

# **Generator Polynomial in Turbo Code System**

**LOOI CHUN HUI**

**A project report submitted in partial fulfilment of the  
requirements for the award of Bachelor of Engineering  
(Hons.) Electronic and Communication Engineering**

**Faculty of Engineering and Science  
Universiti Tunku Abdul Rahman**

**April 2012**

## DECLARATION

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature : \_\_\_\_\_

Name : \_\_\_\_\_

ID No. : \_\_\_\_\_

Date : \_\_\_\_\_

### APPROVAL FOR SUBMISSION

I certify that this project report entitled “**GENERATOR POLYNOMIAL IN TURBO CODE SYSTEM**” was prepared by **LOOI CHUN HUI** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Hons.) Electronic and Communication Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature : \_\_\_\_\_

Supervisor: Dr. Balamuralithara a/l Balakrishnan

Date : \_\_\_\_\_

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of University Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© Year, Name of candidate. All right reserved.

## GENERATOR POLYNOMIAL IN TURBO CODE SYSTEM

### ABSTRACT

In digital wireless system, error control codes have become an important technique for enabling reliable transmission to be achieved over noise and fading channels. Since the introduction of the turbo codes, many researchers have put effort in improving the turbo code performance because over the past decade turbo code have been widely considered as the most powerful error control code. Improving the turbo code performance with choosing the optimum generator polynomial without increasing the complexity in the system and delay. In this project, the researcher is searching for the optimum performance of the generator polynomial under AWGN channel. The study are investigate the performance of turbo code system using difference decoding algorithm and frame size under AWGN channel. The performance results for the turbo code system through the matlab simulation, where also consider the system was investigated using Log-MAP decoder over AWGN channel with different frame sizes. The researcher found that generator polynomial,  $g(D) = [7,5]$ ,  $[13,15]$  and  $[31,17]$  gives the best performance for constraint length,  $K = 3, 4$  and  $5$  respectively. The performance results for the turbo code system are obtained through using matlab simulation. The study are investigate the performance of turbo code system using difference decoding algorithm and frame size under AWGN channel.

## TABLE OF CONTENTS

<b>DECLARATION</b>	<b>ii</b>
<b>APPROVAL FOR SUBMISSION</b>	<b>iii</b>
<b>ABSTRACT</b>	<b>v</b>
<b>TABLE OF CONTENTS</b>	<b>vi</b>
<b>LIST OF TABLES</b>	<b>ix</b>
<b>LIST OF FIGURES</b>	<b>x</b>
<b>LIST OF SYMBOLS / ABBREVIATIONS</b>	<b>xii</b>
<b>LIST OF APPENDICES</b>	<b>xiii</b>

## CHAPTER

<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
1.1	Background	1
1.2	Aims and Objectives	3
1.3	Thesis Outline	4
1.3.1	Chapter Two: Literature Review	4
1.3.2	Chapter Three: Methodology	4
1.3.3	Chapter Four: Discussion	4
1.3.4	Chapter Five: Conclusion	4
<b>2</b>	<b>LITERATURE REVIEW</b>	<b>5</b>
2.1	Communication System	5
2.2	Error Control Code	6
2.2.1	Linear Block Code	6
2.2.2	Hamming Code	8

	2.2.3	Cyclic Codes	9
	2.2.4	BCH Codes	9
	2.2.5	Reed-Soloman code	10
2.3		Convolutional Codes	10
2.4		Turbo Codes	14
	2.4.1	Turbo Codes Structure	14
	2.4.2	Puncturing	16
	2.4.3	Interleaving	16
	2.4.4	Turbo Decoding	17
	2.4.5	Turbo Decoding MAP Algorithm Overview	18
<b>3</b>		<b>METHODOLOGY</b>	<b>19</b>
	3.1	Introduction	19
	3.2	Simulation Setup	19
	3.3	Interpretation of the Simulation Model	21
	3.3.1	Input Data	21
	3.3.2	Decimal-to-binary	22
	3.3.3	Turbo encoder	22
	3.3.4	Channel	23
	3.3.5	Turbo Decoder	23
	3.3.6	Binary-to-decimal	24
	3.4	Step procedure in Matlab	24
	3.5	Turbo Code Distance Spectrum Calculator	25
<b>4</b>		<b>DISCUSSION</b>	<b>28</b>
	4.1	Introduction	28
	4.2	Effects of number of decoding iterations	28
	4.3	Frame size effects	29
	4.4	Effects of generator polynomial	30
	4.4.1	Effects of Generator Polynomial	32
	4.4.2	Effects of decoder algorithm	38
<b>5</b>		<b>CONCLUSION</b>	<b>39</b>

5.1	Conclusion	39
5.2	Future Work	40
<b>REFERENCES</b>		<b>41</b>
<b>APPENDICES</b>		<b>43</b>



## LIST OF TABLES

<b>TABLE</b>	<b>TITLE</b>	<b>PAGE</b>
2.1	Terminology of Convolutional Codes	11
2.2	Equivalent index position for original matrix index in diagonal interleaver (Rekh, rani, & A.Shanmugam, 2000)	17
4.1	Various combination of generator polynomials	31
4.2	Default parameters in simulation	32

