國 立 交 通 大 學

電機學院通訊與網路科技產業研發碩士班

**碩 士 論 文**

# IEEE 802.16e OFDMA TDD 測距程序與整合建構於即時作業系統DSP平台之上行傳收系統

# IEEE 802.16e OFDMA TDD Ranging Process and Uplink Transceiver Integration on DSP Platform with Real-Time Operating System

研 究 生：張順成

指導教授：林大衛　教授

中 華 民 國 九 十 七 年 一 月

# IEEE 802.16e OFDMA TDD 測距程序與整合建構於即時作業系統DSP平台之上行傳收系統

# IEEE 802.16e OFDMA TDD Ranging Process and Uplink Transceiver Integration on DSP Platform with Real-Time Operating System

| | |
|---|---|
| 研　究　生：張順成 | Student：Soon Seng Teo |
| 指導教授：林大衛 | Advisor：Dr. David W. Lin |

國　立　交　通　大　學
電機學院通訊與網路科技產業研發碩士班
碩　士　論　文

A Thesis
Submitted to College of Electrical and Computer Engineering
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master
in
Industrial Technology R & D Master Program on
Communication Engineering

Jan 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年一月

# IEEE 802.16e OFDMA TDD 測距程序與整合建構於即時作業系統 DSP 平台之上行傳收系統

研究生:張順成　　　　　　　　　　　　　　　指導教授:林大衛 博士

國立交通大學電機學院產業研發碩士班

## 摘要

本篇論文首先介紹 IEEE 802.16e 測距程序及在我們研究中所使用到的測距演算法。接著，我們將探討如何在數位訊號處理器平台上運用即時作業系統來整合上行傳收系統。

在WiMAX的規格裡，測距是一個很重要的程序。其中，完成功率的調整、時間和頻率偏移的估計、以及基地台和所有用戶之間同步的測距程序，亦稱初始測距。一個行動裝置必須成功地與基地台完成初始測距程序後，才能建立起通訊網路。在這篇論文裡，我們只探討測距程序中之初始測距的細節及其演算法。然而，其他測距程序，如週期性測距及頻寬要求等議題，將不會在這篇論文裡詳談。

在測距程序裡，基地台會對所有接收到的測距碼作偵測、時間和頻率偏移之估計以及功率程度的量測。然後，基地台以廣播的方式將每組接收到的測距碼以及其所對應的時間、頻率和功率調整資訊回傳。此外，測距程序的成功與否也以廣播的方式告知。

在頻域上，我們大量地遞減測距碼的偵測以及時間和頻率偏移的估計所需的乘法運算量。而與時域上所用互相關演算法比較，我們所提出之演算方式只需要1%的乘法運算量。

在整合上，所實現的上行傳收系統建構於一台個人電腦以及四個Sundance數位處理器模組組成的數位處理器平台。在軟體的開發環境，我們選擇3L Diamond 平台，為3L公司以多處理器而設計及開發的平台系統。

在整合系統上，為了數位處理器的計算效益考量，所有的資料存放格式均是定點數。為了使加速程式執行的速度能夠達到即時處理之需求，我們也對程式做了最佳化處理，如:善加利用數位處理器所提供之函式庫。此外，程式碼的管線化處理及Diamond所支援的特定函式也有助於達到資料的即時處理。

最後，經實驗結果指出，用四個數位處理器模組處理一個上行傳收的架構需要6.5 ms。若我們用一個數位處理器模組來完成此工作則須花費17.16 ms。

# IEEE 802.16e OFDMA TDD Ranging Process and Uplink Transceiver Integration on DSP Platform with Real-Time Operating System

Student: Soon Seng Teo　　　　　Advisor: Dr. David W. Lin

Industrial Technology R & D Master Program of
Electrical and Computer Engineering College
National Chiao Tung University

## Abstract

In this thesis, we firstly introduce the IEEE 802.16e OFDMA TDD ranging process and its algorithms. Secondly, we discuss about how the uplink transceiver system is implemented on digital signal processor (DSP) platform with Real-Time Operating System (RTOS).

Ranging is one of significant processes in the mobile WiMAX standard. Power adjustment, timing and frequency offset estimation, and synchronization between a Base Station (BS) and all users within a cell are done through the ranging process, also known as initial ranging. Any Mobile Station (MS) that attempts to establish a communication link is required to carry out a successful initial ranging process with the BS. In this thesis, we discuss about the details of initial ranging and algorithm. In fact, there still exist other types of ranging processes such as: periodic ranging and bandwidth request. However, initial ranging is the only ranging process being discussed in this thesis.

In the initial ranging process, BS is required to detect different received ranging codes and estimate the timing, frequency offset and the power level for each user that transmits an initial ranging code. The BS then broadcasts the detected ranging codes with adjustment instructions for the timing, frequency and power level. The status notification of either successful ranging or retransmission is also broadcasted.

We accomplish the ranging code detection and the estimation of its timing and frequency offset in the frequency domain. The number of complex multiplications used for ranging code detection and estimation of its timing offset are only 1% of using cross correlation to complete that work in time domain.

In the integration work, we implement the uplink transceiver system on the DSP platform which includes a personal computer and four DSP modules (SMT395) made by Sundance. Also, we select Diamond platform as our development environment, which is a 3L's system for multiprocessor software design and implementation.

On the system, only fixed-point data format is supported because of the computational efficiency in DSP. Our optimization goal is to accelerate the speed of program execution time so that it can achieve the requirement of real-time processing. Thus, greatly make use of DSP library function is one of the optimization techniques. Furthermore, software pipelining and several specific functions supported by Diamond also contribute in achieving the real-time processing.

Finally, the experimental results show that our system needs 6.5 ms to process an uplink frame, whereas that needs 17.16 ms with only one processor.

# 誌謝

本篇論文的順利完成，首先誠摯地感謝指導教授，林大衛老師。感謝老師兩年來的細心指導，以及老師親切隨和的態度，使得 meeting 時有問題都能勇於發問，提高作研究的效率。祝福老師在忙綠之餘，能保有健康的身體。

另外，也要感謝俊榮、崑健和鴻志在研究上給予的協助和討論。也祝福所有實驗室的成員，學業順利，準時畢業。

感謝通訊電子與訊號處理實驗室(commlab)，提供了充足的軟硬體資源，讓我在研究中不虞匱乏。資策會和國科會這兩年來的合作與協助，使實驗室的軟硬體資源更加充裕，誠心感謝。

最後，要感謝的是我的家人，他們的支持讓我能夠心無旁騖的完成學業。

謝謝所有幫助過我、陪我走過這一段歲月的師長、同學與家人。謝謝！

張順成

2008 年.1 月 於新竹

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Orthogonal frequency division multiple access (OFDMA) has gained an increasing interest recently because of its capability to cope with inter symbol interference (ISI), spectral efficiency, robustness in the multipath propagation environment and ability to exploit multiuser diversity. It has been utilized as an important physical layer technique in the IEEE wireless metropolitan area network (MAN) standard 802.16.

An amendment to 802.16-2004, IEEE 802.16e-2005, addresses mobility. It implements a number of enhancements to 802.16-2004, including better support for Quality of Service and the use of Scalable OFDMA.

IEEE 802.16e-2005 is sometimes somewhat erroneously called Mobile WiMAX, after the WiMAX Forum for interoperability. WiMAX is a telecommunications technology aimed at providing wireless data over long distances in a variety of ways, from point-to-point links to full mobile cellular type access [1].

Our study is based on the IEEE 802.16e-2005 standard and can be divided into two parts. The first part is the ranging techniques for IEEE 802.16e OFDMA and the second part is the digital signal processor (DSP) implementation of the uplink transceiver system.

In the first part, we discuss the ranging issue and algorithm. Ranging is one significant process in the mobile WiMAX standard. Power adjustment, timing and frequency offset estimation, and synchronization between a Base Station (BS) and all users within a cell are done through the ranging process.

In the second part, we implement a full-software uplink transceiver system on Texas Instrument (TI)'s DSP. Moreover, we employ various optimization techniques to accelerate the execution speed of the programs in the DSP implementation.

This thesis is organized as follows. We first introduce the IEEE 802.16e Wireless MAN OFDMA standard in chapter 2. Chapter 3 introduces the DSP implementation platform. We discuss the ranging problems and present some solutions in chapter 4. In chapter 5, uplink transceiver integration on DSP platform with real-time operating system is discussed. Finally, the conclusion is given in chapter 6, where we also point out some potential future work.

# Chapter 2

# Introduction to IEEE 802.16e OFDMA

In this chapter, we first introduce some basic concepts regarding OFDMA. Then we give an overview of the IEEE 802.16e OFDMA standard. For the sake of simplicity, we only introduce the specifications that are useful in our study. Other specifications like channel coding, synchronization, channel estimation, etc., are not our concern and are ignored in this introduction. For more details we refer the reader to [2] and [3], from which we take much of the material in this chapter.

## 2.1 Introduction to OFDMA

Orthogonal frequency division multiple access (OFDMA) is a multi-user version of the popular OFDM digital modulation scheme. It is used in the mobility mode of the IEEE 802.16 WirelessMAN air interface standard, often referred to as Mobile WiMAX.

In OFDMA, multiple access is realized by providing each user with a fraction of the available number of subcarriers, as shown in Figure 2.1. In this way, it is similar to ordinary frequency division multiple access (FDMA); however, OFDMA avoids the relatively large guard bands that are necessary in FDMA to separate different users.

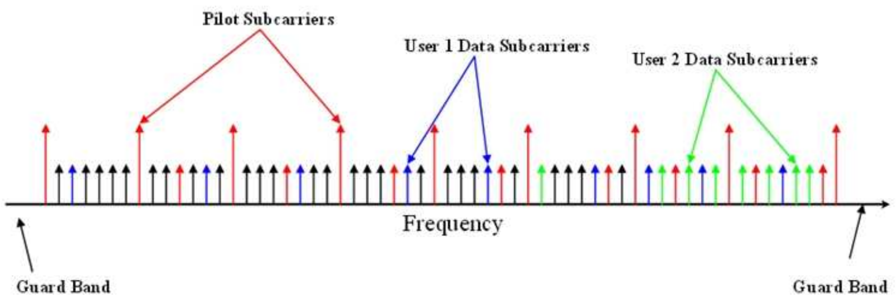


Figure 2.1: Multiple access achieved in OFDMA (from [4]).

OFDMA can also be described as a combination of frequency domain and time domain multiple access, where the resources are partitioned in the time-frequency space, and slots are assigned along the OFDM symbol index as well as OFDM subcarrier index [4].

## 2.2 Introduction to IEEE 802.16e

The IEEE 802.16 family of standards is officially called WirelessMAN. Part of it has been dubbed WiMAX (Worldwide Interoperability for Microwave Access) by an industry group called the WiMAX Forum. The mission of the Forum is to promote and certify compatibility and interoperability of broadband wireless products.

The first 802.16 standard was approved in December 2001. It is a standard for point to multi-point broadband wireless transmission in the 10–66 GHz band, with only a line-of-sight (LOS) capability. It uses a single carrier (SC) physical layer (PHY) technique.

To overcome the disadvantage of the LOS requirement, the 802.16a standard was approved in 2003 to support non-line-of-sight (NLOS) links, operational in both licensed and unlicensed frequency bands from 2 to 11 GHz. It was subsequently integrated with the original 802.16 and 802.16c standards to create the 802.16-2004 standard (initially codenamed 802.16d). With such enhancements, the 802.16-2004 standard has been viewed as a promising alternative for providing the last-mile connectivity by radio link. However, the 802.16-2004 specifications were devised primarily for fixed wireless users.

An amendment to 802.16-2004, IEEE 802.16e-2005, addressing mobility, has been published in Febuary 2006. It specifies four air interfaces at PHY: OFDMA, SCa, OFDM, and SC. This study is concerned with the OFDMA PHY in a mobile communication environment.

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

## 2.3 Concept of OFDMA and Definition of Basic Terms

We present some basic concepts and terminology of OFDMA signaling in this section. The contents of this section have been taken to a large extent from [2] and [3].

### 2.3.1 Frequency Domain and Time Domain Descriptions

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

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

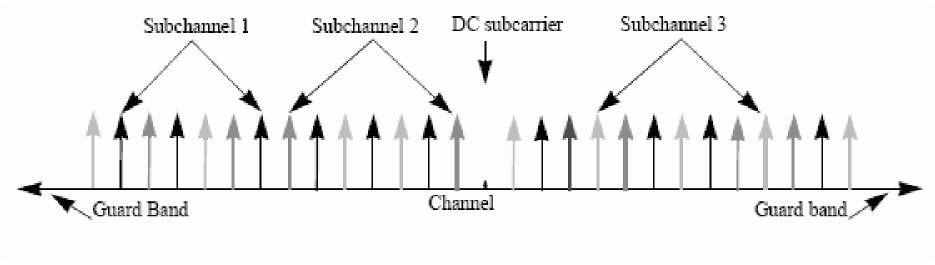


Figure 2.2: OFDMA frequency description (from [2], Figure 214).

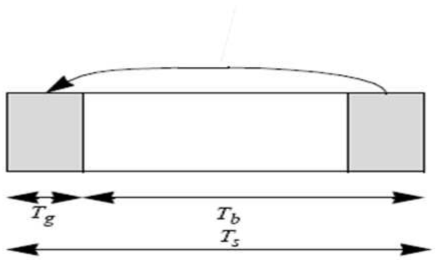


Figure 2.3: OFDMA frequency description (from [2], Figure 213).

In the OFDMA mode, the active subcarriers are divided into subsets of subcarriers, where each subset is termed a subchannel. In the downlink, a subchannel may be intended for different (groups of) receivers; in the uplink, a transmitter may be assigned one or more subchannels and several transmitters may transmit simultaneously. The subcarriers forming one subchannel may, but need not be, adjacent. The concept is shown in Figure 2.2.

Inverse-Fourier-transforming creates the OFDMA waveform; its time duration is referred to as the useful symbol time $T_b$. A copy of the last $T_g$ of the useful symbol period, termed cyclic prefix (CP), is used to collect multipaths while maintaining the orthogonality of the tones. Figure 2.3 illustrates the structure.

### 2.3.2 Parameters

#### Primitive Parameters

Four primitive parameters characterize the OFDMA symbols:

- $BW$: The nominal channel bandwidth.
- $N_{used}$: Number of used subcarriers (which includes the DC subcarrier).
- $n$: Sampling factor. Its value is set as follows: For channel bandwidths that are a multiple of 1.75 MHz, $n = 8/7$; else for channel bandwidths that are a multiple of any of 1.25, 1.5, 2 or 2.75 MHz, $n = 28/25$; else for channel bandwidths not otherwise specified, $n = 8/7$.
- $G$: This is the ratio of CP time to "useful" time, i.e., $T_{cp}/T_s$.

#### Derived Parameters

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

- $N_{FFT}$: Smallest power of two greater than $N_{used}$.
- Sampling frequency: $F_s = floor(n \cdot BW/8000) \times 8000$.
- 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: $T_b/N_{FFT}$.

Table 2.1 specifies the system parameters used in our study.

Table 2.1: System Parameters Used in Our Study

| Parameter | Value |
|---|---|
| System Channel Bandwidth (MHz) | 10 |
| Sampling Frequency (MHz) | 11.2 |
| FFT Size | 1024 |
| Subcarrier Spacing (kHz) | 10.94 |
| Useful Symbol Time ($\mu$sec) | 91.4 |
| Guard Time ($\mu$sec) | 11.4 |
| OFDMA Symbol Time ($\mu$sec) | 102.9 |
| $G$ (Ratio of CP Time to "Useful" Time) | 1/8 |
| $n$ (Sampling Factor) | 28/25 |

### 2.3.3 Definition of Basic Terms

We introduce some basic terms in the OFDMA PHY:

- Slot: A slot in the OFDMA PHY is a two-dimensional entity spanning both a time and a subchannel dimension. It is the minimum unit for data allocation. For DL PUSC (Partial Usage of SubChannels), one slot is one subchannel by two OFDMA symbols. For UL, one slot is one subchannel by three OFDMA symbols.

- Data region: In OFDMA, a data region is a two-dimensional allocation of a group of contiguous subchannels, in a group of contiguous OFDMA symbols. All the allocations refer to logical subchannels. A two dimensional allocation may be visualized as a rectangle, such as the $4 \times 3$ rectangle shown in Fig. 2.4.

- Segment: A segment is a subdivision of the set of available OFDMA subchannels (that may include all available subchannels). One segment is used for deploying a single instance of the MAC.

- Permutation zone: A permutation zone is a number of contiguous OFDMA symbols,

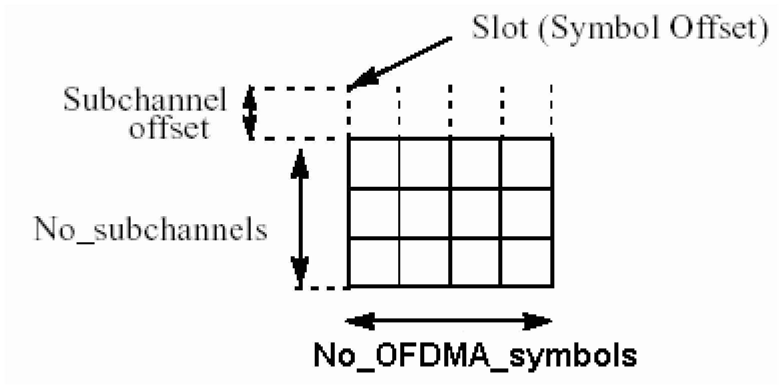


Figure 2.4: Example of the data region which defines the OFDMA allocation (from [2]).

in the DL or the UL, that use the same permutation formula. The DL subframe or the UL subframe may contain more than one permutation zone.

### 2.3.4 Frame Structure

When implementing a time-division duplex (TDD) system, the frame structure is built from BS and SS transmissions. Each frame in the DL transmission begins with a preamble followed by a DL transmission period and an UL transmission period. In each frame, the TTG (transmit/receive transition gap) and RTG (receive/transmit transition gap) shall be inserted between the downlink and uplink and at the end of each frame, respectively, to allow the BS to turn around. Figure 2.5 shows an example of an OFDMA frame with only mandatory zone in TDD mode.

