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 初始下行同步之數位訊號處理器實現

Digital Signal Processor Implementation of Initial Downlink Synchronization for IEEE 802.16m

研 究 生：陳威宇

指導教授：林大衛 博士

中 華 民 國 一 〇 〇 年 九 月

IEEE 802.16m 初始下行同步之數位訊號處理器實現

Digital Signal Processor Implementation of Initial Downlink Synchronization for IEEE 802.16m

研究生：陳威宇　　Student: Wei-Yu Chen

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

September 2011

Hsinchu, Taiwan, Republic of China

中華民國一〇〇年九月

IEEE 802.16m 初始下行同步之數位訊號處理器實現

研究生：陳威宇　　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

本篇論文研究方向為針對 IEEE 802.16m 中初始下行同步裡，實作於數位訊號處理平台上的議題探討。

當一個行動電話開始要進入網路的時候，我們必須與基地台做初始的同步。在初始的同步中，包含了符元時間偏移、載波偏移和前置符元序號(preamble index)需要同步估測。我們利用前置符元的功率較一般資料符元(data symbol)大的特性做功率移動累加，藉由找到累加結果的峰值來估測前置符元的起始位置。由此起始位置向後推算一個符元長度以當作我們所估測出來的前置符元，而與真實的前置符元存在一個相位性錯誤(phase error)。我們利用此估測出來的前置符元推導其近似最大可能性估測(quasi maximum likelihood)以求得小數部分載波偏移和導出前置符元的通道估測的式子。我們在頻域上將此通道估測的式子經由估測出來的小數部分載波偏移補償之後，代入由合理範圍的整數部分載波偏移和不同的前置符元而得到的通道脈衝響應。再計算這些通道脈衝響應不同的精準符碼時間偏移序號64 點功率和並且選出最大的那一個，其所在的整數部分載波偏移、前置符元和精準符碼時間偏移序號即為此聯合估測的結果。

接著，我們把此演算法實作於浮點運算與定點運算，以及比較兩者的效能。最後，把定點運算實現於數位訊號處理平台，並且最佳化我們的程式速度，減少運算複雜度，雖然定點運算會早成效能的衰減，但是結果依然可以接受。

Digital Signal Processor Implementation of Initial Downlink Synchronization for IEEE 802.16m

Student：Wei-Yu Chen　　Advisor：Dr. David W. Lin

Department of Electronics Engineering

Institute of Electronics

National Chiao Tung University

Abstract

In this thesis, the research focus on initial downlink synchronization of IEEE 802.16m, and discuss the implementation issue of DSP.

When a mobile station entering to the network, it needs to perform initial synchronization, including of symbol timing offset, carrier frequency offset and preamble index. We utilize the trait which the power of preamble is larger than it of the common data symbol to compute the moving power sum, and then estimate the left boundary of preamble by finding out the peak value of moving power sum. A symbol period from this estimated boundary is regarded as the estimated preamble, which has a phase noise with the exact preamble. We derive the quasi maximum likelihood estimation from the likelihood function of the estimated preamble to obtain fractional carrier frequency offset (FCFO) and the formula of channel estimation. After compensating the estimated fractional carrier frequency offset to the formula of channel estimation, we substitute several reasonable integral carrier frequency offsets (ICFOs) and primary advanced preambles (PA-Preambles) into this formula and obtain channel impulse responses (CIRs). After that, we compute different fine timing offset index 64-points power sum of these CIRs and find out the peak value whose ICFO, PA-Preamble index, and fine timing offset index are regarded as the result of

the joint estimation.

In order to compare the performance, we implement the algorithm into the floating-point and fixed-point version. In the end, we modified the fixed-point version on the digital signal processor platform, and optimize the speed of our programs to reduce operation complexity. Although the performance is degraded because of fixed-point modification, the results still can be accepted.

誌謝

在交通大學裡度過的這兩年多來要感謝的人太多了，除了謝天以外，首先我想感謝林大衛老師，在這段時間以來的照顧以及在專業知識上給於了很多的幫助，如果沒有老師的話，也不能這麼順利的寫完這篇論文。老師同時教導了我很多做研究的方法及態度，讓我終身受用。

接下來要感謝的是 CommLab 博班的學長，鴻志、彰哲、俊榮、伯森、世璞、朝雄，在課業或研究上也給予我多的幫助以及意見。有你們的幫助，讓我在碩班生涯中成長了很多，也學習到很多。

再來要感謝的是曉盈、俊言、卓翰、智凱、強丹、郁婷、偵源、書瑋、兆軒、凱翔、峻利、復凱、婉瑜、怡茹、頌文以及學弟妹，在這段時間裡陪我吃喝玩樂，不管是課業上的討論還是聊天打屁，有你們的陪伴，讓我碩班生涯添加了很多色彩。

最後要感謝的是我的家人以及馮捷，感謝一直來的支持與鼓勵，在我遇到挫折時包容我、體諒我，讓我在這段期間能夠無後顧之憂，完成這篇論文拿到碩士學位。

本篇論文獻給所有曾經幫助過我的人，因為有你們，讓我成長茁壯，才能成就今天的我。

陳威宇

民國一〇〇年十月 於風城・新竹

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The IEEE 802.16m standard activity is a response to the ITU-R's plan for the fourth-generation mobile communication standard IMT-Advanced, wherein it is specified that the data rate should be at least 100 Mbps in an environment with high mobility and 1 Gbps in a static environment. Since December 2006, the IEEE 802.16 standards group has set up the IEEE 802.16m (i.e., Advanced WiMAX or WiMAX 2) task group. The new frame structure developed by IEEE 802.16m is such that it can be compatible with IEEE 802.16e, reduce communication latency, support relay and coexist with other radio access techniques, so that it can become a promising candidates for 4G.

In this work, we study the digital signal processor (DSP) software implementation of a previously developed initial downlink synchronization method for IEEE 802.16m system with a time division duplex (TDD) mode [1,2]. The initial downlink synchronization involves frequency offset correction, timing recovery and bandwidth detection. In the procedure that we have developed, channel estimation is also obtained simultaneously.

Our DSP implementation uses Texas Instrument (TI) fixed-point DSP platform. We accelerate the execution speed of the programs and utilize difference optimization techniques to reduce the computational complexity.

This thesis is organized as follows. We first introduce the IEEE 802.16m standard in chapter 2. In chapter 3, we present the synchronization algorithm. Chapter 4 introduces

the DSP implementation platform. We discuss the DSP optimization methods and presents the optimization results in chapter 5. Finally, the conclusion is given in chapter 6, where we also point out some potential future work.

The contributions of this work are as follows:

- We modify the program from Matlab code to C code.
- We convert the code to fixed-point for implementation on DSP.
- We employ various optimization techniques to accelerate the execution speed of the programs in the DSP implementation.

Chapter 2

Overview of the IEEE 802.16m Standard

The IEEE 802.16m standard is based on orthogonal frequency division multiplexing (OFDM) and orthogonal frequency division multiple access (OFDMA). In this chapter, we introduce some basic concepts regarding OFDM and OFDMA first. Then we give an overview of the draft IEEE 802.16m standard. For simplicity, we only introduce the specifications that are most relevant to our study. For example channel coding, MAP messages, transmit diversity, etc., are ignored in this introduction.

2.1 Overview of OFDMA [3], [4]

OFDMA is considered one most appropriate scheme for future wireless systems, including 4G broadband wireless networks. In a typical OFDMA system, users may simultaneously transmit their data by modulating mutually exclusive sets of orthogonal subcarriers, so that their signal are 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 subbands for the data transmission. Usually, pilot symbols are employed for the estimation of channel state information (CSI) within the subband. The IEEE 802.16m is one example of such systems. Figure 2.1 shows an uplink (UL) OFDMA system in which users simultaneously transmit to the base station (BS).

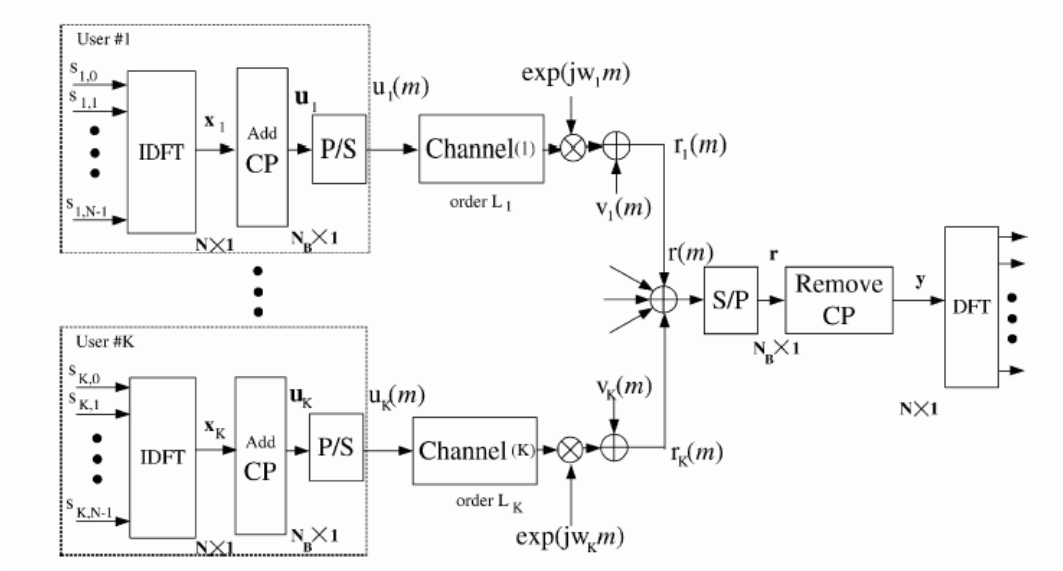

Figure 2.1: Discrete-time model of the baseband OFDMA system (from [3]).

2.1.1 Cyclic Prefix

Cyclic prefix (CP) or guard time is used in OFDM and OFDMA systems to overcome the intersymbol and intercarrier interference problems. The multiuser channel is usually substantially invariant within one-block (or one-OFDM(A)-symbol) duration. The channel delay spread plus symbol timing mismatch is usually smaller than the CP duration. In these conditions, users do not interfere with each other in the frequency domain when their signal are properly synchronized in frequency and in time.

A CP is a copy of the last part of the OFDM(A) symbol, as illustrated in Fig. 2.2. A copy of the last T_g of the useful symbol period is used to collect multipaths from the preious symbol to maintain the orthogonality among subcarriers. However, the transmitter energy increases with the length of the guard time while the receiver energy remains the same, because the CP is discarded in the receiver. So there is a $10\log_{10}(1 - T_g/(T_b + T_g))$ dB loss in power efficiency compared to traditional single-carrier system.

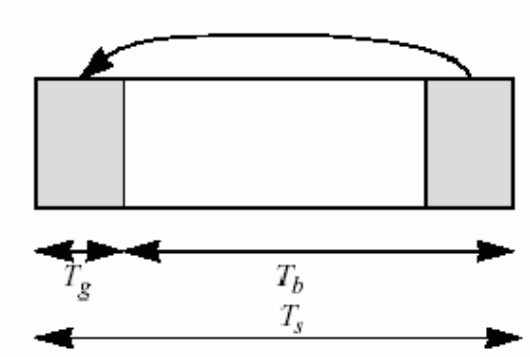

Figure 2.2: OFDMA symbol time structure (Fig. 478 in [5]).

2.1.2 Discrete-Time Baseband Equivalent System Model

The material in this subsection is mainly taken from [4]. 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.

Let the discrete-time baseband channel consists of L multipath components as

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

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 $\mathbf{H} = [H_0, H_1, ..., H_{K-1}]^T$, $\mathbf{h} = [h_0, ..., h_{L-1}, 0, ..., 0]^T$ and $\mathbf{F}$ is 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 is given by

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

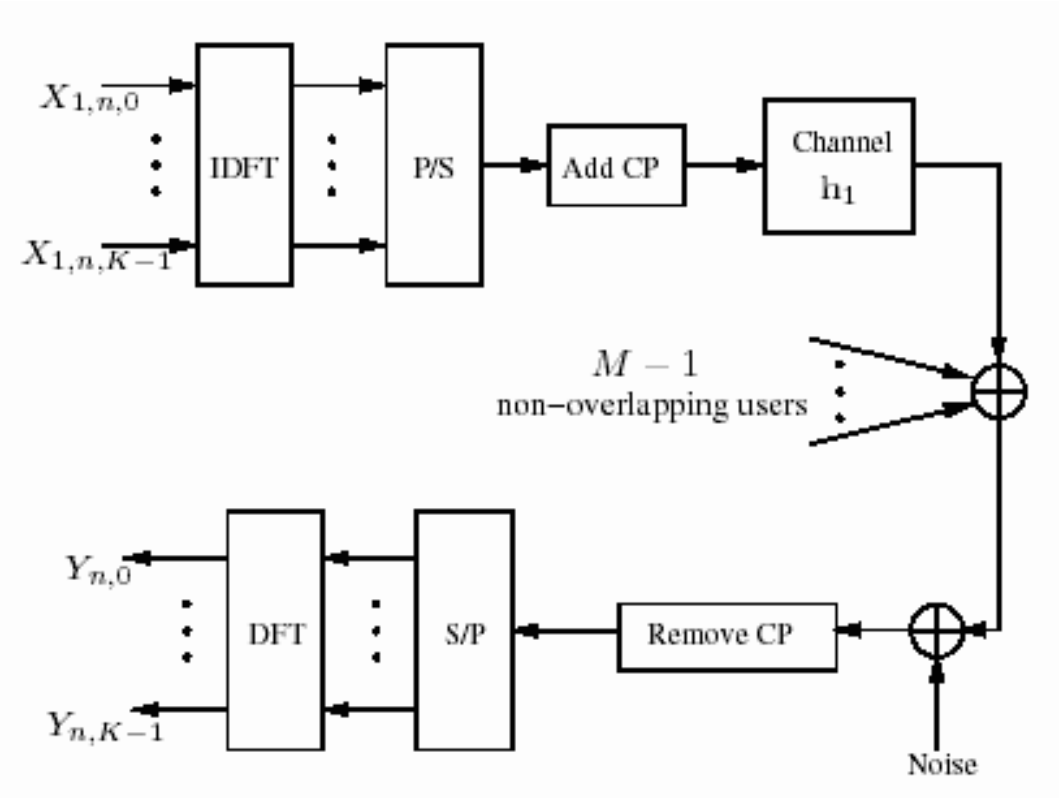

Figure 2.3: Discrete-time baseband equivalent of an OFDMA system with M transmitting users (from [4]).

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

2.2 Basic OFDMA Signal Structure in IEEE 802.16m [5]

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. The AAI 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 both DL and UL. The material of this section is mainly taken from [5].

2.2.1 Resource Units

The OFDMA physical layer (PHY) of IEEE 802.16m organizer the subcarrier and OFDMA symbols into resource units as described below.

- 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, where $P_{sc} = 18$ and $N_{sym} = 6$ for type-1 subframes, $N_{sym} = 7$ for type-2 subframes, and $N_{sym} = 5$ for type-3 subframes. Table 2.2.1 illustrates the PRU sizes for different subframe types. A logical resource unit (LRU) is the basic logical unit for distributed and localized resource allocations. An LRU is $P_{sc} \cdot N_{sym}$ 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 data subcarriers in an LRU depends on the number of allocated pilots.

- Distributed resource unit: A distributed resource unit (DRU) contains a group of subcarriers 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: The 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.

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

2.2.2 Basic Categories of Subcarrier

An 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, including subcarriers in the guard bands and the DC subcarrier.

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

2.2.3 Primitive Parameters 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 Figs. 2.4 and 2.5 for each nominal bandwidth.
- G: This is the ratio of CP time to "useful" time, i.e., T_{cp}/T_s. The following values are supported: 1/16, 1/8, and 1/4.

The following parameters are defined in terms of, i.e., derived from 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}.

2.2.4 Frame Structure

Fig. 2.6 illustrate the AAI basic frame structure. Each 20-ms superframe is divided into four 5-ms radio frames. When using the same OFDMA parameters as in Figs. 2.4 and 2.5 with channel bandwidth of 5, 10, or 20 MHz, each 5-ms radio frame further consists of eight subframes, when $G = 1/8$ and $1/16$. With channel bandwidth of 8.75 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 forms the unit of assignment for either DL or UL transmission. There are four types of subframes:

- Type-1 subframe consists of six OFDMA symbols.
- Type-2 subframe consists of seven OFDMA symbols.
- Type-3 subframe consists of five OFDMA symbols.

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

Figure 2.4: OFDMA parameters (Table 794 in [5]).

- 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 802.16e OFDMA) 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 subframe (i.e., the default transmission time interval [TTI] transmission) or contiguous multiple subframes (i.e., the long TTI transmission). The

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	38	42	42
		TTG + RTG (μs)	199.98	200	136	199.98	199.98
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 sub-frame.			24	48	48	48	96

Figure 2.5: More OFDMA parameters (Table 794 in [5]).

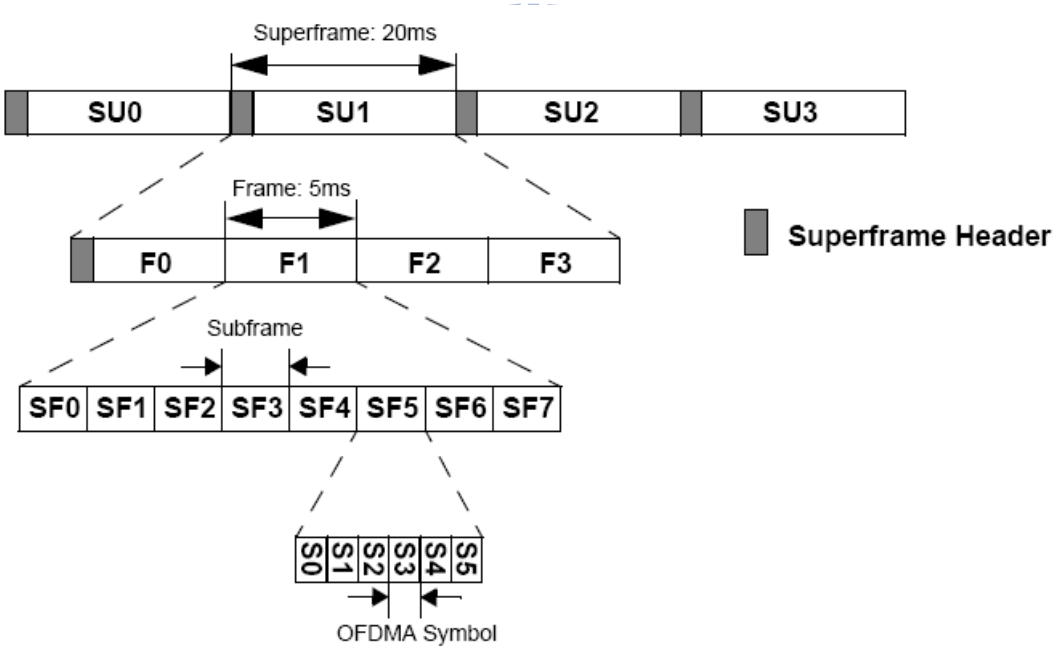

Figure 2.6: Basic frame structure for 5, 10 and 20 MHz channel bandwidths (Fig. 480 in [5]).

long TTI in FDD shall be 4 subframes for both DL and UL. The long TTI in TDD shall be the whole DL (UL) subframes for DL (UL) in a frame. Every superframe shall contain a superframe header (SFH). The SFH shall be located in the first DL subframe of the superframe and shall include broadcast channels.

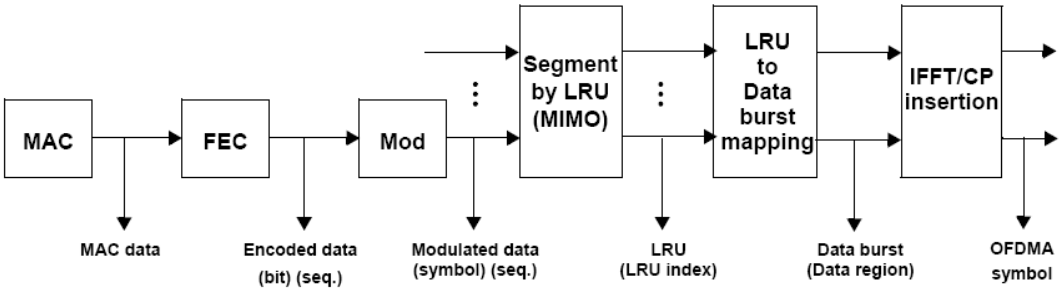

Figure 2.7: Definition of terms on the transmission chain (Fig. 479 in [5]).

2.2.5 Transmitted Signal

The transmitted RF signal, as a function of time, during any OFDMA symbol to given by

$$s(t) = \Re\{e^{j2\pi f_c} \sum_{k=-(N_used-1)/2, k\neq 0}^{(N_used-1)/2} c_k \cdot e^{j2\pi \triangle f(t-T_g)}\} \tag{2.4}$$

where

- t is the time, elapsed since the beginning of the OFDMA symbol, with $0 < t < T_s$,
- c_k is a complex number giving the QAM modulated value of the data to be transmitted on the subcarrier whose frequency offset index is k, during the OFDMA symbol,
- T_g is the guard time,
- $\triangle f$ is the subcarrier frequency spacing, and
- f_c is the carrier frequency.

2.2.6 Transmission Chain

The terms related to the transmission chain are defined as illustrated in Fig. 2.7.

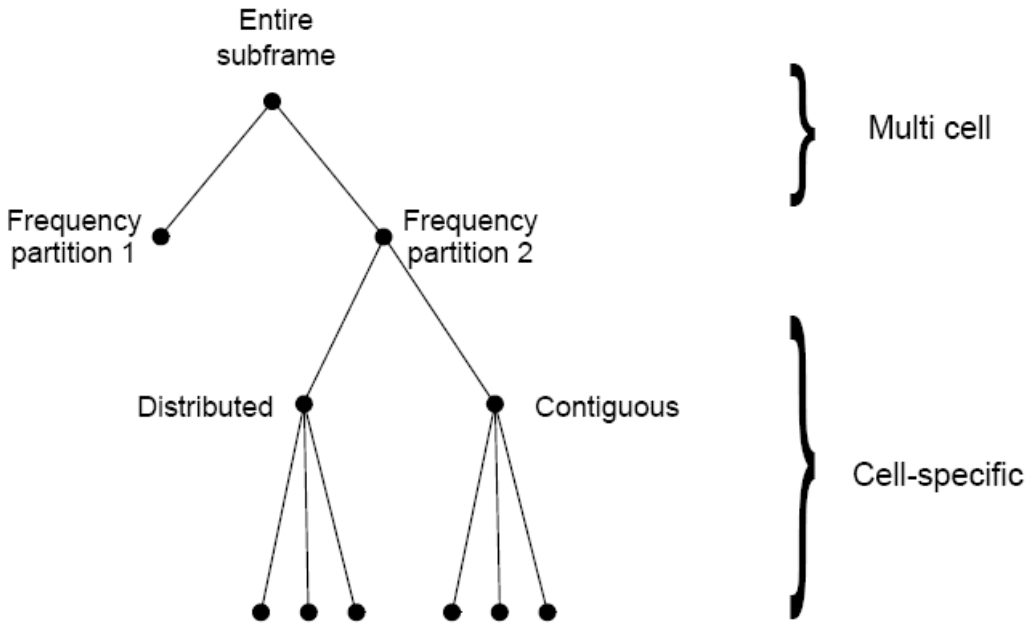

Figure 2.8: Example of downlink physical structure (Fig. 499 in [5]).

2.3 Downlink Transmission in IEEE 802.16m [5]

Again, this section is mainly taken from [5]. Each DL subframe is divided into 4 or fewer frequency partitions, each partition consisting of a set of PRUs 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. 2.8 illustrates the DL physical structure in an example of two frequency partitions with frequency partition 2 including both contiguous and distributed resource allocations.

2.3.1 Subband Partitioning

The PRUs are first subdivided into subbands and minibands where a subband comprises of N_1 adjacent PRUs and a miniband comprises N_2 adjacent PRUs, where $N_1 = 4$ and $N_2 = 1$. Subbands are suitable for frequency selective allocations 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 is denoted by K_{SB}. The number of PRUs allocated to subbands is denoted by L_{SB}, where $L_SB = N_1 K_{SB}$. A 5, 4 or 3-bit field called Downlink Subband Allocation Count (DSAC) determines the value of K_{SB} depending on FFT size. The DSAC is transmitted in the SFH. The remaining 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 $L_MB = N_2 K_{MB}$. The total number of PRUs is denoted as N_PRU where $N_{PRU} = L_{SB} + L_{MB}$. The maximum number of subbands that can be formed is denoted as N_{sub} where $N_{sub} = \lfloor N_{PRU}/N_1 \rfloor$.

Tables 2.2 through 2.4 show the mapping between DSAC and K_{SB} for FFT sizes of 2048, 1024, and 512, respectively. For system bandwidths in range of (10,20], the relation between the system bandwidth and supported number of N_{PRU} is listed in Table 2.5. The mapping between DSAC and K_{SB} is based on Table 2.2, the maximum valid value of K_{SB} is $N_{PRU}/4 - 3$.

For those system bandwidths in range of (5,10], the relation between the system bandwidth and supported number of N_{PRU} is listed in Table 2.6. The mapping between DSAC and K_{SB} is based on Table 2.3, the maximum valid value of K_{SB} is $N_{PRU}/4 - 2$.

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 given by

$$PRU_{SB}[j] = PRU[i], \quad j = 0, 1, ..., L_{SB} - 1, \tag{2.5}$$

Table 2.2: Mapping Between DSAC and K_{SB} for 2048 FFT Size (Table 802 in [5])

DSAC	Number of subbands allocated (K_{SB})	*DSAC*	Number of subbands allocated (K_{SB})
0	0	16	16
1	1	17	17
2	2	18	18
3	3	19	19
4	4	20	20
5	5	21	21
6	6	22	NA.
7	7	23	NA.
8	8	24	NA.
9	9	25	NA.
10	10	26	NA.
11	11	27	NA.
12	12	28	NA.
13	13	29	NA.
14	14	30	NA.
15	15	31	NA.

where

$$i = N_1 \cdot \{\{\lceil \frac{N_{sub}}{N_{sub} - K_{SB}} \rceil \cdot \lfloor \frac{j + L_{MB}}{N_1} \rfloor + \lfloor \lfloor \frac{j + L_{MB}}{N_1} \rfloor \cdot \frac{GCD(N_{sub}, \lceil \frac{N_{sub}}{N_{sub} - K_{SB}} \rceil)}{N_{sub}} \rfloor \} \mod N_{sub}\} + (j + L_{MB}) \cdot N_1 \tag{2.6}$$

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

$$PRU_{MB}[k] = PRU[i], \quad k = 0, 1, ..., L_{MB} - 1, \tag{2.7}$$

Table 2.3: Mapping Between DSAC and K_{SB} for 1024 FFT Dize (Table 803 in [5])

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.

Table 2.4: Mapping Between DSAC and K_{SB} for 512 FFT Size (Table 804 in [5])

DSAC	Number of subbands allocated (K_{SB})	*DSAC*	Number of subbands allocated (K_{SB})
0	0	4	4
1	1	5	NA.
2	2	6	NA.
3	3	7	N.A

where

$$i = \begin{cases} N_1 \cdot \{\lceil \frac{N_{sub}}{N_{sub}-K_{SB}} \cdot \rceil \cdot \lfloor \frac{k}{N_1} \rfloor + \lfloor \lfloor \frac{k}{N_1} \rfloor \cdot \frac{GCD(N_{sub}, \lceil \frac{N_{sub}}{N_{sub}-K_{SB}} \rceil)}{N_{sub}} \rfloor \} \mod N_{sub} \\ \quad +(k) \mod N_1, & K_{SB} > 0, \\ i = k, & K_{SB} = 0. \end{cases} \tag{2.8}$$

Fig. 2.9 illustrates the PRU to PRU_{SB} and PRU_{MB} mappings for a 10 MHz bandwidth with K_{SB} equal to 7.

Table 2.5: OFDMA Parameters for 2048 FFT When Tone Dropping Is Applied (Table 796 in [5])

BW Range, x (MHz)	Number of guard subcarriers		Number of used subcarriers	Number of PRUs (N_{PRU})
	Left	Right		
20.0>x>=19.2	196	195	1657	92
19.2>x>=18.4	232	231	1585	88
18.4>x>=17.5	268	267	1513	84
17.5>x>=16.7	304	303	1441	80
16.7>x>=15.9	340	339	1369	76
15.9>x>=15.0	376	375	1297	72
15.0>x>=14.2	412	411	1225	68
14.2>x>=13.4	448	447	1153	64
13.4>x>=12.5	484	483	1081	60
12.5>x>=11.7	520	519	1009	56
11.7>x>=10.9	556	555	937	52
10.9>x>10.0	592	591	865	48

Table 2.6: OFDMA Parameters for 1024 FFT When Tone Dropping Is Applied (Table 796 in [5])

BW Range, x (MHz)	Number of guard subcarriers		Number of used subcarriers	Number of PRUs (N_{PRU})
	Left	Right		
10.0>x>=9.2	116	115	793	44
9.2>x>=8.4	152	151	721	40
8.4>x>=7.5	188	187	649	36
7.5>x>=6.7	224	223	577	32
6.7>x>=5.9	260	259	505	28
5.9>x>5.0	296	295	433	24

2.3.2 Miniband Permutation

The miniband permutation maps the PRU_{MB}s to Permuted PRU_{MB}s ($PPRU_{MB}$s) to ensure frequency diverse PRUs are allocated to each frequency partition. The mapping from

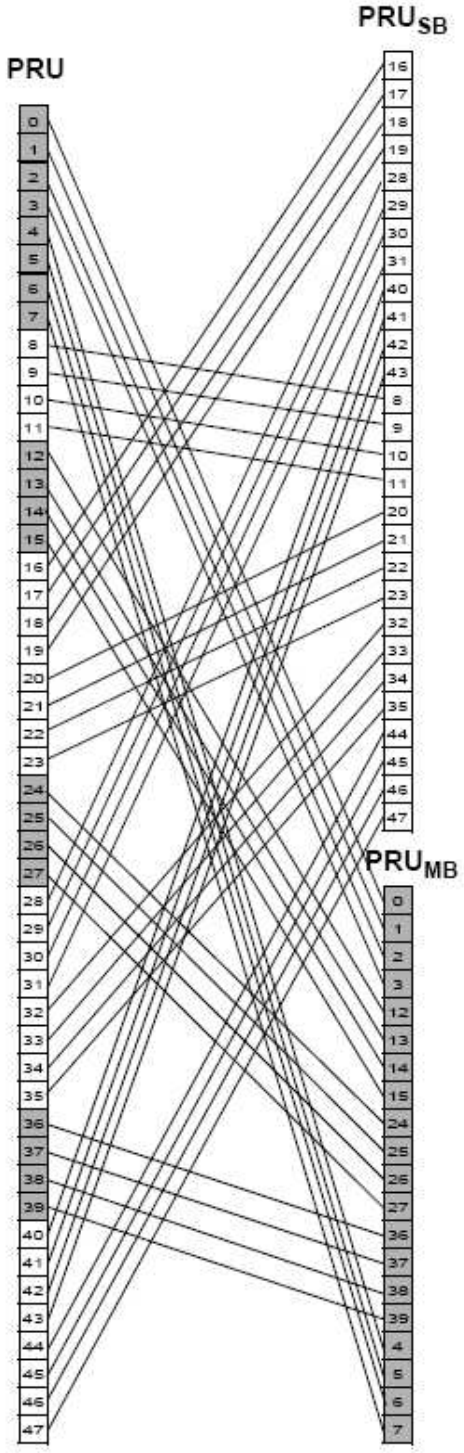

Figure 2.9: PRU to PRU_{SB} and PRU_{MB} mapping for BW = 10 MHz, $K_{SB} = 7$ (Figure 500 in [5]).

PRU_{MB} to ($PPRU_{MB}$s) is given by

$$PRU_{MB}[j] = PRU[i], \quad j = 0, 1, ..., L_{MB} - 1, \tag{2.9}$$

where

$$i = (q(j) \mod D) \cdot P + \lfloor \frac{q(j)}{D} \rfloor, \tag{2.10}$$

$$P = \min(K_{MB}, N_1/N_2), \tag{2.11}$$

$$r(j) = \max(j - ((K_{MB} \mod P) \cdot D), 0), \tag{2.12}$$

$$q(j) = j + \lfloor \frac{r(j)}{D-1} \rfloor, \tag{2.13}$$

$$D = \lfloor \frac{K_{MB}}{P} + 1 \rfloor. \tag{2.14}$$

Fig. 2.10 depicts the mapping from PRUs to PRU_{SB} and $PPRU_{MB}$.

2.3.3 Frequency Partitioning

The PRU_{SB}s and $PPRU_{MB}$s 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. The Frequency Partition Count (FPCT) defines the number of frequency partitions. The Frequency Partition Size (FPS_i) defines the number of PRUs allocated to FP_i. FPCT and FPS_i are determined from DFPC as shown in Table 2.7. A field of 1, 2, or 3-bit parameter, called the Uplink Frequency Partition Subband Count (DFPSC), defines the number of subbands allocated to frequency partition (FP_i), for $i > 0$. When UFPC = 0, DFPSC is equal to 0.

