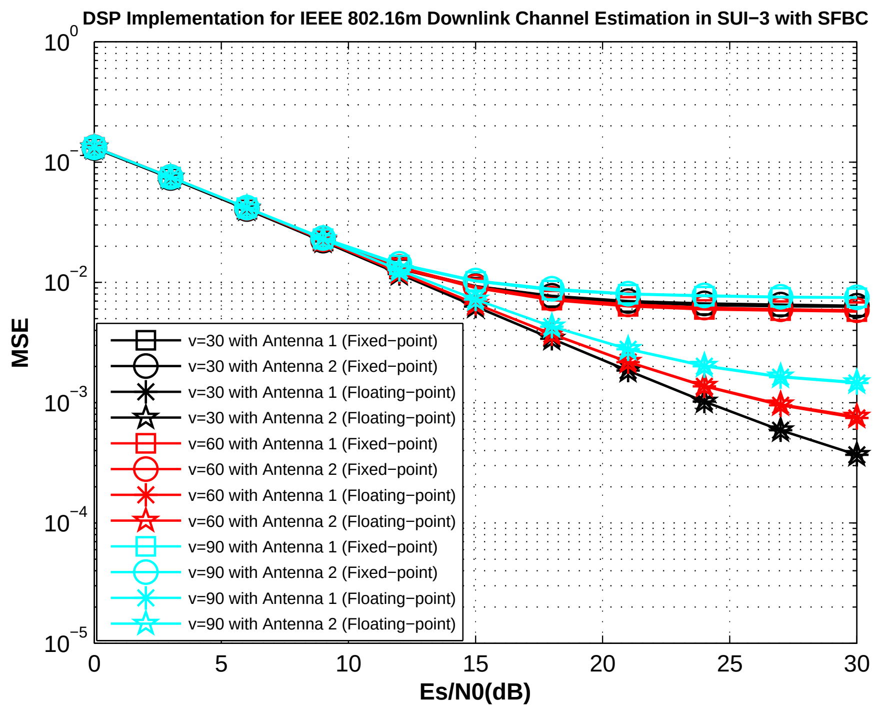


Figure F.1: Fixed-point channel estimation MSE and SER for QPSK with SFBC at different velocities in SUI-1 channel for IEEE 802.16m downlink, where the speed v is km/h.



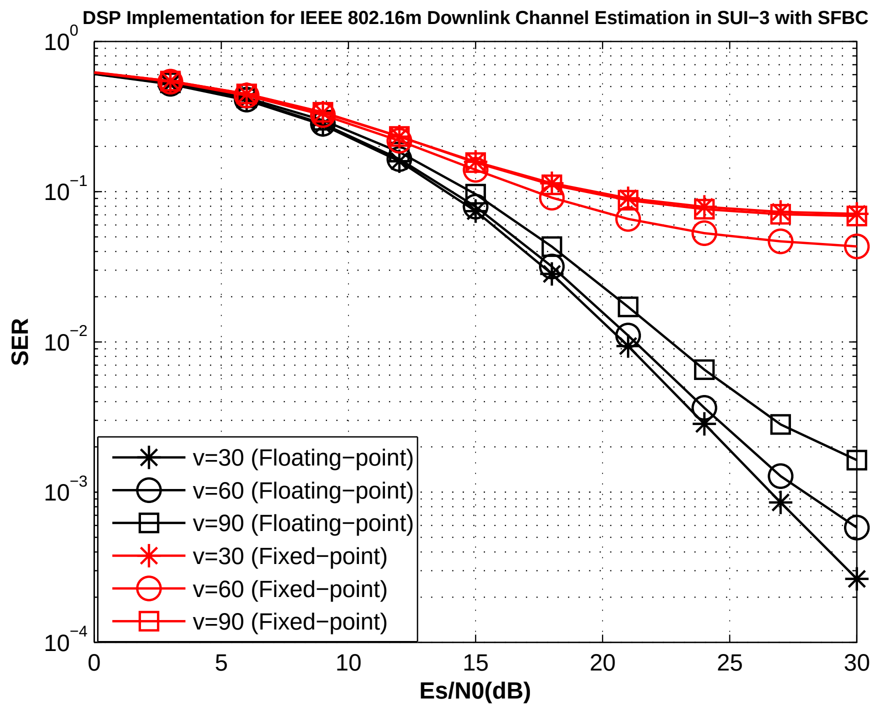
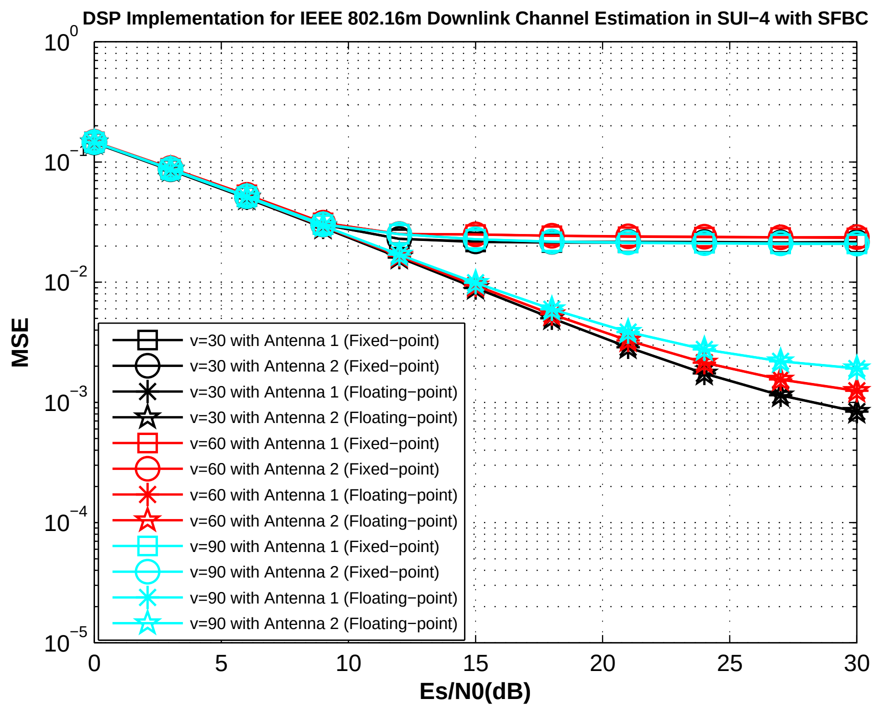


Figure F.2: Fixed-point channel estimation MSE and SER for QPSK with SFBC at different velocities in SUI-3 channel for IEEE 802.16m downlink, where the speed v is km/h.



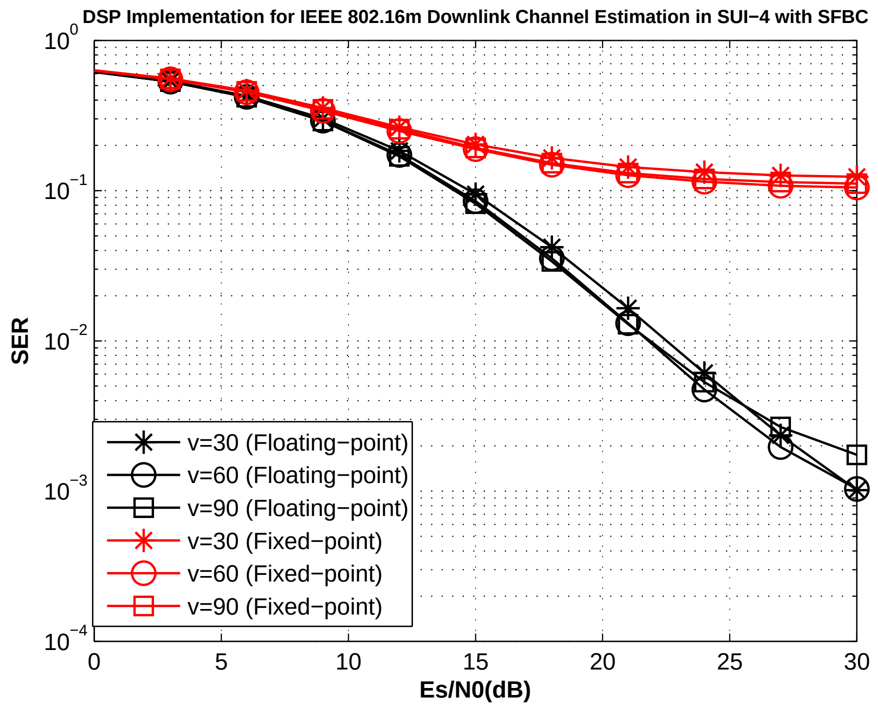
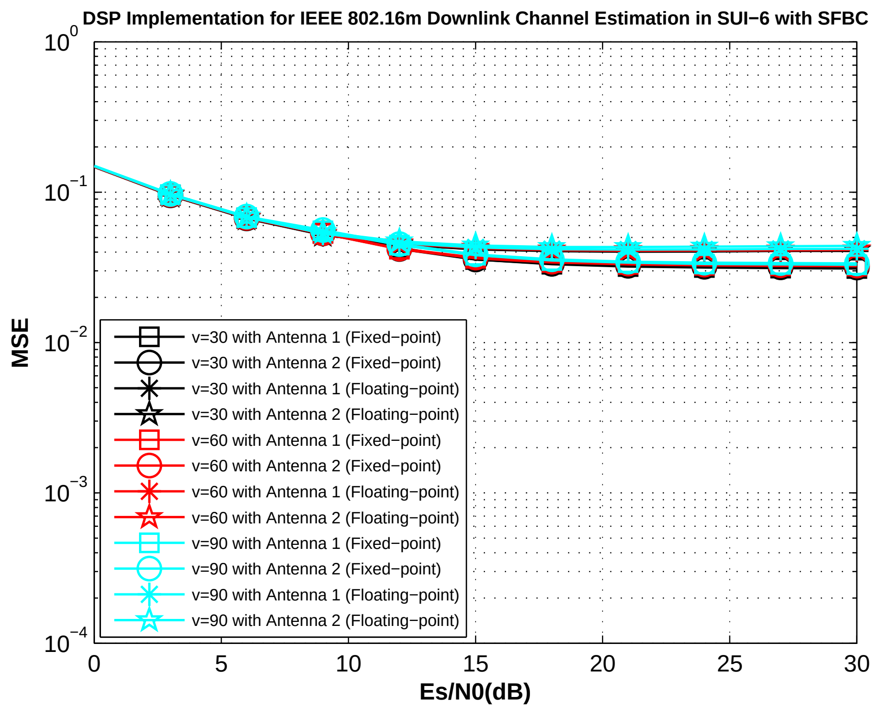


Figure F.3: Fixed-point channel estimation MSE and SER for QPSK with SFBC at different velocities in SUI-4 channel for IEEE 802.16m downlink, where the speed v is km/h.



