Implementation of Convolutional Turbo Codes and Timing / Frequency Tracking for Mobile WiMAX

By

Eng. Amr Mohamed Ahmed Mohamed Hussien

Electronics and Communications Department

Faculty of Engineering, Cairo University

A Thesis Submitted to the

Faculty of Engineering at Cairo University

in Partial Fulfillment of the

Requirement for the Degree of

MASTER OF SCIENCE

in

ELECTRONICS AND COMMUNICATIONS ENGINEERING

FACULTY OF ENGINEERING, CAIRO UNIVERSITY

GIZA, EGYPT

September 2008

Implementation of Convolutional Turbo Codes and Timing / Frequency Tracking for Mobile WiMAX

By

Eng. Amr Mohamed Ahmed Mohamed Hussien

Electronics and Communications Department

Faculty of Engineering, Cairo University

A Thesis Submitted to the

Faculty of Engineering at Cairo University

in Partial Fulfillment of the

Requirement for the Degree of

MASTER OF SCIENCE

in

ELECTRONICS AND COMMUNICATIONS ENGINEERING

Under the Supervision of

Prof. Dr. Serag E.D. Habib

Associate Prof. Mohamed M. Khairy

Assistant Prof. Hossam A. Fahmy

Electronics and Communications Dept.

Faculty of Engineering, Cairo University

FACULTY OF ENGINEERING, CAIRO UNIVERSITY

GIZA, EGYPT

September 2008

Implementation of Convolutional Turbo Codes and Timing / Frequency Tracking for Mobile WiMAX

By

Eng. Amr Mohamed Ahmed Mohamed Hussien

Electronics and Communications Department

Faculty of Engineering, Cairo University

A Thesis Submitted to the

Faculty of Engineering at Cairo University

in Partial Fulfillment of the

Requirement for the Degree of

MASTER OF SCIENCE

in

ELECTRONICS AND COMMUNICATIONS ENGINEERING

Approved by the

Examining Committee

_____

Prof. Dr. Hani Fikry Ragai, Member

_____

Prof. Dr**.** Magdy M. S. El-Soudani., Member

_____

Prof. Dr. Serag. E.D. Habib , Thesis Main Advisor

_____

Associate Prof. Mohamed M. Khairy, Thesis Advisor

FACULTY OF ENGINEERING, CAIRO UNIVERSITY

GIZA, EGYPT

September 2008

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to thank my supervisors, Prof. Serag E. Habib, Dr. Mohamed M. Khairy and Dr. Hossam A. Fahmy as they provided me with advice, knowledge, guidance and support throughout the thesis.

I would like also to thank Eng Abd El-Mohsen Khater, Eng Mohamed Ismail, Eng Mohamed Sayed Khairy and Eng Khalid El-Wazeer who participate in the implementation of WiMAX system, through other master theses in a great collaborative work in order to realize the complete system.

Actually, I also appreciate the help offered by the Electronics and Communications department staff, Faculty of engineering, Cairo University. As they give the means and the spirit to realize a good work.

Many thanks go to my parents and my brothers for their continuous support and encouragement during all working days and nights.

ABSTRACT

Convolutional Turbo Codes (CTC) are widely used in many high speed wireless communication systems standards due to their high performance that approaches that of the Shannon limit. The tremendous demands for high throughput and low power in the current wireless communication applications drive the search for efficient implementation techniques to satisfy these requirements. Although many algorithms have been proposed for decoding Turbo codes, their hardware implementation is still a challenging topic. For 802.16e OFDMA based WiMAX, a reliable data transmission is greatly needed, especially in Non-line of sight (NLOS) communication.

In this thesis we study the optional, double-binary, turbo coding used in 802.16e standard. We developed a complete Matlab model for a Turbo encoder and decoder compatible with this standard. We focus on the hardware implementation of the Turbo encoder and decoder. In our implementation, a new efficient metric normalization scheme is proposed. This scheme reduces the storage requirements of the state metric unit by 12.5% over conventional schemes, and reduces the area requirements of the branch metric unit by approximately 34%. Additionally, we introduce a novel implementation of normalized state metrics using a redundant number system. This novel implementation reduces the worst case delay of state metric unit over conventional implementations.

The second part of this thesis is concerned with the implementation of a tracking system for the sampling clock and the residual carrier frequency offset of 802.16e standard. Compared to single carrier schemes, OFDM systems are sensitive to synchronization errors. Thus, an efficient implementation of synchronization in OFDM is the backbone of the system performance. Sampling clock frequency offset is due to the difference between the sampling clock of the

DAC at the transmitter and that of the ADC at the receiver. Timing and frequency synchronization comprises different stages. In this thesis, we are concerned with the timing and frequency tracking stage. We carried out a study and hardware implementation of a joint algorithm that estimates and corrects both the sampling clock offset and the residual carrier offset. Our hardware implementation features reduced hardware area and preserves a good system performance. An FPGA platform is used to implement these modules.

This thesis is a part of a collaborative work that targets to implement the complete mobile WiMAX system. Other master theses study and implement the other blocks.

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

N	:	CTC block interleaver size
N_{cbps}	:	Number of coded bits per encoded block size
Sc	:	Circulation state
A	:	First systematic output sub-block of the CTC interleaver
B	:	Second systematic output sub-block of the CTC interleaver
$Y1$	:	First Parity output sub-block of the CTC interleaver
$W1$	:	Second Parity output sub-block of the CTC interleaver
$Y2$	:	Third Parity output sub-block of the CTC interleaver
$W2$	:	Fourth Parity output sub-block of the CTC interleaver
u_k	:	Original transmitted bit / symbol a time instant k
$L(u_k)$	:	Log Likelihood Ratio of symbol u_k at time instant k
$L(u_k \vert y)$	:	Conditional Log Likelihood Ratio of symbol u_k at time instant k based on the received codeword y
$\alpha_k(s)$	:	Forward state Probability of state s at time instant k
$\beta_k(s)$	:	Backward state Probability of state s at time instant k
$\gamma_{k-1,k}(s' \rightarrow s)$	:	Branch metric (Transition) probability from state s' to state s between time slots k-1 and k
L_c	:	Channel Reliability
$L_e(u_k)$	:	Extrinsic Likelihood of transmitted bit / symbol at time instant k
$A_k(s)$	:	Forward state Probability in Log domain of state s at time instant k
$B_k(s)$	:	Backward state Probability in Log domain of state s at time instant k
$\Gamma_{k-1,k}(s' \rightarrow s)$	:	Branch metric (Transition) probability in Log domain from state s' to state s between time slots k-1 and k

N_s : Total number of samples in one OFDM symbol window

N_u : Number of useful samples of one OFDM symbol window

N_g : Number of samples in the guard interval

LIST OF ABBREVIATIONS

ACC	:	Accumulator
ACS	:	Add / Compare and Select
ADC	:	Analog to Digital Converter
AES	:	Adaptive Encryption standard
AMC	:	Adaptive Modulation and Coding
AWGN	:	Additive white Gaussian Noise
BER	:	Bit error rate
BS	:	Base Station
BTC	:	Block Turbo codes
CBR	:	Constant Bit rate
CC	:	Convolutional Coding
CIR	:	Channel Impulse Response
CORDIC	:	Coordinate Rotation Digital Computer
CP	:	Cyclic Prefix
CPA	:	Carry Propagation Adder
CSA	:	Carry Save Adder
CTC	:	Convolutional Turbo codes
DAC	:	Digital to Analog Converter
DLL	:	Delay locked loop
DSL	:	Digital Subscriber lines
FCH	:	Frame Control Header
FEC	:	Forward error correction
FFT	:	Fast Fourier Transform
FIFO	:	First Input First Output
FPGA	:	Field Programmable Gate Array
FUSC	:	Full Usage of subcarriers

ICI	:	Intercarrier Interference
IDcell	:	Cell Identification Number
IFFT	:	Inverse Fast Fourier Transform
ISI	:	Intersymbol Interference
LDPC	:	Low Density Parity check
LFSR	:	Linear Feedback shift register
LIFO	:	Last Input First Output
LLR	:	Log Likelihood Ratio
LS	:	Least Square
LUT	:	Look up Table
MAC	:	Multiply / Add and Accumulate
MAP	:	Maximum A-posteriori
MCM	:	Multicarrier Modulation
ML	:	Maximum Likelihood
MS	:	Mobile Station
NLOS	:	Non-Line of sight
OFDM	:	Orthogonal Frequency division Multiplexing
OFDMA	:	Orthogonal Frequency division Multiple Access
PPA	:	Partial Product Array
ppm	:	parts per million
PTMP	:	Point to multi-point
PUSC	:	Partial Usage of subcarriers
QAM	:	Quadrature Amplitude Modulation
QPSK	:	Quadrature Phase shift keying
QoS	:	Quality of service
RCFO	:	Residual Carrier Frequency Offset
SCFO	:	Sampling Clock Frequency Offset
SINR	:	Signal to Interference Noise Ratio

SISO	:	Soft Input Soft Output
SMU	:	State Metric Unit
SOFDMA	:	Scalable Orthogonal Frequency division Multiple Access
SOVA	:	Soft Output Viterbi Algorithm
SPID	:	Subpacket Identification Number
SS	:	Subscriber station
TDD	:	Time division duplex
TDMA	:	Time division Multiple access
TUSC	:	Tile Usage of subcarriers
VBR	:	Variable bit rate
WiMAX	:	Worldwide Interoperability for Microwave access

Chapter 1

Introduction to WiMAX

1.1 What is WiMAX

The IEEE802.16 standard defines a Medium Access Control (MAC) and Air Interface protocol for broadband Wireless Metropolitan area Network (W-MAN). The term broadband refers to high speed data transmission. It can be used as an alternative to the current cabled access networks such as optical fibers and Digital Subscriber lines (DSL). It provides broadband services to people who could not afford wired broadband services before. This standard is referred to as WiMAX; it stands for Worldwide Interoperability for Microwave Access. It meets different types of access [1], such as fixed, portable and mobile access. To satisfy different requirements, two versions are defined. The first is IEEE802.16d-2004, optimized for fixed access and based on Orthogonal Frequency division multiplexing (OFDM). The second is IEEE802.16e-2005, optimized for mobile access in addition to supporting fixed access, and based on Scalable Orthogonal Frequency Division Multiple Accesses (SOFDMA).

WiMAX radio might be able to support data rates up to 70 Mbps and operating channel bandwidth from 1.25 MHZ up to 20 MHZ. WiMAX should support access of a distance up to 50 km between user and base station. This means that it supports Non Line of Sight (NLOS) communication. The various channel bandwidth ranges is supported by scalable OFDMA. For example, a WiMAX system may use 128, 512, 1024 or 2048 bit FFT size corresponding to channel bandwidth 1.25MHz, 5MHz, 10MHZ or 20MHz, respectively. A detailed description of OFDM is included in the next section.

1.2 OFDM and OFDMA

1.2.1 Multicarrier Modulation and OFDM

OFDM is a passband Multi-Carrier Modulation (MCM) scheme [2]. MCM is used to overcome problems of Intersymbol interference (ISI) caused by the channel and achieves a high data rate at the same time. The main problem of ISI is caused when the delay spread of the channel is higher than the symbol time. The delay spread causes the current symbol to affect several successive symbols. This effect increases with the increase of data rate. MCM resolves this simply by dividing the data stream among parallel streams or paths, each path is multiplied by a separate carrier as shown in Figure 1.1, each path has a low symbol rate, but the overall rate of parallel streams achieves a high data rate. In order for these streams not to interfere with each other, carriers should be orthogonal.

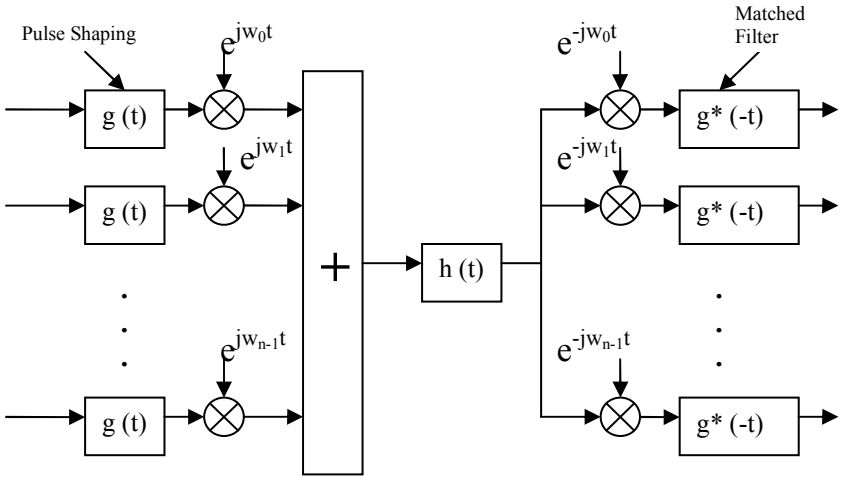


Figure 1.1 Multicarrier Modulation Architecture

Implementation of MCM is achieved via Fast Fourier Transform (FFT). This simplifies hardware implementation where it is almost impossible to achieve perfect orthogonality among all carrier oscillators. However, this is achieved through FFT processing as shown in Figure 1.2.

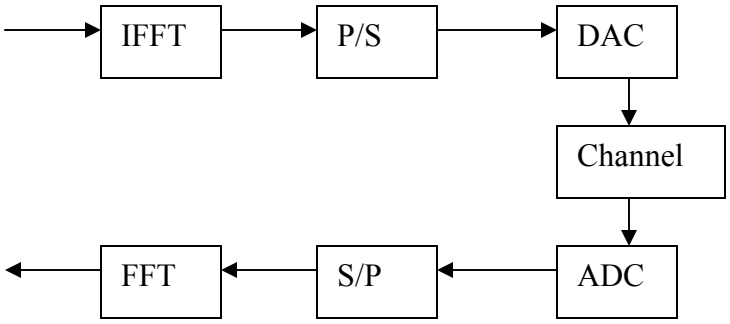


Figure 1.2 OFDM via FFT

However, in case of fading channels, we still have the problem of ISI. In order to eliminate its effect, a guard interval is inserted between consecutive OFDM symbols as shown in Figure 1.3. It should be selected larger than maximum delay spread.

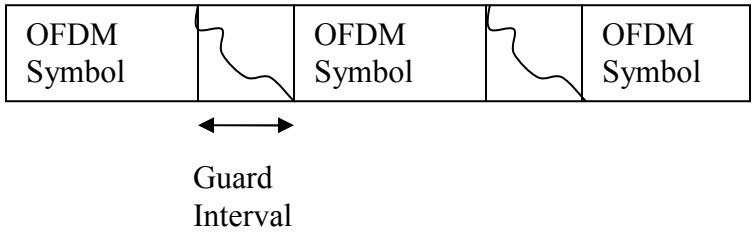


Figure 1.3 OFDM with Guard Interval

Intercarrier Interference (ICI) is another effect from which OFDM symbols suffer. The main reason of ICI problem is mis-synchronization that results from multipath, it will cause subcarriers not to have integer multiple of cycles during the OFDM window. This is considered a loss of orthogonality. To solve this problem, a cyclic prefix (CP) is added before each OFDM window. This is done by simply copying a part of the end of OFDM window to the beginning as shown in Figure 1.4. This ensures that each subcarrier has an integer multiple of cycles in time domain and orthogonality is preserved.

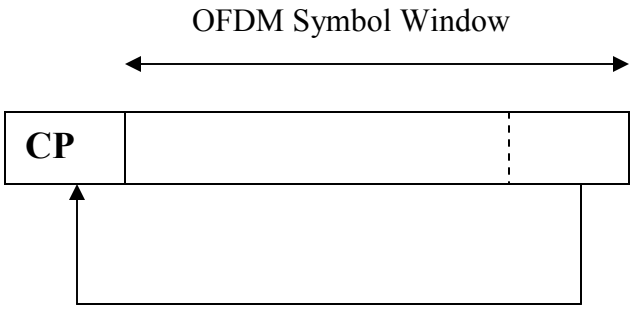


Figure 1.4 OFDM Window with CP

1.2.2 OFDMA

OFDMA employs multiple closely spaced sub-carriers, such as the case of OFDM. However, the sub-carriers are divided into different groups. Each group is defined as a sub-channel. This scheme allows multiple access where each user can be allocated one or more subchannels as shown in Figure 1.5. The sub-carriers that form a sub-channel can be either adjacent or not. In the downlink, a sub-channel may be intended for different receivers. In the uplink, a transmitter may be assigned one or more sub-channels.

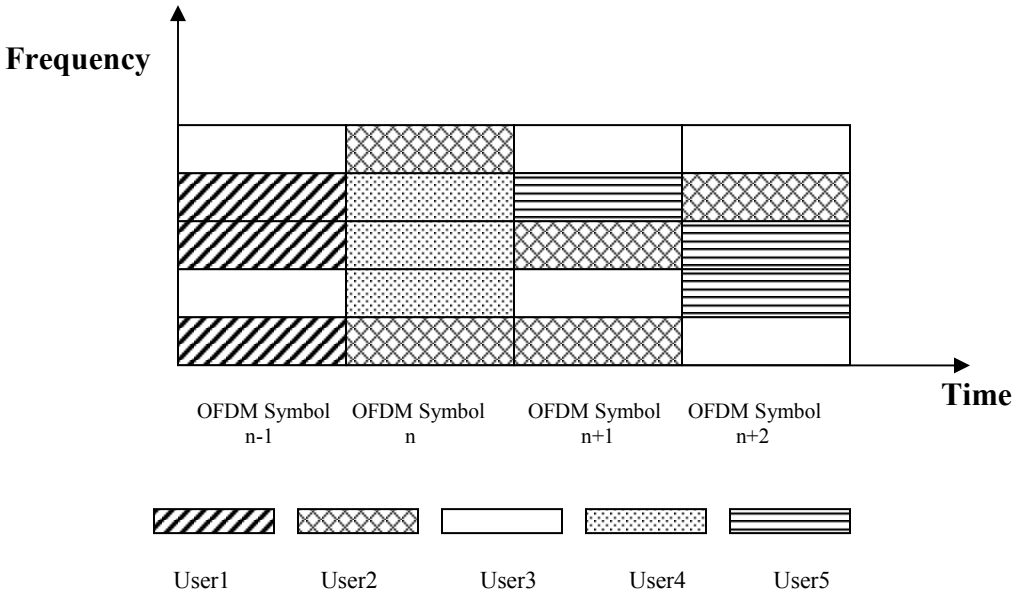


Figure 1.5 OFDMA Multiple access

1.2.3 Scalable OFDMA (SOFDMA)

OFDMA PHY is supposed to have Scalable OFDMA (SOFDMA). This is due to the fact that it allows bandwidth scalability with different FFT sizes. The change of the FFT size means a change in the number of subcarriers. The supported FFT sizes are 128, 512, 1024 and 2048. Only 512, 1024 are mandatory for mobile WiMAX profiles [3]. In case of 802.16e, subcarrier spacing is fixed at 10.94 KHZ. This means that the change in the number of subcarriers indicates a change in bandwidth. Different specified bandwidths are 1.25, 5, 10 and 20 MHZ corresponding to FFT sizes 128, 512, 1024 and 2048 respectively. Adaptive occupied bandwidth provides adaptive data rate.

1.3 OFDMA Symbol Structure

Subcarriers of every OFDMA symbols, like OFDM, are divided into three sets, Data subcarriers, Pilot subcarriers and Null subcarriers as shown in Figure 1.6.

1. **Data subcarriers** are occupied with user data symbols.
2. **Pilot subcarriers** are used for carrying pilot symbols. The pilot symbols are known symbols that can be used for synchronization and channel estimation purposes.
3. **Null subcarriers** have no power allocated to them, including the DC subcarrier and the guard subcarriers. The DC subcarrier is not modulated, to avoid saturation effects or excess power draw at the amplifier. No power is allocated to the guard subcarrier in order to avoid interference effects with adjacent bands.

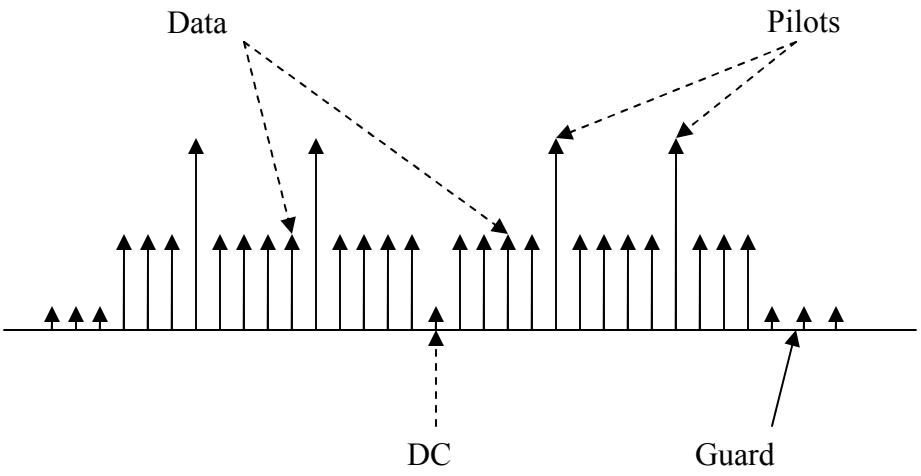


Figure 1.6 OFDMA Symbol Structure

1.4 OFDMA Frame Structure

The OFDMA frame is composed of two subframes, a downlink subframe and uplink subframe operating in a Time division Duplex (TDD) mode; this allows a sharing of bandwidth between uplink and downlink. The downlink subframe contains a downlink preamble, a Frame Control Header (FCH), DL-MAP, UL-MAP and DL-bursts. The preamble is used for time and frequency synchronization and initial channel estimation. FCH provides the frame configuration information, such as coding rate and modulation scheme used. DL-MAP and UL-MAP specify which data regions are allocated for each user. DL-Bursts carry data of several users in case of downlink. For Uplink subframe, it contains UL-bursts which carry data of several users in case of uplink and a ranging subchannel. It is used for ranging purposes. Ranging is a procedure that maintains the quality and reliability of the radio-link communication between the Base Station (BS) and the Mobile Station (MS). When the BS receives the ranging transmission from a certain MS, the BS can estimate various radio-link parameters, such as channel impulse response, Signal to Interference and Noise

Ratio (SINR), and time of arrival. The BS is able to adjust the transmit power level, and so on.

1.5 Subcarrier Permutation schemes

Subcarrier permutation is simply considered as combining different subcarriers into a subchannel. The set of subcarriers that construct a certain subchannel depends on subcarrier permutation schemes. Subcarriers that form a subchannel can be either adjacent or distributed. In IEEE802.16e, different permutation schemes are defined such as Downlink Full Usage of subcarriers (DL-FUSC), Downlink Partial Usage of subcarriers (DL-PUSC), Uplink Partial Usage of subcarriers (UL-PUSC), Tile Usage of Subcarriers and Band Adaptive Modulation and Coding [4]. They are discussed in some details in next sections.

1.5.1 Downlink Full Usage of Subcarriers

In this permutation scheme, each subchannel is constructed from 48 data subcarriers from the same OFDM symbol. These subcarriers are evenly distributed in the OFDM symbol. Number of subchannels in one OFDM symbol differs depending on number of data subcarriers that varies according to FFT size. Figure 1.7 illustrates this permutation scheme.

1.5.2 Downlink Partial Usage of Subcarriers

In case of DL-PUSC, subcarriers are divided into clusters; each cluster consists of 14 adjacent subcarriers over two OFDM symbols. The clusters are then divided into six groups and a subchannel is constructed from two clusters of the same group as indicated in Figure 1.8.

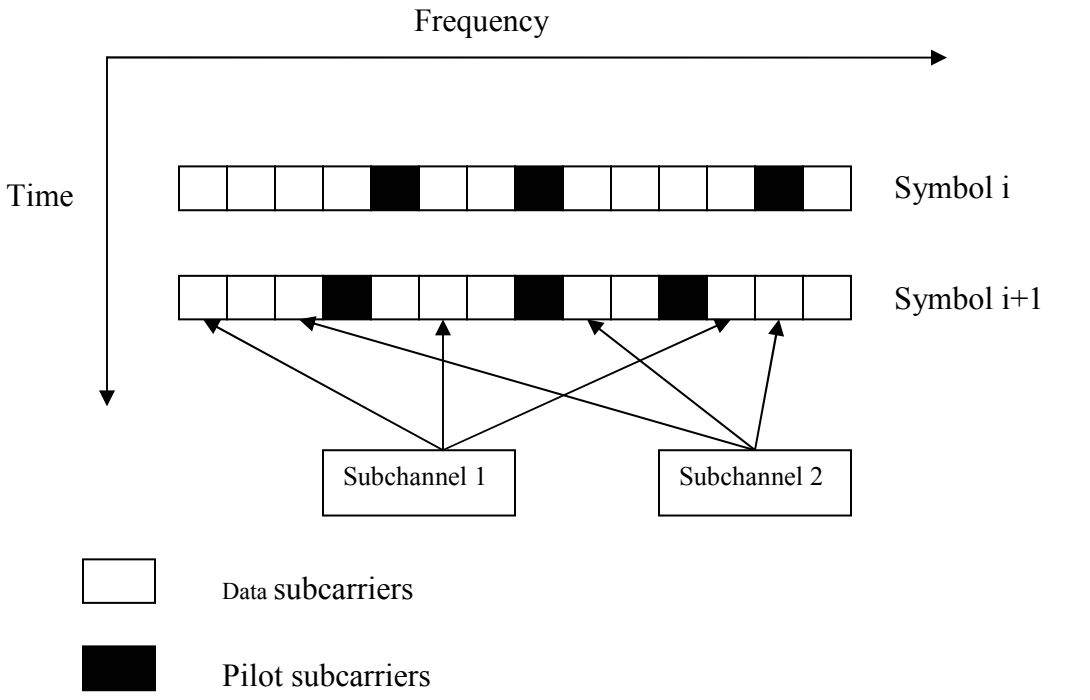


Figure 1.7 Downlink FUSC permutation scheme

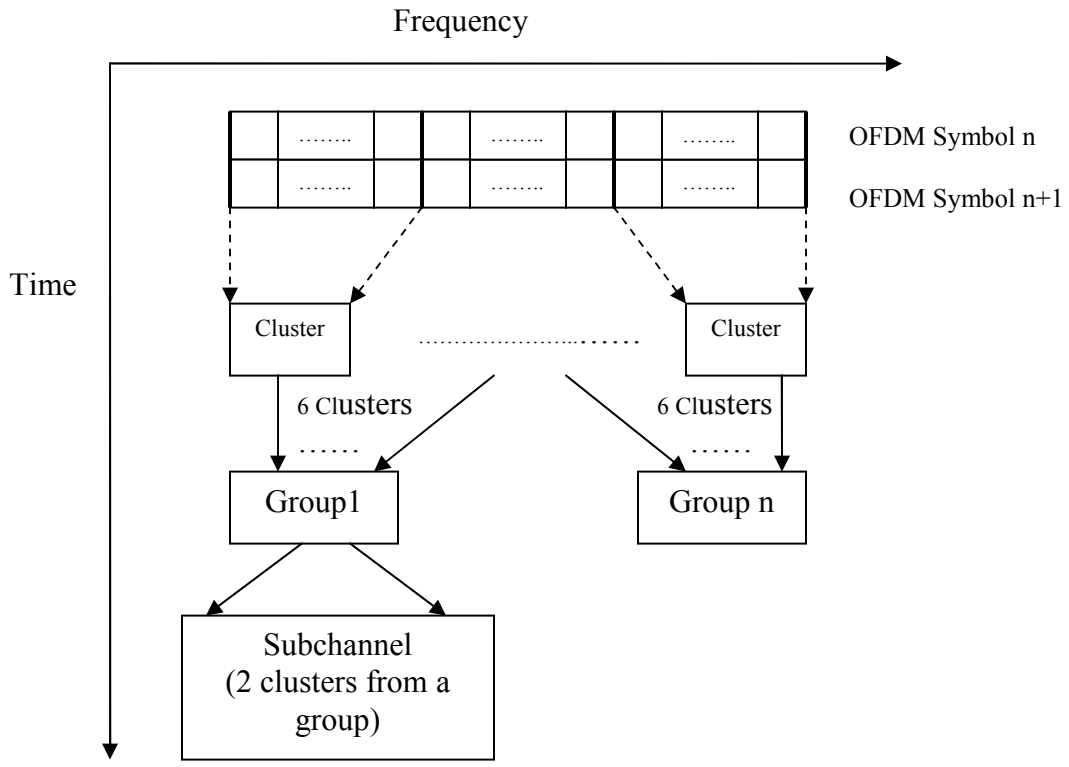


Figure 1.8 Downlink PUSC permutation scheme

1.5.3 Uplink Partial Usage of Subcarriers

In this case, subcarriers are divided into tiles; each tile consists of 12 subcarriers over 3 OFDM symbols, i.e. 4 subcarriers per symbol. The subcarriers of each tile are divided into 8 data subcarriers and 4 pilot subcarriers. Tiles are renumbered pseudo-randomly and divided into 6 groups. Subchannel is constructed from 6 uplink tiles from the same group.

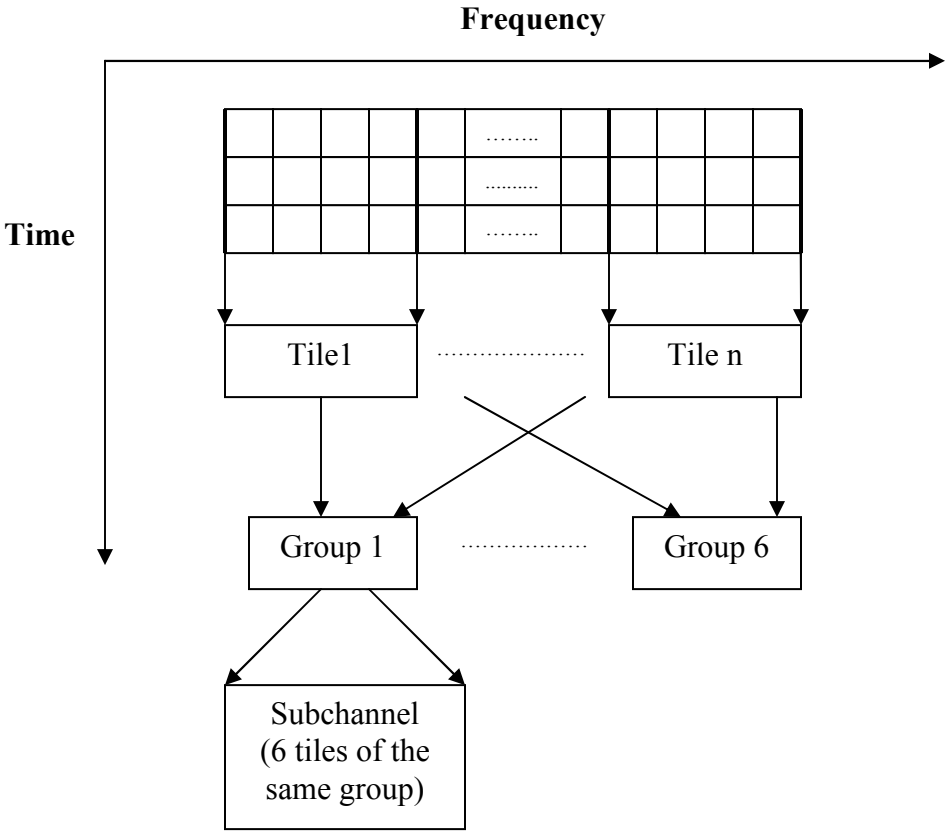


Figure 1.9 Uplink PUSC permutation scheme

1.5.4 Tile Usage of Subcarriers

The Tile Usage of subcarriers (TUSC) is a permutation scheme used in downlink. It is identical to the Uplink PUSC. This has the advantage of downlink and uplink allocation symmetry.

1.5.5 Band Adaptive Modulation and Coding

In the band Adaptive Modulation and Coding (AMC) permutation scheme, subcarriers that construct one subchannel are adjacent. In order to form a subchannel, subcarriers are divided into bins, each bin consists of nine consecutive subcarriers as shown in

Figure 1.10, these nine subcarriers are divided into 8 data subcarriers and one pilot subcarrier. The AMC subchannel can have various shapes; it can be one bin over six consecutive OFDM symbols, two bins over three consecutive OFDM symbols or six consecutive bins over one OFDM symbol.

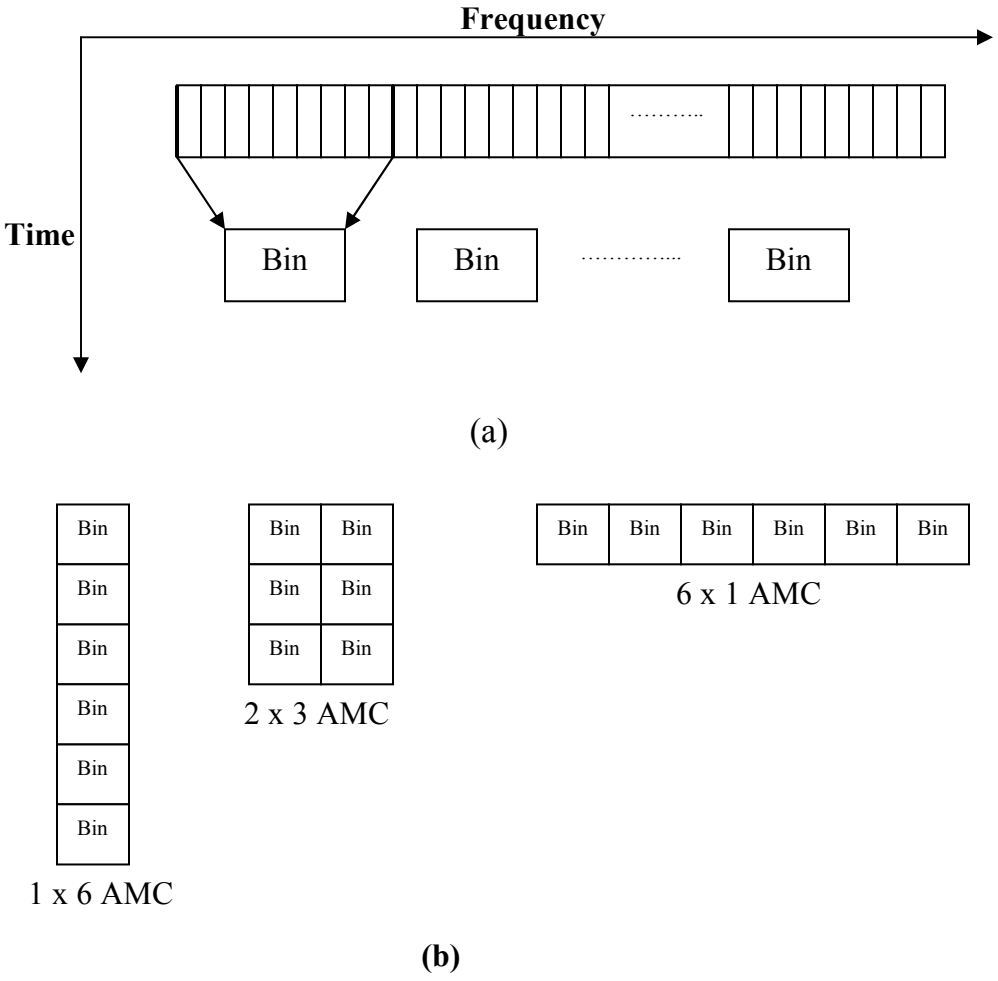


Figure 1.10 (a) AMC Permutation mode; (b) different AMC subchannels

1.6 WiMAX Features

WiMAX is a broadband wireless technology that is rich in features such as Flexibility, Scalability, Quality of Service (QoS), Security, Mobility…etc.

1.6.1 Scalability

Scalable OFDMA on which IEEE802.16e is based provides a scalable bandwidth. This scalable bandwidth allows dynamic support of user roaming across different networks. These networks may have different bandwidth allocations.

1.6.2 QoS

The MAC layer of WiMAX should support a variety of applications with different QoS requirements such as best effort based applications, real time and non-real time applications, constant bit rate (CBR) and variable bit rate (VBR) based applications.

1.6.3 Mobility

WiMAX can support many users in a coverage area up to 50 Km. In order to support mobile applications, the MS and the BS need to introduce several mobility-supporting functions to the existing WiMAX system. Power saving mechanisms should be used. In addition, more frequent channel estimation and power control is specified for the purposes of mobility.

1.6.4 Security

WiMAX supports advanced strong security techniques, such as Advanced Encryption Standard (AES). It also specifies security procedures used to

authenticate and maintain private encryption keys. These private encryption keys are used to encrypt traffic to first-hop neighbors or to the base station. More about security features can be found in [5].

This thesis is focused mainly on the study and implementation of some blocks of the PHY layer of IEEE802.16e standard. This standard defines some mandatory features and other optional features. We present the simulation and implementation of some blocks of the physical layer. In chapter 2, a review of the IEEE802.16e PHY model is illustrated, defining the main mandatory and optional features. The next chapters concentrate on the implemented blocks with performance simulation and hardware implementation.

Chapter 2

802.16e PHY Model

2.1 Introduction

The IEEE802.16 defines four Physical (PHY) layers, they can be summarized as:

1. Wireless-MAN SC: It is based on single carrier modulation, and is designed for frequency ranges higher than 11 GHZ for a LOS operation.
2. Wireless-MAN SCa: It is based on single carrier modulation, and is designed to operate at frequency ranges between 2- 11 GHZ for NLOS purposes.
3. Wireless-MAN OFDM: A PHY layer using a 256 point FFT based OFDM. It is designed for point to multi-point (PTMP) operation in a NLOS conditions. It operates at frequency ranges between 2-11 GHZ. It is also referred to as Fixed WiMAX. Multiple access of different subscriber stations (SSs) is time-division multiple access (TDMA)-based.
4. Wireless-MAN OFDMA: A PHY layer using a 2048 point FFT based OFDMA. It operates in frequency ranges between 2-11 GHZ and supports NLOS communication. It is also referred to as Mobile WiMAX.

2.2 Channel Coding in 802.16e PHY Transmission

The IEEE 802.16e PHY model specifies some mandatory and optional features. The PHY mandatory chain is illustrated in Figure 2.1. It consists of a Randomizer, Forward Error Correction (FEC) block, which specifies convolutional coding as a mandatory FEC block. It is followed by Interleaving block, then QAM mapping before IFFT block [6],[7]. The FEC block size equals an integer number of subchannels and the channel coding is performed on each FEC block. Some parameters in PHY layers are flexible and controlled by higher layers such as FEC block size, coding rate, Modulation type, CP length, and so on.

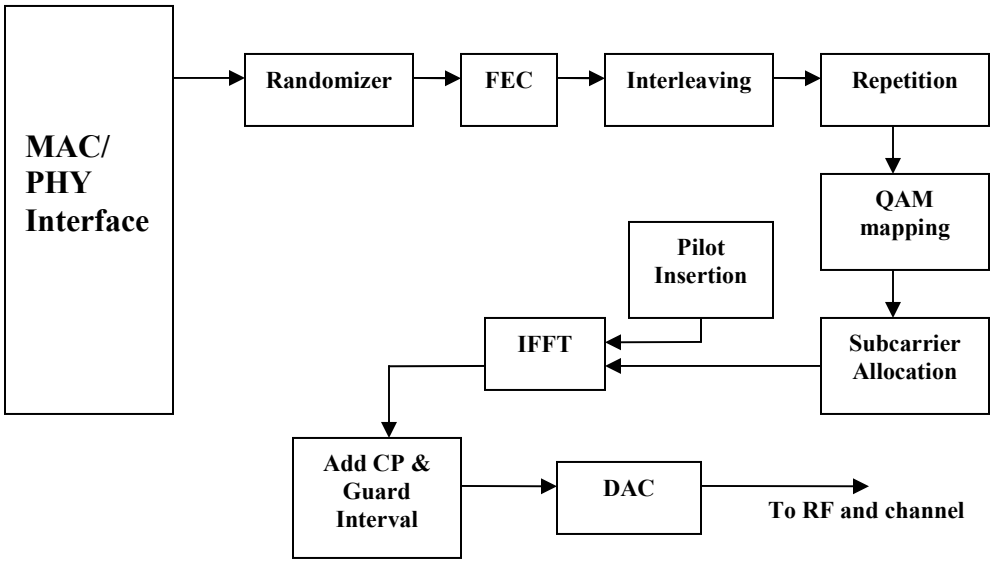


Figure 2.1 Mandatory Channel Coding at transmission

2.2.1 Randomizer

The purpose of the randomization block is to prevent a long sequence of consecutive ones or zeros. This helps in purposes of synchronization at the receiver. Randomization is done on each FEC block separately. It is simply performed with a Mod-2 addition operation between FEC data bits and other generated Pseudo random sequence of bits. This sequence is generated by a Linear Feedback Shift Register (LFSR) as shown in Figure 2.2. It is initialized with a certain known sequence given as (LSB) [0 1 1 0 1 1 1 0 0 0 1 0 1 0 1] (MSB).