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_i and and FPSC fields. The number of subband PRUs and miniband PRUs in each frequency partition are $L_{SB,FP_i} = N_1 \cdot K_{SB,FP_i}$ and $L_{MB,FP_i} = N_2 \cdot K_{MB,FP_i}$, respectively. We have

$$K_{SB,FPi} = \begin{cases} K_{SB}, & i = 0, \\ \text{FPSC}, & i > 0, \end{cases} \tag{2.15}$$

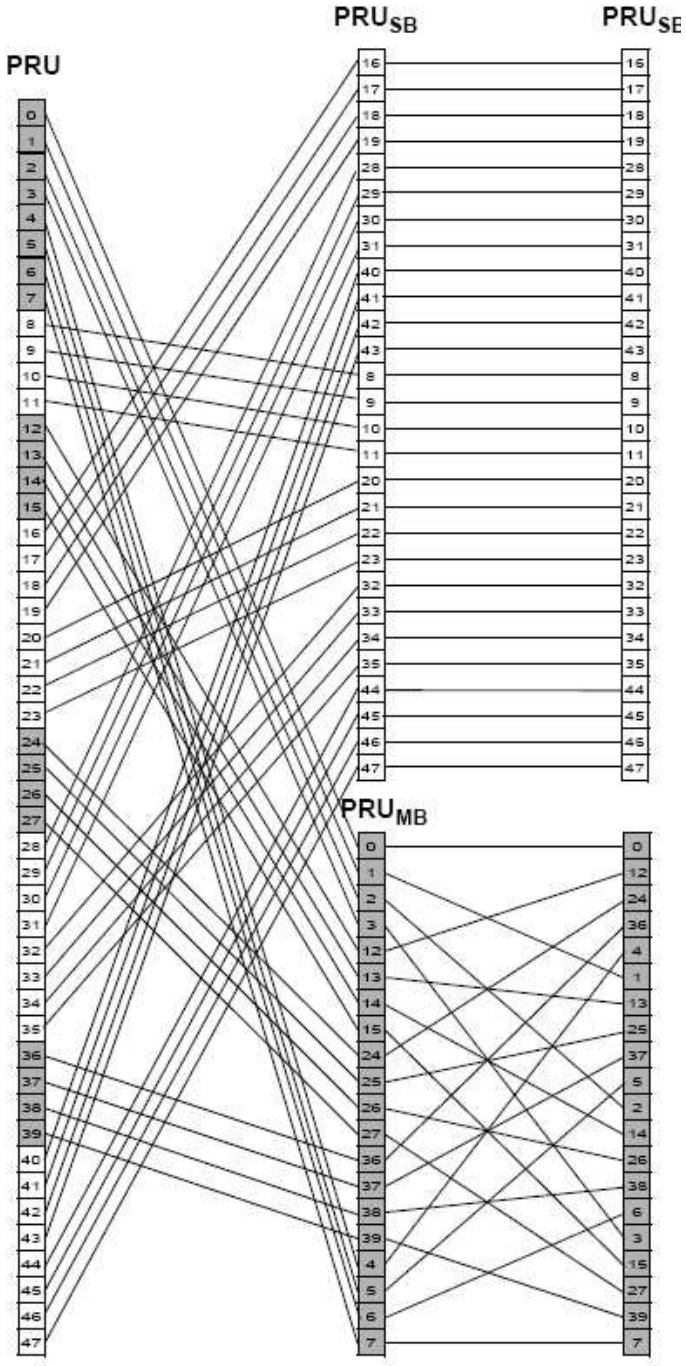

Figure 2.10: Mapping from PRUs to PRU_{SB} and $PPRU_{MB}$ mapping for BW = 10 MHz and $K_{SB} = 7$ (Fig. 501 in [5]).

$$K_{MB,FP_i} = (FPS_i - K_{SB,FP_i} \cdot N_1)/N_2, \quad 0 \leq i < \text{FPSC}. \tag{2.16}$$

Table 2.7: Mapping Between DFPC and Frequency Partition for 1024 FFT Size (Table 806 in [5])

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	$FPS_1 = N_{PRU} - 2*\text{floor}(N_{PRU}/3)$ $FPS_2 = \text{floor}(N_{PRU}/3)$ $FPS_3 = \text{floor}(N_{PRU}/3)$
2	1 : 1 : 1 : 1	4	$N_{PRU}-3*\text{floor}(N_{PRU}/4)$	$\text{floor}(N_{PRU}/4)$
3	3 : 1 : 1 : 1	4	$N_{PRU}-3*\text{floor}(N_{PRU}/6)$	$\text{floor}(N_{PRU}/6)$
4	5 : 1 : 1 : 1	4	$N_{PRU}-3*\text{floor}(N_{PRU}/8)$	$\text{floor}(N_{PRU}/8)$
5	9 : 5 : 5 : 5	4	$N_{PRU}-3*\text{floor}(N_{PRU}*5/24)$	$\text{floor}(N_{PRU}*5/24)$
6	0 : 1 : 1 : 0	2	0	$N_{PRU}/2$ for i=1, 2 0 for i =3
7	1 : 1 : 1 : 0	3	$N_{PRU}-2*\text{floor}(N_{PRU}/3)$	$\text{floor}(N_{PRU}/3)$ for i=1, 2 0 for i =3

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

$$PRU_{FP_i}(j) = \begin{cases} PRU_{SB}(k_1), & 0 \le j < L_{SB,FP_i}, \\ PPRU_{MB}(k_2), & L_{SB,FP_i} \le j < (L_{SB,FP_i} + L_{MB,FP_i}), \end{cases} \tag{2.17}$$

where

$$k_1 = \sum_{m=0}^{i-1} L_{SB,FP_m} + j \tag{2.18}$$

and

$$k_2 = \sum_{m=0}^{i-1} L_{SB,FP_m} + j - L_{SB,FP_i}. \tag{2.19}$$

2.4 Cell-Specific Resource Mapping [5]

The content of this section is mainly taken from [5]. PRU_{FPi}s are mapped to LRUs. All further PRU and subcarrier permutations are constrained to the PRUs of a frequency partition.

2.4.1 CRU/DRU Allocation

The partition between CRUs and DRUs is done on a sector specific basis. A 4 or 3-bit Downlink subband-based CRU Allocation Size ($DCAS_{SBi}$) field is sent in the SFH for each allocated frequency partition. $DCAS_{SBi}$ indicates the number of allocated CRUs for partition FP_i in unit of subband size. A 5, 4 or 3-bit Downlink miniband-based CRU Allocation Size ($DCAS_{MB}$) is sent in the SFH only for partition FP_0 depending on system bandwidth, which indicates the number of allocated miniband-based CRUs for partition FP_0. The number of CRUs in each frequency partition is denoted $L_{CRU,FPi}$, where

$$L_{CRU,FPi} = \begin{cases} CAS_{SBi} \cdot N_1 + CAS_{MB} \cdot N_2, & i = 0, \\ CAS_{SBi} \cdot N_1, & 0 < i < \text{FPSC}. \end{cases} \tag{2.20}$$

The number of DRUs in each frequency partition is denoted $L_{DRU.FPi}$, where $L_{DRU,FPi} = FPS_i - L_{CRU,FPi}$ for $0 \leq i <$ FPSC and FPS_i is the number of PRUs allocated to FP_i. The mapping of PRU_{FPi} to CRU_{FPi} is given by

$$CRU_{FPi}[j] = \begin{cases} PRU_{FPi}[j], & 0 \leq j < CAS_{SBi} \cdot N_1, \quad 0 \leq i < \text{FPSC}, \\ PRU_{FPi}[k + CAS_{SBi} \cdot N_1], & CAS_{SBi} \cdot N_1 \leq j < L_{CRU,FPi}, \quad 0 \leq i < \text{FPSC}, \end{cases} \tag{2.21}$$

where $k = s[j - CAS_{SBi} \cdot N_1]$, with s[ ] being the CRU/DRU allocation sequence defined as

$$s[j] = \{PermSeq(j) + DL\_PermBase\} \mod \{FPS_i - CAS_{SBi} \cdot N_1\}. \tag{2.22}$$

where $PermSeq()$ is the permutation sequence of length $(FPS_i - CAS_{SBi} \cdot N_1)$ and is determined by SEED = IDcell $\cdot$ 343 mod 2^{10}, $DL\_PermBase$ is an interger ranging from 0 to 31, which is set to preamble IDcell. The mapping of PRU_{FPi} to DRU_{FPi} is given by

$$DRU_{FPi}[j] = PRU_{FPi}[k + CAS_{SBi} \cdot N_1], \quad 0 \leq j < L_{DRU,FPi}. \tag{2.23}$$

where $k = s^c[j]$, with $s^c[\,]$ being the sequence which is obtained by renumbering the remainders of the PRUs which are not allocated for CRU from 0 to $L_{DRU,FPi} - 1$.

2.4.2 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 remainder 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 $L_{SC,l}/2$. A permutation sequence $PermSeq()$ performs the DL subcarrier permutation as follows. For each lth OFDMA symbol in the subframe:

1. Allocate the n_l pilots within each DRU as described in [5] Section (16.3.4.4). Denote the data subcarriers of $DRU_{FPi}[j]$ in the lth OFDMA symbol as

$$SC^{FPi}_{DRU,j,l}[k], \quad 0 \le j < L_{DRU,FPi}, \qquad 0 \le k < L_{SC,l}. \tag{2.24}$$

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

$$RSP_{FPi,l}[u] = \{SC^{FPi}_{DRUj,l}[2v], SC^{FPi}_{DRUj,l}[2v+1]\}, \quad 0 \le u < L_{DRU,FPi}L_{SP,l}, \tag{2.25}$$

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

3. Apply the subcarrier permutation formula to map $RSP_{FPi,l}$ into the sth distributed

LRU, $s = 0, 1, \ldots, L_{DRU,FPi} - 1$, where the subcarrier permutation formula is given by

$$SC^{FPi}_{LRUs,l}[m] = RSP_{FPi,l}[k], \quad 0 \le m \le L_{SP,l}, \tag{2.26}$$

with

$$k = L_{DRU,FPi} \cdot f(m, s) + g(PermSeq(), s, m, l, t). \tag{2.27}$$

In the above,

1. $SC^{FPi}_{LRUs,l}[m]$ 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, $0 \le m \le L_{SP,l} - 1$;
3. l is the OFDMA symbol index, $0 \le l \le N_{sym} - 1$;
4. s is the distributed LRU index, $0 \le s \le L_{DRU,FPi} - 1$;
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 $\cdot$ 343 mod 2^{10};
7. $g(PermSeq(), s, m, l, t)$ is a function with value in the range $[0, L_{DRU,FPi} - 1]$, which is defined according to

$$g(PermSeq(), s, m, l, t) = \{PermSeq[\{f(m, s) + s + l\} \mod \{L_{DRU,FPi}\}] + DL\_PermBase\} \mod L_{DRU,FPi}. \tag{2.28}$$

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

2.4.3 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 $d_1 = \lfloor \text{SEED}/2^5 \rfloor + 1$ and $d_2 = \text{SEED} \mod 2^5$.
  2. Initialize the maximum iteration number, $N = 4$.
  3. Initialize an array A with size M to contents $0, 1, \ldots, M-1$ (i.e., $A[i] = i$, for $0 \leq i < M$).
  4. Initialize the counter i to $M - 1$.
  5. 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 and j by 1.
     (b) Calculate the output variable of $y = \{(d_1 \cdot x + d_2) \mod 1031\} \mod M$.
     (c) Repeat the above steps (a) and (b), if $y \leq i$ and $j < N$.
     (d) If $y \leq i$, set $y = y \mod i$.
     (e) Swap the ith and the yth elements in the array, i.e., perform the steps $Temp = A[i]$, $A[i] = A[y]$, and $A[y] = Temp$.
     (f) Decrement i by 1.

Then $PermSeq(i) = A[i]$, where $0 \leq i < M$.

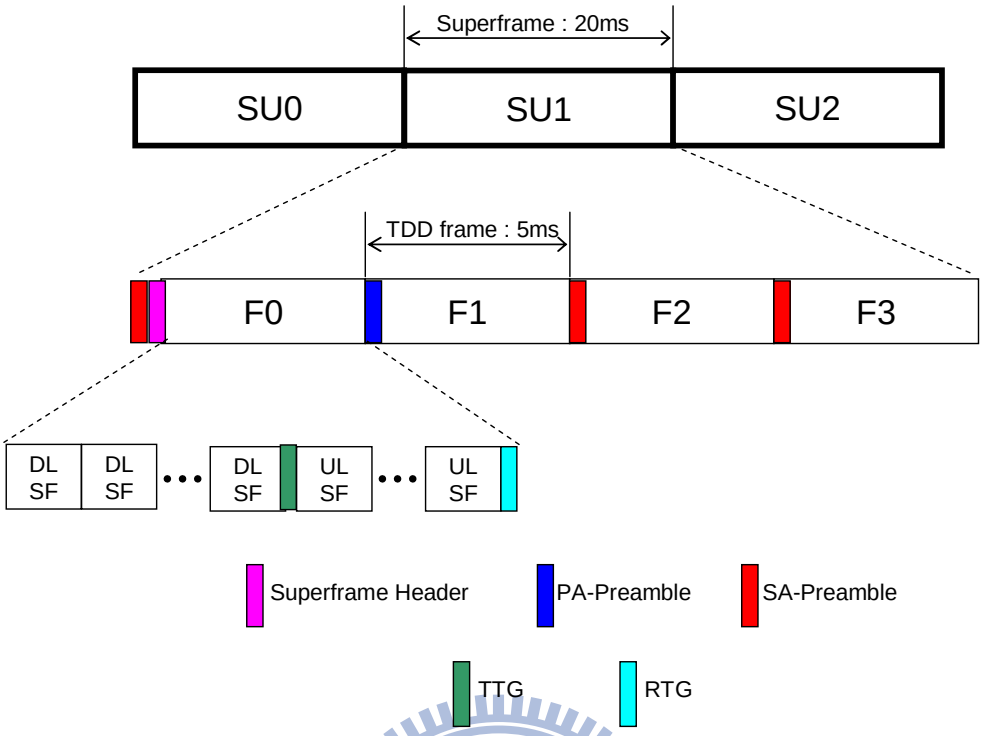

Figure 2.11: Location of the A-Preamble (re-arranged from Fig. 521 in [5]).

2.5 Advanced Preamble (A-Preamble) Structure [5]

The material in this subsection is mainly taken from [5]. There are two types of Advanced Preamble (A-Preamble): primary advanced preamble (PA-Preamble) and secondary advanced preamble (SA-Preamble). One PA-Preamble symbol and three SA-Preamble symbols exist within the superframe. The location of an A-Preamble symbol the first symbol of a frame. PA-Preamble is at the first symbol of second frame in a superframe while SA-Preamble is at the first symbol of each of the remaining three frames. Fig. 2.11 depicts the location of A-Preamble symbols.

2.5.1 Primary Advanced Preamble (PA-Preamble)

The length of sequence for PA-Preamble is 216 regardless of the FFT size. PA-Preamble carries the information of advanced base station (ABS) type, system bandwidth, and carrier

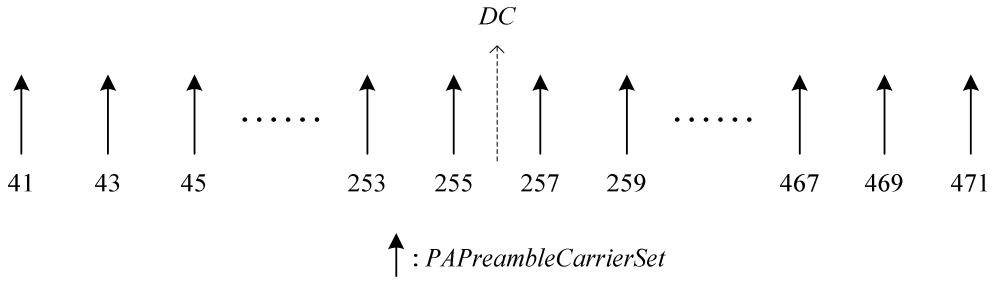

Figure 2.12: PA-Preamble symbol structure of 5-MHz system (Fig. 522 in [5]).

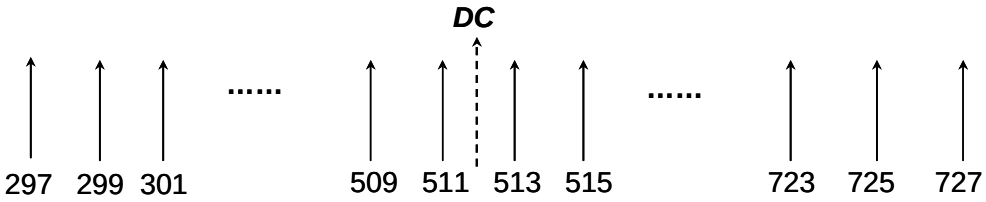

Figure 2.13: PA-Preamble symbol structure of 10 MHz system [1].

configuration.

Take, for example, a 5-MHz system where the subcarrier index 256 is the DC subcarrier. The set of PA-Preamble subcarriers are given by

$$PAPreambleCarrierSet = 2 \cdot k + 41, \tag{2.29}$$

where k is a running index from 0 to 215. Figs. 2.12, 2.13, and 2.14 depict the structures of the PA-Preamble in the frequency domain for systems of different bandwidths. The PA-Preamble always occupies the middle 5-MHz bandwidth whose center is the DC subcarrier and the outside subcarriers are all zero.

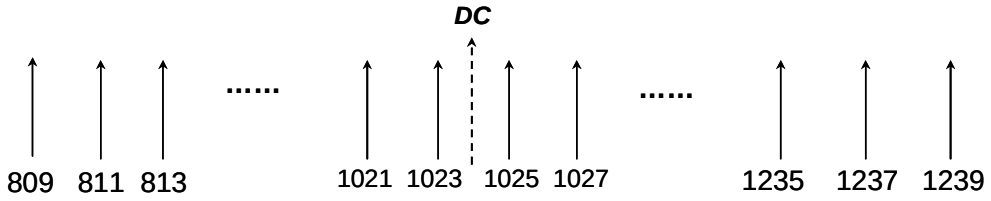

Figure 2.14: PA-Preamble symbol structure of 20 MHz system [1].

Index	Carrier	BW	Series to modulate
0	Fully configured	5 MHz	6DB4F3B16BCE59166C9CEF7C3C8CA5EDFC16A9D1DC01F2AE6AA08F
1		7,8.75,10 MHz	1799628F3B9F8F3B22C1BA19EAF94FEC4D37DEE97E027750D298AC
2		20 MHz	92161C7C19BB2FC0ADE5CEF3543AC1B6CE6BE1C8DCABDDD319EAF7
3		reserved	6DE116E665C395ADC70A89716908620868A60340BF35ED547F8281
4		reserved	BCFDF60DFAD6B027E4C39DB20D783C9F467155179CBA31115E2D04
5		reserved	7EF1379553F9641EE6ECDBF5F144287E329606C616292A3C77F928
6		reserved	8A9CA262B8B3D37E3158A3B17BFA4C9FCFF4D396D2A93DE65A0E7C
7		reserved	DA8CE648727E4282780384AB53CEEBD1CBF79E0C5DA7BA85DD3749
8		reserved	3A65D1E6042E8B8AADC701E210B5B4B650B6AB31F7A918893FB04A
9		reserved	D46CF86FE51B56B2CAA84F26F6F204428C1BD23F3D888737A0851C
10	Partially configured	N/A	640267A0C0DF11E475066F1610954B5AE55E189EA7E72EFD57240F

Figure 2.15: PA-Preamble Series (Table 815 in [5]).

Fig. 2.15 shows the PA-Preamble sequences in hexadecimal format. The defined series is mapped onto subcarriers in ascending order, obtained by converting the series to a binary series and starting the series from the most signification bit (MSB) up to 216 bits with 0 mapped to $+1$ and 1 mapped to -1.

The magnitude boosting levels for FFT sizes 512, 1024, 2048 are 1.9216, 2.6731, 4.6511, respectively. For 512-FFT, as an example, the boosted PA-Preamble at kth subcarrier is

$$c_k = 1.9216 \cdot b_k, \tag{2.30}$$

where b_k represents the PA-Preamble value before boosting ($+1$ or -1).

2.5.2 Secondary Advanced Preamble (SA-Preamble)

The lengths of sequences for SA-Preamble are 144, 288, and 576 for 512-FFT, 1024-FFT, and 2048-FFT, respectively, where subcarrier indexes 256, 512, and 1024, respectively, are

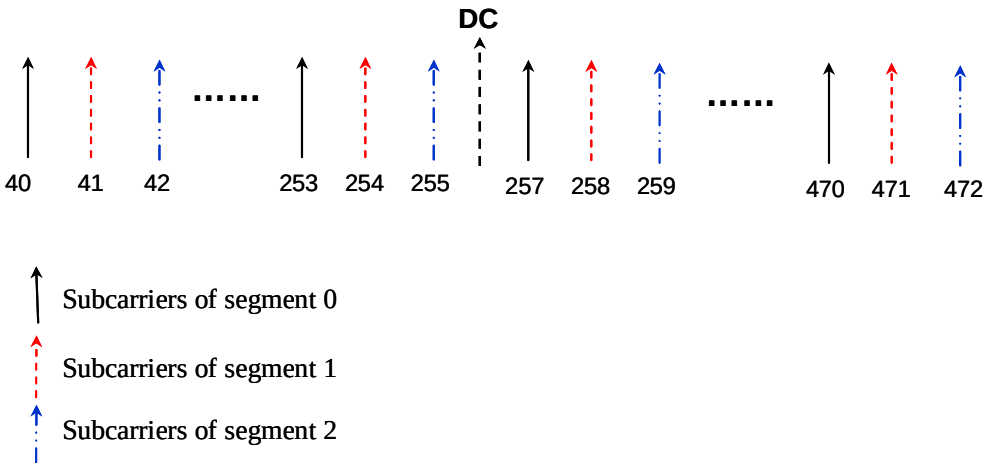

Figure 2.16: SA-Preamble symbol structure of 5 MHz.

the DC subcarrier. The set of SA-Preamble subcarriers are given by

$$SAPreambleCarrierSet_n = n + 3 \cdot k + 40 \cdot \frac{N_{SAP}}{144} + \lfloor \frac{2 \cdot k}{N_{SAP}} \rfloor, \tag{2.31}$$

where n is the index of the SA-Preamble carrier-set with $n = 0$, 1, or 2 representing the segment ID, and k is a running index from 0 to $N_{SAP} - 1$ for each FFT size. Fig. 2.16 illustrates the allocation under 512-FFT.

Each cell ID has an integer value IDcell from 0 to 767. The IDcell is defined as

$$\text{IDcell} = 256n + Idx, \tag{2.32}$$

where n is the segment ID and $Idx = 2 \cdot \mod(q, 128) + \lfloor q/128 \rfloor$ with q being a running index from 0 to 255.

For 512-FFT system, the 144-bit SA-Preamble sequence is divided into 8 main sub-blocks, namely, A, B, C, D, E, F, G, and H. The length of each sub-block is 18 samples (after modulation). Each segment ID has a different set of sequence sub-blocks. Tables 784 to 786 in [5] give the 8 sub-blocks of each segment ID, where 9 hexadecimal numbers are used to represent the 36 bits that are mapped to a QPSK sequence in +1, +j, −1, and $-j$ for each sub-block. Each table contains 128 sequences indexed by q from 0 to 127. The modulation

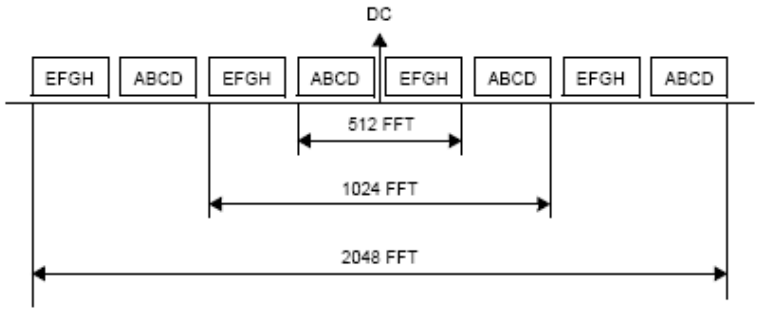

Figure 2.17: The allocation of sequence block for each FFT size (Fig. 524 in [5]).

sequence is obtained by converting each hexadecimal number $X_i^{(q)}$ into two QPSK symbols $v_{2i}^{(q)}$ and $v_{2i+1}^{(q)}$, where i=0, 1, ..., 7, 8. The converting equations are as follows:

$$v_{2i}^{(q)} = \exp(j\frac{\pi}{2}(2 \cdot b_{i,0}^{q} + b_{i,1}^{q})), \qquad v_{2i+1}^{(q)} = \exp(j\frac{\pi}{2}(2 \cdot b_{i,2}^{q} + b_{i,3}^{q})), \tag{2.33}$$

where $X_i^{(q)} = 2^3 \cdot b_{i,0}^{(q)} + 2^2 \cdot b_{i,1}^{(q)} + 2^1 \cdot b_{i,2}^{(q)} + 2^0 \cdot b_{i,3}^{(q)}$.

The other 128 sequences indexed by q from 128 to 255 are obtained by letting $v_k^{(q)} = (v_k^{(q-128)})^*$ where $q = 128, 129, ..., 254, 255$.

Fig. 2.17 shows how the sub-blocks are modulated and mapped (sequentially in ascending order) onto the SA-Preamble subcarrier-set. For higher FFT sizes, the basic blocks (A, B, C, D, E, F, G, H) are repeated in the same order. For instance, in the case of 1024-FFT, sub-blocks E, F, G, H, A, B, C, D, E, F, G, H, A, B, C, and D are modulated and mapped sequentially in ascending order onto the SA-Preamble subcarrier-set according to segment ID.

For 512-FFT, the blocks (A, B, C, D, E, F, G, H) are subject to the following right circular shifts (0, 2, 1, 0, 1, 0, 2, 1), respectively. Fig. 2.18 depicts the symbol structure of SA-Preamble in the frequency domain for 512-FFT. For higher FFT sizes, the same rule applies.

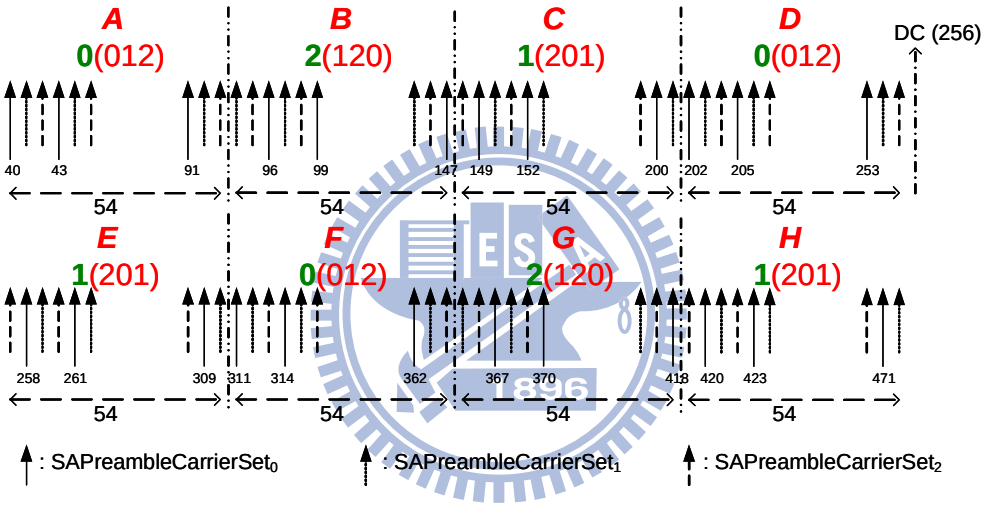

Figure 2.18: SA-Preamble symbol structure for 512-FFT (Fig. 525 in [5]).

Chapter 3

Initial Downlink Synchronization

The downlink synchronization can be divided into two type: initial synchronization and normal synchronization. When the advance mobile station (AMS) receiver enters the network for the first time, it need to perform initial DL synchronization. Afterward, the AMS needs to keep trading the carrier frequency, and the timing, and the power level, which constitutes the work of normal DL synchronization. In this thesis, our study focuses on initial DL synchronization; so we discuss the initial DL synchronization problem of the IEEE 802.16m TDD system and introduce the initial DL synchronization algorithm of [1, 2].

3.1 The Initial Synchronization Problem [1, 2]

In DL signal reception, in principle, the receiver needs to estimate the carrier frequency offset (CFO), carrier phase offset (CPO), sampling frequency offset (SFO), sampling phase offset (SPO), and symbol time offset (STO). Some causes of CFO are mismatch of local oscillators and Doppler shifts due to mobility, and a cause of CPO is phase mismatch in local oscillators. Different sampling rates in the transmitter and the receiver bring about SFO and different sampling phases in the transmitter and the receiver, i.e., SPO. The STO can arise from the unknown propagation delay between the transmitter and the receiver.

If CFO estimation is accurate enough and if STO estimation and correction is constantly performed, then SFO estimation may be unnecessary, because from the beginning of an

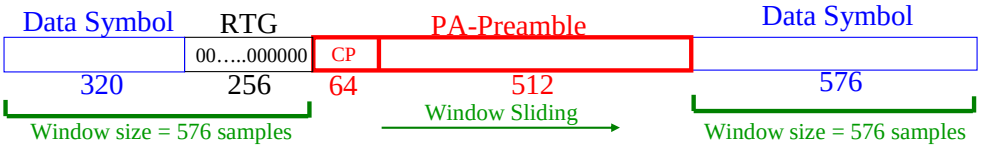

Figure 3.1: Window sliding structure [1].

OFDMA symbol to the end of it the SPO may change very little. The CPO and the SPO can be considered part of channel response and dealt with in channel estimation. As a result, only two issues yet need to be solved, i.e., CFO estimation and STO estimation. These are the focus of the present chapter.

Moreover, because the PA-Preamble in IEEE 802.16m also carries information about the system bandwidth, there is a need to identify it also in the synchronization stage. Our synchronization design thus also takes this into consideration.

3.2 Derivation of the Initial Synchronization Procedure [1, 2]

There are three possible PA-Preamble series, as shown in Fig. 2.15. Because the PA-Pramble series are known, we utilize this knowledge to derive the initial DL synchronization algorithm. Although there are three different PA-Preambles with different bandwidth, 5, 10, and 20 MHz, but the commonality is that all three PA-Preambles, whose length is all 216 points, locate in the middle part of the bandwidth. Therefore, when the MS receives the signal, it only need to observe a 5-MHz bandwidth because there is no PA-Preamble signal outside this bandwidth, whatever the system bandwidth. In other words, we can do downsampling for the 10-MHz and the 20-MHz signal to the 5MHz bandwidth without losing information on PA-Preamble.

The received PA-Preamble (including CP) can be represented as

$$\mathbf{y}_{576} = \mathbf{\Gamma}(\delta) \cdot \mathbf{T}_{576} \cdot \mathbf{h}_{576} + \eta_{576} \tag{3.1}$$

where $\mathbf{y}_{576} = [y_{448}, y_{449}, ..., y_{511}, y_0, y_1, ..., y_{511}]'$, the received PA-Preamble symbol, δ is the normalized carrier frequency offset (what the normalization is whit respect to subcarrier spacing), $\mathbf{T}_{576}$ is the 576×576 Toeplitz matrix of the transmitted PA-Preamble symbol as

$$\mathbf{T}_{576} = \begin{bmatrix} x_{512} & 0 & 0 & . & . & . & . & . & . & . & . & . & 0 & 0 \\ x_{513} & x_{512} & 0 & . & . & . & . & . & . & . & . & . & 0 & 0 \\ . & x_{513} & x_{512} & 0 & . & . & . & . & . & . & . & . & . & . \\ . & . & x_{513} & x_{512} & . & . & . & . & . & . & . & . & . & . \\ x_{575} & . & . & . & . & 0 & . & . & . & . & . & . & . & . \\ x_0 & x_{575} & . & . & . & x_{512} & . & . & . & . & . & . & . & . \\ x_1 & x_0 & x_{575} & . & . & . & . & . & . & . & . & . & . & . \\ . & x_1 & x_0 & . & . & . & . & . & . & 0 & . & . & . & . \\ . & . & x_1 & . & . & x_{574} & . & . & . & x_{512} & 0 & . & . & . \\ x_{509} & . & . & . & . & x_{575} & x_{574} & x_{573} & . & . & x_{512} & 0 & . & . \\ x_{510} & x_{509} & . & . & . & x_0 & x_{575} & x_{574} & . & . & x_{513} & x_{512} & 0 & . \\ x_{511} & x_{510} & x_{509} & . & . & x_1 & x_0 & x_{575} & . & . & x_{515} & x_{514} & x_{513} & x_{512} \end{bmatrix}, \quad (3.2)$$

$\mathbf{h}_{576}$ is the channel response vector, $\mathbf{\Gamma}(\delta)$ is the 576×576 diagonal matrix summarizing the effect of the CFO as

$$\mathbf{\Gamma}(\delta) = \begin{bmatrix} \exp(-j \cdot \frac{2\pi}{512} \cdot \delta \cdot 0) & & & & 0 \\ & \exp(-j \cdot \frac{2\pi}{512} \cdot \delta \cdot 1) & & & \\ & & . & & \\ & & & . & \\ 0 & & & & \exp(-j \cdot \frac{2\pi}{512} \cdot \delta \cdot 575) \end{bmatrix}, \quad (3.3)$$

and η_{576} is the additive white Gaussian noise (AWGN) vector.

3.2.1 Coarse Timing Synchronization

Fig. 3.1 depicts a model about the 576-points power sum with the window sliding. We know the information of TTG + RTG =165 μs in [5], so it is reasonable to suppose RTG is 45 μs, about 256 sampling periods, and CP factor is 1/8 in our study. We can also know the power of PA-Preamble is larger than the common data symbol because the amplitude of PA-Preamble is boosted before transmitting [5].

When the MS receives the PA-Preamble signal subject to delay, multipath propagation, and additive noise, the first task is to estimate the coarse timing to facilitate later work.

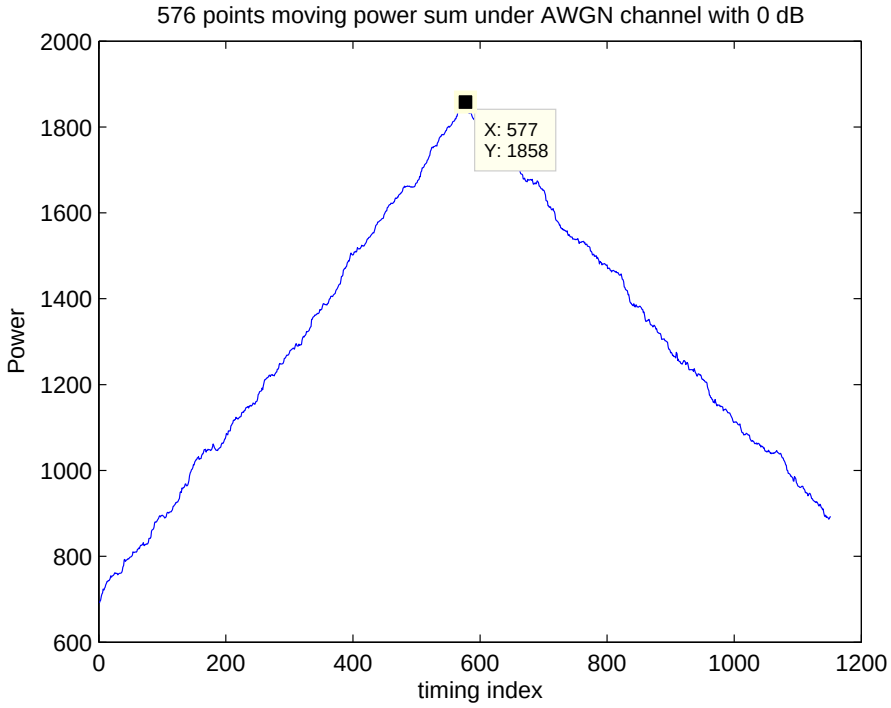

Figure 3.2: 576 points power sum under AWGN in 0 dB [1].