## 2.4 OFDMA Subcarrier Allocation

The OFDMA PHY defines four selectable FFT sizes: 2048, 1024, 512, and 128. For convenience, here we only take the 1024-FFT OFDMA subcarrier allocation for introduction. The subcarriers are divided into three types: null (guard band and DC), pilot, and data.

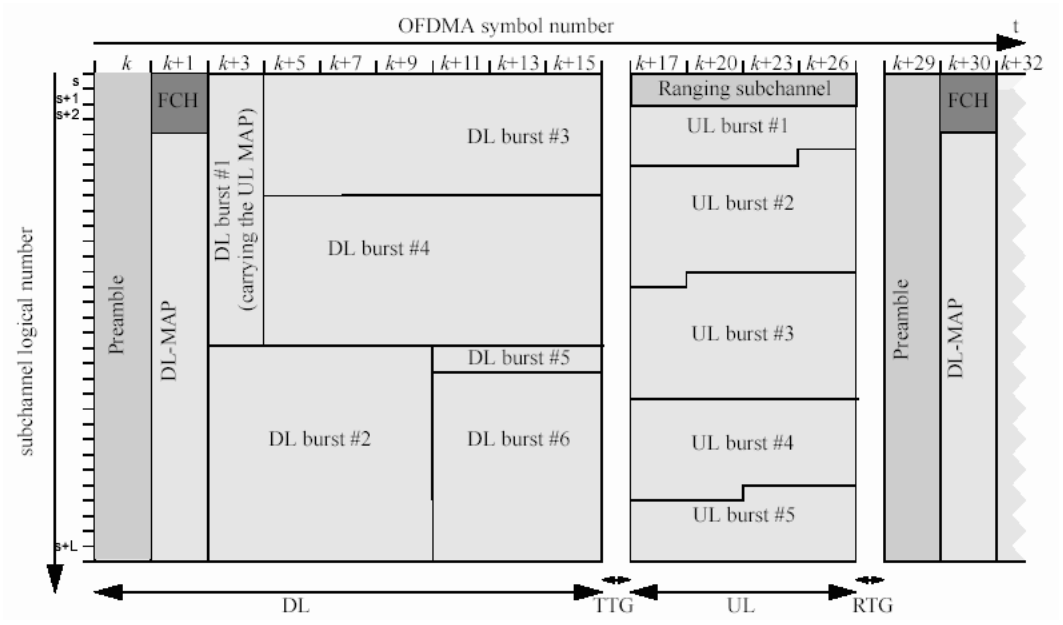


Figure 2.5: Example of an OFDMA frame (with only mandatory zone) in TDD mode (from [3]).

Subtracting the guard tones from $N_{FFT}$, one obtains the set of used subcarriers $N_{used}$. For both uplink and downlink, these used subcarriers are allocated to pilot subcarriers and data subcarriers. However, there is a difference between the different possible zones. For FUSC (full usage of subchannels)and PUSC, in the downlink, the pilot tones are allocated first; what remains are data subcarriers, which are divided into subchannels that are used exclusively for data. Thus, in FUSC, there is one set of common pilot subcarriers, and in PUSC downlink, there is one set of common pilot subcarriers in each major group. In PUSC uplink, each subchannel contains its own pilot subcarriers.

## 2.4.1 Uplink

The contents of this subsection have been taken to a large extent from [2] and [3].

**Data Mapping Rules**

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

Step 1: Allocate OFDMA slots to bursts.

1) Segment the data into blocks sized to fit into one OFDMA slot.

2) Each slot shall span one or more subchannels in the subchannel axis and one or more OFDMA symbols in the time axis (see Figure 2.6 for an example). Map the slots such that the lowest numbered slot occupies the lowest numbered subchannel in the lowest numbered OFDMA symbol.

3) Continue the mapping such that the OFDMA symbol index is increased. When the edge of the UL zone is reached, continue the mapping from the lowest numbered OFDMA symbol in the next available subchannel.

4) An UL allocation is created by selecting an integer number of contiguous slots, according to the ordering of steps 1 to 3. This results in the general burst structure shown by the gray area in Figure 2.6.

Step 2: Map OFDMA slots within the UL allocation.

1) Map the slots such that the lowest numbered slot occupies the lowest numbered subchannel in the lowest numbered OFDMA symbol.

2) Continue the mapping such that the subchannel index is increased. When the last subchannel is reached, continue the mapping from the lowest numbered subchannel in the next OFDMA symbol that belongs to the UL allocation. The resulting order is shown by the arrows in Figure 2.6.

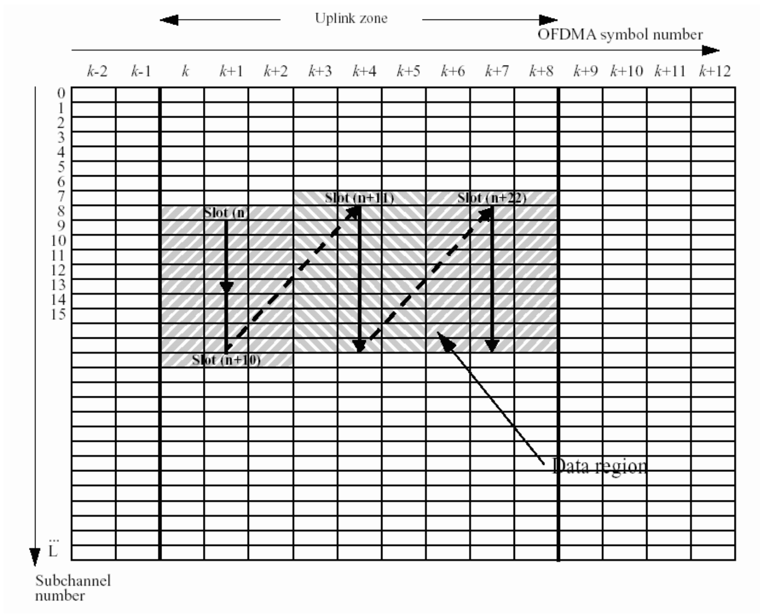


Figure 2.6: Example of mapping OFDMA slots to subchannels and symbols in the uplink (from [3]).

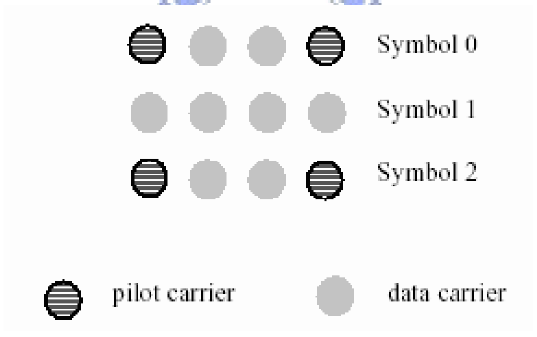


Figure 2.7: Description of an uplink tile (from [2]).

### Carrier Allocations

The uplink supports 35 subchannels in the 1024-FFT PUSC permutation. Each transmission uses 48 data carriers as the minimal block of processing. Each new transmission for the uplink commences with the parameters as given in Table 2.2.

A slot in the uplink is composed of one subchannel in three OFDMA symbols. Within

Table 2.2: OFDMA Uplink Subcarrier Allocation [2], [3]

| Parameter | Value | Notes |
|---|---|---|
| Number of DC subcarriers | 1 | Index 512 (counting from 0) |
| $N_{used}$ | 841 | Number of all subcarriers used within a symbol |
| Guard subcarriers: | 92,91 | Left, right |
| TilePermutation | | Used to allocate tiles to subchannels 11, 19, 12, 32, 33, 9, 30, 7, 4, 2, 13, 8, 17, 23, 27, 5, 15, 34, 22, 14, 21, 1, 0, 24, 3, 26, 29, 31, 20, 25, 16, 10, 6, 28, 18 |
| $N_{subchannels}$ | 35 | |
| $N_{subcarriers}$ | 48 | |
| $N_{tiles}$ | 210 | |
| Number of subcarriers per tile | 4 | Number of all subcarriers within a tile |
| Tiles per subchannel | 6 | |

each slot, there are 48 data subcarriers and 24 pilot subcarriers. The subchannel is constructed from six uplink tiles, each having four successive active subcarriers with the configuration as illustrated in Figure 2.7.

The usable subcarriers in the allocated frequency band shall be divided into $N_{tiles}$ physical tiles with parameters from Table 2.2. The allocation of physical tiles to logical tiles in subchannels is performed according to:

$$Tiles(s,n) = N_{subchannels} \cdot n + (Pt[(s+n)\ mod\ N_{subchannels}] + UL\_PermBase) \text{mod } N_{subchannels} \tag{2.1}$$

where:

- $Tiles(s,n)$ is the physical tile index in the FFT with tiles being ordered consecutively from the most negative to the most positive used subcarrier (0 is the starting tile index),
- $n = 0..5$ is the tile index in a subchannel,

- $Pt$ is the tile permutation function,
- $s$ is the subchannel number in the range $0..N_{subchannels} - 1$,
- $UL\_PermBase$ is an integer value in the range 0..69, which is assigned by a management entity, and
- $N_{subchannels}$ is the number of subchannels for the FFT size given in Table 2.2.

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

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

2) The mapping of data onto the subcarriers follows the equation below. This equation calculates the subcarrier index (as assigned in item 1) to which the data constellation point is to be mapped:

$$Subcarrier(n, s) = (n + 13 \cdot s)\ mod\ N_{subcarriers} \tag{2.2}$$

where:

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

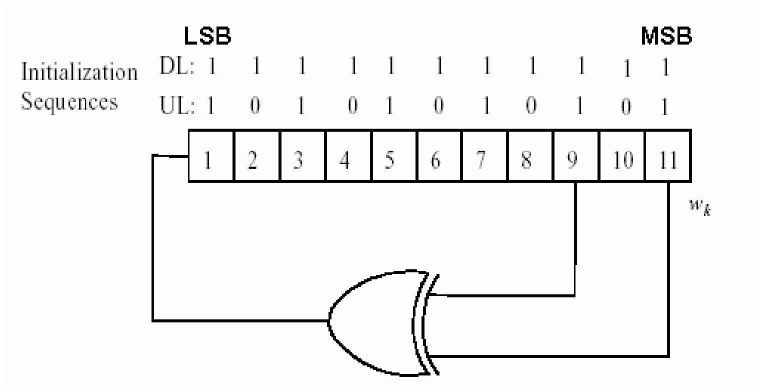


Figure 2.8: PRBS generator for pilot modulation (from [2] and [3]).

### Pilot Modulation

The PRBS (pseudo-random binary sequence) generator depicted in Figure 2.8 is used to produce a sequence, $w_k$. The value of the pilot modulation, on subcarrier $k$, is derived from $w_k$.

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

$$\Re\{c_k\} = 2(\frac{1}{2} - w_k), \quad \Im\{c_k\} = 0. \tag{2.3}$$

## 2.4.2 Downlink

The contents of this subsection have been taken to a large extent from [2] and [3].

### Data Mapping Rules

The downlink data mapping rules are as follows:

1. Segment the data after the modulation block into blocks sized to fit into one OFDMA

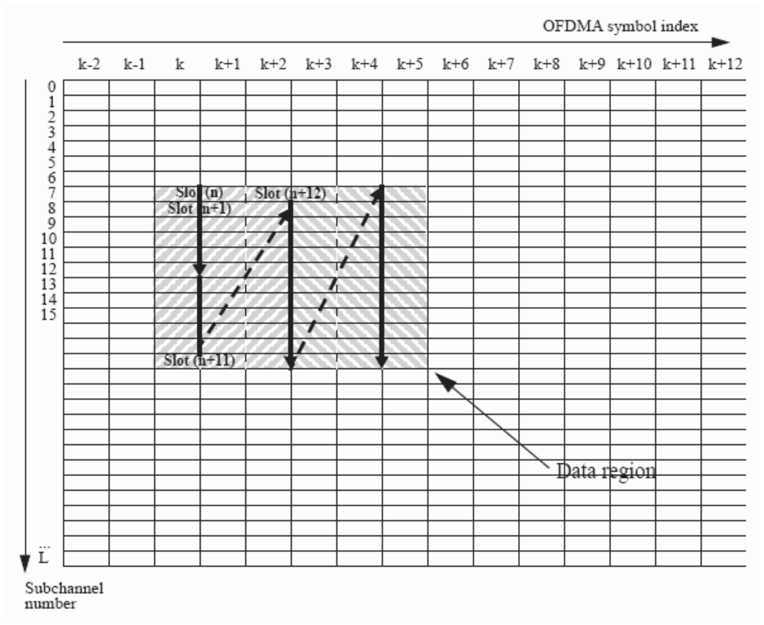


Figure 2.9: Example of mapping OFDMA slots to subchannels and symbols in the downlink in PUSC mode (from [3]).

slot.

2. Each slot shall span one subchannel in the subchannel axis and one or more OFDMA symbols in the time axis, as per the slot definition mentioned before. Map the slots such that the lowest numbered slot occupies the lowest numbered subchannel in the lowest numbered OFDMA symbol.

3. Continue the mapping such that the OFDMA subchannel index is increased. When the edge of the data region is reached, continue the mapping from the lowest numbered OFDMA subchannel in the next available symbol.

Figure 2.9 illustrates the order of OFDMA slots mapping to subchannels and OFDMA symbols.

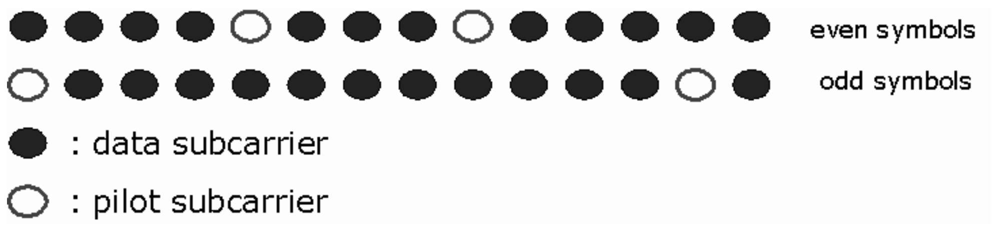


Figure 2.10: Cluster structure (from [3]).

### Subcarrier Allocations

The OFDMA symbol structure is constructed using pilots, data and zero subcarriers. The symbol is first divided into basic clusters and zero carriers are allocated. The pilot tones are allocated first; what remains are data subcarriers, which are divided into subchannels that are used exclusively for data. Pilots and data carriers are allocated within each cluster.

Figure 2.10 shows the cluster structure with subcarriers from left to right in order of increasing subcarrier index. For the purpose of determining PUSC pilot location, even and odd symbols are counted from the beginning of the current zone. The first symbol in the zone is even. The preamble is not counted as part of the first zone. Table 2.3 summarizes the parameters of the OFDMA PUSC symbol structure.

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

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

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

$$LogicalCluster = \begin{cases} RenumberingSequence(PhysicalCluster), & \text{first DL zone,} \\ RenumberingSequence\big((PhysicalCluster+ \\ \qquad 13 \cdot DL\_PermBase) \bmod N_{clusters}\big), & \text{otherwise.} \end{cases} \tag{2.4}$$

3) Divide the clusters into six major groups. Group 0 includes clusters 0–11, group 1

Table 2.3: OFDMA Downlink Subcarrier Allocation under PUSC [2], [3]

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

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

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

$$\begin{aligned} subcarrier(k,s) = {} & (N_{subchannels} \cdot n_k) \\ & + \{p_s[n_k \; mod \; N_{subchannels}] + DL\_PermBase\} \mathrm{mod} \; N_{subchannels} \end{aligned} \tag{2.5}$$

where:

- $subcarrier(k, s)$ is the subcarrier index of subcarrier $k$ in subchannel $s$,
- $s$ is the index number of a subchannel, from the set $[0..N_{subchannels} - 1]$,
- $n_k = (k + 13 \cdot s) \mathrm{mod} \; N_{subcarriers}$, where $k$ is the subcarrier-in-subchannel index from the set $[0..N_{subcarriers} - 1]$,
- $N_{subchannels}$ is the number of subchannels (for PUSC use number of subchannels in the currently partitioned group),
- $p_s[j]$ is the series obtained by rotating basic permutation sequence cyclically to the left $s$ times,
- $N_{subcarriers}$ is the number of data subcarriers allocated to a subchannel in each OFDMA symbol, and
- $DL\_PermBase$ is an integer from 0 to 31.

## Pilot Modulation

Pilot subcarriers are inserted into each data burst in order to constitute the symbol. The PRBS generator depicted in Figure 2.8 is used to produce a sequence, $w_k$.

Each pilot is transmitted with a boosting of 2.5 dB over the average non-boosted power of a data tone. The pilot subcarriers are modulated according to

$$\Re\{c_k\} = \frac{8}{3}\left(\frac{1}{2} - w_k\right), \quad \Im\{c_k\} = 0. \tag{2.6}$$

# Chapter 3

# DSP Implementation Environment

In this chapter, we introduce the DSP platform used in our implementation, which is a Sundance's product. In addition to hardware introduction, we also introduce the software development tools and code development techniques.

## 3.1 DSP Implementation Platform

A typical DSP application using Sundance equipment is made up from a host personal computer (PC) holding one or more "carrier boards", each supporting one or more processing elements (DSPs, I/O devices, or FPGAs) known as the Texas Instruments Module (TIM) [5].

In our DSP application (Figure 3.1), the carrier board (SMT310Q) plugs into a host PC using a PCI slot and does not run any program; it simply holds the TIMs (SMT395), providing them with power and a means of communicating with the rest of the world.

### 3.1.1 Requirements on the PC [5]

Recommended minimum requirements:

- Hardware: 233 MHz Pentium.