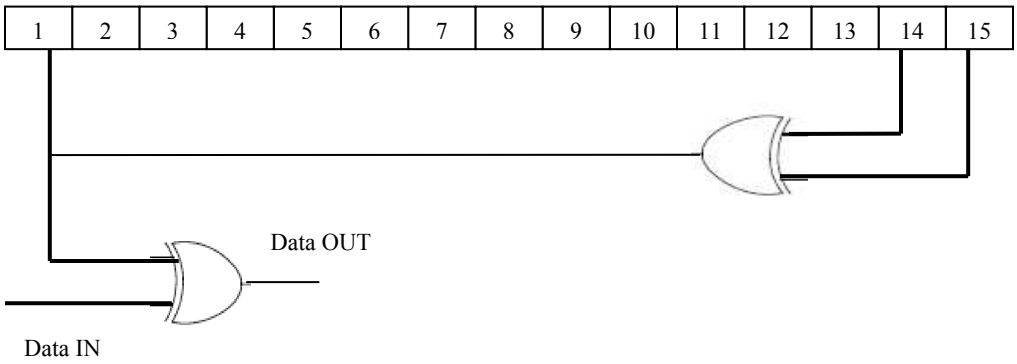


Figure 2.2 Randomizer PRBS

2.2.2 Forward Error correction

The purpose of channel coding is to help the receiver to be able to recover channel errors. This is carried out through transmitting redundant bits beside the original information bits. These redundant bits can be constructed as a function of the original information bits. They help to recover channel errors. Many coding schemes were defined in communication systems to be used for these purposes [8]. In the IEEE802.16e standard, some coding schemes are defined as mandatory coding schemes; others are defined to be optional. The Convolutional Coding (CC) is defined as a mandatory channel coding scheme. The standard also defines other optional coding schemes such as Block Turbo Codes (BTC), Convolutional Turbo Codes (CTC), and Low Density Parity Check Codes (LDPC). In this section we take a look on the mandatory Convolutional Coding used, and in chapter 3, we handle the Convolutional Turbo Codes on which this thesis deals.

Convolutional coding specified in the IEEE802.16e standard is a binary non-recursive convolutional coding. It is considered binary as it deals with one input at a time and is considered non-recursive as it has no feedback. The mandatory CC has a rate ½ and constraint length of 7; this means that it has two outputs for each input, and it has 6 delay elements as shown in Figure 2.3.

The generator polynomials can be specified by placing 1's in case of a feedback connection and 0's elsewhere. We get the following generator polynomials for the two outputs

G1=[1 1 1 1 0 0 1]

G2=[1 0 1 1 0 1 1]

In general, the generator polynomials of the two outputs are specified in octal format as:

$G1 = 171_{OCT}$

$G2 = 133_{OCT}$ (2.1)

The remaining part of the convolutional encoder is the puncturing block which aims to reduce the number of transmitted bits depending on the channel conditions. This is carried out by controlling the code rate. Possible code rates are 1/2, 2/3, and 3/4. The FEC block size is determined by modulation type and code rate.

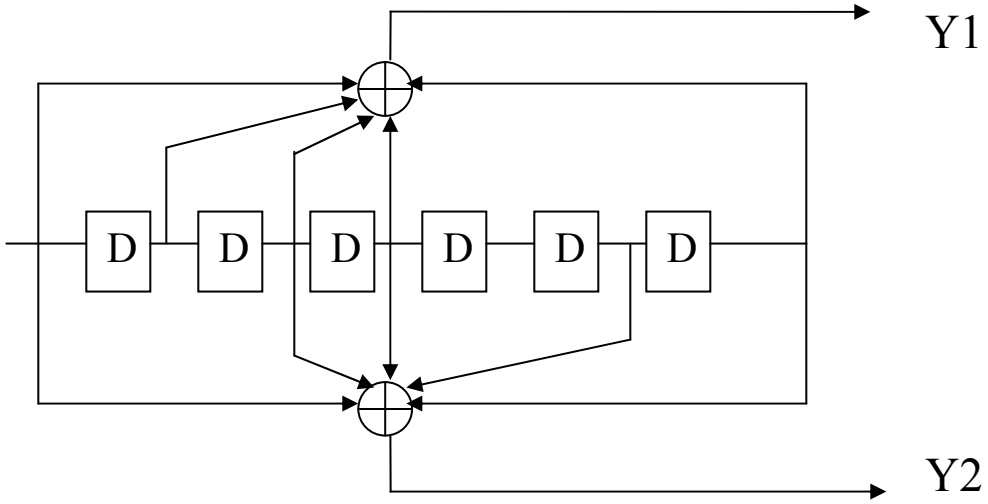


Figure 2.3 Convolutional encoder structure

2.2.3 Interleaving

The next block in channel coding is the interleaving block. The main function of this block is to redistribute the order of transmitted bit such that consecutive bits are allocated to non-adjacent subcarriers in order to avoid burst errors. In case of frequency selective channels, which have a variant frequency response over the user bandwidth, adjacent subcarriers are exposed to similar channel conditions. Burst errors are not desirable as it has a severe effect on decoding. Interleaving is important as it reduces the effect of successive errors by converting burst errors to single separated errors. The interleaver is defined by a

two-step permutation. The first ensures that adjacent coded bits are mapped onto nonadjacent subcarriers. The interleaver block size is the number of coded bits per encoded block size N_{cbps}. The first permutation step depends on N_{cbps}, as indicated in (2.2)

$$m_k = \frac{N_{cbps}}{d}.k \bmod_d + \left\lfloor \frac{k}{d} \right\rfloor \tag{2.2}$$

Where k =0,1,2,……., N_{cbps}-1 and d =16

The second permutation step ensures that adjacent coded bits are mapped alternately onto less or more significant bits of the constellation. This avoids long runs of lowly reliable bits. The second permutation is defined by the formula given in (2.3) as follows

$$j_k = \left(s.\left\lfloor \frac{m_k}{s} \right\rfloor + (m_k + N_{cbps} - \left\lfloor \frac{d.m_k}{N_{cbps}} \right\rfloor \right). \bmod_s \tag{2.3}$$

Where k =0, 1, 2… N_{cbps}-1 and d =16.

Where s is a parameter depending on the modulation scheme as indicated in (2.4).

$$s = \frac{N_{cpc}}{2} \tag{2.4}$$

and N_{cpc} is the number of coded bits per subcarrier, which equals 2 in case of QPSK, 4 in case of 16-QAM , and 6 in case of 64-QAM.

2.2.4 Repetition

After FEC and interleaving, a repetition block may be used only in case of QPSK modulation. The repetition is performed on the unit of slots. First, data bits are segmented into slot. Each group of bits form a slot that should be repeated R times in order to form R contiguous slots. The repetition factor R can be 2, 4, or 6. The repetition coding is used to further increase signal margin over the modulation and FEC mechanisms.

2.2.5 Modulation

In this stage, data and pilot subcarriers should be modulated prior to forwarding to the IFFT block. This is done in two steps: subcarrier randomization and modulation.

2.2.5.1 Subcarrier Randomization

In this case, a PRBS is used to generate a sequence W_k. This sequence is used in data and pilot modulation as indicated in the next two sections. The PRBS used to generate W_k is shown in Figure 2.4. Initialization of PRBS depends on either uplink or downlink, cell identification number (IDcell), and segment number.

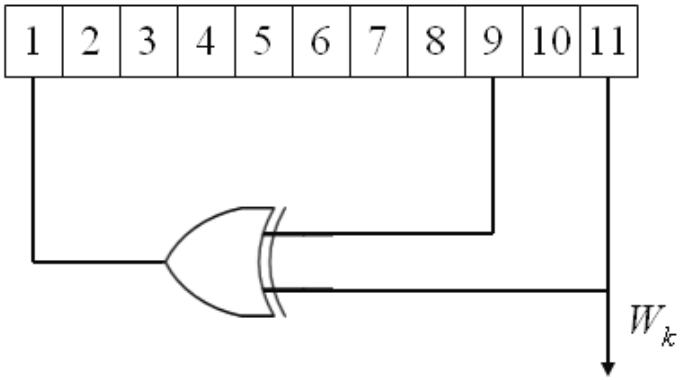


Figure 2.4 PRBS generator for data and pilot modulation

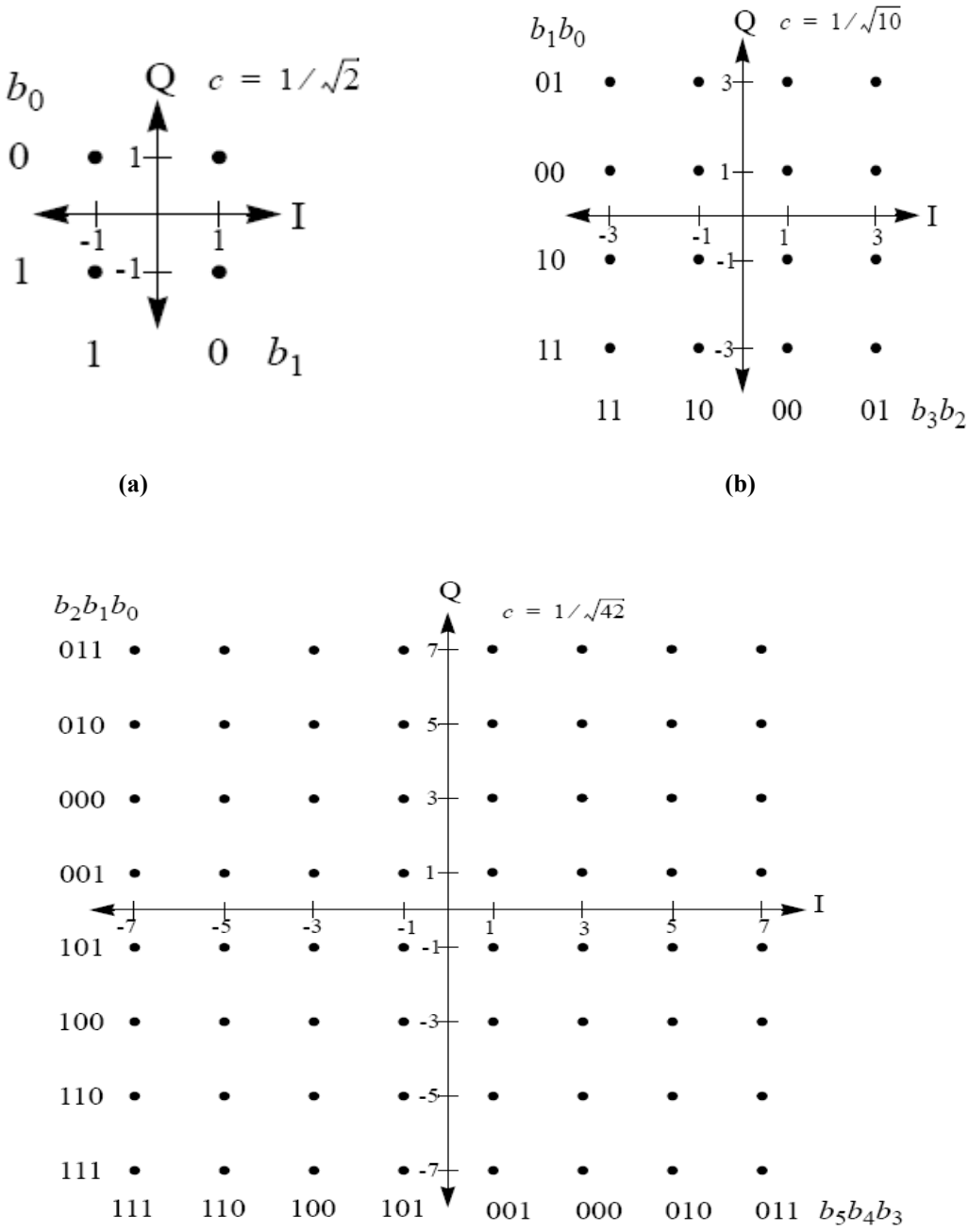
Initialization of PRBS is determined as follows:

- b_0-b_4: Five least significant bits of IDcell as indicated by the frame preamble.
- b_5-b_6: In case of Downlink, It represents the segment number + 1 as indicated by the frame preamble where b_5 is the MSB and b_6 is the LSB. In case of uplink, it is set to all ones.
- b_7-b_{10}: In case of downlink, it is set to all ones and in case of uplink, it is set by the four least significant bits of the frame number, where b_7 is the MSB and b_{10} is the LSB.

2.2.5.2 Data Modulation

The IEEE802.16e defines both QPSK and 16-QAM as mandatory modulation schemes and 64-QAM as an optional one. Figure 2.5 illustrate the constellation diagrams of these modulation techniques. In order to achieve equal average power, the mapped constellation should be multiplied by a factor *c* which depends on the applied modulation type as follows:

- $c = 1/\sqrt{2}$ in case of QPSK
- $c = 1/\sqrt{10}$ in case of 16-QAM
- $c = 1/\sqrt{42}$ in case of 64-QAM



(a) **(b)**

(c)

Figure 2.5 (a) QPSK Constellation diagram (b) 16-QAM Constellation diagram
(c) 64-QAM Constellation diagram

The next step is to multiply each subcarrier by a factor of $2\left(\frac{1}{2}-W_k\right)$ where k is the subcarrier index.

2.2.5.3 Pilot Modulation

As mentioned in section 1.3, some subcarriers are filled with pilots in order to help for channel estimation and synchronization purposes at the receiver. Pilots are modulated as indicated in the formula specified by (2.5) in case of uplink and (2.6) in case of downlink.

In case of uplink, the modulated pilot c_k is given by:

$$\mathrm{Re}\{c_k\} = 2\left(\frac{1}{2}-W_k\right)$$

$$\mathrm{I}m\{c_k\} = 0 \qquad (2.5)$$

In case of downlink, the modulated pilot c_k is given by:

$$\mathrm{Re}\{c_k\} = \frac{8}{3}\left(\frac{1}{2}-W_k\right)$$

$$\mathrm{I}m\{c_k\} = 0 \qquad (2.6)$$

2.2.6 Subcarrier Allocation

In this step, the output transmitted symbols after modulation should be mapped to certain subcarriers. The procedure that determines which data symbols will be allocated to which subcarriers and how to allocate pilots to subcarriers depends on subcarrier permutation scheme specified in section 1.5. It simply maps the logical numbering, which is the order of data symbols to be transmitted, to a physical numbering which is the order of subcarriers before entering the IFFT block. Pilot insertion is performed in parallel to subcarrier allocation, the number

and location of pilots in a certain OFDM symbol is determined according to the applied permutation scheme and adjusted FFT size.

2.2.7 IFFT

The IFFT block is the main block that performs the multicarrier modulation. It is applied to each OFDMA symbol separately. Prior to IFFT, we consider the symbols in the frequency domain. After the IFFT, we consider symbols in the time domain in order to be transmitted over the channel.

As mentioned before, the IEEE802.16e supports FFT sizes of 128, 512, 1024 and 2048 respectively. The IFFT modulation is performed to symbols with complex values after QAM mapping. After construction of OFDM symbol window in time domain, CP is inserted in order to maintain orthogonality of different tones. In IEEE802.16e, CP can be either 1/4, 1/8, 1/16, and 1/32.

2.2.8 RF Section

The last block in the transmitter is a passband modulation. It is carried out by converting the digital baseband signal to analog signal via Digital to Analog Converter (DAC) then multiplying the output baseband stream by RF carrier prior to transmission over the wireless channel.

2.3 Receiver block diagram

During transmission over the channel, transmitted symbols suffer from channel conditions which have severe impact on these symbols such as noise, multipath fading, and interference from other users in the same band and out of band. The output of the channel is transferred as input to the receiver. The function

of the receiver is not only to reverse the operations of the blocks at the transmitter, but also it should recover the channel effects. In this case, we have additional blocks at the receiver to compensate for channel effects. The main supplementary blocks used in the receiver are Timing and Frequency synchronization blocks in addition to channel estimation block. Figure 2.6 illustrates the most common blocks of the receiver.

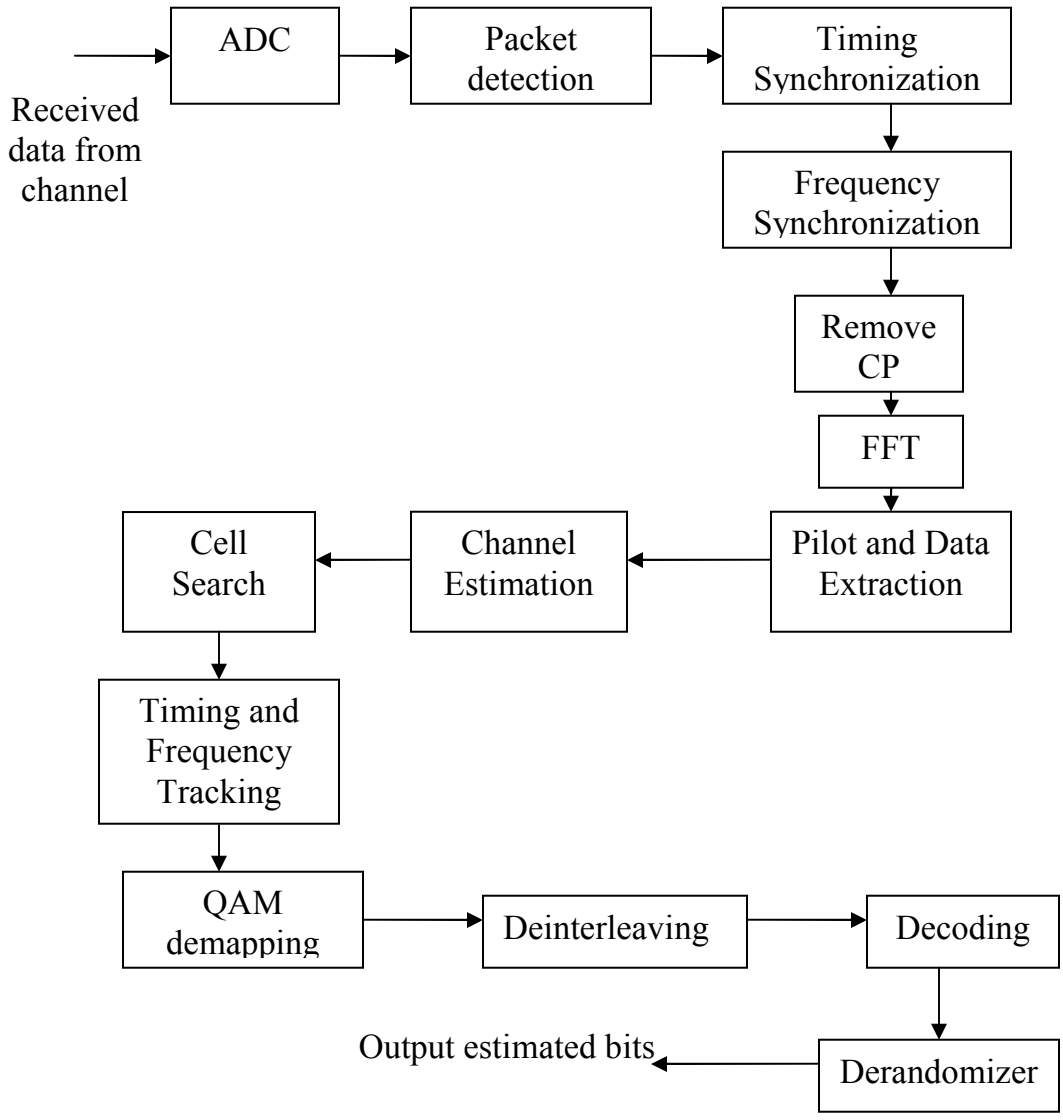


Figure 2.6 Receiver block diagram

2.3.1 Timing Synchronization

Synchronization in Communication systems is a crucial issue. The main purpose of synchronization is to allow the receiver to recognize the start and end of OFDM symbols in order to begin processing of data. If the OFDM window is placed in a wrong position, this is considered a timing offset. This has a severe effect on performance degradation.

Timing synchronization in OFDM systems comprises three stages: Packet detection, Symbol timing and sampling clock tracking. Packet detection enables the receiver to detect that a new frame is being received. Symbol timing enables the receiver to determine the start and end of OFDM symbol. Sampling clock tracking compensates for the clock frequency offset between DAC at transmitter and ADC at receiver. More details about synchronization will be discussed in chapter 6.

2.3.2 Frequency Synchronization

In addition to the Timing offset problem, Frequency offset has its severe impact on system performance. The main reason of frequency offset is the difference between local oscillators at both transmitter and receiver. The main task of the frequency synchronization is to correct the errors produced from the frequency offset. Frequency synchronization is carried out in three steps; coarse frequency offset, fine frequency offset and frequency offset tracking. Chapter 6 presents a detailed description of these steps.

2.3.3 FFT

The main task of the FFT block is to reverse the task of the IFFT at the transmitter. The output of this block is the OFDM symbols in the frequency

domain. After FFT operation, data and pilot subcarriers are extracted from the OFDM symbol and null subcarriers are removed. Prior to the FFT operation, Guard time and CP are removed from the OFDM window, and then the OFDM window with a certain number of samples is prepared for FFT operation to construct OFDM symbol in the frequency domain. After FFT operation, physical mapping for subcarriers should be converted back to its original logical mapping.

2.3.4 Cell Search

Cell search block is used to identify the cell and segment to which the mobile station belongs. This is done with the aid of a preamble. In case of 802.16e, 114 different preambles are used. The preamble detection helps to recognize IDcell and segment number.

2.3.5 Channel estimation

The channel estimation block is used to determine the channel impulse response (CIR). Channel has its effect on both magnitude and phase of subcarriers. This has the effect on rotation of subcarriers in the frequency domain, in addition to attenuation of magnitude. The receiver has to compensate for this error and correct it. Many algorithms have been proposed for channel estimation. These can be found in [9 - 11].

2.3.6 Demapper

The demapper block performs the reverse operation of QAM mapper at the transmitter; it constructs back the original stream of bits from the received QAM symbols. However, it should produce a soft estimate of these bits in order to be used by the decoder.

2.3.7 Decoding

Depending on the coding scheme used at the transmitter, decoding is done at the receiver. In case of mandatory convolutional coding, Viterbi decoding is used at the receiver. Viterbi decoding simply uses the principle of Maximum Likelihood (ML) decoding at the receiver [8]. The operation of the convolutional encoder can be specified as a state machine. The data bits stored in the delay elements represents the current state of the encoder. The inputs and current state determine the output and next state. An extension to the state diagram in time is the trellis diagram [8]. It simply represents transition from one state to another state each time slot depending on the input. For a certain codeword, there is a certain set of transitions that construct a certain path in the trellis diagram. The function of the viterbi decoder is to determine the nearest path to the received codeword and hence, determine the original information bits. More explanation of viterbi decoding can be found in [8],[12].

2.3.8 Derandomizer

Derandomizer retrieves the original data stream that was randomized at the transmitter. The structure of derandomizer is the same as randomizer. A PRBS is used to generate random bits; these bits are modulo-2 added to the output of the decoder to generate final estimated data bits.

2.4 WiMAX PHY Implementation

Implementation of current wireless communication standards is still a challenging topic. The tremendous demands of high throughput and low power consumption needed in current wireless communication applications drives the design of efficient implementation techniques to satisfy these requirements. For

802.16e OFDMA based WiMAX, there is a great challenge to satisfy system requirements to be able to operate over NLOS conditions, over a distance up to 50 miles. This means that reliable transmission and signal processing at receiver should be maintained. In addition, 802.16e supports mobility, so, lower power consumption is a crucial issue in implementation.

Many implementations of several blocks in transmission and reception have been proposed. Implementation of most mandatory blocks can be found in [13], [14]. In this thesis, we study the optional Convolutional Turbo coding used in 802.16e with its hardware implementation. We study also the Sampling clock tracking and frequency offset tracking with a review of some previous work and proposed hardware implementation.

Chapter 3

Turbo Coding

3.1 Introduction

In the IEEE802.16e standard, Turbo Coding is defined as an optional block used in channel coding. The standard defines two types of turbo codes: Block Turbo Coding (BTC) and Convolutional Turbo Coding (CTC). In this thesis, only Convolutional Turbo Coding is implemented. It has an improvement in system performance over mandatory convolutional codes. CTC has been widely used in many high speed wireless communication systems standards due to its high performance that approaches that of Shannon limit. It is introduced in 3GPP, DVB-RCS and WiMAX. Turbo Coding was introduced in 1993 by Berrou, Glavieux, and Thitimajshima [15],[16]. It consists of a set of serial or parallel concatenated constituent encoders. Each one encodes an interleaved version of the original data.

In this thesis, we handle Turbo Coding used in 802.16e standard. This chapter includes a detailed description of CTC encoding represented in the standard, and then several decoding techniques are explained in details. Algorithms that use approximations to simplify hardware implementation are also described. Then we apply these concepts to the specific turbo codes used in this standard. We state the previous work and some proposed improvements.

3.2 Turbo Encoding

3.2.1 Block Description

Convolutional Turbo encoder specified in IEEE802.16e standard is composed of two constituent encoders in addition to an interleaver. The output of CTC encoder consists of systematic bits, and parity bits. Systematic output bits are identical to input bits, and parity bits are outputs of constituent encoders. Each constituent encoder is considered a double binary recursive systematic convolutional encoder. It is called double binary as it has two inputs at the same time. It is considered recursive due to the feedback connection in the convolutional encoder. This feedback leads to that this encoder has an infinite impulse response. Each output depends not only on the current input, but also on all previous input bits.

Double binary Turbo coding has some benefits over ordinary binary Turbo codes, as explained in [17]. These benefits can be summarized as:

1- The substitution of binary codes by double-binary codes has a direct incidence on the erroneous paths in the trellis, which leads to a lowered path error density and reduces correlation effects in the decoding process. This leads to better performance.
2- From hardware implementation point of view, the bit rate at the decoder output is twice that of a binary decoder as the processing is performed on two bits at the same time. So, higher throughput can be achieved with an equivalent complexity per decoded bit.
3- For a certain block size, the latency of the decoder is divided by 2.

In Figure 3.1, it is shown the block diagram of the convolutional Turbo encoder. The figure describes the constituent encoder which has a constraint length of 4, two inputs and two outputs.

Polynomials that define outputs are:

- For Feedback branch: $1+D+D^3$
- For Y parity: $1+D^2+D^3$
- For W parity: $1+D^3$

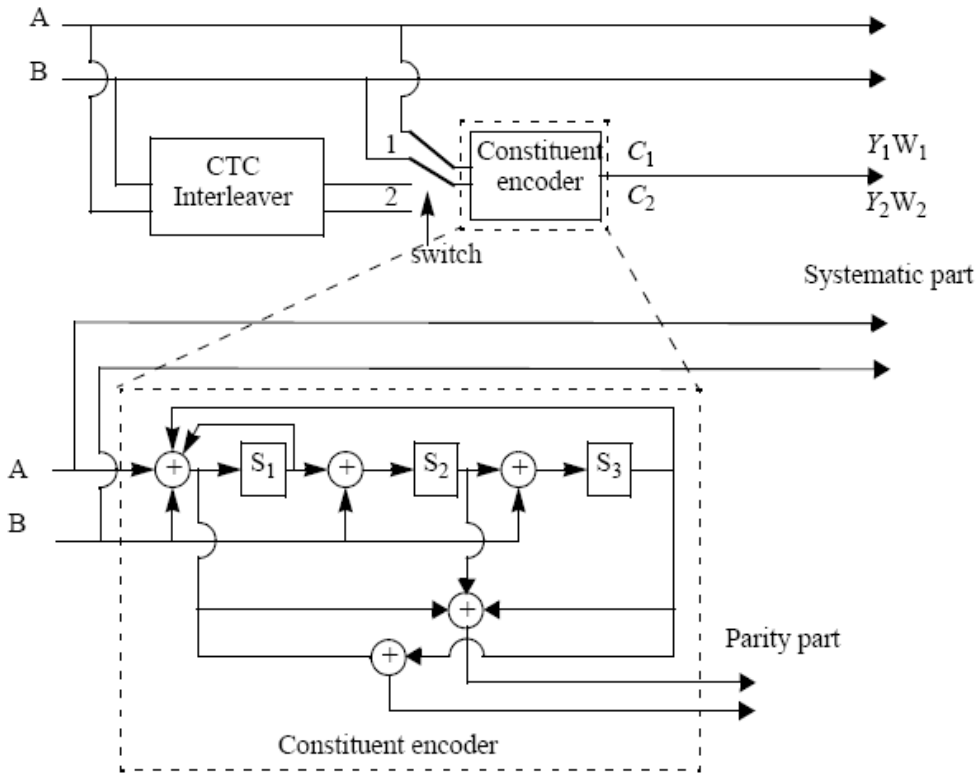


Figure 3.1 CTC encoder structure

3.2.2 CTC Interleaver

The CTC interleaver specified in IEEE802.16e consists of two permutation steps, one is a permutation on the level of each symbol individually, and the

second is on the level of the sequence of all symbols. The following sub-sections illustrate the interleaving operations.

3.2.2.1 Switch alternate couples

In this step, inputs *A, B* are sent in their order one time, swapped for the next time. This operation is repeated for the whole block.

Let the input sequence be $U_0 = [(A_0, B_0), (A_1, B_1), (A_2, B_2), \ldots..(A_{N-1}, B_{N-1})]$. The output of this step is $U_1 = [(A_0, B_0), (B_1, A_1), (A_2, B_2), \ldots.(B_{N-1}, A_{N-1})]$, Where N is the block size of input to interleaver.

The above operation is described as follows:

for $i=0$ to $N-1$

If$(i \bmod 2 == 1)$

$(Ai, Bi) \longrightarrow (Bi, Ai)$ List 3.1

3.2.2.2 Calculate interleaved order of sequence U_1

The sequence U_1 obtained in the previous step should be mapped to a new sequence U_2. Mapping is carried out by the function $P(j)$ defined such that:

$U_2(j) = U_1(P(j))$.

The operation is described as follows:

for $j = 0 \ldots N-1$

switch $j \bmod 4$:

Case 0:

$P(j) = (P_0 . j+1) \bmod N$

Case 1:

$P(j) = (P_0 . j+1+N/2+P_1) \bmod N$

Case 2:

$P(j) = (P_0 . j+1+P_2) \bmod N$

Case 3:

$P(j) = (P_0 . j+1+N/2+P_3) \bmod N$ List 3.2

The output sequence of the interleaver is given as $U_2 = [U_1(P(0)), U_1(P(1)), \ldots\ldots U_1(P(N-1))]$. This will be the input to the second constituent encoder. The mentioned parameters P_0, P_1, P_2 *and* P_3 are specified in the standard. They depend on block size N.

The above procedure calculates the sequence of interleaved bits $P(j)$ from the original sequence j. In case of 802.16e, the input stream of bits should be read by the interleaver with the interleaved sequence $P(j)$. Then the new sequence is outputted linearly. A detailed hardware description will be given in chapter 5.

3.2.3 Determination of Circulation states

In case of ordinary convolutional encoders, tail bits are included at the end of each block to force trellis diagram to reach zero state. In case of turbo codes, such a tail biting scheme can not be used due to the recursive nature of constituent encoders used in turbo encoders, Padding with zeros will not ensure reaching to zero state. On the other hand, if we can perform this to one constituent encoder, we can not perform it to the two constituent encoders simultaneously. A tail biting scheme used in turbo codes is called circular coding. It ensures that for a certain input sequence with a certain block size, there exists a certain state which is called circulation state (Sc) such that if we begin encoding with initial state Sc, we will ensure that final state at the end of the block is also Sc.

The circulation state Sc is specified from a look up table provided by the standard. In our case, we have 8 states ($0 \leq S \leq 7$). As we have two constituent encoders, we calculate two circulation states Sc_1, Sc_2.

The circulation states Sc_1, Sc_2 are determined by the following operations:

1) Initialize the encoder with state 0. Encode the sequence in the natural order for the determination of $Sc1$ or in the interleaved order for determination of $Sc2$. In both cases the final state of the encoder is $S0_{N-1}$

2) According to the length N of the sequence, determine $Sc1$ or $Sc2$ as given in Table 3-1.

Table 3-1 Circulation state (Sc) look up table

$Nmod_7$	$S0_{N-1}$							
	0	1	2	3	4	5	6	7
1	0	6	4	2	7	1	3	5
2	0	3	7	4	5	6	2	1
3	0	5	3	6	2	7	1	4
4	0	4	1	5	6	2	7	3
5	0	2	5	7	1	3	4	6
6	0	7	6	1	3	4	5	2

3.2.4 Subpacket generation

The next step after encoding is to generate subpackets with various coding rates depending on channel conditions; the 1/3 CTC encoded codeword goes through interleaving block then puncturing is performed to generate subpackets.

3.2.4.1 Symbol separation

All of the output symbols of the encoder are demultiplexed into six subblocks denoted *A, B, Y1, Y2, W1* and *W2* with the first N encoder output symbols going to the *A* subblock, the second N encoder output going to the *B* subblock, the third to the *Y1* subblock, the fourth to the *Y2* subblock, the fifth to the *W1* subblock, the sixth to the *W2* subblock.

3.2.4.2 Subblock interleaving

Puncturing specified by the standard depends on selection of consecutive symbols out of the whole 6*N* symbols of one subpacket. In order to perform puncturing to non-consecutive symbols, another permutation is carried out via subblock interleaving block. The purpose of this step is to interleave each of the six subblocks separately. The sequence of the interleaver output symbols is generated by a procedure specified by the standard. It resembles any ordinary interleaver where input symbols are written into an array with a certain order and then are read from that array with a different order. In this case, symbols are written in an order from 0 to N-1, then read out from an order with the i^{th} symbol is read from address ADi (i=0…N-1).
The procedure is constructed as follows:

1- Determine the subblock interleaver parameters, *m* and *J* that depend on the block size. They are given in Table 3-2

2- Initialize *i* and *k* to 0.

3- Form a tentative output address T_k according to the formula

$$T_k = 2^m (k \bmod J) + BRO_m\left(\left\lfloor \frac{k}{J} \right\rfloor\right) \qquad (3.1)$$

where $BRO_m(y)$ indicates the reversed m-bit value of y, (i.e $BRO_m(6)=3$).

4- If T_k is less than *N* then $ADi = T_k$ and increment *i* and *k* by 1. Otherwise, discard T_k and increment *k* only.

5- Repeat steps 3 and 4 until all *N* interleaver output addresses are obtained.

Table 3-2 Parameters for the subblock interleavers

Block size (bits) N_{EP}	N	Subblock interleaver parameters	
		m	J
28	24	3	3
72	36	4	3
96	48	4	3
144	72	5	3
192	96	5	3
216	108	5	4
240	120	6	2
288	144	6	3
360	180	6	3
384	192	6	3
432	216	6	4
480	240	7	2

3.2.4.3 Symbol grouping

The output of subblock interleaver shall consist of *A* subblock, *B* subblock, a symbol by symbol multiplexed block of *Y1* and *Y2* and finally a symbol by symbol block of *W1* and *W2*. This output sequence should be punctured in the following step, symbol selection (puncturing). Figure 3.2 illustrates the process of sub-block interleaving, symbol grouping and symbol selection.

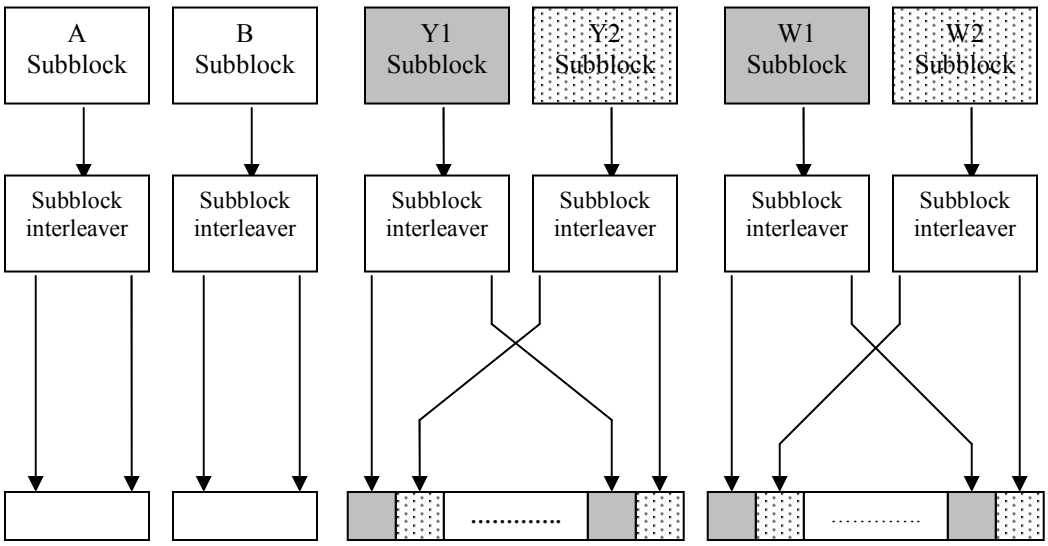


Figure 3.2 Block diagram of the interleaving and symbol grouping

3.2.4.4 Symbol selection (Puncturing)

The last step in Turbo encoding is symbol selection. Its output is a punctured subpacket with various possible coding rates. This rate depends on different parameters and it should be configured according to channel conditions. The selected symbols indices depend on:

N_{EP}: Number of bits in the encoder packet (before encoding).

N_{SCHk}: Number of concatenated slots of K^{th} subpacket.

m_k: the modulation order for the K^{th} subpacket (m_k = 2 for QPSK, 4 for 16-QAM, and 6 for 64-QAM).

$SPID_k$: Subpacket ID for the K^{th} subpacket, (for the first subpacket, $SPID_{k=0}$ = 0).

The index of the i-th symbol for the K^{th} subpacket shall be

$$S_{K,i} = \left(F_K + i\right) \bmod \left(3.N_{EP}\right) \tag{3.2}$$

Where

$$i = 0,1,2.....L_{K-1}$$

$$L_k = 48.N_{SCHk}.m_k$$

$$F_k = \left(SPID_k.L_k\right) \bmod \left(3.N_{EP}\right) \tag{3.3}$$

In case of HARQ support, K represents sub-packet ID. It is considered 0 in case of non HARQ support. In this case, Equation (3.3) is reduced to this formula

$$S_{K,i} = i \bmod (3.N_{EP}) \tag{3.4}$$

At the end of this step, the punctured sub-packet is available and we have the final output of Turbo encoder.

The above form of equation can be simplified as follows

$$i = 0,1,2.....\frac{2N}{code\_rate} - 1$$

$$F_k = \left(SPID_k.\frac{2N}{code\_rate}\right) \bmod \left(6N\right)$$

$$S_{k,i} = \left(F_k + i\right) \bmod \left(6N\right) \tag{3.5}$$

The term F_k represents an offset from the beginning of the subpacket, and the selected symbols have indices begins with (F_k) mod_{6N} to $\left(F_K + \frac{2N}{code\_rate} - 1\right)$ mod_{6N}.

This process is illustrated in Figure 3.3.

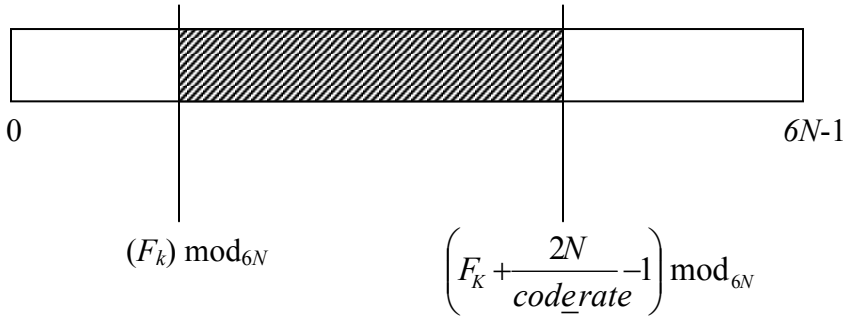


Figure 3.3 CTC Puncturing process

3.3 Turbo decoding

3.3.1 Introduction

Most proposed turbo decoding schemes are based on iterative decoding. The turbo decoder consists of two component decoders as indicated in Figure 3.4. The key idea on which iterative decoding is based on is that each decoder produces a soft estimate of the original information bits, this estimation is used by the other decoder, to produce a better estimation. The new estimation is used again by the first decoder to enhance its estimation and so on. The estimation is better with the increase of the number of iterations.

Each component decoder is based on soft input soft output decoding. The soft representation of the information bits is carried out in a form of a Log Likelihood Ratio (LLR). The soft output of each decoder provides a-priori probability of the information bits to be used by the other decoder. The a-priori information is also called extrinsic information.

Each component decoder operation is based on the received systematic, and parity bits from the channel, in addition to the extrinsic information from the other decoder. At the beginning of the first iteration, the decoder has no a-priori information about information bits. It has only channel information on systematic and parity bits. Thus, the input a-priori information is set initially to zero. The extrinsic information generated by each decoder is the key difference among successive iterations.

Many algorithms were proposed for turbo coding such as Max A-posteriori (MAP) [18] and Soft output Viterbi algorithm (SOVA). Each is based on iterative decoding where performance increases with the increase of number of iterations.

Increasing number of iterations introduces a complexity in implementation of decoder. A compromise should be held between Hardware implementation complexity and required performance.