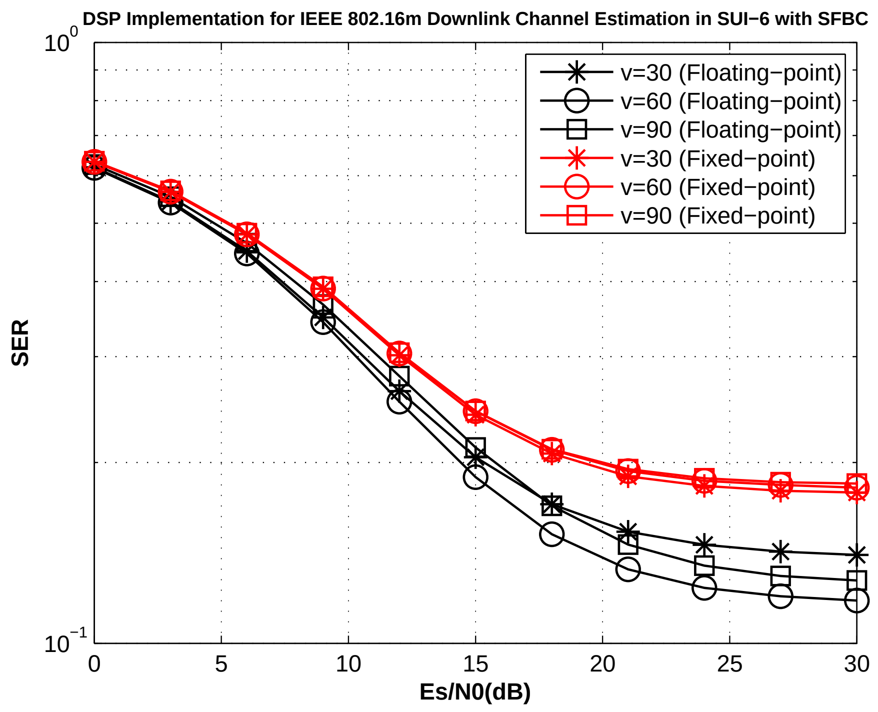
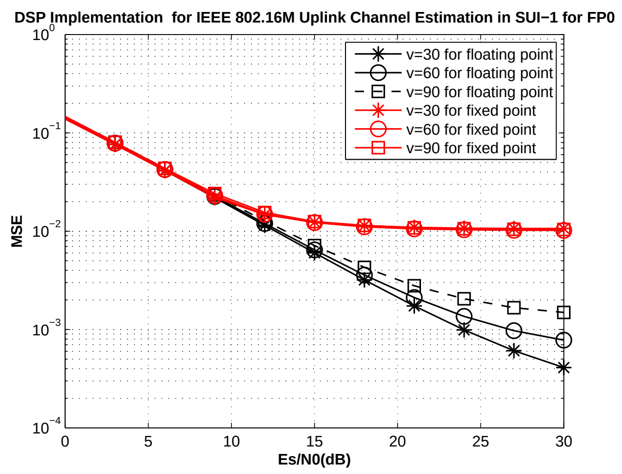


Figure F.4: Fixed-point channel estimation MSE and SER for QPSK with SFBC at different velocities in SUI-6 channel for IEEE 802.16m downlink, where the speed v is km/h.

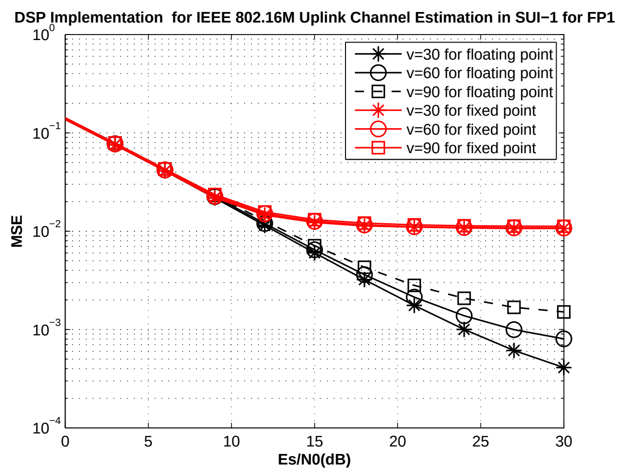
Appendix G

Additional Uplink SISO Simulation Result for Fixed-Point

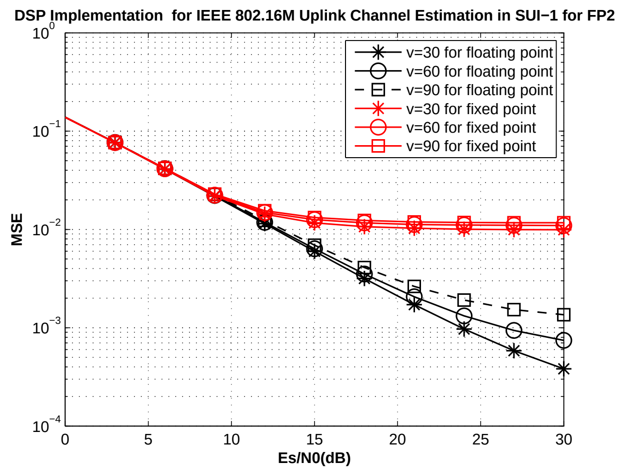
In this appendix, we place additional uplink SISO numerical result used fixed-point. Figs. G.1 to G.8 shows the SER and MSE for each SUI channel.



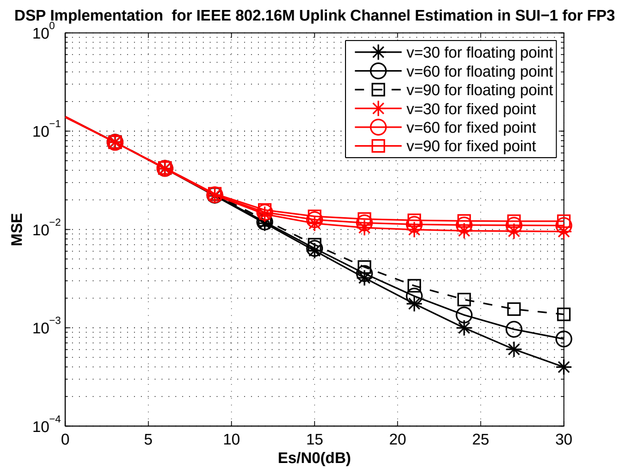
(a)



(b)

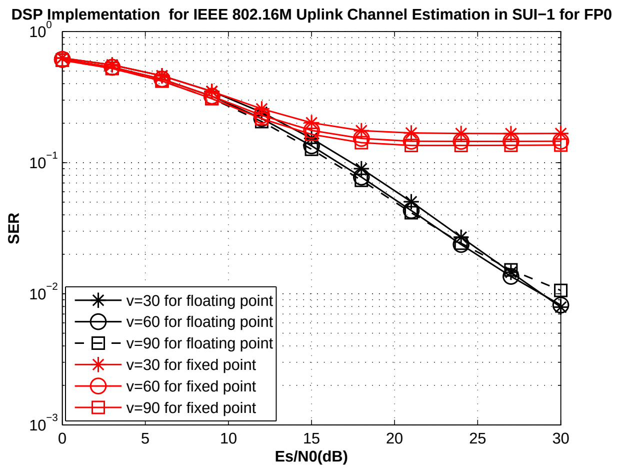


(c)

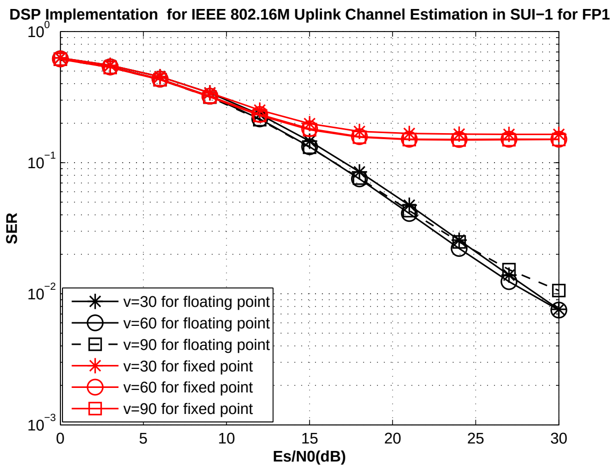


(d)

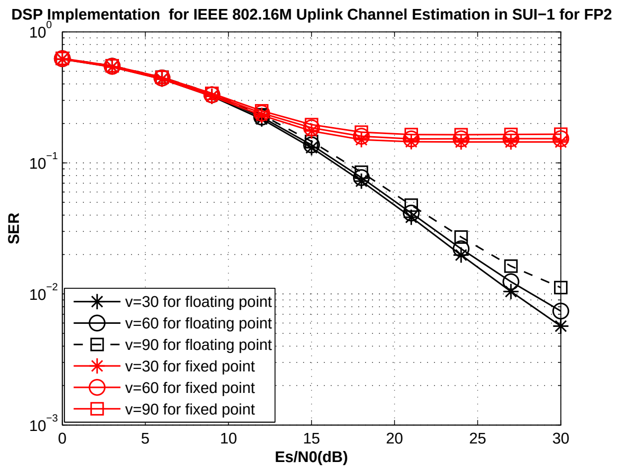
Figure G.1: Fixed-point channel estimation MSE use QPSK modulation for different frequency partitions at different velocities in SUI-1 channel for IEEE 802.16m uplink, where the speed v is km/h.