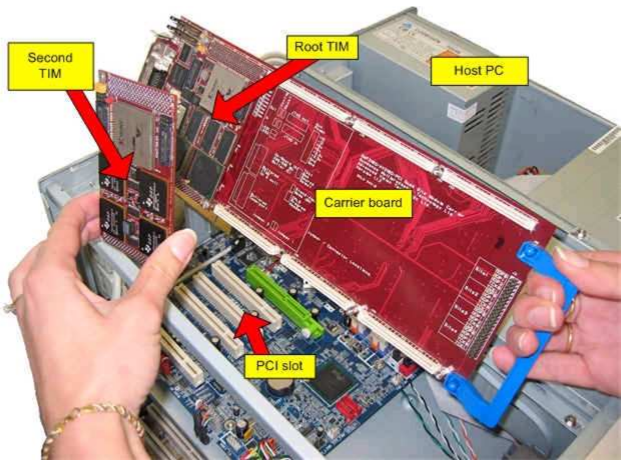


Figure 3.1: DSP application using Sundance equipment (from [5]).

- Memory: 64 MB of RAM.
- Operating system: Windows ME, Windows NT4 SP6, Windows 2000, Windows XP Home or Windows XP Pro. Note that Windows 3.x, Windows 95, and Windows 98 are not supported.
- Disk space: Depends on the software we choose to install and the options selected. The host software needs approximately 50 MB.
- Power supply: Depends on how many hardware devices are being driven, but typically for three carrier boards populated with TIMs may require a 20 amp (3.3 V) power supply.

### 3.1.2 The Carrier Board (SMT310Q) [6]

The SMT310Q is a quad site module carrier developed to provide access to TIM modules over the PCI bus running at 33 MHz with a 32-bit data bus. As shown in Figure 3.2, the main connection to the half-length PCI bus is via the module's global bus. A single ComPort

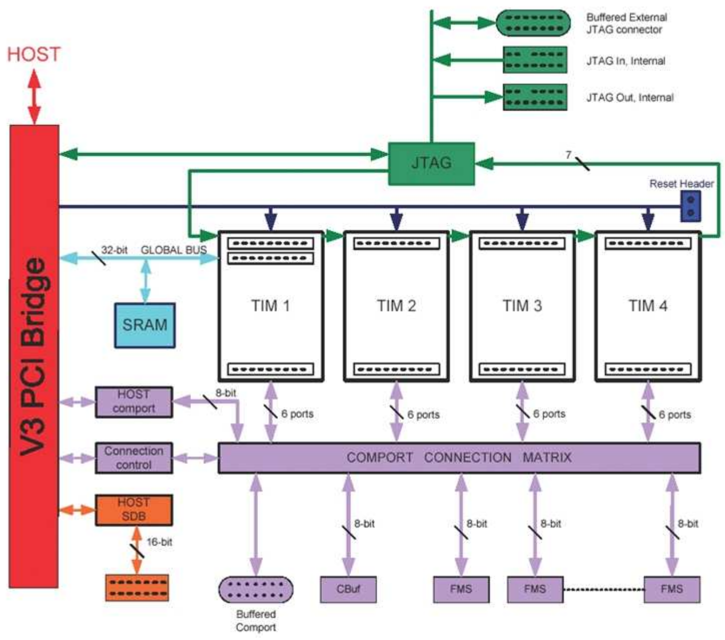


Figure 3.2: Carrier board SMT310Q (from [6]).

is also mapped to the PCI bus providing support for application boot and data transfer. 1 Mbyte of SRAM is mapped on to the global bus and can be accessed by the module as a global resource or by the PCI bridge. A 3.3 volt supply is available at the fixing pillars for the module. This supply is taken from the PCI edge connector.

### 3.1.3 Host Connection

As shown in Figure 3.3, the first slot on a multi-TIM carrier board has an extra global bus connector that is connected to the carrier's host interface and on-board resources. A TIM on the first TIM slot of a board is known as the Root TIM, it can access the carrier board resources.

The Root TIM of a carrier board plugged into a host PC has two ways to communicate with the host:

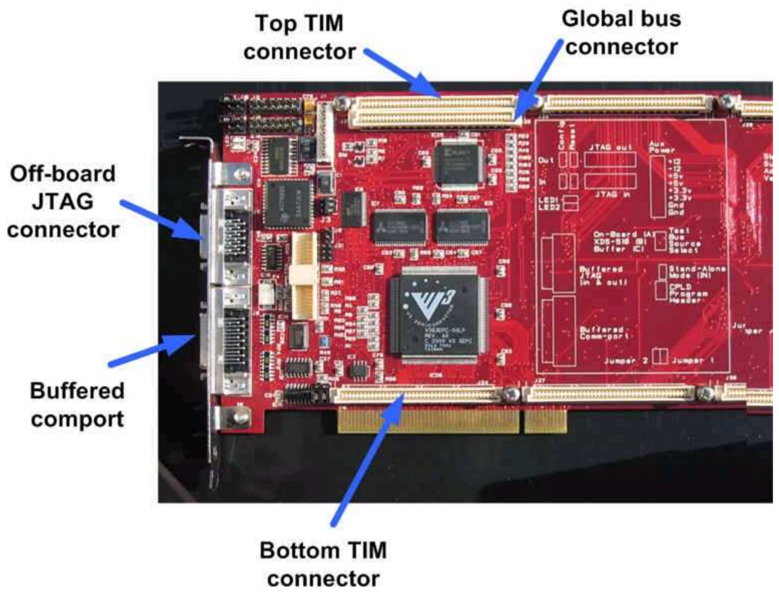


Figure 3.3: Carrier board communitcation (from [5]).

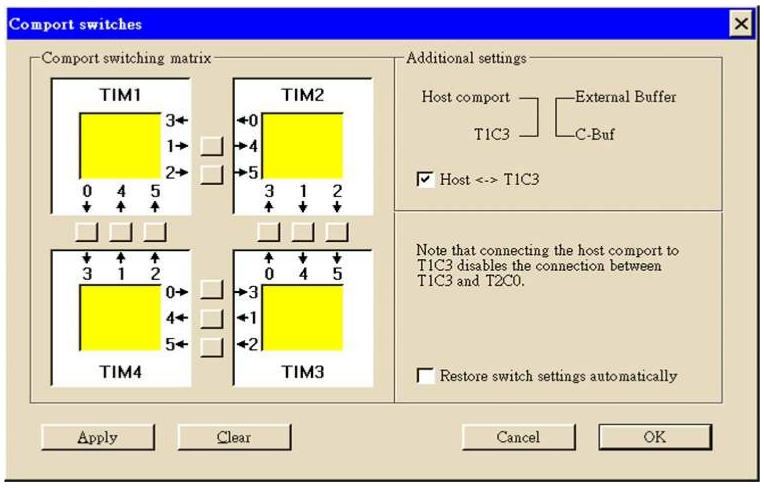
- Host ComPort.
- PCI connection to host.

The host ComPort connection provides a communication path between the host and the TIM in the first site of the carrier board. It typically runs at 2 MB/s. ComPort 3 of the first TIM site is normally connected to the host ComPort on the carrier board, as shown in Figure 3.4's additional settings.

The Sundance carrier boards include a PCI bridge chip that gives the first TIM access to host resources, such as interrupts, I/O and host memory ranges. The TIM can use this PCI connection for high-speed data transfer between the carrier board and the host memory.

### 3.1.4 The Texas Instruments Module (SMT395)

The TIMs used in our implementation are Sundance's SMT395 shown in Figure 3.5. It is based on the 1 GHz 64-bit TMS320C6416T fixed-point DSP, manufactured on 90 nm wafer


Comport switches

Comport switching matrix

Additional settings

Host comport — T1C3 — External Buffer / C-Buf

[✓] Host <-> T1C3

Note that connecting the host comport to T1C3 disables the connection between T1C3 and T2C0.

[ ] Restore switch settings automatically

Figure 3.4: ComPort switches (from [5]).

technology. The SMT395 is supported by the TI Code Composer Studio and 3L Diamond RTOS (real-time operating system) to enable full multi-DSP systems with minimum efforts by the programmers [6].

Some features of SMT395 are:

- 1 GHz TMS320C6416T fixed-point DSP processor with L1, L2 cache and SDRAM.
- 8000 MIPS peak DSP performance.
- Xilinx Virtex II Pro FPGA XC2VP30-6 in FF896 package.
- 256 Mbytes of SDRAM at 133 MHz.
- Eight 2 Gbit/sec Rocket Serial Links (RSL) for inter module communication.
- Two Sundance High-speed Bus (50, 100 or 200 MHz) ports at 32 bits width.
- Six ComPorts up to 20 Mbytes/s each for inter-DSP communication.
- 8 Mbytes flash ROM for configuration and booting.

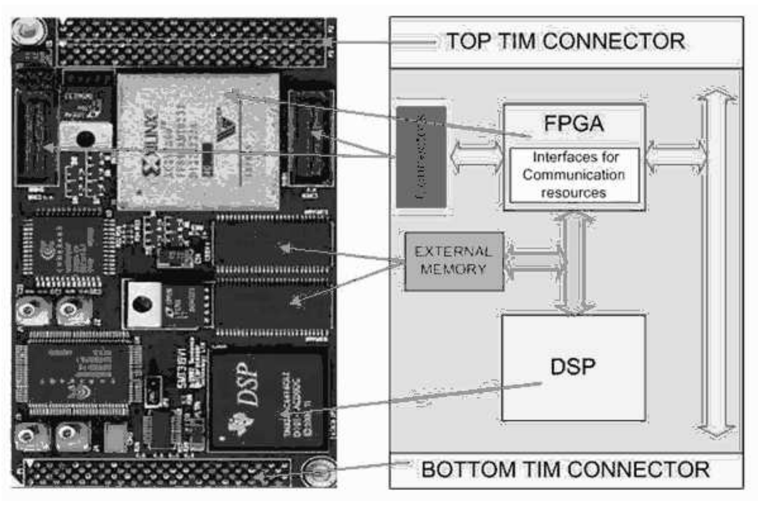


Figure 3.5: The SMT395 module (from [5]).

**DSP Chip [7]**

The TMS320C6416T DSP is a fixed-point DSP in the TMS320C64x series of the TMS320C6000 DSP platform family. It is based on the VelociTI very-long-instruction-word (VLIW) architecture developed by TI.

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

- VLIW CPU with eight functional units, including two multipliers and six arithmetic logic units:
  - Executes up to eight instructions per cycle.
  - Allows designers to develop highly effective RISC-like code for fast development time.
- Instruction packing:
  - Gives code size equivalence for eight instructions executed serially or in parallel.

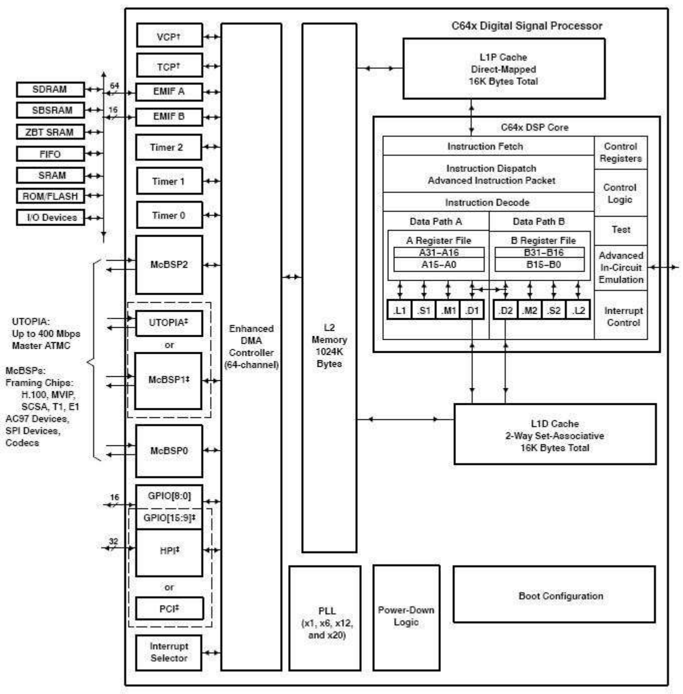


Figure 3.6: Functional block and CPU diagram of the DSP [8].

  – Reduces code size, program fetches, and power consumption.

- Efficient code execution on independent functional units:

  – Assembly optimizer for fast development and improved parallelization.

  – Efficient C complier on DSP benchmark suite.

- Conditional execution of all instructions:

  – Reduces costly branching.

  – Increases parallelism for higher sustained performance.

**Central Processing Unit [8]**

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 Figure 3.6, each of which containing four functional units and one register file. The four functional units are as follows. The first unit is for multiplier operations (.M). The second unit is for arithmetic and logic operations (.L). The third is for branch, byte shifts, and arithmetic operations (.S). And the last is for linear and circular address calculation to load and store with external memory operations (.D). The details of the functional units are described in Table 3.1.

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 multiple functional units in the same execute packet.

Table 3.1: Functional Units and Operations Performed (from [9])

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

**Memory Architecture and Peripherals**

The C64x is a two-level cache-based architecture. The level 1 cache is separated into program and data spaces. The level 1 program cache (L1P) is a 128 Kbit direct mapped cache and the level 1 data cache (L1D) is a 128 Kbit 2-way set-associative mapped cache. The level 2 (L2) memory consists of 8 Mbit memory space for cache (up to 256 Kbytes) and unified mapped memory. The external memory interface (EMIF) provides interfaces for the DSP core and external memory, such as synchronous-burst SRAM (SBSRAM), synchronous DRAM (SRAM), SDRAM, FIFO and asynchronous memories (SRAM and EPROM). The EMIF also provides 64-bit-wide (EMIFA) and 16-bit-wide (EMIFB) memory read capability. The C64x contains some peripherals such as enhanced direct-memory-access (EDMA), host-port interface (HPI), PCI, three multichannel buffered serial ports (McBSPs), three 32-bit general-purpose timers and sixteen general-purpose I/O pins. The EDMA controller handles all data transfers between the level-two (L2) cache/memory and the device peripheral. The C64x has 64 independent channels. The HPI is a 32-/16-bit wide parallel port through which a host processor can directly access the CPUs memory space. The PCI port supports connection of the DSP to a PCI host via the integrated PCI master/slave bus interface.

### 3.1.5 Interconnection Mechanisms

As shown in Figure 3.7, there are several different ways to exchange data between Sundance hardware modules and other peripheral devices; the data rate required by application will usually determine which one to use. The data exchange mechanisms include:

- Communication Ports (ComPorts).
- Sundance Digital Link (SDL).
- Sundance Digital Bus (SDB).

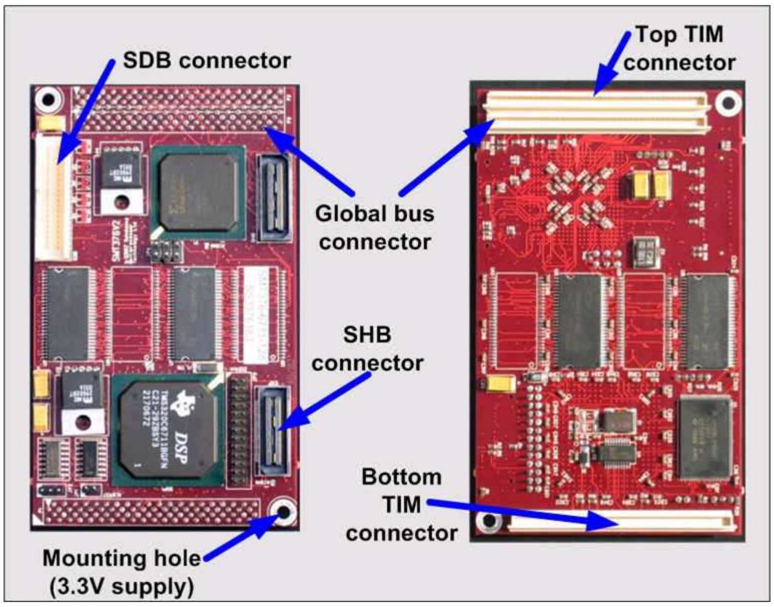


Figure 3.7: Interconnection mechanisms of TIM (from [5]).

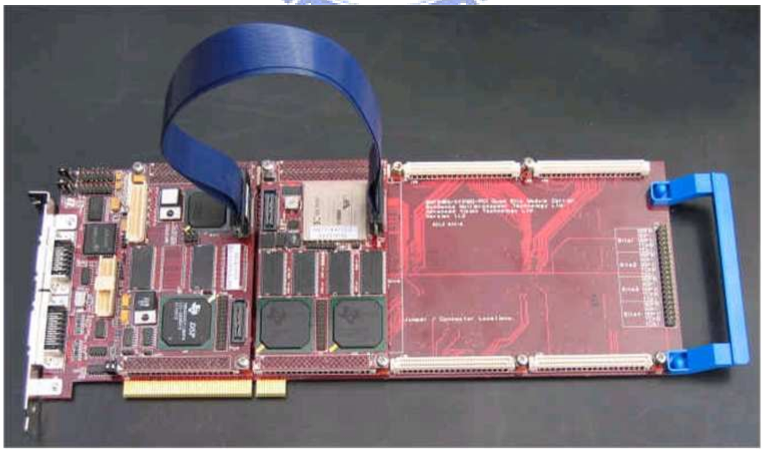


Figure 3.8: Sundance High-speed Bus (from [5]).

- Sundance High-speed Bus (SHB).
- Rocket-IO Serial Link (RSL).
- Global bus.

In our system, we use four TIMs and employ SHB cables to connect them. The SHB cable transfers 32 bits at a time. For each 32-bit access made by the DSP, the firmware implementation directly accesses the 32 physical bits provided by the SHB physical device. Figure 3.8 illustrates two TIMs connected by a SHB cable.

## 3.2 Code Development Environment

In our study, we use two software development tools, Code Composer Studio (CCS) and 3L Diamond. Sundance advises their customers to design software solutions for multi-DSP systems and to implement hardware solutions for mixed multi-FPGA/DSP systems with the 3L Diamond products. We introduce CCS in this section, and present an introduction to 3L Diamond in Chapter 5 which discusses the integration work.

### 3.2.1 Code Composer Studio [10]

Code Composer Studio is a Texas Instruments product. It provides the compiler, assembler, and linker that we need to be able to generate programs that will execute on DSP TIMs. It also provides software for debugging programs by watching and interacting with their execution.

Code Composer Studio furnishes its own development environment to load code to particular processors; all processors in the system must be loaded separately. As this can be inconvenient and error-prone and does not include any support for Sundance peripherals, Sundance recommends the developer to develop the code using the Diamond real-time operating system from 3L, for efficient use of both the developer's time and the DSP resources.