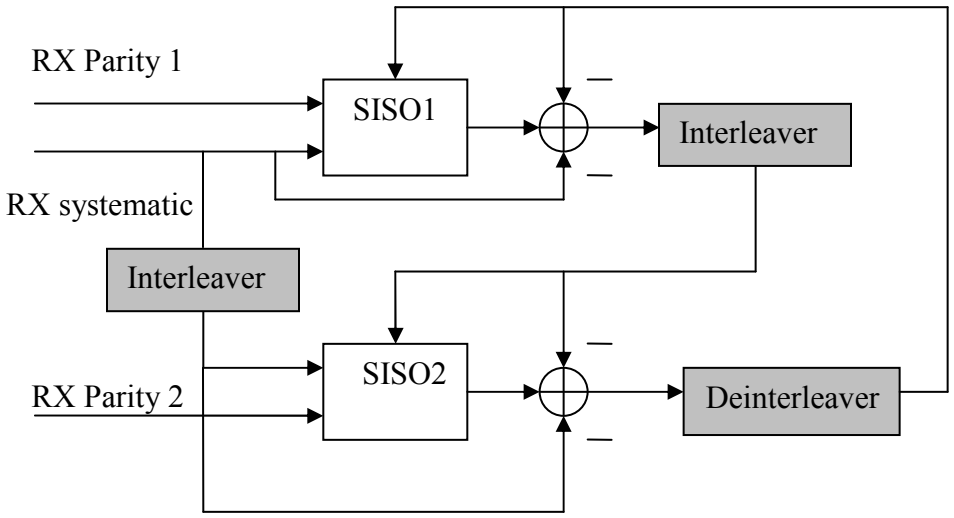


Figure 3.4 Generic Architecture of Turbo decoder

3.3.2 Log Likelihood Ratio (LLR)

The soft output of each decoder is based on LLR. In case of ordinary binary turbo codes, and for a certain data bit u_k, the LLR $L(u_k)$ is defined as the logarithm of the ratio of probability that u_k=+1 to the probability that u_k =-1. This means the ratio between a-priori probabilities.

$$L(u_k) = \ln\left(\frac{P(u_k = +1)}{P(u_k = -1)}\right) \tag{3.6}$$

Unlike LLR, the conditional LLR $L(u_k \mid y)$ is commonly used in decoding techniques. It is based on the ratio of a-posteriori probabilities. Its equation is given as follows

$$L(u_k \mid y) = \ln\left(\frac{P(u_k = +1 \mid y)}{P(u_k = -1 \mid y)}\right) \tag{3.7}$$

where y is the received codeword. This ratio of the a-posteriori probabilities will be used by the decoder to provide soft representation of the decoded bits.

However, we deal with the case of double binary Turbo decoding. In this case, we are in need to define a symbol based LLR. In this case, three LLRs are defined as follows

$$L(u_k(a,b)\,|\,y) = \ln\left(\frac{P((u_k = a,\ b)\,|\,y)}{P((u_k = -1,-1)\,|\,y)}\right) \tag{3.8}$$

This equation defines three LLRs corresponding to the set of input $u_k = (a,b)$ corresponding to $(a,b) = (-1,+1)$, $(+1,-1)$, or $(+1,+1)$ respectively. They are normalized with respect to $P(u_k) = ((-1,-1)\,|\,y)$. These LLRs are used in double binary turbo codes as an alternative to the LLR defined in (3.8) used in ordinary binary turbo codes. As a consequence, three extrinsic likelihood ratios are produced by each component decoder to be used by the other decoder.

3.3.3 Maximum A-posteriori probability (MAP) algorithm

The MAP algorithm was first proposed by Bahl, Cocke, Jelinek, and Raviv in 1974. It is also named as BCJR algorithm due to the names of its inventors. This algorithm aims at maximizing the a-posteriori probability at each time slot [18]. This differs from the case of Viterbi algorithm that is used with ordinary convolutional codes, which minimizes the probability of error for the whole path in the trellis. In the next section, the decoding process of ordinary binary turbo decoding is described, and then we will apply it to our case of double binary turbo decoding.

MAP algorithm is a Soft Input Soft Output (SISO) algorithm. It not only provides a decision for the decoded bit, but it can also provide a soft estimation of it, which is used by the other component decoder.

The decoding process is based on LLR as follows, Equation (3.8)can be written as

$$L(u_k \mid y) = \ln\left(\frac{P(u_k = 1 \mid y_0 y_1 \ldots y_{N-1})}{P(u_k = 0 \mid y_0 y_1 \ldots y_{N-1})}\right) \tag{3.9}$$

where N represents the block size of the received codeword. The probability of the original bit to be either zero or one depends on the whole codeword. It can be seen from a different point of view if the codeword is divided into three parts. The received codeword before the time slot k, $y_{j<k}$, the received codeword at time slot k, y_k and the received codeword after the time slot k, $y_{j>k}$.

Each time slot is represented by a set of transitions among states as shown in Figure 3.5. These are specified by the trellis diagram which depends on the structure of the encoder.

Consider at time slot k, the transition from state *s'* to state *s*, some transitions corresponds to u_k=+1 and the others corresponds to u_k=-1.

According to [19], We can rewrite equation 3-8 as follows

$$L(u_k \mid y) = \ln\left(\frac{\sum_{s' \to s \Rightarrow u_k = +1} P(S_{k-1} = s' \wedge S_k = s \wedge y)}{\sum_{s' \to s \Rightarrow u_k = -1} P(S_{k-1} = s' \wedge S_k = s \wedge y)}\right) \tag{3.10}$$

where the notation $\wedge$ means intersection. Equation (3.9) illustrates that the a-posteriori probability at a given time slot can be expressed by the sum of probabilities of transitions from state *s'* to state *s* corresponding to the information bit u_k.

We can expand the probability term $P(S_{k-1} = s' \wedge S_k = s \wedge y)$ as mentioned into equation 5.19 of [19]. We conclude that

$$P(S_{k-1} = s' \wedge S_k = s \wedge y) = P(y_{j<k} \mid s).P([y_k \wedge s] \mid s').P(s' \wedge y_{j>k}) \tag{3.11}$$

$$P(S_{k-1} = s' \wedge S_k = s \wedge y) = \alpha_{k-1}(s').\gamma_{k-1 \to k}(s' \to s).\beta_k(s) \tag{3.12}$$

- The term $\alpha_{k-1}(s')$ is called the Forward estimation of state probability of state *s'* at time slot *k*-1.
- The term $\gamma_{k-1\to k}(s'\to s)$ is called Branch metric probability or the transition probability from state *s'* to state *s* between time slots *k*-1 and *k*.
- The term $\beta_k(s)$ is called Backward estimation of state probability of state *s* at time slot *k*.

So, in order to calculate LLR, we need to calculate the previous three probabilities for each transition, and then LLR is calculated as mentioned in equation (3.10).
The next section presents a detailed explanation of calculation of each of the three probabilities in MAP algorithm.

3.3.3.1 Branch Metric Calculation

The branch metric $\gamma_{k-1\to k}(s'\to s)$ indicates the probability of transition on each branch for all branches of the corresponding trellis at a certain time slot.
As indicated from(3.11), (3.12)

$$\gamma_{k-1\to k}(s'\to s) = P([y_k \wedge s] \mid s') \tag{3.13}$$

This probability can be represented as a product of two probabilities, as mentioned in 5.32 of [19]. These probabilities are the channel probability and the A-priori probability.

$$\gamma_{k-1\to k}(s'\to s) = P(y_k \mid x_k).P(u_k) \tag{3.14}$$

Where y_k represents the received codeword at time instant *k*. It consists of the received systematic and parity bits, x_k represents the original transmitted systematic and parity bits corresponding to each branch in the trellis. The term u_k represents original information bit at time slot *k*. It is illustrated from (3.14) that branch metric probability is determined by the probability of transition on this branch, which is determined by the channel probability in addition to the

probability of original information bit corresponding to this branch at this time slot, which is the a-priori probability.

The channel probability is based on the information from received systematic and parity bits. It can be shown in a Gaussian channel with variance σ^2 and fading amplitude a that

$$P(y_k \mid x_k)\alpha^{al} \exp\left(\frac{L_C}{2}\sum_{m=1}^{n} y_{km}x_{km}\right) \tag{3.15}$$

where the term Lc is called channel reliability which depends on both SNR and fading amplitude as given in [19] as follows

$$L_c = 2a\frac{E_b}{\sigma^2} \tag{3.16}$$

Where E_b is the transmitted energy per bit and a is the fading amplitude.

Finally, we can represent the branch metric as the path metric used in conventional viterbi decoder in addition to the a-priori probability as shown below:

$$\gamma_{k-1\to k}(s' \to s)\alpha^{al} \exp\left(\frac{L_C}{2}\sum_{m=1}^{n} y_{km}x_{km}\right).P\left(u_k\right) \tag{3.17}$$

3.3.3.2 Forward estimation state probabilities

In addition to branch metric probability mentioned in the previous section, MAP algorithm takes into consideration state probabilities. Forward estimation of state probabilities indicates probability of each state in case of moving in the forward direction in the trellis diagram, i.e at each time slot forward state probability of each state means the probability that transition in this time slot begins from this state given the received codeword prior to this time slot.

This is given as mentioned in (3.11), (3.12) as

$$\alpha_{k-1}(s) = P(y_{j<k} \mid s)$$

Calculation of a state probability α_k at a certain time slot k depends on state probabilities $\alpha_{k-1}(S')$ of previous time slot and the transition probabilities, which are the branch metrics.

Calculation of this probability, as indicated in [19], is given by the recursive formula:

$$\alpha_k(s) = \sum \alpha_{k-1}(s').\gamma_{k-1\to k}(s' \to s) \tag{3.18}$$

In Figure 3.5, it is shown the trellis diagram of Turbo encoder used in IEEE802.16e WiMAX. As this standard uses double binary turbo codes, each state has four output branches.

In order to calculate forward state probability of state 0 at time slot k, we get it as

$$\begin{aligned}\alpha_k(0) = {} & \alpha_{k-1}(0).\gamma_{k-1\to k}(0 \to 0) + \alpha_{k-1}(1).\gamma_{k-1\to k}(1 \to 0) \\ & + \alpha_{k-1}(6).\gamma_{k-1\to k}(6 \to 0) + \alpha_{k-1}(7).\gamma_{k-1\to k}(7 \to 0)\end{aligned} \tag{3.19}$$

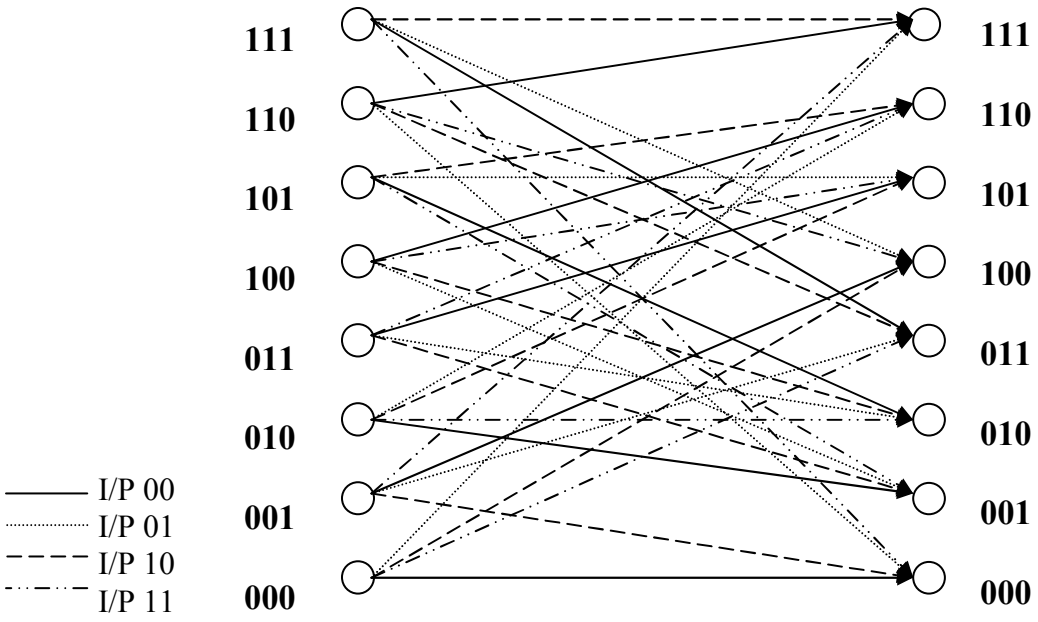


Figure 3.5 Trellis diagram of Double binaryTurbo encoder used in IEEE802.16e WiMAX

Initially, at the first decoding iteration, no a-priori information is given about state probabilities. In this case, we consider them equiprobable.

This means that

$$\alpha_0(s) = \frac{1}{n} \forall s \tag{3.20}$$

where n is the number of states, which equals 8 states in our case.

As circular coding is used as mentioned in 3.2.3, the initial state Sc is well known.

State probabilities should be initialized as follows

$$\begin{aligned} &\alpha_0(Sc) = 1 \\ &\alpha_0(S \neq Sc) = 0 \end{aligned} \tag{3.21}$$

3.3.3.3 Backward estimation state probabilities

Backward state probability of a certain state at a certain time slot indicates probability of transition to this state given a certain received codeword after this time slot. The calculation of the backward state probabilities is similar to that of forward state probabilities; it depends of state probabilities at the next time slot and branch metrics.

It is calculated by the recursive formula given below:

$$\beta_k(s) = \sum \beta_{k+1}(s').\gamma_{k \to k+1}(s' \to s) \tag{3.22}$$

Initializing backward state probabilities is similar to the case of forward state probabilities. This is given as described below:

$$\begin{aligned} &\beta_N(Sc) = 1 \\ &\beta_N(S \neq Sc) = 0 \end{aligned} \tag{3.23}$$

3.3.3.4 LLR Computation

The final step after calculation of the branch metrics and state probabilities at each time slot of the codeword is to calculate the LLRs. These LLRs represent the decoder soft output. We can re-write equation (3.10) as follows

$$LLR = \ln\left(\frac{\sum_{s'\to s\Rightarrow uk=+1} \alpha_{k-1}(s').\gamma_{k-1\to k}(s'\to s).\beta_k(s)}{\sum_{s'\to s\Rightarrow uk=-1} \alpha_{k-1}(s').\gamma_{k-1\to k}(s'\to s).\beta_k(s)}\right) \quad (3.24)$$

The output decoded bits can be calculated from LLRs by applying a hard decision to these soft values.

As turbo decoders are based on iterative decoding, the extrinsic likelihood probabilities are calculated from LLRs. Extrinsic likelihood represents how much information the decoder adds about the decoded bits. It is obtained by subtracting the input values to the decoder from its output LLRs as follows

$$L_e(u_k) = LLR - L_C.y_{ks} - L(u_k) \quad (3.25)$$

The above equation indicates the calculation of extrinsic LLR.

Where LLR is the soft output Log Likelihood Ratio from the decoder

Lc channel reliability

y_{ks} is the received systematic bit

$L\ (u_k)$ is the input A-priori probability

The extrinsic LLR should be bypassed to the other component decoder as an A-priori probability used in next iteration. A schematic description of calculation of extrinsic LLR is shown in Figure 3.6.

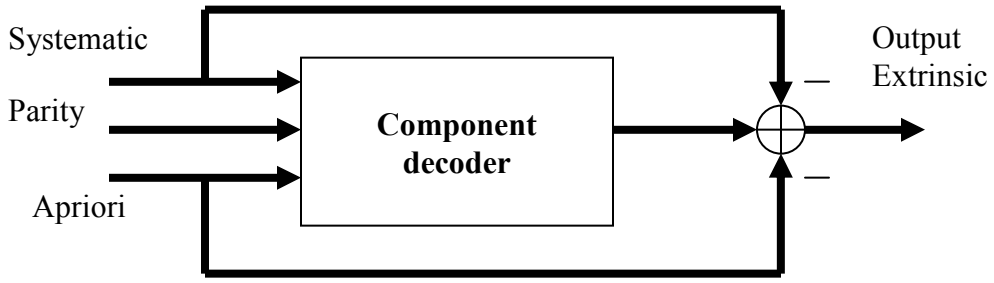


Figure 3.6 Extrinsic Likelihood calculation

3.3.3.5 Estimation of Circulation state

One important step is to estimate the circulation state (Sc) for each codeword. Several techniques were proposed to estimate Sc, Some techniques proposed to use a prologue decoder for estimation and another decoder to decode again after the identification of Sc. This solution adds more complexity for implementation, as it will increase latency, power consumption, area and resources.

Other proposed techniques depend on the iterative nature of the decoder. This means that Sc is estimated inherently from one iteration to the next one. At the first iteration, the decoder has no information about Sc. It begins decoding assuming equiprobable forward and backward initial states. At the end of the first iteration, the decoder obtains a reasonable estimation of Sc; it begins decoding in second iteration assuming the Sc estimated from first one. At the end of the second iteration, the decoder obtains better estimation of Sc, and so on. The decoder begins next iteration assuming Sc estimated from previous iteration.

This is a reasonable method of estimation as it adds no more complexity in hardware implementation. The estimation is based on maximizing the sum of forward state probability at the last time slot and backward state probability at first time slot as follows

$$Sc = \{S \Leftrightarrow \max(\alpha_N(S) + \beta_0(S))\} \tag{3.26}$$

3.3.4 Max Log MAP Approximation

It is shown that MAP algorithm includes enormous calculations of state and branch metric probabilities, including large number of multiplications, exponentials and Logarithm calculations which complicates the hardware

implementation. Simplification to MAP algorithm is necessary to simplify its implementation.

One possible approximation is to use state and branch metric probabilities in Log domain, this means using Log Number systems (LNS) as an alternative way to represent these probabilities. Using LNS converts all multiplications to additions and removes exponentials. This approximation is called Log-MAP approximation [20].

The state and Branch Metric probabilities are defined in LNS as follows:

$$\begin{aligned} \mathrm{A}_k(s) &= \ln\left(\alpha_k(s)\right) \\ \mathrm{B}_k(s) &= \ln\left(\beta_k(s)\right) \\ \Gamma_{k-1\to k}(s'\to s) &= \ln\left(\gamma_{k-1\to k}(s'\to s)\right) \end{aligned} \tag{3.27}$$

Using LNS is called Log MAP approximation; an extended simplification can be done by using MAX Log MAP approximation [20],[21] that depends on Jacobi logarithm approximation as indicated below:

$$\ln\left(\sum e^{x_i}\right) \approx \max\{x_i\} \tag{3.28}$$

3.3.4.1 Calculation of branch metric probabilities

The branch metric probability in log domain $\Gamma_k(S)$ is calculated as follows

$$\begin{aligned} \Gamma_k(S) &= \ln\left(\gamma_k(S)\right) \\ &= \text{const.} + \ln(\mathrm{P}(\mathrm{u}_k)) + \frac{L_C}{2}\sum_{m=1}^{n} y_{km}x_{km} \end{aligned} \tag{3.29}$$

The constant term can be omitted in the calculation of LLR, so no need to consider it.

If we define $L(u_k) = \ln(\frac{P(u_k = +1)}{P(u_k = -1)})$, the LLR of the a-priori probability, we obtain

$$L(u_k) = \ln(\frac{P(u_k = +1)}{1 - P(u_k = +1)}) \tag{3.30}$$

$$P(u_k = \pm 1) = (\frac{e^{-L(u_k)/2}}{1 + e^{-L(u_k)}}).e^{u_k L(u_k)} \tag{3.31}$$

Finally, we can represent the branch metric by the form given in (3.32)

$$\Gamma_k(S) = c\text{onst.} + \frac{1}{2}u_k L(u_k) + \frac{L_C}{2}\sum_{m=1}^{n} y_{km} x_{km} \tag{3.32}$$

3.3.4.2 Calculation of forward state metric probabilities

The recursive form of equation (3.18) can be rewritten in the log domain as

$$\mathrm{A}_k(S) = \max\{\mathrm{A}_{k-1}(S') + \Gamma_{k-1\to k}(S' \to S)\} \tag{3.33}$$

This means that in case of the Turbo code standard for which this thesis is concerned, the calculation of the state metric probability in LNS implies four additions to previous state metrics by corresponding branch metrics. The resultant state metric probability is the maximum of the four results. This has its significant effect on simplifying implementation of this algorithm with a little degradation in the system performance**.**

3.3.4.3 Calculation of backward state metric probabilities

In a similar manner to the calculation of forward state metrics, backward state metrics are computed. The recursive formula will be

$$\mathrm{B}_k(S) = \max\{\mathrm{B}_{k-1}(S') + \Gamma_{k-1\to k}(S' \to S)\} \tag{3.34}$$

Again, in this standard, calculation of backward state metrics implies four additions and comparison operation.

3.3.4.4 LLR Computation

In case of Max Log MAP, LLR given in (3.24) is computed by applying MAX Log MAP approximation taking into consideration that

$$\alpha_k(S) = e^{A_k(S)}$$

$$\beta_k(S) = e^{B_k(S)}$$

$$\gamma_{k-1\to k}(S' \to S) = e^{\Gamma_{k-1\to k}(S'\to S)}$$

In this case, we obtain

$$LLR = \ln\left(\sum_{s'\to s \Rightarrow uk=+1} \alpha_k(s').\gamma_{k-1\to k}(s'\to s).\beta_k(s)\right) - \ln\left(\sum_{s'\to s \Rightarrow uk=-1} \alpha_{k-1}(s').\gamma_{k-1\to k}(s'\to s).\beta_k(s)\right)$$

$$LLR = \max_{s'\to s \Rightarrow uk=+1}\left\{A_{k-1}(s')+\Gamma_{k-1\to k}(s'\to s)+B_k(s)\right\} - \left\{A_{k-1}(s')+\Gamma_{k-1\to k}(s'\to s)+B_k(s)\right\} \quad (3.35)$$

The computed LLR represents the soft output of the decoder. In order to calculate extrinsic LLR; equation (3.25) is used without any modifications.

Another factor is that the Max Log MAP algorithm removes the decoder dependency on SNR. This can be observed from (3.32), the SNR becomes a scaling factor multiplied by another term representing the cross correlation between received data and original data corresponding to each branch. Initially, the decoder has no a-priori information about the original information bit; thus $L(u_k)=0$.

Calculation of $A_k(S)$ and $B_k(S)$ indicates that they will also be scaled with the same scaling factor. This scaling factor will be scaled with all quantities used in

decoding. A scaling factor will not affect the decision performed in LLR. The term SNR can be omitted when calculating branch metric probabilities. The assumption for which this is based on is that SNR is constant over the same codeword.

Estimation of circulation states is the same as mentioned in section 3.3.3.5, except that initializing state metrics here is different. In this case

$$A_0(Sc) = 0$$
$$A_0(S \neq Sc) = -\infty \tag{3.36}$$

And

$$\mathrm{B}_N(Sc) = 0$$
$$\mathrm{B}_N(S \neq Sc) = -\infty \tag{3.37}$$

Another version of Log MAP algorithm is called MAX* Log MAP (MAX-STAR Log MAP) algorithm which add a correction term to the max approximation as follows

$$\ln(e^{x_1} + e^{x_2}) = \max(x_1, x_2) + f_c(x_1, x_2) \tag{3.38}$$

where $f_c(x_1, x_2)$ is the correction term added and equals to $\ln(1 + e^{-|x_1 - x_2|})$

When applying max* algorithm, the SNR term affects branch and state metrics calculation and it shouldn't be neglected.

3.3.5 Sliding Window Max Log MAP Approximation

In addition to MAX Log MAP approximation, further approximations were proposed to compensate for latency and large storage requirements for MAX Log MAP, especially for large block sizes. One proposed algorithm as mentioned in [22] is called Sliding Window (SW) MAX Log MAP algorithm.

The key idea behind sliding window approximation is to divide the received codeword into smaller windows or sub-blocks. No need to wait for the

whole codeword, but the backward recursion begins when first sub-block only is completely received. This plays a key role in reducing the storage requirements, no need to store branch metrics and state metrics for the whole codeword, but only for one sub-block. After the completion of reception of the first sub-block, it is ready to calculate the backward state probabilities and LLRs of symbols of the first sub-block. The forward probabilities of second sub-block are calculated simultaneously.

A timing sequence description of SW MAX Log MAP algorithm is provided in Figure 3.7. It shows the operation of how states are computed for different sub-blocks with time.

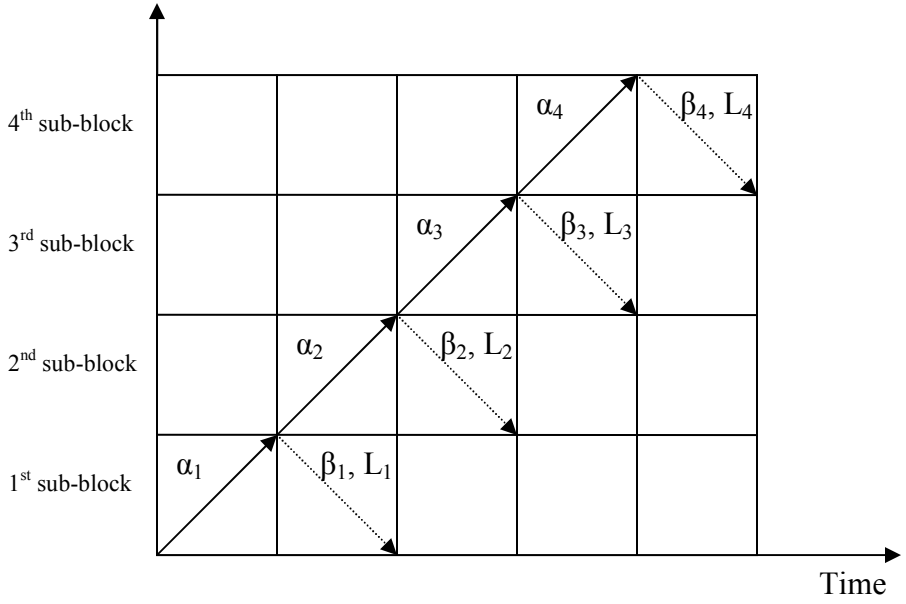


Figure 3.7 Timing Sequence of Sliding Window Max Log MAP

At the end of each sub-block, backward states are being calculated. A problem raises that no pre-estimation of values of state probabilities at the end of the window to initialize backward states. A possible solution is to assume equiprobable states at this time slot. This has its impact on degrading the system

performance. More about simulation results of these approximations are provided in chapter 4.

In order to overcome the effect of performance degradation, some proposed techniques use a guard window to have a rough estimation of initial value of backward state metrics. The guard window begins tracing back not from the end of the current window, but from a further time slot in the next window, this depends on the guard window size. As window size and guard window size increases, we have a better performance.

There are various techniques specified for sliding Window Max Log MAP algorithm, some techniques begin by computation of backward recursion of each sub-block, then compute forward recursion. Other techniques begin with forward recursion then calculate backward recursion at traceback. In this thesis the second type is considered in simulations and implementation. The steps of the considered sliding window Max Log MAP algorithm can be summarized as follows:

1- Begin calculation of Forward state probabilities by initializing

$$A_0(Sc) = 0$$
$$A_0(S \neq Sc) = -\infty$$

2- At the end of first sub-block, begin the backward recursion where backward states should be initialized as:

$$\mathrm{B}_{w+g}(S) = 0 \qquad \forall S$$

Where w is the window size and g is the guard window size. We begin backward recursion at end of each sub-block assuming equiprobable states.

3- Once backward recursion is calculated, LLRs can be calculated and then extrinsic LLRs can also be calculated. The resulting bits after decision should be stacked in order to obtain decoded bits in order.

4- The operation should be repeated for the next window, but initialization of forward state metrics is calculated in the same way of ordinary MAX Log MAP. The process of SW MAX Log MAP is shown in Figure 3.8.

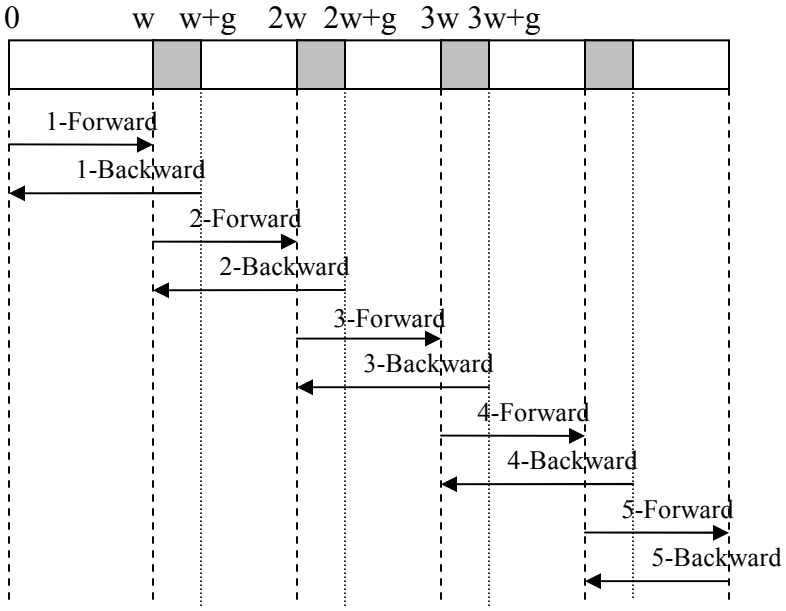


Figure 3.8 Sliding Window operation

3.3.6 Double binary Turbo decoding

Up to now, we consider the case of Binary Turbo Codes; in case of IEEE 802.16e WIMAX standard, it uses double binary Turbo codes. This section illustrates how the ordinary turbo decoding algorithms are modified to handle the case of double binary turbo codes. In case of binary turbo codes, each bit is represented by a single LLR, but in case of double binary turbo codes, we define three LLRs [23] as mentioned in (3.8). Each component decoder has input systematic and parity bits and three extrinsic LLRs. By applying this definition of LLRs, the decoder can perform decoding on a symbol wise operation without separating the couples of the symbol. A description of the decoder block is shown in Figure 3.9.

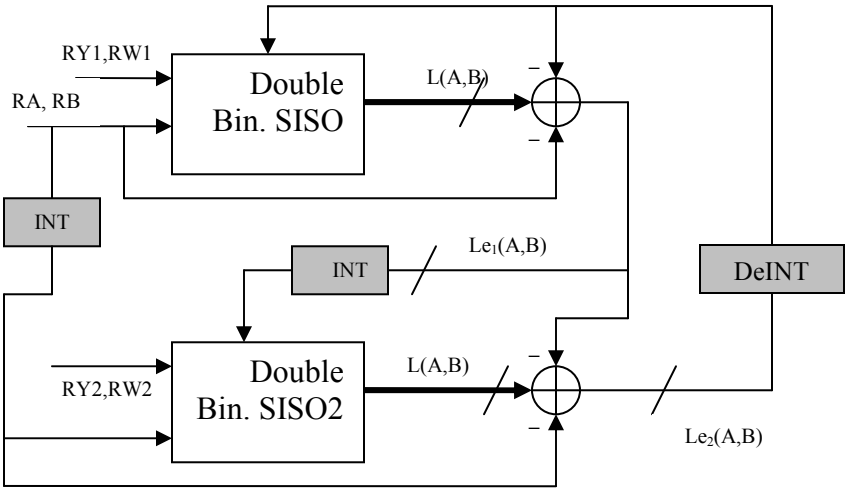


Figure 3.9 Structure of Double Binary Turbo decoder

Calculation of branch and state metrics is straight forward. Assume the received systematic bits are R_A and R_B and the received parity bits are R_{Y1}, R_{Y2}, R_{W1} and R_{W2}. The first component decoder has inputs R_A, R_B, R_{Y1}, R_{W1}, Le (0,1), Le (1,1) and Le (1,1).

To calculate branch metric at any time slot, a cross correlation is carried out between received data and original data corresponding to each branch.

$$\gamma_{k-1\to k}(A,B) = R_A * A + R_B * B + R_{Y1} * Y1 + R_{W1} * W1 + Le(A,B) \tag{3.39}$$

$$A,\ B,\ Y1,\ W1 \in \{1,-1\}$$

where A, B, Y1 and W1 are the original systematic and parity bits corresponding to each branch in the trellis.

Calculation of forward and backward metrics is straightforward as in the case of binary turbo codes. After the calculation of branch metrics, forward and backward metrics, the decoder should calculate LLRs by calculating the likelihood of each branch.

$$T_k(a,b) = \max_{S'\to S:(a,b)} \left\{ \mathrm{A}_k(S') + \Gamma_{k\to k+1}(S'\to S) + \mathrm{B}_{k+1}(S) \right\} \tag{3.40}$$

where $T_k(a,b)$ represents Likelihood of the branch that corresponds to transition from state *s'* to state *s* for original input sequence (a,b).

Finally, three LLRs are calculated as

$$L_k(a,b) = T_k(a,b) - T_k(0,0) \tag{3.41}$$

and we get that $L_k(0,0)$ always equals to zero.

After calculation of LLRs, three extrinsic LLRs $L^o_{e,k}(1,1)$, $L^o_{e,k}(1,0)$, $L^o_{e,k}(0,1)$ should be calculated to be bypassed to the other component decoder. The term $L^o_{e,k}(a,b)$ indicates output extrinsic likelihood of symbol $u_k = (a,b)$ at time slot *k*. The final decision of decoded bits is performed according to output LLRs obtained from (3.41)

$$L_k(A) = \max\ (T_k(1,0), T_k(1,1)) - \max\ (T_k(0,1) - T_k(0,0))$$

$$L_k(B) = \max\ (T_k(0,1), T_k(1,1)) - \max\ (T_k(1,0) - T_k(0,0)) \tag{3.42}$$

After Calculation of both $L_k(A)$, $L_k(B)$, we are able to estimate both original information bits $\hat{A}$, $\hat{B}$. This should be done at the last decoding iteration.

Chapter 4

Simulation results of WiMAX CTC

4.1 Introduction

This chapter contains several simulations and performance analysis of WiMAX CTC. These simulations compare between various Turbo decoding schemes and show the effect of decoding approximations on the system performance. In addition, they illustrate the effect of different channel conditions on the WiMAX CTC performance. Finally, we achieve the fixed point model which represents the system performance after Hardware implementation.

4.2 Turbo codes performance in AWGN channels

4.2.1 Effect of Number of iterations

As illustrated in chapter 3, Turbo decoding algorithms are based on iterative decoding. In this case, increasing the number of iterations provides an improvement in the original data estimation. Figure 4.1 illustrates the performance analysis of MAX Log MAP algorithm for a rate 1/3 turbo decoder with interleaver size of 240 couples over AWGN channel. It is simulated for a number of turbo iterations up to 8 iterations.

It is indicated from the simulation results that the increase in the number of iterations enhances the BER performance. It is obvious that the rate of BER enhancement decreases with the increase in the number of iterations. The BER curve begins to saturate with a large number of decoding.

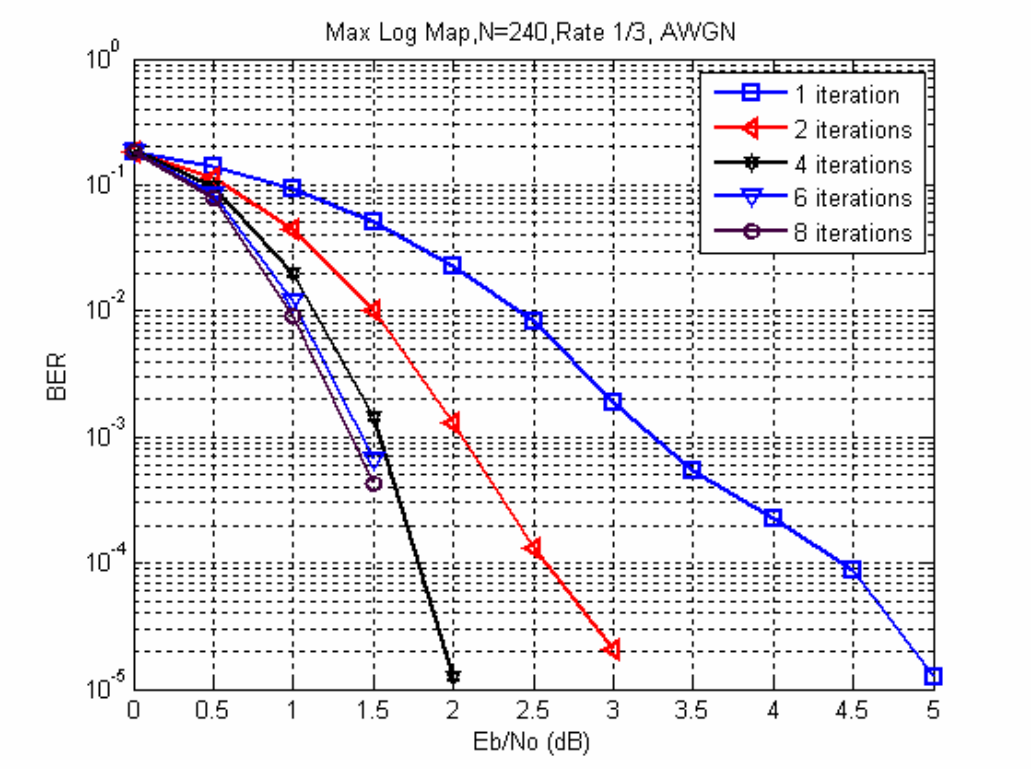


Figure 4.1 Effect of number of iterations in MAX Log MAP

We conclude that the increase in the number of iterations too much may be inefficient as the gain in performance will be insignificant with respect to the additional hardware complexity and decoding latency.

4.2.2 Improvement over mandatory Convolutional Coding

This section demonstrates the difference in performance between Convolutional Turbo codes and the ordinary Convolutional Codes used in mobile WiMAX. Simulation is performed in AWGN environment. It is shown that Convolutional Coding outperforms CTC for only the first CTC decoding iteration, while CTC outperforms Convolutional Coding beyond the first iteration. Figure 4.2 illustrates that 2 CTC decoding iterations achieves an enhancement of about 1 dB over Convolutional Coding and 8 CTC decoding iterations achieves an improvement of about 2 dB.

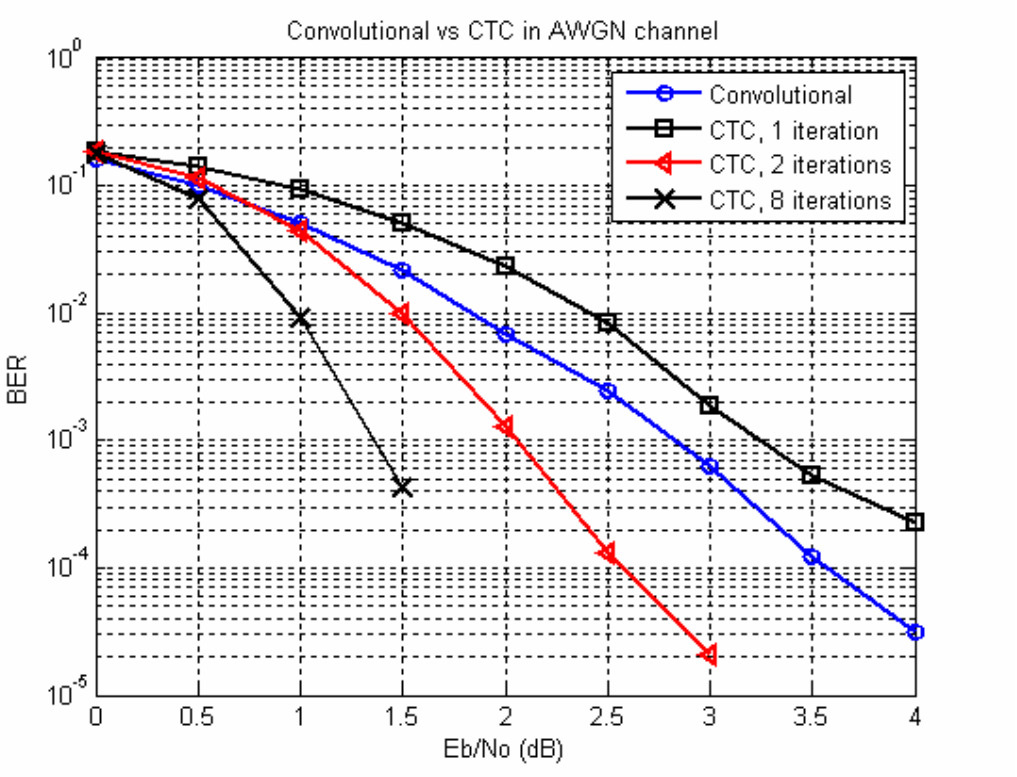


Figure 4.2 Convolutional vs CTC performance

These simulation results in Figure 4.2 derive an important conclusion. It is not efficient to use CTC decoder for a single decoding iteration. This leads to a lower performance and higher complexity. At least CTC should be designed for two iterations. Four decoding iterations can be considered a reasonable compromise between performance, complexity and latency.

4.2.3 Effect of Turbo interleaver block size

Simulation results indicate that Turbo codes performance varies according to the interleaver block size. It is shown that the increase of CTC interleaver size enhances the BER performance for the same SNR. Figure 4.3 illustrates the performance of MAX Log MAP algorithm for interleaver block sizes of 24, 96, 192 and 240 respectively. Simulation is performed for 4 turbo decoder iterations and coding rate of 1/3 in AWGN channel environment.

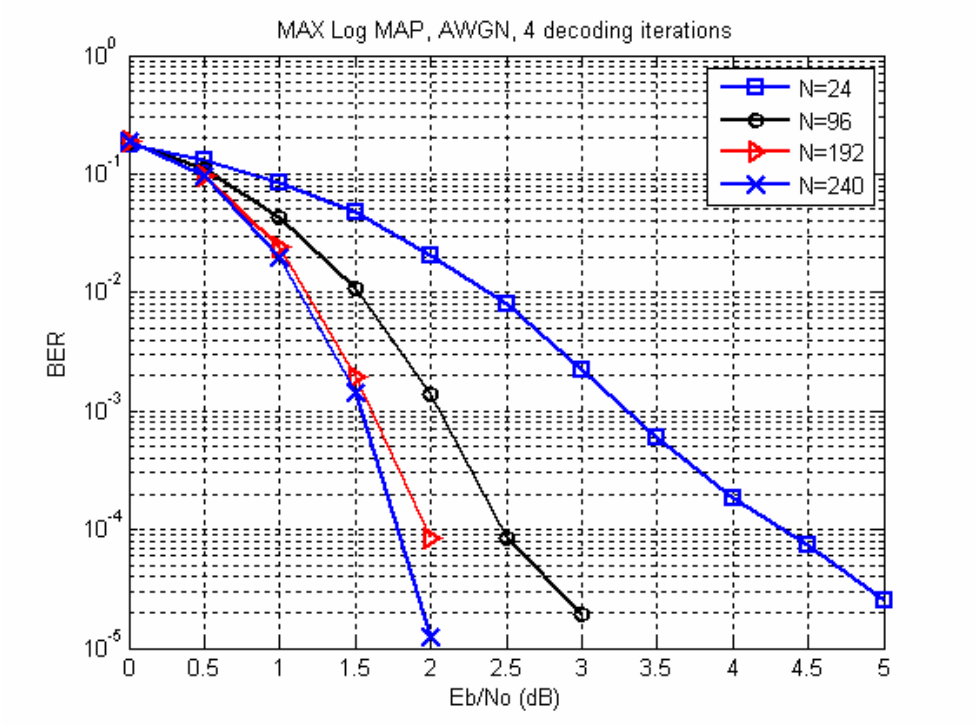


Figure 4.3 Interleaver block size effect

It is shown that in case of interleaver size of 240 couples, the performance outperforms that of lower sizes. Depending on the channel conditions and estimated SNR, the block size *N* is adjusted by the MAC layer in order to achieve the desired BER. The cost of BER enhancement is the decoding latency for larger block sizes.