Refer to Fig. 3.1. We consider summing the signal power in a 576-point window. With the window sliding, we can decide the coarse timing as the point with the maximum power sum. This technique can actually be interpreted as quasi-maximum likelihood (ML) noncoherent detection of the preamble timing.

According to [1], Figs. 3.2 and 3.3 show the results of power sum with the window sliding in 0 dB of signal-to-noise ratio (SNR), under the AWGN channel and the SUI-5 channel with mobility 350 km/h. The rayleighchan, a Matlab function, leads to an initial delay of the generated channel, even if we set the delay of the direct path zero. Figs. 3.5 depict this phenomenon and we must compensate it in [1].

Note that the PA-Preamble timing we get by the above method has an offset to the real PA-Preamble timing due to multipath and noise effects. We will handle these problems in fine timing synchronization.

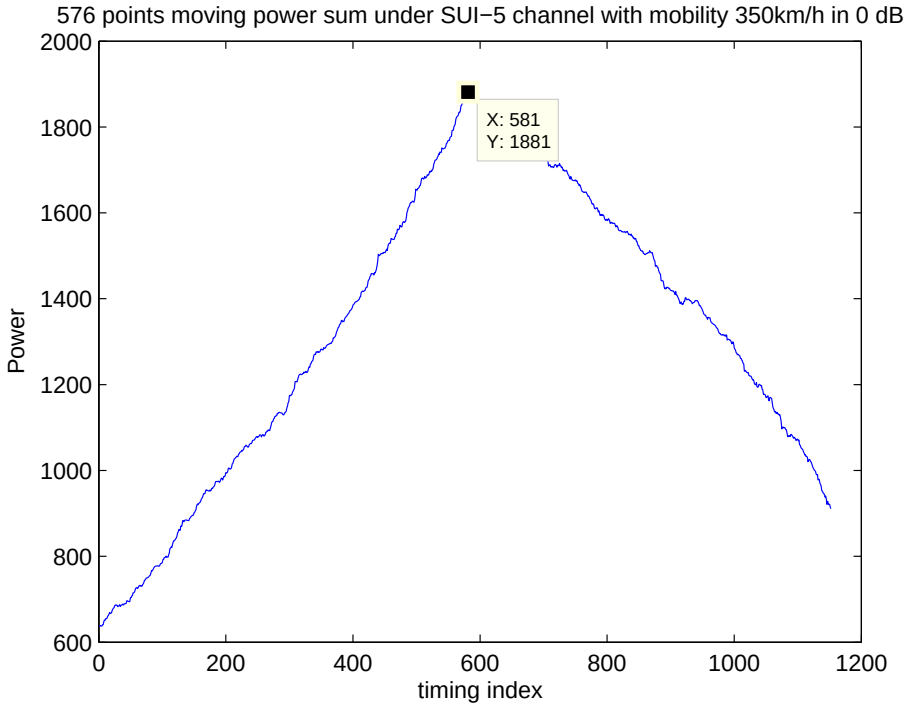

Figure 3.3: 576 points power sum under SUI-5 at mobility 350 km/h in 0 dB [1].

3.2.2 Estimation of Fractional Carrier Frequency Offset

Eq. (3.1) gives the received PA-Preamble signal. We attempt an ML estimation of δ from it. It turns out that a truly ML estimation is quite complex because $\mathbf{T}_{576}$ is not circulant. However, if the coarse timing lands us in the CP and if we sacrifice the available signal power in the CP, then we can obtain a reduced-complexity solution. Let $\mathbf{y}_{512}$ denote the received PA-Preamble symbol after removal of the CP. It is given by

$$\mathbf{y}_{512} = \mathbf{\Gamma}(\delta) \cdot \mathbf{T}_{x_n} \cdot \mathbf{h} + \eta, \tag{3.4}$$

where $\mathbf{x}_n = [x_0, x_1, ..., x_{511}]'$ (the transmitted PA-Preamble symbol), $\mathbf{T}_{x_n}$ is a 512×512 circulant matrix given by

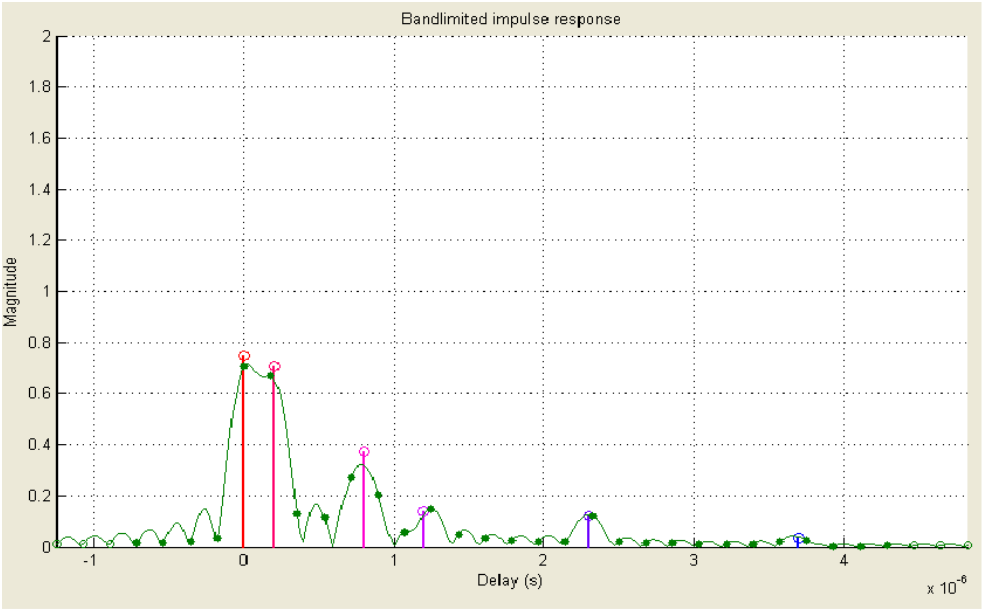

Figure 3.4: Channel impulse response of PB channel [1].

$$\mathbf{T}_{x_n} = \begin{bmatrix} x_0 & x_{511} & x_{510} & x_{509} & . & . & . & x_2 & x_1 \\ x_1 & x_0 & x_{511} & x_{510} & . & . & . & x_3 & x_2 \\ . & x_1 & x_0 & x_{511} & . & . & . & x_3 & x_2 \\ x_{63} & . & x_1 & . & . & . & . & . & . \\ . & x_{63} & . & . & . & . & . & . & . \\ . & . & . & . & . & . & . & x_{510} & . \\ x_{509} & . & . & . & . & . & . & x_{511} & x_{510} \\ x_{510} & x_{509} & . & . & x_{63} & . & . & x_0 & x_{511} \\ x_{511} & x_{510} & x_{509} & . & . & x_{63} & . & x_1 & x_0 \end{bmatrix}, \tag{3.5}$$

h is the channel impulse response vector,

$$\mathbf{\Gamma}(\delta) = \begin{bmatrix} \exp(-j \cdot \frac{2\pi}{512} \cdot \delta \cdot 0) & & & & \\ & \exp(-j \cdot \frac{2\pi}{512} \cdot \delta \cdot 1) & & 0 & \\ & & & . & \\ & & & . & \\ & & 0 & & \\ & & & & \exp(-j \cdot \frac{2\pi}{512} \cdot \delta \cdot 511) \end{bmatrix}, \tag{3.6}$$

and η is an AWGN vector. Due to possibly incorrect identification of the PA-Preamble starting time from the coarse timing synchronization, there may be a circular shift of the elements in the **h** vector from their original positions.

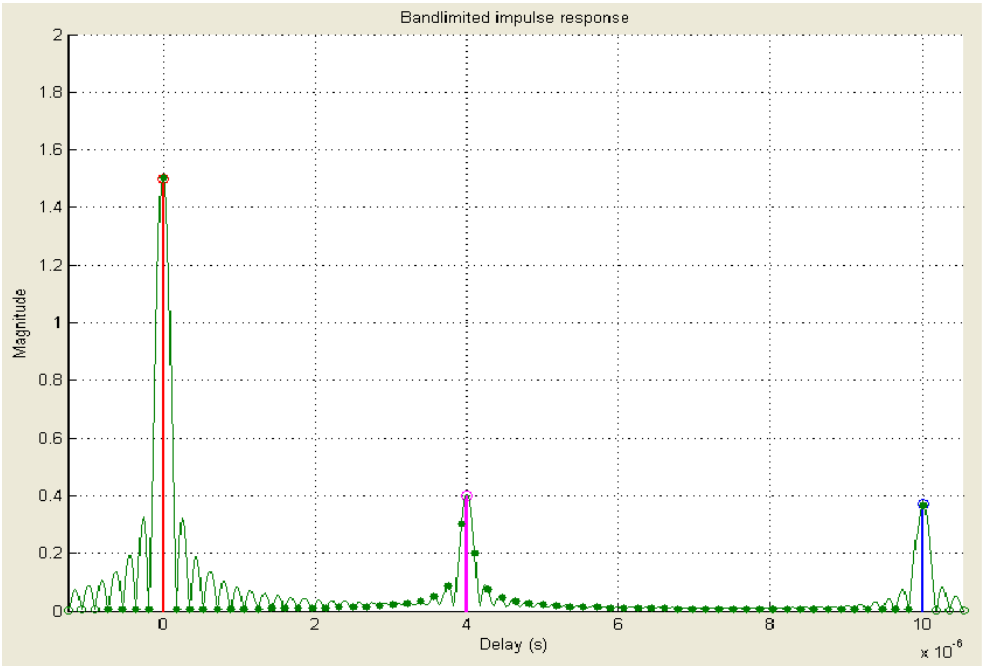

Figure 3.5: Channel impulse response of SUI-5 channel [1].

Eq. (3.4) can then be rewritten as:

$$\mathbf{y}_{512} = \mathbf{\Gamma}(\delta) \cdot \mathbf{F}^H \cdot \mathbf{F} \cdot \mathbf{T}_{x_n} \cdot \mathbf{F}^H \cdot \mathbf{F} \cdot \mathbf{h} + \eta \tag{3.7}$$

$$= \mathbf{\Gamma}(\delta) \cdot \mathbf{F}^H \cdot (\mathbf{F} \cdot \mathbf{T}_{x_n} \cdot \mathbf{F}^H) \cdot (\mathbf{F} \cdot \mathbf{h}) + \eta \tag{3.8}$$

$$= \mathbf{\Gamma}(\delta) \cdot \mathbf{F}^H \cdot \mathbf{D}_k \cdot \mathbf{H} + \eta, \tag{3.9}$$

where $\mathbf{F}$ is the normalized 512×512 FFT matrix, $\mathbf{F}^H$ is the corresponding normalized IFFT matrix, $\mathbf{H}$ is the channel frequency response vector, and $\mathbf{D}_k$ is a diagonal matrix of the PA-Preamble sequence in the frequency domain, with k being the PA-Preamble index.

The likelihood function of $\mathbf{y}_{512}$ can be written as:

$$p(\mathbf{y}_{512}|\delta, \mathbf{H}, k) = \frac{1}{(2\pi\sigma_\eta^2)^{512}} \cdot \exp(-\frac{1}{2\sigma_\eta^2}\|\mathbf{y}_{512} - \mathbf{\Gamma}(\delta) \cdot \mathbf{F}^H \cdot \mathbf{D}_k \cdot \mathbf{H}\|^2), \tag{3.10}$$

in the likelihood function, there are three unknowns, namely δ, $\mathbf{H}$ and k. The ML estimation is thus given by

$$\arg\max_{\delta,\mathbf{H},k} p(\mathbf{y}_{512}|\delta,\mathbf{H},k) \tag{3.11}$$

$$= \arg\min_{\delta,\mathbf{H},k} \|\mathbf{y}_{512} - \mathbf{\Gamma}(\delta)\cdot\mathbf{F}^H\cdot\mathbf{D}_k\cdot\mathbf{H}\|^2 \tag{3.12}$$

$$= \arg\min_{\delta,k}\min_{\mathbf{H}|\delta,k} \|\mathbf{y}_{512} - \mathbf{\Gamma}(\delta)\cdot\mathbf{F}^H\cdot\mathbf{D}_k\cdot\mathbf{H}\|^2 \tag{3.13}$$

$$\Rightarrow \arg\min_{\delta,k} \|\mathbf{y}_{512} - \mathbf{\Gamma}(\delta)\cdot\mathbf{F}^H\cdot\mathbf{D}_k\cdot\mathbf{D}_k^H\cdot\mathbf{F}\cdot\mathbf{\Gamma}^H(\delta)\cdot\mathbf{y}_{512}\|^2. \tag{3.14}$$

Note that (3.14) arises because the inner minimization of (3.13) is achieved with $\mathbf{H} = \mathbf{D}_k^H\cdot\mathbf{F}\cdot\mathbf{\Gamma}^H(\delta)\cdot\mathbf{y}_{512}$ as can be obtained via standard least-square estimation technique. Since $\mathbf{D}_k\cdot\mathbf{D}_k^H$ is the same whatever for add k, we cannot solve for the optimal k from (3.14), but must find it through above other means, In addition, the minimization target in (3.14) is a function of δ only. Thus it is equivalent to:

$$\arg\min_{\delta} \|\mathbf{y}_{512} - \mathbf{\Gamma}(\delta)\cdot\mathbf{F}^H\cdot\mathbf{D}_k\cdot\mathbf{D}_k^H\cdot\mathbf{F}\cdot\mathbf{\Gamma}^H(\delta)\cdot\mathbf{y}_{512}\|^2 \tag{3.15}$$

$$= \arg\min_{\delta} \|[I - \mathbf{\Gamma}(\delta)\cdot\mathbf{F}^H\cdot\mathbf{D}_k\cdot\mathbf{D}_k^H\cdot\mathbf{F}\cdot\mathbf{\Gamma}^H(\delta)]\cdot\mathbf{y}_{512}\|^2 \tag{3.16}$$

$$= \arg\min_{\delta} \mathbf{y}_{512}^H\cdot[I - \mathbf{\Gamma}(\delta)\cdot\mathbf{F}^H\cdot\mathbf{D}_k\cdot\mathbf{D}_k^H\cdot\mathbf{F}\cdot\mathbf{\Gamma}^H(\delta)]^2\cdot\mathbf{y}_{512} \tag{3.17}$$

$$= \arg\max_{\delta} \mathbf{y}_{512}^H\cdot\mathbf{\Gamma}(\delta)\cdot\mathbf{F}^H\cdot\mathbf{D}_k\cdot\mathbf{D}_k^H\cdot\mathbf{F}\cdot\mathbf{\Gamma}^H(\delta)\cdot\mathbf{y}_{512} \tag{3.18}$$

$$= \arg\max_{\delta} \gamma^{\mathbf{H}}(\delta)\cdot[\mathbf{Y}^{\mathbf{H}}\cdot\mathbf{F}^{\mathbf{H}}\cdot\mathbf{D_k}\cdot\mathbf{D}_{\mathbf{k}}^{\mathbf{H}}\cdot\mathbf{F}\cdot\mathbf{Y}]\cdot\gamma(\delta) \tag{3.19}$$

where $\gamma(\delta) = [\exp(-\mathbf{j}\cdot\frac{\mathbf{2\pi}}{\mathbf{512}}\cdot\delta\cdot\mathbf{0}), \exp(-\mathbf{j}\cdot\frac{\mathbf{2\pi}}{\mathbf{512}}\cdot\delta\cdot\mathbf{1}), ..., \exp(-\mathbf{j}\cdot\frac{\mathbf{2\pi}}{\mathbf{512}}\cdot\delta\cdot\mathbf{511})]'$, and $\mathbf{Y}$ is a diagonal matrix whose ith diagonal element is the ith element in $\mathbf{y}_{512}$.

Since the quantity $\mathbf{D}_k\cdot\mathbf{D}_k^H$ is the same for all three PA-Preamble series, the bracketed term in (3.19) is a known quantity for a given received PA-Preamble signal. Let $\mathbf{M} = \mathbf{Y}^H\cdot\mathbf{F}^H\cdot\mathbf{D}_k\cdot\mathbf{D}_k^H\cdot\mathbf{F}\cdot\mathbf{Y}$. Then the quantity to be maximized can be expressed as

$$
\begin{aligned}
&\gamma^{\mathbf{H}}(\delta)\cdot\mathbf{M}\cdot\gamma(\delta)\\
&=[1,e^{-a},e^{-2a},...,e^{-511a}]\cdot\begin{bmatrix} m_{0,0} & m_{0,1} & . & . & . & . & . & . & m_{0,511}\\ m_{1,0} & m_{1,1} & m_{1,2} & . & . & . & . & . & m_{1,511}\\ m_{2,0} & m_{2,1} & m_{2,2} & m_{2,3} & & & & & .\\ . & . & . & . & . & & & & .\\ . & . & & . & . & . & & & .\\ . & . & & & . & . & . & & .\\ m_{509,0} & . & & & & . & . & . & .\\ m_{510,0} & m_{510,1} & & & & & . & . & m_{510,511}\\ m_{511,0} & m_{511,1} & m_{511,2} & . & . & . & . & m_{511,510} & m_{511,511}\end{bmatrix}\cdot\begin{bmatrix}1\\ e^{a}\\ e^{2a}\\ .\\ .\\ .\\ e^{510a}\\ e^{511a}\end{bmatrix}\\
&=(m_{0,0}+e^{-a}\cdot m_{1,0}+...+e^{-511a}\cdot m_{511,0})+(m_{0,1}+e^{-a}\cdot m_{1,1}+...+e^{-511a}\cdot m_{511,1})\cdot e^{a}\\
&\quad+...+(m_{0,511}+e^{-a}\cdot m_{1,511}+...+e^{-511a}\cdot m_{511,511})\cdot e^{511a}\\
&=(m_{0,0}+m_{1,1}+...+m_{511,511})\cdot e^{0}+(m_{0,1}+m_{1,2}+...+m_{510,511})\cdot e^{a}+...+(m_{0,511}\cdot e^{511a})\\
&\quad+(m_{1,0}+m_{2,1}+...+m_{511,510})\cdot e^{-a}+(m_{2,0}+m_{3,1}+...+m_{511,509})\cdot e^{-2a}+...+(m_{511,0}\cdot e^{-511a})\\
&=\sum_{n=-511}^{511}M_n\cdot e^{j\cdot 2\pi\cdot n\cdot\delta/512},
\end{aligned}
\tag{3.20}
$$

where $a=j\cdot 2\cdot\pi\cdot\delta/512$, $m_{p,q}$ is the (p,q)th element of $\mathbf{M}$, and $M_n=\sum_{n=0}^{511}m_{n,n}$, the sum of the nth diagonal of $\mathbf{M}$.

Note that since $\mathbf{D}_k\cdot\mathbf{D}_k^H$ is diagonal, $\mathbf{W}\triangleq\mathbf{F}^H\cdot\mathbf{D}_k\cdot\mathbf{D}_k^H\cdot\mathbf{F}$ is a circulant matrix. Indeed, because $\mathbf{D}_k\cdot\mathbf{D}_k^H$ is nearly periodic (with mostly every other element equal to 1 while others equal to zero) along the diagonal, $\mathbf{W}$ is nearly tri-diagonal and so is $\mathbf{M}$. The three diagonal sums are given by

$$M_0 = \sum_{i=0}^{511} w_{i,i}\cdot|y_{i,i}|^2, \tag{3.21}$$

$$M_{-256} = \sum_{i=256}^{511} y_{i,i}^H\cdot w_{i,i-256}\cdot y_{i-256,i-256}, \tag{3.22}$$

$$M_{256} = \sum_{i=256}^{511} y_{i-256,i-256}^H\cdot w_{i-256,i}\cdot y_{i,i}, \tag{3.23}$$

where $y_{i,i}$ is the ith diagonal element of $\mathbf{Y}$, and $w_{i,i}$ is the ith diagonal of $\mathbf{W}$. Note that $M_{-256}=M_{256}^*$. Substituting (3.21)–(3.23) into (3.20) with all other terms set to null in order to reduce the effect of noise. We utilize the mathematic format of FFT of these three dominant terms to estimate fractional carrier frequency (FCFO) by finding the peak value and derive as

$$
\begin{aligned}
X[f] &= \sum_{n=0}^{511} M_n \cdot e^{-j\cdot 2\cdot \pi \cdot n\cdot f/512} && (3.24)\\
&\approx M^*_{256} + M_0 \cdot e^{-j\cdot 2\cdot \pi \cdot f/512} + M_{256} \cdot e^{-j\cdot 4\cdot \pi \cdot f/512} && (3.25)\\
&= e^{-j\cdot 2\cdot \pi \cdot f/512} \cdot (M^*_{256} \cdot e^{j\cdot 2\cdot \pi \cdot f/512} + M_0 + M_{256} \cdot e^{-j\cdot 2\cdot \pi \cdot f/512}) && (3.26)\\
&= e^{-j\cdot 2\cdot \pi \cdot f/512} \cdot (2 \cdot \Re\{M_{256} \cdot e^{-j\cdot 2\cdot \pi \cdot f/512}\} + M_0) && (3.27)\\
&= e^{-j\cdot 2\cdot \pi \cdot f/512} \cdot [2 \cdot (\Re\{M_{256}\} \cdot \cos(\frac{2\cdot\pi}{512}\cdot f)\\
&+ \Im\{M_{256}\} \cdot \sin(\frac{2\cdot\pi}{512}\cdot f)) + M_0] && (3.28)\\
&= e^{-j\cdot 2\cdot \pi \cdot f/512} \cdot [2 \cdot \sqrt{\Re\{M_{256}\}^2 + \Im\{M_{256}\}^2} \cdot (\frac{\Re\{M_{256}\}}{\sqrt{\Re\{M_{256}\}^2 + \Im\{M_{256}\}^2}}\\
&\cdot \cos(\frac{2\cdot\pi}{512}\cdot f) + \frac{\Im\{M_{256}\}}{\sqrt{\Re\{M_{256}\}^2 + \Im\{M_{256}\}^2}} \cdot \sin(\frac{2\cdot\pi}{512}\cdot f)) + M_0] && (3.29)\\
&= e^{-j\cdot 2\cdot \pi \cdot f/512} \cdot [2 \cdot ||M_{256}|| \cdot (\cos(\theta - \frac{\pi\cdot f}{256})) + M_0], && (3.30)
\end{aligned}
$$

where $\theta = -\arctan\frac{\Im\{M_{256}\}}{\Re\{M_{256}\}} = \delta\cdot\pi$. Therefore, the peak value happens when $\theta - \frac{\pi\cdot f}{256} = \delta\cdot\pi - \frac{\pi\cdot f}{256} = 0$, and then, $\delta = 0.0039\cdot f$. Note that the FFT size corresponds to the resolution of estimating δ and the resolution of this derivation is 0.0039. Moreover, we can conclude $\delta = -\frac{1}{\pi}\arctan\frac{\Im\{M_{256}\}}{\Re\{M_{256}\}}$, and this final result is quite similar to that of the Moose algorithm [8].

3.2.3 Jointly Integral CFO, PA-Preamble Index, Channel Estimation and Fine Timing Offset Searching

CFO is separated into two parts, FCFO and integral carrier frequency offset (ICFO), and the former have been estimation in the previous subsection. We can expect that the power of channel impulse response (CIR), the inverse fourier transform of **H** as obtained in (3.14), will be more concentrated if we compensate with the accurate CFO and use the correct one of the three possible PA-Preamble symbols. For example, Figs. 3.6 and 3.7 depict two CIRs obtained from using a combination of correct CFO and correct PA-Preamble index and from

using a combination of incorrect values. The simulation environment we choose in Figs. 3.6 and 3.7 is PB channel, 120 km/h, 0 dB in SNR, the correct ICFO 8, the correct PID 1 (10-MHz), the wrong ICFO 6, and the wrong PID 0 (5-MHz). We consider there are 21 possible ICFO explained in the Eq. (3.31), 3 PA-Preamble symbols and 256 timing locations in CIR, therefore, $21 \times 3 \times 64 = 16128$ candidates in total. The method we use here is to do 64 points sum of squared CIR for these candidates and find out one which has the maximum of power sum. The reason why we choose the searching range of ICFO from -20 to 20 is that we assume a maximum mismatch of the local oscillator frequency of ± 80 ppm, so that a wireless system with carrier frequency 2.5 GHz ± 18.28 subcarriers of offset at the 10.9375 kHz subcarrier spacing of IEEE 802.16m, as given by

$$\frac{2.5G \cdot 80ppm}{10.94K} \approx 18.28. \tag{3.31}$$

For the fine timing, since it is reasonable to assume that the CIR is mostly concentrated over a length not exceeding the CP length, we decide the ICFO, the PA-Preamble index and the fine timing offset by finding which one of all candidates has the maximum power sum over the CP length.

3.2.4 Overall Block Diagram

In summary, Fig. 3.8 shows the resulting overall block diagram of the derived initial DL synchronization method.

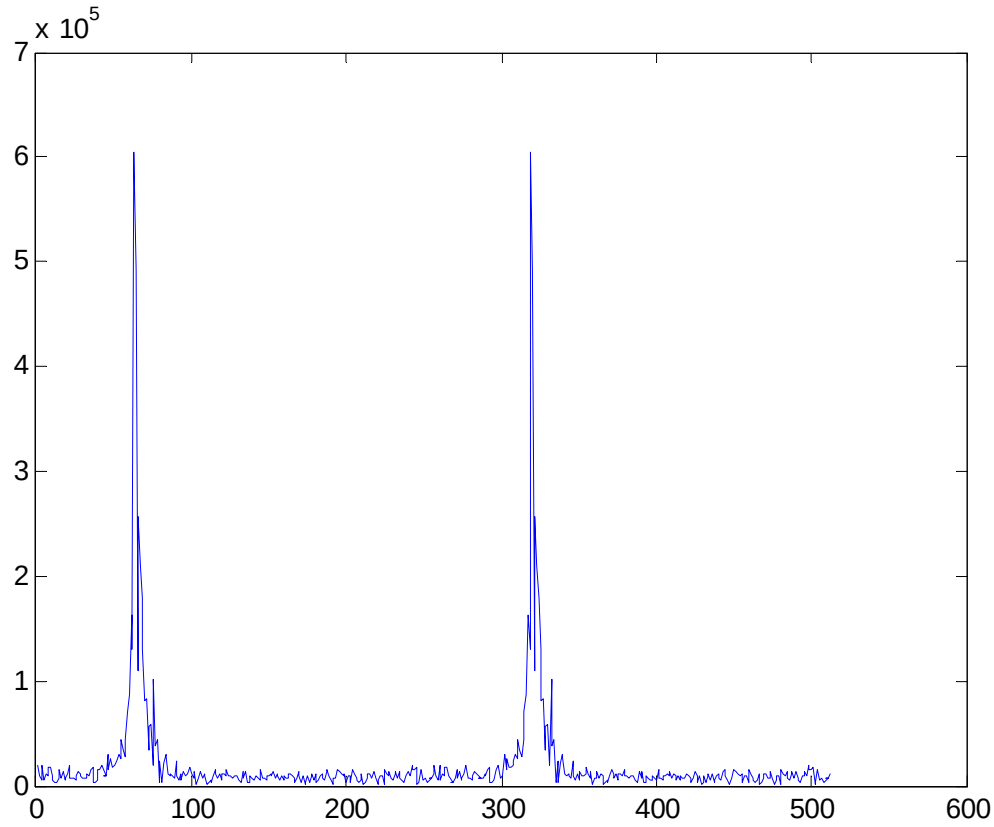

Figure 3.6: The estimated CIR with accurate ICFO, 8, compensating and correct PA-Preamble index, 1, under PB channel with 120 km/h, 0dB in SNR.

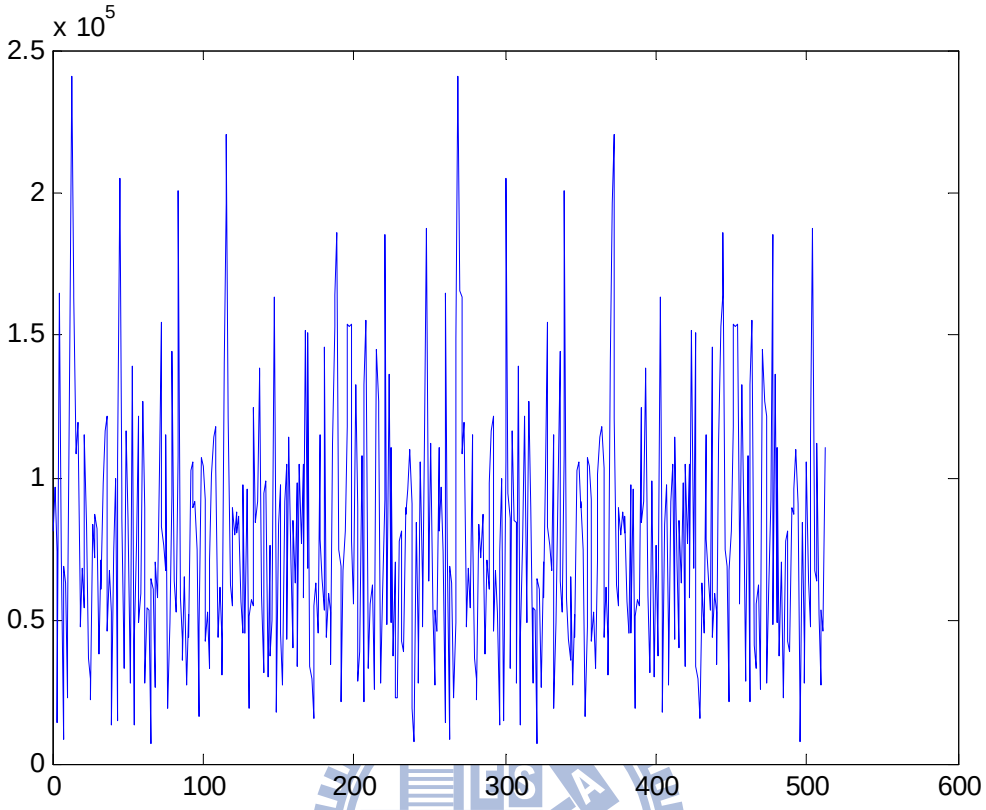

Figure 3.7: The CIR with the inaccurate ICFO, 6, compensating and incorrect PA-Preamble index, 0, under PB channel with 120 km/h, 0dB in SNR.

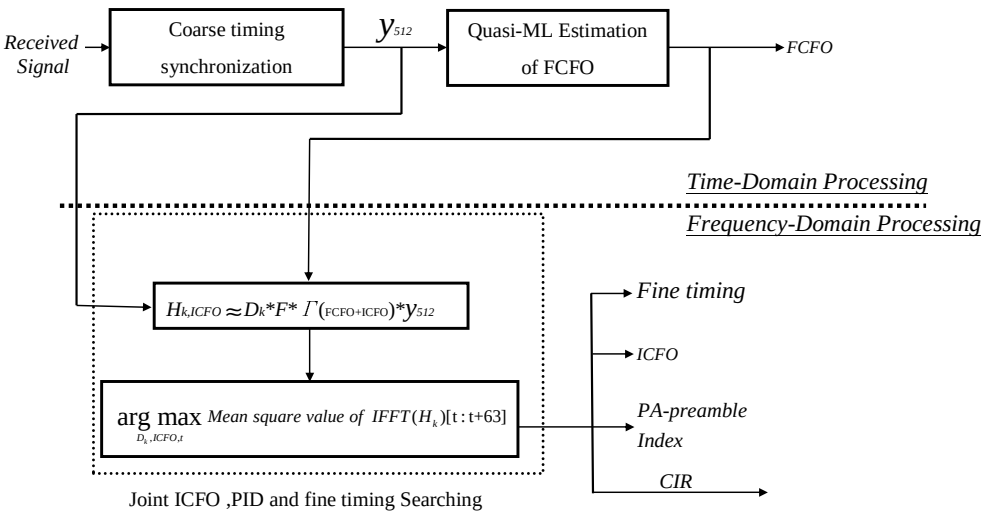

Figure 3.8: Block diagram of algorithm for initial DL synchronization [1].

Chapter 4

Introduction to the DSP Implementation Platform

In this chapter, we introduce the architecture of the DSP chip because we implement the initial synchronization on DSP chip. We use the DSP chip on the module is the TMS320C6416T made by Texas Instrument (TI). We introduce and the DSP chip, and what is more, we present the software development tool, Code Composer Studio (CCS), the code development technique.

4.1 The DSP Chip [11]

The TMS320C6416T DSP is a fixed-point DSP in the TMS320C64x series of the TMS320C6000 DSP platform family. It is based on the advanced VelociTI very-long-instruction-word (VLIW) architecture developed by TI. A functional block and DSP core diagram of TMS320C64x series is shown in Fig. 4.1.

The C64x core CPU consists of 64 general-purpose 32-bit registers and eight functional units. Features of C6000 device include the following.