Some main features of the CCS are listed below:

- Real-time analysis.
- Source code debugger common interface for both simulator and emulator targets:
  - C/C++/assembly language support.
  - Simple breakpoints.
  - Advanced watch window.
  - Symbol browser.
- DSP/BIOS host tooling support (configure, real-time analysis and debug).
- Data transfer for real time data exchange between host and target.
- Profiler to understand code performance.
- DSP/BIOS support:
  - Pre-emptive multi-threading.
  - Interthread communication.
  - Interupt handling.
- 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 libraries include many C-callable, assembly-optimized, general-purpose signal-processing and image/video processing rou-

tines. These routines are typically used in computationally intensive real-time applications where optimal execution speed is critical.

### 3.2.2 Code Development Flow [11]

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. Figure 3.9 illustrates the three phases in the code development flow. Because phase 3 is usually too detailed and time consuming, most of the time we should not need to go into phase 3 to write linear assembly code unless the software pipelining efficiency is too bad or the resource allocation is too unbalanced.

## 3.3 Code Optimization and Acceleration

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

- Use the clock( ) and printf( ) functions in C/C++ to time and display the performance of specific code regions. Use the stand-alone simulator (load6x) to run the code for this purpose.
- Use the profile mode of the stand-alone simulator. This can be done by compiling the code with the -mg option and executing load6x with the -g option. Then enable the clock and use profile points and the RUN command in the Code Composer debugger

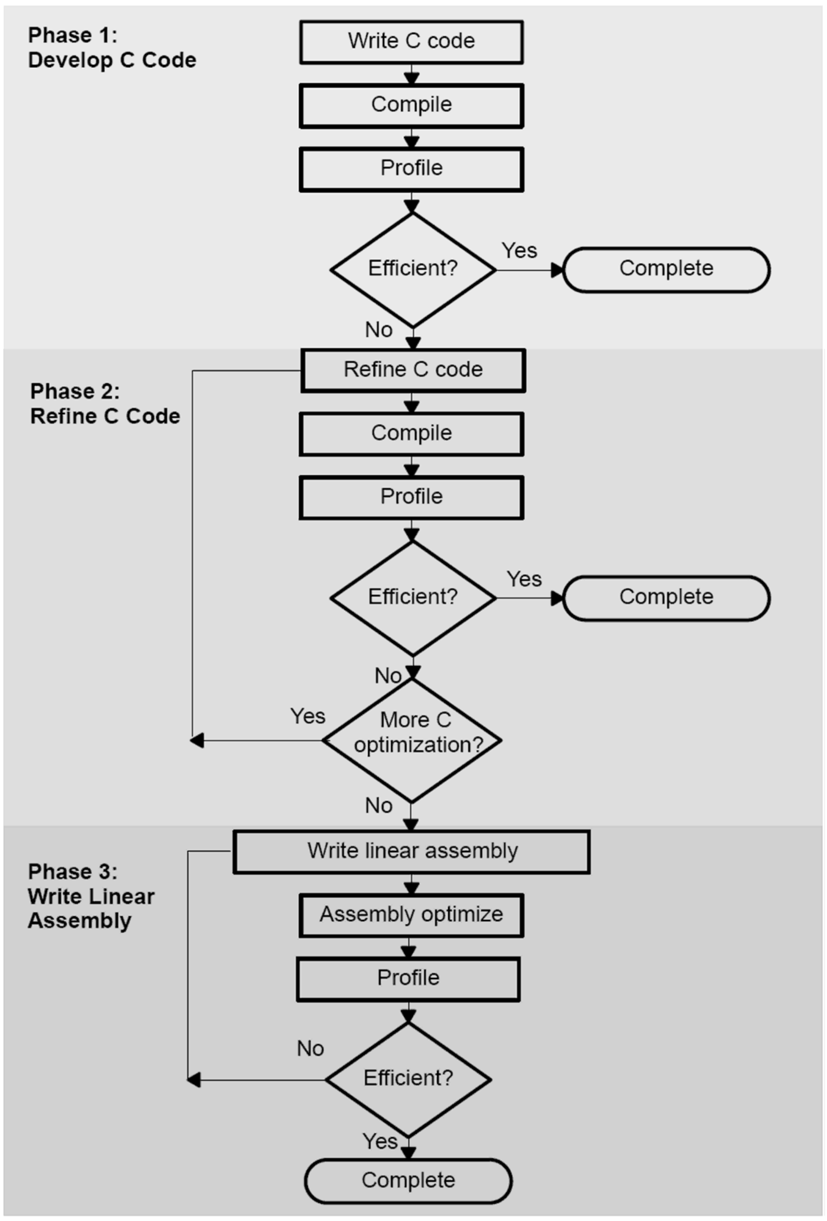


Figure 3.9: Code development flow for TI C6000 DSP (from [11]).

to track the number of CPU clock cycles consumed by a particular section of code. Use View Statistics to view the number of cycles consumed.

Usually, we use the second technique above to analyze the C code performance. The feedback of the optimization result can be obtained with the -mw option. It shows some important results of the assembly optimizer for each code section. We take these results into consideration in improving the computational speed of certain loops in our program.

### 3.3.1 Compiler Optimization Options [11]

In this subsection, we introduce the compiler options that control the operation of the compiler. The CCS compiler offers high-level language support by transforming C/C++ code into more efficient assembly language source code. The compiler options can be used to optimize the code size or the executing performance.

Some compiler options that are more important in our code development are:

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

  - Removes unused assignments.
  - Eliminates local common expressions.

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

- -o3. Performs all -o2 optimizations, and:
  - Removes all functions that are never called.
  - Simplifies functions with return values that are never used.
  - Inline calls to small functions.
  - Reorders function declarations so that the attributes of called functions are known when the caller is optimized.
  - Propagates arguments into function bodies when all calls pass the same value in the same argument position.
  - Identifies file-level variable characteristics.

- -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.

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

### 3.3.2 Software Pipelining [12]

Software pipelining is a technique used to schedule instructions from a loop so that multiple iterations of the loop can be executed in parallel. Figure 3.10 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 is known as the pipelined loop prolog, and the area below the kernel is known as the pipelined loop epilog.

But under the conditions listed below, the compiler will not do software pipelining:

- If a register value lives too long, the code is not software-pipelined.

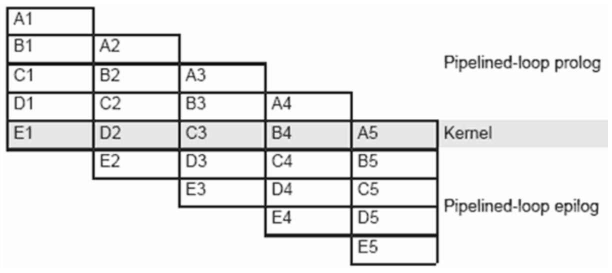


Figure 3.10: Software-pipelined loop (from [13]).

- If a loop has complex condition code within the body that requires more than five condition registers, the loop is not software pipelined.
- A software-pipelined loop cannot contain function calls, including code that calls the run-time support routines.
- In a sequence of nested loops, the innermost loop is the only one that can be software-pipelined.
- If a loop contains conditional break, it is not software-pipelined.

### 3.3.3 Loop Unrolling

Loop unwinding can be used to make programs more suitable for parallel processing. The idea of loop unrolling is to save time by reducing the number of overhead instructions that the computer has to execute in a loop, thus improving the cache hit rate and reducing branching. To achieve this, the instructions that are called in multiple iterations of the loop are combined into a single iteration.

### 3.3.4 Other Acceleration Rules

Other code acceleration rules include reducing memory access, using bit shifts for multiplication or division, declaring constants as constants that are not variable, accessing the memory sequentially, and minimizing use of conditional breaks or complex condition codes in loops.

# Chapter 4

# Ranging Technique for IEEE 802.16e OFDMA

In OFDMA, the active subcarriers are divided into subsets of subcarriers termed subchannels which are assigned to multiple users for simultaneous transmissions. The subcarriers of each subchannel may not necessarily be adjacent. To maintain the orthogonality among the subcarriers in the uplink of OFDMA systems, the signals from all active users should arrive at the BS synchronously [14]. This is accomplished by an initial uplink synchronization called a ranging process.

Ranging process includes initial ranging, periodic ranging, bandwidth request and handover-ranging. However, we mainly discuss initial ranging in this chapter.

## 4.1 OFDMA Ranging

The contents of this section have been taken to a large extent from [2] and [3].

A ranging channel is composed of one or more groups of six adjacent subchannels, where the groups are defined starting from the first subchannel. Optionally, ranging channel can be composed of one or more groups of eight adjacent subchannels. We choose the former in our study.

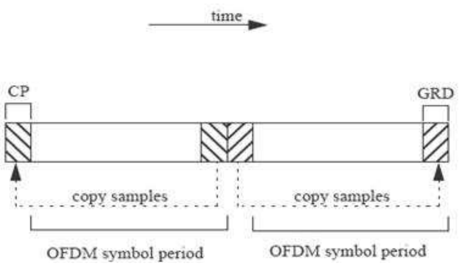


Figure 4.1: Initial-ranging/handover-ranging transmission for OFDMA (from [3]).

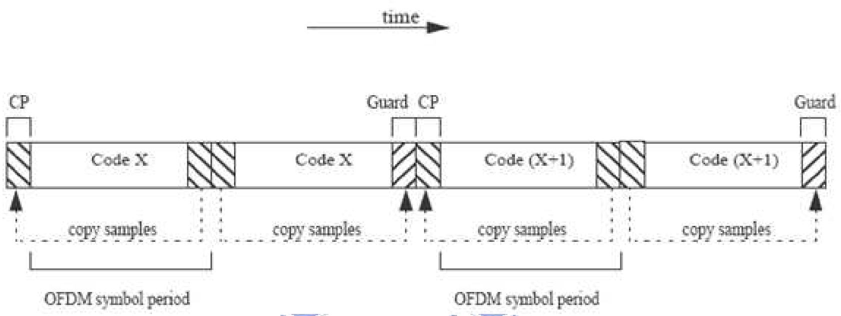


Figure 4.2: Initial-ranging handover-ranging transmission for OFDMA, using two consecutive initial ranging codes (from [3]).

Subchannels are considered adjacent if they have successive logical subchannel numbers. The indices of the subchannels that compose the ranging channel are specified in the UL-MAP message. Users are allowed to collide on this ranging channel. To effect a ranging transmission, each user randomly chooses one ranging code from a bank of specified binary codes. These codes are then BPSK modulated onto the subcarriers in the ranging channel, one bit per subcarrier.

### 4.1.1 Initial-Ranging/Handover-Ranging Transmissions [2], [3]

The initial ranging codes are used for any MS that wants to synchronize to the system for the first time. Handover ranging codes are used for ranging against a target BS during handover.

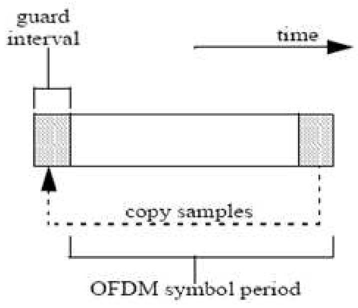


Figure 4.3: Periodic-ranging or bandwidth-request transmission for OFDMA using one code (from [3]).

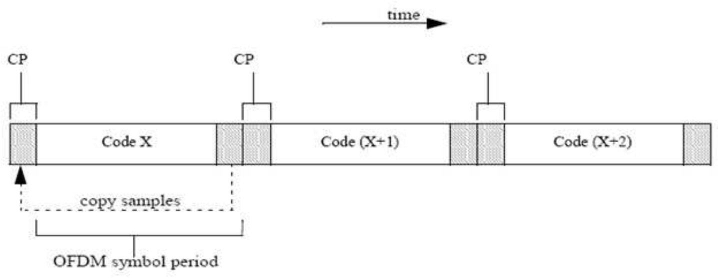


Figure 4.4: Periodic-ranging or bandwidth-request transmission for OFDMA using three consecutive codes (from [3]).

An initial-ranging transmission is performed using two or four consecutive symbols as shown in Figures 4.1 and 4.2, respectively. The same ranging code is transmitted on the ranging channel during each symbol, with no phase discontinuity between the two symbols. In the latter case (Figure 4.2), the BS can allocate two consecutive initial-ranging/handover-ranging slots, onto which the MS transmits the two consecutive initial-ranging/handover-ranging codes (the starting code shall always be a multiple of 2). We choose the former in our study.

### 4.1.2 Periodic-Ranging and Bandwidth-Request Transmissions [2], [3]

Periodic ranging transmissions are sent periodically for system periodic ranging. Bandwidth-requests transmissions are for requesting uplink allocations from the BS.

These transmissions shall be sent only by MS that have already synchronized to the system. To perform either a periodic-ranging or bandwidth-request transmission, the MS can send a transmission in one of the following ways:

- Modulate one ranging code on the ranging subchannel for a period of one OFDMA symbol. Ranging subchannels are dynamically allocated by the MAC layer and indicated in the UL-MAP.
- Modulating three consecutive ranging codes (starting code shall always be a multiple of 3) on the ranging subchannel for a period of three OFDMA symbols (one code per symbol). Ranging subchannels are dynamically allocated by the MAC layer and indicated in the UL-MAP.

  A time domain illustration of the former and latter are shown in Figure 4.3 and Figure 4.4.

### 4.1.3 Ranging Codes [2], [3]

The binary codes are the pseudonoise (PN) codes produced by the PRBS generator described in Figure 4.5, which implements the polynomial generator $1 + x^1 + x^4 + x^7 + x^{15}$. The PRBS generator is initialized by the seed b14..b0 = 0,0,1,0,1,0,1,1,s0,s1,s2,s3,s4,s5,s6 where s6 is the LSB of the PRBS seed, and s6:s0 = UL_PermBase, where s6 is the MSB of UL_PermBase.

The length of the ranging codes is 144 bits, which are subsequences of the pseudonoise sequence appearing at its output ($C_k$). These bits are used to modulate the subcarriers in

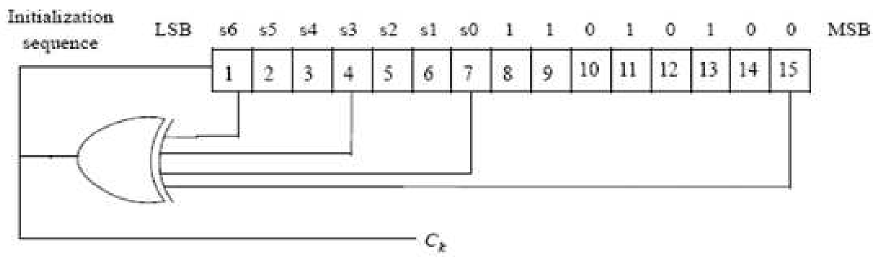


Figure 4.5: PRBS generator for ranging code generation (from [3]).

a group of six adjacent (logocally) subchannels. The bits are mapped to the subcarriers in increasing frequency order of the subcarriers, such that the lowest indexed bit modulates the subcarrier with the lowest frequency index and the highest indexed bit modulates the subcarrier with the highest frequency index. The index of the lowest numbered subchannel in the six shall be an integer multiple of six. The six subchannels are called a ranging subchannel.

The first 144-bit code obtained by clocking the PN generator as specified become the first ranging code. The next ranging code is produced by taking the output of the 145th to 288th clock instances of the PRBS generator, etc. The number of available codes is 256, numbered 0..255. Each BS uses a subgroup of these codes, where the subgroup is defined by a number $S$, $0 \leq S \leq 255$. The group of codes will be between $S$ and $((S+O+N+M+L)$ mod 256).

- The first $N$ codes produced are for initial ranging. Clock the PRBS generator $144 \times (S \bmod 256)$ times to $144 \times ((S + N) \bmod 256) - 1$ times.

- The next $M$ codes produced are for periodic ranging. Clock the PRBS generator $144 \times ((N + S) \bmod 256)$ times to $144 \times ((N + M + S) \bmod 256) - 1$ times.

- The next $L$ codes produced are for bandwidth requests. Clock the PRBS generator

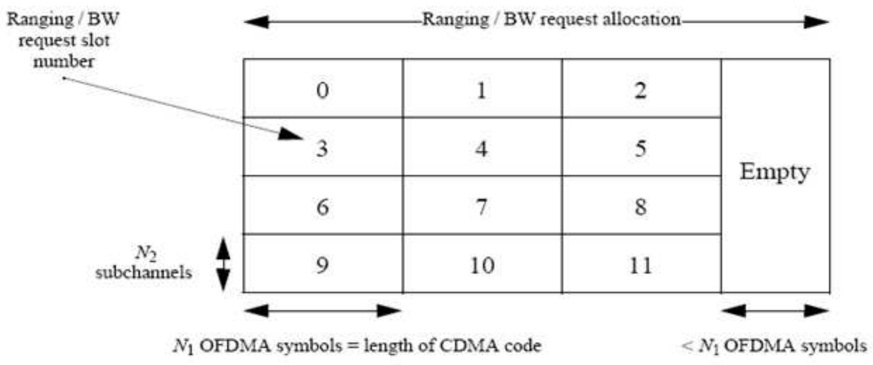

Ranging / BW request slot number
Ranging / BW request allocation
0
1
2
3
4
5
6
7
8
9
10
11
Empty
$N_2$ subchannels
$N_1$ OFDMA symbols = length of CDMA code
< $N_1$ OFDMA symbols

Figure 4.6: Ranging/BW request opportunities (from [3]).

$144 \times ((N + M + S) \bmod 256)$ times to $144 \times ((N + M + L + S) \bmod 256) - 1$ times.

- The next $O$ codes produced are for handover ranging. Clock the PRBS generator $144 \times ((N + M + L + S) \bmod 256)$ times to $144 \times ((N + M + L + O + S) \bmod 256) - 1$ times.