4.2.4 MAX vs MAX* Log MAP

This section illustrates the effect of neglecting the correction term in MAX Log MAP algorithm. This correction term was previously mentioned in Figure 4.4. We present a comparison between MAX Log MAP algorithm with the MAX* Log MAP algorithm which considers the correction term. Simulation is performed for a block size *N* of 240, code rate of 1/3 and 4 decoding iterations in AWGN channel environment.

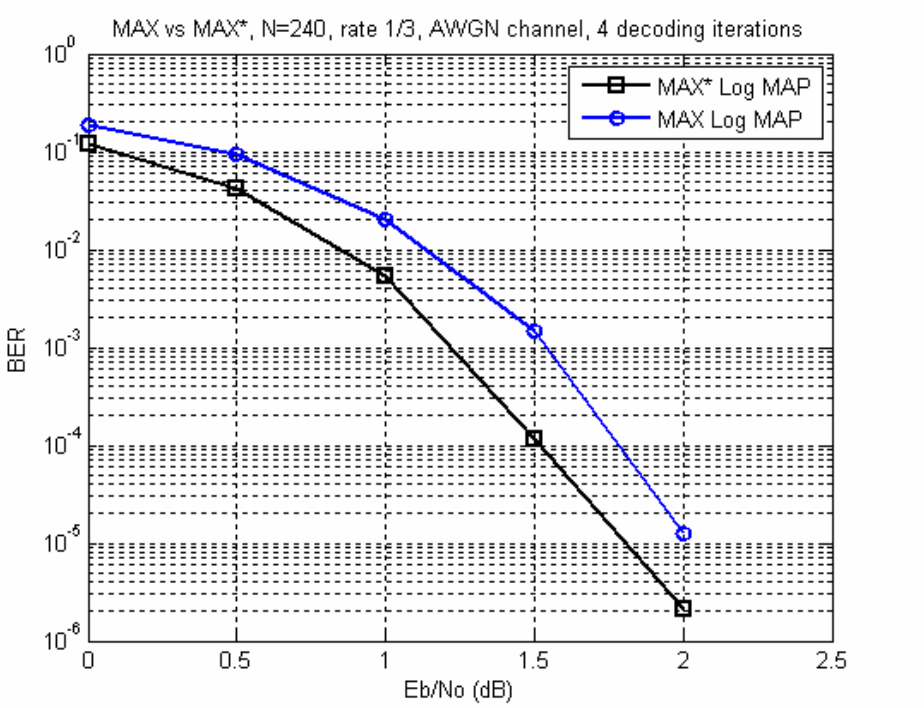
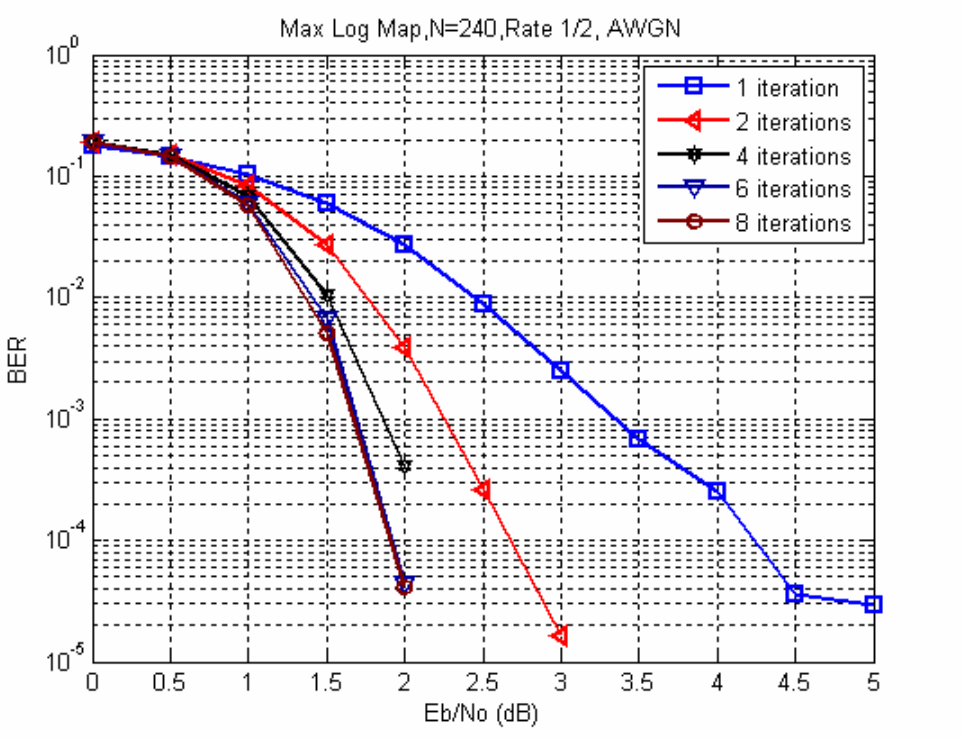


Figure 4.4 Comparison between Max and Max* performance

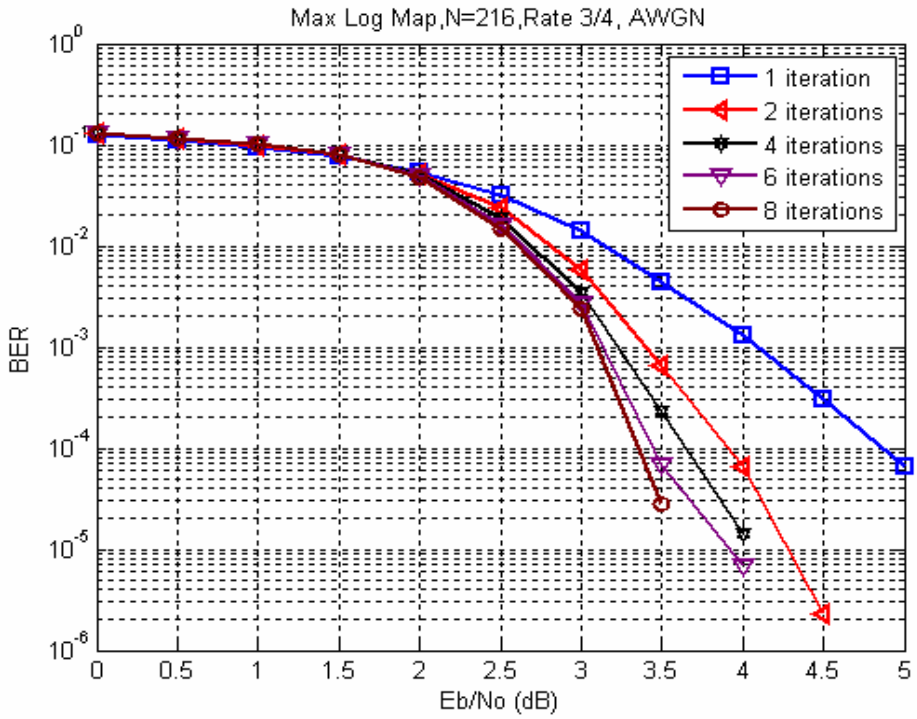
From the simulation results, we find that the MAX Log MAP approximation results in a loss of about 0.25 dB of the BER performance compared to MAX* algorithm.

4.2.5 Effect of Symbol selection (Puncturing)

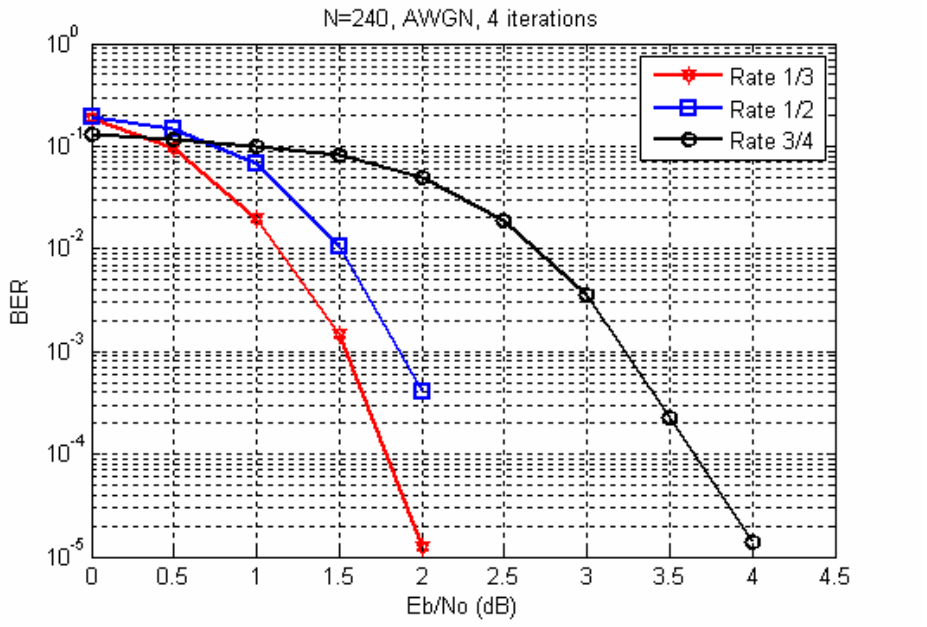
Symbol selection is performed to reduce number of coded bits per information symbol. Simulation results indicate that puncturing affects the BER performance of Turbo codes. In 802.16e CTC encoder, variable code rates of 1/2, 3/4, and 5/6 are defined. It is shown that the increase in the code rate results in a degradation of Turbo codes performance. The process of puncturing should be adaptive according to the channel conditions. Figure 4.5 illustrates the effect of symbol selection in case of Rate ½ and Rate ¾ coding respectively.


Max Log Map,N=240,Rate 1/2, AWGN
1 iteration
2 iterations
4 iterations
6 iterations
8 iterations
BER
Eb/No (dB)

(a)


Max Log Map,N=216,Rate 3/4, AWGN
1 iteration
2 iterations
4 iterations
6 iterations
8 iterations
BER
Eb/No (dB)

(b)



(c)

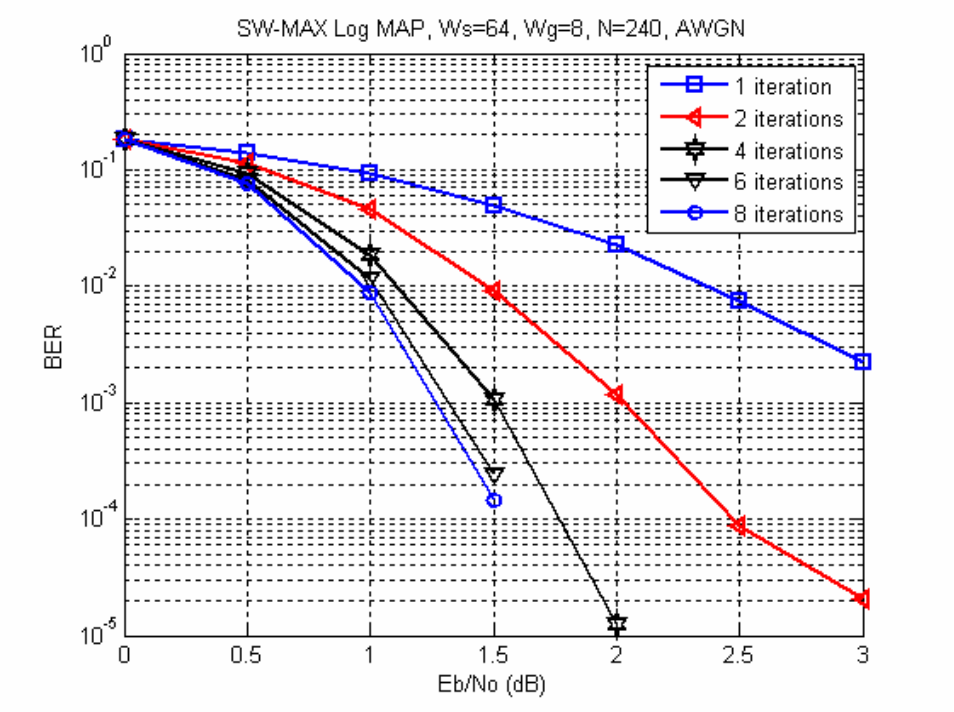
Figure 4.5 (a) Rate ½ performance

(b) Rate ¾ performance

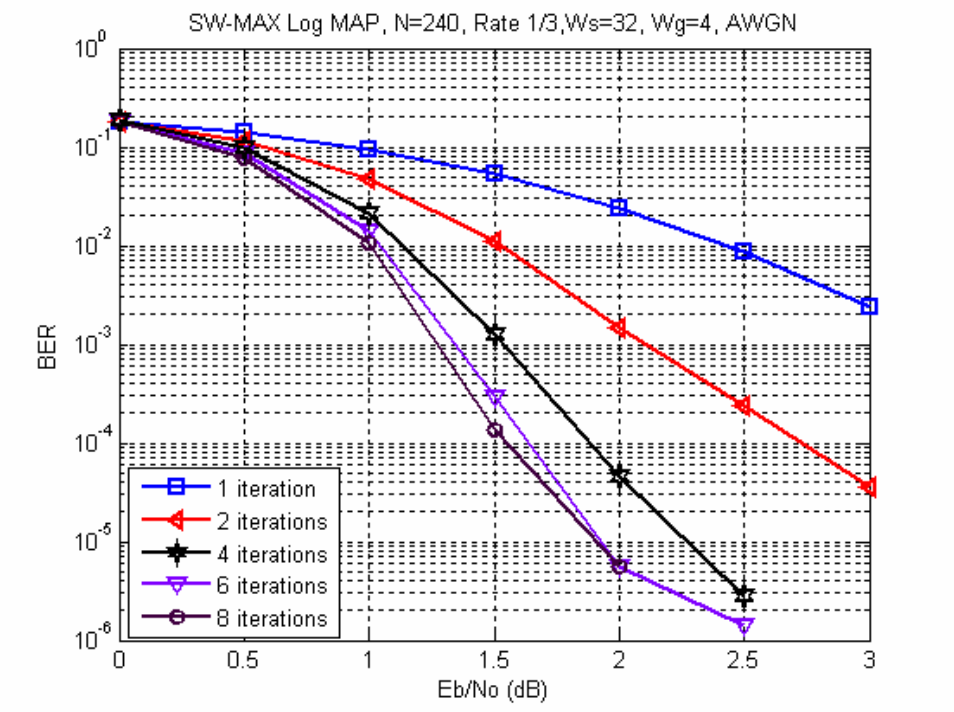
(c) Comparison among various Coding rates

4.2.6 Sliding Window MAX Log Map approximations

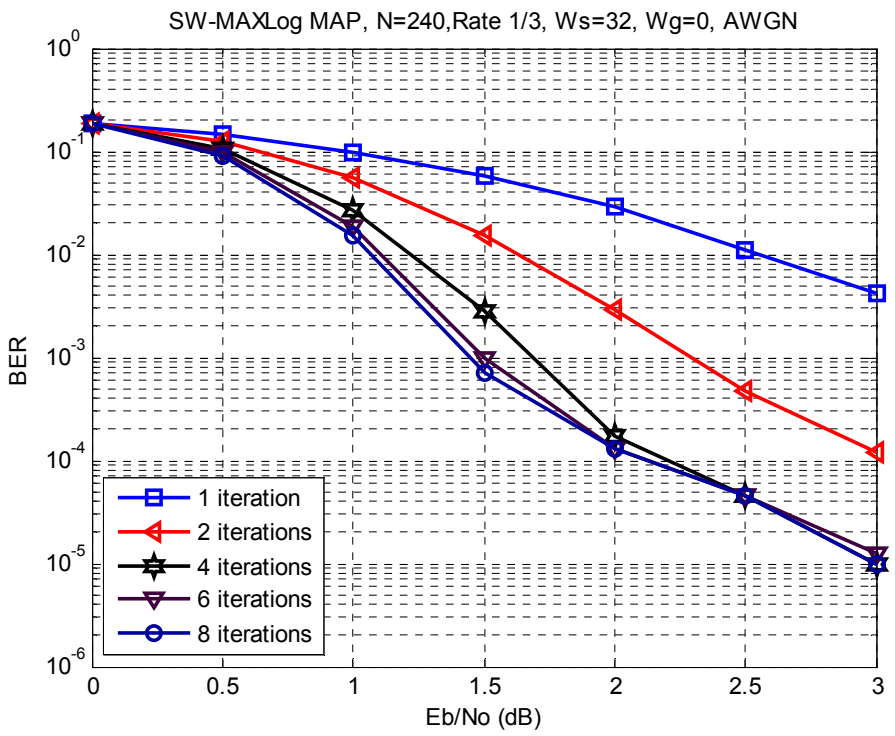
In this section, effect of Sliding window MAX Log MAP approximation is illustrated. The BER performance is tested for different window sizes (Ws) and guard window sizes (Wg). The simulation results are shown in Figure 4.6 a, b and c.


SW-MAX Log MAP, Ws=64, Wg=8, N=240, AWGN
BER
Eb/No (dB)
1 iteration
2 iterations
4 iterations
6 iterations
8 iterations

(a)


SW-MAX Log MAP, N=240, Rate 1/3,Ws=32, Wg=4, AWGN
BER
Eb/No (dB)
1 iteration
2 iterations
4 iterations
6 iterations
8 iterations

(b)



(c)

Figure 4.6 (a) BER for SW MAX Log MAP (Ws=64, Wg =8)
(b) BER for SW MAX Log MAP (Ws=32, Wg =4)
(c) BER for SW MAX Log MAP (Ws=32, Wg =0)

It is obvious that the system performance is exposed to some degradation with the change of the guard window size (Wg). In Figure 4.7, the effect of removing guard window degrades the system performance. The simulation is held for case of block size N=240, Window size Ws=32, and AWGN channel. The simulation results indicate that the case of Wg=0 increases the BER. This is due to total removal of the information of the backward metrics from some time slots.

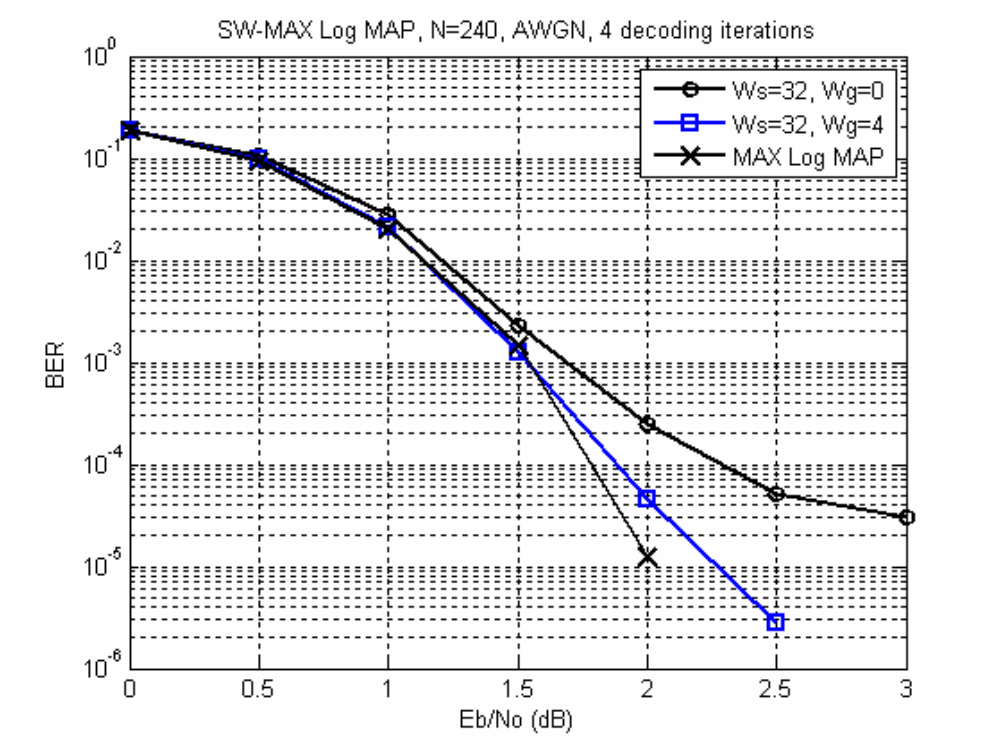


Figure 4.7 Guard Window effect

4.3 Simulations of Turbo codes in fading channels

As practical channels are not simply considered as AWGN channels, several channel models have been standardized to simulate the effects of practical channels on transmitted signals. It is important to study effect of Turbo codes in fading channels. This section provides several simulation outputs of Turbo codes in fading channels for different coding rates. Simulations is performed for both QPSK rate ½ and rate ¾ with OFDM, block size N=240 and MAX Log MAP decoding technique. The fading channel model used is that proposed for IEEE802.16m standard for urban macrocell. It models a NLOS propagation and high mobility (up to 350 Km/h) [24]. In this model, channel is modeled with 20 taps; each tap consists of a set of rays with fixed offset angles. The delay and power of each tap is also specified. Table 4-1 indicates these parameters. There are

other propagation models specified for IEEE 802.16m standard. For more details, please refer to [24].

Table 4-1 Proposed Channel characteristics for urban macrocell for IEEE 802.16m

Tap #	Delay(ns)	Power(dB)	Angle of departure (AoD)	Angle of arrival (AoA)
1	0	-6.4	61	-19.5
2	60	-3.4	44	-16.4
3	75	-2.0	-34	-15.0
4	145	-3.0	0	-13.0
5	150	-1.9	33	-14.9
6	190	-3.4	-44	-16.4
7	220	-3.4	-67	-13.4
8	335	-4.6	52	-17.7
9	370	-7.8	-67	-20.8
10	430	-7.8	-67	-20.8
11	510	-9.3	-73	-22.3
12	685	-12.0	-83	-25
13	725	-8.5	-70	-21.5
14	735	-13.2	-87	-26.2
15	800	-11.2	80	-24.2
16	960	-20.8	109	-33.8
17	1020	-14.5	91	-27.5
18	1100	-11.7	-82	-24.7
19	1210	-17.2	99	-30.2
20	1845	-16.7	98	-29.7

In Figure 4.8, simulation is performed to QPSK modulation technique in case of rate ½ and rate ¾ coding rates in a fading environment. It is simulated for 8 decoding iterations. From the simulation output, it is shown that CTC outperforms Convolutional Coding with the same coding rate at higher SNR, while ordinary Convolutional Codes have better performance at lower SNR.

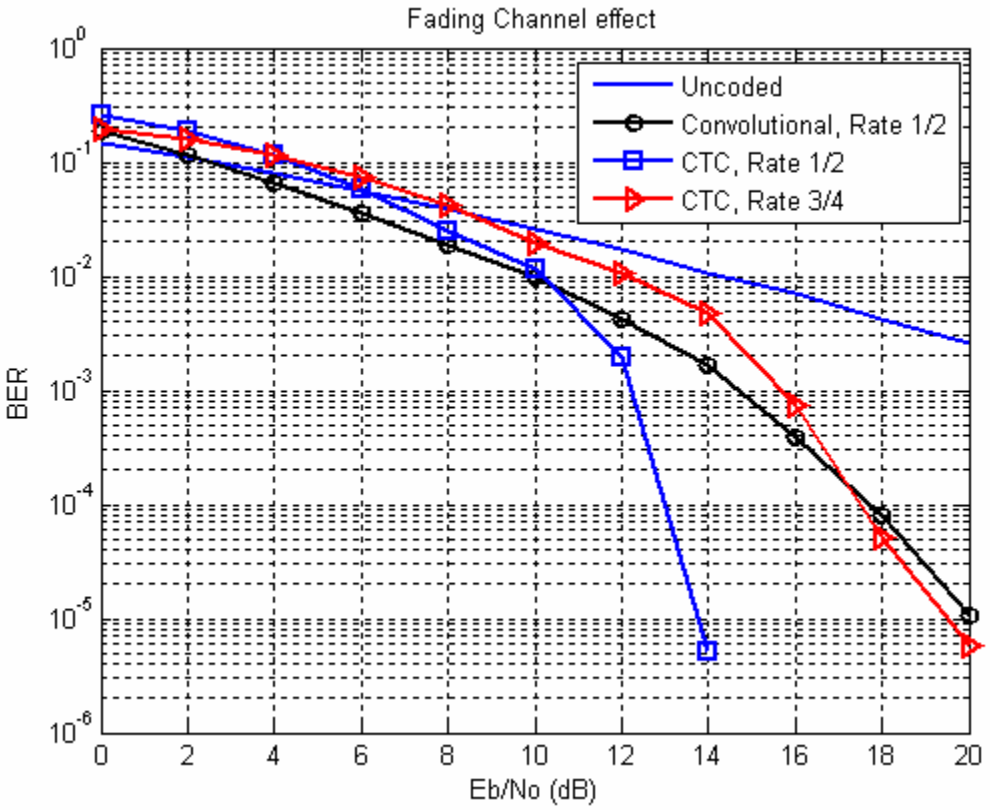


Figure 4.8 QPSK rate ½ and rate 3/4 a fading environment

4.4 Analysis using fixed point arithmetic

Fixed point analysis is a mandatory step before hardware implementation. It is important for purposes of seeking for an effective quantization with optimal number of bits of both received signals and internal signals without affecting coding performance. Received signals are represented by output systematic and parity signals from the channel. Internal signals are the branch and state metrics

and likelihoods. Many papers addressed the problem of Turbo decoder quantization and fixed point analysis [25 – 28].

In this section, fixed point simulation results is presented showing the optimal number of quantization bits for both input signals and internal signals.

4.4.1 Quantization of received signals

In Figure 4.9, quantization of input signals is indicated, it is shown that 4 bits for input data has a good performance, it approaches the performance of the floating point model but 3 bits results in a loss that exceeds 0.5 dB. This BER curve is for 4 iterations of turbo decoding.

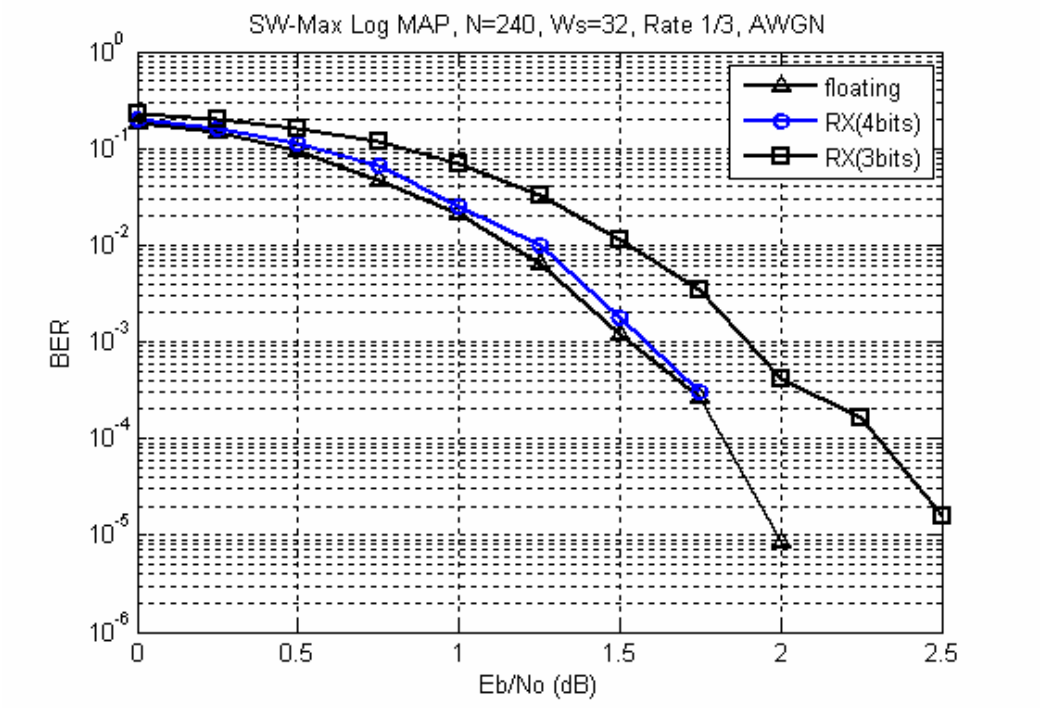


Figure 4.9 Fixed point vs Floating point model for received signals

4.4.2 Quantization of internal signals

It is shown the effect of quantization of extrinsic likelihood on system performance. Choosing the number of bits is affected by saturation limits of extrinsic likelihood, and affects values of other internal signals. Simulation parameters are fixed for number of bits of received data = 4 bits, rate 1/3, AWGN channel, Block size N=240, Window size (Ws)=32 and guard window (Wg)=4. This curve is plotted for 6 iterations of turbo decoding.

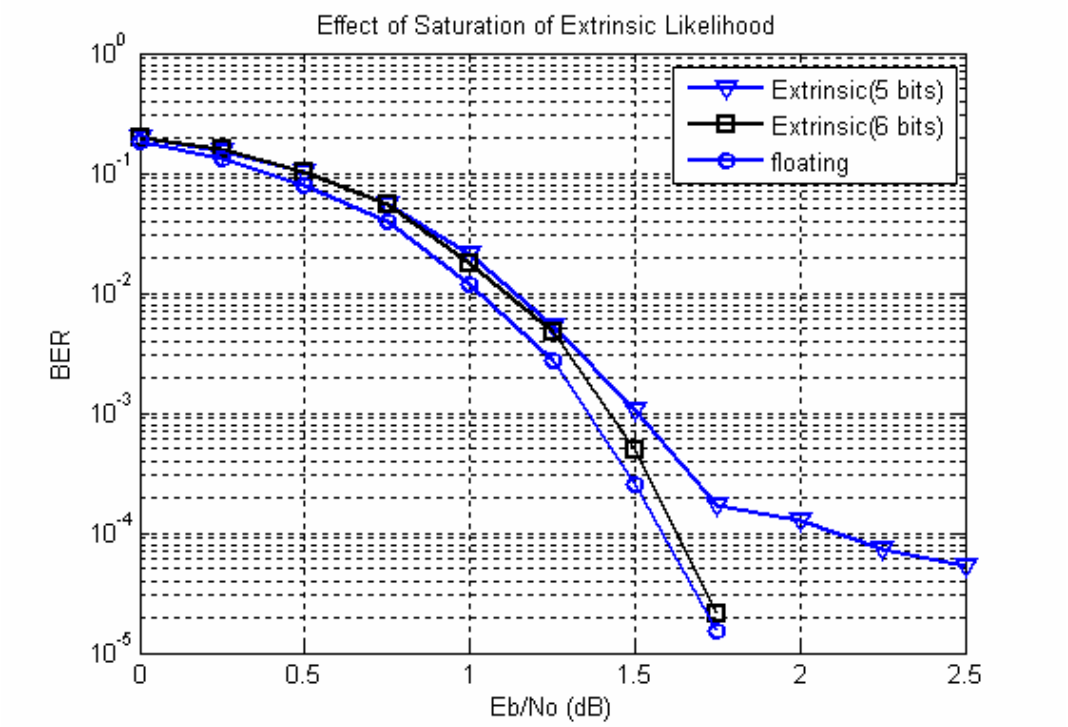


Figure 4.10 Effect of saturation of extrinsic likelihoods

Table 4-2 summarizes the number of quantization bits used for received and internal signals of turbo decoder

Table 4-2 Number of quantization bits for signals used in turbo decoder

Signal	Number of quantization bits
Received signals	4 bits
Branch metrics	4 to 7 bits
State metrics	8 bits
Extrinsic Likelihood	6 bits

The branch metrics are represented in a number of bits that ranges from 4 to 7 bits. This means that not all the branch metrics are represented in the same number of bits. We find that 4 bits are sufficient to represent some metrics, and the maximum is represented in no more than 7 bits. This is due to the proposed branch metric normalization method which is described in details in 5.3.2.1.

Chapter 5

Hardware Implementation of Turbo coding

5.1 Introduction

This chapter presents a hardware implementation of various blocks used in 802.16e Turbo encoder and Turbo decoder. It also discusses various aspects of optimization techniques used to guarantee good performance suitable for high data rate requirements by current wireless communication standards. Although many researchers addressed the turbo decoding implementation, some problems still represent a crucial issue such as metric representation in optimum number of bits, the minimum number of bits used to represent both input words and internal words. Another issue is the metric normalization, which will be discussed in section 5.3.3, to solve the problem of arithmetic overflow, arises from recursive computation. In this thesis, we present the previous work in this issue, and introduce a novel effective normalization technique suitable for the reduction of number of bits, memory requirements and avoiding arithmetic overflow without affecting the BER performance. An efficient implementation of this normalization scheme is also described using a redundant number system representation.

The platform of hardware prototyping and testing is Field Programmable Gate Array (FPGA). The target FPGA is STRATIX II. At last synthesis output of each block is presented**.**

5.2 Hardware Implementation of Turbo Encoder

As described in chapter 3, Turbo encoder consists of two constituent encoders and an interleaver. It uses double binary recursive systematic constituent encoders. It is considered as a rate 1/3 encoder as it has 2 input streams and six output streams.

The I/O block description of Turbo encoder is illustrated in Figure 5.1.

The input signals to this encoder are *A, B, Block_ID*. The first two inputs represent input information bits to be encoded, while *Block_ID* input determines some information about block such as Block size *N*. Other inputs are used for control such as *CLK, RST*. This encoder has six output signals which consist of two systematic and four parity coded bits. A *valid_out* signal is used to indicate that output is ready.

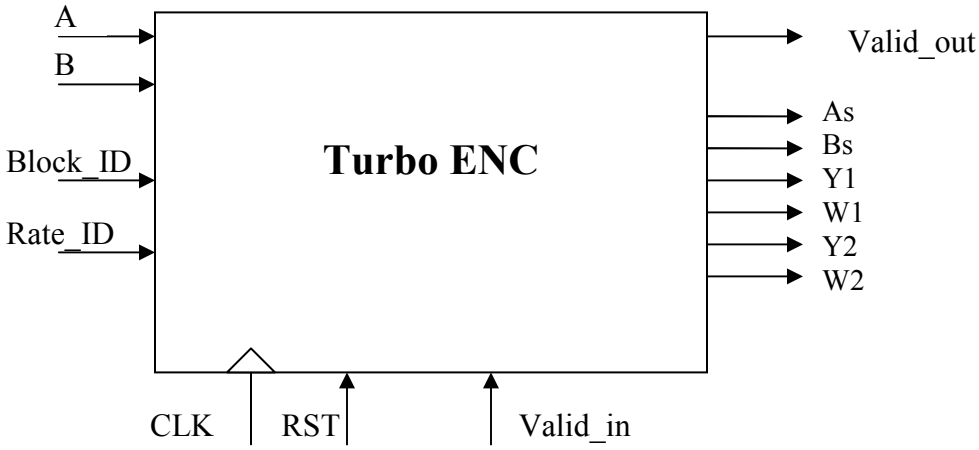
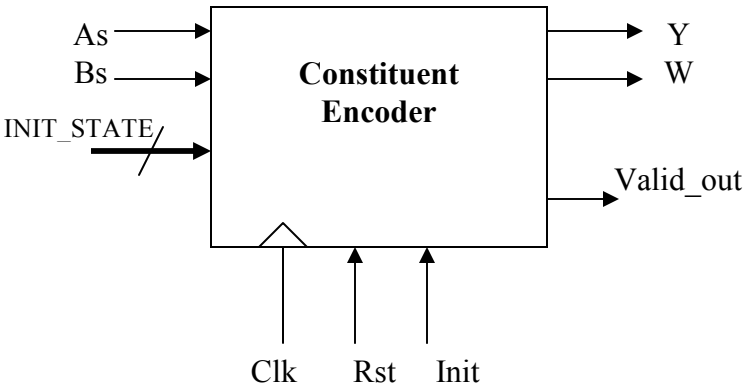


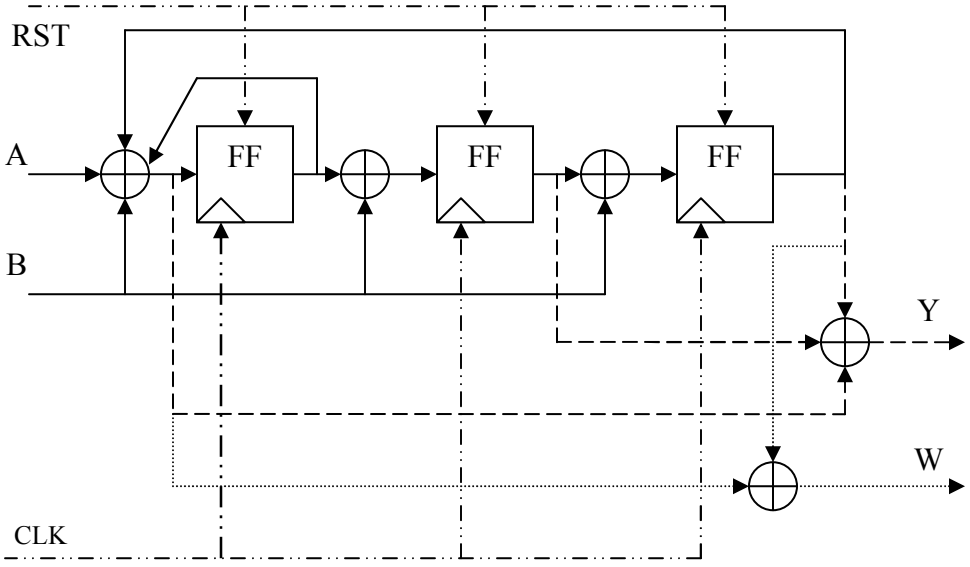
Figure 5.1 Turbo Encoder Block diagram

5.2.1 Constituent encoders

Each constituent encoder consists of three Flip flops and four mod-2 adders as indicated in Figure 5.2. The implementation of this block is very simple. Each constituent encoder has 2 inputs and 2 outputs. Other I/O signals are used such as *CLK*, asynchronous *RST*, *INIT_STAT*, *INIT* and *Valid_out* signals. The *INIT_STAT* signal loads the encoder with the initial state which is used in circular encoding as discussed in section 3.2.3. The loading process is controlled by the *INIT* input signal.



(a)



(b)

Figure 5.2 (a) Block diagram of Constituent encoder
(b) Structure of Constituent encoder

5.2.2 CTC Interleaver design

The function of the interleaver is to change the order of the incoming symbols; it consists of two steps as described in section 3.2.2. The first step is to exchange the order of bits of the input symbol alternatively. For even symbols,

swap A, B and for odd symbols keep their original order. The swapping criterion is simply implemented using two multiplexers. The Selection line of the MUXs changes with the symbol rate; this means that it equals half the input clock rate. Figure 5.3 illustrates the block diagram of the first stage of the interleaver with two input bits *A*, *B* and two swapped output bits *A1*, *B1*.

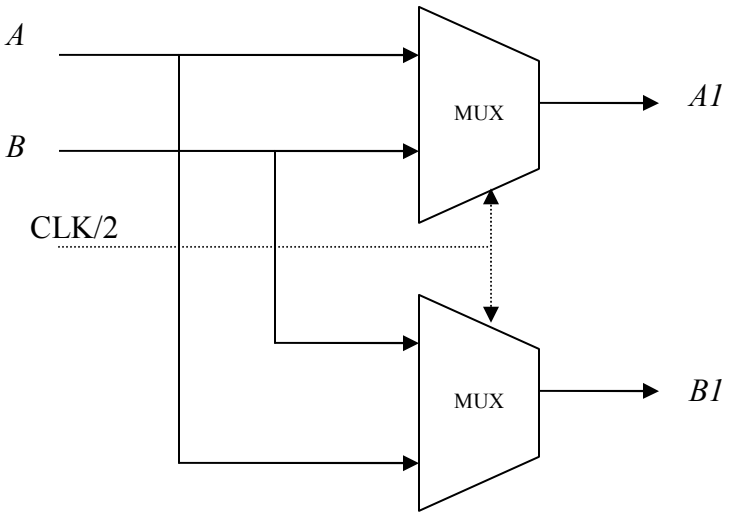


Figure 5.3 Interleaver first stage

The next step is to change the order of input symbols for the complete block of size N. This is implemented with a RAM module where input symbols are written with a certain sequence of addresses and read with a different sequence. The sequence of addresses is specified in the standard. In fact one RAM module is not sufficient as it will result in an overrun error. One possible solution is to use two RAM modules where writing and reading are performed in both modules alternatively.

The conventional architecture of this block consists of address generator and two RAM modules as indicated in Figure 5.4. The address generator has two outputs, one represents the linear address used in reading and the other represents the interleaved address used in writing.