- Eight functional units, including two multipliers and six arithmetic-logic units
  - Executes up to eight instructions per cycle
  - Allows designers to develop effective RISC-like code for fast development time

- Instruction packing
  - Gives code size equivalence for eight instructions executed serially or in parallel
  - Reduces code size, program fetches, and power consumption
- Conditional execution of all instructions
  - Reduces costly branching
  - Increases parallelism for higher sustained performance
- Efficient code execution on independent functional units
  - Efficient C complier on DSP benchmark suite
  - Assembly optimizer for fast development and improved parallelization
- 8/16/32-bit data support, providing efficient memory support for a variety of applications
- 40-bit arithmetic options add extra precision for vocoders
- 32×32-bit integer multiply with 32- or 64-bit result
- Saturation and normalization provide support for key arithmetic operations
- Field manipulation and instruction extract, set, clear, and bit counting support common operation found in control and data manipulation applications
- Each multiplier can perform two 16×16-bit or four 8×8 bit multiplies every clock cycle
- Quad 8-bit and dual 16-bit instruction set extensions with data flow support
- Support for non-aligned 32-bit (word) and 64-bit (double word) memory accesses

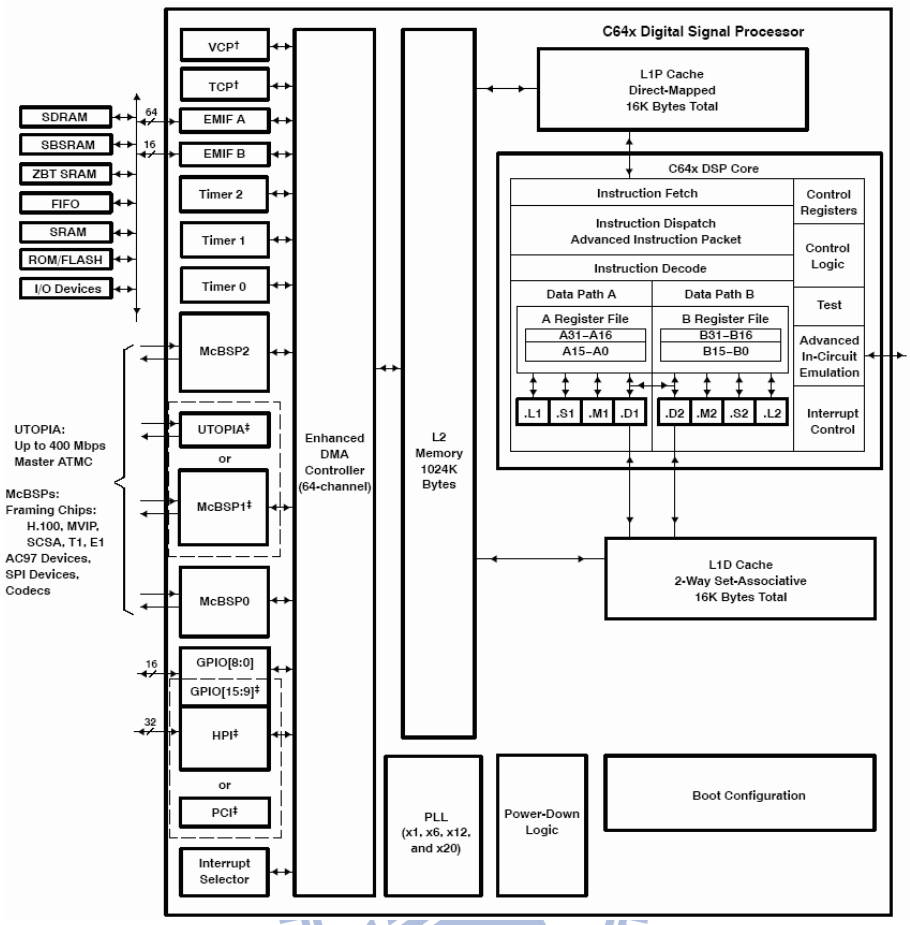

Figure 4.1: Functional block and CPU (DSP core) diagram [12].

- Special communication-specific instructions have been added to address common operations in error-correcting codes
- Bit count and rotate hardware extends support for bit-level algorithms

In the follwing subsections, we introduce three parts of the TMS320C64x DSP including CPU, memory, and peripherals.

4.1.1 Central Processing Unit

The C64x DSP core contains 64 32-bit general purpose registers, program fetch unit, instruction decode unit, two data paths each with four function units, control register, control logic,

advanced instruction packing, test unit, emulation logic and interrupt logic. The program fetch, instruction fetch, and instruction decode units can arrange eight 32-bit instructions to the eight function units every CPU clock cycle. The processing of instructions occurs in each of the two data paths (A and B) shown in Fig. 4.1, each of which contains four functional units and one register file. The four functional units are as follows: A multiplier (.M), a arithmetic and logic operations (.L), a unit for branch, byte shifts, and arithmetic operations (.S), and a unit for linear and circular address calculation to load and store with external memory operations (.D). The details of the functional units are described in Table 4.1.

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

4.1.2 Memory Architecture and Peripherals

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

The external memory interface (EMIF) provides interfaces for the DSP core and external memory, such as synchronous-burst SRAM (SBSRAM), synchronous DRAM (SRAM), SDRAM, FIFO and asynchronous memories (SRAM and EPROM). The EMIF also provides

Table 4.1: Functional Units and Operations Performed [11]

Parameter	Value
.L unit(.L1, .L2)	32/40-bit arithmetic and compare operations 32-bit logical operations Leftmost 1 or 0 counting for 32 bits Normalization count for 32 and 40 bits Byte shifts Data packing/unpacking 5-bit constant generation Dual 16-bit and Quad 8-bit arithmetic operations Dual 16-bit and Quad 8-bit min/max operations
.S unit (.S1, .S2)	32-bit arithmetic operations 32/40-bit shifts and 32-bit bit-field operations 32-bit logical operations Branches Constant generation Register transfers to/from control register file (.S2 only) Byte shifts Data packing/unpacking Dual 16-bit and Quad 8-bit compare operations Dual 16-bit and Quad 8-bit saturated arithmetic operations
.M unit (.M1, .M2)	16 x 16 multiply operations 16 x 32 multiply operations Dual 16 x 16 and Quad 8 x 8 multiply operations Dual 16 x 16 multiply with add/subtract operations Quad 8 x 8 multiply with add operations Bit expansion Bit interleaving/de-interleaving Variable shift operations Rotation Galois Field Multiply
.D unit (.D1, .D2)	32-bit add, subtract, linear and circular address calculation Loads and stores with 5-bit constant offset Loads and stores with 15-bit constant offset(.D2 only) Loads and stores doubles words with 5-bit constant Loads and store non-aligned words and double words 5-bit constant generation 32-bit logical operations

64-bit-wide (EMIFA) and 16-bit-wide (EMIFB) memory read capability.

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

4.2 TI's Code Development Environment [13]

The Code Composer Studio (CCS) is a key element of the DSP software and development tools from Texas Instruments. The tutorial [14] introduces the key features of CCS and the programmer's guide [15] gives a reference for programming TMS320C6000 DSP devices. A programmer needs to be familiar with coding development flow and CCS for building a new project on the DSP platform efficiently.

4.2.1 Code Composer Studio

The CCS combines the basic code generation tools with a set of debugging and real-time analysis capabilities which supports all phases of the development cycle shown in Fig. 4.2. Some main features of the CCS are listed below:

- Real-time analysis.
- Source code debugger common interface for both simulator and emulator targets.
  - C/C++ assembly language support.
  - Simple breakpoints.

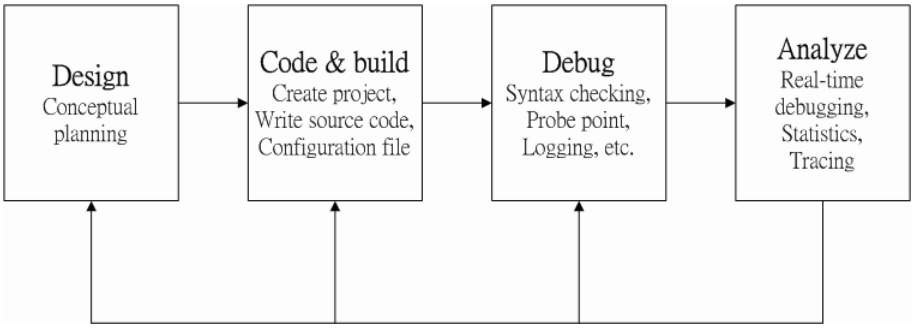

Figure 4.2: Code development cycle [14].

  - Advanced watch window.
  - Symbol browser.

- DSP/BIOS support.
  - Pre-emptive multi-threading.
  - Interthread communication.
  - Interupt handing.

- Chip Support Libraries (CSL) to simplify device configuration. CSL provides C-program functions to configure and control on-chip peripherals.

- DSP libraries for optimum DSP functionality. The DSP library includes many C-callable, assembly-optimized, general-purpose signal-processing and image/video processing routines. These routines are typically used in computationally intensive real-time applications where optimal execution speed is critical. The TMS320C64x digital signal processor library (DSPLIB) provides some routines for:
  - Adaptive filtering.
  - Correlation.
  - FFT.
  - Filtering and convolution.

- Math.
- Matrix functions.
- Miscellaneous.

Some of these routine is used in our implementation, such as FFT. We introduce it in a later chapter.

4.2.2 Code Development Flow [15]

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

4.3 Code Optimization on TI DSP Platform [15, 16]

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

- One of the preliminary measures of code is the time it takes the code to run. Use the clock( ) and printf( ) functions in C/C++ to time and display the performance of specific code regions. We can use the stand-alone simulator (load6x) to run the code for this purpose. We need to subtract out the overhead of calling the clock( ) function.

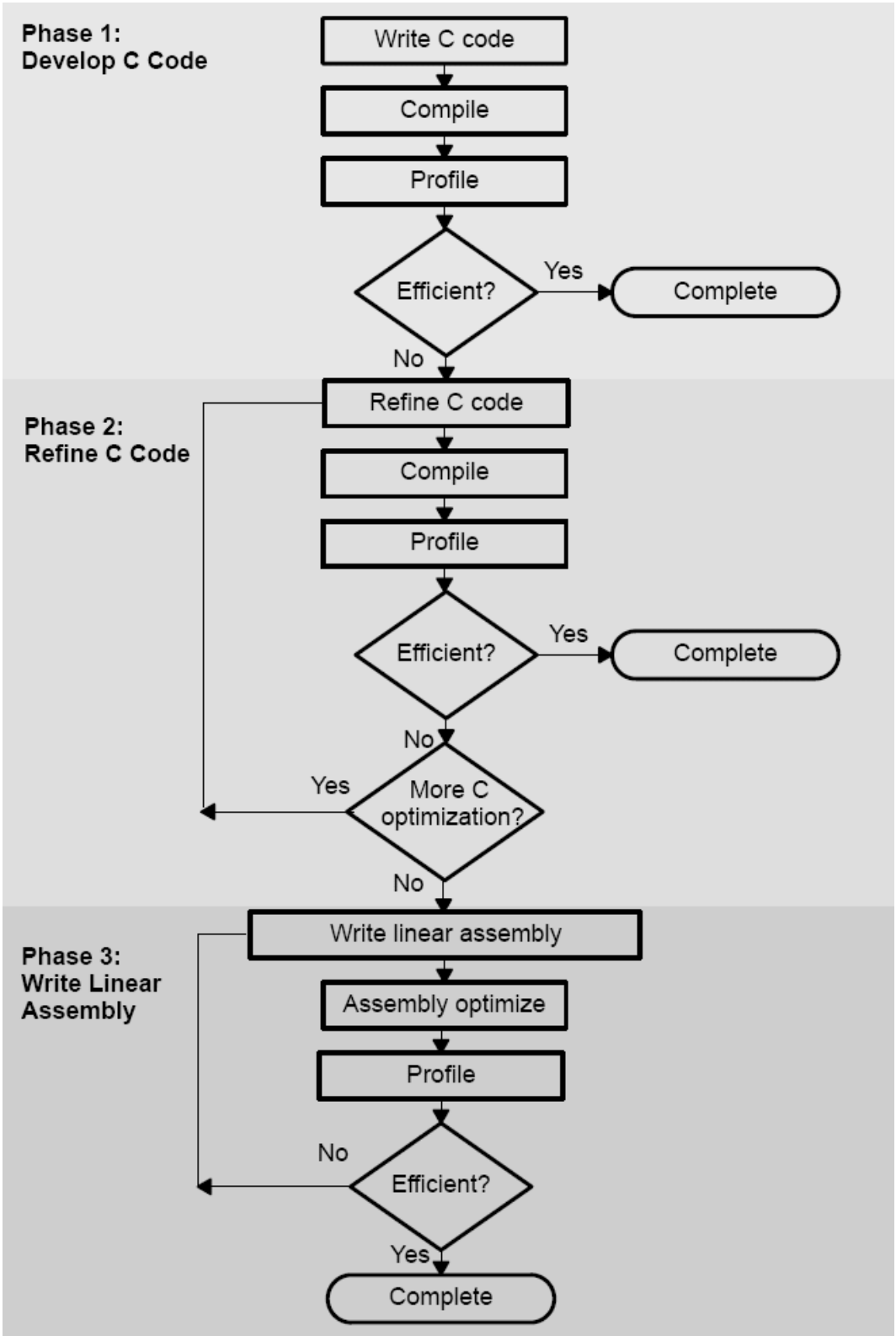

Figure 4.3: Code development flow for C6000 (from [15]).

- Use the profile mode of the stand-alone simulator. This can be done by executing load6x with the –g option. The profile results will be stored in a file with the .vaa extension. One may refer to the TMS320C6000 Optimizing Compiler Users Guide for more information.

- Enable the clock and use profile points and theRUNcommandin theCode Composer debugger to track the number of CPU clock cycles consumed by a particular section of code. One may use View Statistics to view the number of cycles consumed.

- The critical performance areas in a code are most often loops. An easiest way to optimize a loop is by extracting it into a separate file that can be rewritten, recompiled, and run with the stand-alone simulator (load6x).

We can also evaluate the performance results by running the code and looking at the instructions generated by the compiler.

4.3.1 Compiler Optimization Options

In this subsection, we introduce the compiler options that control the operation of the compiler. The C6000 compiler offers high-level language support by transforming a C/C++ code into more efficient assembly language source code. The compiler tools include a shell program (cl6x), which can be used use to compile, assembly optimize, assemble, and link programs in a single step. To compiler shell can be invoked by issuing the command

cl6x [options] [filenames] [-z [linker options] [object files]]

For a complete description of the C/C++ compiler and the options discussed in [15], see the TMS320C6000 Optimizing Compiler User Guide [14]. The major compiler options we use are -o3, -k, -pm -op2, -mh<n>, -mw, and -mi.

- -o*n*: The "*n*" denotes the level of optimization (0, 1, 2, and 3), which controls the type and degree of optimization.

  - -o3: highest level optimization, whose main features are:
    * Performs software pipelining.
    * Performs loop optimizations, and loop unrolling.
    * Removes all functions that are never called.
    * Reorders function declarations so that the attributes of called functions are known when the caller is optimized.
    * Propagates arguments into function bodies when all calls pass the same value in the same argument position.
    * Identifies file-level variable characteristics.

- -k: Keep the assembly file to analyze the compiler feedback.

- -pm -op2: In the CCS compiler option, -pm and -op2 are combined into one option.
  - -pm: Gives the compiler global access to the whole program or module and allows it to be more aggressive in ruling out dependencies.
  - -op2: Specifies that the module contains no functions or variables that are called or modified from outside the source code provided to the compiler. This improves variable analysis and allowed assumptions.

- -mh<n>: Allows speculative execution. The appropriate amount of padding, n, must be available in data memory to insure correct execution. This is normally not a problem but must be adhered to.

- -mw: Produce additional compiler feedback. This option has no performance or code size impact.

- -mi: Describes the interrupt threshold to the compiler. If the compiler knows that no interrupts will occur in the code, it can avoid enabling and disabling interrupts before and after software-pipelined loops for improvement in code size and performance. In

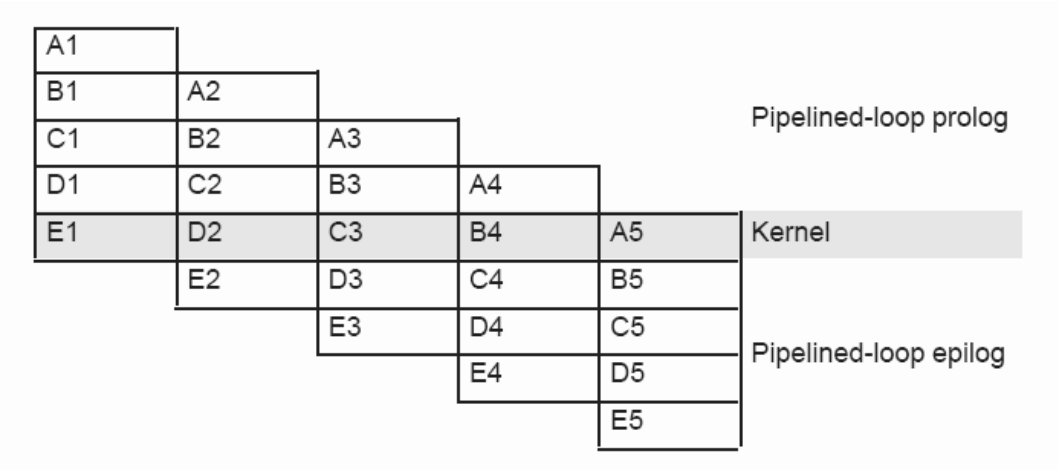

Figure 4.4: Software-pipelined loop (from [11]).

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

4.3.2 Software Pipelining

Software pipelining is a technique used to schedule instructions from a loop so that multiple iterations of the loop execute in parallel. When we use the -o2 and -o3 compiler options, the compiler attempts to software pipeline the code with information that it gathers from the program. Fig. 4.4 illustrates a software-pipelined loop. The stages of the loop are represented by A, B, C, D, and E. In this figure, a maximum of five iterations of the loop can execute at one time. The shaded area represents the loop kernel. In the loop kernel, all five stages execute in parallel. The area above the kernel is known as the pipelined loop prolog, and the area below the kernel is known as the pipelined loop epilog.

Because loops present critical performance areas in a code, the TI document advises one to consider the following areas to improve the performance of the C code:

- Trip count.
- Redundant loops.

- Loop unrolling.
- Speculative execution.

4.3.3 Loop Unrolling

Another technique that improves performance is unrolling the loop; that is, expanding small loops so that each iteration of the loop appears in the code. This optimization increases the number of instructions available to execute in parallel. We can use loop unrolling when the operations in a single iteration do not use all of the resources of the C6000 architecture. There are three ways loop unrolling can be performed:

- The compiler can automatically unroll the loop.
- The programmer can suggest that the compiler unroll the loop using the UNROLL pragma.
- The programmer can unroll the C/C++ code by self.

In our work, we use the compiler to help us loop unrolling itself.

Chapter 5

Fixed-Point Implementation of Initial Downlink Synchronization

In this chapter, we consider the fixed-point implementation of the initial downlink synchronization algorithm on DSP, and we present the simulation results, including both floating-point and fixed-point. Fig. 5.1 shows our simulation process, we use Matlab to simulate the wireless channel.

5.1 Floating-Point Simulation Results

In this section, we present the floating-point simulation results for C program, the system parameters for our simulation are defined in Table 5.1, and we modify the C code form Matlab code to do simulation under different channel environments and velocities. The power delay profiles (PDPs) used include Standford University Interim (SUI) [17] and Pedestrian B (PB) [9]. Our SNR values are from 0 to 20 dB, which is the ratio of the variance of PA-Preamble samples to that of the noise samples. The mobile velocity is from 0 to 120 km/h, and the carrier frequency offset (CFO) is 8.42884 subcarrier spacings, so the integral carrier frequency offset (ICFO) is 8 subcarrier spacings and fractional carrier frequency offset (FCFO) is 0.42884 subcarrier spacings. The simulation results are obtained with 1000 runs of simulation for each difference SNR.

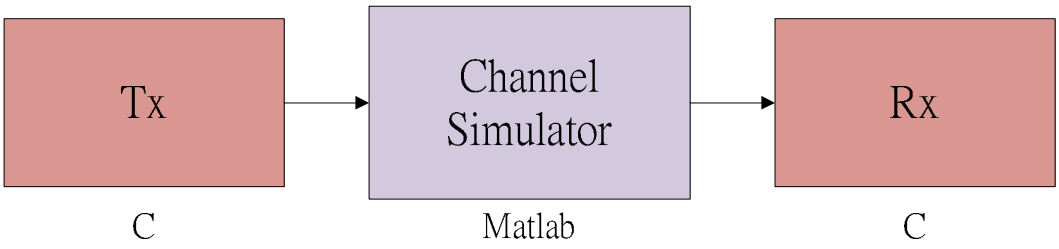

Figure 5.1: Block diagram of simulation procedure.

Table 5.1: System Parameters Used in Our Study

Parameters	Values		
System Channel Bandwidth (MHz)	5	10	20
Sampling Frequency (MHz)	5.6	11.2	22.4
FFT Size	512	1024	2048
Subcarrier Spacing (kHz)	10.94	10.94	10.94
Useful Symbol Time (μsec)	91.4	91.4	91.4
Guard Time (μsec)	11.4	11.4	11.4
OFDMA Symbol Time (μsec)	102.9	102.9	102.9

5.1.1 Coarse Timing Estimation

The target of coarse timing estimation is to find a staring timing sample for PA-Preamble, and the correct PA-Preamble time index is 576 in our simulation. Figs. 5.2 shows the histograms of coarse timing samples under AWGN channel at 0 dB and 10 dB, and it is clear that the higher SNR gives a better performance. Figs. 5.3 and 5.4 illustrate the histograms under SUI-1 at SNR values of 0 and 10 dB and velocities of 10 and 90 km/h, respectively. It is seen that SNR affects the performance more than the velocity. Figs. 5.5 and 5.6 illustrate the histograms under PB at similar SNR values and velocities. The correct timing index under SUI-1 and PB channel is 583, where the 6 samples difference with AWGN is due to the property of the Matlab function for simulating the multipath channel as discussed in chapter 3. The accuracy of coarse timing estimation affects the MSE of FCFO estimation.

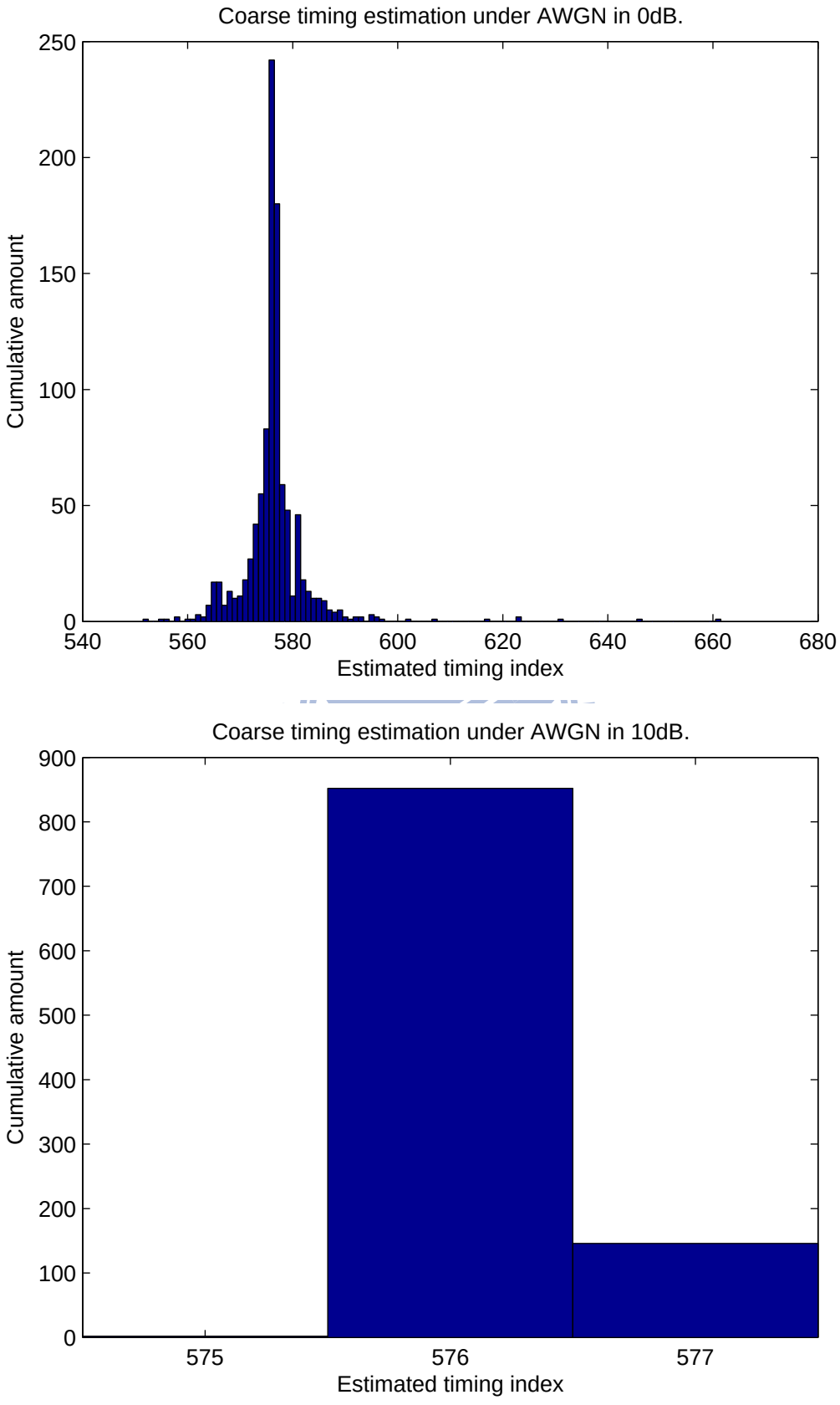

Figure 5.2: Histograms of coarse timing estimation under AWGN channel in different SNR.

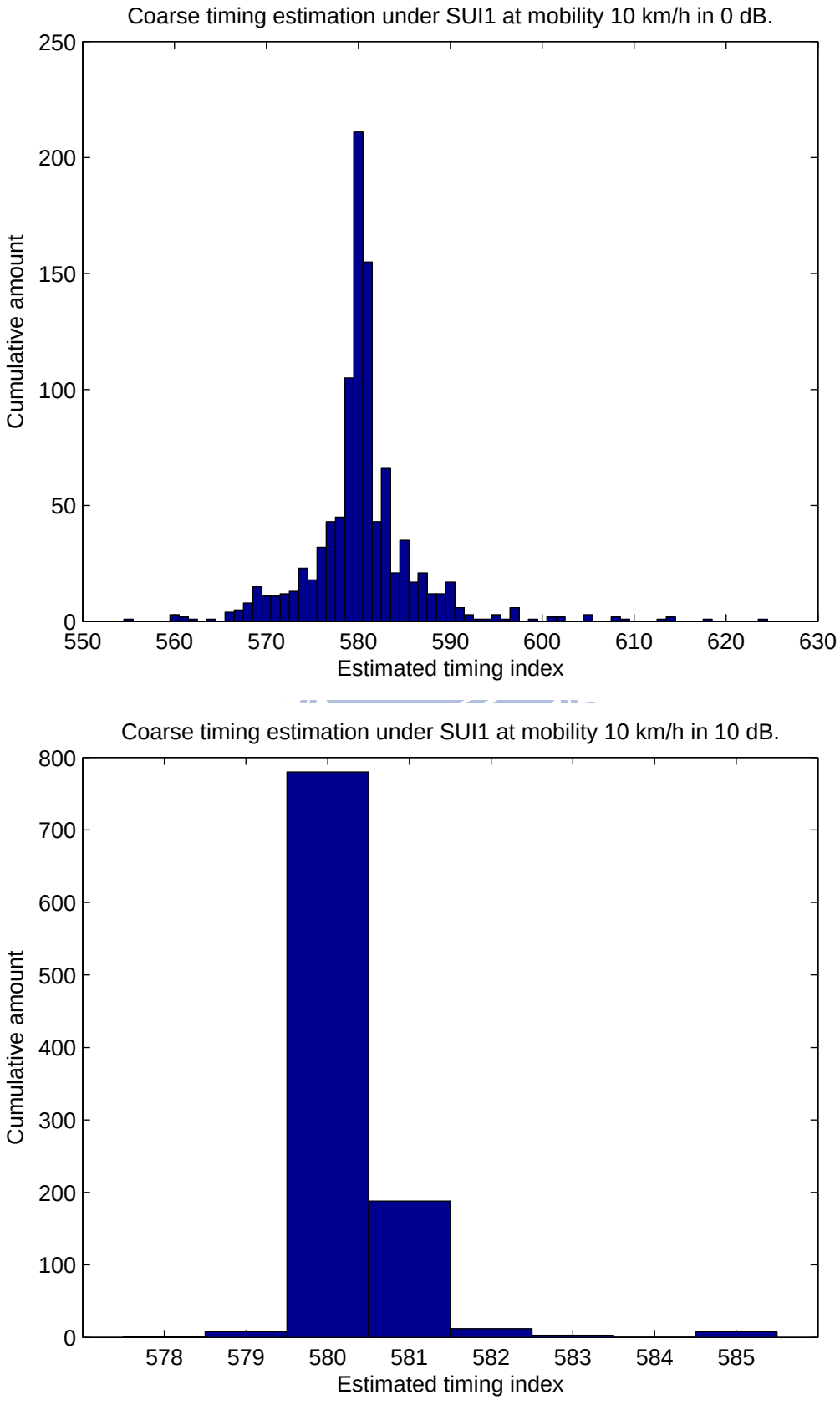

Figure 5.3: Histograms of coarse timing estimation under SUI-1 channel in different SNR value for a velocity of 10 km/h.

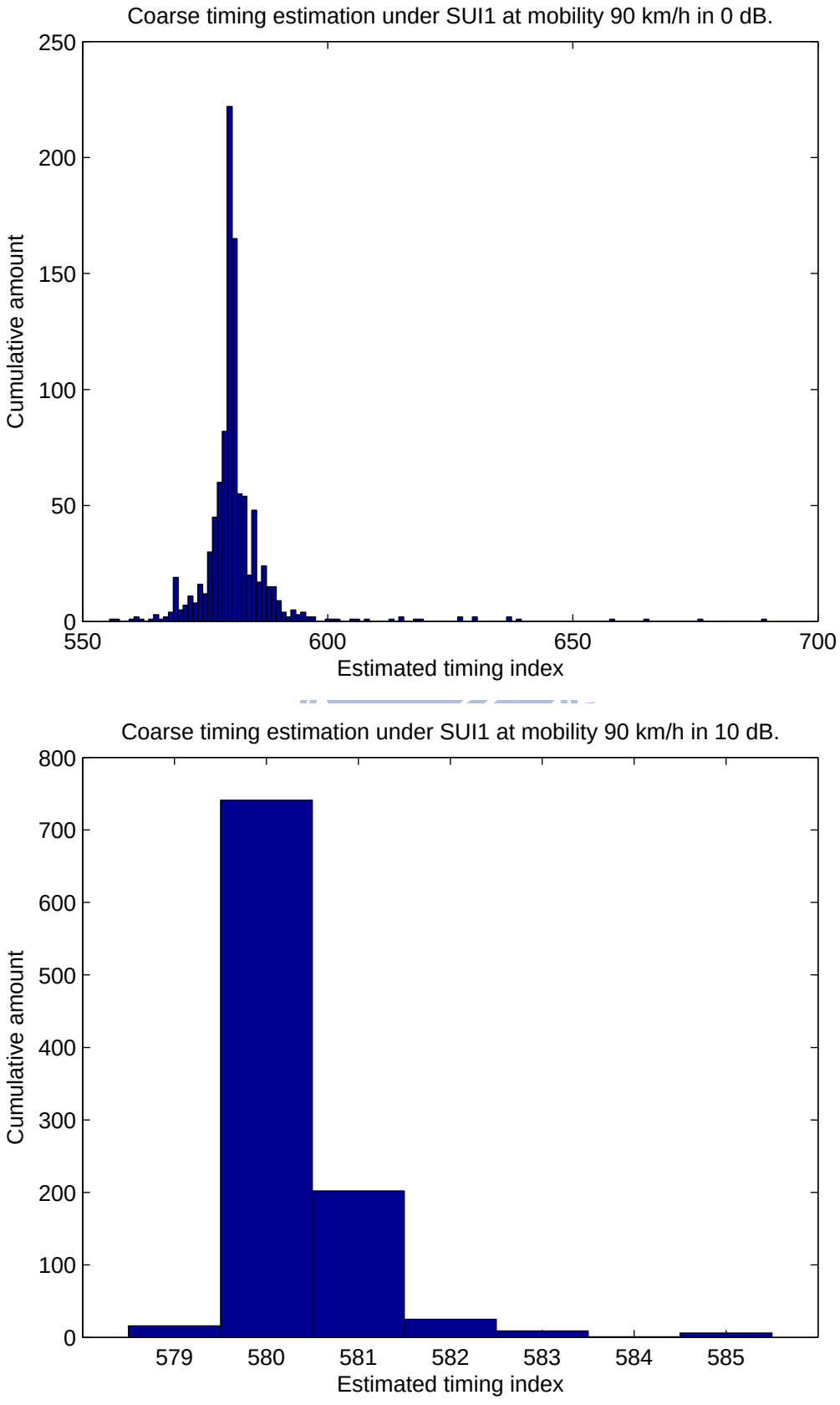

Figure 5.4: Histograms of coarse timing estimation under SUI-1 channel in different SNR value for a velocity of 90 km/h.