### 4.1.4 Ranging and BW Request Opportunity Size [3]

For CDMA ranging and BW request, the ranging opportunity size is the number of symbols required to transmit the appropriate ranging/BW request code (1, 2, 3 or 4 symbols), and is denoted $N_1$. $N_2$ denotes the number of subchannels required to transmit a ranging code. In each ranging/BW request allocation, the opportunity size ($N_1$) is fixed and conveyed by the corresponding UL_MAP_IE that defines the allocation. As shown in Figure 4.6, the ranging allocation is subdivided into slots of $N_1$ OFDMA symbols by $N_2$ subchannels, in a time first order, i.e., the first opportunity begins on the first symbol of the first subchannel of the ranging allocation, the next opportunities appear in ascending time order in the same subchannel, until the end of the ranging/BW request allocation (or until there are less than $N_1$ symbols in the current subchannel), and then the number of subchannel is incremented by $N_2$. The ranging allocation is not required to be a whole multiple of $N_1$ symbols, so a gap may be formed (that can be used to mitigate interference between ranging and data transmissions). Each CDMA code will be transmitted at the beginning of the corresponding slot.

## 4.2 Initial Ranging Process and Task [2], [3]

This section mainly introduces initial ranging process and the task needing to be done at the receiver side.

Once a MS senses a BS, for network entry it first scans for a DL channel and synchronizes itself with the BS. It learns the uplink channel characteristics through the Uplink Channel Descriptor (UCD) MAC management message. At this point, the MS shall scan the UL-MAP message to find an initial ranging interval and acquire transmit parameters.

At the transmitter side, when MSs perform initial ranging, they shall transmit a randomly chosen frequency domain ranging code on the ranging channel in a randomly chosen ranging time slot.

Since more than one MSs may choose the same time slot, the received ranging signal may include several MSs' ranging information. So, at the receiver side, the BS is required to detect different received ranging codes, estimate the timing, frequency offset and the power level of each user that transmits an initial ranging code.

The BS then broadcasts the ranging response which includes the detected ranging codes and ranging slots with adjustment instructions for the timing, frequency and power level. This information is used by the MS that sent the CDMA ranging code to identify the ranging response message that corresponds to its ranging request. The status notifications of either ranging successful or retransmission are also broadcasted.

According to the ranging response, the MS knows what action should be taken. More details about the ranging response are specified in [3].

Ranging code detection and the estimation of its timing offset and frequency offset are some of the main subjects of our work.

## 4.3 Initial Ranging Algorithm and Simulation Results

In this section, we present our ranging algorithm and the simulation results. As mentioned before, the tasks in ranging are:

- Detection of ranging code,
- Estimation of timing offset,and
- Estimation of frequency offset.

### 4.3.1 Initial Ranging Algorithm

Figure 4.7 depicts the overall structure of the ranging receiver system. Ranging code's detection and timing offset estimation will complete at the same time. If we determine that the received signal contains a ranging signal, then we will estimate the frequency offset of the received ranging signal. When the detected signal is too weak, we will drop the received signal and will not estimate the frequency offset.

In the figure, the FFT processor processes the time domain data from the receiver filter and outputs frequency domain data. The ranging subcarrier selector extracts subcarrier values on which a ranging code may be loaded. As shown in Figure 4.1, since the ranging signal

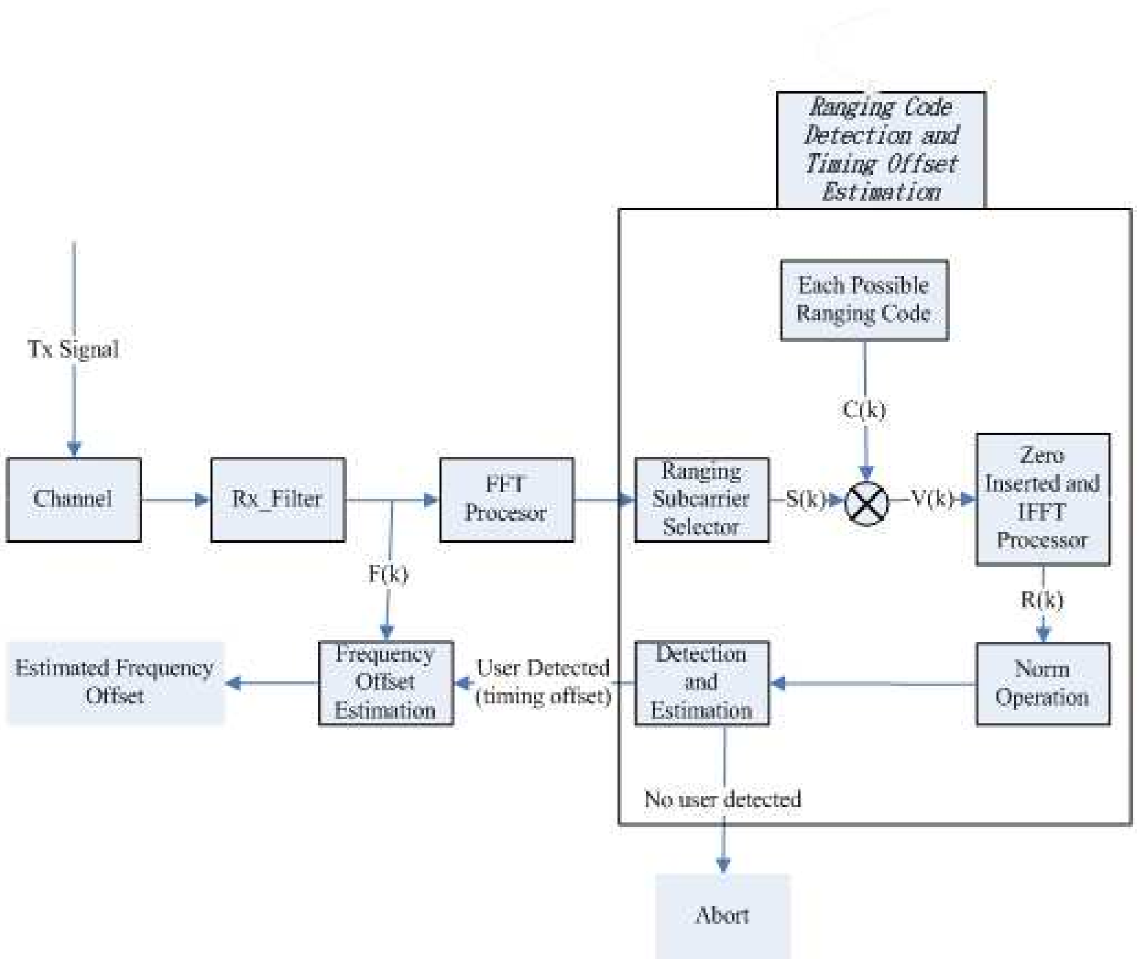

Ranging Code Detection and Timing Offset Estimation
Tx Signal
Channel
Rx_Filter
FFT Procesor
Each Possible Ranging Code
C(k)
Ranging Subcarrier Selector
S(k)
V(k)
Zero Inserted and IFFT Processor
R(k)
F(k)
Norm Operation
Detection and Estimation
User Detected (timing offset)
Frequency Offset Estimation
Estimated Frequency Offset
No user detected
Abort

Figure 4.7: Ranging signal receiver.

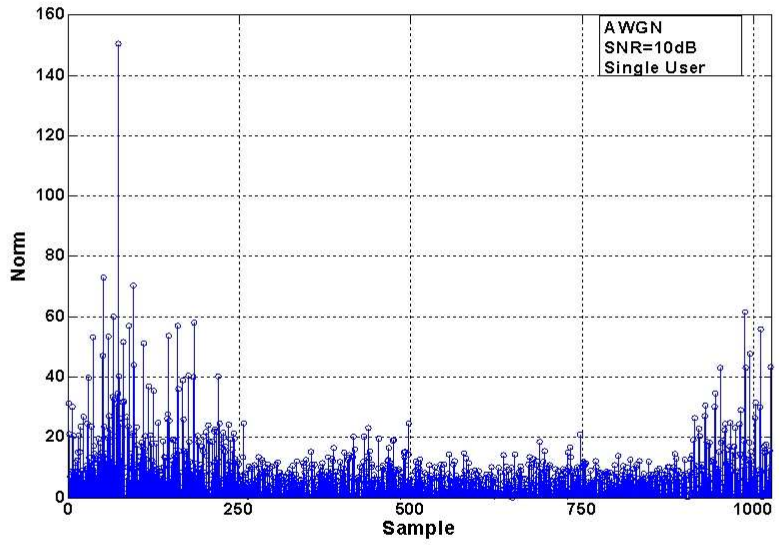


Figure 4.8: Ranging simulation with detected ranging code.

spans two successive OFDMA symbols, the ranging subcarrier selector extracts subcarriers from the second OFDMA symbol of the FFT processor's output, too. Then the multiplier multiplies the subcarrier values received from the ranging subcarrier selector by the possible ranging codes.

Herein, the output of multiplier ($V(k)$) is composed of 144 samples. Zeros are inserted to the frequency positions and the IFFT processor processes the result and outputs $R(k)$ which is composed of 1024 complex samples. The total number of zeros inserted is 880, to make the OFDMA symbol containing zero in all channels beside the ranging channels. After IFFT, the signal becomes a time domain symbol. The next step is norm operation, which calculates the norm of $R(k)$. If the detected signal contains a ranging signal and multiplies by a correct ranging code, it will have a large value at the correct timing offset, as shown in Figure 4.8. Figure 4.9 shows a simulation result of received signal multiplies by an incorrect ranging code. More simulation results and analysis will be presented in the next subsection.

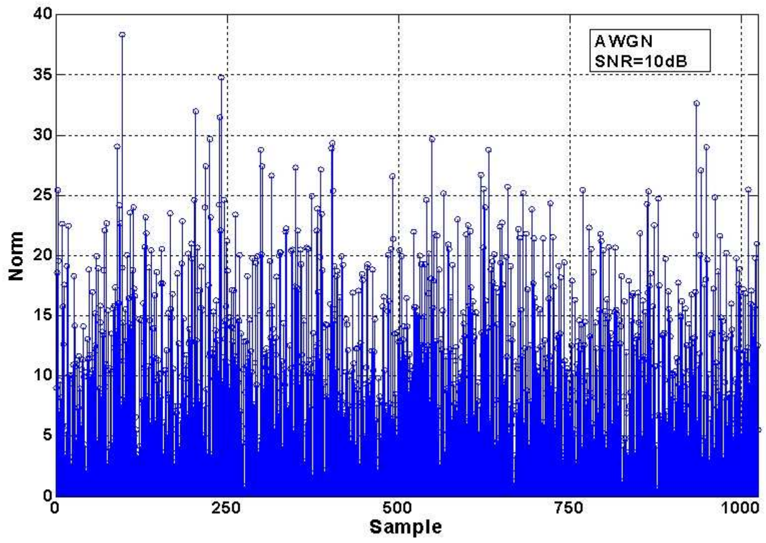


Figure 4.9: Ranging simulation without detected ranging code.

Ranging code detection and the estimation of its timing offset can also be accomplished in the time domain by using cross correlation, it needs $1024 \times 1024$ complex multiplications to complete that work. In our study, using IFFT to achieve same purposes needs $1024 \times log_2 1024$ complex multiplications, it is only 1% in complex multiplications of using cross correlation in the time domain.

As mentioned before, we estimate the frequency offset of the received ranging signal if a ranging signal is detected. As shown in Figure 4.7, the inputs to the frequency offset estimation block are $F(k)$ and the timing offset value. Figure 4.10 describes the block's function.

There are three successive OFDMA symbols in an uplink slot. If any uplink slot contains a ranging signal, then the ranging signal is loaded in this slot according to OFDMA symbol offset value specified by UL-MAP. $F(k)$ denotes the time period of three successive OFDMA symbols that contains at least one ranging code, the timing offset $Y$ corresponds to a detected

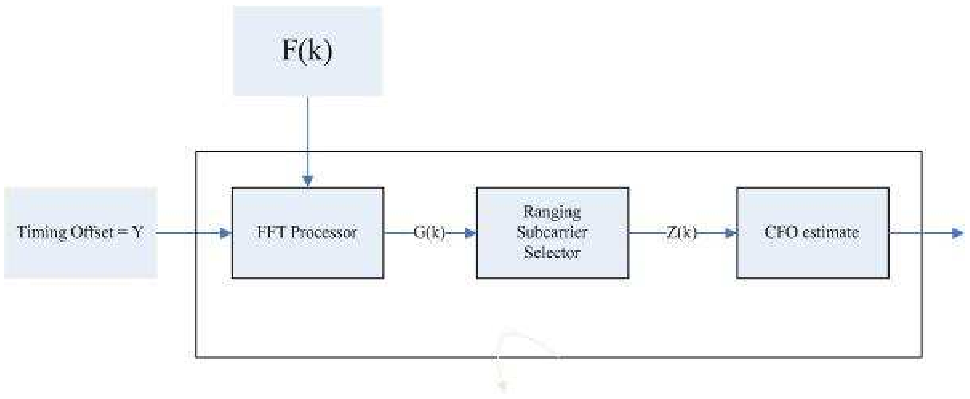


Figure 4.10: Frequency offset estimation method.

ranging code's timing offset. We pick out two successive OFDMA symbols from $F(k)$ starting time at $Y$, to form the input of the FFT processor.

The input to the ranging subcarrier selector $G(k)$ is obtained from the output of the FFT processor. As mentioned before, the ranging subcarrier selector extracts the ranging subcarriers from $G(k)$, and outputs $Z(k)$ which are composed of two successive OFDMA symbols, ranging subcarriers.

The last step of Figure 4.10 implements the equation below [16]:

$$\hat{\triangle f} = \frac{\angle(\sum_{k=0}^{N_r-1} Z^*(k) \times Z(k+N_r))}{2\pi} \tag{4.1}$$

where $N_r = 144$ and $Z(k)$ is the output of the ranging subcarrier selector, which consists of 288 complex values.

From (4.1), the detectable range of frequency offset values is $(-0.5, 0.5)$ (see Figure 4.11), which will cover the range of expected frequency offsets in initial ranging.

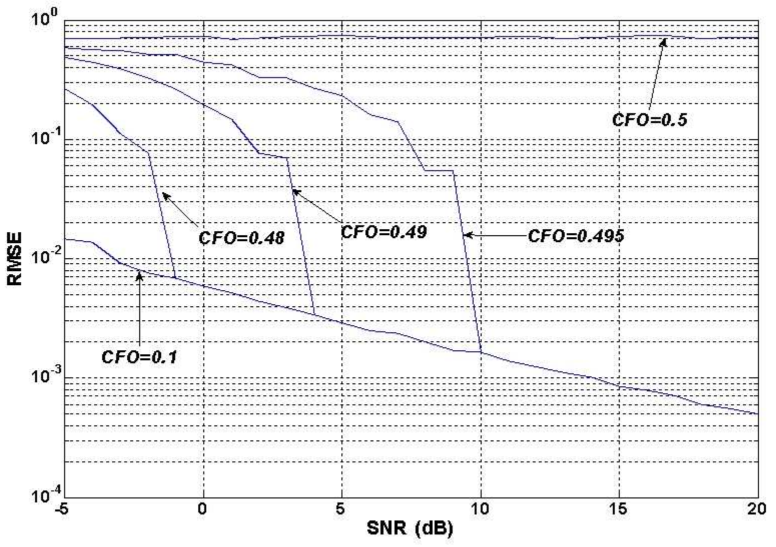


Figure 4.11: CFO simulation in AWGN.

### 4.3.2 Simulation Results

In this subsection, we use last subsection's algorithm and present some simulation results about timing offset and frequency offset's estimation and detection of ranging codes.

Figure 4.12 shows a simulation result of ranging code detection and timing offset estimation with ranging channel containing one single user and five users, respectively. It is obvious that more ranging users causes more interference. In AWGN channel, we can set a threshold to decide whether the ranging channel contains ranging code or not. For example, if we choose 100 as threshold, when the calculated norm has a value greater than 100, then we say that the received signal contains ranging code, and the location of this value is the estimated timing offset. This is a simple method, but its performance is bad in multipath channels.

We can use another way to decide whether the received OFDMA symbol contains a

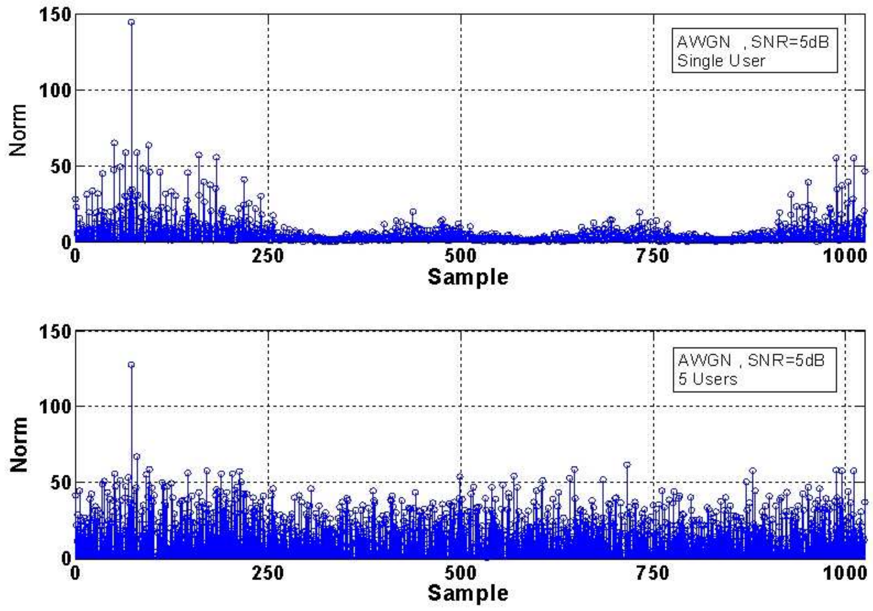


Figure 4.12: Ranging code detection (1 user vs. 5 users).