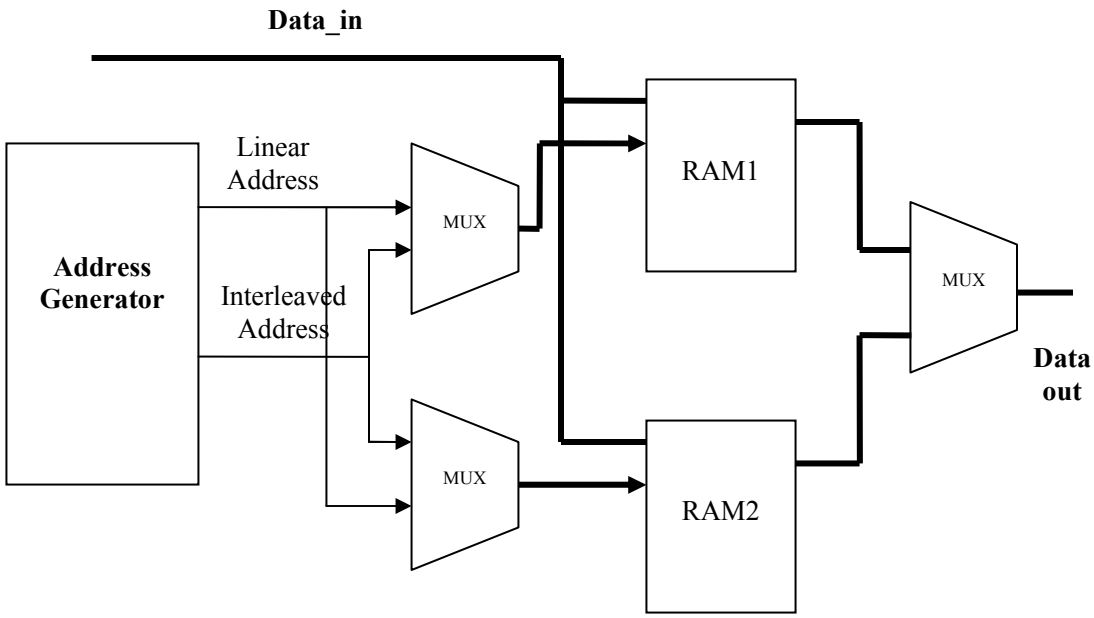


Figure 5.4 Interleaver structure

The address generator has two outputs, one represents the linear address, and the other represents the interleaved address. The sequence of generating linear address is simply carried out using a Mod-N counter. The sequence of generating the interleaved address is performed by the procedure specified in the standard (List 3.1). In conventional architectures, interleaver address generator can be implemented via a Look Up Table (LUT). However, in our case, LUT implementation consumes large storage capacity that reaches up to 12 Kbits approximately. The alternative solution is to implement the logic function of the address generator. Section 5.2.2.1illustrates the address generator architecture using LUT implementation, and in section 5.2.2.2, the proposed implementation is presented.

5.2.2.1 LUT Implementation

The LUT implementation of the address generator has the benefit of a straightforward design. In our case, the proposed architecture is given in Figure

5.5, where memory organization is divided into several banks, a bank corresponding to each block size *N*. Only one bank is enabled at a time, this plays a role in reducing power consumption relative to the case of implementing the LUT as one memory bank. Another issue is that accessing one bank with smaller memory depth decreases the memory access time.

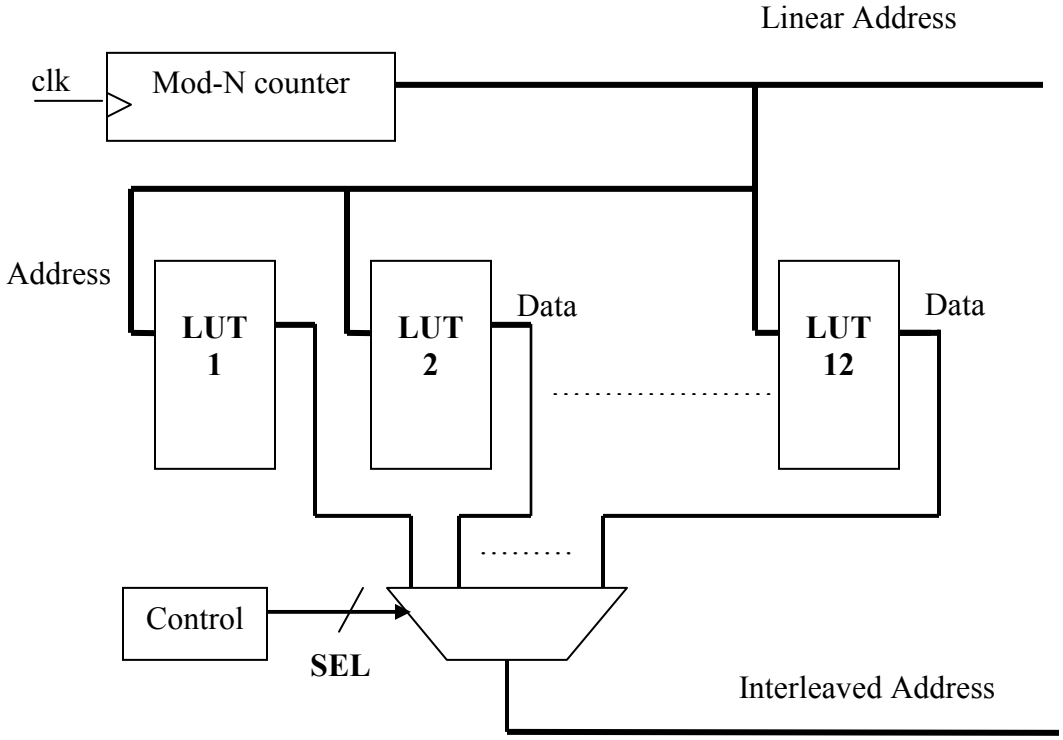


Figure 5.5 Address generator using LUT

5.2.2.2 Proposed Address generator Implementation

The proposed structure of the address generator is shown in Figure 5.6. To generate the interleaved address, an efficient implementation is carried out by replacing the multiplication with a simple accumulator. This has its significant reduction in hardware resources, area and power consumption beside enhancement of speed.

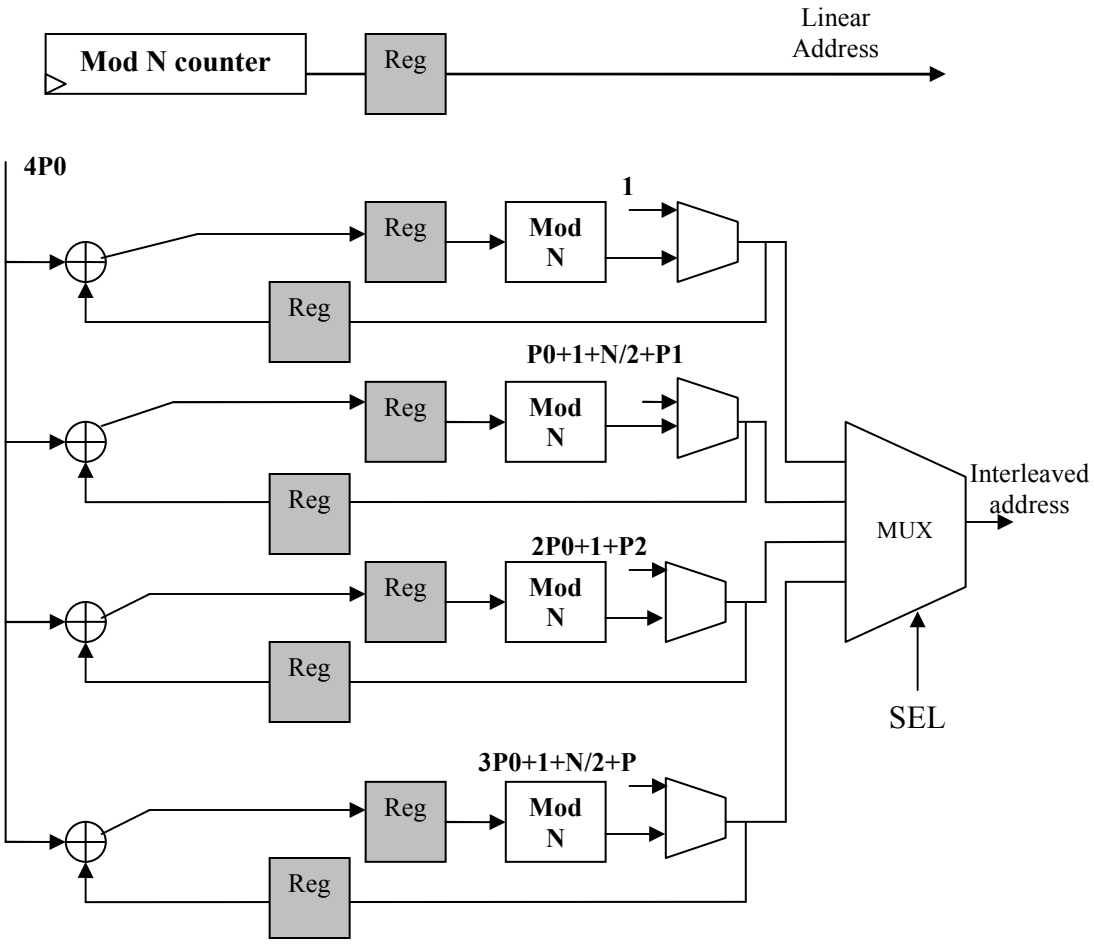


Figure 5.6 Proposed address Generator structure

The key idea behind this implementation is re-writing of the equations mentioned in List 3.1 to a new set of equations as shown below. This new form has the same function and simplifies the hardware implementation at the same time.

P(0) = 1

P(1) = $(P_0+1+N/2+P_1)\ mod_N$

P(2) = $(2P_0+1+P_2)\ mod_N$

P(3) = $(3P_0+1+N/2+P_3)\ mod_N$

for j = 4 to N-1

$$P(j) = (P(j-4) + 4P_0)\ mod_N \qquad \text{List 5.1}$$

end

These initial values represented by P(0), P(1), P(2) and P(3) are stored in a specific ROM module, then the remaining addresses are calculated recursively. The contents of the ROM module are specified in Table 5-1.

Table 5-1 Interleaver parameters stored in ROM

N	$(P_0+1+N/2+P_1)\ mod_N$	$(2P_0+1+P_2)\ mod_N$	$(3P_0+1+N/2+P_3)\ mod_N$
24	18	11	4
36	12	23	34
48	14	27	40
72	54	23	4
96	8	39	46
108	12	79	90
120	14	27	40
144	20	107	126
180	12	23	34
192	12	71	82
216	14	27	40
240	14	87	100

A further optimization can be added to address generator indicated in Figure 5.7. By taking into consideration that not all adders are used simultaneously, a resource optimization is available through using only one adder and multiplexing its four inputs. This can also be applied to the MOD-N block. In the new structure the critical path may be slightly increased due to additional multiplexers and demultiplexers, but it is much smaller compared to significant decrease in resources and area.

In addition, the implementation of MOD-N is not simply carried out by considering the least significant k-bits of the input to this block, instead a divider is needed. However, to avoid division, this implementation can be carried out through successive subtractions as given in equation (5.1). The problem that arises

from successive subtraction is the variable latency which is not desired in hardware implementation.

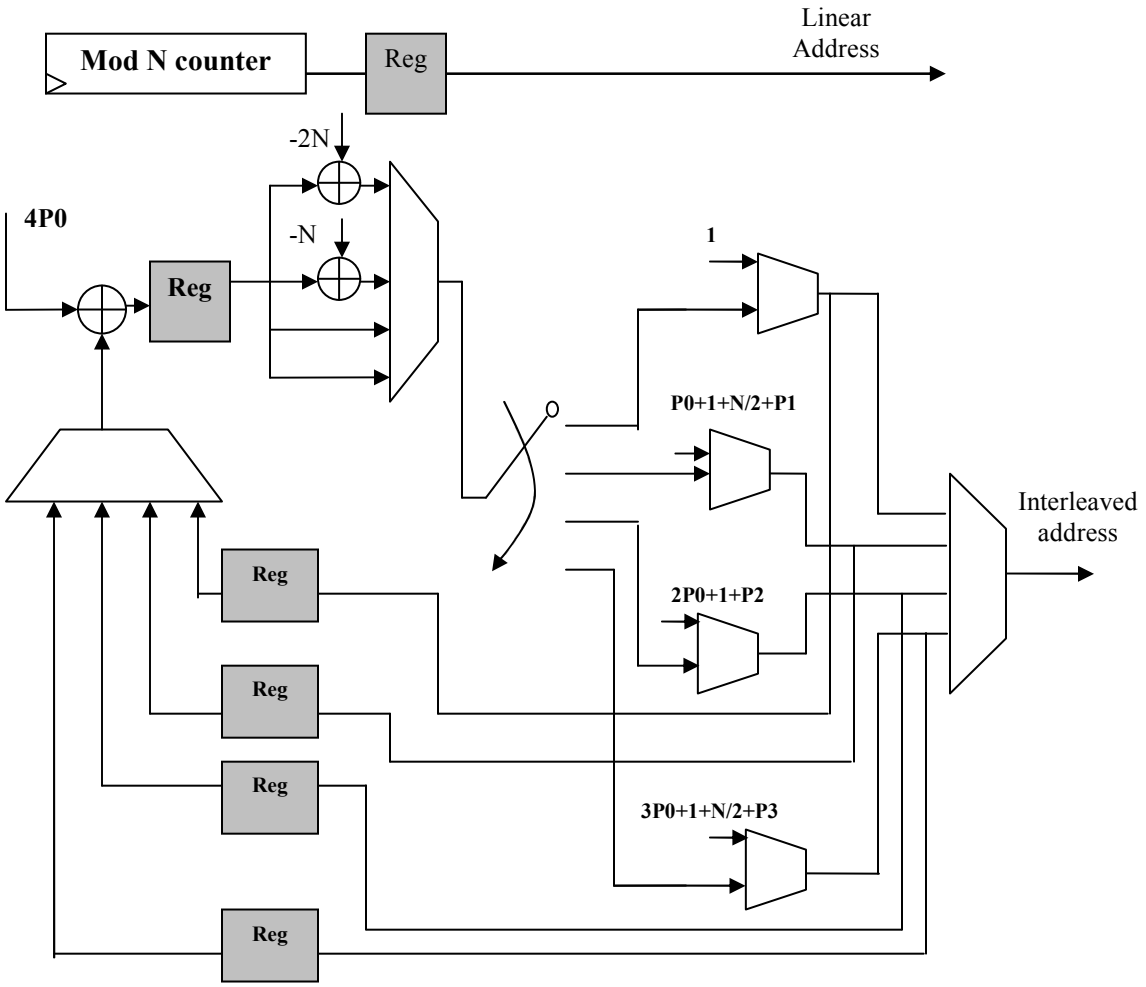


Figure 5.7 Optimized address generator structure

$$X \bmod_N = X - \left\lfloor \frac{X}{N} \right\rfloor N \tag{5.1}$$

In our case, for all possible values of X, we notice that we need to calculate only *X*, *X-N*, and *X-2N* in the worst case. This simplifies the implementation to use only two subtractions. In order to avoid variable latency, they can be computed in parallel. An exhaustive testing was performed and indicated that this

implementation scheme works properly. The output of this block is connected back to the accumulator before calculation of the subsequent interleaved address.

The interleaver introduces a certain delay that depends on the block length. In order to guarantee that both constituent encoders generate their output simultaneously, a queue is used to introduce an equivalent delay before the first constituent encoder. The block diagram of the encoder becomes as indicated in Figure 5.8

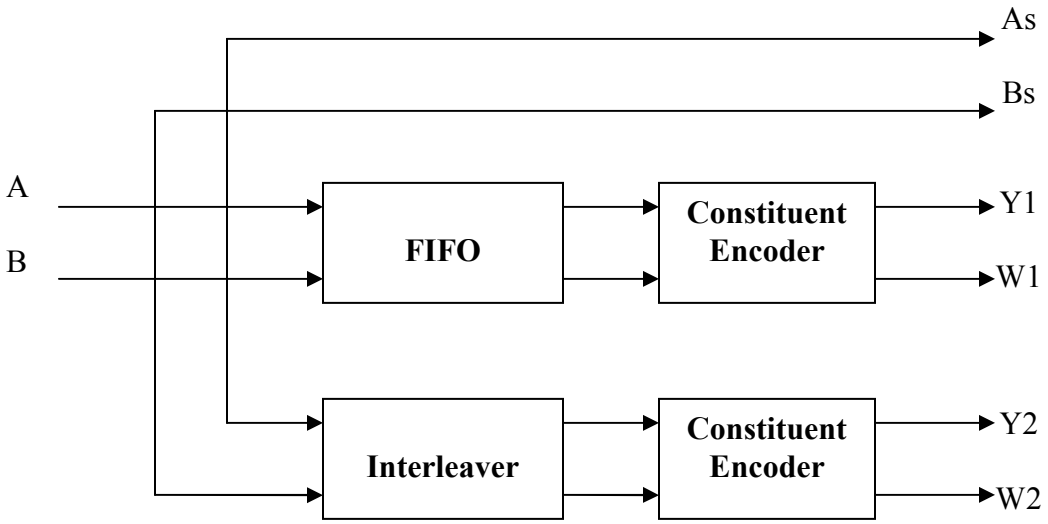


Figure 5.8 Block diagram of CTC encoder

5.2.3 Circulation state look up table

The tail biting scheme used in IEEE802.16e turbo encoder is circular coding, this scheme guarantees that the initial state is the same as final state. The sequence of determination of circulation state Sc was described in section 3.2.3. This is implemented with a ROM module that contains different circulation states corresponding to different block sizes (N) and final state $S0_{N-1}$.

After determination of the circulation state, re-encoding of block takes place after initializing each of the constituent encoders with the correct circulation state. This means that incoming data should be buffered again while being encoded for the first time, this is performed using two queues, one to buffer the original stream and the other to buffer the interleaved stream. Two other constituent encoders are used to encode the original stream after being initialized by circulation state.

The construction of Sc ROM module is simple that its address consists of two parts, the final Sc of first encoding concatenated with the value of $Nmod_7$. Each of them consists of 3 bits. The overall ROM address consists of 6 bits; each location inside ROM consists of 3 bits that determines the corresponding Sc. ROM contents are initialized with respect to Sc Table 3-1. The ROM output is connected to the *init_stat* input signal of constituent encoder and this signal is triggered by the control input *INIT* signal which is activated at the end of each block. The resulting block diagram of Turbo encoder is shown in Figure 5.9.

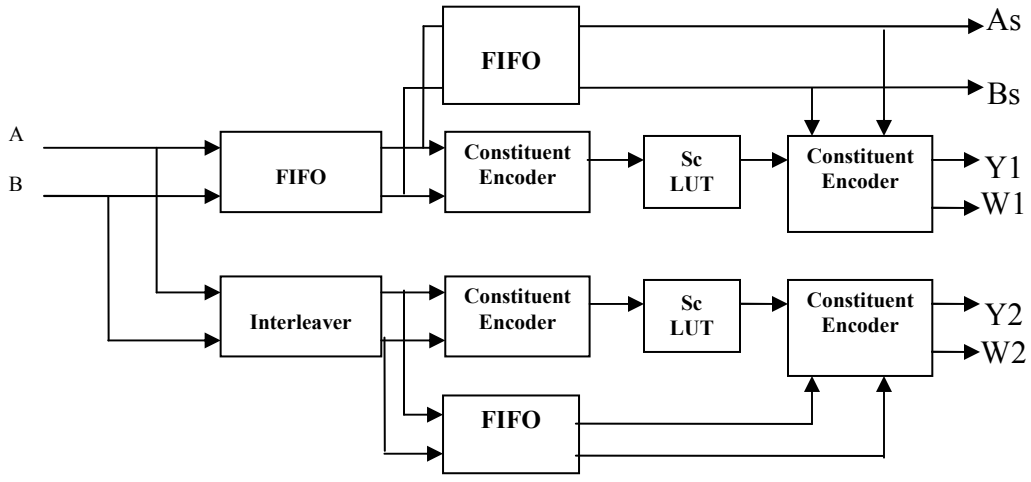


Figure 5.9 Circular Rate 1/3 Turbo Encoder

5.2.4 Sub-packet generation

The main blocks in sub-packet generation is sub-packet interleaving, symbol grouping and puncturing as discussed in section 3.2.4.

5.2.4.1 Implementation of sub-block interleaver

The sub-block interleaver has the same structure as the CTC interleaver discussed in 5.2.2. It consists of two RAM modules in addition to the interleaver address generator. In this case, one address generator is sufficient to generate linear and interleaved address for all six sub-blocks simultaneously. In order to generate interleaved address, we need to implement the procedure discussed in 3.2.4.2. The flow chart in Figure 5.10 illustrates the operation of interleaved address generation.

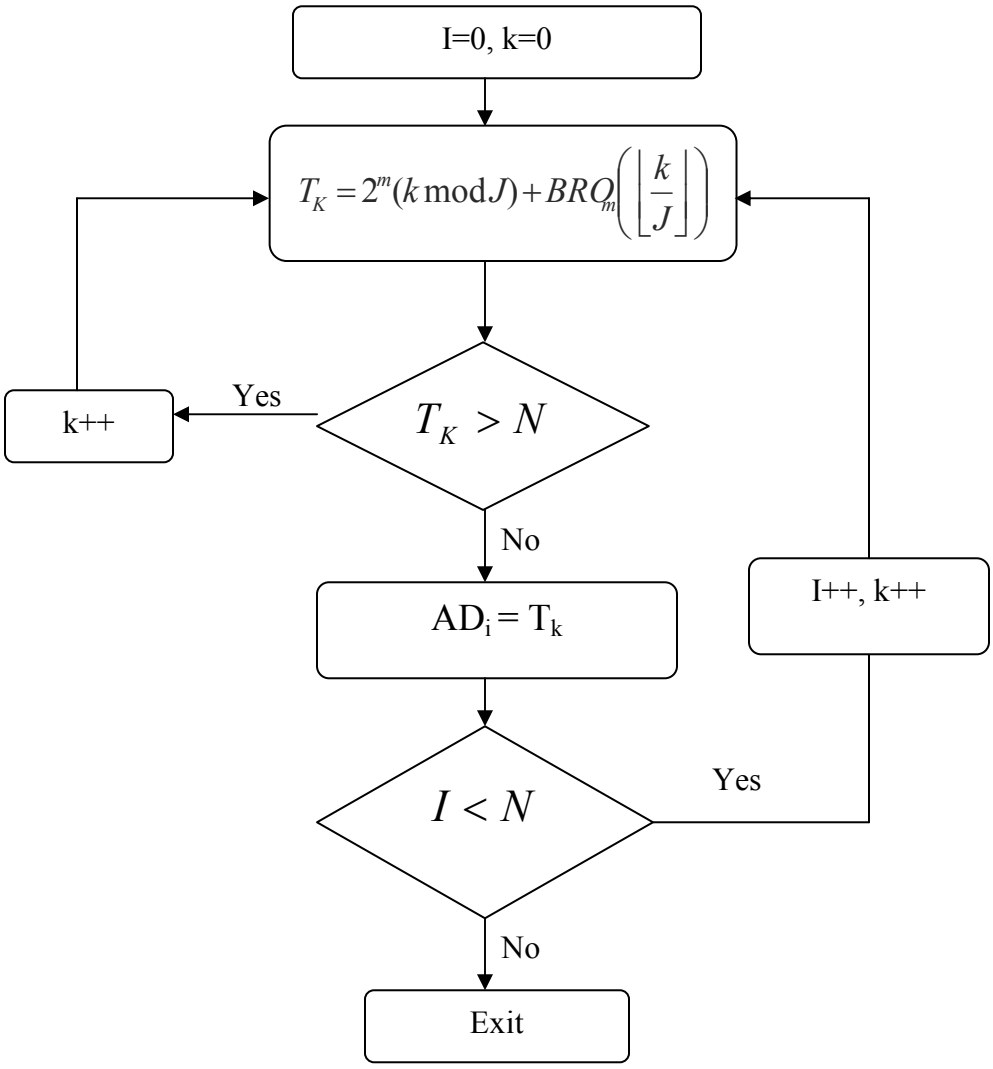


Figure 5.10 Sub-block interleaver address generation flow chart

In this thesis, we propose an efficient implementation for the sub-block interleaver address generator. In order to calculate T_k, we notice that addition operation is simply carried out using concatenation of two values. Moreover, these two values can be simply generated using two counters as follows:

1- 2-bit counter is used to calculate the value of $k \bmod J$. This counter is triggered each clock cycle.

2- m-bit counter is used to calculate the value of $BRO_m\left(\left\lfloor \frac{k}{J} \right\rfloor\right)$. The order of the output of this counter is reversed.

The tentative computed address T_K is then compared to the value of sub-block size N. The problem arising from this comparison is the added latency and recursive calculation of T_K. However, it is found that we need at most one recursive calculation at a time. In order to remove latency, we propose an efficient implementation to perform comparison of the next address in parallel to current tentative address computation. If the comparator output indicates that T_k>N, we should reset the 2-bit counter and increment the m-bit counter. The block diagram of the proposed address generator is given in Figure 5.11

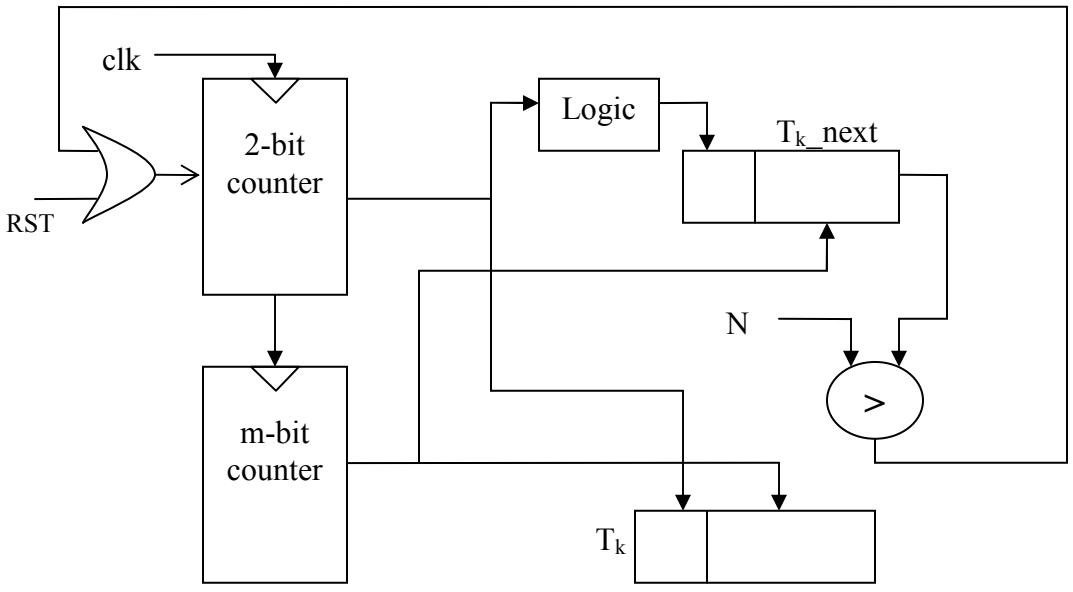


Figure 5.11 Sub-block interleaver address generator

5.3 Hardware Implementation of Turbo decoder

5.3.1 General Architecture

As explained in chapter 3, Turbo decoder consists of two component decoders, each one corresponding to one constituent encoder. The decoder should be implemented as Soft Input Soft Output (SISO) decoder using any decoding techniques specified in chapter 3. In this thesis, Sliding Window Max Log MAP algorithm is used for SISO decoder implementation. This algorithm is widely used in implementation of turbo decoders. Many proposed implementation techniques were addressed in order to reduce the area, delay, and power consumption and enhance performance.

Each SISO decoder, as indicated in Figure 5.12, has two received systematic symbols, two received parity symbols and three extrinsic likelihoods needed in double binary as explained before. Other control inputs are *CLK* and *RST* signals. It has two outputs *A_out, B_out* that corresponds to decoded bits. Other outputs are *Le_01, Le_10, Le_11* which represent extrinsic likelihoods. A *valid_out* signal is used for indication of ready output. *Sc_in* and *Sc_out* indicate input and output circulation states simultaneously. *Block_start* signal is an input signal which is activated at the start of a block for each iteration it is decoded.

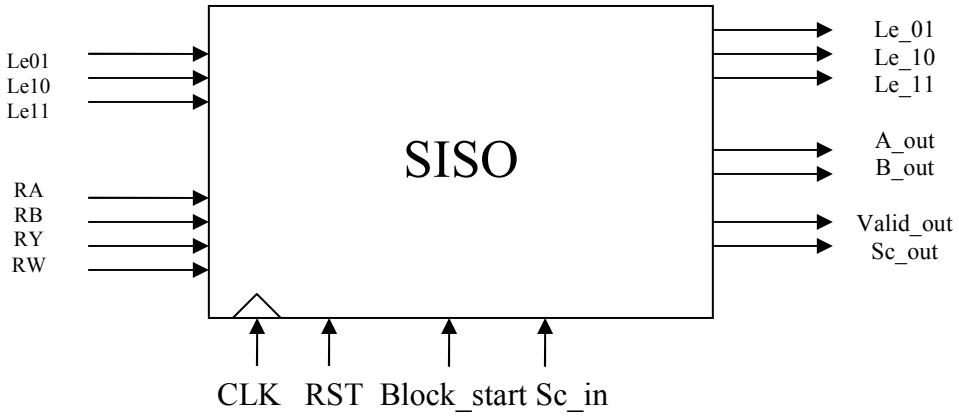


Figure 5.12 SISO decoder Block description

The implementation of each SISO decoder implies the calculation of forward state metric (ALPHA), Backward state metric (BETA) and Branch metric (GAMMA) at each time slot. In case of SW-Log MAP, each block is divided into windows while backward estimation is calculated for each window separately. The window size specifies the memory storage requirements of both branch and forward state metrics. The proposed architecture of the decoder is given in Figure 5.13.

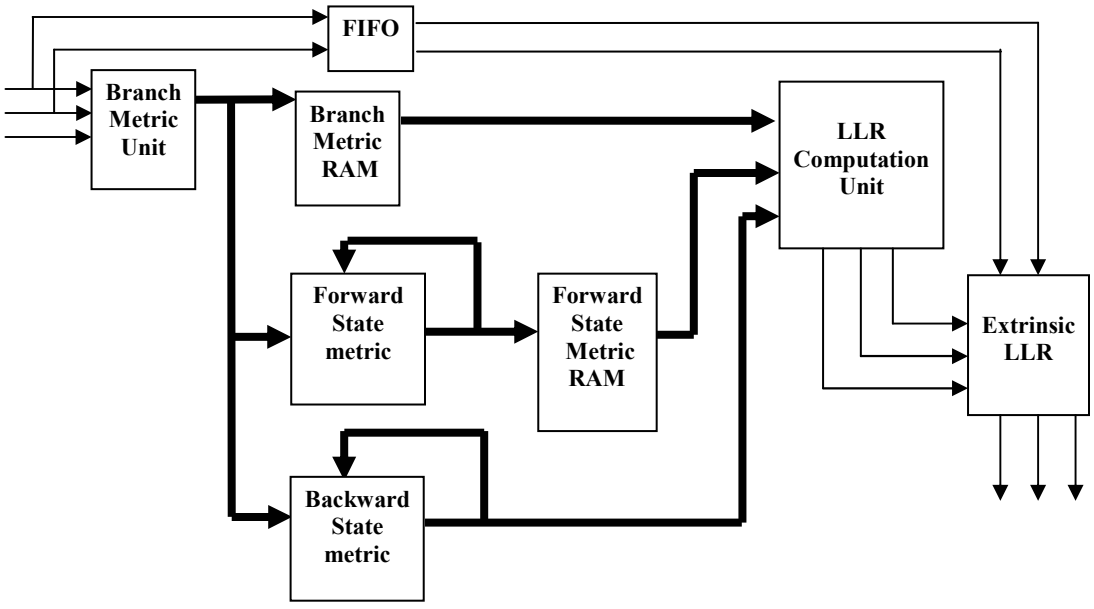


Figure 5.13 SISO Architecture

5.3.2 Branch Metric Block (GAMMA)

As explained before, the calculation of each branch metric implies a cross correlation between received systematic and parity data bits with original bits corresponding to this branch. In case of 802.16e turbo decoder, the trellis diagram has 8 states, each has four output branches. This implies the calculation of 32 branch metrics each time slot. In fact, the calculations may be halved. Only 16 metrics are sufficient, the other 16 metrics are the same, as shown from the state transition table given below.

Table 5-2 Turbo decoder state transition table

	I/P 00 OP/next state	I/P 01 OP/next state	I/P 10 OP/next state	I/P 11 OP/next state
S0	00 / 0	11 / 7	11 / 1	00 / 6
S1	11 / 3	00 / 4	00 / 2	11 / 5
S2	10 / 4	01 / 3	01 / 5	10 / 2
S3	01 / 7	10 / 0	10 / 6	01 / 1
S4	00 / 1	11 / 6	11 / 0	00 / 7
S5	11 / 2	00 / 5	00 / 3	11 / 4
S6	10 / 5	01 / 2	01 / 4	10 / 3
S7	01 / 6	10 / 1	10 / 7	01 / 0

The Calculation of each branch metric is calculated as given in equation (3.39), where the values A, B, Y1, Y2 Є {-1, 1} So, the implementation of each metric is simply carried out with a multi-operand adder, as shown in Figure 5.14.a. Each multi-operand adder is constructed from a set of Carry Save adders (CSA) and the last stage is the Carry Propagation adder (CPA).

After the calculation of the branch metrics, they should be stored in RAM modules to be used later in calculation of LLRs. This is implemented through parallel RAM modules, as indicated in Figure 5.14.b one module for each metric calculated. The depth of each RAM module depends on the window size.

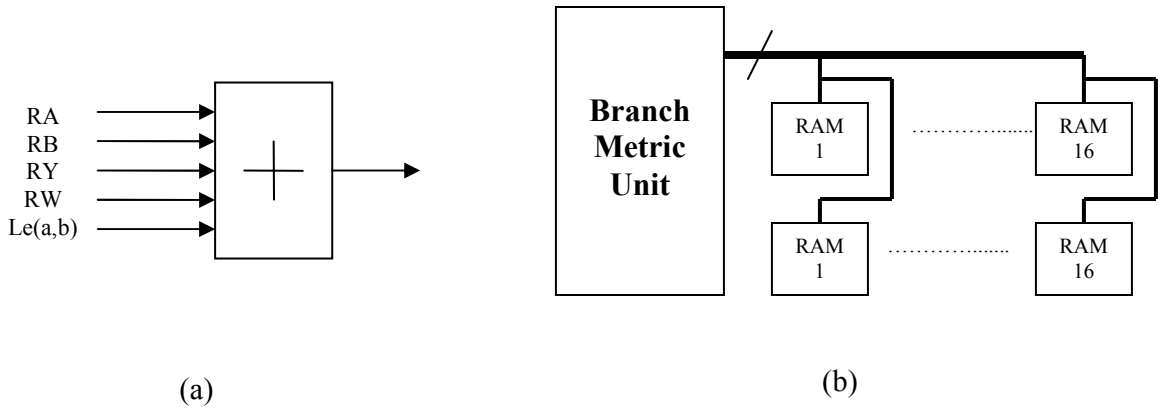


Figure 5.14 (a) Branch metric Multi-operand Adder (b) Branch metric Memory organization

As in case of SW-MAX Log MAP, at backward recursion, LLRs can be calculated immediately, so there is no need to store backward metrics in memory. Also, at backward recursion, the values of branch metrics are read from memory. At the same time, branch metrics of next window are calculated and stored in memory. In order to handle this case, we propose to use two modules for each window; Reading and Writing in are performed alternatively between the two groups.

5.3.2.1 Proposed Branch metric Normalization scheme

In this thesis, a hardware efficient branch metric normalization scheme is used. In this scheme, all calculated branch metrics are normalized with respect to the all zeros branch metric, which is the first branch in the trellis. The key idea behind this normalization is that the main concern is not in the values of the metrics themselves, but it is in the difference between them.

The benefit of normalization to zero metric is a significant reduction in hardware and storage requirements. This is obvious as equation (3.39) will be reduced to

$$\Gamma_{k-1\to k}(AB \Leftrightarrow 00) = R_Y(k)*Y + R_W(k)*W$$

$$\Gamma_{k-1\to k}(AB \Leftrightarrow 01) = R_B(k) + R_Y(k)*Y + R_W(k)*W + Le_k(0,1)$$

$$\Gamma_{k-1\to k}(AB \Leftrightarrow 10) = R_A(k) + R_Y(k)*Y + R_W(k)*W + Le_k(1,0)$$

$$\Gamma_{k-1\to k}(AB \Leftrightarrow 11) = R_A(k) + R_B(k) + R_Y(k)*Y + R_W(k)*W + Le_k(1,1) \quad (5.2)$$

Where $Y_1, W_1 \in \{0, 1\}$

The reduction obtained is the decrease in the number of the required additions than the case of the conventional calculation schemes. This has its effect on speed enhancement by reducing the critical path delay. In this case a specific hardware is designed to each metric separately. In addition to hardware reduction, it reduces the number of bits required to represent some branch metrics. In other words, each metric can be represented in a lower number of bits optimized for this metric.

In this normalization scheme, only 15 branch metrics are needed to be calculated, no need for storage of 16 metrics as the previous schemes. This results also in reduction in memory modules needed. Another benefit of this scheme is the reduction of the critical path in some branch metric units, due to a lower number of CSAs. This also means smaller power consumption.

Table 5-3 Resource reduction of proposed normalization

	Without Normalization	Proposed Normalization
Number of CSAs for each unit	4 units with 1 CPA 8 units with 2 CSAs +1 CPA 4 units with 3 CSAs + 1 CPA	3 units have only 1 CPA 5 units with 1 CSA+1 CPA 4 units with 2 CSAs+1 CPA 1unit with 3 CSAs +1 CPA
Total area Estimation in terms in number of CSAs and CPAs	28 CSAs + 16 CPAs	16 CSAs + 13 CPAs

From the results obtained in Table 5-3, we get a reduction of the area by approximately 34% over the conventional scheme without normalization. Moreover, as we have lower number of bits for some branch metrics, we obtain a reduction in the memory requirements over the conventional implementation. The results obtained in Table 5-4 indicate that for our case of SW-MAX Log MAP, of window size Ws=32, we need 6656 bits to store all branches of a certain window and 6208 memory bits in case of proposed normalization. This means a reduction of about 6.7% of the memory requirements.

Table 5-4 Reduction in storage due to proposed normalization

	Without Normalization	Proposed Normalization
Branch metric memory bits	6656 bits	6208 bits

A further simplification can be applied to the special case of non HARQ support. In this case, we find that for all coding rates, we obtain punctured parity outputs *W1*, *W2*. If we consider this at the receiver, *RW1* and *RW2* signals are always considered zeros. Taking this into consideration, we need only to calculate 8 branch metrics and we obtain the new set of branch metric equations as follows

$$\Gamma_{k-1\to k}(AB \Leftrightarrow 00) = R_Y(k) * Y$$

$$\Gamma_{k-1\to k}(AB \Leftrightarrow 01) = R_B(k) + R_Y(k) * Y + Le_k(0,1)$$

$$\Gamma_{k-1\to k}(AB \Leftrightarrow 10) = R_A(k) + R_Y(k) * Y + Le_k(1,0)$$

$$\Gamma_{k-1\to k}(AB \Leftrightarrow 11) = R_A(k) + R_B(k) + R_Y(k) * Y + Le_k(1,1) \qquad (5.3)$$

The branch metric unit consists in this case of 7 Multi-operand adders, they are classified as follows:

- 2 units with 1 CPA

- 3 units with 1 CSA + 1 CPA
- 1 unit with 2 CSAs + 1 CPA

The total number is 5 CSAs and 6 CPAs. This means an approximate additional decrease in the branch metric unit area by about 75% of the original scheme, and 62% of our proposed scheme with normalization. Moreover, the required number of memory bits will be reduced to 2944 bits. This means a reduction of the storage requirements by 55.77% of the original scheme and by 52.58% of the proposed technique.

5.3.3 Forward State Metric Block (ALPHA)

The purpose of the forward state metric unit is to calculate forward state metrics of the eight states and store them in memory for the computation of LLRs. The block diagram of the forward metric unit is shown in Figure 5.15. The input states are either the states of the previous iteration or the circulation states in the first iteration.

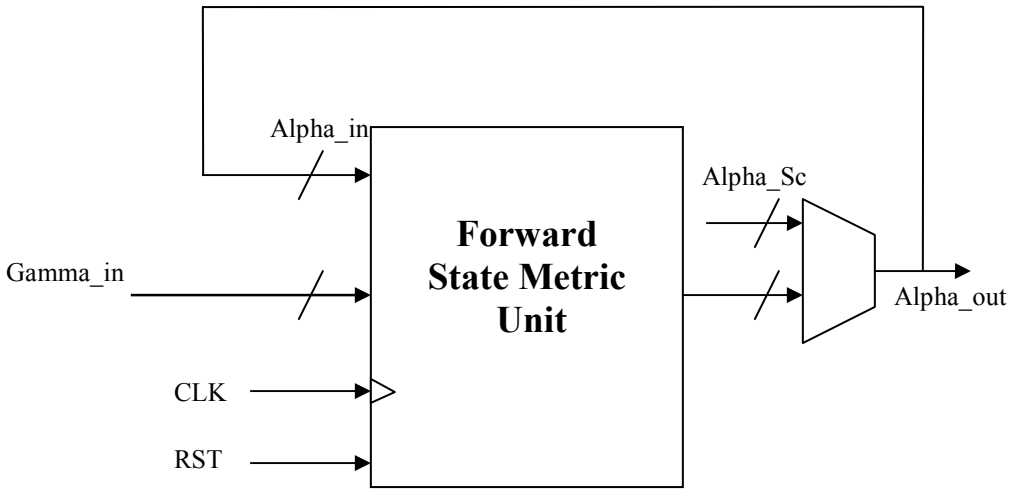


Figure 5.15 Forward State metric Unit

5.3.3.1 State Metric Unit Implementation

The state metric unit, consists mainly of an Add/Compare and Select (ACS) unit as shown in Figure 5.16. The main drawback in implementing state metrics is the recursive computation. This may lead to an arithmetic overflow. To avoid overflow, a large number of bits is needed for representation of state metrics. This means more area, hardware resources, higher storage requirements, and increased delay.