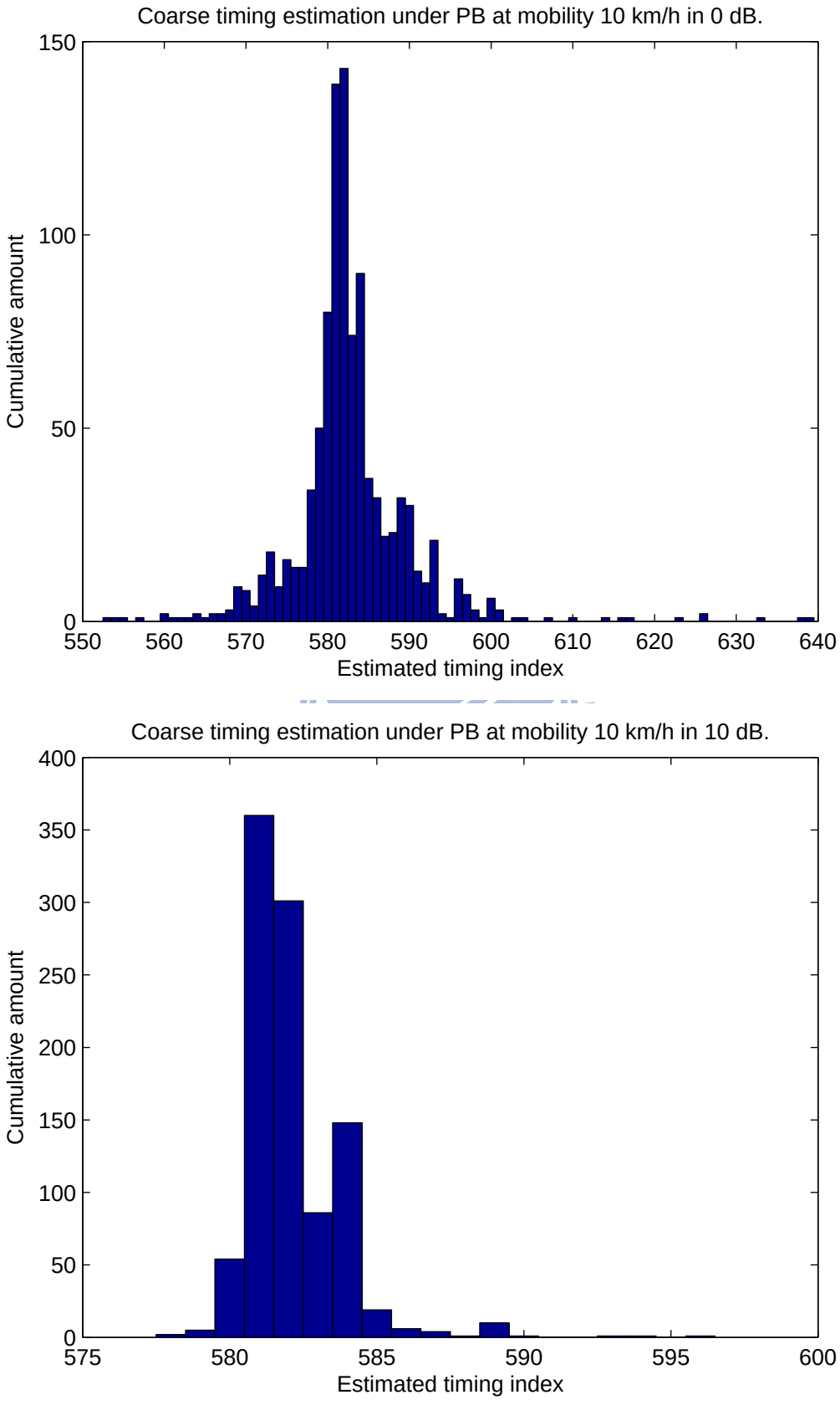

Figure 5.5: Histograms of coarse timing estimation under PB channel in different SNR value for a velocity of km/h.

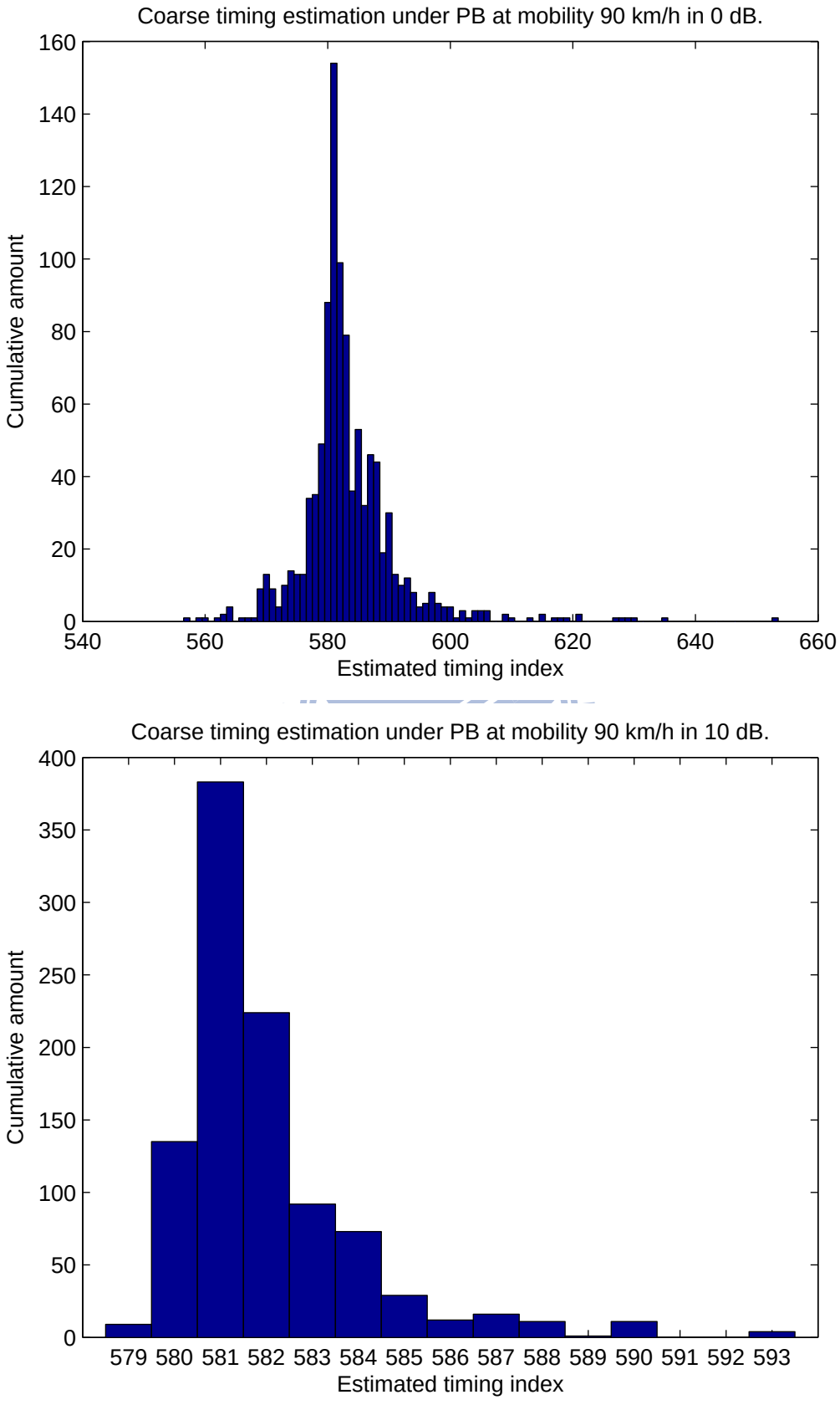

Figure 5.6: Histograms of coarse timing estimation under PB channel in different SNR value for a velocity of 90 km/h.

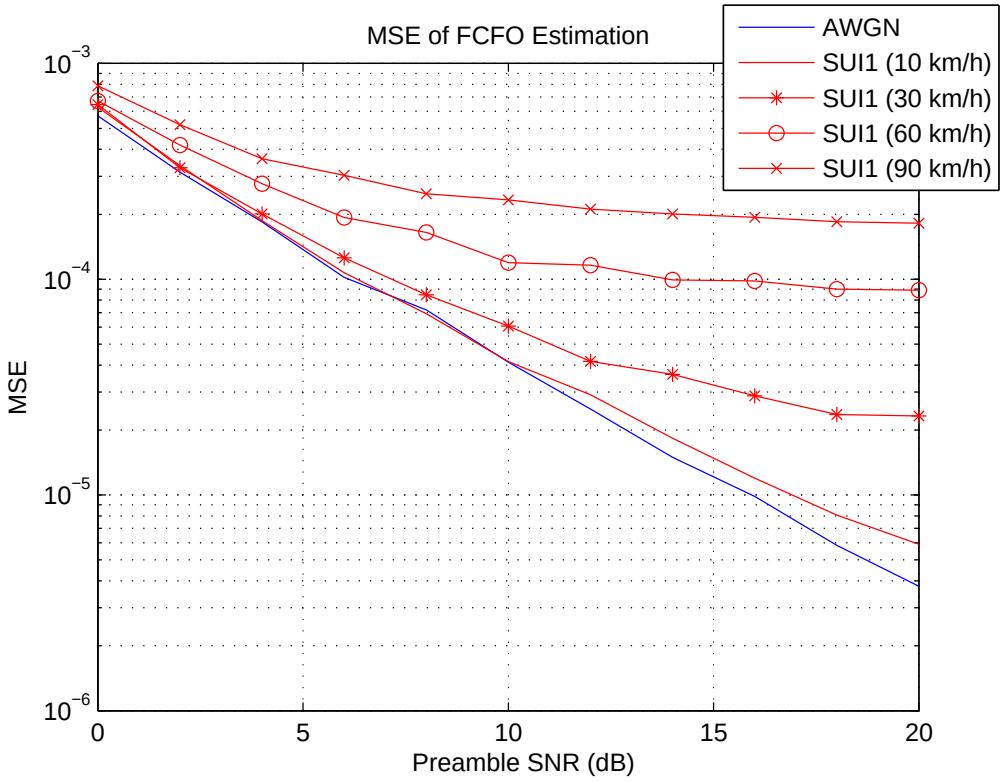

Figure 5.7: Mean square error of FCFO estimation under SUI-1 and AWGN channels.

5.1.2 Fractional CFO Estimation

Figs. 5.7 to 5.9 show the mean square error of fractional carrier frequency offset estimation under AWGN, SUI-1, SUI-3 and PB channels at different mobile velocities. The simulation results perform similar to results of reference [2], because our SNR definition is the same with [2].

5.1.3 Joint Estimation of Integral Carrier Frequency Offset, PID and Fine Timing

In this subsection, we present the simulation of ICFO estimation results in Figs. 5.10 to 5.12. The simulation parameters are:

- ICFO: 8 subcarrier spacings.

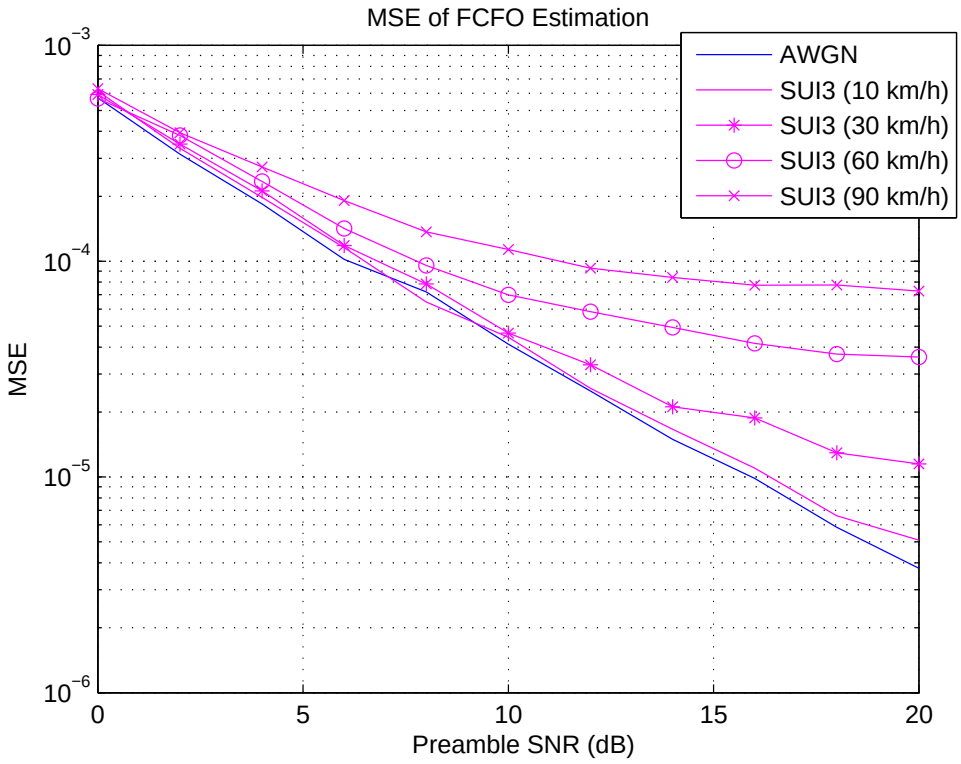

Figure 5.8: Mean square error of FCFO estimation under SUI-3 and AWGN channels.

- PA-Preamble index (PID): 1 (10MHz).
- Channel models: AWGN, SUI-1, PB.
- Mobile velocities: 10 km/h, 90 km/h.
- SNR value: 0 dB, 10 dB.

They illustrates the histograms under AWGN and SUI-1 channels at SNR values of 0 and 10 dB and velocities 10 and 90 km/h, respectively. Figs. 5.13 to 5.15 show the histograms of PID detection under AWGN and SUI-1 channels at similar SNR and velocity setting. They show that the ICFO and PID estimation are quite accurate at different SNR and channel conditions. Figs. 5.16 to 5.20 illustrate the performance of fine timing estimation under AWGN, SUI-1 and PB channels in 0 dB, 10 dB, and 20 dB of SNR at speeds 10 km/h and 90 km/h, respectively. We define "error" to be that the estimated timing index does

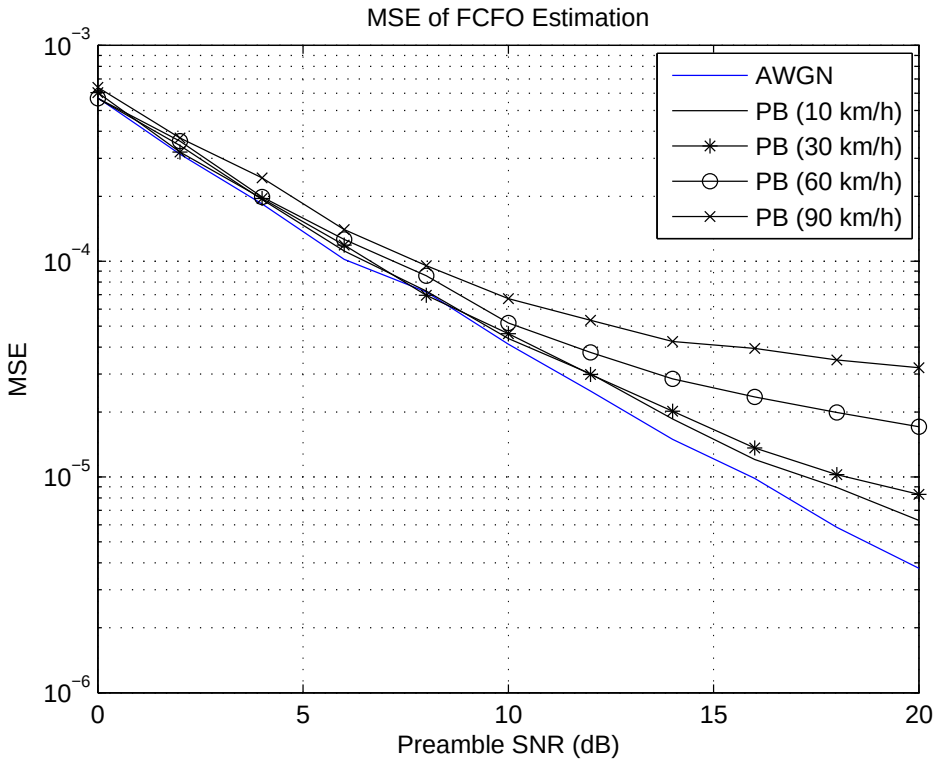

Figure 5.9: Mean square error of FCFO estimation under PB and AWGN channels.

Table 5.2: The error rate of timing estimation.

	AWGN	PB_10km	PB_90km	SUI1_10km	SUI1_90km
0 dB	0.013	0.113	0.118	0.011	0.021
10 dB	0	0.001	0.005	0	0
20 dB	0	0	0	0	0

not locate between the boundary of delay spread and the right-hand end of CP. Then we can calculate the error rate, show in Table 5.2. The simulation results of overall timing estimation is similar with reference [1].

5.2 Fixed-Point Implementation

Usually, we use floating-point processing to verity the performance of the algorithms. But fixed-point processing improves power efficiency, speed and hardware cost. Hance large-

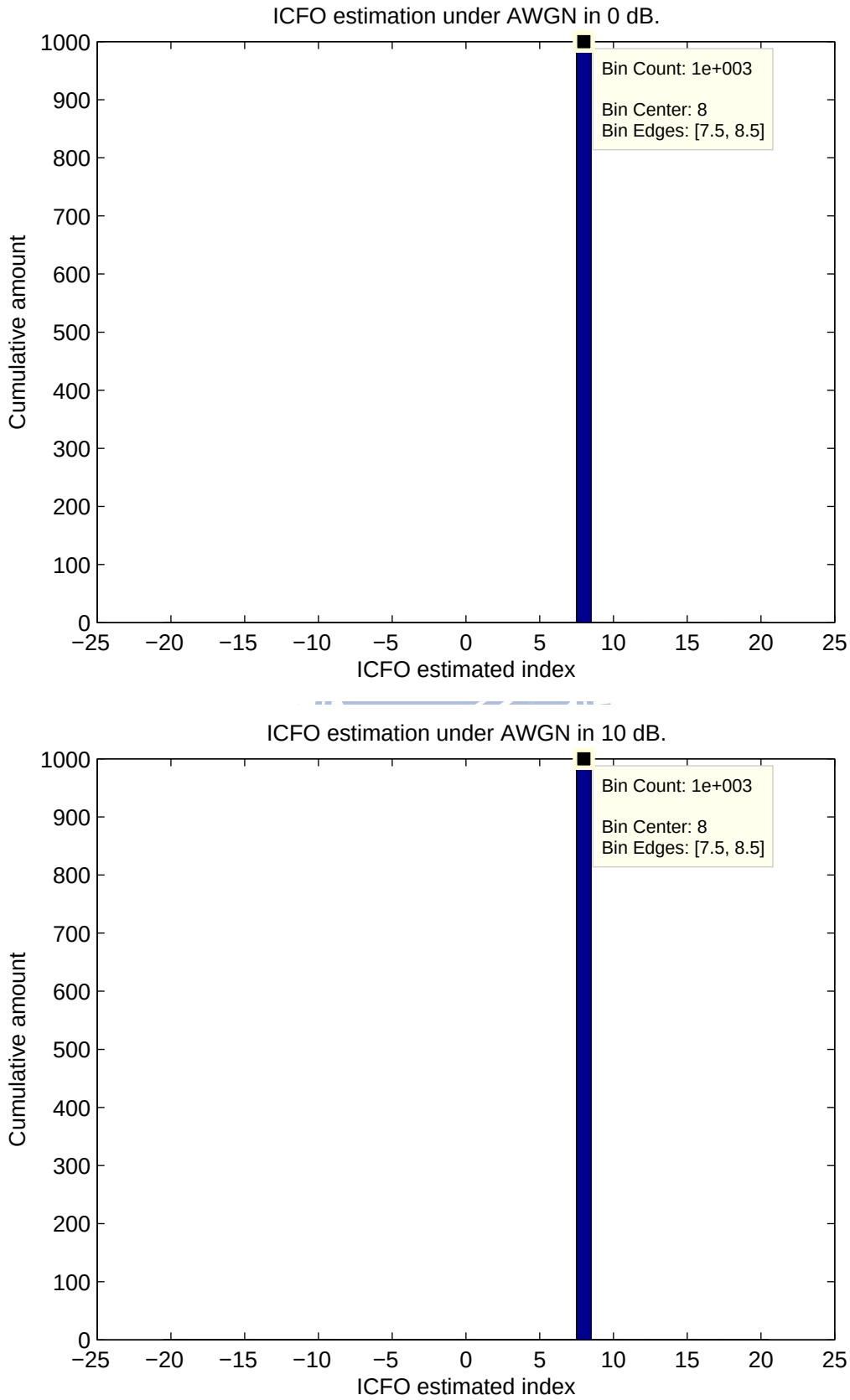

Figure 5.10: Histograms of integer CFO estimation under AWGN channel in different SNR values.

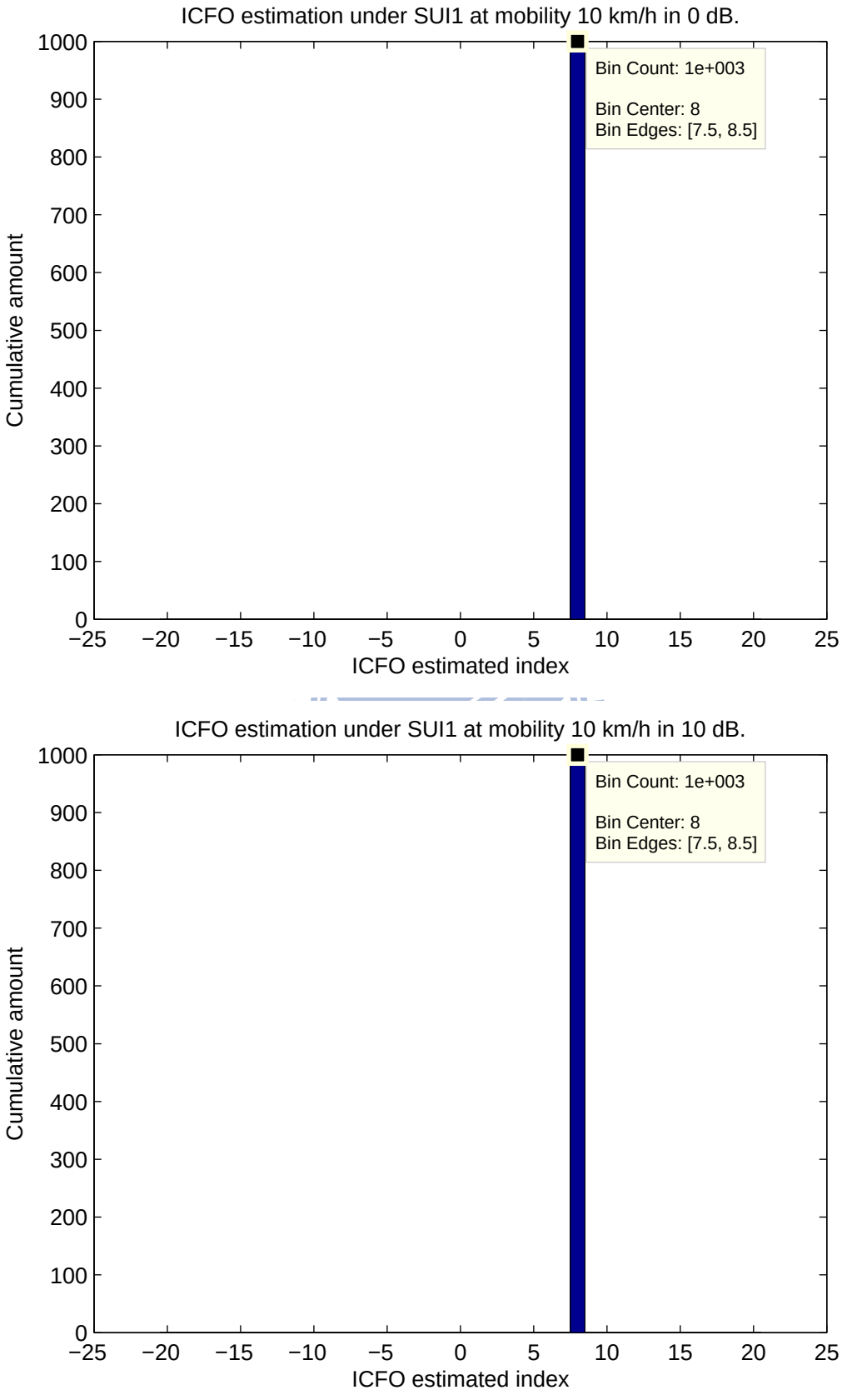

Figure 5.11: Histograms of integer CFO estimation under SUI-1 channel in different SNR values at a velocity of 10 km/h.

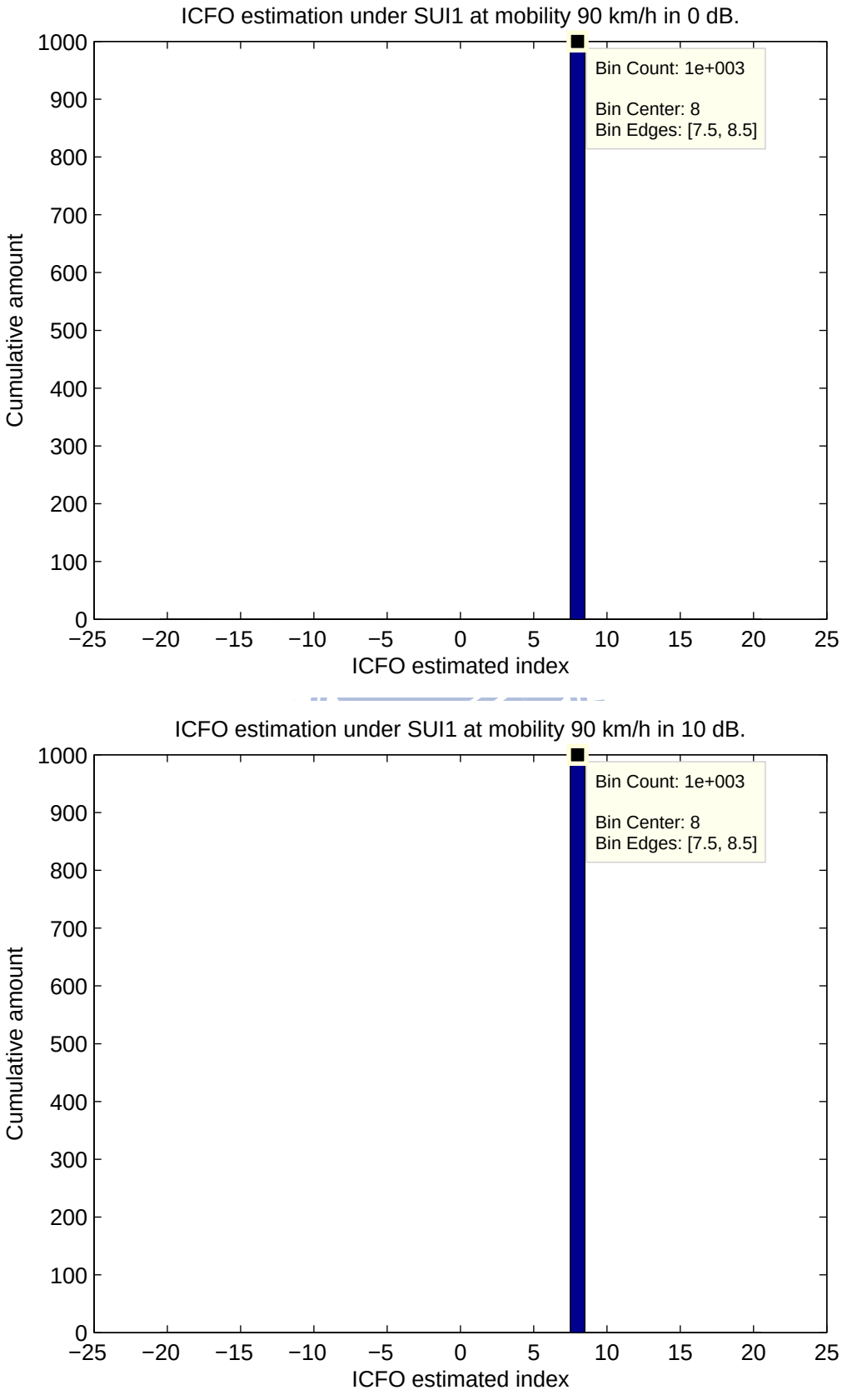

Figure 5.12: Histograms of integer CFO estimation under SUI-1 channel in different SNR values at a velocity of 90 km/h.

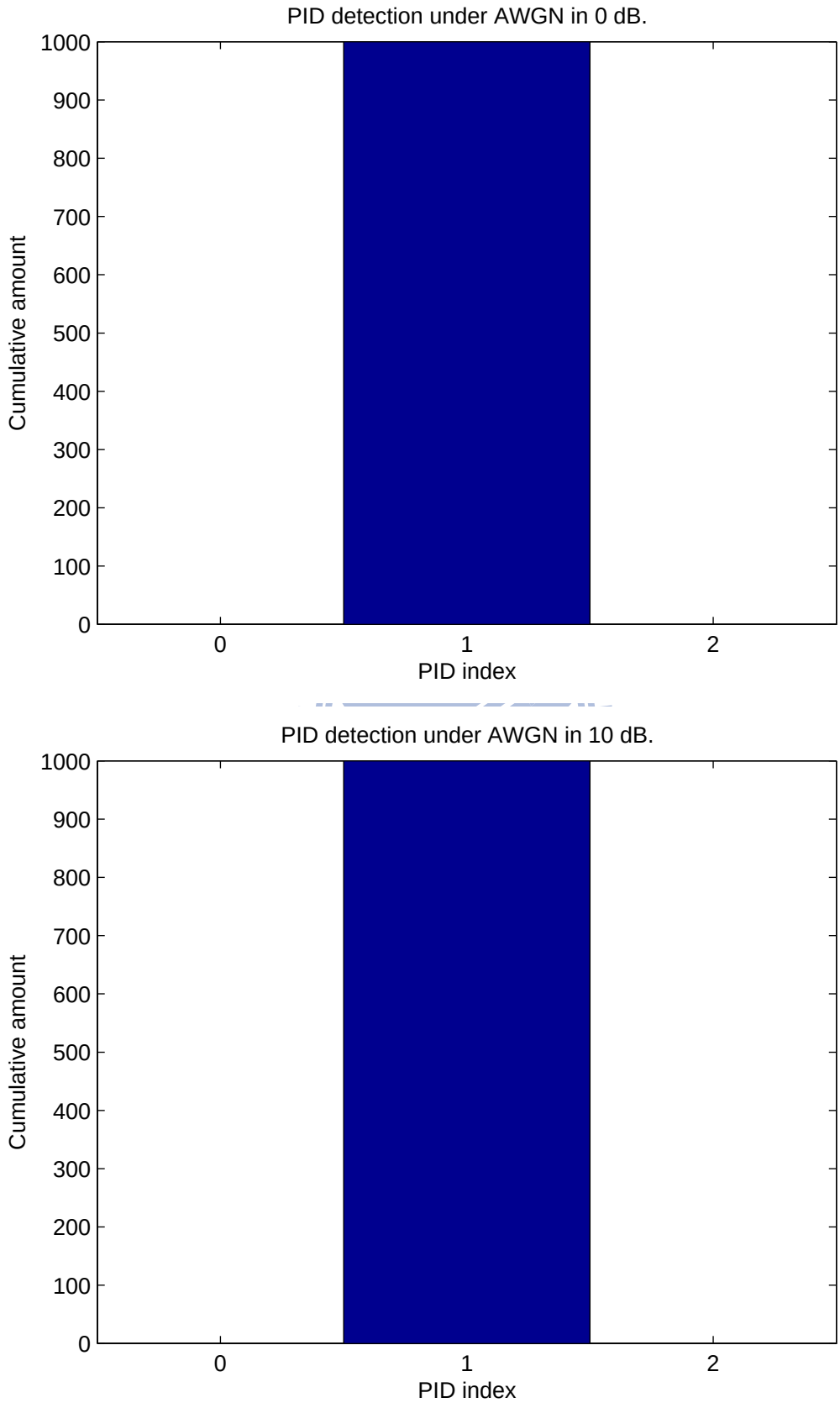

Figure 5.13: Histograms of PID detection under AWGN channel in different SNR values.

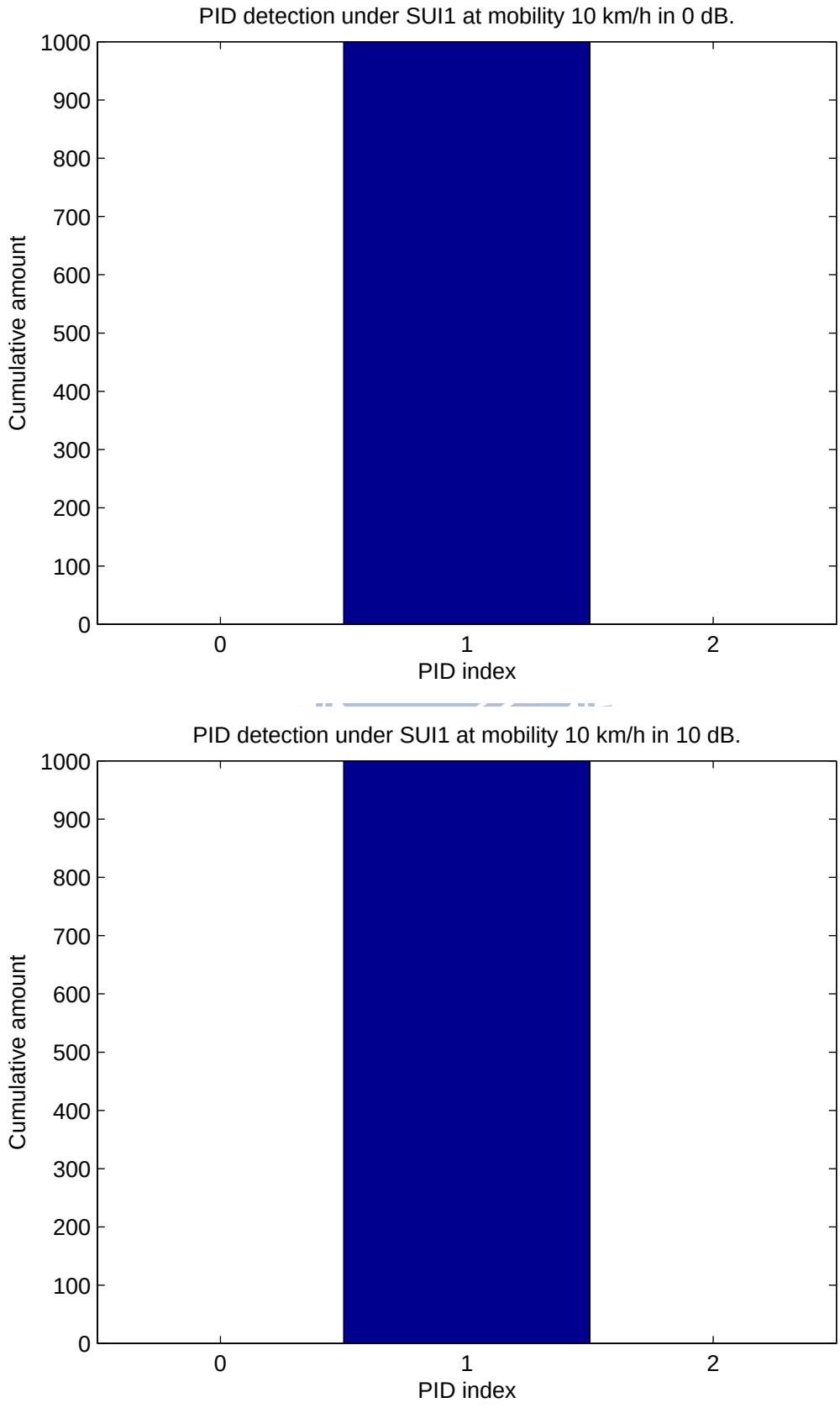

Figure 5.14: Histograms of PID detection under SUI-1 channel in different SNR values at a velocity of 10 km/h.