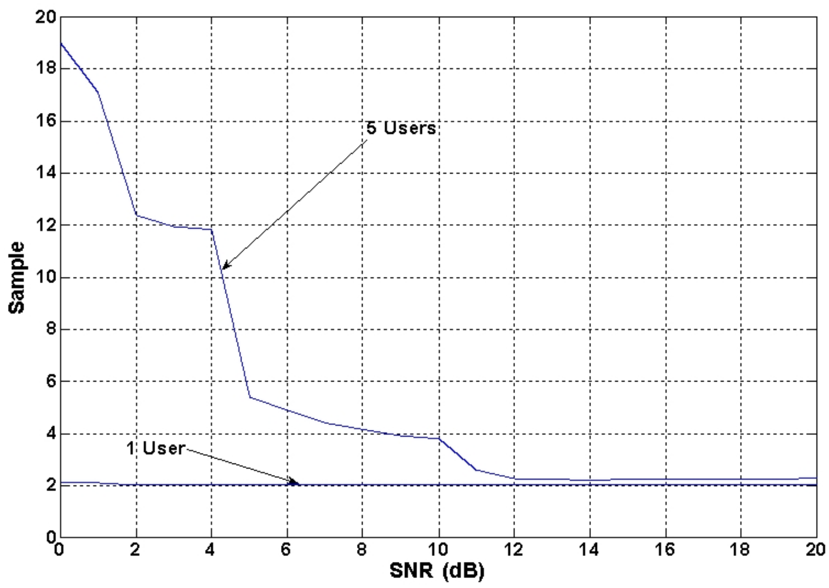


Figure 4.13: RMSE of timing offset estimation.

ranging code. Choose three thresholds $H$, $H_1$, $H_2$, where $H_1$ is greater than $H_2$. We calculate the ratio of the mean of norm value greater than $H_1$ ($h_1$) to the mean of norm value smaller than $H_2$ ($h_2$). If this ratio ($h_1/h_2$) is greater than $H$, we say that the received OFDMA symbol contains ranging code. The choice of $H$, $H_1$ and $H_2$ determines the tradeoff between false alarm and miss detection. To find the timing offset, let the maximum value of the norm be divided by a number ($h_t$), then we get a new threshold. We choose the first location of norm greater than this threshold as the estimated timing offset [15].

We use the method mentioned in the last paragraph to run a simulation under the SUI-3 channel and show the result in Figure 4.13. The values of $H$, $H_1$, and $H_2$ in our simulation are 85.333, 21.333, 11.022, respectively. We calculate the RMSE of timing offset estimation based on the first path for one single user and five users, respectively. The determination of optimum parameters are not easy work, especially for different numbers of users under multipath propagation. This work is left to potential future work.

In our system, frequency offset value is within the range of $[-0.1, 0.1]$, where the speed is 120 km/hr and the central frequency is 3.5 GHz. Figure 4.14 shows the simulation result of received signal containing five ranging codes with CFO = 0.1 under AWGN channel. Compared with the bottom figure of Figure 4.12, Figure 4.14 has no evident difference due to effect of CFO.

Figure 4.15 shows the failure rate of ranging code detection. Single user and three users in ranging channel under SUI-3 channel are shown in top and bottom of figure, respectively. The effect of CFO decreases the success rate of ranging code detection by about 1% for single user and about 3% for three users in ranging channel.

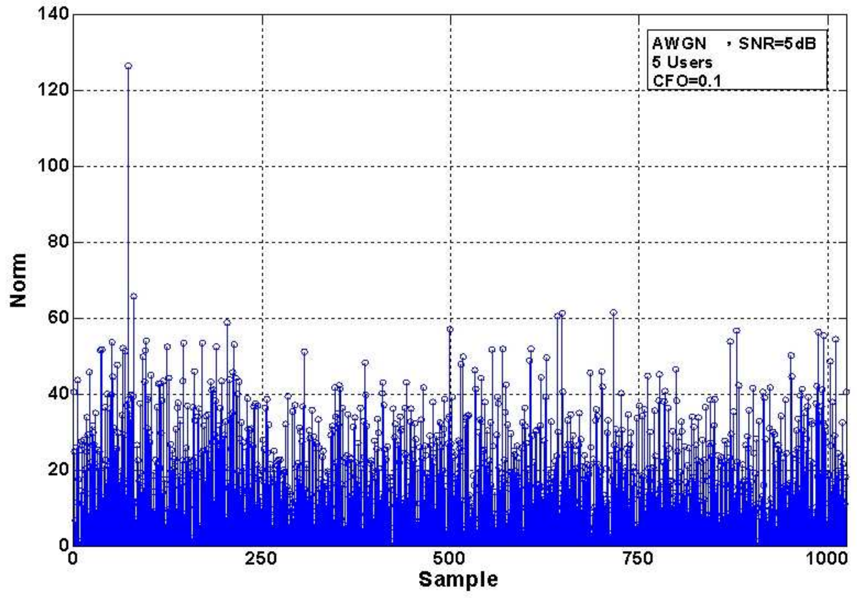


Figure 4.14: Ranging code detection with CFO in AWGN.

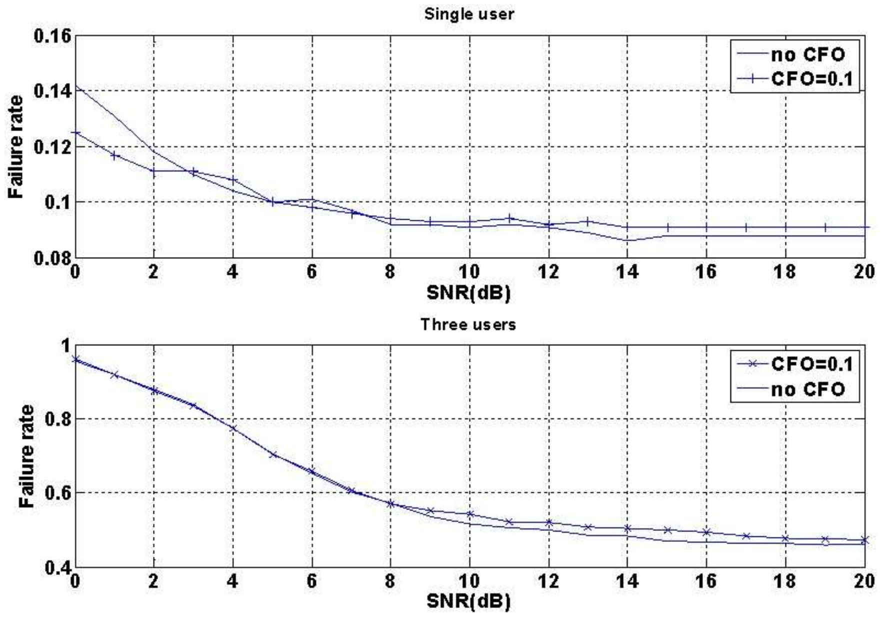


Figure 4.15: Failure rates of ranging code detection in SUI-3.

# Chapter 5

# Integration of IEEE 802.16e OFDMA TDD Uplink Transceiver System on DSP Platform with RTOS

In this chapter, we first introduce the software platform used in our integration system. Then we discuss how the uplink transceiver system is implemented on DSP platform with a real-time operating system (RTOS).

## 5.1 Introduction to 3L Diamond

Diamond is 3L's system for multiprocessor software design and implementation. Diamond uses the communicating sequential processes (CSP) model to give a simple but powerful way of developing applications that make use of one or more processors [17]. The 3L Ltd has been working closely with Sundance, aiming to provide simple-to-use, reliable and flexible development environment for the Sundance hardware.

The way to build and run applications using Diamond differs substantially from the more traditional techniques used in other environments, particularly the Code Composer Studio (CCS). CCS has been designed to produce applications for single processor systems; multiprocessor systems are seen as several separate applications that just happen to be executed at the same time. Diamond takes the opposite view and considers multiprocessor systems as an integrated whole [18].

Diamond uses a three-stage approach to building an application:

1. Compile source files with the Texas Instruments (TI) compiler;
2. Use the TI linker, usually several times, to generate a number of separate task files;
3. Use the configurer to combine task files into a single application file that contains everything needed to get application running on a network of DSPs. Allocation of memory is done automatically by the configurer.

Once a Diamond application has been constructed, it is loaded into DSPs and executed by a server program running on the PC.

Figures 5.1(a) and (b) show the building process with CCS and Diamond respectively. As mentioned before, CCS is really a single-processor system; it treats each processor separately. The processor types, memory layout, I/O devices being used and connections between processors are needed when we build the application under CCS. We have to completely control everything, which may take much time and experience, and it is also hard to make significant changes [19].

Compared with CCS, Diamond is designed for multiprocessor systems and does a lot of the work for us due to an extra configuration step.

As shown in Figure 5.1(b), each individual task is built by compiling all its source files with the C compiler and using the linker to combine the resulting object files with the necessary modules from the run-time library. Repeating this for every task in the application results in a number of task image files. The program called configurer combines all task image files to form a single executable application file. A user-supplied textual configuration file

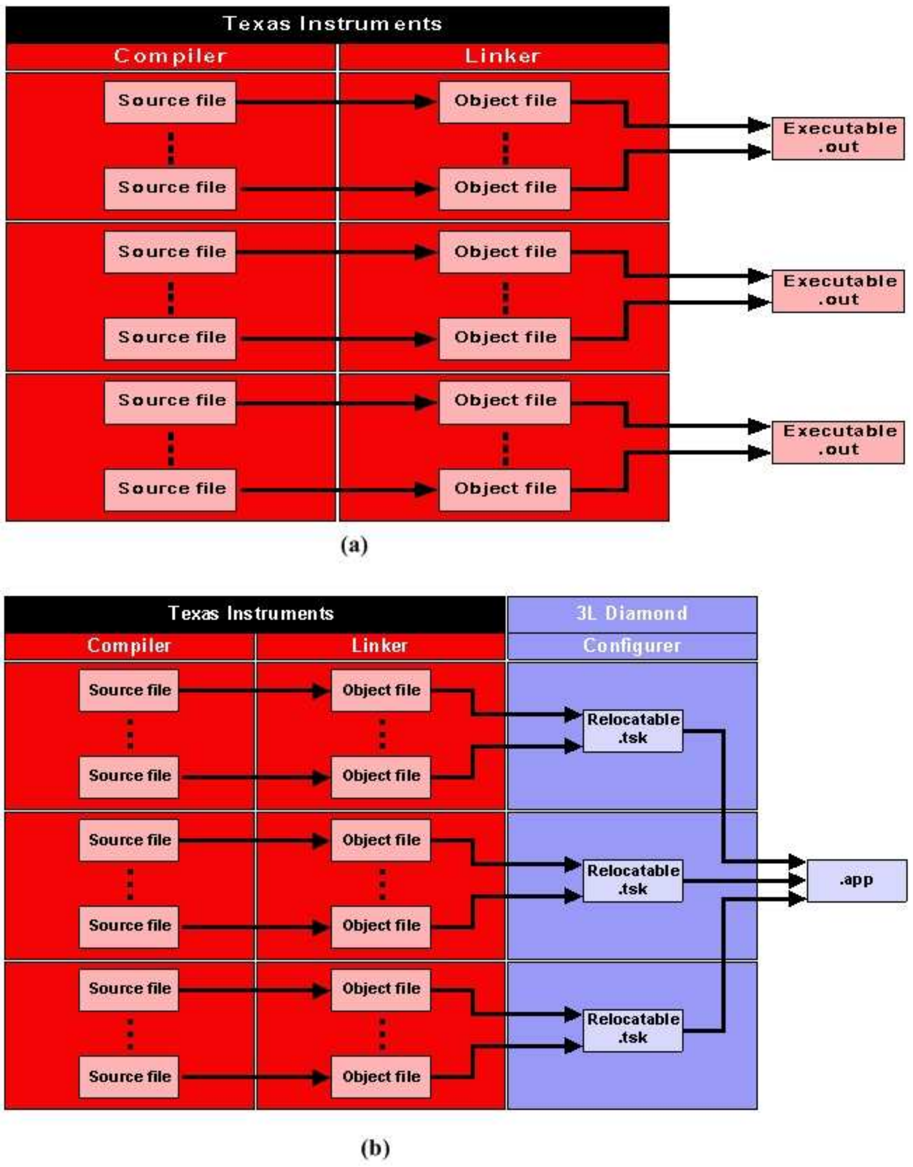


Figure 5.1: Building with CCS and Diamond (from [19]). (a) Building with CCS. (b) Building with Diamond.

drives the configurer and specifies:

1. Hardware structure: available processors and link connections between them.

2. Software structure: task to be included and channel connections between them.

3. How to map the software onto the hardware.

The configurer allocates memory for the tasks and combines them into a single application file that can be loaded into the specified hardware network and executed using the Diamond command, 3L x. The configurer is also responsible for determining which system tasks need to be loaded. To change the way in which tasks are connected together or the processors on which they are to run, it is not necessary either to change the source code or to recompile or re-link the tasks themselves. This means, it is possible to develop an application while running all the tasks on one processor, and then reconfigure it, without any other change, to run on a network. Physical channels may be transparently substituted for virtual ones in a similar way.

We use a simple Diamond model to explain some terminology of Diamond in Figure 5.2.

The hardware on which a Diamond application runs is described as a network of processors, each with a number of links connected by pairs by wires. Each wire is a two-way communication path between two processors. As shown in Figure 5.2(a), processor P1 has two links: link 0 connects to processor P2 using wire W1, and link 1 connects to processor P3 using wire W2. As long as we can connect the processors using supported devices, Diamond can create applications for systems of any complexity and any intermixture of hardware types.

The starting point in every Diamond application is a processor called root. The root acts as the base of the network and a reference for locating all other processors (Figure 5.2(b)).

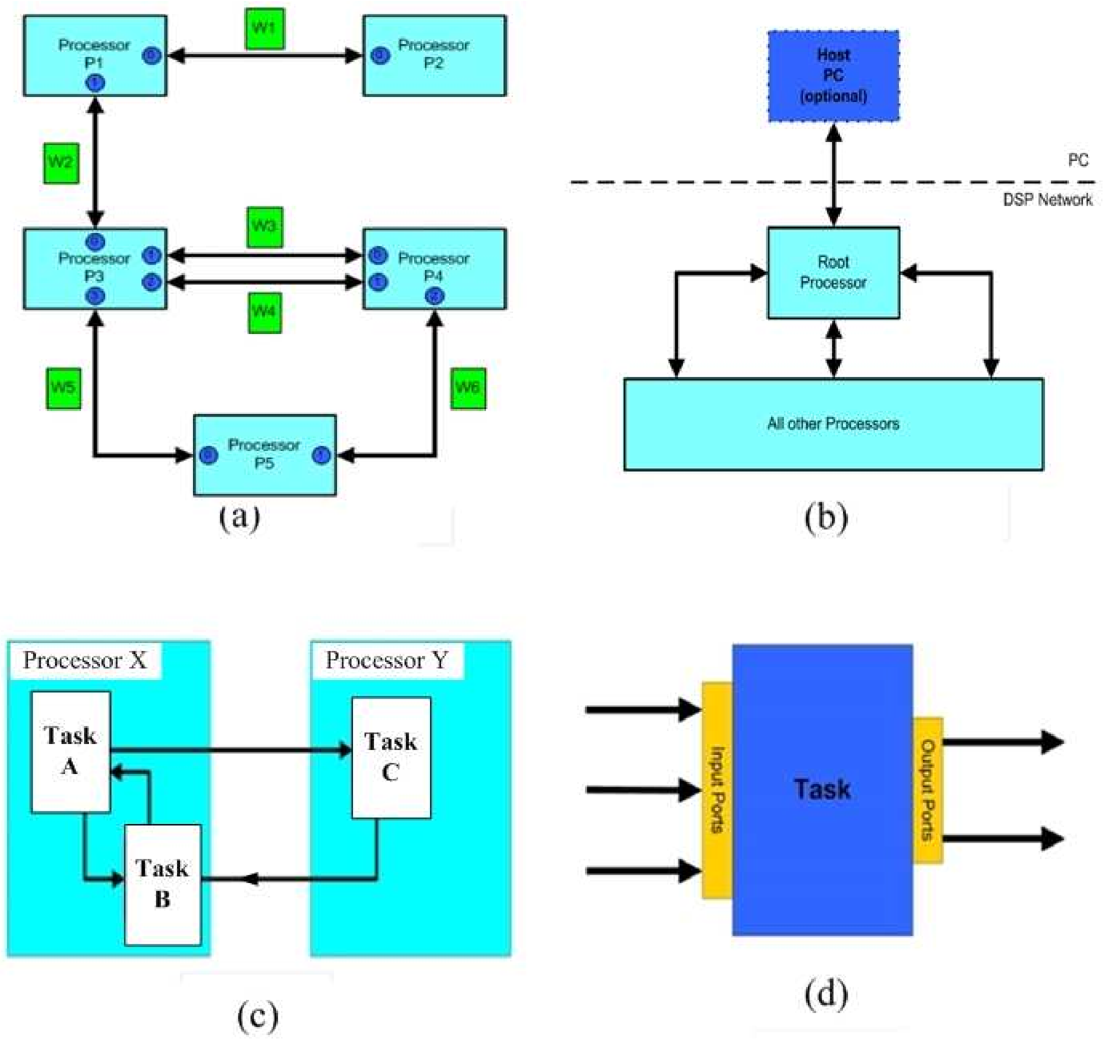


Figure 5.2: Diamond model (from [18]). (a) Processors, wire, and link. (b) Host and processor. (c) Task. (d) Ports.

A complete application is a collection of one or more concurrently executing tasks connected by channels (Figure 5.2(c)). A channel can transfer messages from one process to one other process. A channel can only carry messages in one direction: if communication in both directions between two processes is required, two channels must be used. Diamond tasks are relocatable, that is, they do not contain fixed memory addresses, memory allocation is done at a configurer stage.

Each task has a vector of input ports and a vector of output ports that are used to connect tasks together. A task is like a software black box, communicating with the outside only via its ports, as shown in Figure 5.2(d). We join tasks by connecting output ports to input ports using channels, and this collection of tasks is combined into a single application file by a utility called the configurer [18].