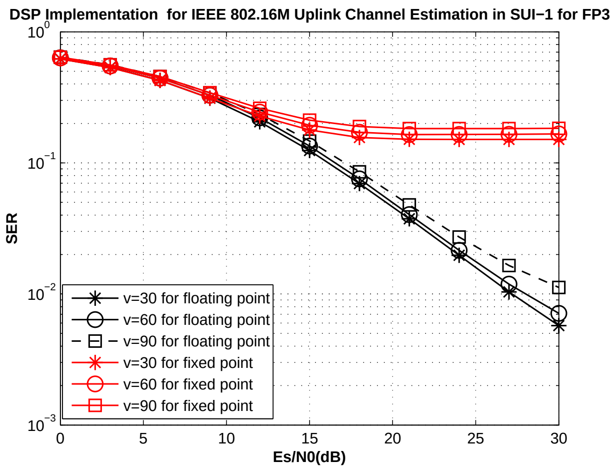
(a)



(b)

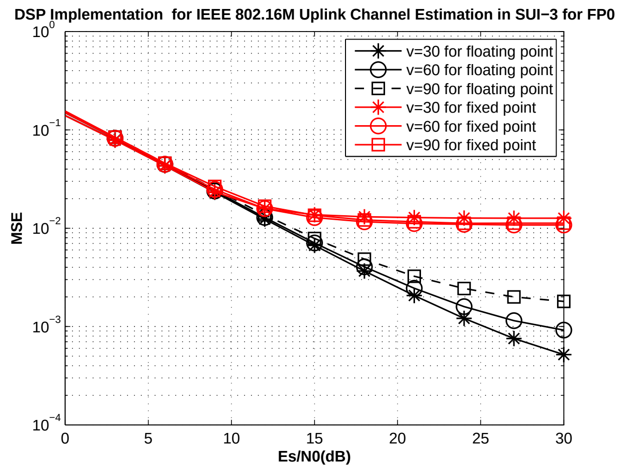


(c)

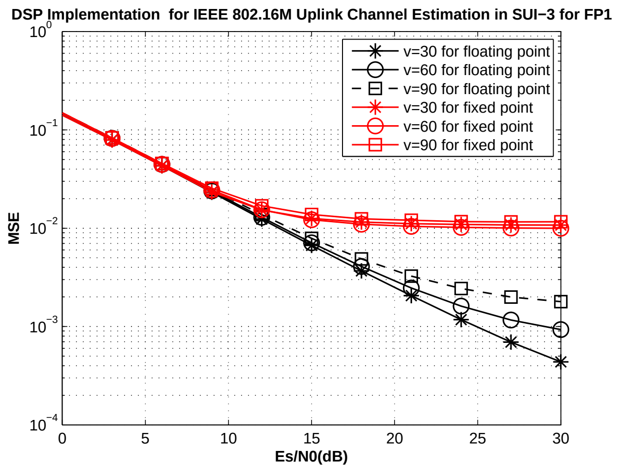


(d)

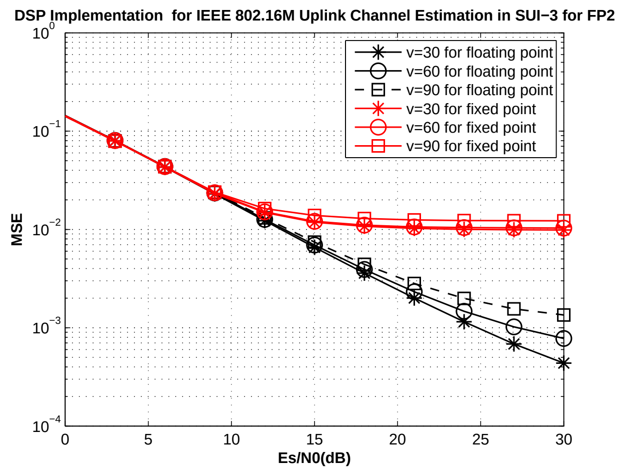
Figure G.2: Fixed-point channel estimation SER use QPSK modulation for different frequency partitions at different velocities in SUI-1 channel for IEEE 802.16m uplink, where the speed v is km/h.



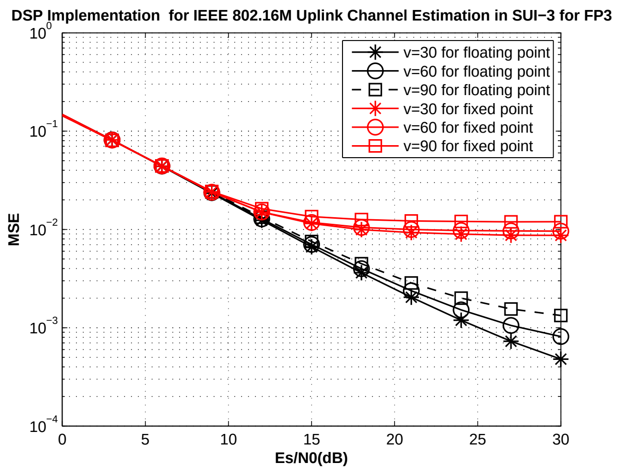
(a)



(b)

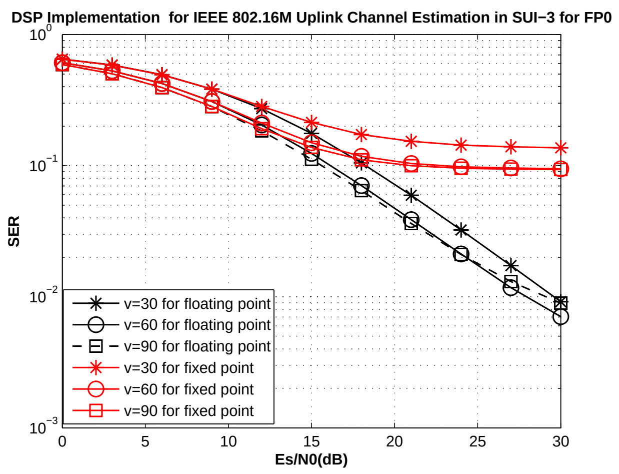


(c)

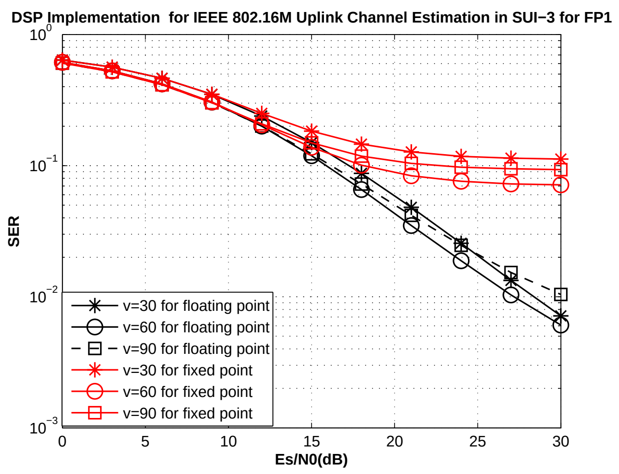


(d)

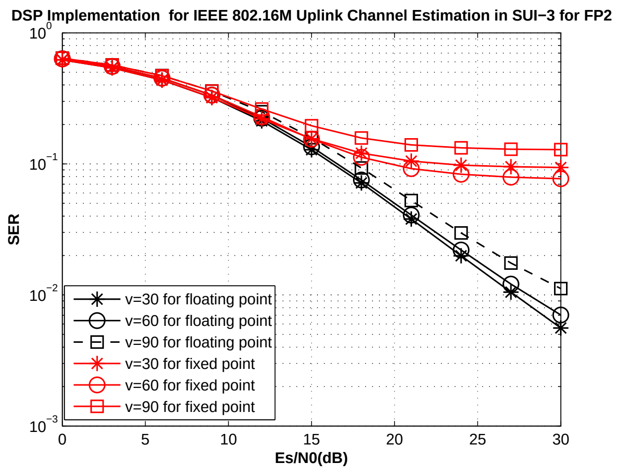
Figure G.3: Fixed-point channel estimation MSE use QPSK modulation for different frequency partitions at different velocities in SUI-3 channel for IEEE 802.16m uplink, where the speed v is km/h.



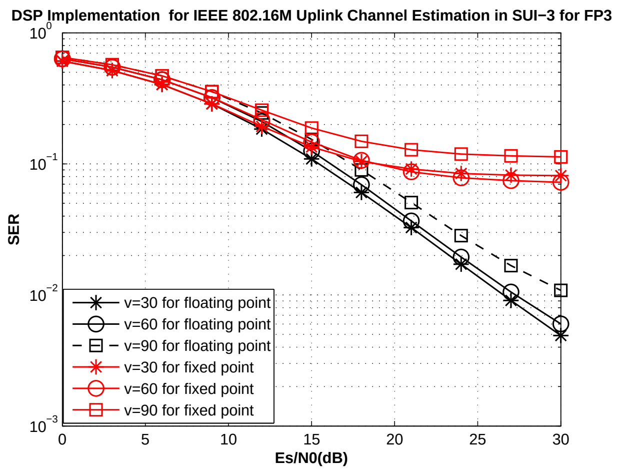
(a)



(b)

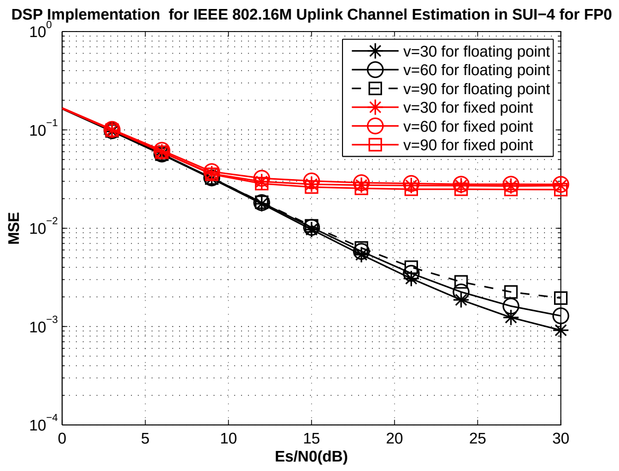


(c)