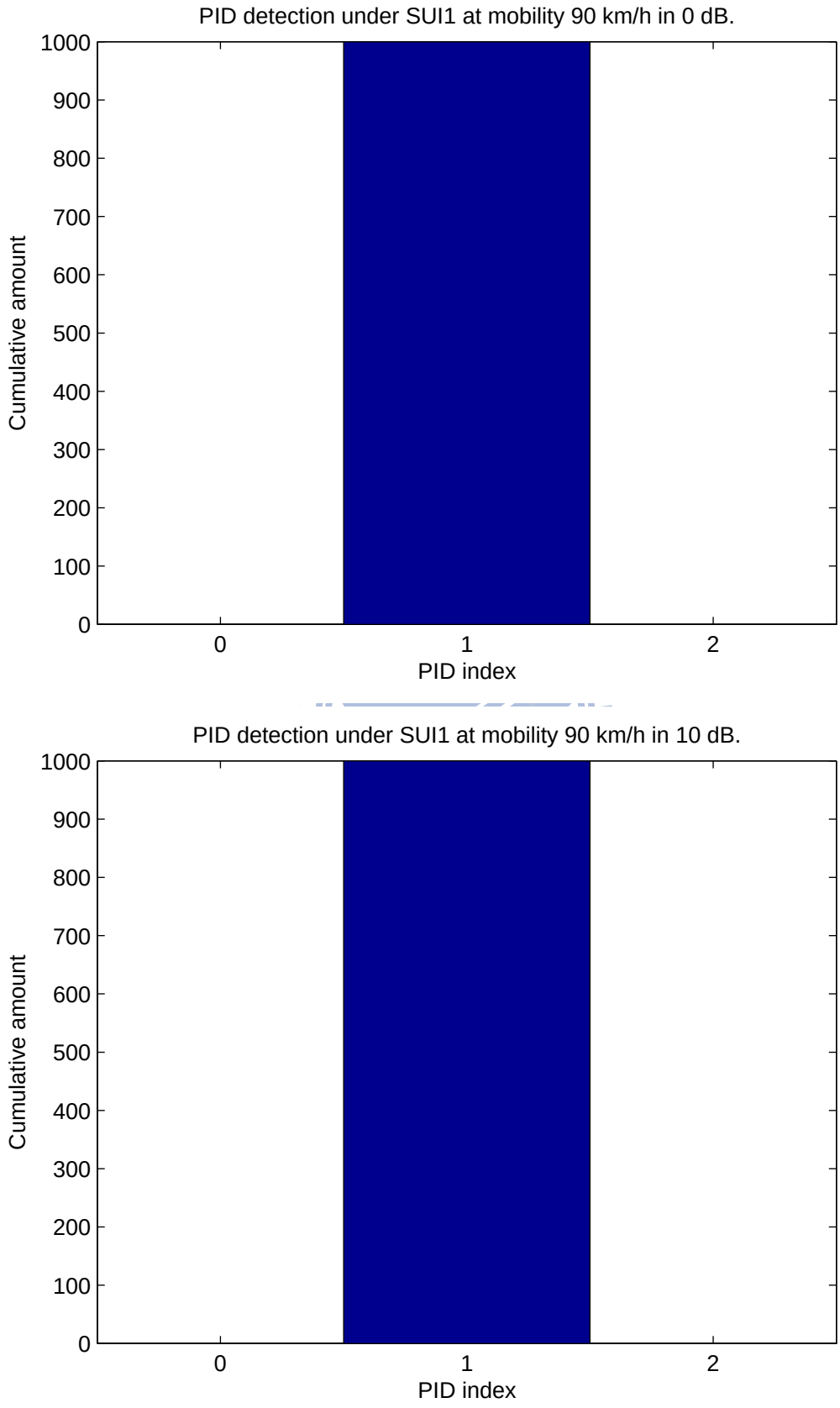

Figure 5.15: Histograms of PID detection under SUI-1 channel in different SNR values at a velocity of 90 km/h.

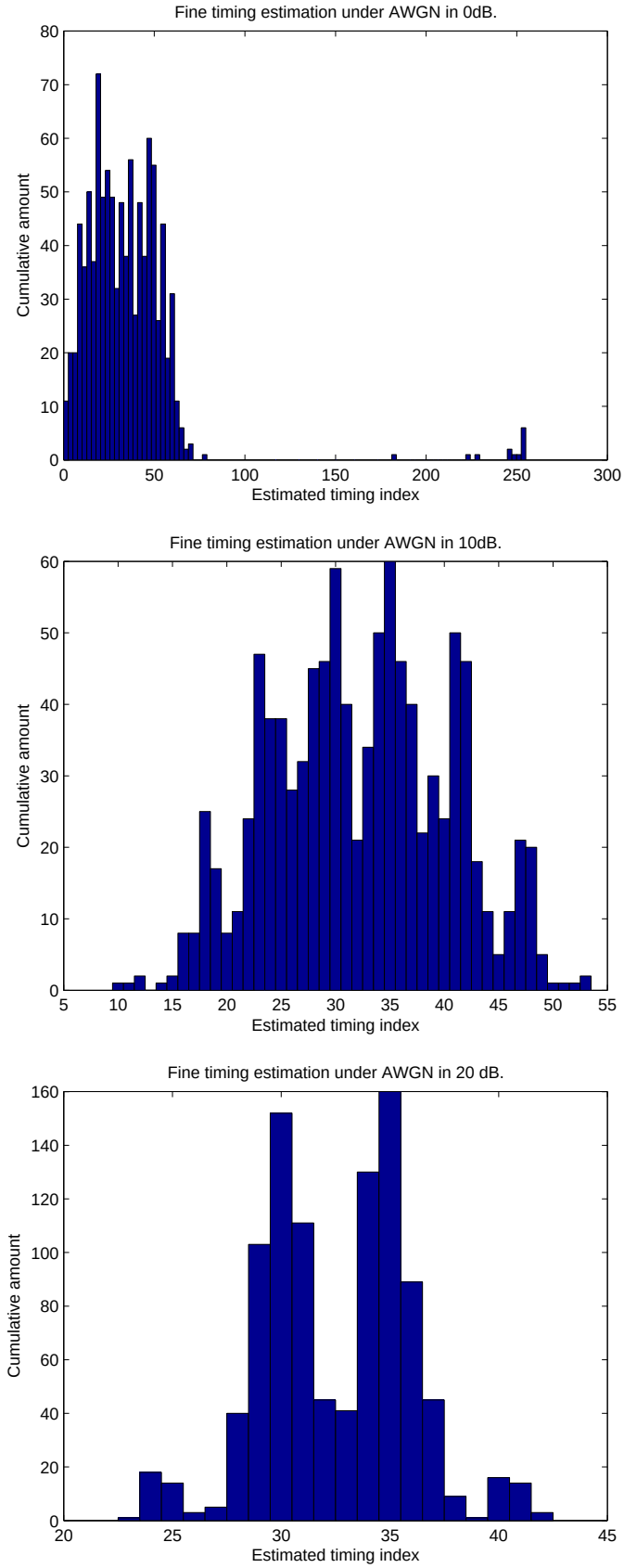

Figure 5.16: Histograms of fine timing estimation under AWGN channel in the different SNR values.

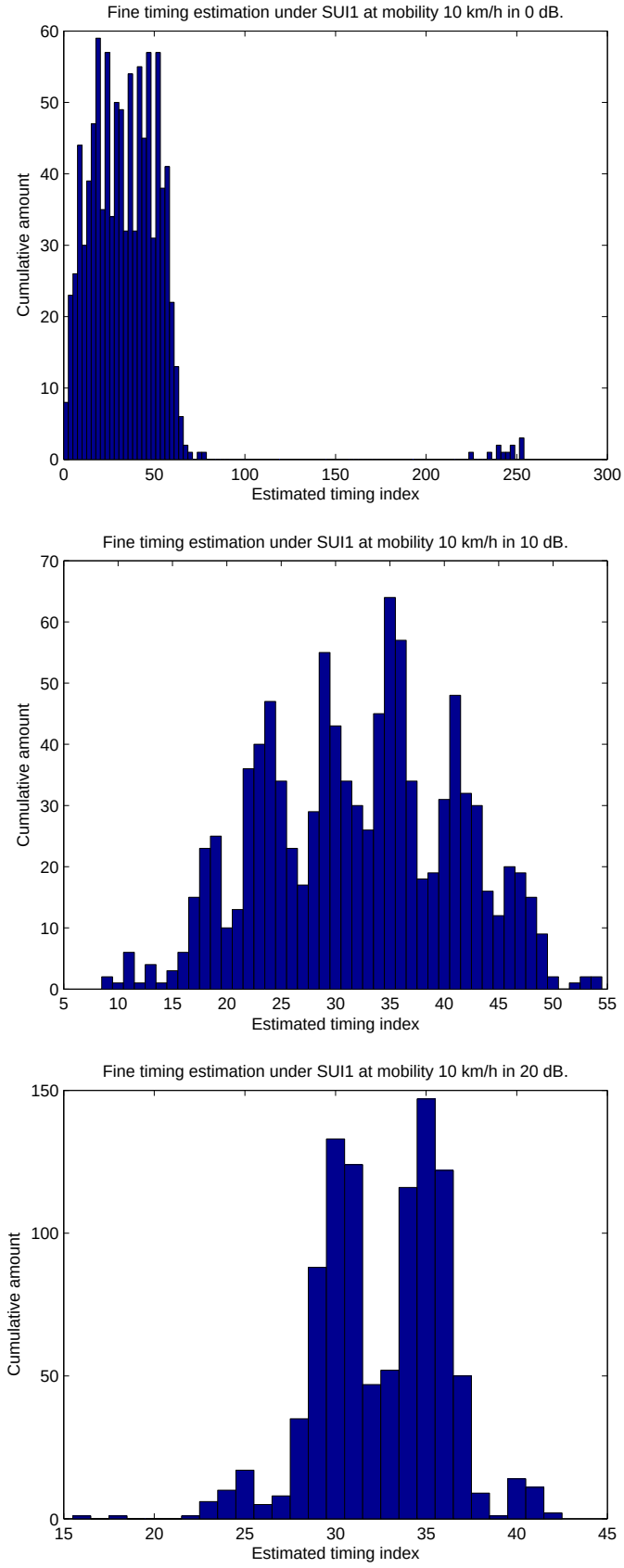

Figure 5.17: Histograms of fine timing estimation under SUI-1 channel in different SNR values at a velocity of 10 km/h.

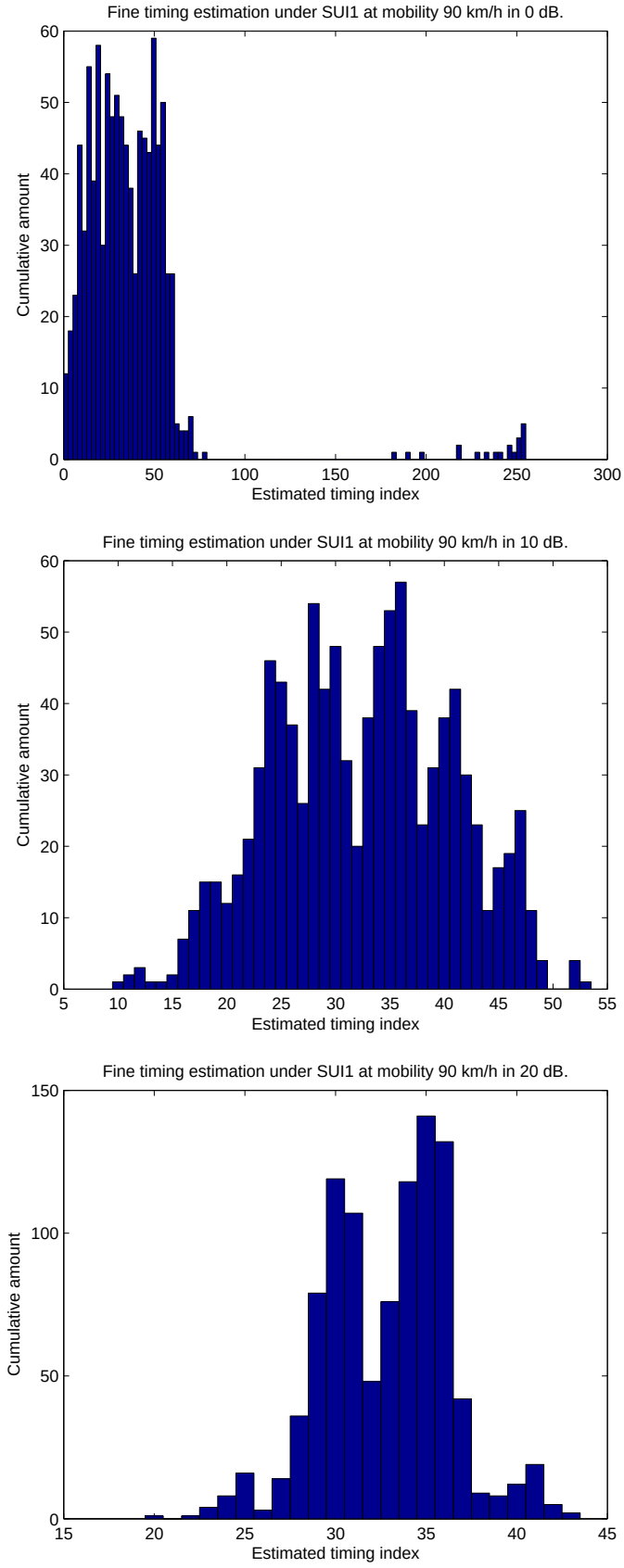

Figure 5.18: Histograms of fine timing estimation under SUI-1 channel in different SNR values at a velocity of is 90 km/h.

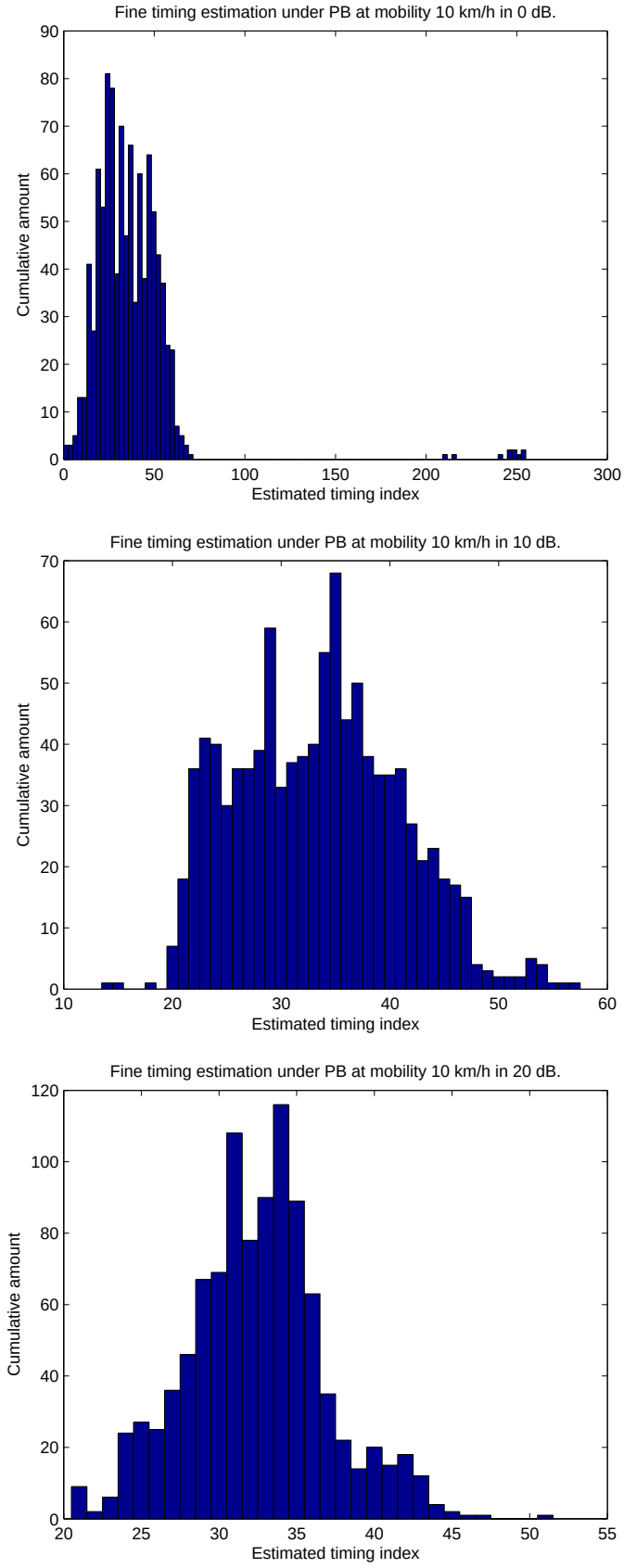

Figure 5.19: Histograms of fine timing estimation under PB channel in different SNR values at a velocity of 10 km/h.

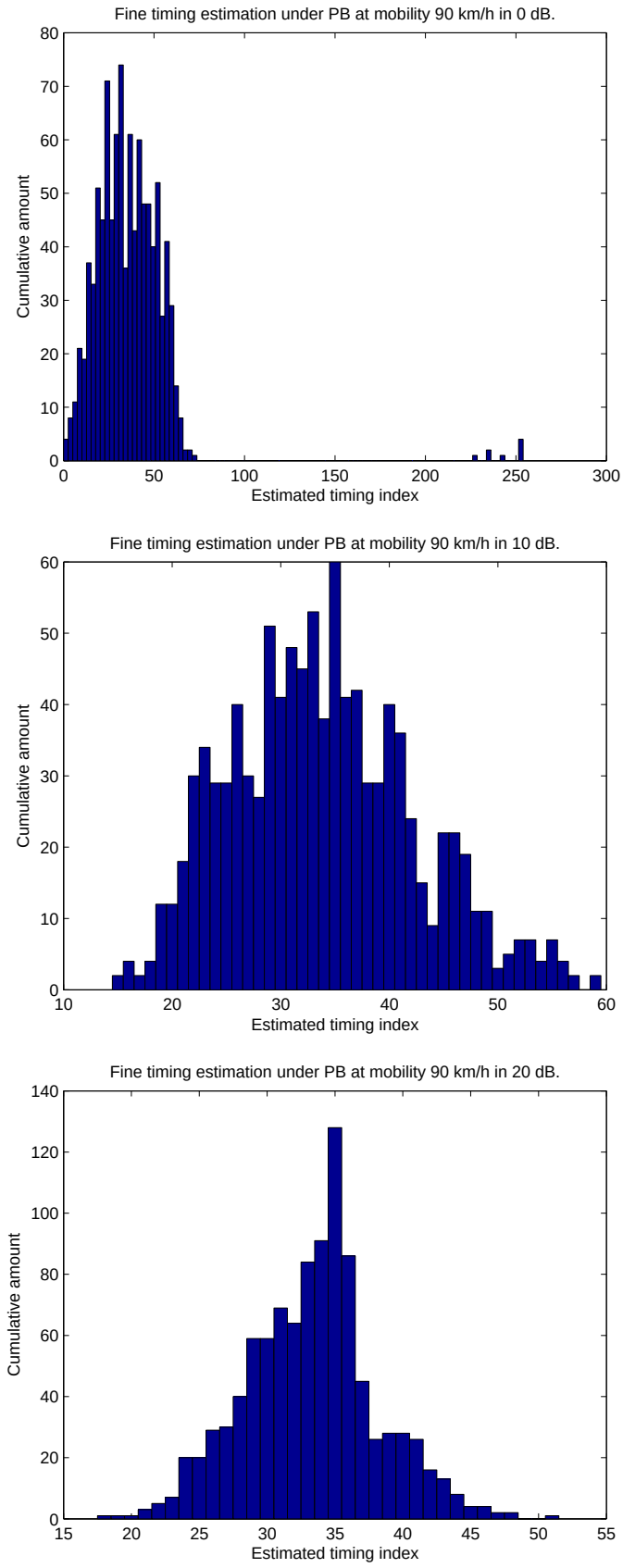

Figure 5.20: Histograms of fine timing estimation under PB channel in different SNR values at a velocity of 90 km/h.

volume practical implementation normally employ fixed-point processing. In this section, we present the initial downlink synchronization algorithm implementation in fixed-point processing using TI's TMS320C6416T DSP. We also try to utilize coding style and intrinsic functions to reduce cycle counts on DSP.

According to chapter 4, we know that the C6416T CPU has a VLIW architecture that contains 8 parallel 32-bits function units. The 8 units include two multipliers and six that can do a number of arithmetic, logic and memory access operations, and it is flexible so that each function unit can do double 16-bit or quadruple 8-bit operations. In our work, we choose 16-bit data type mostly, because 16-bit computation has enough accuracy for most of the functions we implement.

Fig. 5.21 shows the fixed-point data formats used in the different places in our algorithm, where Qx.y means there are x bits before the binary points and y bits after. In our case, x+y = 15 because the sign takes 1 bit. We choose Q7.8 to be the data format at many places, because coarse timing estimation needs to accumulate the squared norm of data. The Q7.8 format can avoid overflow in coarse timing estimation. In fact, we find that the Q7.8 data format has enough accuracy for our experiment. In the following subsections, we discuss the details of the blocks in the algorithm.

5.2.1 Coarse Timing Estimation and Removal of Cycle Prefix

The first step in the procedure is coarse timing estimation to find the approximate location of PA-Preamble. Figs. 3.1 shows our signal structure, where we compute the signal power in a finite window size and slide the window. According to the IEEE 802.16m standard, the PA-Preamble magnitude is boosted by a factor of 1.9216, 2.6731 or 4.6511 compared to regular data signal. To the maximum power position should be a good indicator of what the PA-Preamble is. After coarse timing estimation, we remove the CP from the 576 points starting at the estimated point to get 512 points of data. Actually, because the estimated point by coarse timing estimation may be located within the CP, what we in fact do is to take

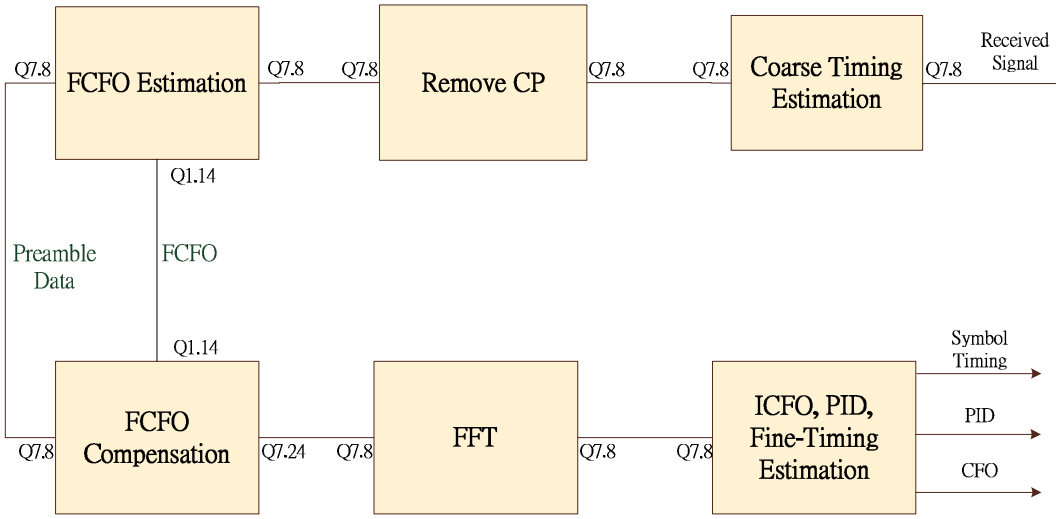

Figure 5.21: Fixed-point data formats used in DSP implementation.

the first 512 points starting from the coarse timing point, which is equivalent to discard the last 64 point of the 576 points, because it is more probable to get a complete PA-Preamble this way.

5.2.2 Fractional Carrier Frequency Offset Estimation and Compensation

FCFO estimation is the second step in the procedure. Fig. 5.22 shows that we correlate the first 256 points and the last 256 points of PA-Preamble to calculate the FCFO, which is obtained as the arc-tangent of the correlation. For efficiency in DSP implementation, we use a lookup table to implement the arctan() function. For dynamic range, we create a table for the arcsin() function to estimate the FCFO in place of a table of the arctan() function. The table contains 2048 entries uniformly spanning the range $[\sin 0, \sin 0.25\pi)$, and the table entries are normalized with respect π so that they span the range $[0, 0.25)$.

In frequency offset compensation, we create two lookup tables for the sin() and the cos() functions, each containing 2048 entries uniformly spanning the range $[0, \pi \div 2)$. Since the values of sin() and cos() are from -1 to 1, we choose Q.15 as the data format. Hence, when

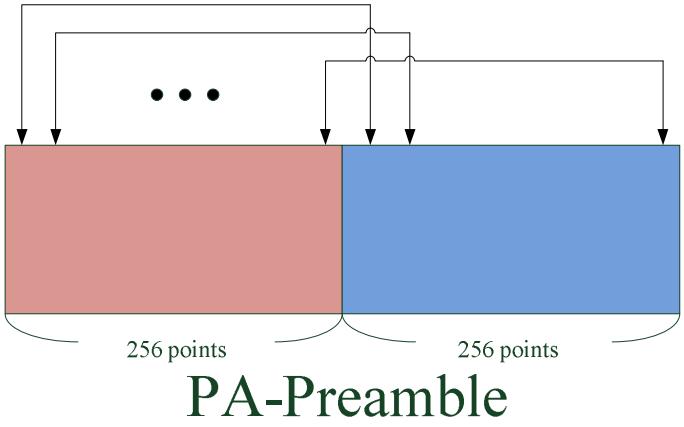

Figure 5.22: Calculating the correlation in received PA-Preamble.

the FCFO is compensated, the data format becomes Q7.24. Then we change the data format from Q7.24 to Q7.8 in order to avoid overflow in ICFO estimation.

5.2.3 Integer Carrier Frequency Offset Estimation and PID Detection

The last step of the procedure is ICFO estimation and PID detection. For this, we operate in the frequency domain. Since ICFO is just a shift in the subcarrier indexes in the frequency domain, it is relatively simple to implement in C program. According to (3.14), we calculate the channel frequency response and transform it to the time domain. Since the CIR length is supposed be not exceeding 64 points, we can assume that the correct choice of ICFO and PID should yield the maximum squared value, sum for the resulting CIR. The flow chart is shown in Fig. 5.23.

5.3 Fixed-Point Simulation Results

In this section, we show the fixed-point simulation results and compare them with the floating-point simulation results under different channel models. All simulation parameters and environments are the same as those given in section 5.1.

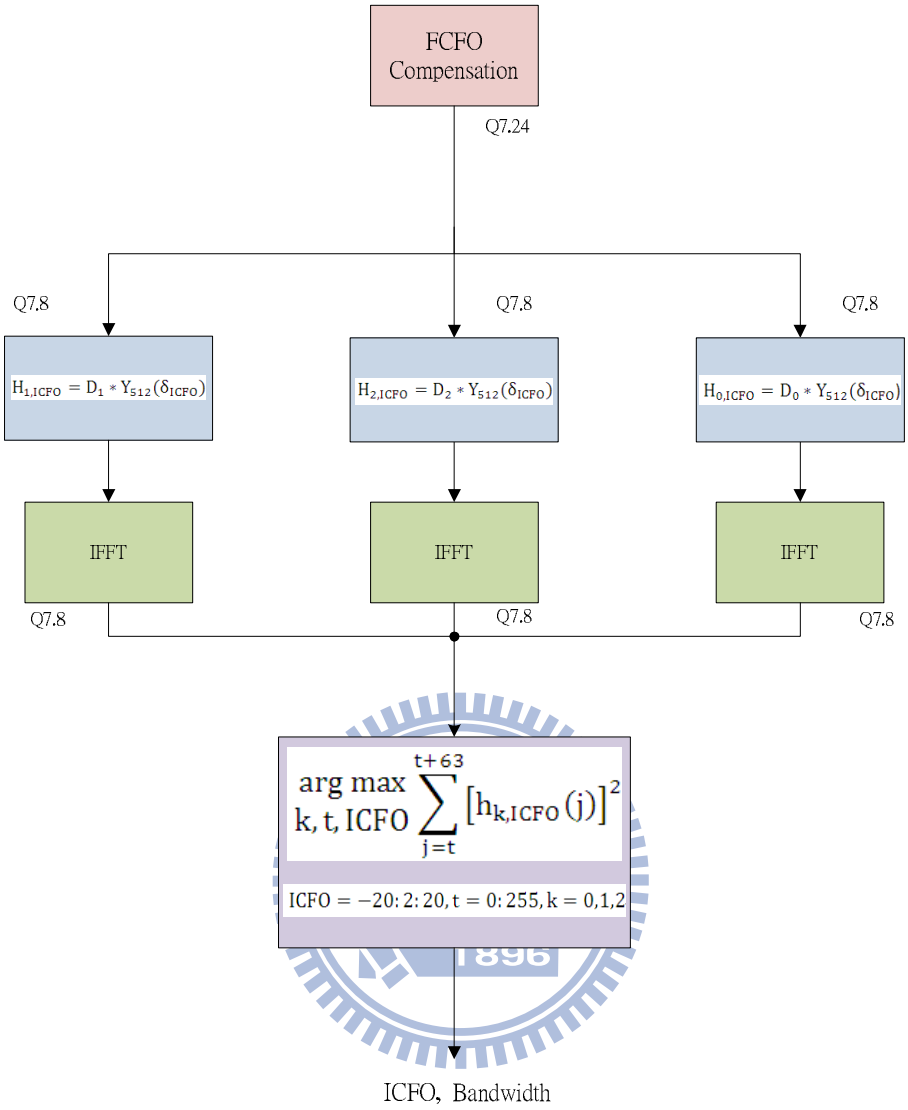

Figure 5.23: ICFO estimation and PID detection flow chart.

5.3.1 Coarse Timing Estimation

Fig. 5.24 shows the histograms of coarse timing samples under AWGN channel with 0 and 10 dB of SNR. Figs. 5.25 and 5.26 show the histograms under SUI-1 at SNR values of 0 and 10 dB and velocities of 10 and 90 km/h, respectively. In Figs. 5.27 and 5.28, we show the histograms under PB channel at SNR values of 0 and 10 dB and velocities of 10 and 90 km/h, respectively. Note that the simulation results are almost the same with the floating-point results shown in Figs. 5.2 to 5.6.

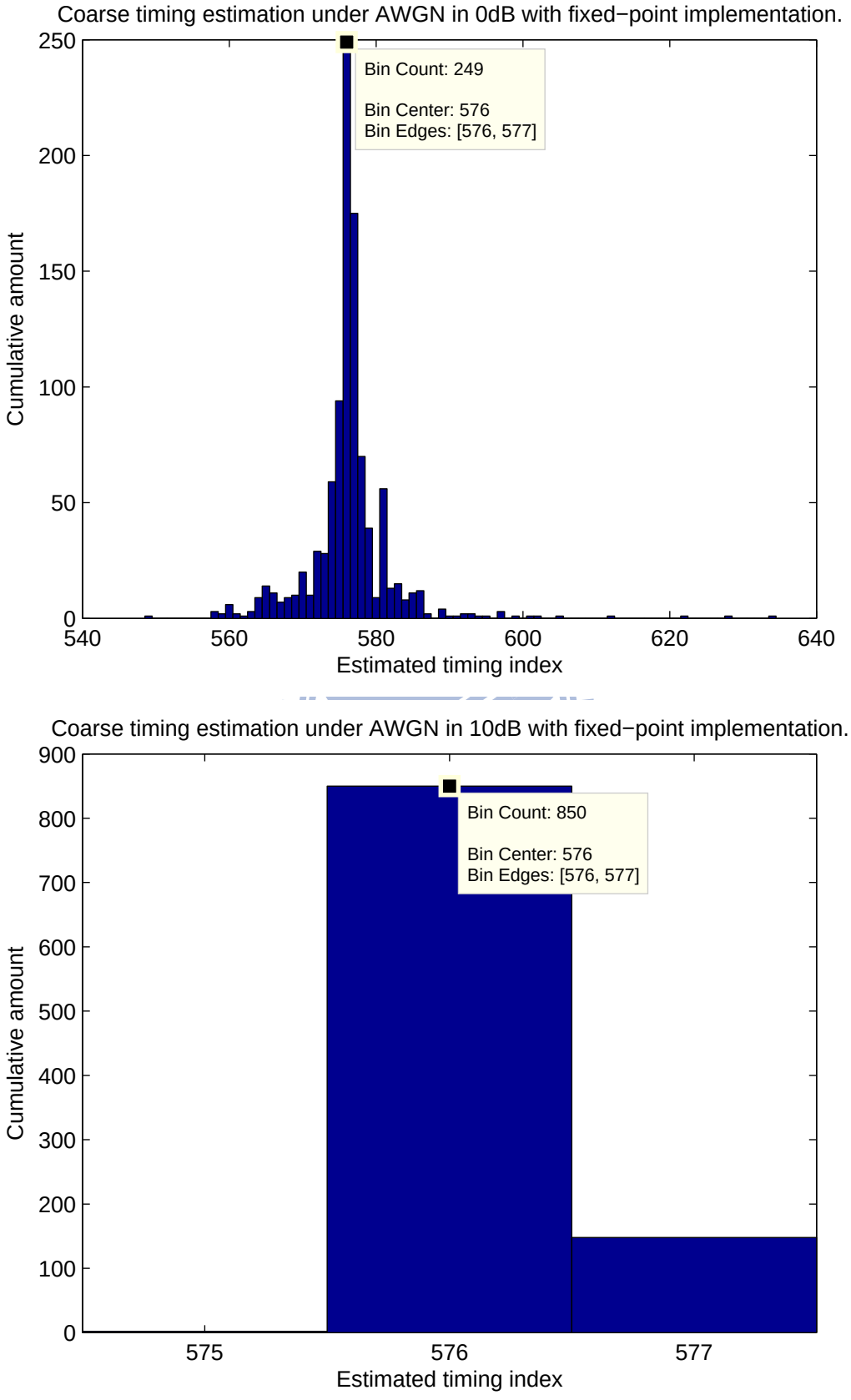

Figure 5.24: Histograms of coarse timing estimation under AWGN channel in different SNR values.