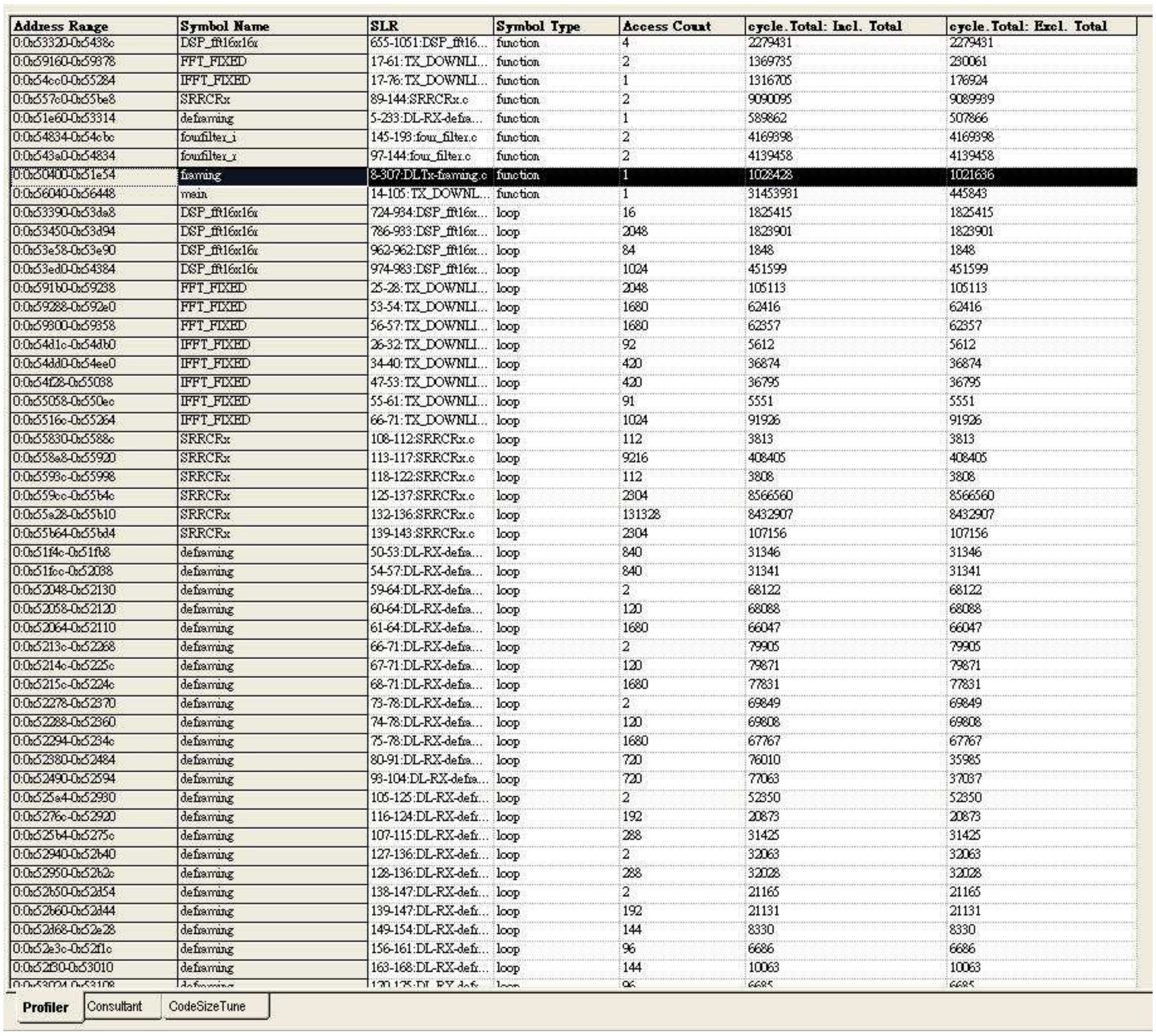
## 5.2 Integration Work

The work of integration can be divided into two parts. The first part is to develop the function of each module and optimize them. The second part is final integration on DSP platform, in which we decide the loading of each processor and connect them with physical link.

### 5.2.1 Optimization

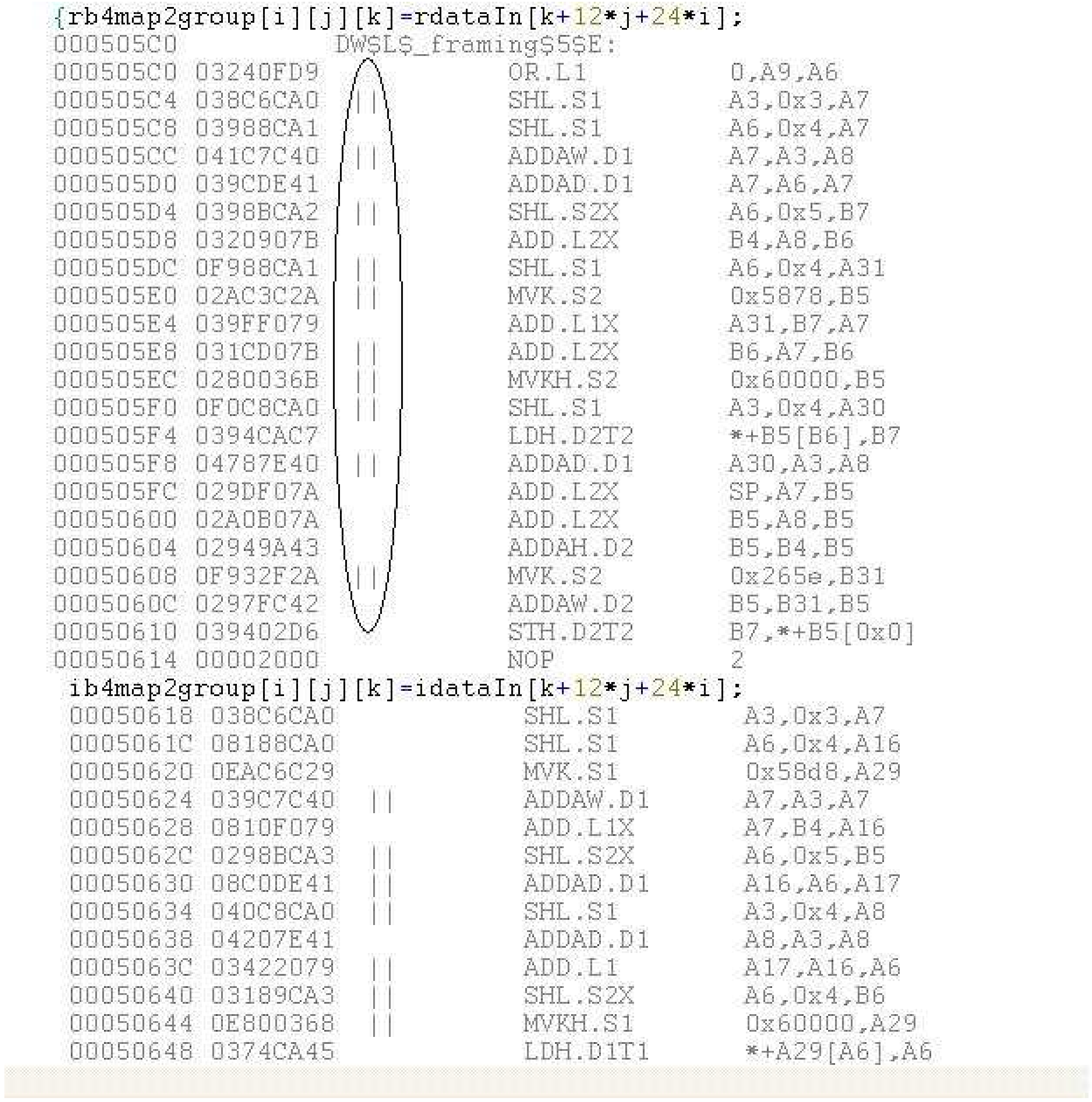
In the first part, we use CCS to help the optimization. After developing the module, we utilize CCS's profile function to analyze the profile data of each module. Figure 5.3 shows the consumption of cycles for each module. Our concern is about the values in the last column. For example, the framing function needs 1,021,636 cycles for each run.

Aiming at the functions or loops which need a large number of cycles, then, we optimize



| Address Range | Symbol Name | SLR | Symbol Type | Access Count | cycle. Total: Incl. Total | cycle. Total: Excl. Total |
|---|---|---|---|---|---|---|
| 0:0x53320-0x5438c | DSP_fft16x16r | 655-1051:DSP_fft16... | function | 4 | 2279431 | 2279431 |
| 0:0x59160-0x59378 | FFT_FIXED | 17-61:TX_DOWNLI... | function | 2 | 1369735 | 230061 |
| 0:0x54cc0-0x55284 | IFFT_FIXED | 17-76:TX_DOWNLI... | function | 1 | 1316705 | 176924 |
| 0:0x557c0-0x55be8 | SRRCRx | 89-144:SRRCRx.c | function | 2 | 9090095 | 9089939 |
| 0:0x51e60-0x53314 | deframing | 5-233:DL-RX-defra... | function | 1 | 589862 | 507866 |
| 0:0x54834-0x54cbc | fourfilter_i | 145-193:four_filter.c | function | 2 | 4169398 | 4169398 |
| 0:0x543a0-0x54834 | fourfilter_r | 97-144:four_filter.c | function | 2 | 4139458 | 4139458 |
| 0:0x50400-0x51e54 | framing | 8-307:DLTx-framing.c | function | 1 | 1028428 | 1021636 |
| 0:0x56040-0x56448 | main | 14-105:TX_DOWNL... | function | 1 | 31453931 | 445843 |
| 0:0x53390-0x53da8 | DSP_fft16x16r | 724-934:DSP_fft16x... | loop | 16 | 1825415 | 1825415 |
| 0:0x53450-0x53d94 | DSP_fft16x16r | 786-933:DSP_fft16x... | loop | 2048 | 1823901 | 1823901 |
| 0:0x53e58-0x53e90 | DSP_fft16x16r | 962-962:DSP_fft16x... | loop | 84 | 1848 | 1848 |
| 0:0x53ed0-0x54384 | DSP_fft16x16r | 974-983:DSP_fft16x... | loop | 1024 | 451599 | 451599 |
| 0:0x591b0-0x59238 | FFT_FIXED | 25-28:TX_DOWNLI... | loop | 2048 | 105113 | 105113 |
| 0:0x59288-0x592e0 | FFT_FIXED | 53-54:TX_DOWNLI... | loop | 1680 | 62416 | 62416 |
| 0:0x59300-0x59358 | FFT_FIXED | 56-57:TX_DOWNLI... | loop | 1680 | 62357 | 62357 |
| 0:0x54d1c-0x54db0 | IFFT_FIXED | 26-32:TX_DOWNLI... | loop | 92 | 5612 | 5612 |
| 0:0x54dd0-0x54ee0 | IFFT_FIXED | 34-40:TX_DOWNLI... | loop | 420 | 36874 | 36874 |
| 0:0x54f28-0x55038 | IFFT_FIXED | 47-53:TX_DOWNLI... | loop | 420 | 36795 | 36795 |
| 0:0x55058-0x550ec | IFFT_FIXED | 55-61:TX_DOWNLI... | loop | 91 | 5551 | 5551 |
| 0:0x5516c-0x55264 | IFFT_FIXED | 66-71:TX_DOWNLI... | loop | 1024 | 91926 | 91926 |
| 0:0x55830-0x5588c | SRRCRx | 108-112:SRRCRx.c | loop | 112 | 3813 | 3813 |
| 0:0x558a8-0x55920 | SRRCRx | 113-117:SRRCRx.c | loop | 9216 | 408405 | 408405 |
| 0:0x5593c-0x55998 | SRRCRx | 118-122:SRRCRx.c | loop | 112 | 3808 | 3808 |
| 0:0x559cc-0x55b4c | SRRCRx | 125-137:SRRCRx.c | loop | 2304 | 8566560 | 8566560 |
| 0:0x55a28-0x55b10 | SRRCRx | 132-136:SRRCRx.c | loop | 131328 | 8432907 | 8432907 |
| 0:0x55b64-0x55bd4 | SRRCRx | 139-143:SRRCRx.c | loop | 2304 | 107156 | 107156 |
| 0:0x51f4c-0x51fb8 | deframing | 50-53:DL-RX-defra... | loop | 840 | 31346 | 31346 |
| 0:0x51fcc-0x52038 | deframing | 54-57:DL-RX-defra... | loop | 840 | 31341 | 31341 |
| 0:0x52048-0x52130 | deframing | 59-64:DL-RX-defra... | loop | 2 | 68122 | 68122 |
| 0:0x52058-0x52120 | deframing | 60-64:DL-RX-defra... | loop | 120 | 68088 | 68088 |
| 0:0x52064-0x52110 | deframing | 61-64:DL-RX-defra... | loop | 1680 | 66047 | 66047 |
| 0:0x5213c-0x52268 | deframing | 66-71:DL-RX-defra... | loop | 2 | 79905 | 79905 |
| 0:0x5214c-0x5225c | deframing | 67-71:DL-RX-defra... | loop | 120 | 79871 | 79871 |
| 0:0x5215c-0x5224c | deframing | 68-71:DL-RX-defra... | loop | 1680 | 77831 | 77831 |
| 0:0x52278-0x52370 | deframing | 73-78:DL-RX-defra... | loop | 2 | 69849 | 69849 |
| 0:0x52288-0x52360 | deframing | 74-78:DL-RX-defra... | loop | 120 | 69808 | 69808 |
| 0:0x52294-0x5234c | deframing | 75-78:DL-RX-defra... | loop | 1680 | 67767 | 67767 |
| 0:0x52380-0x52484 | deframing | 80-91:DL-RX-defra... | loop | 720 | 76010 | 35985 |
| 0:0x52490-0x52594 | deframing | 93-104:DL-RX-defra... | loop | 720 | 77063 | 37037 |
| 0:0x525a4-0x52930 | deframing | 105-125:DL-RX-defr... | loop | 2 | 52350 | 52350 |
| 0:0x5276c-0x52920 | deframing | 116-124:DL-RX-defr... | loop | 192 | 20873 | 20873 |
| 0:0x525b4-0x5275c | deframing | 107-115:DL-RX-defr... | loop | 288 | 31425 | 31425 |
| 0:0x52940-0x52b40 | deframing | 127-136:DL-RX-defr... | loop | 2 | 32063 | 32063 |
| 0:0x52950-0x52b2c | deframing | 128-136:DL-RX-defr... | loop | 288 | 32028 | 32028 |
| 0:0x52b50-0x52d54 | deframing | 138-147:DL-RX-defr... | loop | 2 | 21165 | 21165 |
| 0:0x52b60-0x52d44 | deframing | 139-147:DL-RX-defr... | loop | 192 | 21131 | 21131 |
| 0:0x52d68-0x52e28 | deframing | 149-154:DL-RX-defr... | loop | 144 | 8330 | 8330 |
| 0:0x52e3c-0x52f1c | deframing | 156-161:DL-RX-defr... | loop | 96 | 6686 | 6686 |
| 0:0x52f30-0x53010 | deframing | 163-168:DL-RX-defr... | loop | 144 | 10063 | 10063 |
| 0:0x53024-0x53108 | deframing | 170-175:DL-RX-defr... | loop | 96 | 6685 | 6685 |

Figure 5.3: Profile data of each module.



```
{rb4map2group[i][j][k]=rdataIn[k+12*j+24*i];
000505C0           DW$L$_framing$5$E:
000505C0 03240FD9              OR.L1          0,A9,A6
000505C4 038C6CA0  ||          SHL.S1         A3,0x3,A7
000505C8 03988CA1              SHL.S1         A6,0x4,A7
000505CC 041C7C40  ||          ADDAW.D1       A7,A3,A8
000505D0 039CDE41              ADDAD.D1       A7,A6,A7
000505D4 0398BCA2  ||          SHL.S2X        A6,0x5,B7
000505D8 0320907B              ADD.L2X        B4,A8,B6
000505DC 0F988CA1  ||          SHL.S1         A6,0x4,A31
000505E0 02AC3C2A  ||          MVK.S2         0x5878,B5
000505E4 039FF079              ADD.L1X        A31,B7,A7
000505E8 031CD07B  ||          ADD.L2X        B6,A7,B6
000505EC 0280036B  ||          MVKH.S2        0x60000,B5
000505F0 0F0C8CA0  ||          SHL.S1         A3,0x4,A30
000505F4 0394CAC7              LDH.D2T2       *+B5[B6],B7
000505F8 04787E40  ||          ADDAD.D1       A30,A3,A8
000505FC 029DF07A              ADD.L2X        SP,A7,B5
00050600 02A0B07A              ADD.L2X        B5,A8,B5
00050604 02949A43              ADDAH.D2       B5,B4,B5
00050608 0F932F2A  ||          MVK.S2         0x265e,B31
0005060C 0297FC42              ADDAW.D2       B5,B31,B5
00050610 039402D6              STH.D2T2       B7,*+B5[0x0]
00050614 00002000              NOP            2
 ib4map2group[i][j][k]=idataIn[k+12*j+24*i];
 00050618 038C6CA0             SHL.S1         A3,0x3,A7
 0005061C 08188CA0             SHL.S1         A6,0x4,A16
 00050620 0EAC6C29             MVK.S1         0x58d8,A29
 00050624 039C7C40  ||         ADDAW.D1       A7,A3,A7
 00050628 0810F079             ADD.L1X        A7,B4,A16
 0005062C 0298BCA3  ||         SHL.S2X        A6,0x5,B5
 00050630 08C0DE41  ||         ADDAD.D1       A16,A6,A17
 00050634 040C8CA0  ||         SHL.S1         A3,0x4,A8
 00050638 04207E41             ADDAD.D1       A8,A3,A8
 0005063C 03422079  ||         ADD.L1         A17,A16,A6
 00050640 03189CA3  ||         SHL.S2X        A6,0x4,B6
 00050644 0E800368  ||         MVKH.S1        0x60000,A29
 00050648 0374CA45             LDH.D1T1       *+A29[A6],A6
```

Figure 5.4: Software pipeline information.

them and compare again the consumption of cycles. There are several methods that can accelerate our code and reduce the execution time. For example, we can utilize the TI C64x DSP library that includes many C-callable, assembly optimized, and general-purpose signal-processing routines to accelerate our implementation. We also make use of several useful compiler options supported by TI's compiler. As mentioned in chapter 3, the compiler options can be used to optimize code size or execution performance. The major compiler options we utilize are -o3, -k, and -pm -op2.

Furthermore, we can take advantage of the software pipeline information, as shown in Figure 5.4. Software pipelining is a technique used to schedule instructions from a loop so that multiple iterations of the loop can be executed in parallel.