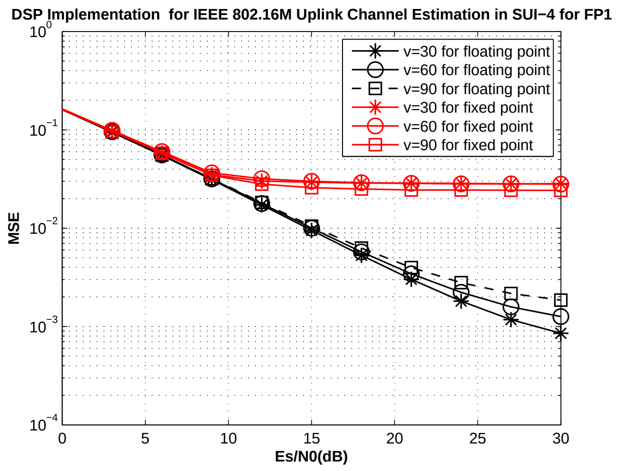


(d)

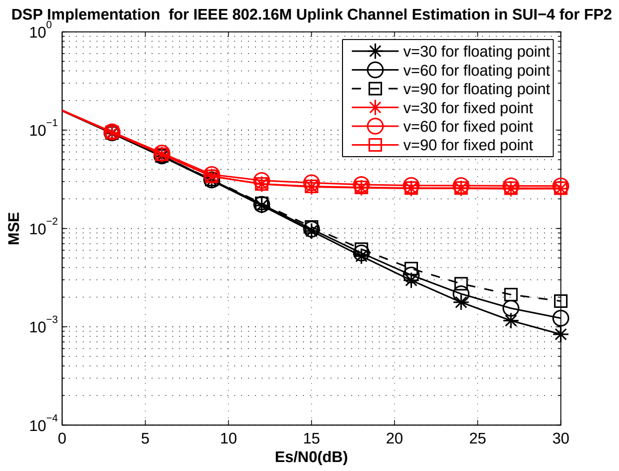
Figure G.4: Fixed-point channel estimation SER use QPSK modulation for different frequency partitions at different velocities in SUI-3 channel for IEEE 802.16m uplink, where the speed v is km/h.



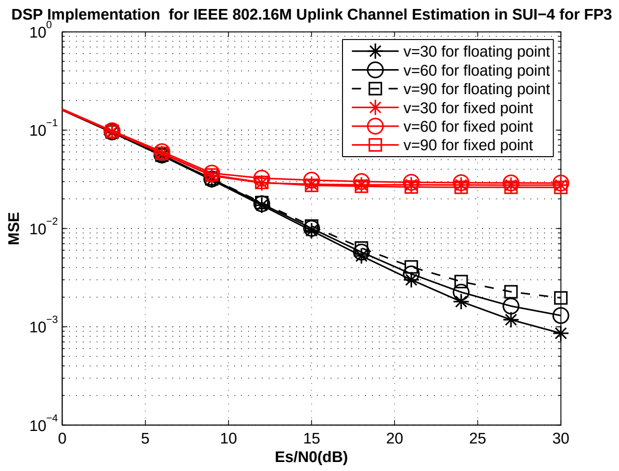
(a)



(b)

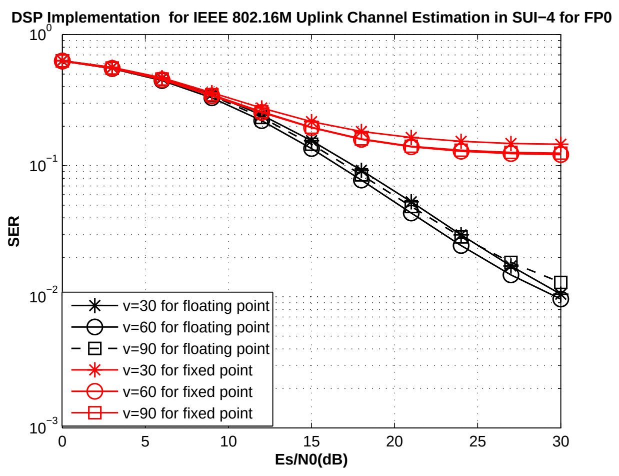


(c)

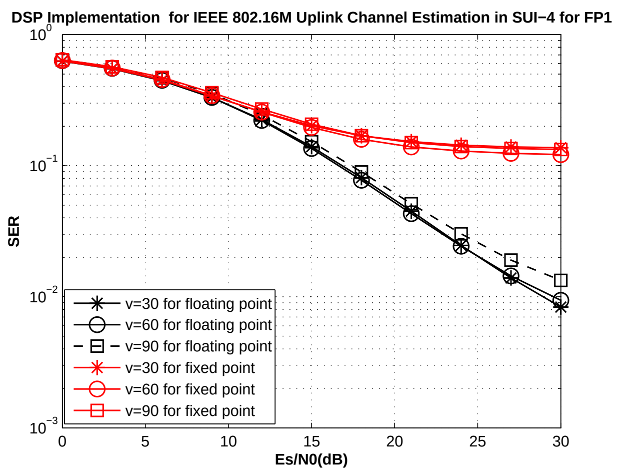


(d)

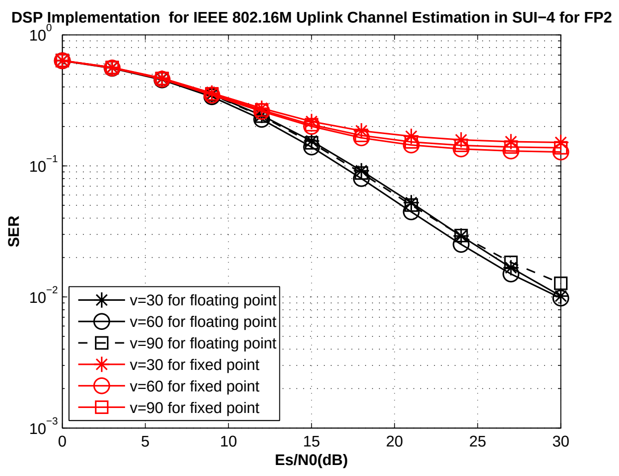
Figure G.5: Fixed-point channel estimation MSE use QPSK modulation for different frequency partitions at different velocities in SUI-4 channel for IEEE 802.16m uplink, where the speed v is km/h.



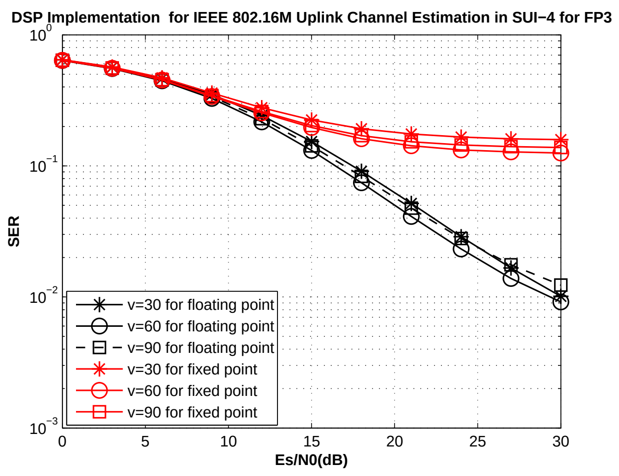
(a)



(b)

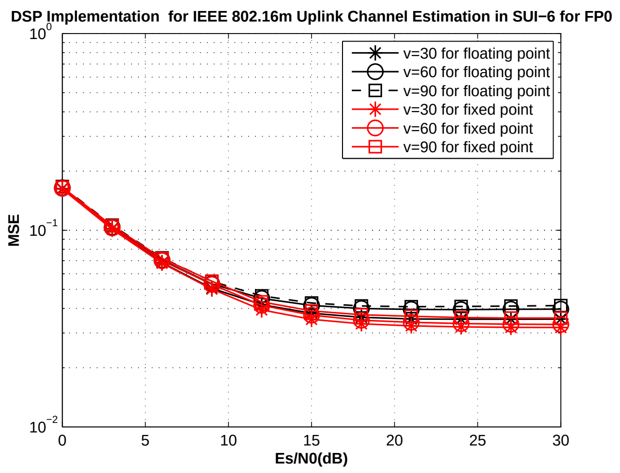


(c)

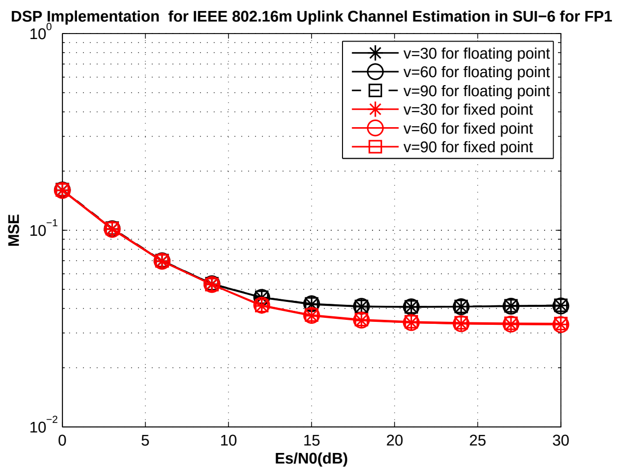


(d)

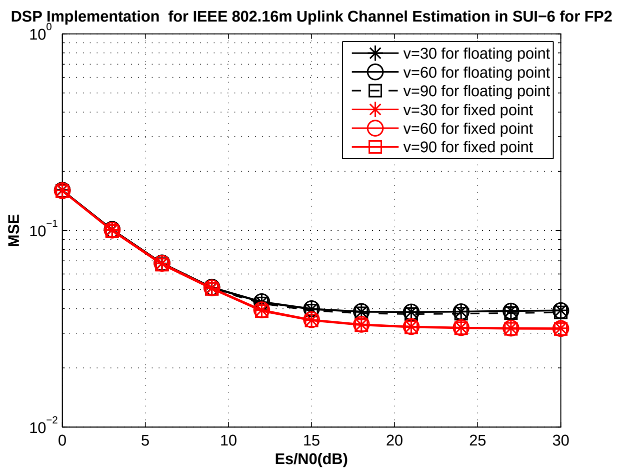
Figure G.6: Fixed-point channel estimation SER use QPSK modulation for different frequency partitions at different velocities in SUI-4 channel for IEEE 802.16m uplink, where the speed v is km/h.