Many papers addressed the problem of the state metric arithmetic overflow. To overcome this problem, state metric normalization is carried out. Two normalization techniques were proposed by researchers; Rescaling and Modulo-Normalization. These two techniques maintain the dynamic range of the state metrics. The key idea is that the main concern is not in the value of the state metric itself, but in the value of the difference between the state metrics. Taking this into consideration, we can have a more efficient representation of state metrics.

5.3.3.2 Normalization by rescaling

Normalization by rescaling is carried out via subtraction of the maximum or minimum state metric from each state metric [29],[30]. This preserves the dynamic range and required number of bits to represent state metrics. Some other techniques proposed to normalize branch metric instead of the state metrics [31]. The main drawback of state metric normalization is the increase in the critical path of the state metric unit. It is considered the bottleneck of the SISO decoder that limits the maximum frequency of operation. The critical path implies Addition, comparison, MUX, and normalization which includes both comparison and subtraction.

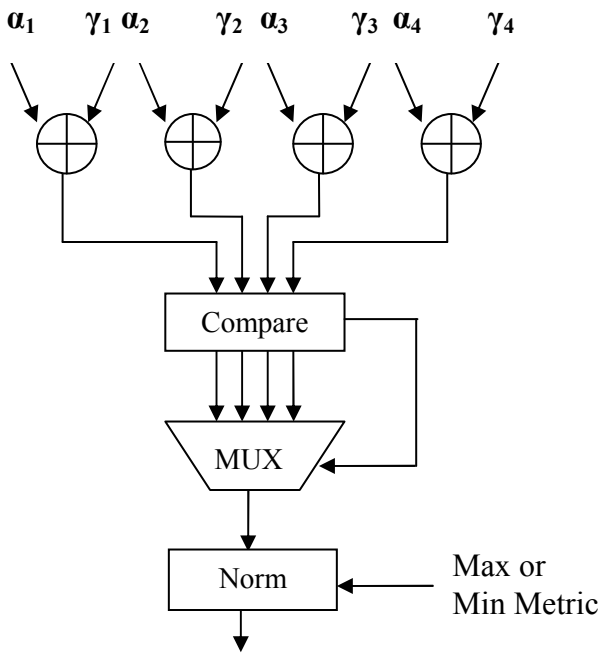


Figure 5.16 State metric unit

5.3.3.3 Modulo-Normalization

In case of modulo-normalization, instead of subtraction of the maximum or minimum metric, the state metrics are represented in a mod_{2^b} based operation [32]. The calculation of LLR is invariant with respect to the mod_{2^b} as the difference between the original state metrics does not change in case of modulo-representation. This idea was proposed first time by Hekstra [33], who applied it to viterbi decoding. To illustrate the idea, assume that we have a bound on the value of branch metrics such that

$$\left| \gamma_{k-1\to k}\left(s' \to s\right) \right| \le B_{\max} \tag{5.4}$$

where $B_{\max}$ represents the upper bound on the value of any branch metric. It can be proved that the upper bound on the difference of state metrics is

$$\begin{aligned} \Delta_k\left(s_1, s_2\right) &= \left| \alpha_k\left(s_1\right) - \alpha_k\left(s_2\right) \right| \\ \Delta_{\max} &= \left(2B_{\max} + \ln(2)\right)m \end{aligned} \tag{5.5}$$

where m is the memory order of the convolutional code used in CTC encoder. This proof can be found in [33].

We define $\tilde{\alpha}_k(s) = \alpha_k(s) \bmod_{2^b}$

$$\left(\tilde{\alpha}_k(s_1) - \tilde{\alpha}_k(s_2)\right) \bmod_{2^b} = \alpha(s_1) - \alpha(s_2) \tag{5.6}$$

In order for (5.6) to be satisfied, the number of bits b should be chosen such that

$$b = \lceil \log_2(\Delta_{\max}) \rceil + 1 \tag{5.7}$$

Moreover, the number of bits b' used to represent the LLRs must guarantee invariance in calculation of LLR after performing $\bmod_{2^b}$ operation. As a result, as mentioned in (17) of [32], the number of bits b' is set to

$$b' = \lceil \log_2(2\Delta_{\max} + \mathrm{B}_{\max}) \rceil + 1 \tag{5.8}$$

This normalization scheme has its benefits in speeding up the operation, as no extra hardware is needed for the normalization unit. However, this scheme has its disadvantage in the larger number of bits needed to represent the state metrics compared to the case of normalization by subtraction. In this case, we find that at least 10-bit representation is required for state metrics. This means an increase in memory storage requirements.

In this thesis, we propose an implementation of normalization scheme that is based on rescaling, so it preserves the number of bits, and at the same time it removes the normalization unit from the critical path. This is carried out through redundant representation of normalized state metrics. In the next section, we present an introduction to the redundant number representation, and then in the succeeding one, we introduce how the redundant representation is applied to the state metric normalization.

5.3.3.4 Redundant Number Representation

Redundant number representation is defined in arithmetic operations as a way to increase the speed of the addition operation [34]. Carry propagation is considered the bottleneck that limits the speed of any addition operation. The delay of carry propagation varies according to the addition technique which can be ripple carry adders, Carry look-ahead, Conditional sum adder ...etc.

In redundant number system, carry-free addition is achieved. The key idea is the extension of number representation of a radix β system such that it is not limited to [0... β-1]. For example, in the decimal radix 10 system, we represent any number with the set of digits [0, 1…9]. In case of redundant representation, we allow a representation with further digits such as 10, 11… 18 so any number can be represented with a set of digits [0…18]. This representation eliminates carry propagation in addition as shown in the following example:
Assume we need to add two numbers 362910 and 278635. The ordinary addition which is held via carry propagation will be

```
  1  1  1
  3  6  2  9  1  0
+ 2  7  8  6  3  5
_____________________
  6  4  1  5  4  5
```

And with redundant representation

```
  3   6   2   9   1   0
+ 2   7   8   6   3   5
_____________________
  5  13  10  15   4   5
```

The previous example illustrates the redundant number system in addition in case of inputs are in the non-redundant format. Moreover, we can consider an example if the input operands are in the redundant format. One can think that if the inputs

occupy the digit range [0, 18], the output is extended to the range [0, 36]. However, any digit in the range [0, 36] can be decomposed into an interim sum in the range [0, 16] and a transfer digit (carry) in the range [0, 2].i.e. it is represented as [0, 1, 2… 36]=10 x [0, 1, 2] + [0, 1, 2….16] , Then, one additional concurrent addition stage is necessary to recover the output in the range [0, 18]. To illustrate this idea, consider the following example

	11	9	17	10	12	18
+	6	12	9	10	8	18
	17	21	26	20	20	36
	↓	↓	↓	↓	↓	↓
	7	11	16	0	0	16
1	1	1	2	2	2	
1	8	12	18	2	2	16

We find that we have two concurrent addition levels. Another representation for the same example can be as follows

	11	9	17	10	12	18
+	6	12	9	10	8	18
	17	21	26	20	20	36
	↓	↓	↓	↓	↓	↓
	7	1	6	10	10	16
1	2	2	1	1	2	
1	9	3	7	11	12	16

We find two different representations for the same result; this is why it is a redundant number system representation. If we convert it back to the non-redundant format, we have the same result for the two different representations. It is 1 9 3 8 2 3 6 .

In case of a redundant representation, addition in all digit positions is performed concurrently; this is called carry save additions. A possible redundant form on which we can represent the binary systems is the set of digits $\{1, 0, \bar{1}\}$. In this case, each of the 3 digits is represented using two bits. Assume we need to subtract 10011 from 01010, the result in redundant format will be $\bar{1}100\bar{1}$. This idea can be applied to the case of metric normalization with subtraction. Instead of performing subtraction of $A_k^n(s_1) = A_k(s_1) - A_k(s_0)$, the direct combination of $A_k(s_1)$, $A_k(s_0)$ is considered a redundant representation of $A_k^n(s_1)$.

5.3.3.5 Proposed Normalization using redundant representation

In this thesis, we propose to normalize the state metrics with respect to state 0 instead of maximum or minimum state. In this scheme, the normalization block comprises subtraction only instead of comparison and subtraction. This means a decrease in the critical path delay. Moreover, this scheme removes the memory bank used to store state 0 metric. Table 5.3 illustrates the memory reduction due to this normalization scheme. It is shown that the proposed normalization scheme reduced the required storage by 6.7% of the branch metric memory and 12.5% of state metric memory.

Table 5-5 Comparison between number of storage bits of conventional and proposed schemes

	Conventional Normalization	Proposed Normalization
State metric memory bits	4096 bits	3584 bits

Additionally, we introduce a novel implementation for the proposed normalization. This is carried out via redundant representation of normalized state

metrics. In this scheme, instead of performing normalization after Add/Compare and Select operation, the un-normalized state metrics are forwarded to the next recursion. This form of un-normalized metrics is a redundant representation of the normalized metric. The normalization step is combined with the addition of the next recursion in one step. The key idea behind improvement of this implementation is that the CPA delay is converted to a CSA delay which is significantly lower than CPA delay.

In this case,

$$A_k(s_j) = \max\{A_{k-1}(s_i) + \Gamma_{k-1\to k}(s_i \to s_j) - A_{k-1}(0)\} \quad (5.9)$$

$$B_k(s_j) = \max\{B_{k-1}(s_i) + \Gamma_{k-1\to k}(s_i \to s_j) - B_{k-1}(0)\} \quad (5.10)$$

The critical path of the proposed implementation implies 1 CSA, 1 CPA, Comparison and MUX as shown in Figure 5.17. The double line arrow represents an operand in redundant format.

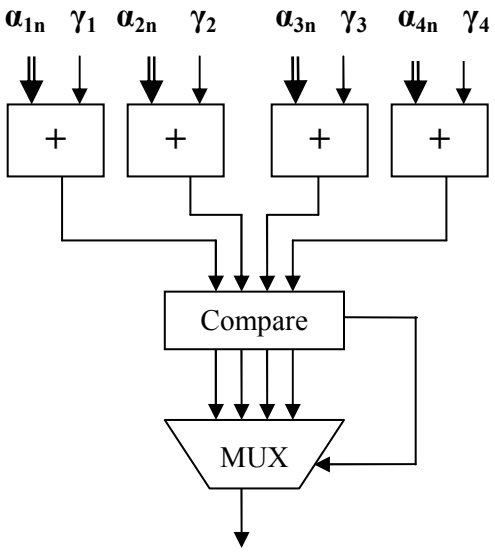


Figure 5.17 Reduced State metric unit

A further reduction in worst case delay is achieved by taking advantage of full redundancy. This is carried out by removal of the CPA. In this case, we deal with the computed values in redundant format as a separate sum and carry vectors.

Comparison stage has its inputs and outputs in redundant format and the output of this unit is also in redundant format.

The worst case delay in this case comprises 3 CSAs, redundant comparison and MUX stage as shown in Figure 5.18.

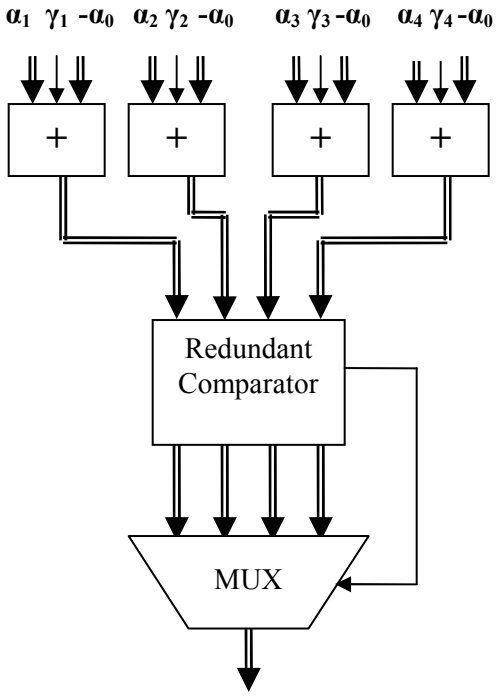


Figure 5.18 full redundant reduced State metric unit

The redundant comparator is implemented such that it has two stages; each stage has a delay which is considered O(log(n)). To illustrate the operation of the comparator that deals with redundant operands, we present the ordinary comparator with delay O(log(n)) and then show how we extend it to handle redundant operands. The key idea of the O(log(n)) comparator that compares between *X*, *Y* is to generate two signals *L* (stands for Larger than), *E* (stands for Equal to) at each bit position such that :

```
if( X(i)> Y(i))
        L(i) = 1
else
        L(i) = 0
if( X(i)= Y(i))
        E(i) = 1
else
        E(i) = 0
```
List 5.2

The next step is to combine two neighboring bit positions to generate a second level *L1*, *E1* signals such that:

$L1(j) = L(2j+1) + (L(2j).\ E(2j))$

$E1(j) = E(2j+1).\ E(2j)$

At each step, the number of *L*, *E* signals is halved until we reach to the final decision. This takes a delay of log(n). Implementation of the above procedure is carried out via simple logic gates.

The ordinary comparison is based on that $X(i), Y(i) = \{0,1\}$. In case of operand in redundant format, we have $X(i), Y(i) = \{0,1,2\}$. The operation of generating *L*, *E* signals is illustrated in Table 5-6.

Table 5-6 Comparison between ordinary and redundant comparator

	Ordinary Comparator	Redundant Comparator
$L(i)=1$	$X(i)=1$ and $Y(i)=0$	$X(i)=2$ and $Y(i)=0$ $X(i)=2$ and $Y(i)=1$ $X(i)=1$ and $Y(i)=0$
$E(i)=1$	$X(i)=1$ and $Y(i)=1$ $X(i)=0$ and $Y(i)=0$	$X(i)=2$ and $Y(i)=2$ $X(i)=1$ and $Y(i)=1$ $X(i)=0$ and $Y(i)=0$

It is shown that the difference between the ordinary and redundant comparator occurs only in first step, the remaining steps are similar.

A further optimization of the critical path delay is suitable by taking into consideration that the comparison does not depend on the operand $-\alpha_0$. We can combine addition of $-\alpha_0$ with comparison. This results in a removal of 2 CSA levels from the critical path. The final architecture of the SMU will be as shown in Figure 5.19.

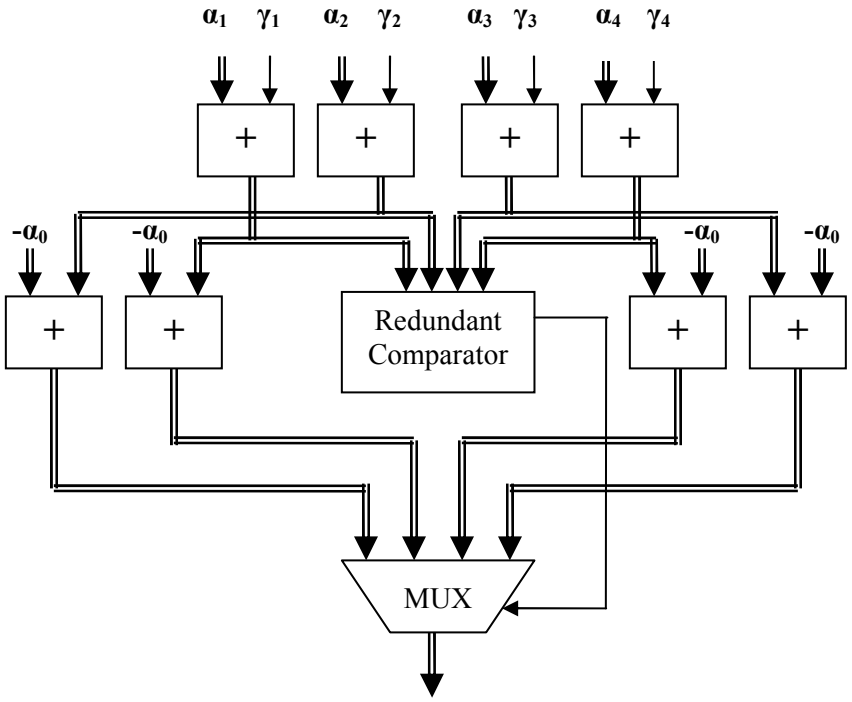


Figure 5.19 Enhanced full redundant State metric unit

The drawback of our proposed normalization technique is the increase in the area due to the increase in the number of CSAs and comparators that deal with redundant operands. Another drawback is the increase in memory as we need to store state metric of state 0. However, in order to preserve memory storage, we propose to normalize states by subtraction before storing into memory. This is performed via a 2-stage pipelined architecture as shown in Figure 5.20

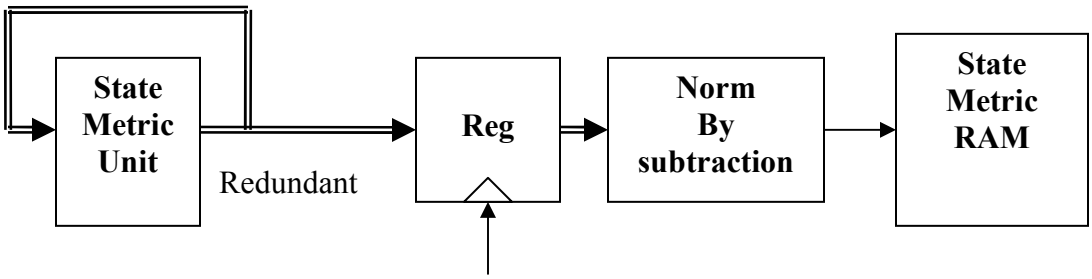


Figure 5.20 Proposed State Metric RAM interface

These different implementation techniques are tested using Mentor Graphics Precision RTL synthesis tool. The design platform is Altera-STRATIX II FPGA, EP2S15F484C family. The synthesis results are performed before place and route. Table 5-7 represents area and delay report of the four different architectures; Normalizing with respect to maximum or minimum, normalization with respect to state 0, redundant representation of normalized state metrics, and full redundant architecture.

Table 5-7 Area-Delay report for different state metric architectures

	Normalize to minimum or maximum	Normalize to state 0	Redundant Normalized state metrics	Full redundant architecture
Area (Number of LUTs)	812	644	928	1424
Critical path delay	17.88 ns	11.477 ns	11.279 ns	8.26 ns
Maximum Frequency	55.928 MHZ	87.13 MHZ	88.66 MHZ	121.065 MHZ

The results in Table 5-7 indicate that the second architecture is the best area-saving architecture and the fourth one is the best delay-saving one. The full redundant architecture increases the maximum frequency with 113.7% over the first architecture and 37.65% over the second one. However, it increases the area by 75.49% over the first architecture and 123% over the second one. We conclude that the redundant representation speeded up the operation at the cost of increasing the hardware area.

5.3.4 Backward Metric Unit

The backward state metric unit implementation is similar to that of forward state metric unit, except that no need to use extra memory to store backward state metrics. Some implementations consider one unit to be used for both forward and backward state computation, however, we need to take advantage of full speed SISO architecture, so separate unit are assumed in our implementation. Moreover, in our proposed implementation, LLRs are computed as soon as Backward metrics are ready. At the beginning of the traceback for each sliding window, all backward metrics are assumed to be equiprobable, at the last window, we initialize metrics such that circulation state Sc has the largest metrics.

5.3.5 LLR Computation Unit

The purpose of this unit is to calculate the soft output LLRs. In order to calculate the three LLRs as explained in section 3.3.6, we need to calculate four soft outputs as given by equation (3.40).

$$T_k(a,b) = \max\{\mathrm{A}_{k-1}(s_i) + \Gamma_{k-1\to k}(s_i \to s_j) + \mathrm{B}_k(s_j)\}$$

For each value of $u_k = (a,b)$, we have 8 corresponding branches on which we add corresponding forward, branch and backward metrics for each, then select the maximum value. This is carried out via ACS unit as shown in Figure 5.21.

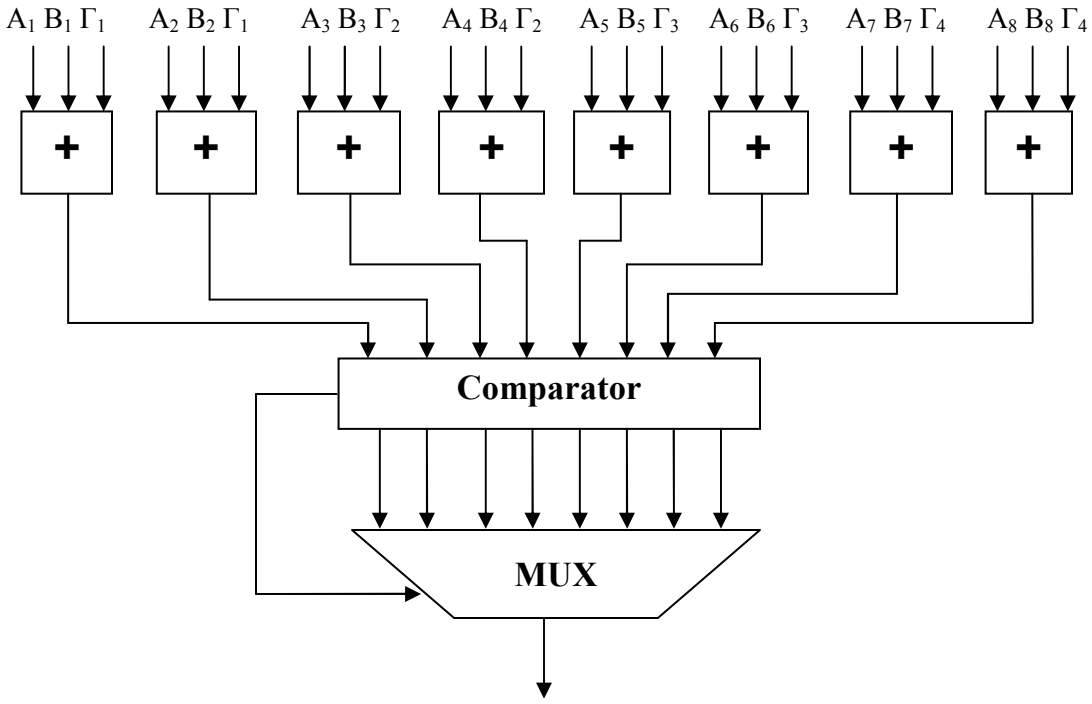


Figure 5.21 LLR Computation unit

In our case, we need to calculate four values; one corresponding to each symbol $u_k = (a,b)$. After this step, normalization of $T_k(a,b)$ with respect to $T_k(0,0)$ takes place in order to calculate the three LLRs. After calculation of LLRs, extrinsic LLRs should be calculated and final estimated bits are also calculated. However, our proposed implementation combines normalization of LLRs with the calculation of extrinsic LLRs in one step.

5.3.6 Extrinsic LLR Computation Unit

Extrinsic LLRs represent the a-priori information that is bypassed from one component decoder to the other component decoder. The calculation of the extrinsic LLR is carried out through subtraction of input systematic and extrinsic LLR from the corresponding output obtained LLR as follows

$$
\begin{aligned}
L^{o}_{e,k}(0,1) &= T_k(0,1) - T_k(0,0) - R_B - L_{e,k}(0,1) \\
L^{o}_{e,k}(1,0) &= T_k(1,0) - T_k(0,0) - R_A - L_{e,k}(1,0) \\
L^{o}_{e,k}(1,1) &= T_k(1,1) - T_k(0,0) - R_A - R_B - L_{e,k}(1,1)
\end{aligned}
\tag{5.11}
$$

The normalization of the LLRs is combined in the calculation of extrinsic LLRs. This has the benefit of converting CPA needed for normalization into a CSA, which should have much smaller delay.

The problem of the calculation of extrinsic LLRs is the increase of the dynamic range with the increase in the number of iterations. Consequently, this increases the number of bits of extrinsic LLRs, branch metrics and state metrics. In order to resolve this problem, saturation of extrinsic likelihoods is carried out. This is implemented through a saturating adder/subtractor. Its main function is to saturate at the maximum or minimum values in case of overflow, so that it guarantees that the output is in the range $\left[-\frac{2^n}{2}, \frac{2^n}{2}-1\right]$ for n-bit precision. The main issue is to select the minimum suitable number of bits to represent extrinsic likelihoods, and preserve good performance at the same time. Fixed point analysis indicates that a 6-bit representation is considered the optimum number of bits for extrinsic likelihoods.

As shown in Figure 5.22, each of the three extrinsic likelihoods is calculated via multi-operand addition followed by a MUX for saturation purposes. For $L^{o}_{e,k}(0,1)$ and $L^{o}_{e,k}(1,0)$, the multi-operand adder consists of 2 CSA levels followed by 1 CPA level. The multi-operand adder of $L^{o}_{e,k}(1,1)$ consists of 3 CSA levels followed by 1 CPA level.

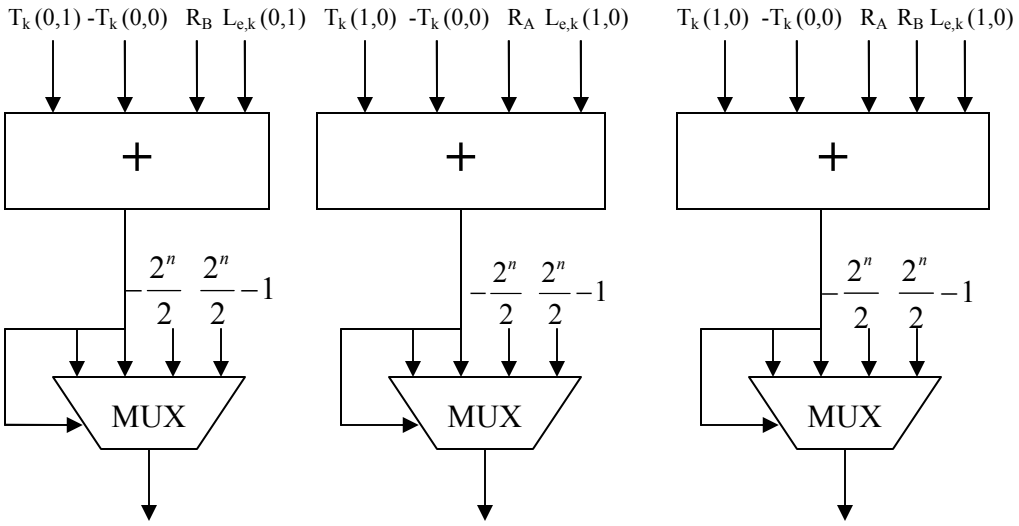


Figure 5.22 Extrinsic LLR computation unit

Extrinsic likelihoods are used by the next component decoder as a-priori information in improving the decoding estimation. In SW-Log MAP with our proposed architecture, the obtained likelihoods are in the reverse order as they are generated in the backward recursion phase. Some implementations proposed to use a Last Input First Output (LIFO) for likelihoods after they are generated [31]. However, this has its drawback in increased latency. In this thesis, we propose a lower latency implementation. It depends on passing the extrinsic likelihoods through an interleaver / deinterleaver before the second component decoder. In order to remove latency from LIFO block, we propose that generated likelihoods are passed directly through the interleaver / deinterleaver. On one hand, this permits the removal of the LIFO latency. On the other hand, we can not use the address generator in section 5.2.2.2. This forces us to use the LUT implementation of the address generator specified in 5.2.2.1 which consumes a larger memory area.

5.4 Synthesis Results

In order to test our implementation for satisfying performance requirements, all the implemented blocks are synthesized on Altera FPGA. The target device is Altera StratixII EP2S15F484C3 using Quartus II software tools, targeting optimization for speed. We obtain the following results as indicated in Table 5-8

Table 5-8 Synthesis results for CTC encoder

Block	Number of LUTs	Number of Registers	Number of Memory bits	Maximum Frequency of operation
Constituent encoder	5	5	—	Maximum achieved (500 MHZ)
CTC interleaver	139	83	1024	194 MHZ
Sc ROM	—	—	192	—
Subblock interleaver	73	32	3072	177 MHZ
CTC encoder	232	200	6480	164 MHZ

From the results given in Table 5-8, we conclude that our implementation for turbo encoder blocks has around 2% Logic utilization, with an operating frequency much higher than that required by WiMAX.

Table 5-9 Synthesis results for Turbo decoder components

Block	Number of LUTs	Number of Registers	Number of Memory bits	Maximum Frequency of operation
Branch metric Unit	244	149	6208	384.32 MHZ
State Metric Unit	699	120	3584 (For Forward state unit only)	154 MHZ
2-stage pipelined LLR Computation Unit	610	237	—	204.58 MHZ
Extrinsic LLR Computation unit	118	84	—	304 MHZ
SISO + Interleaver / Deinterleaver	2926	1112	36704	150.15 MHZ

From the above results, we conclude that our SISO component can be used four times and satisfies the timing requirements of the IEEE 802.16e standard. This means that we can use one SISO block to achieve two successive decoding iterations. In order to have four decoding iterations, two SISO blocks are required. There are other proposed architectures in the literature. The WiMAX CTC decoder architecture proposed in [35] targets Xilinx XC4VLX80-11FF1148 chip and operates at 125MHZ. However, the CTC decoder proposed in [35] supports H-ARQ, but our decoder does not support it. The main difference between HARQ support and non HARQ support is that the HARQ supports interleaver block sizes up to 2400 couples, but in our case, the maximum CTC block size is 240 couples (480 bits). This has the impact on the interleaver memory size. The WiMAX CTC decoder proposed in [36] operates at maximum frequency of 200 MHZ, but it targets 0.18 μm 4-Metal CMOS standard cell.

Chapter 6

Sampling clock and Frequency Tracking

6.1 Introduction

Synchronization in OFDM systems has been a crucial issue. OFDM systems are much more sensitive to offset in carrier frequency than single carrier schemes with the same bit rate. Mis-synchronization leads to a loss of orthogonality among different subcarriers, and hence we have the problem of ICI.

Good synchronization techniques play a key role in system performance, and they drive the need of efficient implementation techniques. Many techniques have been proposed in order to handle OFDM synchronization. OFDM synchronization can be basically divided into Symbol (Timing) synchronization and Carrier frequency synchronization.

Timing Synchronization in OFDM systems is used in order to achieve synchronization and alignment to the received OFDM symbol windows. The mis-synchronization can lead to a severe effect in decoding. The OFDM Timing Synchronization comprises three steps; Frame detection, Fine symbol timing and Sampling clock frequency tracking. The first step, Frame detection, is responsible for detecting an incoming frame at the receiver terminal. This is performed by continuously sensing the energy at the receiver input and comparing it to a threshold. The second step is fine symbol timing, which is responsible for detection of the beginning and end of the OFDM symbol. It represents a fine estimation over the first step. More information about symbol timing techniques can be found in [11], [37]. The third step, in contrast, is responsible for tracking the sampling clock frequency error that occurs between sampling clock at Digital to Analog Converter (DAC) at transmitter and sampling clock at Analog to Digital Converter (ADC) at receiver. In this thesis, we consider only the sampling clock frequency tracking step. We represent the effect of the sampling error on the

received subcarriers and show an algorithm used to correct this sampling error. Finally, the hardware implementation of this algorithm is represented. The Hardware implementation of the Frame detection and Symbol Timing blocks is described in [13].

Similar to the Timing synchronization, the Frequency synchronization is used to compensate for the effect of frequency error between local oscillator at transmitter and local oscillator at receiver. It also comprises three steps; Coarse Frequency offset estimation, Fine Frequency offset estimation and Residual Carrier Frequency offset tracking. The frequency offset can be divided into an integer part and a fractional part. Fine frequency offset is responsible for estimation of the fractional part and coarse frequency offset is used in estimation of the integer part [38]. The frequency offset tracking is used to further compensate for mis-estimation that may occur from the two previous steps, or the continuous variation of oscillator frequency that may depend with environmental conditions. In this thesis, we concern with the frequency tracking step.

6.2 Effect of sampling clock frequency offset

Sampling Clock Frequency Offset (SCFO) occurs as a result of difference of oscillator frequencies at transmitter DAC and receiver ADC. This offset has its effect in both time and frequency domains. Figure 6.1 illustrates the sampling error phenomena with solid lines indicating exact sampling time slots, and dashed lines indicating sampling time slot drift due to sampling error. In IEEE 802.16e, it is specified that at the station set, the sampling clock frequency shall be synchronized and locked to the base station (BS) with a tolerance of maximum 5 parts per million (ppm) as specified by IEEE 802.16e standard [7]. The SCFO has its impact in both time domain and frequency domain. In the time domain, it causes a drift in the OFDM symbol window. In the frequency domain, SCFO

causes a change in subcarrier phases. The two effects should be handled. In order to handle the effect of SCFO, many techniques were proposed, some depend on using closed loop techniques based on Delay Locked Loop (DLL) [39]. Other techniques based on open loop synchronization [40], [41].

Open loop techniques that depend on pilot subcarriers or preambles are suitable for digital implementation platforms. The next section describes in details the effect of SCFO in both time domain and frequency domain then the tracking algorithm is described.

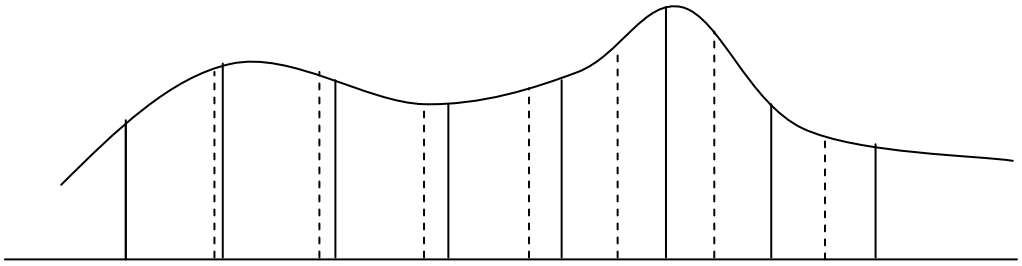


Figure 6.1 Sampling error phenomena

6.2.1 Effect of sampling error in time domain

In the time domain, the effect of SCFO appears as a drift in the OFDM symbol window; this drift accumulates each OFDM symbol. After a while, this drift will cause irreducible error that can not be recovered in the frequency domain. The operation of OFDM symbol window drift can be described as follows:

For an OFDM symbol with N_s samples, if the OFDM symbol index is l, then the expected interval is $[(l\text{-}1)N_s\ ,\ lN_s\ ]$, but due to the drift, it will be $[(l\text{-}1)(1+\Delta)N_s\ ,\ l\ (1+\Delta)\ N_s\ ]$ as shown in Figure 6.2. It is obvious that the total drift in time domain is a factor of the symbol index l. In fact the problem will occur if the total drift exceeds half the sample time. In this case, one sample should be added if the sampled version is faster than the original or dropped if the sampled version is slower. This operation is defined as ROB/STUFF or ADD/DROP mechanism.

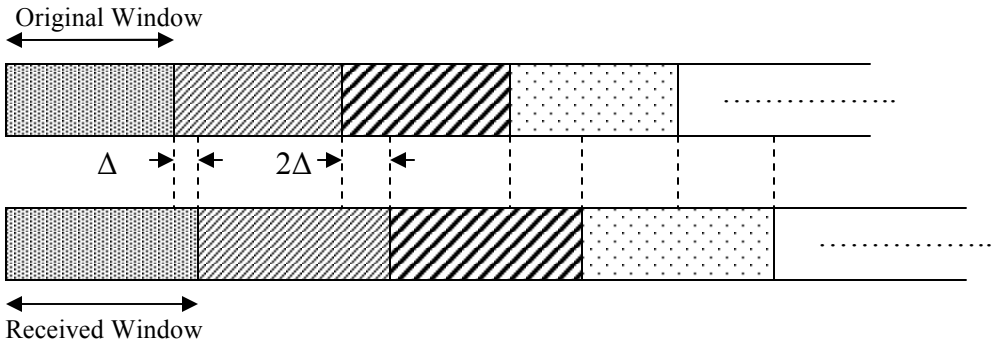


Figure 6.2 OFDM Symbol window drift

6.2.2 Effect of sampling error in frequency domain

SCFO represents a time error between sampling time T_s at transmitter and sampling time T_r at receiver. This offset in time will be converted to a phase shift in subcarrier phases in the frequency domain after the FFT block at receiver. The effect of phase rotation in the frequency domain can be expressed in a mathematical form as follows:

Let N_u be the useful number of samples in one OFDM symbol window, it should be equal to FFT size.

n is the sample index in a certain OFDM symbol, $-\frac{N_u}{2} \le n \le \frac{N_u}{2} - 1$

$N_s = N_u + N_g$ is the total number of samples of OFDM symbol window in time domain including useful samples N_u and guard interval samples N_g.

m is the sample index in the time domain, which can be expressed as

$$m = lN_s + n - \frac{N_u}{2} \qquad l = 1,2,3,.... \tag{6.1}$$

Then, for a certain OFDM symbol with index l, a subcarrier with index K is expressed as

$$X(K) = \sum_{n=-\frac{N_u}{2}}^{\frac{N_u}{2}-1} x(m).\, e^{-j\frac{2\pi K n}{N_u}}$$

After applying the effect of SCFO, we have the new time index is $m(1+\Delta)$ instead of m. In this case, a subcarrier with index K is expressed as

$$X'(K) = \sum_{n=-\frac{N_u}{2}}^{\frac{N_u}{2}-1} x'(m).\, e^{-j\frac{2\pi K\left[n(1+\Delta)+\left(l.N_s-\frac{N_u}{2}\right)\Delta\right]}{N_u}} \tag{6.2}$$

where Δ is the relative sampling error and it is equal to

$$\Delta = \frac{T_r - T_s}{T_s} \tag{6.3}$$

$$X'(K) = e^{j2\pi K\left(l\frac{N_s}{N_u}-0.5\right)\Delta} \sum_{n=-\frac{N_u}{2}}^{\frac{N_u}{2}-1} x'(m).e^{-j\frac{2\pi K(n(1+\Delta))}{N_u}}$$

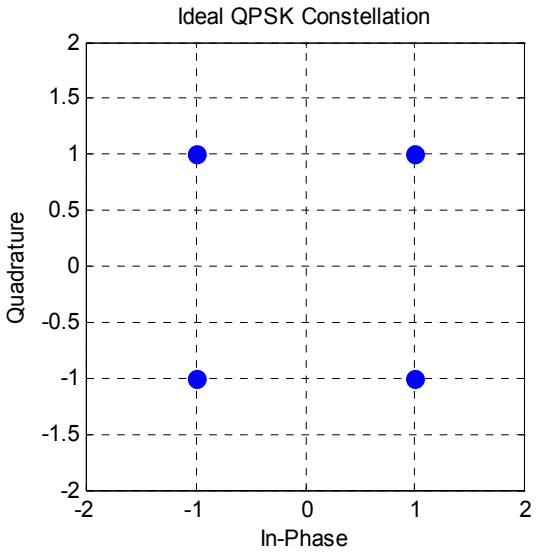
By neglecting the value of the relative sampling error Δ with respect to 1 in the exponent, we get

$$X'(K) \approx e^{j2\pi K\left(l\frac{N_s}{N_u}-0.5\right)\Delta} X(K) \tag{6.4}$$

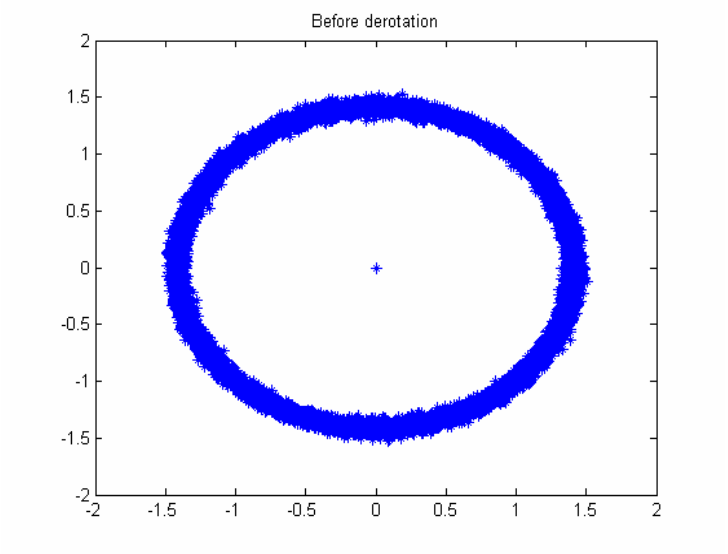
We conclude that the effect of sampling error represented by a delay in the time domain is converted to a linear phase shift in the frequency domain. A similar results can be obtained from [41].

It can be proven from equation (6.4) that the phase error line is approximately equal to $e^{j2\pi k(l\frac{N_s}{N_u}-0.5)\Delta}$; this means that the first OFDM symbol has a phase error with slope $2\pi\left(\frac{N_s}{N_u}-0.5\right)$, the second OFDM symbol has a phase error with slope $2\pi\left(2\frac{N_s}{N_u}-0.5\right)$, and so on. Figure 6.4 plots the phase error with the subcarriers for successive OFDM symbols.

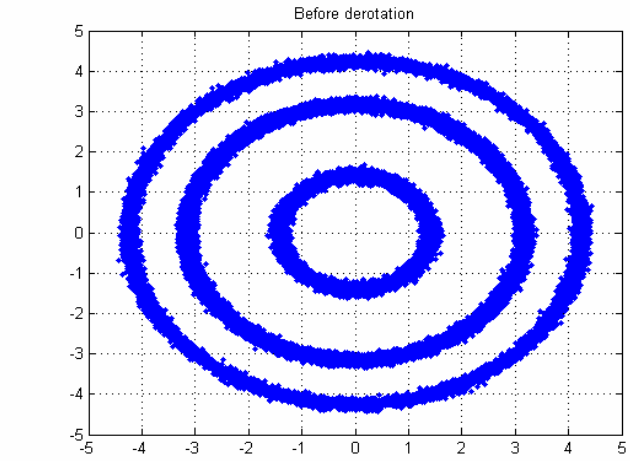
This phase shift is represented by a rotation to the constellation diagram as indicated in Figure 6.3.a,b,c. Figure 6.3.a shows the ideal QPSK constellation where Figure 6.3.b and Figure 6.3.c show the effect of SCFO on rotating the constellation for QPSK and 16-QAM respectively.