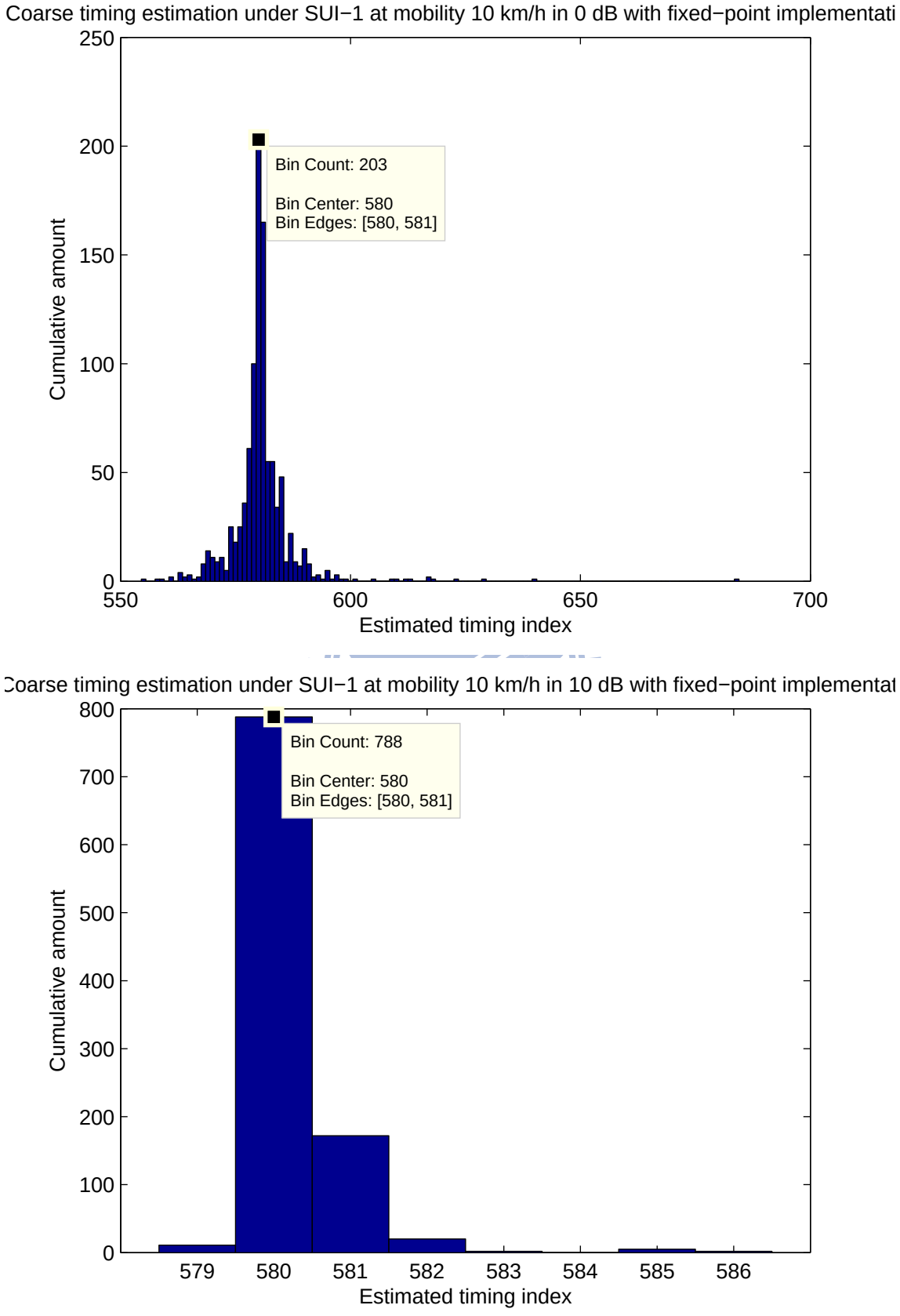

Figure 5.25: Histograms of coarse timing estimation under SUI-1 channel in different SNR values at a velocity of 10 km/h.

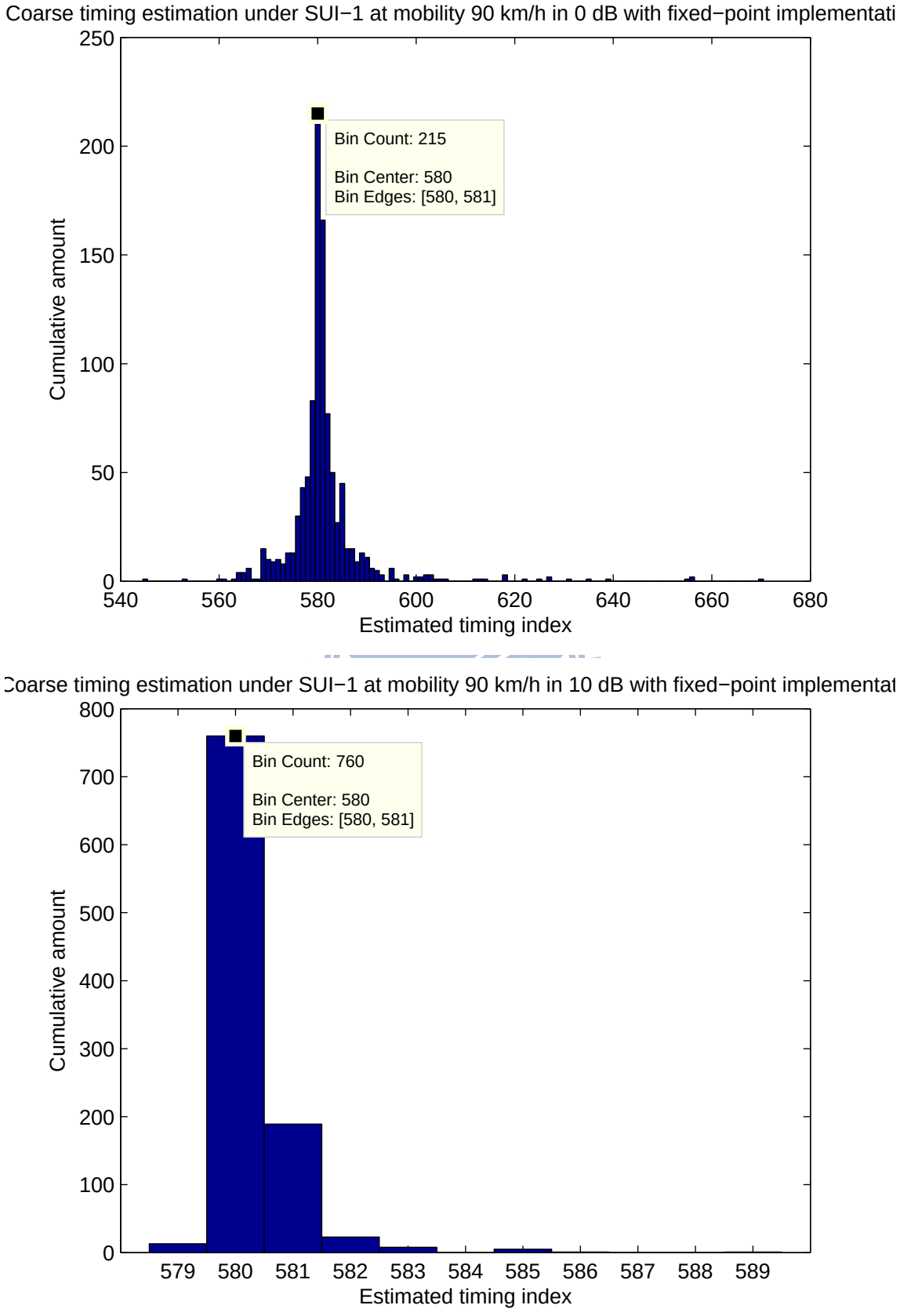

Figure 5.26: Histograms of coarse timing estimation under SUI-1 channel in different SNR values at a velocity of 90 km/h.

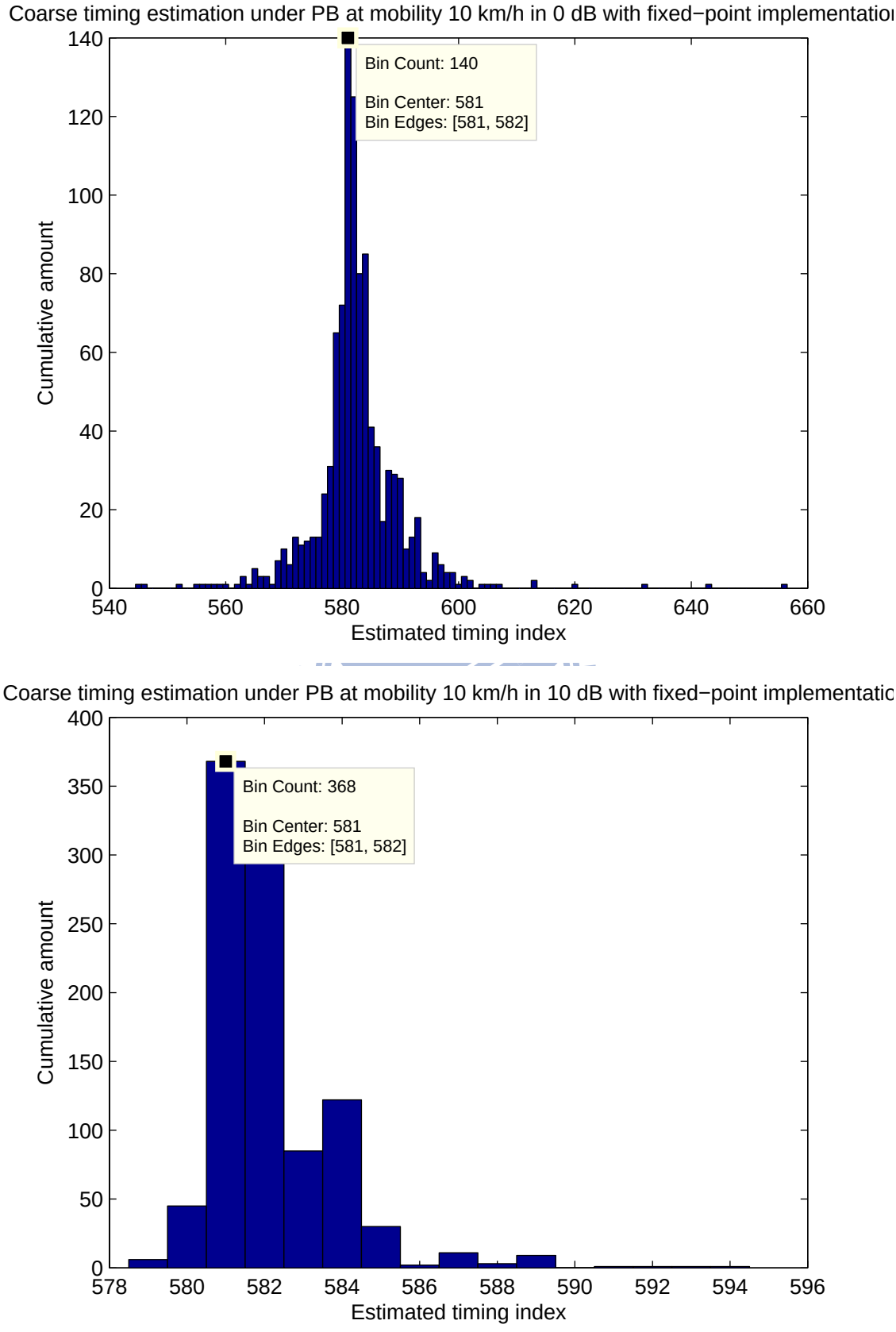

Figure 5.27: Histograms of coarse timing estimation under PB channel in different SNR values at a velocity of 10 km/h.

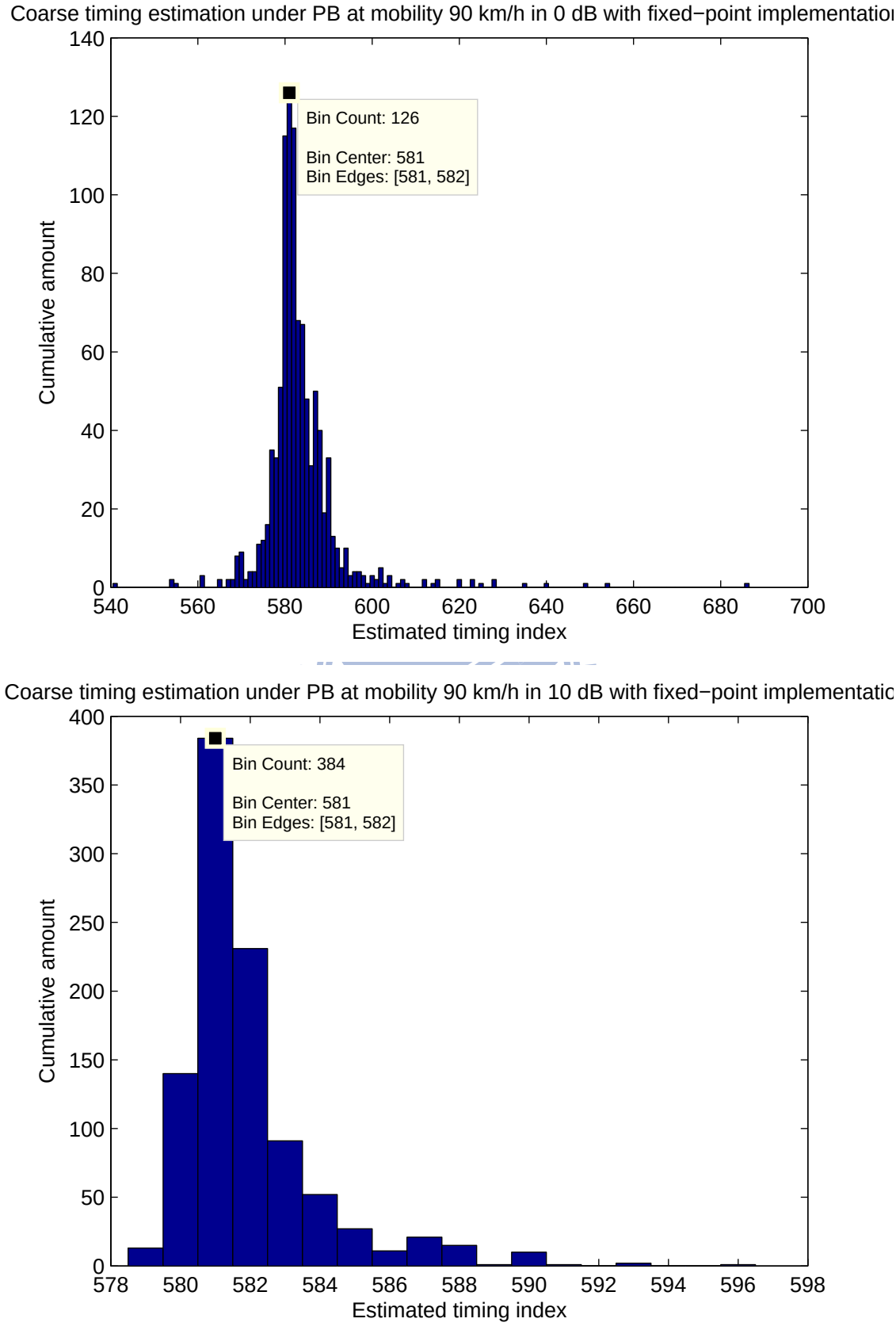

Figure 5.28: Histograms of coarse timing estimation under PB channel in different SNR values at a velocity of 90 km/h.

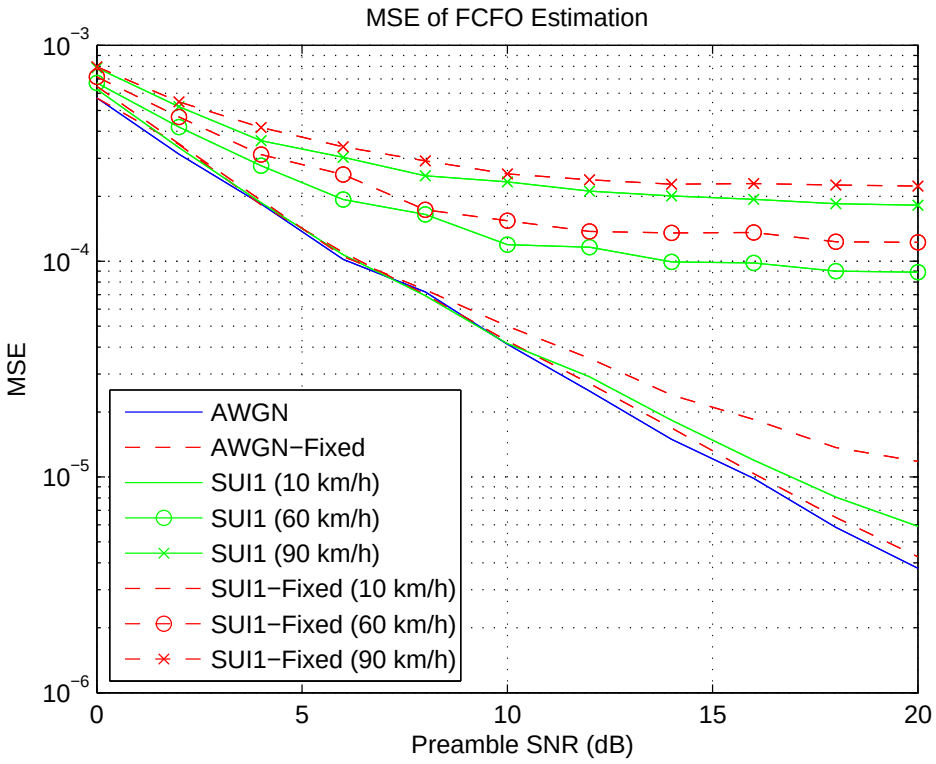

Figure 5.29: Mean square error of FCFO estimation under SUI-1 and AWGN channels with fixed-point and floating-point computation.

5.3.2 Fractional CFO Estimation

Figs. 5.29 to 5.31 show the MSE of fractional CFO estimation in SUI-1, SUI-3, PB and AWGN channels at speeds 10, 30, 60 and 90 km/h with fixed-point and floating-point computation. From the simulation results, we can see that the performance curves for floating-point and fixed-point computation are only a little different.

5.3.3 Jointly Estimation of Integral Carrier Frequency Offset, PID and Fine Timing

Figs. 5.32 to 5.34 show the estimation performance of integer CFO under AWGN and SUI-1 channels at speeds of 10 and 90 km/h at SNR of 0 and 10 dB, respectively, and Figs. 5.35 to 5.37 show the estimation performance of PID under AWGN and SUI-1 channels at speeds of

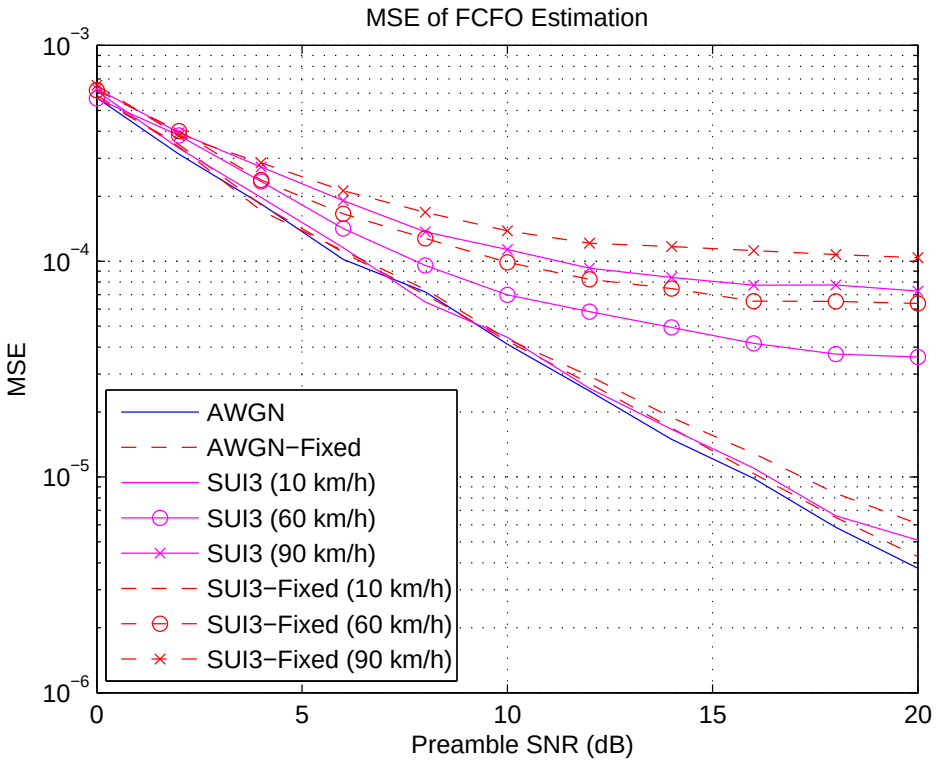

Figure 5.30: Mean square error of FCFO estimation under SUI-3 and AWGN channels with fixed-point and floating-point computation.

10 and 90 km/h at SNR of 0 and 10 dB. Table 5.3 shows the error rate of timing estimation. We can see that floating-point and fixed-point implementation have the same results shown in Figs. 5.10 to 5.15, in all cases simulated.

Table 5.3: The error rate of timing estimation.

	AWGN	PB_10km	PB_90km	SUI1_10km	SUI1_90km
0 dB	0.017	0.121	0.155	0.031	0.025
10 dB	0	0.002	0.001	0	0
20 dB	0	0	0	0	0

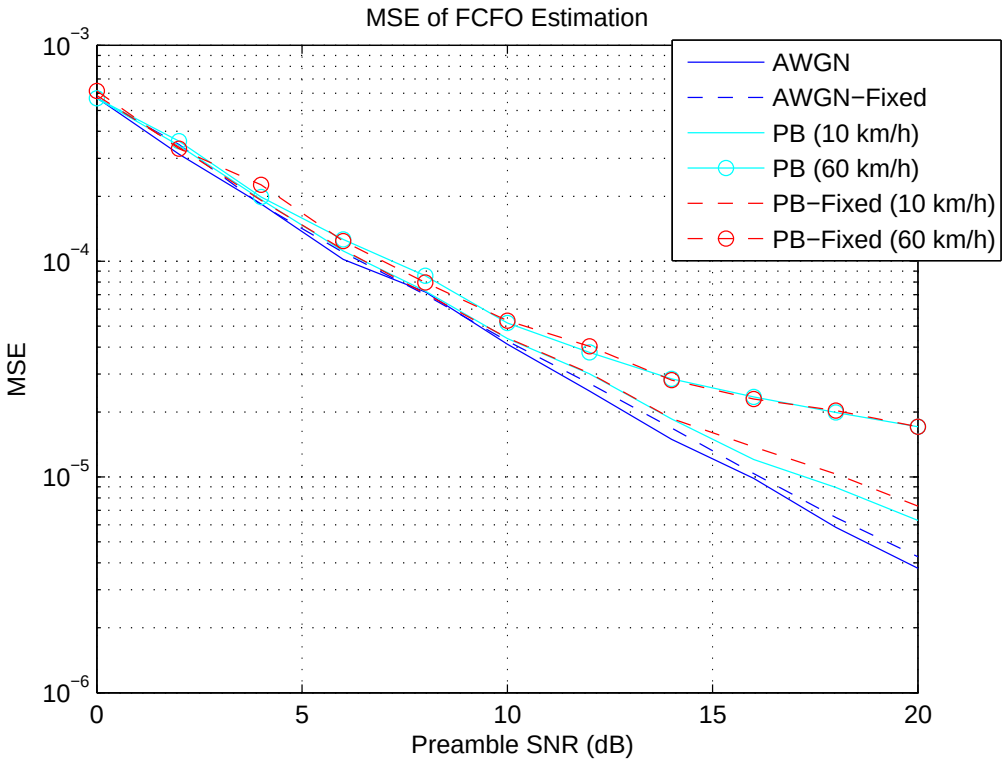

Figure 5.31: Mean square error of FCFO estimation under PB and AWGN channels with fixed-point and floating-point computation.

5.4 Speeding Up of DSP Implementation

In this section, we discuss how to reduce cycle counts in DSP implementation. The optimization techniques used include compiler option, intrinsic functions and DSP library function. We set the level of optimization of compiler option to -o3, which performs software pipelining and loop optimizations, and we do not perform loop unrolling ourselves. In the following, we concentrate the discussion on the use of intrinsic functions and DSP library functions in the function blocks.

5.4.1 Speeding Up of Coarse Timing Estimation

Calculating the magnitude-square of a complex number needs two multiplication, so accumulating 576 magnitude-squares for 1152 time position would require $576 \times 2 \times 1152 = 1,327,104$

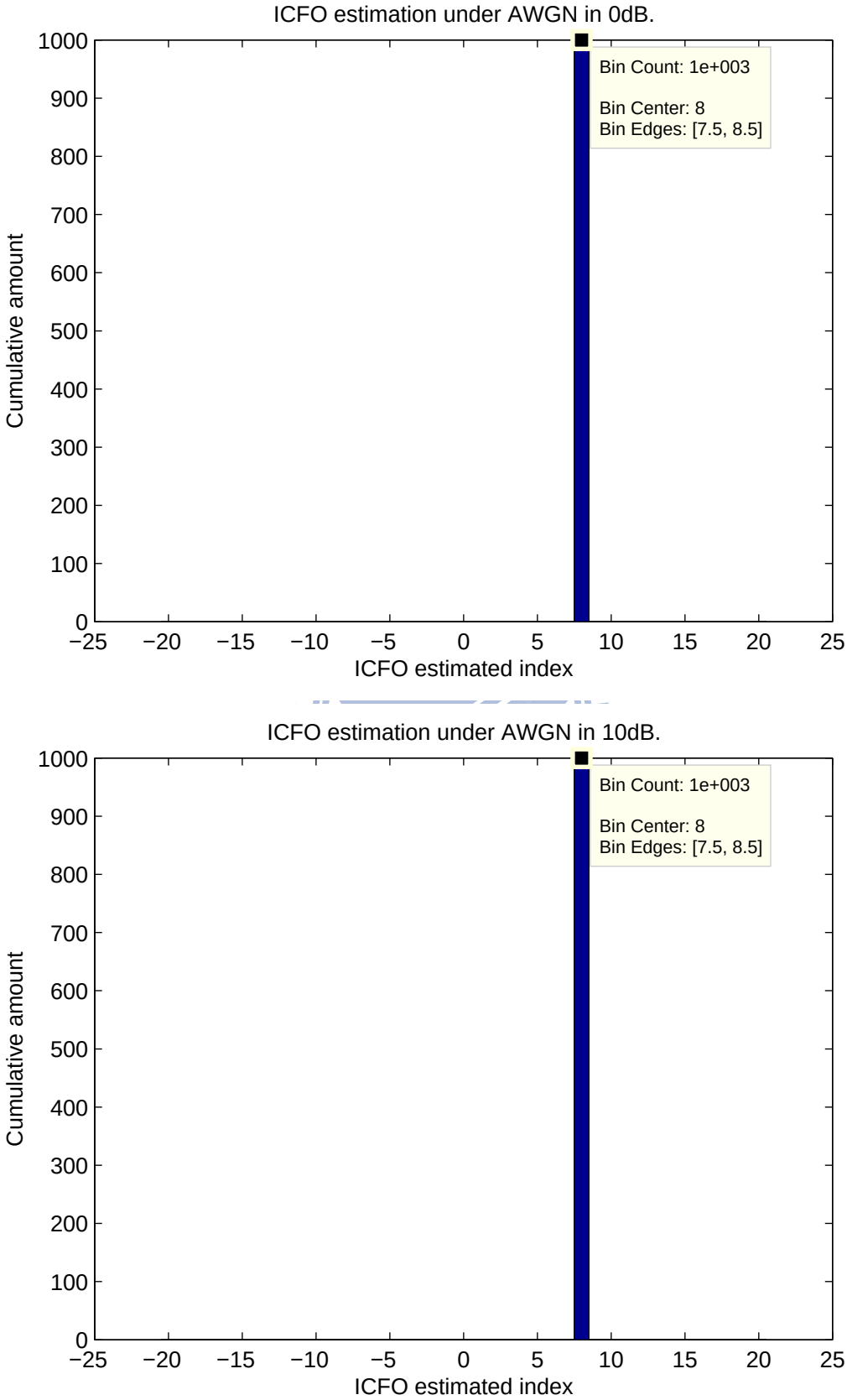

Figure 5.32: Histograms of integer CFO estimation under AWGN channel in different SNR values with fixed-point implementation.

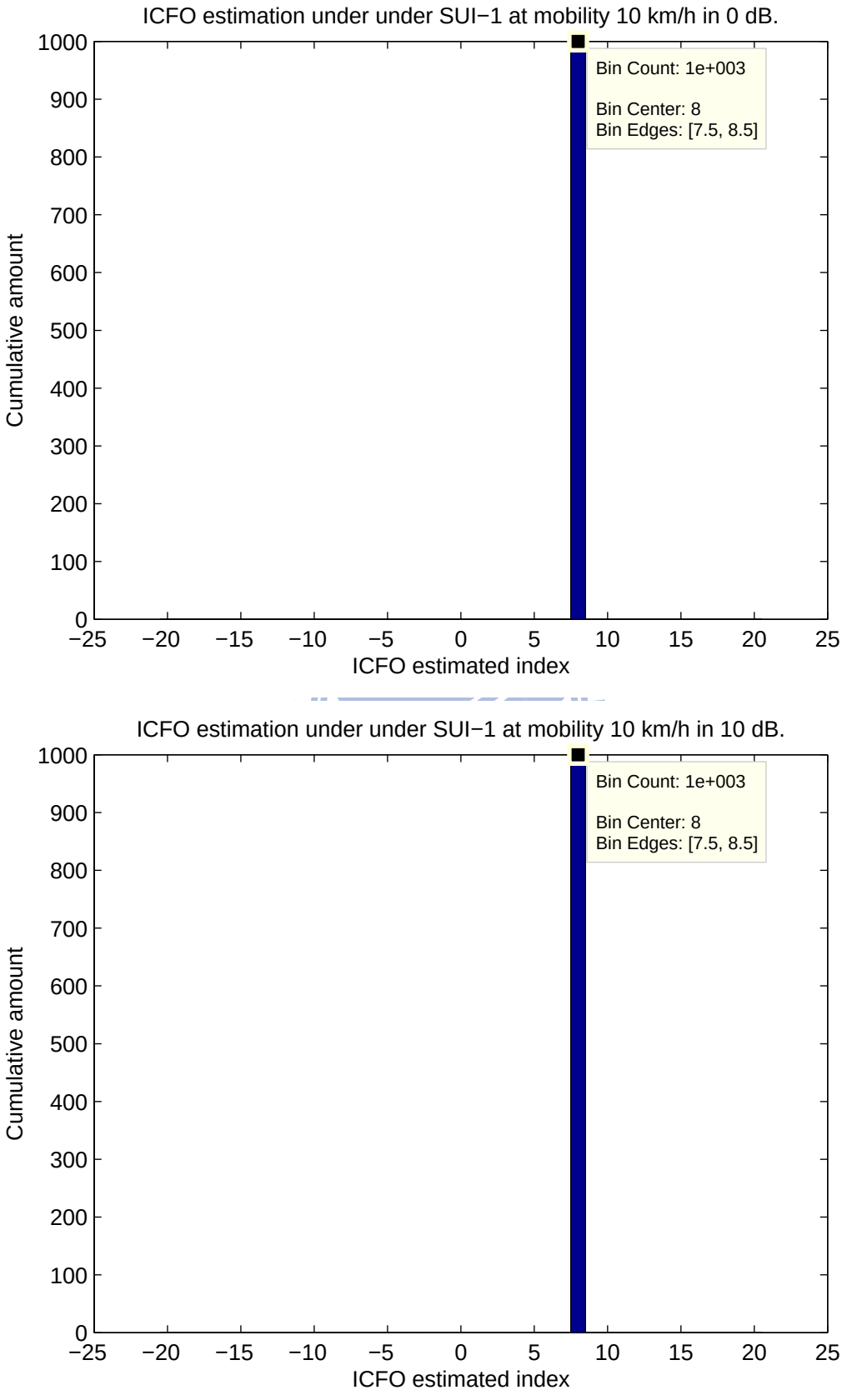

Figure 5.33: Histograms of integer CFO estimation under SUI-1 channel in different SNR values at a velocity of 10 km/h with fixed-point implementation.