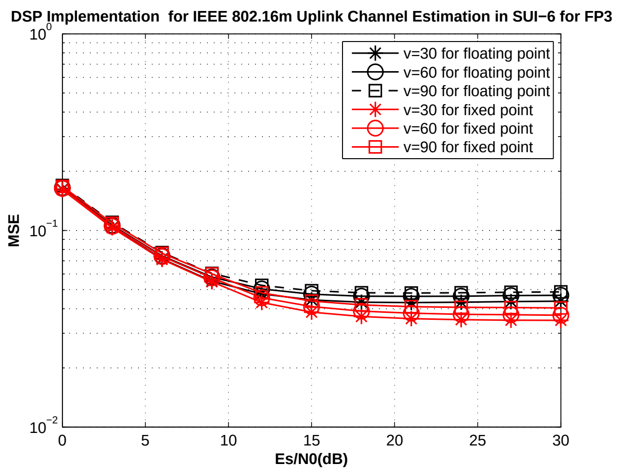
(a)



(b)

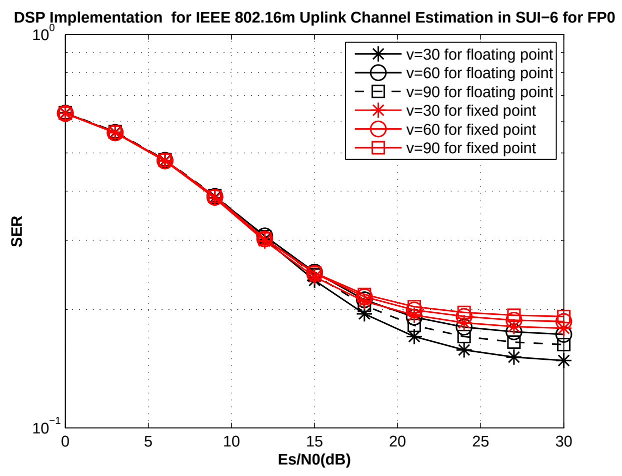


(c)

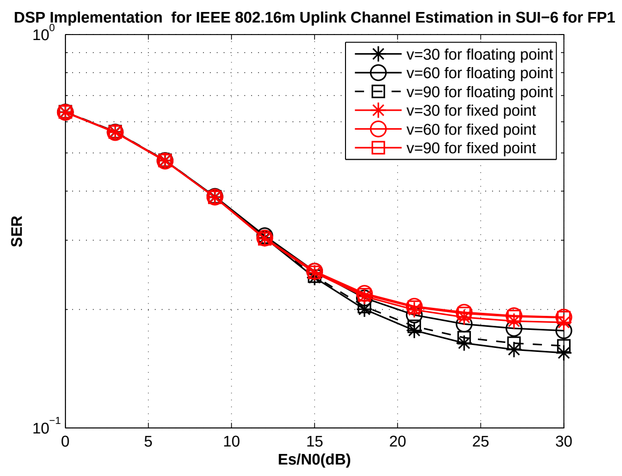


(d)

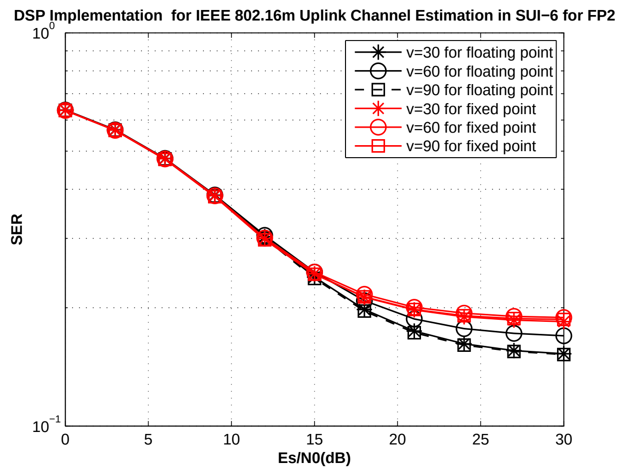
Figure G.7: Fixed-point channel estimation MSE use QPSK modulation for different frequency partitions at different velocities in SUI-6 channel for IEEE 802.16m uplink, where the speed v is km/h.



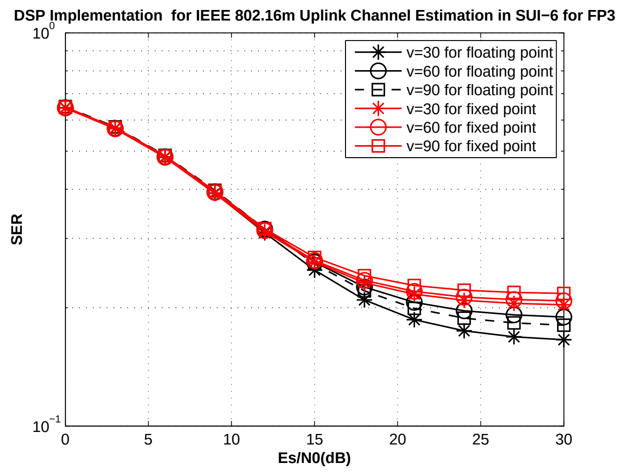
(a)



(b)



(c)



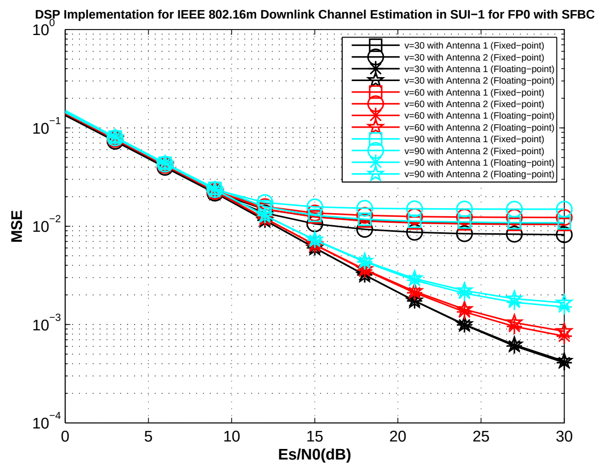
(d)

Figure G.8: Fixed-point channel estimation SER use QPSK modulation for different frequency partitions at different velocities in SUI-6 channel for IEEE 802.16m uplink, where the speed v is km/h.

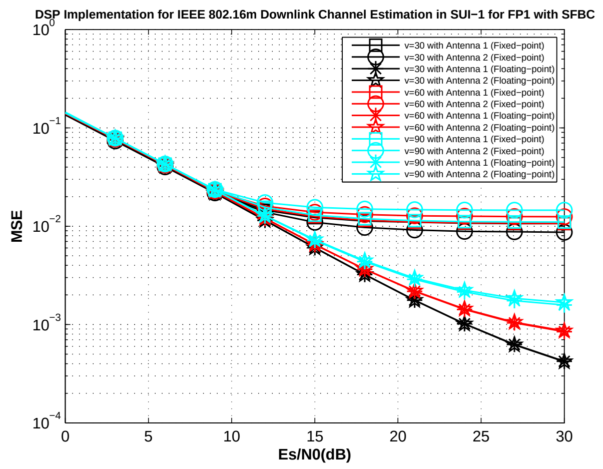
Appendix H

Additional Uplink SFBC MIMO Simulation Result for Fixed-Point

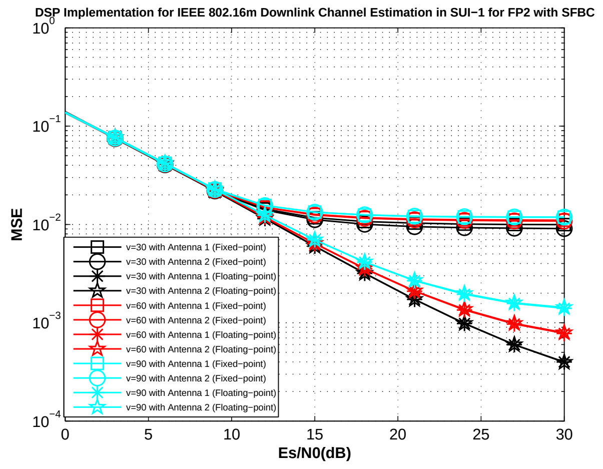
In this appendix, we place additional uplink SFBC MIMO numerical result used fixed-point. Figs. H.1 to H.8 shows the SER and MSE for each SUI channel.