(a) Ideal QPSK constellation



(b) Rotated QPSK constellation



(c) Rotated 16-QAM constellation

Figure 6.3 (a) Ideal QPSK constellation (b) Rotated QPSK constellation (c) Rotated 16-QAM constellation

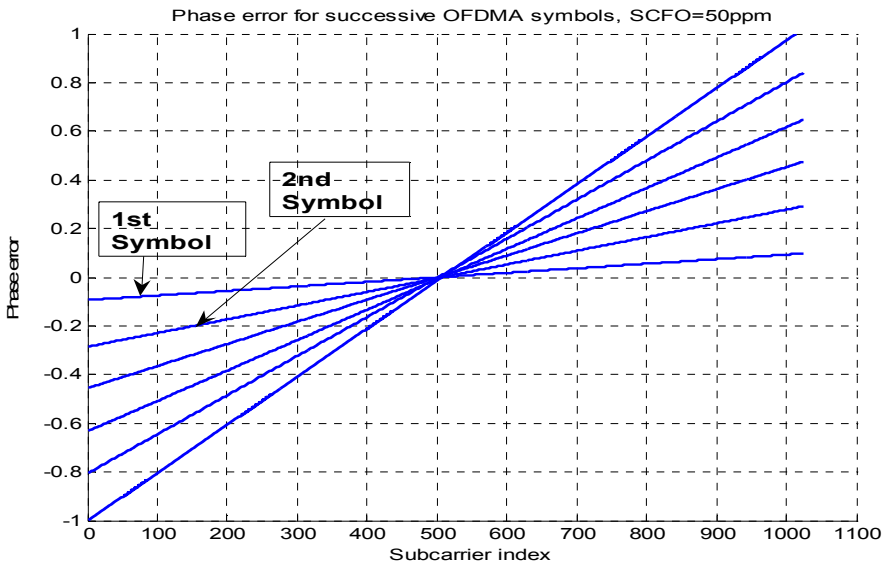


Figure 6.4 Phase error line for successive OFDM symbols

6.2.3 SCFO Synchronization algorithm

SCFO synchronization implies two steps: Correcting phase error in frequency domain, and correcting drift in time domain. This synchronization technique is carried out with the aid of pilot subcarriers and was proposed in [41].

The key idea behind this algorithm is to use the pilot subcarriers to estimate the phase rotation of the data subcarriers. After this estimation, a derotation of data subcarriers is carried out to compensate for the effect of SCFO error. At the same time, the add/drop mechanism is done via controlling the length of the removed CP at the receiver before the FFT operation. The number of removed samples of the CP can be either increased or decreased according to the drift of the OFDM symbol window.

Many techniques have been proposed to estimate the phase rotation of the pilot subcarriers. Some of these techniques, as mentioned in [11], depend on cross correlation between pilots of a certain OFDM symbol with pilots of the previous OFDM symbol. However, in case of 802.16e, we find that in some permutation

schemes, pilot locations can differ among successive symbols. It can be defined in a certain set for odd OFDM symbols and another set in even symbols. The case studied in this thesis is the most commonly used FUSC permutation with FFT size 1024.

The used technique depends on estimating the pilot phases for each OFDM symbol separately. This is carried out through cross correlation of the received pilot subcarriers with the pre-known transmitted pilot subcarriers. This can be described as follows:

Let $P_{k,l}$ be the modulated pilot subcarrier with index k for OFDM symbol with index l , the received pilot subcarrier is indicated as $Rp_{k,l}$ such that

$$Rp_{k,l} = P_{k,l}.e^{j\phi} \tag{6.5}$$

We obtain the angle rotation

$$\phi_{k,l} = \angle\left(P_{k,l} \,.\, \mathrm{conj}\,(Rp_{k,l})\right) \tag{6.6}$$

The value of $\phi_{k,l}$ is calculated for all pilots. The next step is to estimate the equation of phase error line in order to estimate phase rotation of other data subcarriers. The most accurate used algorithm is to fit the obtained pilot phases to the nearest line. This is done via Least Square (LS) Linear Curve Fitting algorithm.

6.2.3.1 Phase tracking via LS linear curve Fitting

The Least Square (LS) algorithm is used to obtain the best curve *f(x)* that fits to a set of points (x_i, y_i). The linear curve fitting is used to obtain the best straight line that fits to some set of points. The key idea behind this is that it minimizes the error between line and data points as follows:

For a set of points (x_i, y_i), and the line equation $f(x_i) = ax_i + b$, we can define the sum of squared errors as:

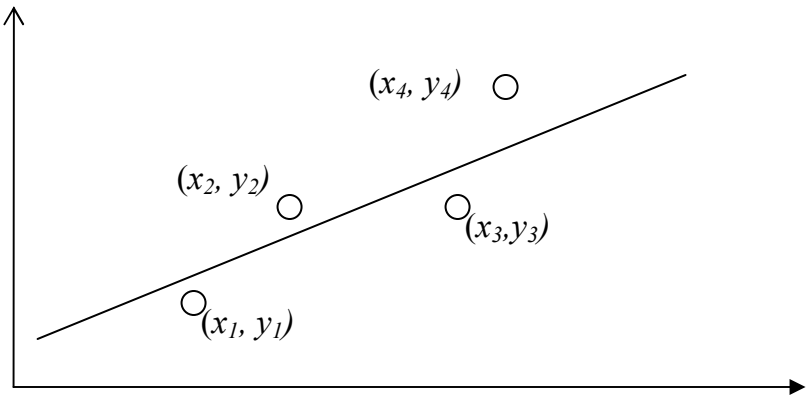


Figure 6.5 LS linear curve Fitting

$$err = \sum \left(y_i - f(x_i) \right)^2$$

$$err = \sum \left(y_i - ax_i - b_i \right)^2 \tag{6.7}$$

The mission of LS algorithm is to calculate *a, b* coefficients such that the error is minimized.

$$\frac{\partial err}{\partial a} = -2 \sum x_i \left(y_i - ax_i - b_i \right) = 0 \tag{6.8}$$

$$\frac{\partial err}{\partial b} = -2 \sum \left(y_i - ax_i - b_i \right) = 0 \tag{6.9}$$

Solving equations(6.8), (6.9) we can re-write them as follows

$$a \sum x_i^2 + b \sum x_i = \sum x_i . y_i$$

$$a \sum x_i + b * n = \sum y_i$$

Then, the obtained set of equations can be written in a matrix form as

$$\begin{bmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} \begin{bmatrix} b \\ a \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum x_i y_i \end{bmatrix} \tag{6.10}$$

where n is the number of points.

In our case, y_i represents estimated phase of pilot with index x_i.

Phase estimation of the remaining data tones is carried out through the phase line equation

$$\phi^{e}_{k,l} = a \,.\, k + b \tag{6.11}$$

where $\phi^{e}_{k,l}$ indicates the estimated phase error of subcarrier with index *k* for OFDM symbol *l*. *a* is the slope and *b* is the bias or intercept.

The last step is to correct the phase error through subcarrier derotation

$$Z^{e}_{k,l} = Z_{k,l} \;.\; \exp\left(\text{-j}\; \varphi^{e}_{k,l}\right) \tag{6.12}$$

6.2.3.2 Symbol Re-timing with ROB/STUFF

The next step with phase tracking is called symbol re-timing. It plays a key role in synchronization process as it compensates for the drift caused to OFDM symbol window. Symbol re-timing is performed through controlling the length of the removed CP before the FFT operation. This also is called ADD/DROP mechanism. The process of removing one extra symbol to the CP or dropping one symbol is needed when the drift in the OFDM symbol window exceeds $\frac{1}{2}T_s$. It can be proven that a drift in the OFDM symbol window will exceed $\frac{1}{2}T_s$ when the difference in phase error between the first and last subcarriers in the same symbol exceeds a value of π. This procedure is described as follows:

For each OFDM symbol of index *l*

$$If\ (\varphi^{e}(\frac{FFT\_SIZE}{2}-1,l)-\varphi^{e}(-\frac{FFT\_SIZE}{2},l)\geq\pi)\ then$$

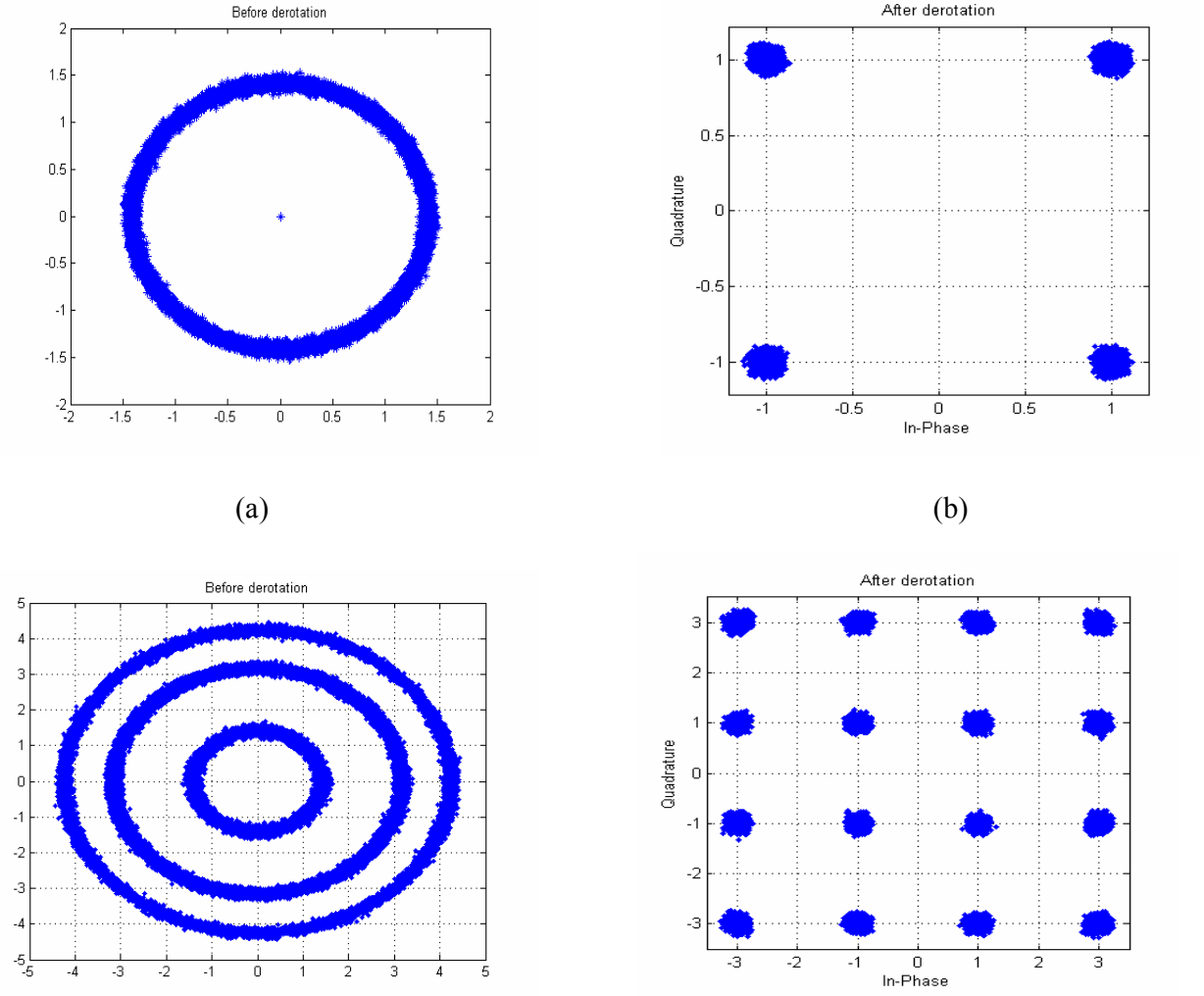
Remove CP-1

$$If\ (\varphi^{e}(\frac{FFT\_SIZE}{2}-1,l)-\varphi^{e}(-\frac{FFT\_SIZE}{2},l)\leq-\pi)\ then$$

Remove CP+1

LS linear curve fitting is the best algorithm to estimate phase error line, as it is less sensitive to AWGN channel effects.

Figure 6.6 illustrates the resultant constellations before and after the phase recovery in constellation for QPSK and 16-QAM respectively.

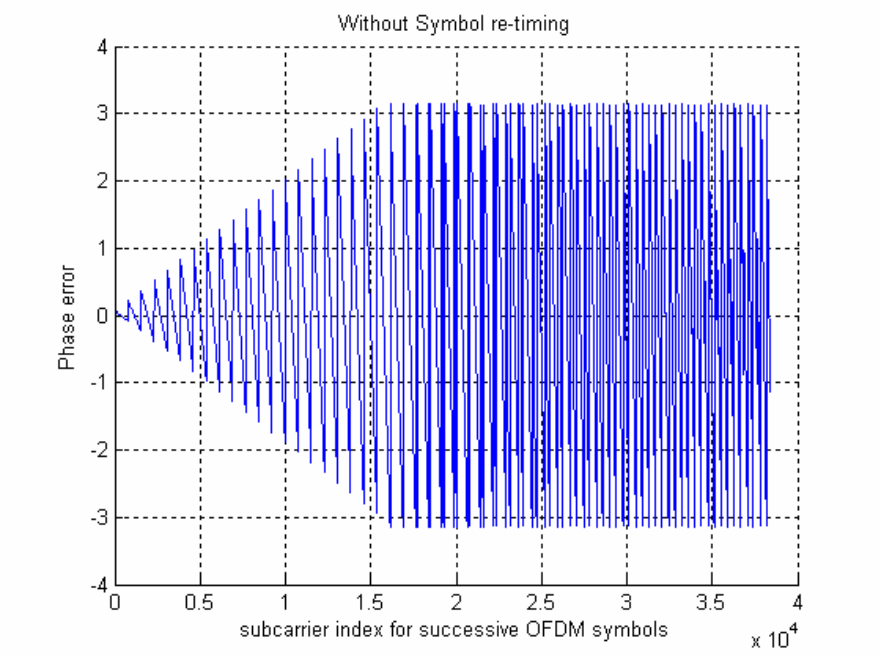


(a) (b)

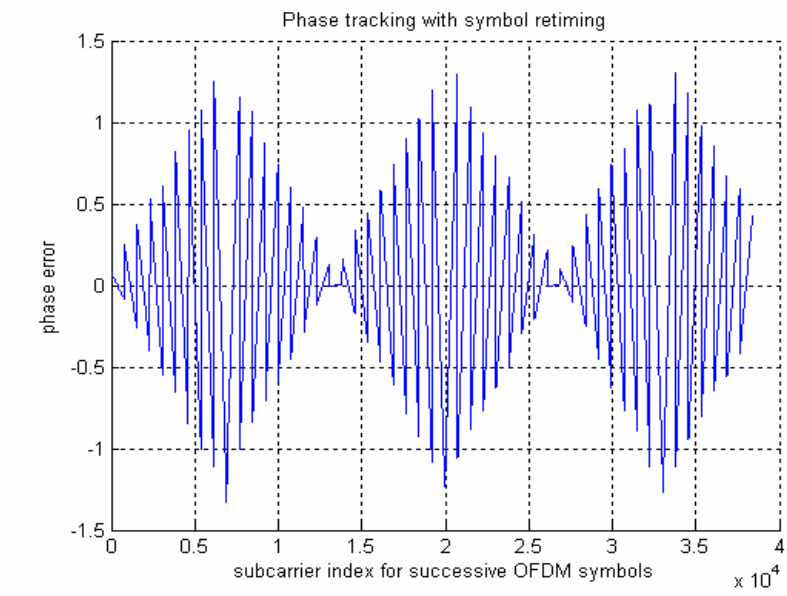
(c) (d)

Figure 6.6 (a) QPSK before de-rotation (b) QPSK after de-rotation
(c) 16-QAM before de-rotation (d) 16-QAM after de-rotation

In Figure 6.7, phase error tracking is indicated in case of symbol re-timing and without symbol re-timing. It is shown that without symbol re-timing, the phase error accumulates, until no further tracking can correct it.



(a)



(b)

Figure 6.7 (a) Phase tracking without Add/drop mechanism
(b) Phase tracking with Add/drop mechanism

6.3 Effect of Residual Carrier Frequency offset

In addition to the timing offset discussed before, OFDM is more sensitive to frequency offset than single carrier schemes. The main reason of the frequency offset is the mismatch between the local oscillator at the transmitter and the receiver. Other factors such as Doppler shift in high speed mobile systems, may participate in the increase of the frequency offset.

Frequency offset results in a loss of orthogonality among subcarriers, which results in the ICI. Many papers addressed the problem of frequency offset in OFDM systems. Some proposed techniques use the pilot subcarriers to estimate the frequency offset [42]. Others are proposed to use time domain techniques, such as redundancy in CP to estimate frequency offset [43]. Some approaches depend on Phase locked loop to correct the frequency offset [44].

The Residual Carrier Frequency offset (RCFO) is a result from a non-perfect estimation from the Coarse and fine frequency offset stages. The local oscillator carrier frequency may also change slightly due to environmental conditions. The function of the frequency offset tracking stage is to track RCFO and correct its effect. In this thesis, we focus on the frequency offset tracking stage, illustrating its effect and tracking algorithm.

The RCFO results in a phase offset in each subcarrier of every OFDM symbol. It can be seen as a rotation of the constellation axis in frequency domain. This is in contrast to the effect of SCFO, which has a linear phase error that varies with subcarrier index and OFDM symbol number. Figure 6.8 illustrates the effect of phase rotation due to RCFO in a certain OFDM symbol in case of 16-QAM modulation.

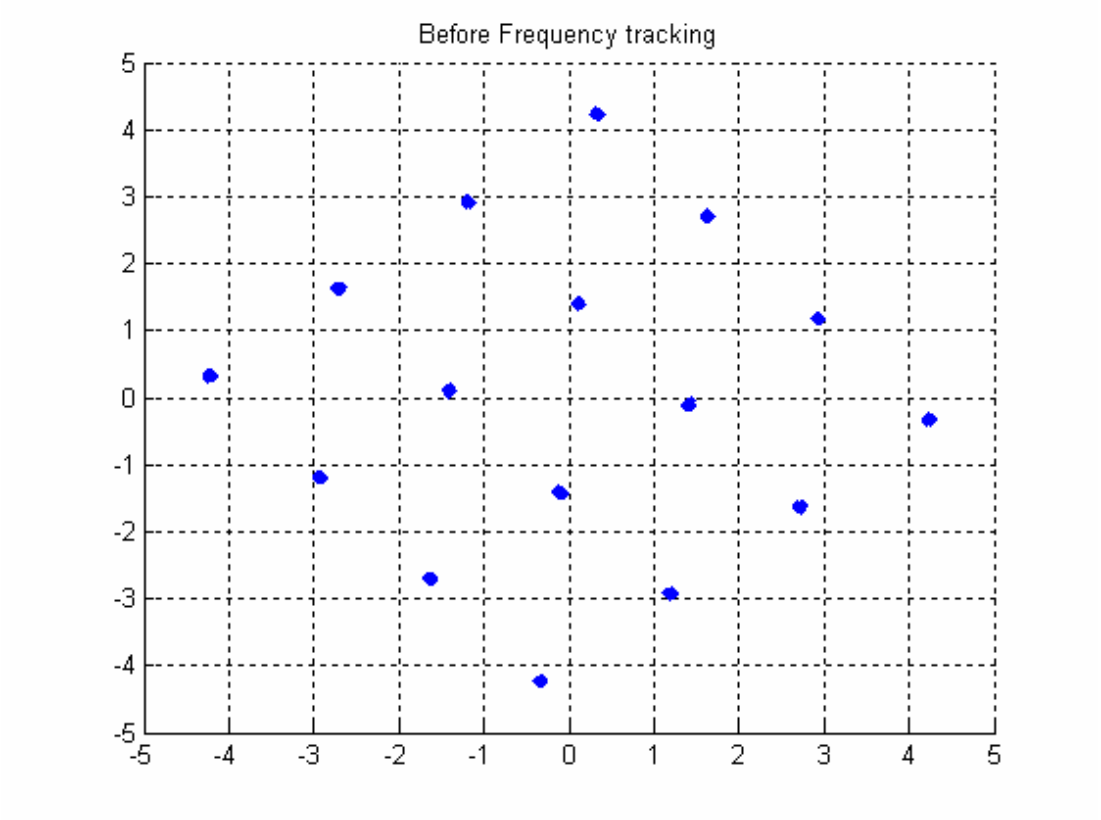


Figure 6.8 Constellation rotation due to RCFO

This effect can be derived as follows:

$$X(K) = \sum_{n=-\frac{N_u}{2}}^{\frac{N_u}{2}-1} x(n) e^{-j\frac{.2\pi kn}{N}}$$

In case of RCFO Δf, we obtain

$$x'(m) = x(m).e^{j.2\pi\Delta f m T_s}$$

where $m = n + lN_s - \frac{N_u}{2}$

$$X'(K) = \sum_{n=-\frac{N_u}{2}}^{\frac{N_u}{2}-1} x'(m) e^{-j\frac{.2\pi kn}{N}}$$

$$X'(K) = \sum_{n=-\frac{N_u}{2}}^{\frac{N_u}{2}-1} x(m).e^{-j\frac{.2\pi kn}{N}}.e^{-j\,2\pi\,\Delta f\left(lN_s+n-\frac{N_u}{2}\right)T_s}$$

$$X'(K) = e^{-j2\pi\Delta f\left(lN_s - \frac{N_u}{2}\right)T_s} \sum_{n=-\frac{N_u}{2}}^{\frac{N_u}{2}-1} x(n).e^{-j\frac{2\pi kn}{N}}.e^{-j2\pi\Delta fnT_s}$$

$$X'(K) = e^{-j2\pi\Delta f\left(lN_s - \frac{N_u}{2}\right)T_s} \sum_{n=-\frac{N_u}{2}}^{\frac{N_u}{2}-1} x(n).e^{-j\frac{2\pi n}{N_u}(k+\Delta fT_sN_u)}$$

$$X'(K) \approx e^{-j2\pi\Delta f\left(lN_s - \frac{N_u}{2}\right)T_s} X(K) \qquad (6.13)$$

It is shown that RCFO results in a phase offset that is proportional to OFDM symbol index. The added term $\Delta f.T_s.N_u$ is very small and can be neglected. Its effect begins to occur with higher values of Δf as an increase in ICI. In general, it should have a small value after coarse and fine frequency offset stages. The value of phase offset differs among successive OFDM symbols as shown in Figure 6.9.

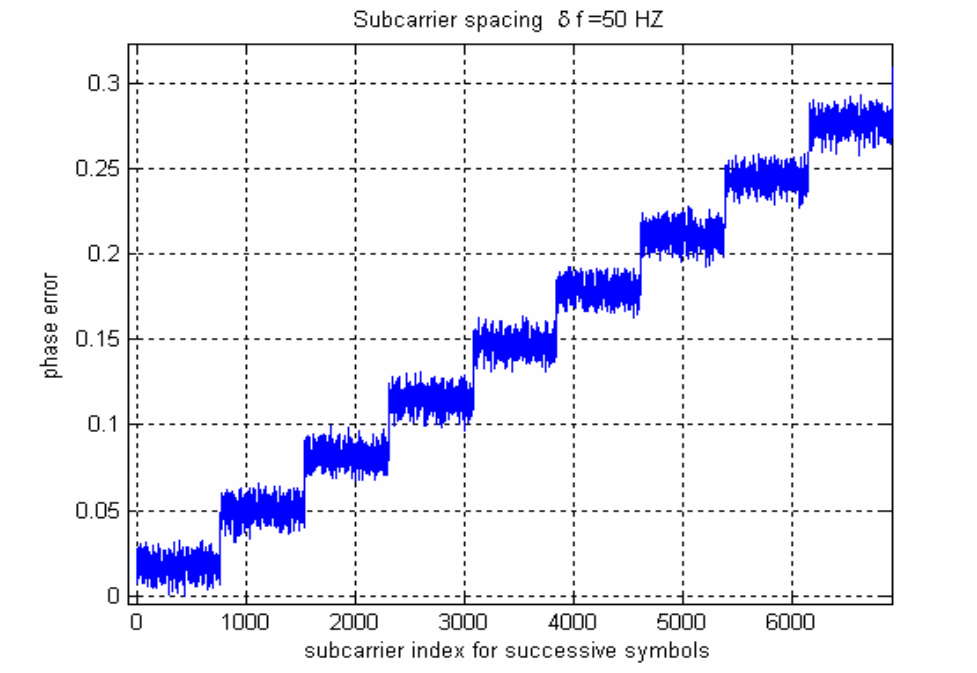


Figure 6.9 Effect of RCFO on phase error

We conclude that combining the effect of both RCFO and SCFO, the RCFO results in a bias in the phase error of each OFDM symbol, and SCFO determines the slope of phase error line for each OFDM symbol. The used technique for estimation and correction of SCFO can be used as a joint estimation of both SCFO and RCFO. The LS algorithm determines the bias and the slope of phase error in each OFDM symbol. The bias is caused mainly by RCFO and the slope is affected by SCFO. Figure 6.10 represents the phase error due to the combined effect of both RCFO and SCFO. The estimated phase of any tone will be the combined effect due to both RCFO and SCFO. Derotation of data subcarriers is a correction to both SCFO and RCFO effects.

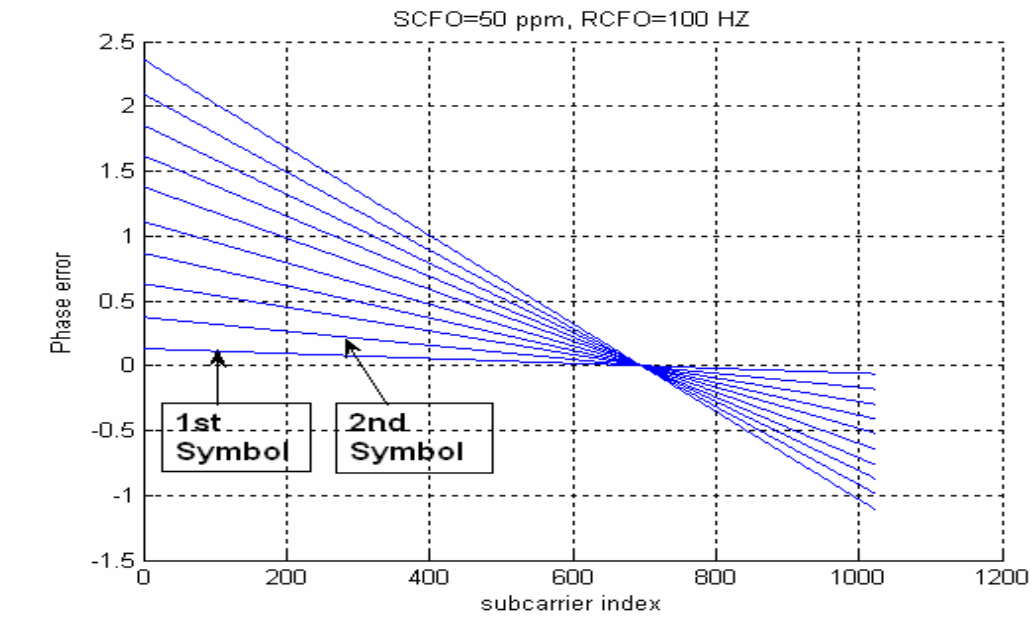


Figure 6.10 Phase error for combined SCFO and RCFO

6.4 Simulation results

6.4.1 LS algorithm performance

This section illustrates the performance of the LS estimation algorithm in case of RCFO and SCFO. Simulation parameters assumes the case of FUSC permutation scheme and FFT size =1024

Number of used subcarriers N_{used} = 851
Number of pilot subcarriers = 82
Number of left guard subcarriers = 87
Number of right guard subcarriers = 86
Subcarrier spacing = 10.94 kHZ.
Useful OFDM symbol time (T_u) = 91.4 μs
Total OFDM symbol time (T_s) = 102.9 μs
Cyclic Prefix (CP) = 1/8

It is shown the BER for different values of E_b/N_o for AWGN channel in Figure 6.11. Different values of SCFO and RCFO used in simulation are large enough compared to practical values. The LS algorithm is still efficient in more severe conditions.

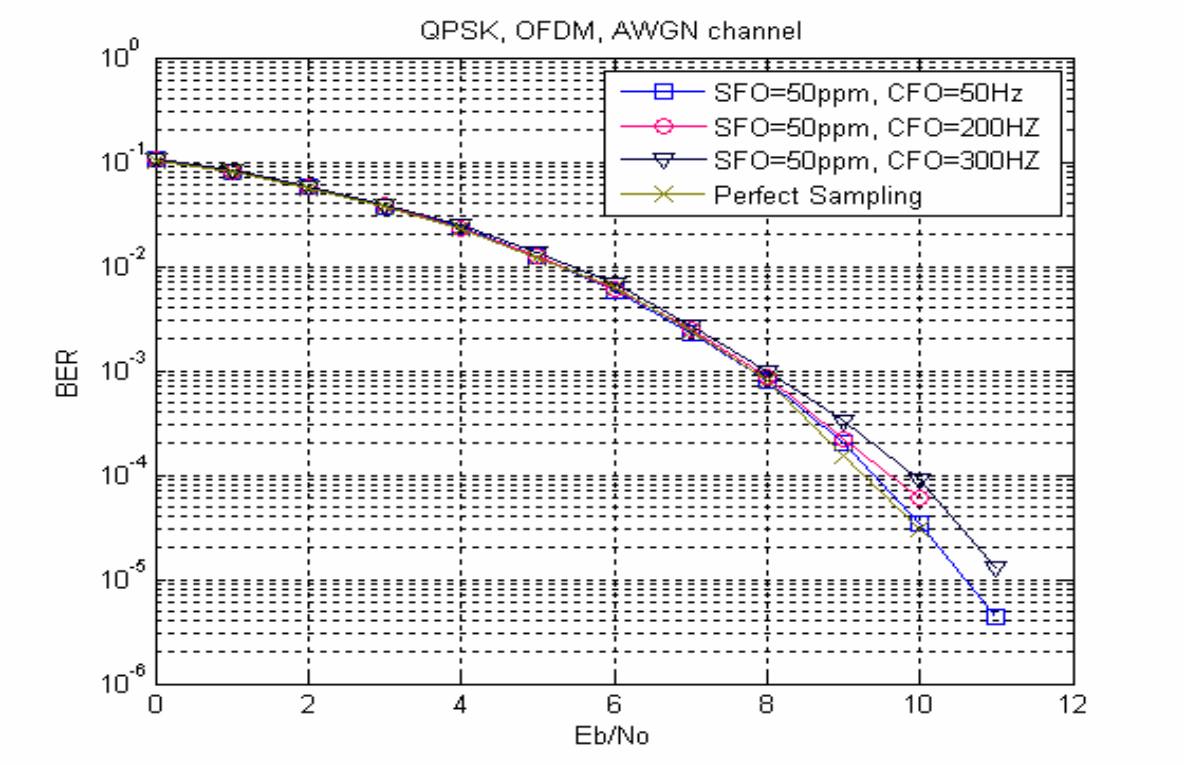


Figure 6.11 BER vs Eb/No for different RCFO values

6.5 Hardware Implementation:

In this section, we discuss the FPGA implementation of sampling clock and frequency tracking block.

6.5.1 Block diagram

The function of this block is to estimate the phase rotation of subcarriers and perform a derotation. This is performed by an estimation of pilot phases, followed by an estimation of the phases of other subcarriers. The required steps are

1- Estimation of Pilot Phases
2- Estimation of phase line coefficients
3- Estimation of data subcarriers phases
4- Subcarrier derotation

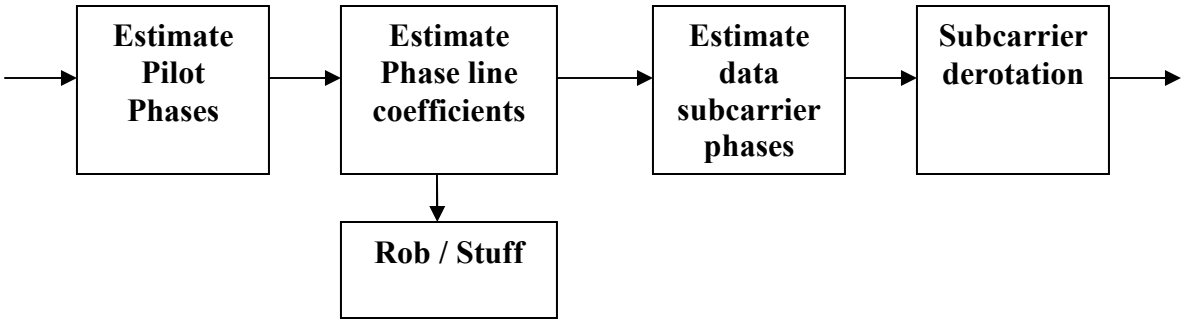


Figure 6.12 Sampling clock and frequency tracking block diagram

6.5.2 Pilot Phase estimation Block

This block is responsible for the estimation of received pilot phases, depending on the information of the transmitted pilots. Estimation of pilot phases can be implemented with the aid of CORDIC rotation. It is a simple algorithm that is used to rotate a vector with a certain phase through successive Add and Shift operations.

Inputs to this block are *Rx_re*, *Rx_im* that represent real and imaginary parts of received subcarrier, pilot_flag signal is activated to indicate that the received subcarrier is a pilot subcarrier. The output of this block is *Rot_rx_re*, *Rot_rx_im* that represent real and imaginary part of rotated subcarrier respectively. *Est_angle* is the estimated angle of pilot subcarrier, which will be used by next block to estimate data subcarriers. *Valid_out* signal is activated once the output is valid.

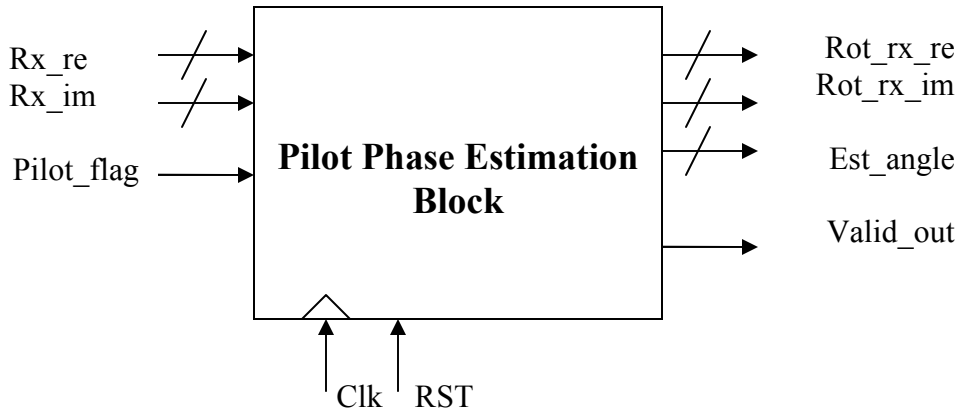


Figure 6.13 Phase estimation block diagram

6.5.2.1 CORDIC algorithm:

CORDIC stands for **CO**ordinate **R**otation **DI**gital **C**omputer. It is used as an alternative to complex multiplication to rotate vectors [34].The key idea behind CORDIC operation can be described as follows

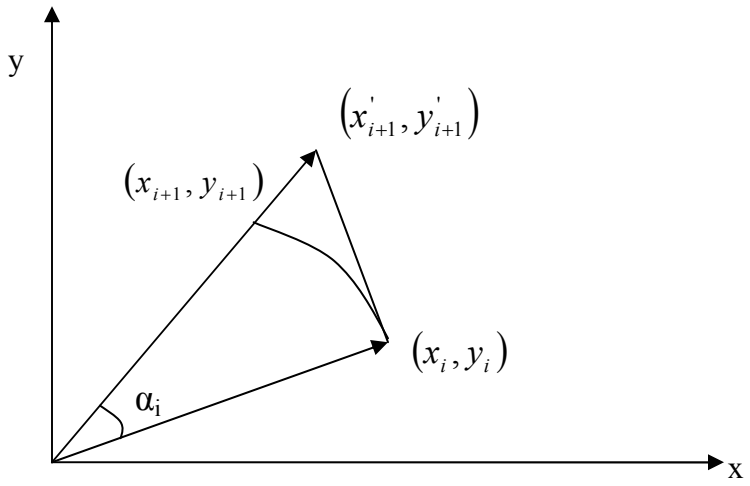


Figure 6.14 Basic CORDIC rotation

If the initial value of a certain vector is given as $x_1 = (1,0)$, and it is rotated by an angle z, the new value will be $x_2 = (\cos z, \sin z)$. The value of cos z or sin z can is the real or imaginary value of x_2 after rotation. The CORDIC algorithm is based

on rotating a vector with a single angle through successive rotations of constant pre-calculated angles. As the number of iterations increases, we obtain a better accuracy. Figure 6.14 illustrates a sample CORDIC iteration.

Assume that a vector with coordinates (x_i, y_i) is to be rotated by an angle α_i to the new coordinates (x_{i+1}, y_{i+1}). The values of the new coordinates are

$$x_{i+1} = x_i \cos\alpha_i - y_i \sin\alpha_i$$

$$x_{i+1} = \frac{x_i - y_i \tan\alpha_i}{\sqrt{\left(1+\tan^2\alpha_i\right)}}$$

$$y_{i+1} = y_i \cos\alpha_i + x_i \sin\alpha_i$$

$$y_{i+1} = \frac{y_i + x_i \tan\alpha_i}{\sqrt{\left(1+\tan^2\alpha_i\right)}}$$

$$z_{i+1} = z_i - \alpha_i \tag{6.14}$$

After m iterations, we obtain $z_m = z_0 - \sum \alpha_i$

In order to simplify these calculations, the values of α_i are chosen to be $\tan^{-1}\left(2^{-i}\right)$ such that multiplications are converted to simple add and shift operations. The term $\sqrt{\left(1+\tan^2\alpha_i\right)}$ in the denominator can be omitted such that rotations are converted to pseudo-rotations, which are a scaled version from the conventional rotations as shown in Figure 6.14. After pseudo-rotations, we obtain the new coordinates as $\left(x'_{i+1}, y'_{i+1}\right)$. The general form of CORDIC rotation is obtained as follows

$$x'_{i+1} = x'_i - y'_i . d_i . 2^{-i}$$

$$y'_{i+1} = y'_i + x'_i . d_i . 2^{-i}$$

$$z_{i+1} = z_i - d_i . \tan^{-1} 2^{-i} \tag{6.15}$$

where $d_i \in \{-1, 1\}$

The values of angles $\tan^{-1}2^{-i}$ are pre-calculated and stored in a LUT. The structure of the basic CORDIC hardware consists of two adders and a LUT as shown in Figure 6.15.

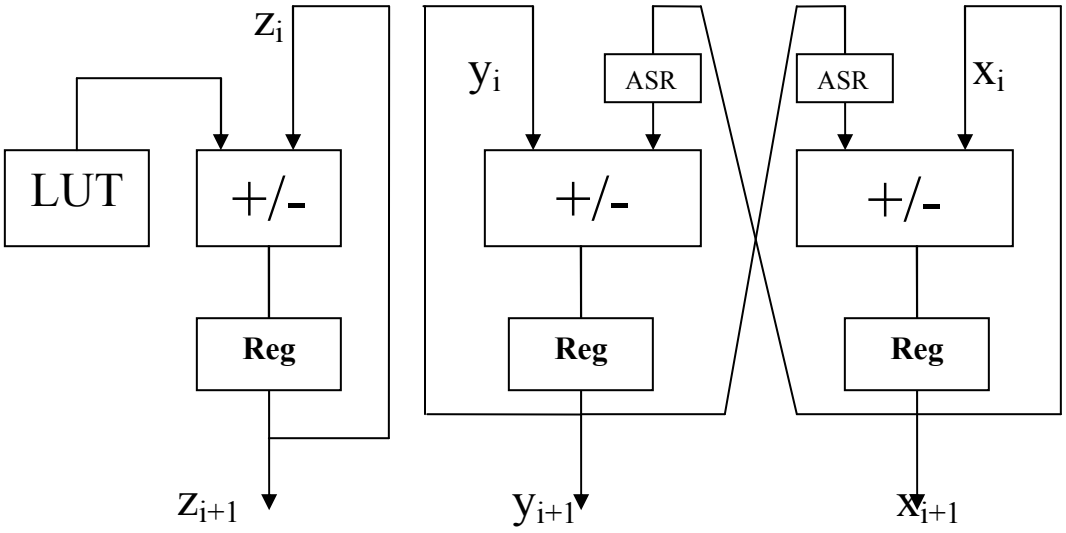


Figure 6.15 Basic CORDIC Hardware

The approximate values of angles $\tan^{-1}2^{-i}$ which should be stored in a LUT are given in Table 6-1 as shown

Table 6-1 Approximate values of $\tan^{-1}2^{-i}$

i	1	2	3	4	5	6	7	8	9	10
$\tan^{-1}2^{-i}$ (degrees)	45^0	26.6^0	14^0	7.1^0	3.6^0	1.8^0	0.9^0	0.4^0	0.2^0	0.1^0
$\tan^{-1}2^{-i}$ (radian)	0.785	0.464	0.245	0.124	0.062	0.031	0.016	0.008	0.004	0.002

To perform a set of rotation, the values of d_i are selected for each rotation such that the final angle equals $\sum \alpha_i$.