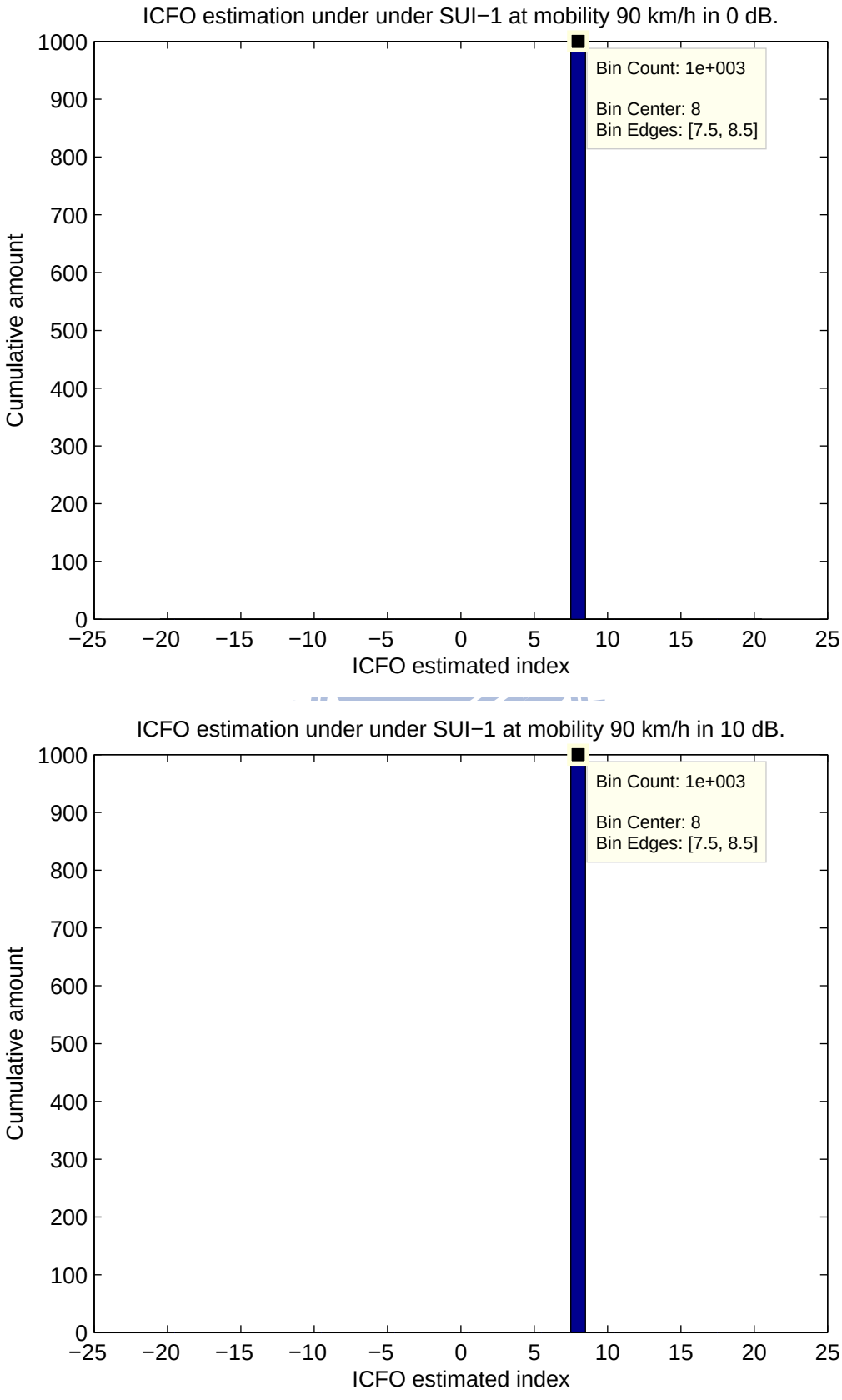

Figure 5.34: Histograms of integer CFO estimation under SUI-1 channel in different SNR values at a velocity of 90 km/h with fixed-point implementation.

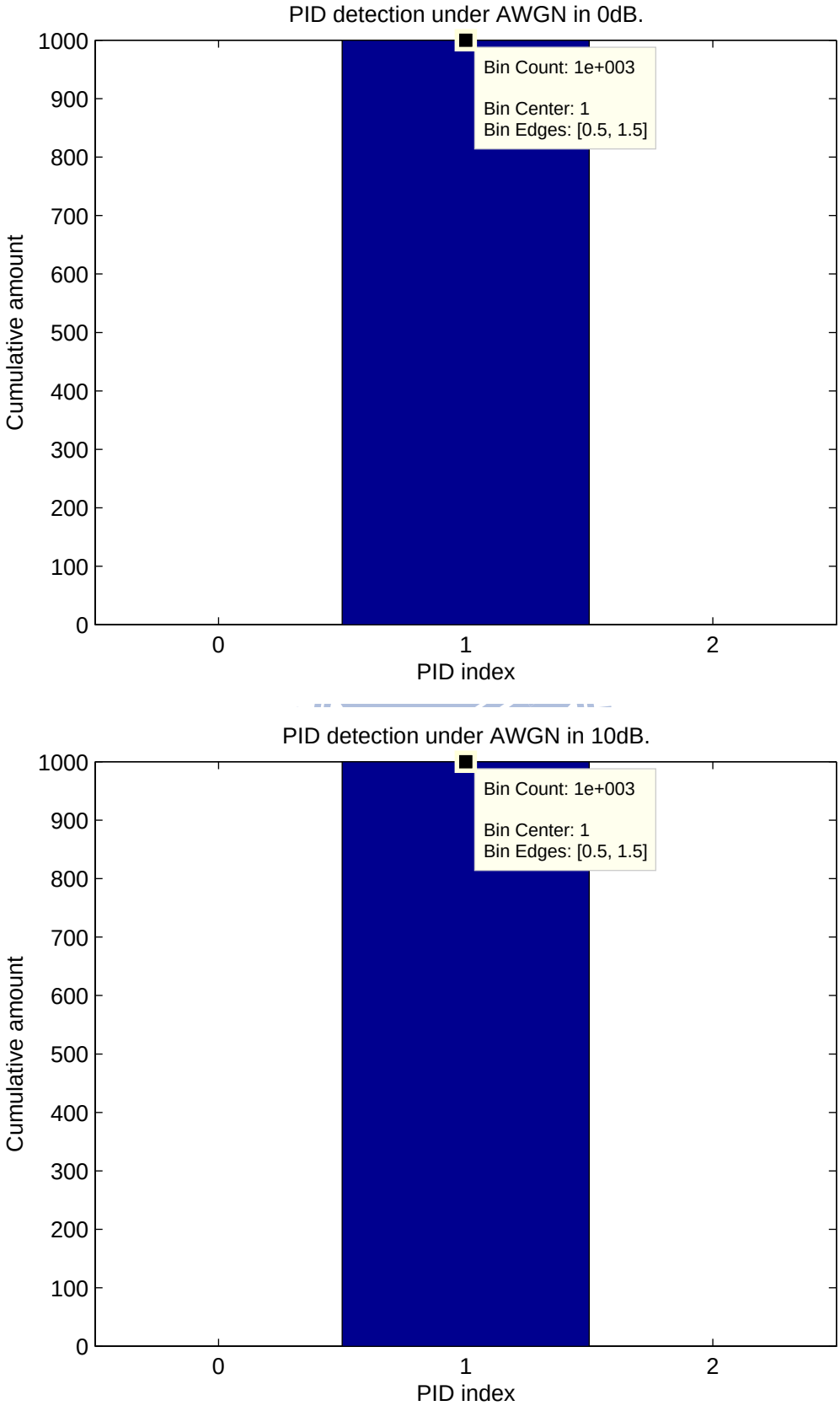

Figure 5.35: Histograms of PID detection estimation under AWGN channel in different SNR values with fixed-point implementation.

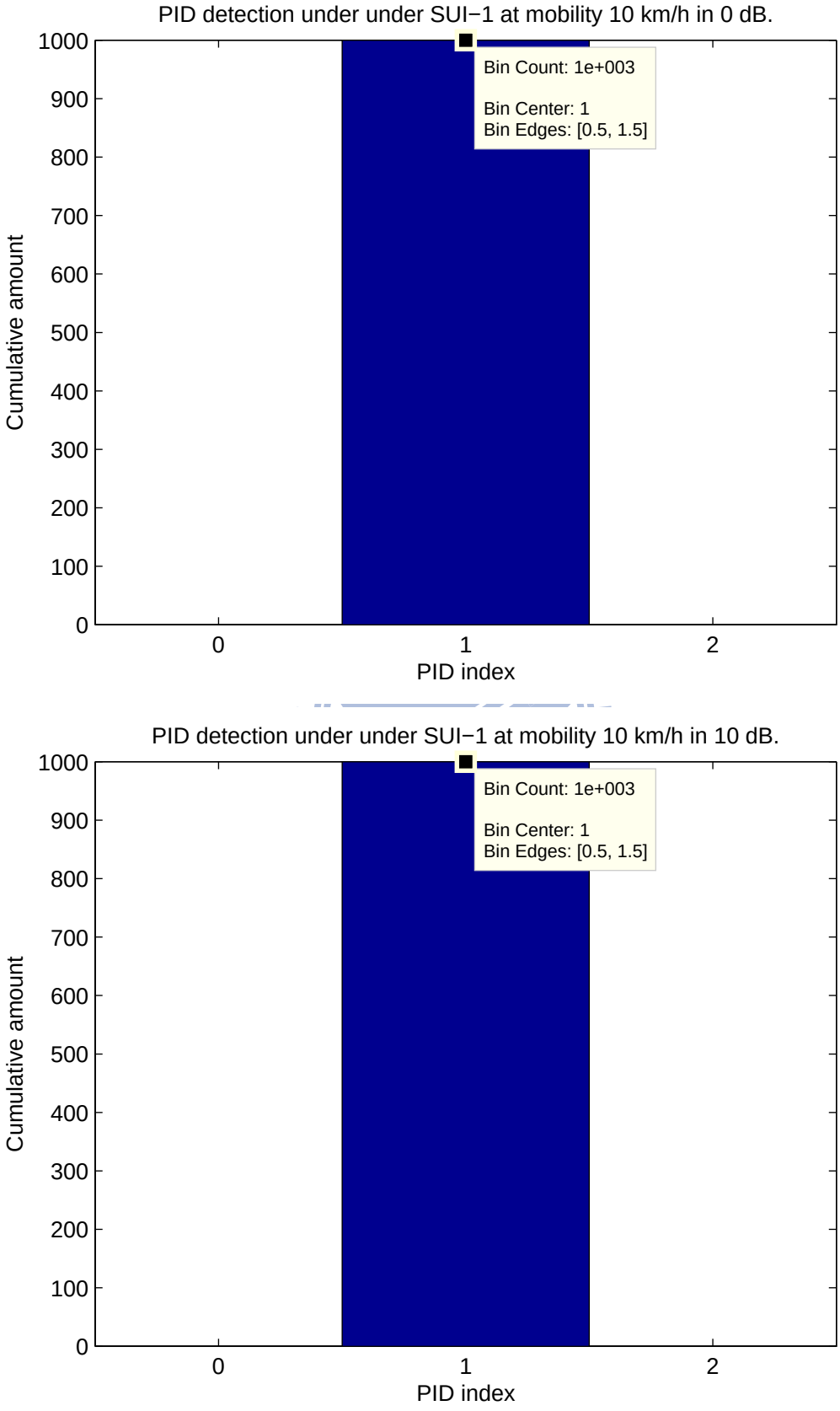

Figure 5.36: Histograms of PID detection under SUI-1 channel in different SNR values at a velocity of 10 km/h with fixed-point implementation.

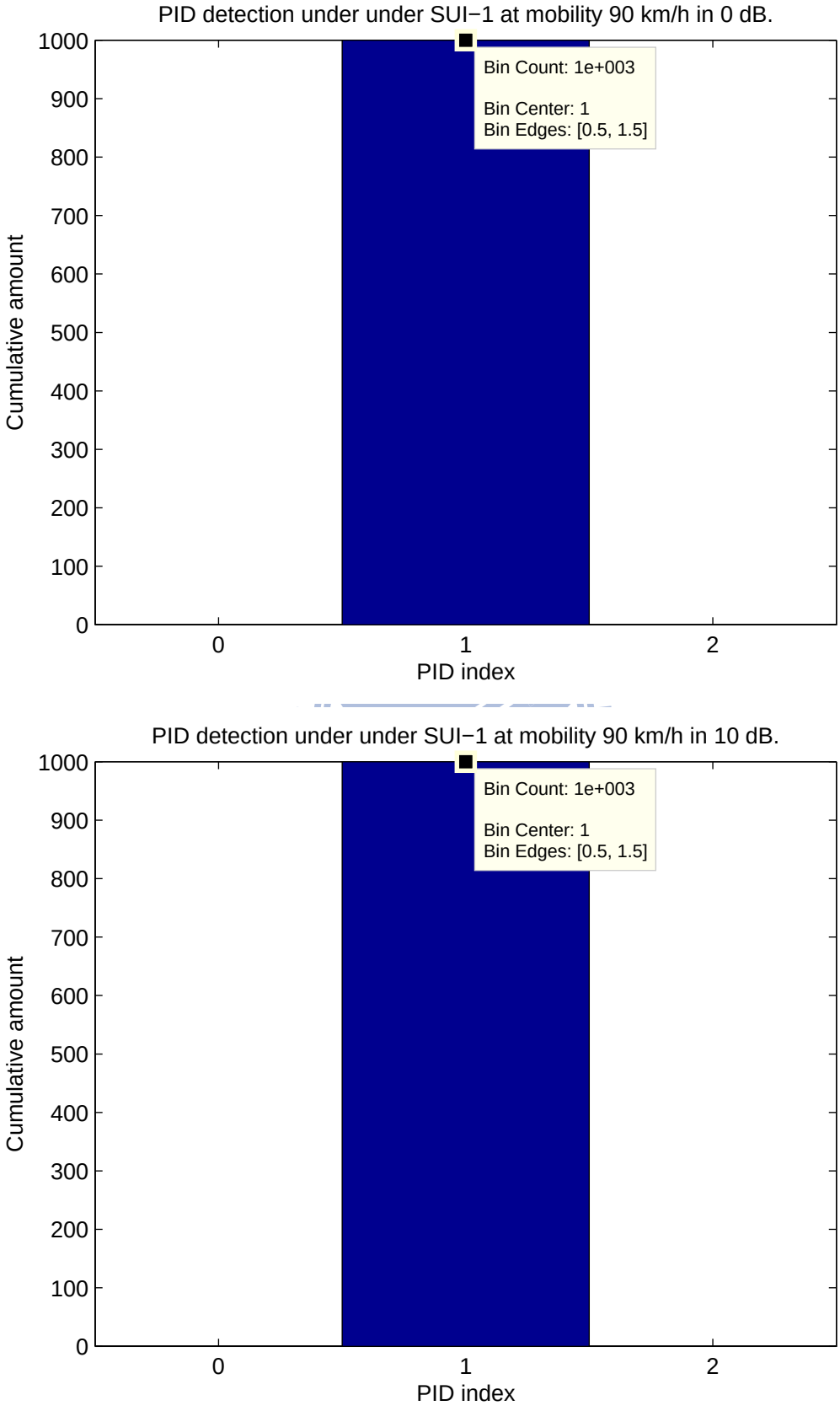

Figure 5.37: Histograms of PID detection under SUI-1 channel in different SNR values at a velocity of 90 km/h with fixed-point implementation.

Table 5.4: Coarse Timing Estimation Results for Optimization Level 3

Functions	Avg. Clock Cycles
Main Loop	7722
Initial Loop	139
Maximum PowerLevel	4756

Table 5.5: Coarse Timing Estimation Results for Optimization Level 1

Functions	Avg. Clock Cycles
Main Loop	41448
Initial Loop	16140
Maximum PowerLevel	37026

multiplications. However, most of accumulated quantities appear repeatedly across successive time positions as shown in Fig. 5.38. Using this fact, we may compute the sum of magnitude-squares as

$$Power(N+1) = Power(N) - R(N) + R(N+576), \tag{5.1}$$

$$Power(0) = \sum_{n=0}^{575} R(n), \tag{5.2}$$

where $R(N)$ is the magnitude-squares of the received signal sample at time N and $Power(N)$ is the sum as indicated in Fig. 5.38.

The compiler automatically utilizes the assembly instruction **MPY2** that computes two 16×16 multiplication in parallel. We show the improved C code and the corresponding assembly code in Figs. 5.39 to 5.41. Hence, the coarse timing estimation needs about $576 \times 2 + 1152 \times 4 = 5760$ multiplications. According to Table 5.4, the efficiency of the coarse timing estimation is $(5760 \div 4 \div 7861) \times 100 = 18.3\%$, and Table 5.5 shows the cycle counts for compiler option of optimization level 1, the efficiency is $(5760 \div 2 \div 57588) \times 100 = 5\%$. The exection time of optimization level 1 is worse than optimization level 3 since the optimization level 1 does not performs loop unloring, software-pipleining and call assembly instruction **MPY2**.

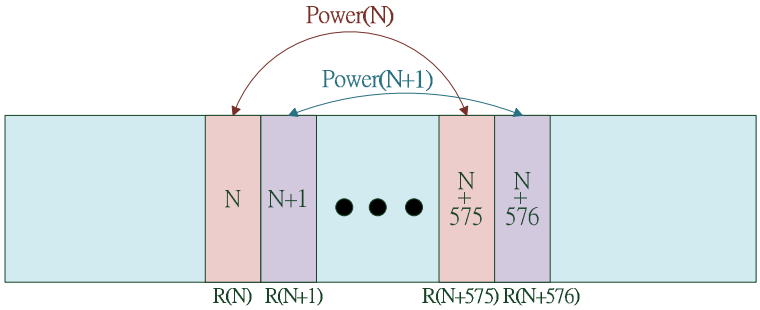

Figure 5.38: Summation of magnitude-squares for coarse timing estimation.

5.4.2 Using DSP Library Function for FFT and IFFT [18]

TI supplies a DSP library that contains the FFT/IFFT function `DSP_fft16x16r()` that implements a cache-optimized complex forward mixed radix FFT with scaling, rounding and digit reversal. The input data x[], output data y[] and coefficients w[] are 16-bit numbers. The output is returned in the array y[] in normal order. Each complex value is stored as interleaved 16-bit real and imaginary parts. The code uses a special ordering of FFT coefficients (also called twiddle factors). This DSP library function takes $\lceil \log_4(nx) - 1 \rceil \times (\frac{5}{4} \times nx + 25) + \frac{5}{4} \times nx + 26$ cycles and the codesize is 868 bytes, where nx is FFT size.

5.4.3 Speeding Up of ICFO, PID, Fine Timing Estimation

In integer CFO estimation, we utilize the signal structure in the frequency domain, hence we need not compute the CIR the corresponding with PA-Preamble subcarrier is 0. Furthermore, we use the same method with coarse timing estimation to calculate the sum of CIR. Therefore, it is needs $216 \times 2 \times 3 \times 21 = 27,216$ multiplications to compute the CIR and $(64 \times 2 + 256 \times 4) \times 3 \times 21 = 72,576$ multiplications to compute the sum of CIR.

According to Table 5.6, the efficiency of sum of CIR is $(72576 \div 4 \div 75411) \times 100 = 24.1\%$ and the efficiency of CIR computation is $(27216 \div 2 \div 33705) \times 100 = 40.4\%$. Table 5.7 shows the cycle counts for compiler option of optimization level 1, the efficiency of sum of CIR is $(72576 \div 2 \div 406602) \times 100 = 8.9\%$, and the efficiency of CIR computation is

Table 5.6: ICFO, PID, Fine Timing Estimation Results for Optimization Level 3

Functions	Avg. Clock Cycles
Sum of CIR	75411
CIR Computation	33705
MaxFixed	33453
Others	6042

Table 5.7: ICFO, PID, Fine Timing Estimation Results for Optimization Level 1

Functions	Avg. Clock Cycles
Sum of CIR	406602
CIR Computation	326970
MaxFixed	547848
Others	340204

$(27216 \div 2 \div 326970) \times 100 = 4.16\%$.

5.5 DSP Optimization Results

Table 5.8 shows the number of clock for each function used in the initial DL synchronization procedures, and number of clock cycles does not including TI library in this table. Obviously, IFFT function takes the most percentage of total cycles, because number of IFFT times is the 21 (ICFO candidates)×3 (bandwidth) = 63 times, and IFFT function time positions would require about 14,000 cycles per time. The `DSP_fft16x16r()` function is already highly optimized. Table 5.9 shows the number of clock cycles including and excluding memory access. Table 5.10 shows the code size of the program for different optimization levels. In our system, the clock frequency of TMS320C6416T DSP is 1 GHz, so the total execution time of initial DL synchronization procedures is 1.181 ms.

Table 5.8: DSP Optimization Results

Functions	Avg. Clock Cycles	Percentage of Total Cycles (%)
Coarse Timing Estimation	10239	0.96
FCFO Estimation	5911	0.554
Compensation	2141	0.2
FFT	14046	1.318
IFFT	884394	82.963
ICFO Estimation	148611	13.941
Remove CP	668	0.063
Total cycles	1066010	100

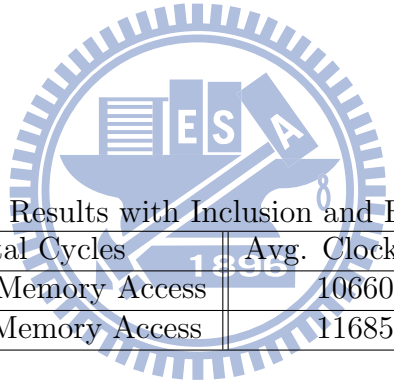

Table 5.9: DSP Optimization Results with Inclusion and Exclusion of Memory Access

Total Cycles	Avg. Clock Cycles
Exclude Memory Access	1066010
Include Memory Access	1168531

Table 5.10: Code Size Results

Program	Memory
Optimization Level 3 Program Code Size (-o3)	448.613 KB
Optimization Level 1 Program Code Size (-o1)	439.781 KB
Data Memory Size	327.68KB

```
for(k=1;k<(Ntotal-WindowSize);k++)                    //Coarse Timing Estimation
000B7848 00802041            MVK.D1        1,A1
000B784C 00008F29  ||        MVK.S1        0x011e,A0
000B7850 0238542A  ||        MVK.S2        0x70a8,B4
000B7854 033C9C43            ADDAW.D2      SP,B4,B6
000B7858 0F3BB42A  ||        MVK.S2        0x7768,B30
000B785C 0A0403E3            MVC.S2        CSR,B20
000B7860 023FDC42  ||        ADDAW.D2      SP,B30,B4
000B7864 039818F1            OR.D1X        0,B6,A7
000B7868 02BA042B  ||        MVK.S2        0x7408,B5
000B786C 0ED3C9F2  ||        AND.D2        -2,B20,B29
000B7870 03109AF1            ADD.D1X       4,B4,A6
000B7874 02BCBC43  ||        ADDAW.D2      SP,B5,B5
000B7878 00F403A2  ||        MVC.S2        B29,CSR
000B787C 921003E4     [!A1]  LDDW.D2T1     *+B4[0x0],A5:A4
000B7880 019818F1            OR.D1X        0,B6,A3
000B7884 0B4A4030  ||        MPY2.M1       A18,A18,A23:A22
000B7888 0A0C6030            MPY2.M1       A3,A3,A21:A20
000B788C 01A018F1            OR.D1X        0,B8,A3
000B7890 02420030  ||        MPY2.M1       A16,A16,A5:A4
000B7894 0C0C6030            MPY2.M1       A3,A3,A25:A24
000B7898 0192C8C0            SUB.D1        A4,A22,A3
000B789C 018E88C0            SUB.D1        A3,A20,A3
000B78A0 01906840            ADD.D1        A4,A3,A3
000B78A4 01E06840            ADD.D1        A24,A3,A3
000B78A8 91989675     [!A1]  STW.D1T1      A3,*A6++[0x4]
000B78AC 01DC65E0  ||        SUB.S1        A3,A23,A3
000B78B0 018EA8C0            SUB.D1        A3,A21,A3
000B78B4 01946840            ADD.D1        A5,A3,A3
000B78B8 03110943            ADD.D2        B4,0x8,B6
000B78BC 01E46840  ||        ADD.D1        A25,A3,A3
000B78C0 919802F4     [!A1]  STW.D2T1      A3,*+B6[0x0]
000B78C4 034C16A3            OR.S2X        0,A19,B6
000B78C8 931803E6  || [!A1]  LDDW.D2T2     *+B6[0x0],B7:B6
000B78CC 0918C032            MPY2.M2       B6,B6,B19:B18
000B78D0 034418F3            OR.D2X        0,A17,B6
000B78D4 081CE032  ||        MPY2.M2       B7,B7,B17:B16
000B78D8 0418C032            MPY2.M2       B6,B6,B9:B8
000B78DC 03252032            MPY2.M2       B9,B9,B7:B6
000B78E0 031A48C2            SUB.D2        B6,B18,B6
000B78E4 031A08C2            SUB.D2        B6,B16,B6
000B78E8 0420C842            ADD.D2        B8,B6,B8
000B78EC C07CF021     [ A0]  BDEC.S1       L172,A0
000B78F0 03190842  ||        ADD.D2        B6,B8,B6
000B78F4 93184077     [!A1]  STW.D1T2      B6,*-A6[0x2]
000B78F8 031A68C2  ||        SUB.D2        B6,B19,B6
```

Figure 5.39: Assembly code of the coarse timing estimation (1/3).

```
000B7900 031437E7  ||         LDDW.D2T2     *B5++[0x1],B7:B6
000B7904 091C3764  ||         LDDW.D1T1     *A7++[0x1],A19:A18
000B7908 0324C843             ADD.D2        B9,B6,B6
000B790C 08203764  ||         LDDW.D1T1     *A8++[0x1],A17:A16
000B7910 031CC843             ADD.D2        B7,B6,B6
000B7914 04243766  ||         LDDW.D1T2     *A9++[0x1],B9:B8
000B7918 808429C1     [ A1]   SUB.D1        A1,0x1,A1
000B791C 931092F6  || [!A1]   STW.D2T2      B6,*++B4[0x4]
000B7920 021003E4             LDDW.D2T1     *+B4[0x0],A5:A4
000B7924 019818F1             OR.D1X        0,B6,A3
000B7928 0B4A4030  ||         MPY2.M1       A18,A18,A23:A22
000B792C 0A0C6030             MPY2.M1       A3,A3,A21:A20
000B7930 01A018F1             OR.D1X        0,B8,A3
000B7934 02420030  ||         MPY2.M1       A16,A16,A5:A4
000B7938 0C0C6030             MPY2.M1       A3,A3,A25:A24
000B793C 0192C8C0             SUB.D1        A4,A22,A3
000B7940 018E88C0             SUB.D1        A3,A20,A3
000B7944 01906840             ADD.D1        A4,A3,A3
000B7948 01E06840             ADD.D1        A24,A3,A3
000B794C 01DC65E1             SUB.S1        A3,A23,A3
000B7950 01989674  ||         STW.D1T1      A3,*A6++[0x4]
000B7954 018EA8C0             SUB.D1        A3,A21,A3
000B7958 01946840             ADD.D1        A5,A3,A3
000B795C 03110943             ADD.D2        B4,0x8,B6
000B7960 01E46840  ||         ADD.D1        A25,A3,A3
000B7964 019802F4             STW.D2T1      A3,*+B6[0x0]
000B7968 034C16A3             OR.S2X        0,A19,B6
000B796C 031803E6  ||         LDDW.D2T2     *+B6[0x0],B7:B6
000B7970 0918C032             MPY2.M2       B6,B6,B19:B18
000B7974 034418F3             OR.D2X        0,A17,B6
000B7978 081CE032  ||         MPY2.M2       B7,B7,B17:B16
000B797C 0418C032             MPY2.M2       B6,B6,B9:B8
000B7980 03252032             MPY2.M2       B9,B9,B7:B6
000B7984 031A48C2             SUB.D2        B6,B18,B6
000B7988 031A08C2             SUB.D2        B6,B16,B6
000B798C 00D003A3             MVC.S2        B20,CSR
000B7990 0420C842  ||         ADD.D2        B8,B6,B8
000B7994 03190842             ADD.D2        B6,B8,B6
000B7998 031A68C3             SUB.D2        B6,B19,B6
000B799C 03184076  ||         STW.D1T2      B6,*-A6[0x2]
000B79A0 031A28C2             SUB.D2        B6,B17,B6
000B79A4 0324C842             ADD.D2        B9,B6,B6
000B79A8 031CC842             ADD.D2        B7,B6,B6
000B79AC 031092F6             STW.D2T2      B6,*++B4[0x4]
```

Figure 5.40: Assembly code of the coarse timing estimation (2/3).

```
PowerLevel_fixed[k]=PowerLevel_fixed[k-1]-((RecevieRe_fixed[k-1]*RecevieRe_fixed[k-1])+(RecevieIm_fixed[k-1]*RecevieIm_fixed[k-1]))
000B79B0 0239732A          MVK.S2      0x72e6,B4
000B79B4 03BC9C43          ADDAW.D2    SP,B4,B7
000B79B8 02BBB32A  ||      MVK.S2      0x7766,B5
000B79BC 0FBB232B          MVK.S2      0x7646,B31
000B79C0 04BCBC42  ||      ADDAW.D2    SP,B5,B9
000B79C4 0F3A032B          MVK.S2      0x7406,B30
000B79C8 029C02E6  ||      LDW.D2T2    *+B7[0x0],B5
000B79CC 0EBDF22B          MVK.S2      0x7be4,B29
000B79D0 033FFC42  ||      ADDAW.D2    SP,B31,B6
000B79D4 029802E4          LDW.D2T1    *+B6[0x0],A5
000B79D8 043FDC42          ADDAW.D2    SP,B30,B8
000B79DC 022002E4          LDW.D2T1    *+B8[0x0],A4
000B79E0 0914A033          MPY2.M2     B5,B5,B19:B18
000B79E4 023FBC42  ||      ADDAW.D2    SP,B29,B4
000B79E8 081003E6          LDDW.D2T2   *+B4[0x0],B17:B16
000B79EC 0414A031          MPY2.M1     A5,A5,A9:A8
000B79F0 01A402E4  ||      LDW.D2T1    *+B9[0x0],A3
000B79F4 0F9C42C4          LDH.D2T1    *+B7[0x2],A31
000B79F8 0F1842C5          LDH.D2T1    *+B6[0x2],A30
000B79FC 03108030  ||      MPY2.M1     A4,A4,A7:A6
000B7A00 0EA042C4          LDH.D2T1    *+B8[0x2],A29
000B7A04 0E2442C5          LDH.D2T1    *+B9[0x2],A28
000B7A08 02CA05E2  ||      SUB.S2      B16,B18,B5
000B7A0C 020C6030          MPY2.M1     A3,A3,A5:A4
000B7A10 01951D70          SUB.S1X     B5,A8,A3
000B7A14 01986840          ADD.D1      A6,A3,A3
000B7A18 00000000          NOP
000B7A1C 0DF38C81          MPY.M1      A28,A28,A27
000B7A20 04106840  ||      ADD.D1      A4,A3,A8
000B7A24 041022F5          STW.D2T1    A8,*+B4[0x1]
000B7A28 01CD15E0  ||      SUB.S1X     A8,B19,A3
000B7A2C 01FFEC81          MPY.M1      A31,A31,A3
000B7A30 020D28C0  ||      SUB.D1      A3,A9,A4
000B7A34 027BCC81          MPY.M1      A30,A30,A4
000B7A38 031C8840  ||      ADD.D1      A7,A4,A6
000B7A3C 0377AC81          MPY.M1      A29,A29,A6
000B7A40 0294C840  ||      ADD.D1      A5,A6,A5
000B7A44 029042F4          STW.D2T1    A5,*+B4[0x2]
000B7A48 031023E6          LDDW.D2T2   *+B4[0x1],B7:B6
000B7A4C 00006000          NOP         4
000B7A50 01987D70          SUB.S1X     B6,A3,A3
000B7A54 018C88C0          SUB.D1      A3,A4,A3
000B7A58 01986840          ADD.D1      A6,A3,A3
000B7A5C 01EC6840          ADD.D1      A27,A3,A3
000B7A60 019062F4          STW.D2T1    A3,*+B4[0x3]
+((RecevieRe_fixed[k+WindowSize-1]*RecevieRe_fixed[k+WindowSize-1])+(RecevieIm_fixed[k+WindowSize-1]*RecevieIm_fixed[k+WindowSize-1]))
```

Figure 5.41: Assembly code of the coarse timing estimation (3/3).

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this thesis, we first presented the overall procedure of initial DL synchronization of the IEEE 802.16m TDD system, and verified them through floating-point computation. Second, we implemented the initial DL synchronization to fixed-point computation and compared the performance with floating-point computation. Finally, we optimized procedure of initial DL synchronization on TI's C6416T digital signal processor.

In the procedure of initial DL synchronization, we used coarse timing estimation to estimate the PA-Preamble location, thus, we obtained the FCFO from the quasi-ML estimation. In the end, we utilized the characteristic of the power centralization of CIR to estimate the ICFO, PID and fine timing offset.

For DSP implementation, we chosen Q7.8 to be our data format, and verified the performance through simulation results is close to the floating-point computation. We used optimization techniques that including preamble character, intrinsic functions and DSP library function to reduce the computation time. According to Table 5.8, if the clock frequency of TMS320C6416T DSP is 1 GHz, execution time of initial DL synchronization procedures is 1.181 ms.

6.2 Future Work

There are several possible extension for our research:

- Reduce codesize of procedure of initial DL synchronization since we do not discuss in this thesis.
- Implement code of fixed-point vision on SMT395 board since we only use CCS simulator in this thesis.
- Use intrinsic function to reduce more cycle count.
- Integration the system of overall procedure of DL communication such as channel coding, synchronization and channel estimation on DSP implementation.

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., February 2010.

[2] P.-S. Wang, K.-W. Lu, D. W. Lin, and P. Ting, "Quasi-maximum likelihood initial downlink synchronization for IEEE 802.16m," in *Proc. IEEE Int. Workshop Signal Processing Advances Wirel. Commun.*, June 2011, pp. 506–510.

[3] 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.

[4] 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.*, 2006, vol. 7, pp. 3035–3039.

[5] IEEE 802.16 Task Group m Draft 9, *Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems — Advanced Air Interface (working document).* IEEE 802.16m, Oct. 6, 2010.

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

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

[8] P.H. Moose, "A technique for orthogonal frequency division multiplexing frequency offset correction," *IEEE Trans. Commun.,* vol. 42, no. 10, pp. 2908–2914, Oct. 1994.

[9] Y. Chunxuan, A. Reznik, G. Sternberg, Y. Shah, "On the secrecy capabilities of ITU Channels," in *IEEE Vehicular Technology Conference,* Oct. 2007, pp. 2030–2034.

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

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

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

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

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

[15] Texas Instruments, *TMS320C6000 Programmer's Guide.* Literature no. SPRU198J, Apr. 2010.

[16] Texas Instrument, *TMS320C6000 Optimizing Compiler User Guide.* Literature no. SPRU187S, Mar. 2011.

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

[18] Texas Instrument, *TMS320C64x DSP Library Programmer's Reference.* Literature no. SPRU565B, Oct. 2003.

[19] J. J. van de Beek *et al.*, "ML estimation of time and frequency offset in OFDM systems," *IEEE Trans. Signal Processing,* vol. 45, no. 7, pp. 1800–1805, July 1997.