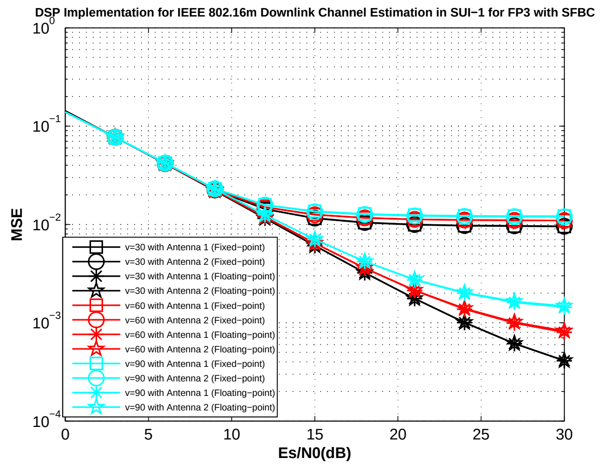
(a)



(b)

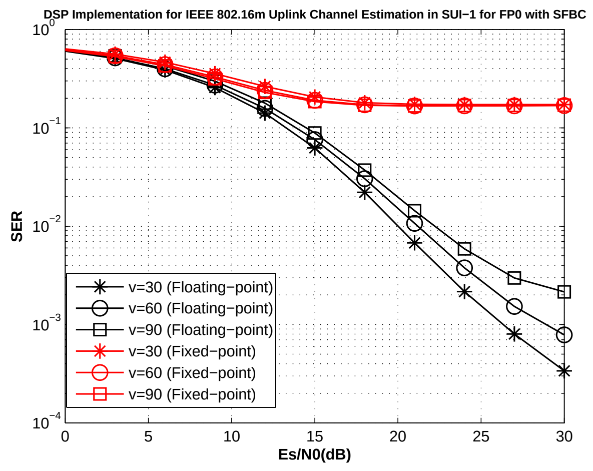


(c)

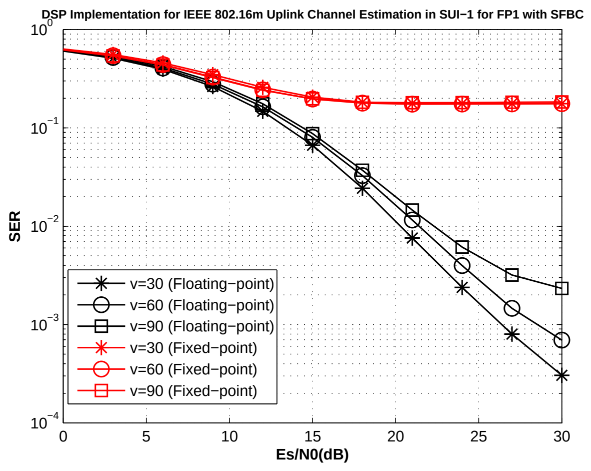


(d)

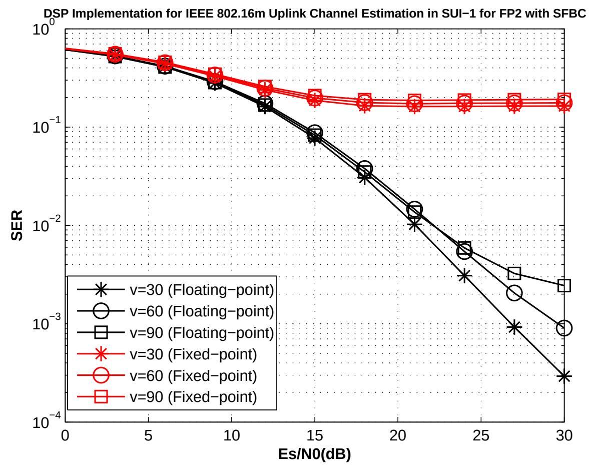
Figure H.1: Fixed-point channel estimation MSE use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-1 Fixed-point channel for IEEE 802.16m uplink, where the speed v is km/h.



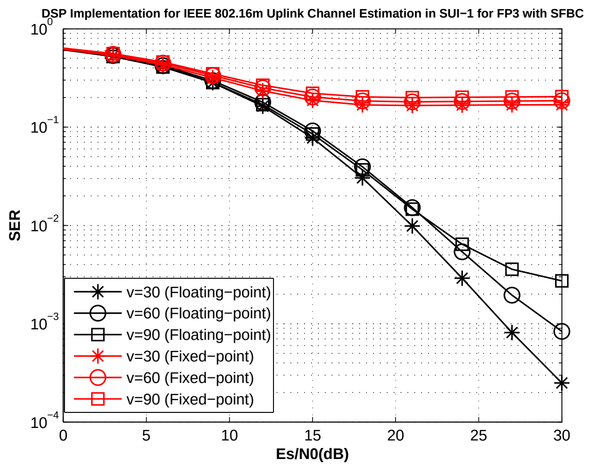
(a)



(b)

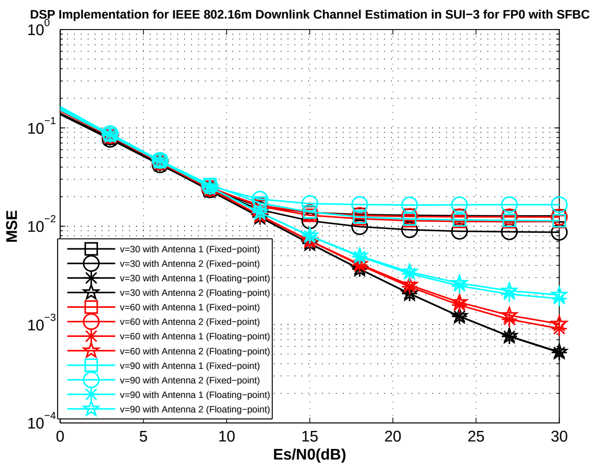


(c)

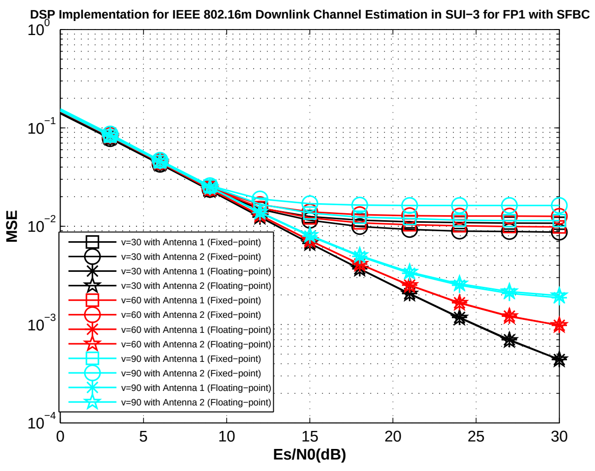


(d)

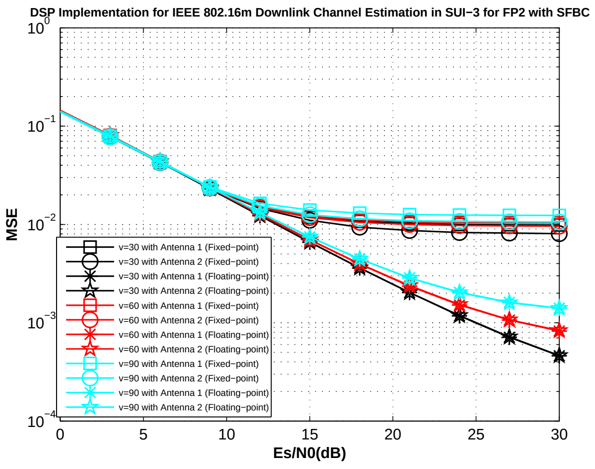
Figure H.2: Fixed-point channel estimation SER use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-1 Fixed-point channel for IEEE 802.16m uplink, where the speed v is km/h.



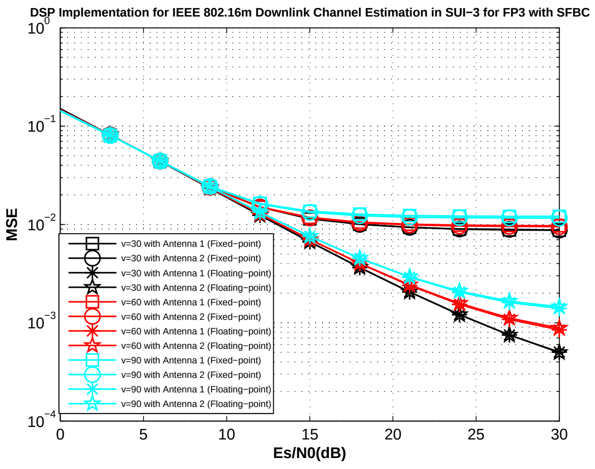
(a)



(b)

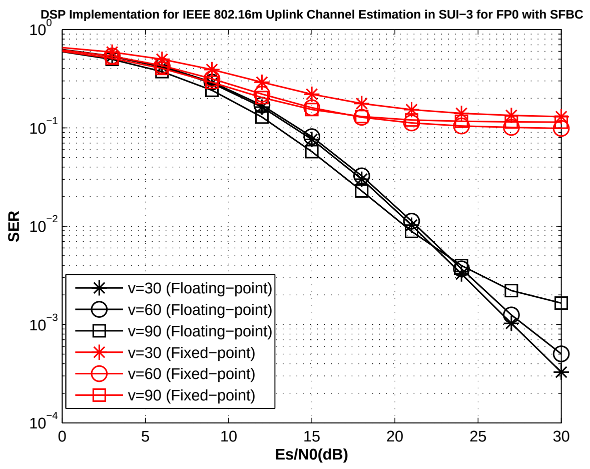


(c)

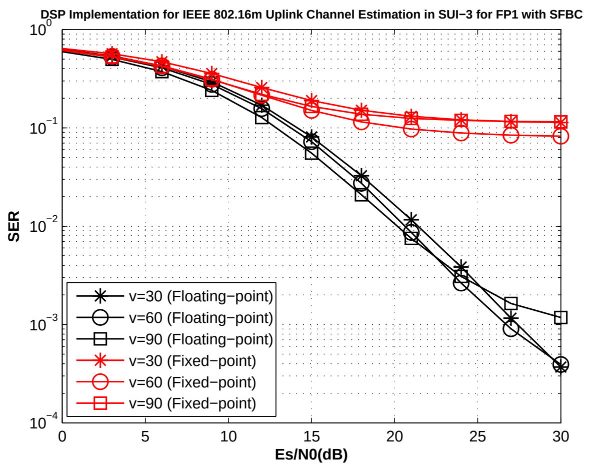


(d)

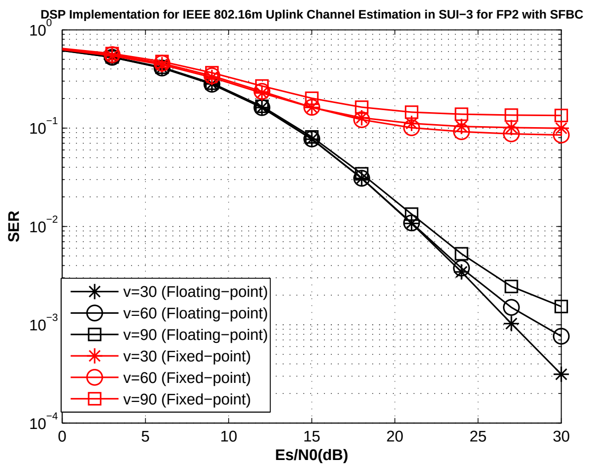
Figure H.3: Fixed-point channel estimation MSE use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-3 Fixed-point channel for IEEE 802.16m uplink, where the speed v is km/h.