A multiplication by the constant

$$K = \prod \frac{1}{\sqrt{1+\tan^2 \alpha_i}} = \prod \frac{1}{\sqrt{1+2^{-2i}}}$$

$K = 1.646760258121$ normalizes the final values and converts the rotations into conventional rotations.

This constant is used in calculation of final value of x_m, y_m such that:

$$x_m = \frac{1}{K} x_m'$$

$$y_m = \frac{1}{K} y_m'$$

The CORDIC algorithm operates in two modes; Rotation and Vectoring. They are both based on the aforementioned procedure, except that they differ in the mechanism of rotation. In case of rotation mode, the vector is rotated according to a target rotation angle so that the factor d_i is determined according to the sign of the angle z_i at iteration i. It is suitable when we need to rotate a vector with a certain angle. It is suitable for final subcarrier derotation.

In case of vectoring mode, the target is to rotate the vector with a certain angle α such that its imaginary part approaches zero. The decision of rotation is based on the sign of real and imaginary part of rotated vector after each iteration. It is suitable for pilot phase estimation.

The entity block of CORDIC unit is shown in Figure 6.16. This version of CORDIC algorithm is a digit recurrence technique; this means that it calculates one correct bit per iteration. It is also defined as Radix-2 CORDIC. Its advantage is the design simplicity, but latency increase with the increase of number of iterations. In order to preserve higher throughput, pipelined architectures can be used, or either high radix CORDIC such as Radix-4 CORDIC may be used. The

Radix-4 CORDIC is a faster version of conventional Radix-2 CORDIC algorithm. It combines two iterations of conventional CORDIC into one iteration, so it generates two correct bits per iteration. The total number of iterations is halved. The cost of improving the speed is the larger hardware complexity. In Radix-4 CORDIC, there are 4 sets of rotation angles for each iteration. Moreover, the constant *K* is not simply determined as in the case of Radix-2 CORDIC.

In our implementation, the conventional Radix-2 CORDIC is more convenient to be used. The input received signals are quantized in 8-bit precision. Hence, 8 rotations are required for Radix-2 CORDIC. The synthesis results discussed in section 6.6 indicate that the mobile WiMAX timing requirements are still satisfied with the usage of one CORDIC unit for complete 8 iterations.

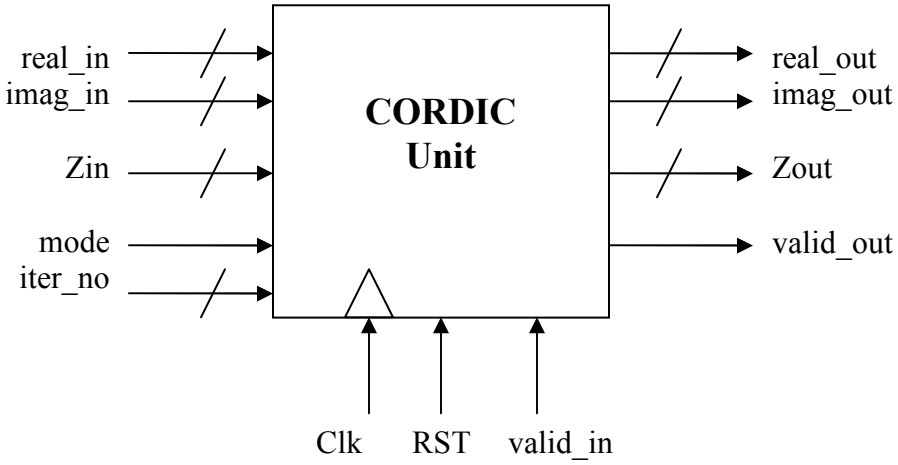


Figure 6.16 CORDIC Unit entity

As shown in Figure 6.16, the CORDIC unit has the quantized real and imaginary part of input vector that are represented in *real_in* and *imag_in*. Each is represented in an 8-bit precision. The *mode* input controls the mode of operation to be in rotation or vectoring mode. The *iter_no* input controls the shift operation inside the CORDIC unit. The Z_{in} input represents the desired rotation angle in

rotation mode. The rotated vector is represented in *real_out* and *imag_out*. The *valid_out* signal is activated as soon as output is ready.

6.5.2.2 Pilot rotation using CORDIC

In this section we apply the CORDIC algorithm in estimation of pilot rotation. The original transmitted pilot subcarriers are known to have no imaginary part. In order to estimate the phase rotation of a rotated pilot subcarrier, successive CORDIC rotations are carried. The vectoring mode is used such that each rotation targets to moves the imaginary part towards zero as shown in Figure 6.17. In this case, the value of d_i is determined such that y_{i+1} approaches zero value. Table 6-2 illustrates determination of rotation factor d_i for each iteration.

Table 6-2 Determination of CORDIC rotation factor d_i

x_i	y_i	d_i
Positive	Positive	-1
Positive	Negative	+1
Negative	Positive	+1
Negative	Negative	-1

The rotation factor d_i is calculated by a simple XOR logic function between the sign bits of real part x_i and imaginary part y_i. After the last iteration, the sequence of d_i*'s* can be used as an address to memory which stores the corresponding phase or it can be determined through recursive additions.

In our implementations, eight iterations are used for CORDIC with a pipelined architecture to achieve higher throughput. Received data subcarriers should be stored in a FIFO block until they are used by the subsequent block after

estimation of data subcarrier phases. Moreover, pilot estimated phases should be stored in a RAM module until they are used in phase line coefficients estimation.

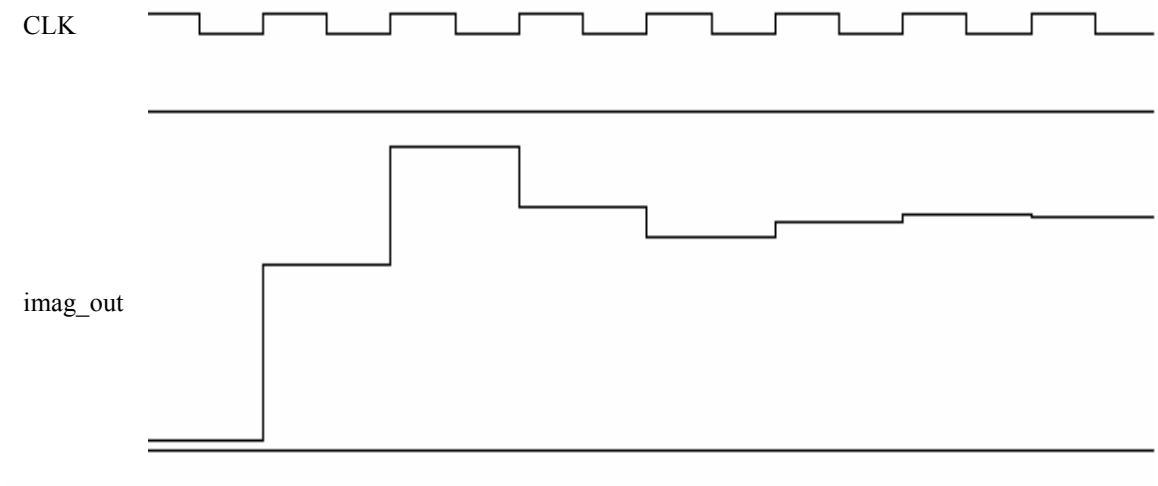


Figure 6.17 Convergence of imaginary part in vectoring mode

6.5.3 Phase Coefficient Computation block

The main purpose of this block is to calculate the slope and bias of the phase line equation, based on estimated pilot subcarriers phases. Figure 6.18 illustrates the entity of this block. It has inputs pilot_angle that represents the estimated angle of pilot subcarrier, and outputs a_coef, b_coef that represent phase error line slope and bias respectively.

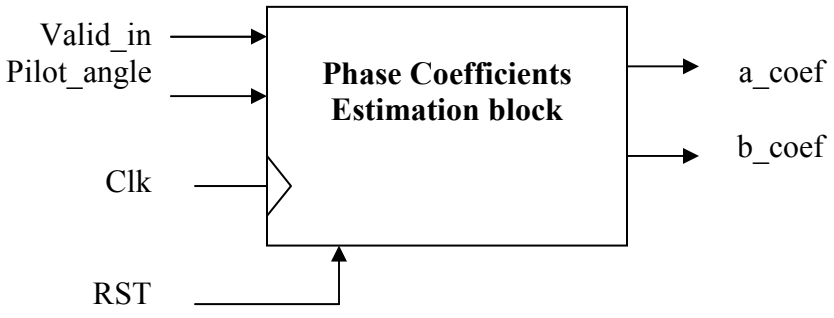


Figure 6.18 Phase Coefficients entity

In the proposed implementation, LS algorithm is used to estimate phase coefficients. In this thesis, we study the case of FFT size =1024, and FUSC permutation scheme. In this case 82 pilot subcarriers are defined. The locations of pilot subcarriers are divided into constant sets and variable sets. Constant sets are fixed locations for all OFDM symbols, while variable sets vary depending on whether it is an even or odd OFDM symbol. There are two constant sets and two variable sets, defined as indicated in Table 6-3 with respect to subcarrier index

Table 6-3 Pilot locations for FUSC permutation with 1024 FFT size

	Pilot locations
Constant set 0	[-415 -271 -127 17 161 305]
Constant set 1	[-343 -199 -55 89 233]
Variable set 0	[-424 -400 -376 -352 -328 -304 -280 -256 -232 -208 -184 -160 -136 -112 -88 -64 -40 -16 8 32 56 80 104 128 152 176 200 224 248 272 296 320 344 368 392 416]
Variable set 1	[-388 -316 -244 -172 -100 -28 44 116 188 260 332 404 -412 -340 -268 -196 -124 -52 20 92 164 236 308 380 -364 -292 -220 -148 -76 -4 68 140 212 284 356]

For even OFDM symbol, pilot locations consist of constant sets and variable sets. However, for odd OFDM symbols, they consist of constant sets, and variable sets + 6.

In this case, we construct the LS matrix as follows:

For even symbols

$$A = \begin{bmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} = \begin{bmatrix} 82 & -889 \\ -889 & 4898731 \end{bmatrix} \tag{6.16}$$

For odd symbols

$$A = \begin{bmatrix} n & \sum x_i \\ \sum x_i & \sum x_i^2 \end{bmatrix} = \begin{bmatrix} 82 & -463 \\ -463 & 4897879 \end{bmatrix} \tag{6.17}$$

where n represents the number of pilot subcarriers

x_i represents the set of locations of the pilot subcarriers

The phase line coefficients can be estimated as follows

$$\begin{bmatrix} b \\ a \end{bmatrix} = A^{-1}.\begin{bmatrix} \sum y_i \\ \sum x_i y_i \end{bmatrix}$$

$$= \begin{bmatrix} 0.0122 & 2.2175\times10^{-6} \\ 2.2175\times10^{-6} & 2.0454\times10^{-7} \end{bmatrix}\begin{bmatrix} \sum y_i \\ \sum x_i y_i \end{bmatrix} \text{ for even symbols,}$$

$$= \begin{bmatrix} 0.0122 & 1.1534\times10^{-6} \\ 1.1534\times10^{-6} & 2.0428\times10^{-7} \end{bmatrix}\begin{bmatrix} \sum y_i \\ \sum x_i y_i \end{bmatrix} \text{ for odd symbols,}$$

Implementation of this block implies two main units, Accumulator (ACC) to calculate $\sum y_i$ and Multiply/Add and Accumulate (MAC) unit as shown in Figure 6.19.

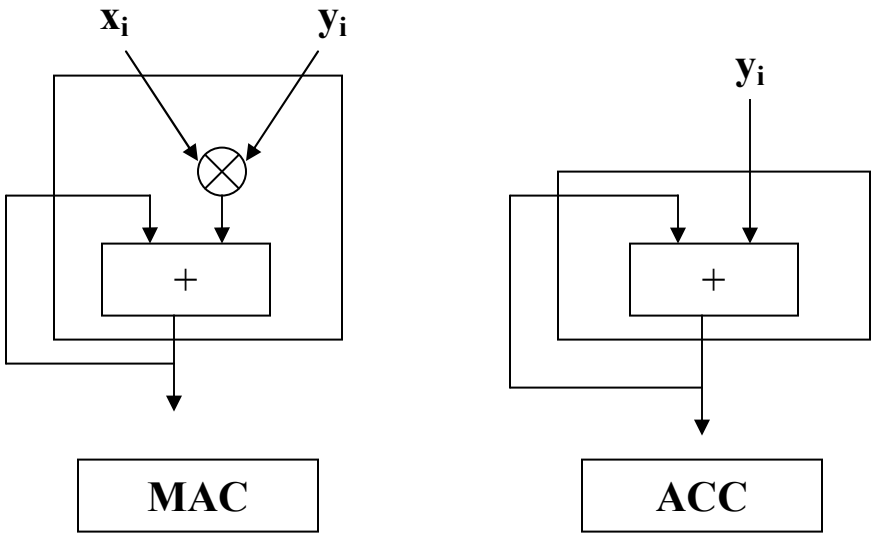


Figure 6.19 ACC and MAC units

The next step is to calculate *a*, *b* coefficients through matrix multiplications. Having a constant matrix simplifies multiplication operations. However, in this thesis, we propose a further approximation that simplifies the implementation of this unit. This approximation removes the matrix multiplication and has an insignificant loss in estimation performance. This approximation simply implies the calculation of *a, b* coefficients as follows

$$a \approx \frac{1}{2^{22}} \sum x_i y_i$$

$$b \approx \frac{1}{n} \sum y_i = \frac{1}{82} \sum y_i \approx \frac{1}{2^7} \sum y_i + \frac{1}{2^8} \sum y_i \tag{6.18}$$

The key idea of this approximation is that the coefficient multiplied by $\sum x_i y_i$ in calculation of *b* coefficient has a very small weight. On the other hand, in the calculation of *a* coefficient, although the weights of the two factors are close, the aggregate weight of $1.1534 \times 10^{-6} \sum y_i$ is much smaller than $2.0428 \times 10^{-7} \sum x_i y_i$, and the coefficient of $\sum x_i y_i$ can be approximated to $\frac{1}{2^{22}}$.

This approximation removes the excess hardware needed for a, b calculation. Calculation of the slope *a* is carried out via a simple shift operation, and the calculation of the bias *b* is carried out via constant multiplication which can be performed with only one addition operation. This approximation has a small effect on degradation of the system performance. The comparison between BER performance in case of matrix multiplication and approximation is illustrated in Figure 6.20.

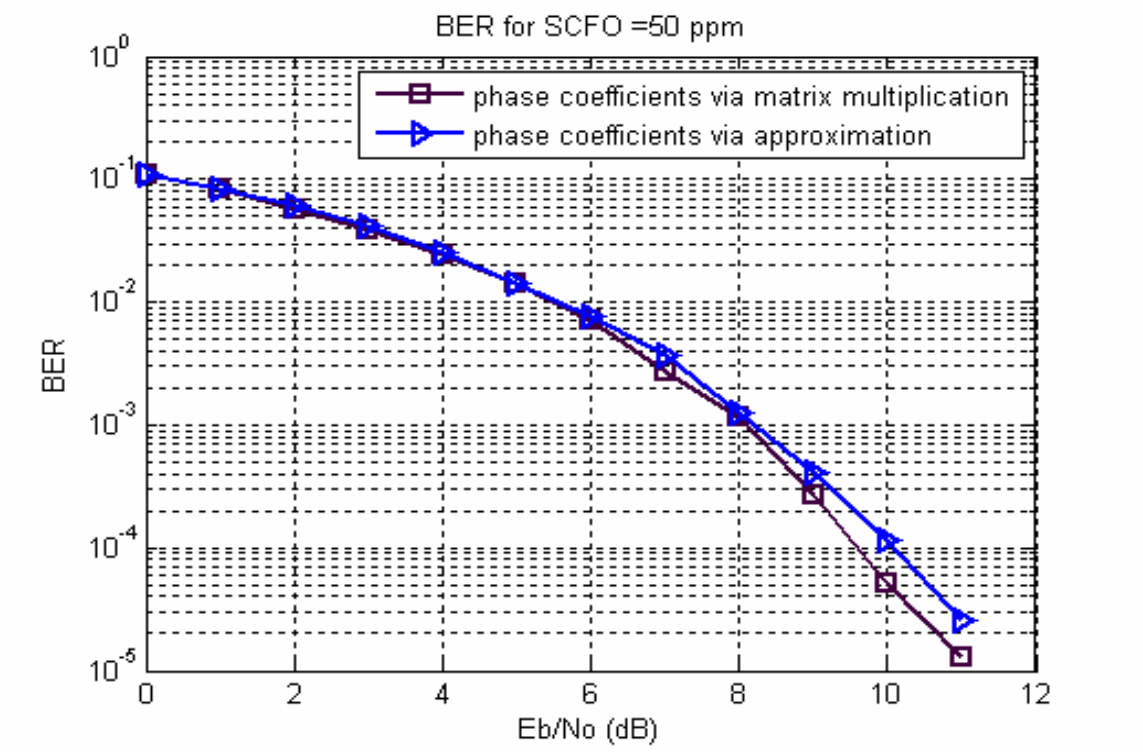


Figure 6.20 Comparison of the perfect and approximated phase coefficients

The above approximation indicates that the output of the MAC unit should be shifted to the right. A large number of least significant bits should be truncated. In this thesis, a proposed implementation of the MAC unit is carried out which leads to a significant reduction in the area of the MAC unit. The proposed implementation of the MAC unit is not constructed from a multiplier followed by an adder, but a common used implementation is to perform the multiplication and addition together as one operation in one Partial Product Array (PPA) [45]. This is carried out by inserting the last operand to be added as an extra partial product inside the PPA. When we insert the additional operand inside the PPA, we obtain it as shown in Figure 6.22. The next operation is to reduce the whole PPA using any Partial Products reduction techniques. In this implementation, we use the PPA proposed in [46] for signed multiplication. In our case, we need 10 x 10 signed multiplier. This PPA is illustrated in Figure 6.21. If we have $X=X_9\ldots.. X_1X_0$, $Y=Y_9\ldots..Y_1Y_0$, then $P_{ij}=X_iY_j$

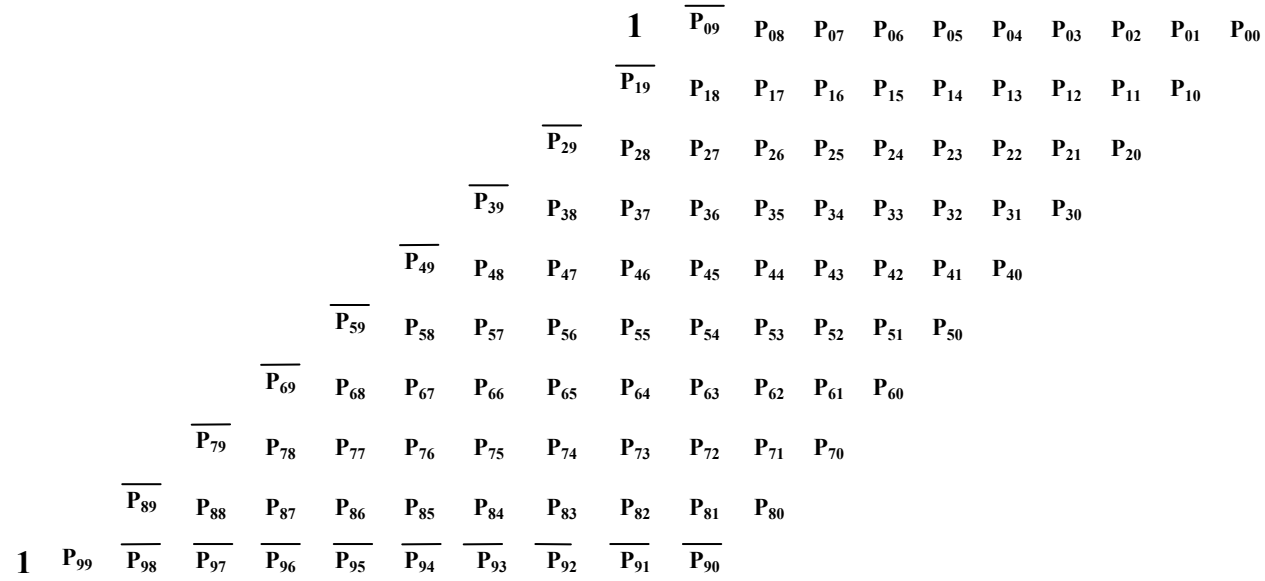


Figure 6.21 PPA for 10 x 10 signed multiplier

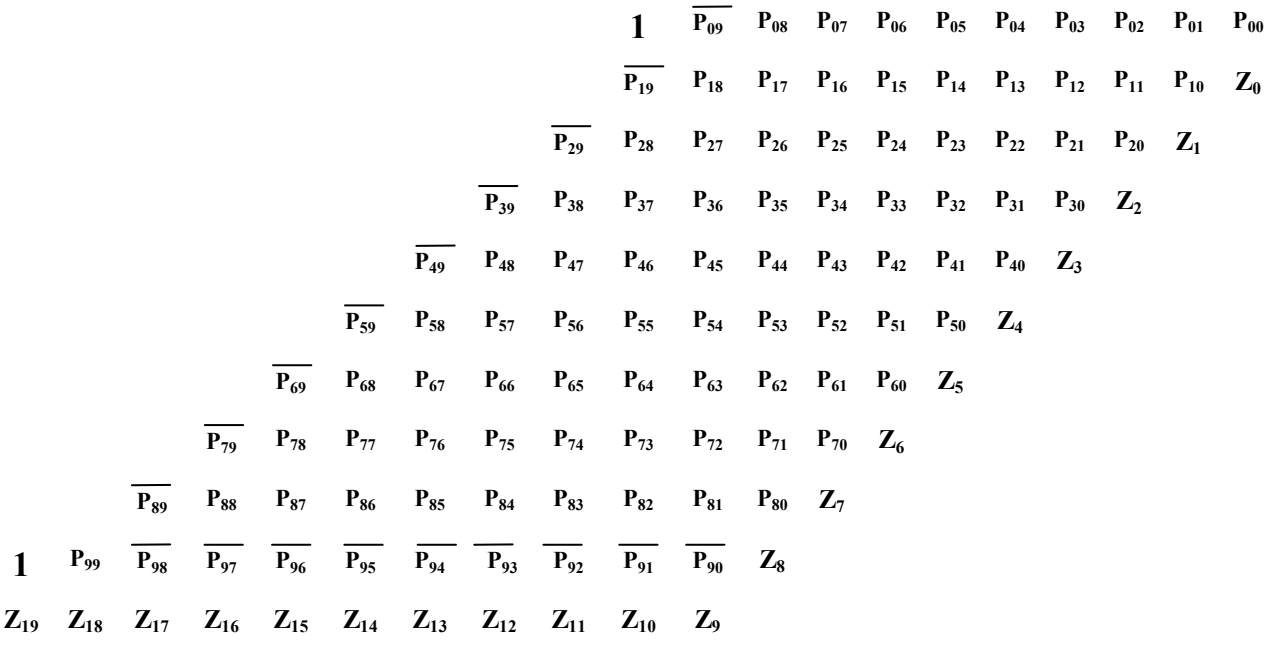


Figure 6.22 MAC operation in one PPA

Additionally, we propose another improvement that reduces area and delay in a significant way. This is achieved through applying a truncation to a part of the PPA instead of constructing the entire PPA. As we need to consider only a few of most significant bits of the result. The fixed point analysis indicates that we need only to consider the 5 most significant bits of the result. In our analysis, we can determine the number of least significant bit positions to truncate as follows

$$2^{N_1} > 82\sum_{j=1}^{N_2}\sum_{i=1}^{j} 2^{(N_2-i)}$$

Where N_1 represents the bit position from which we consider the final output, N_2 represents the most significant bit position of the truncated part of the PPA. The number of bits that are truncated should have insignificant effect on the final result. This means that their effect is considered as one carry input to the least significant bit of the considered part of the final result. We have 82 accumulations. In order to determine the number of truncated bits N_2, consider one multiplication operation. We find that the largest value of the truncated part of the PPA is $\frac{2^{N_2}}{2}+\left(\frac{2^{N_2}}{2}+\frac{2^{N_2}}{4}\right)+\left(\frac{2^{N_2}}{2}+\frac{2^{N_2}}{4}+.....+1\right)$, we need this to be smaller than 2^{N_1}. This summation is multiplied by 82 as we have 82 accumulations corresponding to the number of pilots. Our analysis shows that for N_1=20, we get N_2=10. This leads to a truncation of about half of the entire PPA, which in sequence leads to a saving of approximately half of the original area. The resulting truncated PPA is shown in Figure 6.23.

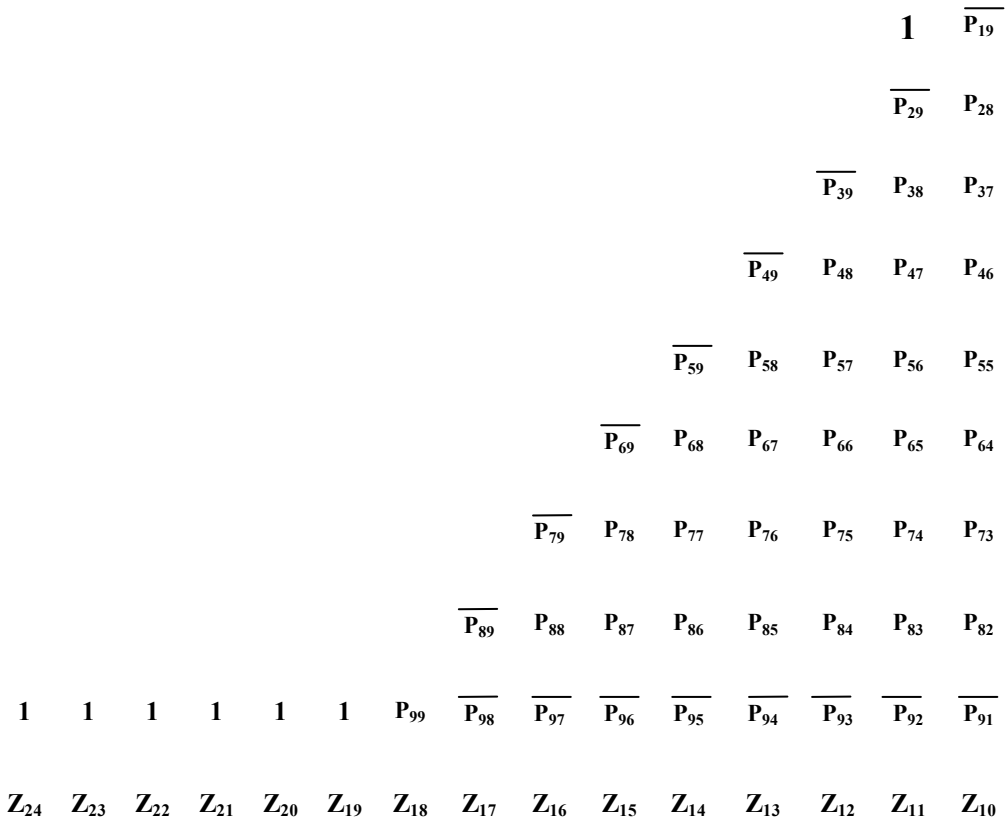


Figure 6.23 Proposed truncated MAC PPA

6.5.4 Data subcarriers Phase estimation block

The next step after calculation of phase line coefficients is to calculate the phase of data subcarriers. This is carried out by a simple MAC unit. Subcarrier index is generated via a 10 bit subcarrier index counter. The output of this counter is multiplied by the coefficient *a* then added to the estimated bias *b*. This operation is demonstrated in Figure 6.24.

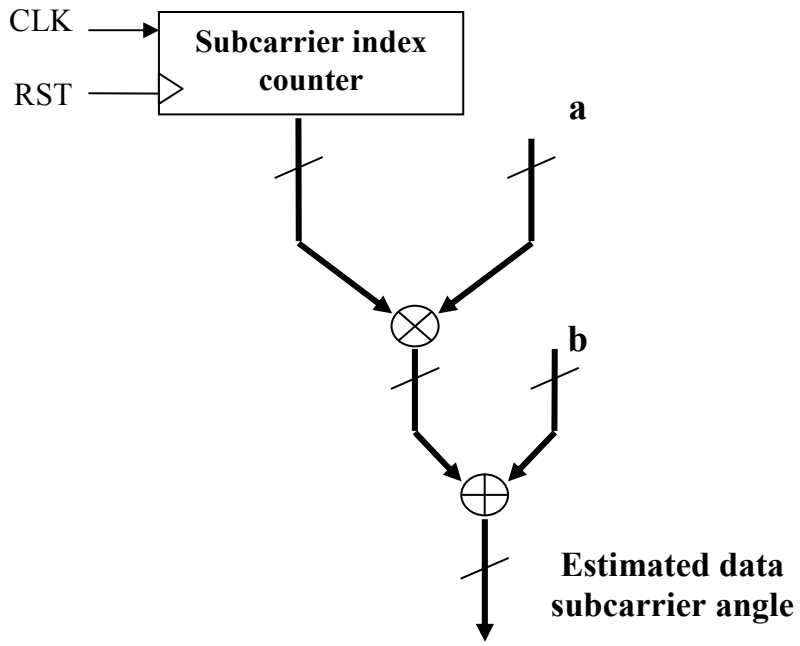


Figure 6.24 Phase estimation hardware

6.5.5 Subcarrier de-rotation via CORDIC

The last step after estimating the phase of each subcarrier is to perform de-rotation in order to correct the effect of both RCFO and SCFO. The de-rotation operation is simply implemented via CORDIC algorithm. The CORDIC unit used for subcarrier de-rotation is similar to the one used in pilot phase estimation, but it operates in rotation mode instead. In case of pilot phase estimation, the rotation phase is not known and d_i is selected such that the imaginary part approaches zero value. However in the de-rotation case, we need to satisfy that phase reaches zero value after m iterations such that:

$$z_m = z_0 - \sum d_i . \tan^{-1} 2^{-i} \tag{6.19}$$

The value of z_m represents the difference between desired angle and rotated angle. The decision on value of d_i is performed such that the angle z_{i+1} approaches zero.

6.6 Synthesis Results

The implemented blocks of sampling clock and frequency tracking unit are synthesized on Altera StratixII FPGA platform using Altera Quartus tools, targeting optimization for speed. The target device is EP2S15F484C3. We obtain the following synthesis results

Table 6-4 Synthesis results for Sampling clock and Frequency tracking

Component	Number of LUTs	Number of Registers	Number of memory bits	Maximum frequency of operation
Pilot Phase detection	101	106	___	152 MHZ
Phase Coefficients estimation block	178	142	___	327.23 MHZ
Data subcarriers phase estimation	11 + 2 DSP block 9-bit elements	19	___	250.44 MHZ
Subcarrier derotation block	101	106	___	152 MHZ
The complete Timing / Frequency tracking block	347 + 2 DSP block 9-bit elements	300	17560	145.31 MHZ

From the synthesis results obtained in Table 6-4, we conclude that one Radix-2 CORDIC unit is suitable to be used in Pilot phase detection and pilot phase estimation. It can be used for successive 8 CORDIC iterations and satisfy the symbol timing requirements of IEEE 802.16e standard.

Chapter 7

Conclusion and Future work

In this thesis, we present the simulation model of optional CTC used in IEEE802.16e mobile WiMAX. It is found that it has a better performance over the mandatory convolutional coding schemes for higher number of iterations. We also present the hardware implementation of CTC encoder and decoder with efficient implementation techniques that target area reduction or speed enhancement over the existing conventional techniques. Our implementation targets the FPGA design platform. The implementation was held to satisfy the system requirements and throughput. We introduced a novel implementation of state metric unit normalization using the redundant number system. However, the new implementation is more suitable in custom design platforms rather than FPGA. The improvement in speed can be rather insignificant in FPGA compared to the increase in the area. This is due to that some other components affect the delay in FPGA such as routing delay and interconnect. However, this improvement is significant in case of custom design.

We also presented an efficient quantization of turbo decoder with the optimum number of bits compared to that in the literature. Moreover, we achieved high speed SISO architecture for the FPGA platform. Our SISO decoder operates at 150 MHZ. It is faster than other architectures that targets the FPGA platform and mentioned in the literature.

We also present the Timing and Frequency tracking simulations and hardware implementation using least square error linear curve fitting. It is found that LS algorithm is the best that minimizes the effect of the channel noise in correction of the sampling error and residual carrier frequency offset effect.

Additionally, it is simple in hardware implementation. We presented an optimized implementation for a common used special case of FUSC with FFT size 1024. Our implementation is suitable for the other permutation schemes. We can think about extending our implementation to be generic and handle different permutation schemes and FFT sizes.

The work is still open for future improvements. Our implementation of Turbo codes and sampling clock/Frequency tracking is suitable for hardware implementation on other platforms using standard cells and ASIC. The FPGA is used only for proto-typing. But in order to achieve a turbo decoder chip, our implementation should target the ASIC design. Other optional coding schemes may be studied and implemented such that LDPC codes. They also have a good performance that competes with that of CTC. Another issue is the study of channel estimation with timing and frequency tracking. We implement a simple algorithm that assumes perfect channel estimation, but for case of non-channel estimation, this algorithm fails to achieve its performance. We can search for a joint algorithm for channel estimation and clock / Frequency tracking.

REFERENCES

[1] Senza Fili "Fixed, nomadic, portable and mobile applications for 802.16-2004 and 802.16e WiMAX network" Consulting on behalf of the WIMAX Forum, November 2005.

[2] Henrik Schulze and Christian Luders "Theory and Applications of OFDM and CDMA Wideband Wireless Communications" ISBN, 2005

[3] Loutfi Nuaymi "WiMAX technology for broadband wireless access" ENST Bretagne, France, Wiley, 2007

[4] Jeffrey G. Andrews, Arunabha Ghosh, Rias Muhamed "Fundamentals of WiMAX" Prentice Hall, 2007

[5] Syed Ahson, Mohammad Ilyas "WiMAX standards and security" CRC press, September 2005.

[6] "Air Interface for Fixed Broadband Wireless Access Systems" IEEE P802.16-REVd/D5, May 2004.

[7] "Air Interface for Fixed Broadband Wireless Access Systems" IEEE Std 802.16e-2005

[8] Simon Haykin "Communication Systems" 4th edition, Bill Zobrist, McMaster University, 2001, pp 626-695

[9] Yushi Shen and Ed Martinez **"**Channel Estimation in OFDM Systems", Freescale Semiconductor, Inc., 2006.

[10] S. Colieri; M. Ergen; A. Puri; A. Bahai "A study of channel estimation in OFDM systems**"**, IEEE 56th proceeding on Vehicular Technology, 2002, Vol.2, pp.894-898

[11] John Terry, Juha Heiskala "OFDM Wireless LANs, A theoretical and practical Guide" Sams Publishing, 2001

[12] G.D.Forney "The Viterbi Algorithm" IEEE proceeding, Mar.1973, vol.61, pp. 268-278.

[13] Abd-Elmohsen Khater "Simulation and Implementation of the timing synchronization, cell identification, and FFT for the OFDMA-based mobile WiMAX 802.16e", MSc. Thesis, Faculty of Engineering, Cairo University, 2007

[14] M. Ismail Ali "Simulation and implementation of the Frequency Synchronization and Viterbi decoding for the OFDMA-based mobile WiMAX 802.16e", MSc. Thesis, Faculty of Engineering, Cairo University, 2007

[15] Berrou, C., Glavieux, A., and Thitimajshima, P., "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes," IEEE Proceedings of the Int. Conf. on Communications, Geneva, Switzerland, May 1993 (ICC'93), Vol. 2, pp. 1064-1070.

[16] Berrou, C. and Glavieux, A. "Near Optimum Correcting Coding and Decoding: Turbo Codes" IEEE Transactions on Communications, October 1996, Vol. 44, pp.1261-1271

[17] C.Douillard, M. Jézéquel, C. Berrou, N. Brengarth, J. Tousch and N. Pham "The Turbo Code Standard for DVB-RCS"

[18] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," IEEE Trans. Inf. Theory, Mar. 1974, vol. IT-20, no. 2, pp. 284–287.

[19] L.Hanzo, T.H.Liew, B.L.Yeap "Turbo Coding, Turbo equalization and Space time Coding for transmission over fading channels" Wiley, 2002

[20] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in Proc. IEEE Int. Conf.Communications, Jun. 1995, pp. 1009–1013.

[21] Jörg Vogt and Adolf Finger "Improving the Max-Log-MAP Turbo Decode" Electronics Letters Vol 36, Issue 23, 9 Nov 2000 pp. 1937 - 1939

[22] M. Marandian, J. Fridman, Z. Zvonar, and M. Salehi1 **"**Performance Analysis of Sliding Window Turbo Decoding Algorithms for 3GPP FDD Mode", International Journal of Wireless Information Networks, 2002, Vol. 9, pp. 39-54

[23] Matthew C. Valenti, Shi Cheng, and Rohit Iyer Seshadri "Turbo and LDPC Codes for Digital Video Broadcasting" Turbo codes Applications book, pp. 301-319

[24] Roshni Srinivasan, Jeff Zhuang, Louay JalloulRobert Novak, Jeongho Park "Draft IEEE 802.16m Evaluation Methodology Document" IEEE contributions, IEEE C802.16m-07/080r3, August 2007

[25] Marco A. Castellon, Ivan J. Fair, Duncan G. Elliott "Fixed-Point Turbo Decoder Implementation suitable for Embedded Applications" IEEE, Canadian Conference on Electrical and Computer Engineering CCECE/CCGEI, Saskatoon, May 2005, pp.1065-1068

[26] Heiko Michel and Norbert Wehn "Turbo-Decoder Quantization for UMTS" Communications Letters, IEEE Volume 5, Issue 2, Feb 2001 pp.:55 – 57

[27] Zhongfeng Wang, Hiroshi Suzukit and Keshab K. Parhi "VLSI Implementation issues of Turbo decoder design for wireless applications" IEEE Workshop on Signal Processing Systems, 1999 pp.:503 – 512

[28] S. S. Pietrobon, "Implementation and performance of a TURBO/MAP decorder", International Journal of Satellite Communications and Networking, 1998, Vol. 16, pp. 23-46

[29] Zhongfeng Wang, Hiroshi Suzukit and Keshab K. Parhi "VLSI Implementation issues of Turbo decoder design for wireless applications" IEEE Workshop on Signal Processing Systems, 1999 pp.:503 – 512

[30] S. S. Pietrobon, "Implementation and performance of a TURBO/MAP decorder", International Journal of Satellite Communications, 1998

[31] J.H.Han, A.T.Erdogan, T.Arslan "High speed Max-Log-Map Turbo SISO decoder Implementation using Branch Metric Normalization" Proceedings of IEEE Computer Society Annual Symposium on VLSI, 2005.

[32] B. Shim and H.G. Myung **"**A novel metric representation for low-complexity log-MAP decoder" IEEE International Symposium on Circuits and Systems, 2005, Vol.6 , pp.5830-5833

[33] A.P Hekstra "An alternative to metric rescaling in Viterbi decoders" IEEE transactions on Communications, 1989, Vol.37, pp 1220-1222

[34] B. Parhami "Computer Arithmetic, Algorithms and Hardware Design", Electrical and Computer Engineering department. University of California, Santa Barbara, Oxford University Press 2000

[35] C. Anghel, A. A. Enescu, O. M. Bugiugan, C. R. Cacoveanu "FPGA implementation of a CTC Decoder for H-ARQ complaint WiMAX systems" International Conference on Design and technology of integrated systems in Nanoscale Era, September 2007, pp.82-86

[36] Ji-Hoon Kim, In-Cheol Park "Double binary Circular Turbo decoding based on Border metric Encoding " IEEE transactions on Circuits and Systems, Jan 2008, Vol.55, pp. 79-83

[37] Yong Wang1, Ge Jian-hua, Bo Ai, Li Zong-qiang, and Nie Yuan-fei "A Novel Scheme for Symbol Timing in OFDM WLAN Systems"

[38] Haiyun Tang, Kam Y. Lau, and Robert W. Brodersen. "Synchronization Schemes for Packet OFDM System"

[39] Baoguo YANG, B. Letaief, Roger S. Cheng, and Zhigang "An Improved Combined Symbol and Sampling Clock Synchronization Method for OFDM Systems",

[40] Wang Dan , Hu Ai qun "A Combined Residual Frequency and Sampling Clock Offset Estimation for OFDM Systems" IEEE Asia Pacific Conference on Circuits and Systems APCCAS, 2006, pp.1184-1187

[41] Jen-Ming Wu and Chun-Hung Chou "Baseband Sampling Clock Frequency Synchronization for WiMAX Systems". <*http://mail.com.nthu.edu.tw/~jmwu/record/SFO_paper_v5.pdf*>, 2006

[42] W. D. Warner and C. Leung, "OFDM/FM frame synchronization for mobile radio data communication," IEEE Trans. Veh. Technol., Aug. 1993, vol. 42, pp. 302-313.

[43] Jan-Jaap van de Beek, Magnus Sandell and Per Ola B.rjesson "ML Estimation of Time and Frequency Offset in OFDM Systems" In IEEE Transactions on Signal Processing, July 1997, vol. 45, no. 7, pp. 1800-1805

[44] M. R. Dacca, G. Levin, and D. Wulich "Frequency offset tracking in OFDM based on Multi-carrier PLL" MILCOM 2000 Page(s):912 - 916 vol.2

[45] Israel Koren "Computer Arithmetic Algorithms" 2nd edition, University of Massachusetts, Amherst, A.K.Peters Ltd, 2002

[46] C.R. BAUGH and B.A. WOOLEY, "A two's complement parallel array multiplication algorithm" IEEE transactions on Computers, Dec. 1973 ,Vol. C-22 ,pp. 1045-1047