Loop unrolling is suitable for use with software pipelining. The compiler needs more waiting time for the decision of branch operation when our code has conditional instructions. If we unroll the loop, some of the overhead for branching instructions can be reduced.

### 5.2.2 Integration on DSP Platform with RTOS

In fact, the tasks of an application may be spread over a large network of processors. We use four processors (SMT395 with C6416 at 1 GHz and 256M DRAM) to implement our integrated system and choose 3L Diamond as our software platform of integration.

We illustrate the multi-processor network with Figure 5.5, taken from [18]. Each channel that communicates from one processor to another is carried by a link, a physical connection or wire, between two processors. It may be implemented in a variety of ways, but it is capable of supporting communication in both directions.

If a channel connects tasks that are on different processors, the messages on that channel are routed through a link between the two processors. Each link can support only two physical channels, one in each direction. We can choose to use these two physical channels

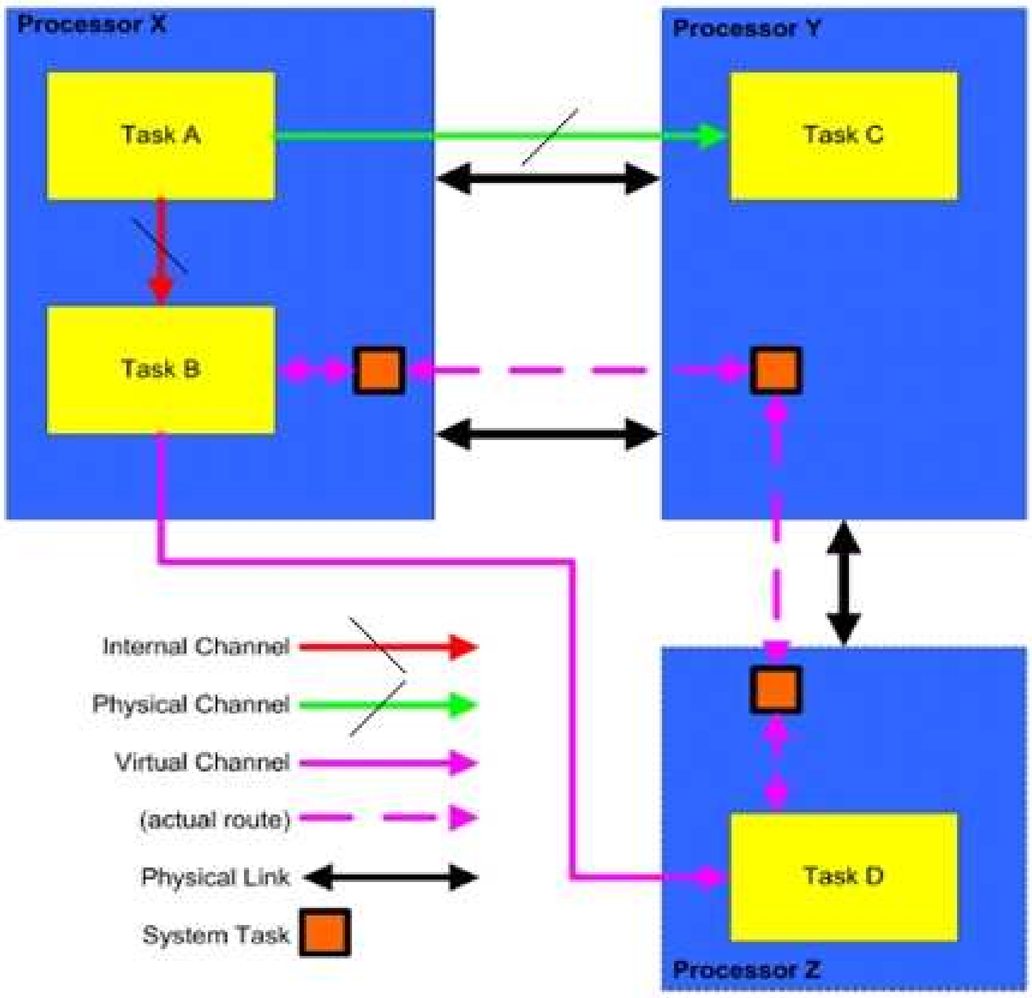


Figure 5.5: Multi-processor networks (from [18]).

explicitly or allow Diamond to manage them for us to give any number of slower virtual channels. The network can only be built if there are enough links between two processors to support all the required channels. Any task may communicate with any other task, regardless of where it is in the network. Messages can be routed via the inter-processor links over virtual channels.

Our integrated system, shown in Figure 5.6, comprises more than ten modules. Arrows indicate inputs and outputs. Solid arrows indicate data while dotted arrows indicate control information. Tables 5.1, 5.2 and 5.3 show the details of each module's function and each I/O parameter's properties.

As mentioned before, Diamond uses a three-stage approach to building an application: compiling source files, using TI linker to generate a number of separate task files, and using configurer to combine task files into a single application file. At the last stage, we need to determine which task should be loaded into each processor.

In order to balance the loading of each processor, we compute the time consumption of each module in processing an uplink slot (three OFDMA symbols). The reason that we choose slot as a unit is a module processes and outputs data of an uplink slot at a time to the next module in our system.

Table 5.4 shows each DSP's load and the required DSP computational load for processing an uplink frame. Then, we complete the last stage and get a single application file that can be loaded into the multi-processor network and execute it using the Diamond command, 3L x.

Finally, our system needs 6.5 ms to process an uplink frame (1440 bytes/frame), where the coding modulation scheme and code rate used in simulation is QPSK and 1/2, respectively. If we reconfigure and use a single processor to implement the integrated system, it needs 17.156 ms to process an uplink frame.

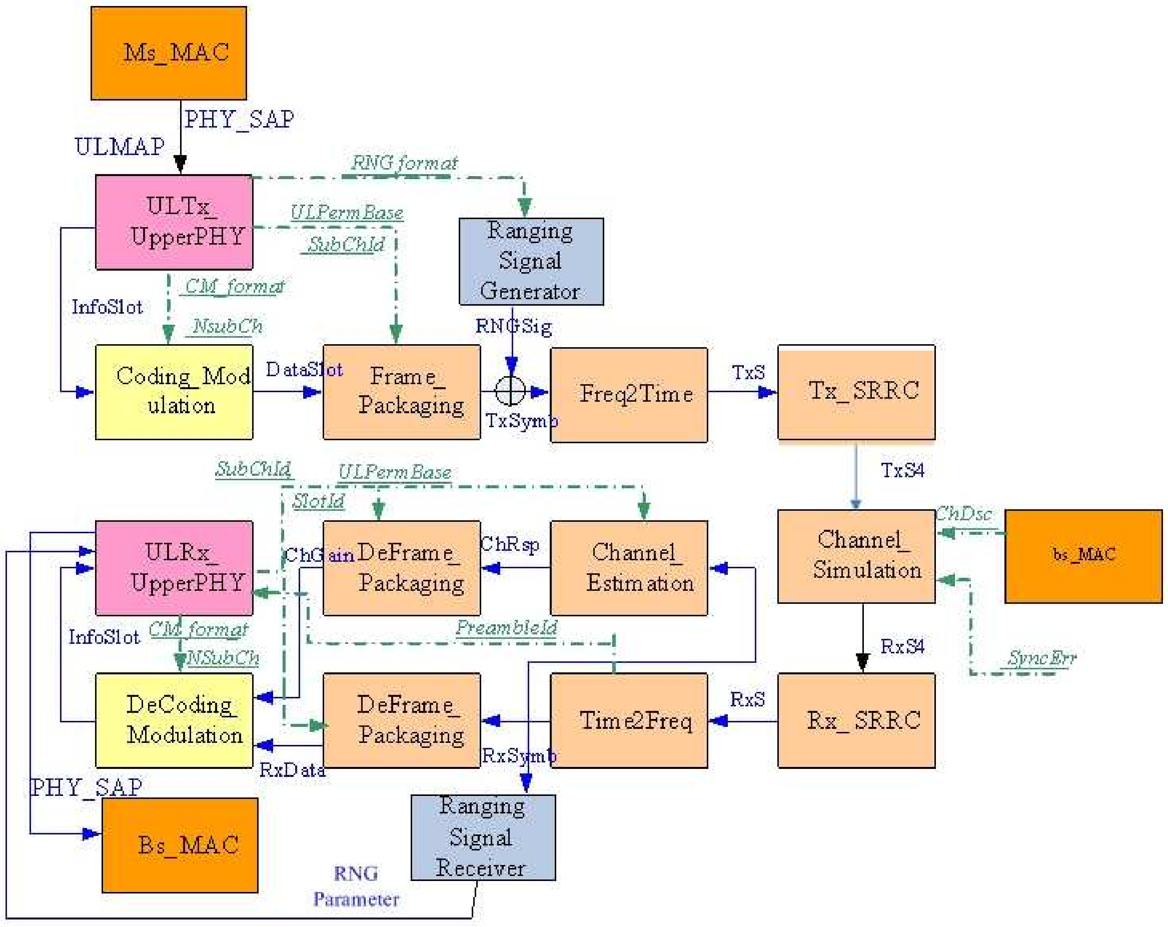


Figure 5.6: Uplink transceiver system.

Table 5.1: Functions of Modules in Figure 5.6

| Name | Input Parameters | Output Parameters |
|---|---|---|
| Ms_MAC | N/A | ULMAP, PHY_SAP |
| ULTx_UpperPHY | ULMAP, PHY_SAP | InfoSlot, NsubCh, SubChId, CM_format, ULPermBase, RNG_format. |
| Coding_Modulation | InfoSlot, CM_format, NSubCh | DataSlot, PilMod |
| Frame_Packaging | DataSlot, PilMod, SubChId, ULPermBase | TxSymb |
| Ranging Signal Generator | RNG_format | RNGSig |
| Freq2Time | TxSymb | TxS |
| Tx_SRRC | TxS | TxS4 |
| bs_MAC | N/A | ChDsc |
| Channel_Simulation | TxS4, ChDsc, SyncErr | RxS4 |
| Rx_SRRC | RxS4 | RxS |
| Time2Freq | RxS | RxSymb, PreambleId |
| Channel_Estimation | RxSymb, ULPermBase, SubChId, SlotId | ChRsp |
| Ranging Signal Receiver | RxSymb | RNGParameter |
| DeFrame_Packaging | RxSymb, ChRsp, SubChId, ULPermBase, SlotId | RxData, ChGain |
| DeCoding_Modulation | RxData, CM_format, ChGain | InfoSlot |
| ULRx_UpperPHY | InfoSlot, PreambleId | CM_format, SubChId, ULPermBase, SlotId, PHY_SAP |
| BS_MAC | PHY_SAP | N/A |

Table 5.2: Descriptions of Modules in Figure 5.6

| Name | Descriptions |
|---|---|
| MS_MAC | Mobile Station of MAC |
| ULTx_UpperPHY | Uplink Transmitter UpperPHY |
| Coding_Modulation | Coding and Modulation |
| Frame_Packaging | Frame Packaging |
| Ranging Signal Generator | Generate the Ranging Signal |
| Freq2Time | Frequency to Time Process |
| Tx_SRRC | Transmit Filter |
| bs_MAC | Virtual Base Station MAC |
| Channel_Simulation | Channel Simulator |
| Rx_SRRC | Receiver Filter |
| Time2Freq | Time to Frequency Process |
| Channel_Estimation | Channel Estimation |
| Ranging Signal Receiver | Estimate and Detect Ranging Signal |
| DeFrame_Packaging | De-Frame packaging |
| DeCoding_Modulation | Coding-Modulation Decoder |
| ULRx_UpperPHY | Uplink Receiver UpperPHY |
| BS_MAC | Base Station of MAC |

Table 5.3: Descriptions of Parameters in Figure 5.6

| Name | Descriptions | Type |
|---|---|---|
| ULMAP | Uplink MAP Information | Data |
| PHY_SAP | Information Stream UpperPHY | Data |
| InfoSlot | Information Bits in a Slot Unit | Data |
| DataSlot | Modulated Data in a Slot Unit | Data |
| TxSymb | Transmit Frequency Domain Symbol | Data |
| RNGSig | Ranging signal Frequency Domain Symbol | Data |
| TxS | Transmit Signals | Data |
| TxS4 | 4x Oversampling Transmit Time Domain Signals | Data |
| RxS4 | 4x Oversampling Received Time Domain Signals | Data |
| RxS | Received Signals | Data |
| RxSymb | Received Frequency Domain Symbol | Data |
| ChRsp | Channel Frequency Response | Data |
| RxData | Received Data | Data |
| ChGain | Channel Gain at the Corresponding Data Carrier | Data |
| RNGParameter | Detected Ranging code, Timing-offset, Frequency-offset, Power-offset | Data |
| CM_format | Coding-modulation Format | Control |
| SubChId | Sub-channel Indexing | Control |
| PreambleId | Preamble Index | Control |
| ULPermBase | Uplink Permutation Base | Control |
| ChDsc | Channel Descriptor | Control |
| SyncErr | Given Synchronization Error in Simulations | Control |
| RNG_format | Ranging Format | Control |
| SlotId | Given Deframing Slot Indexing | Control |

Table 5.4: Required DSP Computational Load

| DSP | Module | DSP Computational Load (1 UL subframe/ 5 ms) |
|---|---|---|
| DSP 1 | TxUphy+FEC+Framing+Freq2Time +Ranging Signal Generator | 0.82 |
| | Tx-Filter | 0.25 |
| DSP 2 | Channel Simulator: | |
| | AWGN | 0.82 |
| | SUIs | 1.52 |
| | ETSI_A | 2.56 |
| DSP 3 | Rx-Filter | 0.54 |
| | Time2Freq+Ranging Signal Receiver +Channel Estimation | 0.23 |
| DSP 4 | RxUphy+De-FEC+De-Framing | 1.16 |

# Chapter 6

# Conclusion and Future Work

## 6.1 Conclusion

In this thesis, we first presented the ranging techniques of the IEEE 802.16e OFDMA TDD system. Second, we implemented the integration of IEEE 802.16e-2005 OFDMA TDD uplink transceiver system on DSP Platform with RTOS (3L Diamond).

In the first part, we mainly discussed the initial ranging process and presented several simulations. In our simulation environments, the frequency offset value was within the range of $[-0.1, 0.1]$, and it did not cause an effect in evidence when we needed to determine whether the received signal contained ranging signal or not. Thus, we could concentrate on developing ranging code detection and timing offset estimation, ignoring the effect of frequency offset.

In the second part, we introduced the software platform used in our integrated system, 3L Diamond. Then, we discussed about how the uplink transceiver system was implemented on DSP platform with RTOS. Our DSP application was made up from a host PC holding one carrier board (SMT310Q) which supported four TIMs (SMT395). We also utilized several methods that could accelerate our code and reduce the execution time. Another version of DSP application employed single TIM rather than four TIMs also was accomplished in our study. Compared with using single TIM, we employed four TIMs to perform overall operations that reduced up to 62% of overall processing time.

## 6.2 Future Work

There are several improvements and extensions can be considered in the future:

- Ranging Process

  In this thesis, we only discussed initial ranging. However, it is possible that includes bandwidth request or handover ranging into our study.

  As mentioned in chapter 4, we pointed out a potential future work when we handled ranging code detection under multipath channels.

- Integration System

  In convolution code, the C64x is equipped with a Viterbi decoder co-processor [20]. Using this co-processor may be helpful in raising the decoding speed.

  As mentioned in last section, using four TIMs can reduce up to 62% of processing time. We may rewrite or re-organize our code, this way may accelerate our code and further reduce the execution time close to 75%.

# Bibliography

[1] WiMAX Introduction on Wikipedia: http://en.wikipedia.org/wiki/802.16e

[2] IEEE Std 802.16-2004, *IEEE Standard for Local and Metropolitan Area Networks — Part 16: Air Interface for Fixed Broadband Wireless Access Systems.* New York: IEEE, June. 2004.

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

[4] OFDMA Introduction on Wikepedia: http://en.wikipedia.org/ofdma

[5] Sundance, Sundance.chm, Apr. 2006.

[6] Sundance home page: http://www.sundance.com/default.asp

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

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

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

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

[11] Texas Instruments, *TMS320C6000 Programmers Guide.* Literature no. SPRU198I, Mar. 2006.

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

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

[14] Xiaoyu Fu and Hlaing Minn, "Initial uplink synchronization and power control (ranging process) for OFDMA Systems," *in IEEE Global Telecommun. Conf.,* vol. 6, Nov.-Dec, 2004, pp.3999–4003.

[15] Jae-Hyok Lee, Evgeny Gontcharov, Jae-Ho Jeon, and Seung-Joo Maeng, it "Apparatus and method for detecting user in a communication system," United States patent application, pub. no.US2007/0002959 A1, Jan. 4, 2007.

[16] Yue Zhou, Zhaoyang Zhang, and Xiangwei Zhou, "OFDMA initial ranging for IEEE 802.16e based on time-domain and frequency-domain approaches," in *Int. Conf. Commun. Technology,* Nov. 2006, pp.1–5.

[17] 3L Diamond Company Homepage: http://www.3l.com/Diamond/Diamond.htm

[18] 3L Diamond, *3L Diamond User Guide: Sundance Edition V3.1.* Sep. 12, 2006.

[19] 3L Diamond, *An Introduction to 3L Diamond on Sundance Hardware.* http://www.3l.com/Documents/Overview.ppt

[20] Texas Instruments, *TMS320C64x DSP Viterbi-Decoder Coprocessor (VCP) Reference Guide.* Literature no. SPRU533D, Sep. 2004.

# 作者簡歷

**姓名**：張順成

**生日**：1982 年 8 月 12 日

**出生地**：馬來西亞

**學歷**：交通大學電信工程系學士

交通大學通訊與網路科技產專碩士

**研究領域**：通訊系統及數位訊號處理

**論文題目**：IEEE 802.16e OFDMA TDD 測距程序與整合建構於即時作業系統DSP 平台之上行傳收系統

(IEEE 802.16e OFDMA TDD Ranging Process and Uplink Transceiver Integration on DSP Platform with Real-Time Operating System)