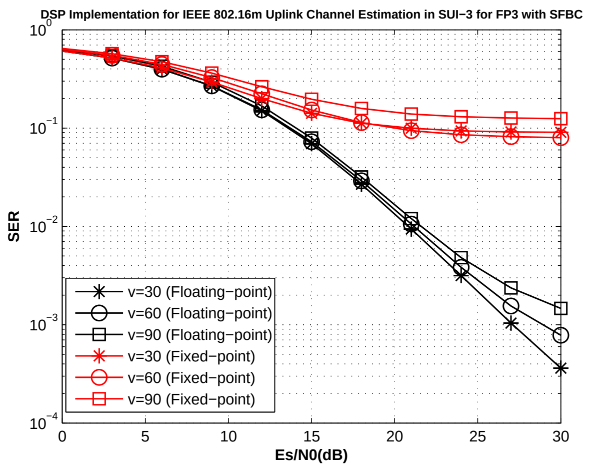
(a)



(b)

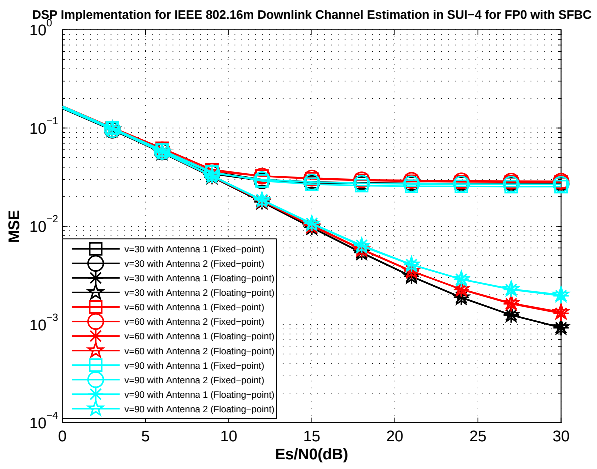


(c)

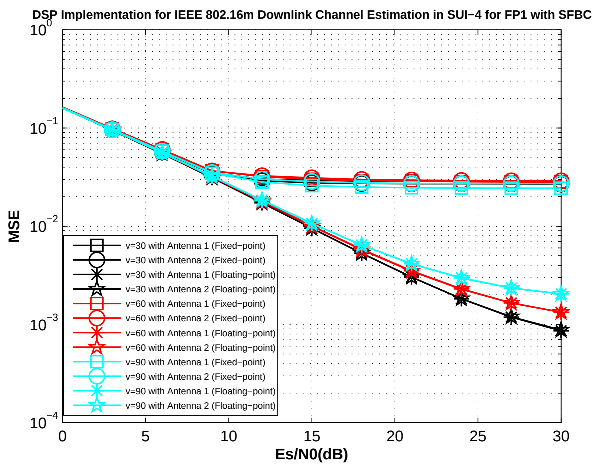


(d)

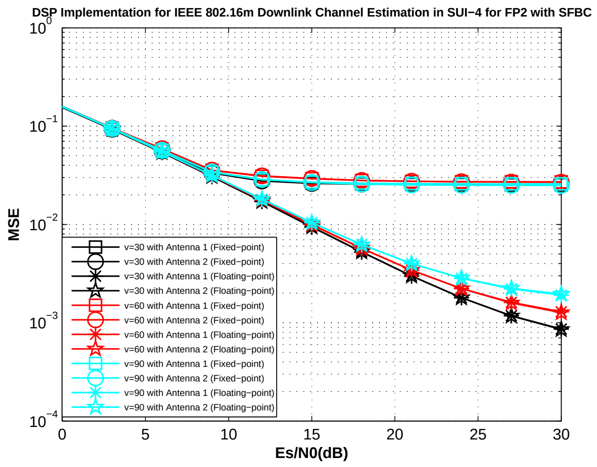
Figure H.4: Fixed-point channel estimation SER use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-3 Fixed-point channel for IEEE 802.16m uplink, where the speed v is km/h.



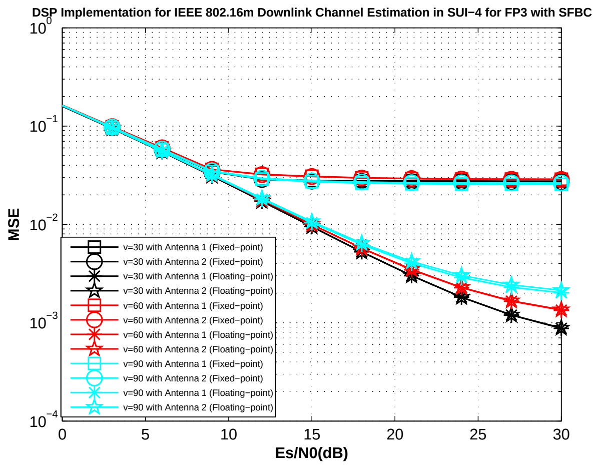
(a)



(b)

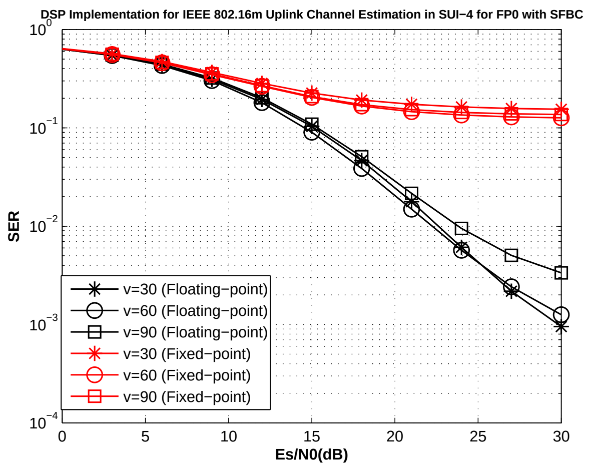


(c)

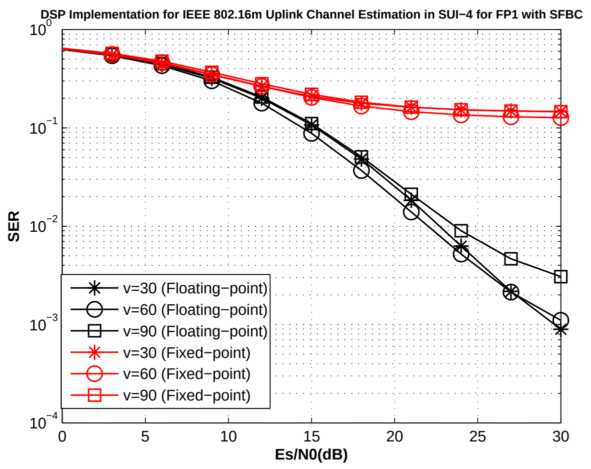


(d)

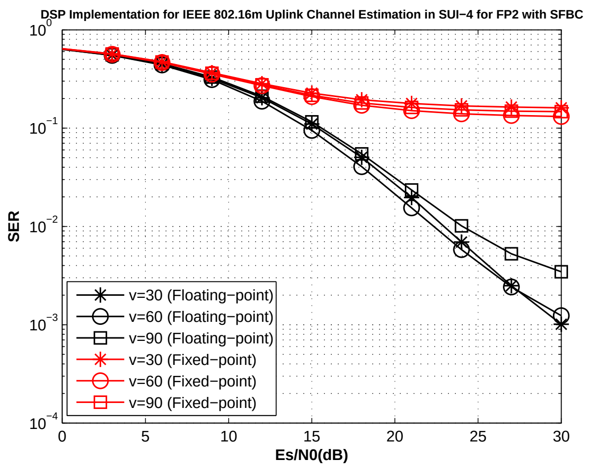
Figure H.5: Fixed-point channel estimation MSE use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-4 Fixed-point channel for IEEE 802.16m uplink, where the speed v is km/h.



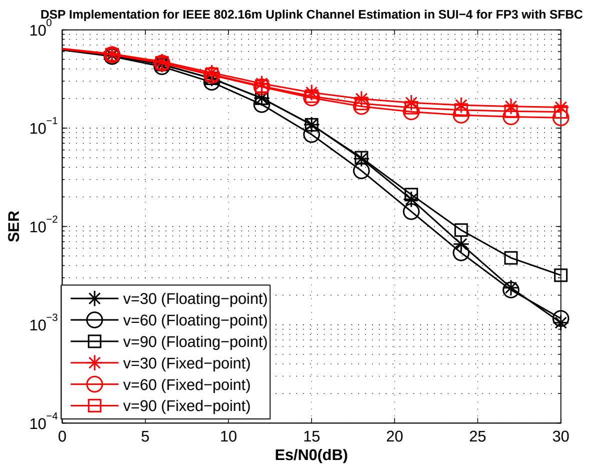
(a)



(b)

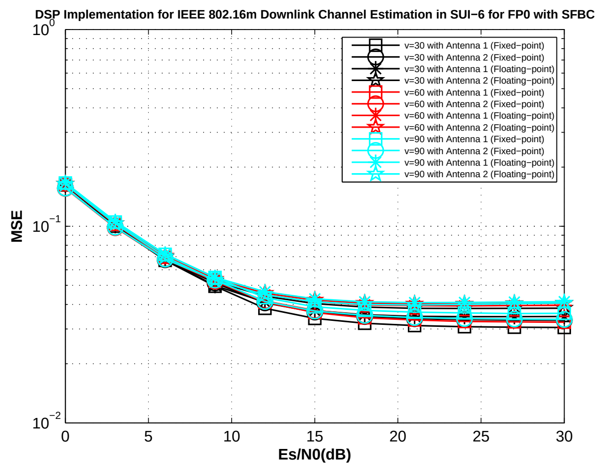


(c)

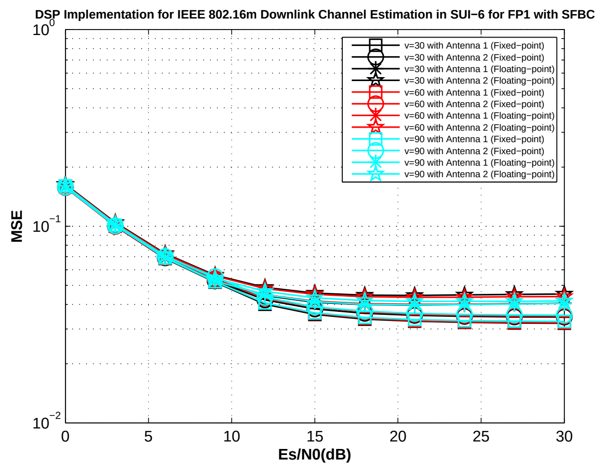


(d)

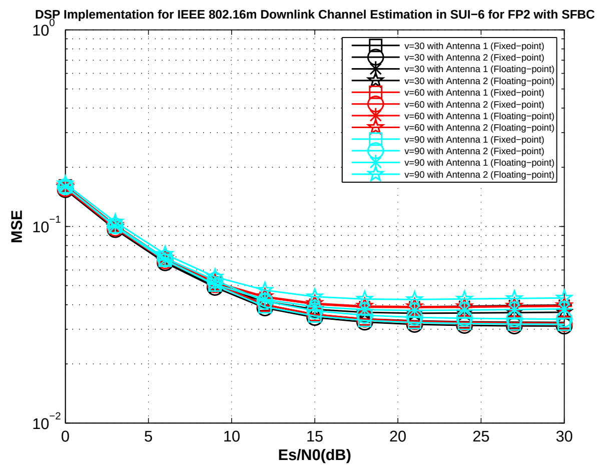
Figure H.6: Fixed-point channel estimation SER use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-4 Fixed-point channel for IEEE 802.16m uplink, where the speed v is km/h.



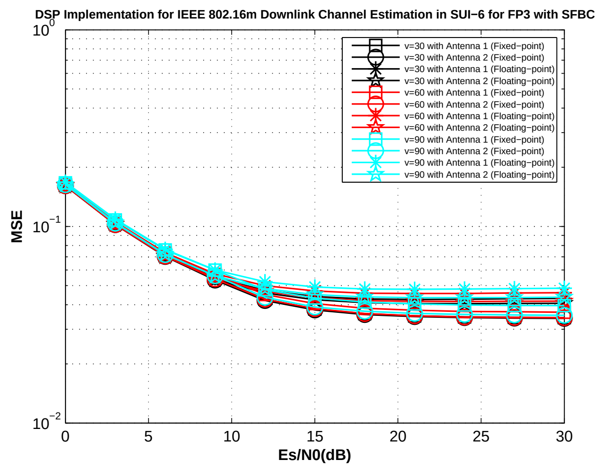
(a)



(b)

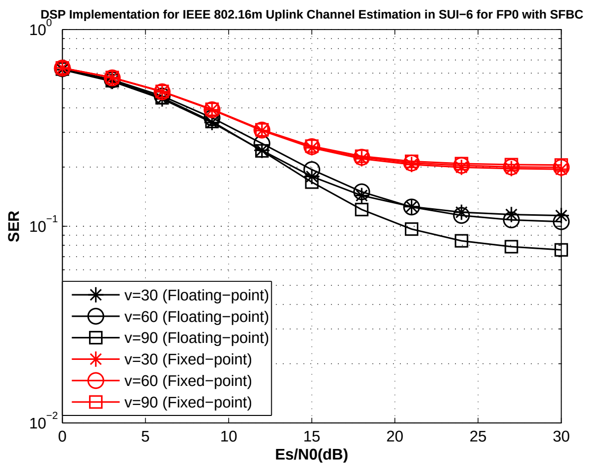


(c)

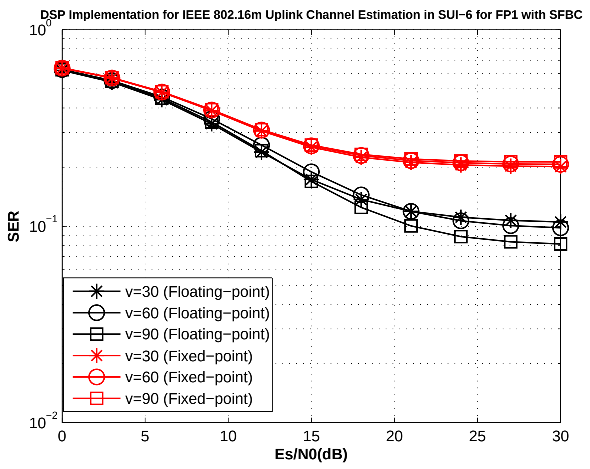


(d)

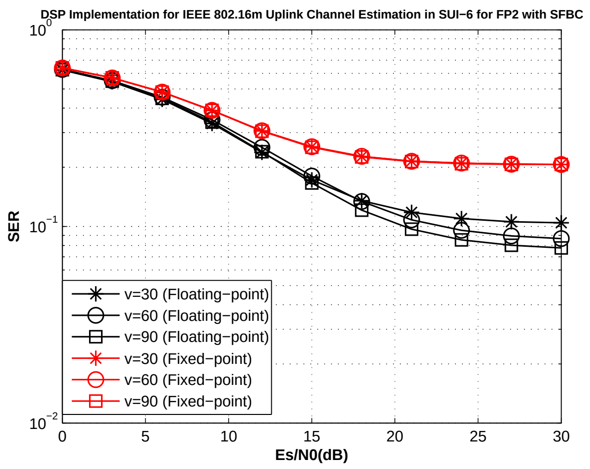
Figure H.7: Fixed-point channel estimation MSE use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-6 Fixed-point channel for IEEE 802.16m uplink, where the speed v is km/h.



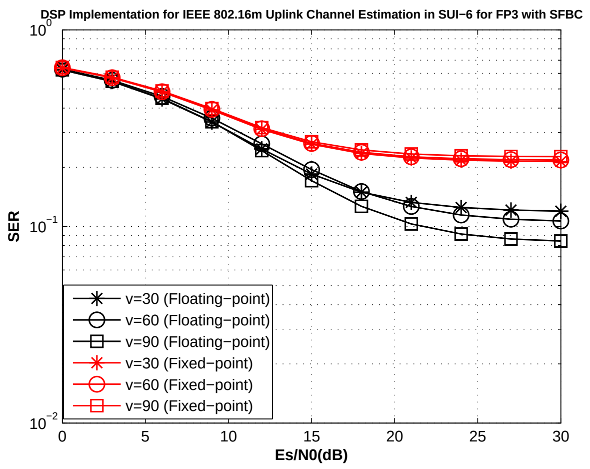
(a)



(b)



(c)



(d)

Figure H.8: Fixed-point channel estimation SER use QPSK modulation with SFBC for different frequency partitions at different velocities in SUI-6 Fixed-point channel for IEEE 802.16m uplink, where the speed v is km/h.

Bibliography

[1] Kai-Wei Lu, "Initial downlink synchronization for IEEE 802.16m," M.S. thesis, Industrial Technology R & D Master Program on Communication Engineering, National Chiao Tung University, Hsinchu, Taiwan, R.O.C., Feb. 2010.

[2] Chih-Wei Wang, "LMMSE channel estimation for IEEE 802.16e and 802.16m OFDMA DownLink," M.S. thesis, Department of Electronics Engineering and Institute of Electronics, National Chiao Tung University, Hsinchu, Taiwan, R.O.C., June 2009.

[3] K.-C. Hung and D. W. Lin, "Pilot-based LMMSE channel estimation for OFDM systems with power-delay profile approximation," *IEEE Trans. Veh. Technology*, vol. 59, no. 1, pp. 150–159, Jan. 2010

[4] H. Stark and J. W. Woods, *Probability and Random Processes, 3rd ed.* Upper Saddle River, NJ: Prentice-Hall, 2002.

[5] P. Hoeher, S. Kaiser, and P. Robertson, "Two-dimensional pilot-symbol-aided channel estimation by Wiener filtering," in *Proc. IEEE Int. Conf Acoust. Speech Signal Processing*, vol. 3, 1997, pp. 1845–1848.

[6] O. Edfors, M. Sandell, J.-J. van de Beek, S. K. Wilson, and P. O. Borjesson, "OFDM channel estimation by singular value decomposition," *IEEE Trans. Commun.*, vol. 46, no. 7, pp. 931–939, July 1998.

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

[8] L. Wilhelmsson, B. Bernhardsson, and L. Anderson, "Channel estimation by adaptive interpolation," U.S. patent 7,433,433, Oct. 7, 2008.

[9] 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.

[10] 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.

[11] IEEE 802.16 Task Group m, *Part 16: Air Interface for Broadband Wireless Access Systems — Advanced Air Interface (working document).* Doc. no. IEEE 802.16m/D9, October 6, 2010.

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

[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] V. Erceg *et al.*, "Channel models for fixed wireless applications," IEEE standards contribution no. IEEE 802.16.3c-01/29r4, July 2001.

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

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

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

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

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

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

[21] Hsiung Wei Chang, Liang Chen and Kuang Hui Hsu, *DSP Chip: Fundamental and Application.* (In chinese) Wu Nan, 2004.

[22] B. Farhang-Boroueny, *Adaptive Filters: Theory and Applications.* Wiley, 1998.