## LIST OF FIGURES

FIGURE	TITLE	PAGE
2.1	Block diagram of communication system. (Korhonrn, 2003)	6
2.2	Linear block codes (Korhonrn, 2003)	8
2.3	Convolutional encoder with constraint length $K=3$ , rate $r=1/2$ (Berrou C. , 2010)	11
2.4	Convolutional encoder with constraint length $K=4$ , rate $r=1/2$ (Berrou C. , 2010)	12
2.5	Code Tree for the Convolutional Code (Berrou C. , 2010)	12
2.6	State diagram of the convolutional codes (Berrou C. , 2010)	13
2.7	Portion of Trellis diagram for convolutional encoder with constraint length $K=3$ , rate $r=1/2$ (Berrou C. , 2010)	13
2.8	Structure of a Turbo Encoder (Rekh, rani, & A.Shanmugam, 2000)	15
2.9	Structure of the RSC Encoder (Rekh, rani, & A.Shanmugam, 2000)	15
2.10	Structure of Turbo Decoding (Proakis, 1995)	17
3.1	Block Diagram of the simulation model	21
3.2	The encoder flow chart	22
3.3	The turbo decoder flow chart	23
3.4	User interface of the Turbo Code simulator	24

3.5	The matlab workspace with the result	25
3.6	Turbo Code Distance Spectrum Calculator	26
3.7	The screen shot of the notepad results obtained from simulation	27
4.1	Turbo code performance for frame length, $N=500$ for iteration 3, 4 and 5 with $g(D)=[7,5]_{10}$ , code rate $=1/3$ under AWGN channel	29
4.2	Comparison of frame length, $N=500$ and $1200$ for bit error rate versus $E_b/N_0$ with $g(D)=[7,5]$ , code rate $=1/3$ , number of iteration 7.	30
4.3	Turbo code performance for constraint length, $K=3$	32
4.4	The simulation results of number of codewords and weight distribution	33
4.5	Turbo code performance for constraint length, $K=4$	34
4.6	The simulation result of No. of codewords and weight distribution for constraint length $K=4$	35
4.7	Turbo code performance for constraint length, $K=5$	36
4.8	The simulation result of No. of codewords and weight distribution from “Turbo Code Distance Spectrum Calculator”	37
4.9	Effects of decoder algorithm	38

**LIST OF SYMBOLS / ABBREVIATIONS**

ARQ	Automatic repeat request
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
CDMA	Code Division Multiple Access
ECC	Error Control Coding
FEC	Forward Error Correction
GSM	Global System for Mobile Communication
LLR	Log- Likelihood Ratio
MAP	Maximum A Posteriori
ML	Maximum Likelihood
PCCC	Parallel Concatenated Convolutional Code
RSC	Recursive Systematic Convolutional
SNR	Signal to noise ratio
SOVA	Soft-Output Viterbi Algorithm
UMTS	Universal Mobile Telecommunication System
WCDMA	Wide Code Division Multiple Access

**LIST OF APPENDICES**

<b>APPENDIX</b>	<b>TITLE</b>	<b>PAGE</b>
A	TURBO CODE DEMO CODING	43

## **CHAPTER 1**

### **INTRODUCTION**

#### **1.1 Background**

Nowadays communication is become part of our life. World War II greatly promoted the development of mobile communications. Communication system is very helpful to connect people from one point to another one point and change people life style.

The first generation of mobile cellular telecommunications systems appeared in the 1980s. The first generation was not the beginning of mobile communications, as there were several mobile radio networks in existence before then, but they were not cellular systems either. The capacity of these early networks was much lower than that of cellular networks, and the support for mobility is weak.

The second-generation (2G) mobile cellular systems use digital radio transmission for traffic. Thus, the boundary line between first and second generation systems is obvious: it is the analog/digital split. The 2G networks have much higher capacity than the first generation systems. One frequency channel is simultaneously divided among several users. Hierarchical cell structures – in which the service area is covered by macocells, microcells, and picovells – enhance the system capacity even further.

Recently there has been an attempt in the GSM community to enhance GSM to meet the requirements of cordless markets. Cordless telephone system (CTS) us a scheme in which GSM mobiles can be used at home via a special home base station,

in a manner similar to the present-day cordless phones. This scheme can be seen as an attempt of the GSM phone vendors to get into the cordless market.

3G have been several competing proposals for a global 3G standard. Below, these are grouped based on their basic technology, WCDMA, advanced TDMA, hybrid CDMA/TDMA, and orthogonal frequency division multiplexing (OFDM).

WCDMA system is 5MHz or more, and this 5MHz is also the nominal bandwidth of all 3G WCDMA proposals. This bandwidth was chosen because it is enough to provide data rates of 144 and 384 kbps, and even 2Mbps in good conditions and bandwidth is always scarce, and the smallest possible allocation should be used. The bandwidth also can resolve more multipath than narrower bandwidths, thus improving performance.

Error control code is used for detecting and correcting the error in transmission. Each coding has different features and functions. The error control code can be classified as block code, convolution code, trellis coded modulation and turbo code.

Block codes accept information in successive blocks producing an overall encoded block. In other words, block codes are codes that a block includes group the source data. The most simple linear block code is repetition code.

Convolution codes do not break up the data stream into blocks, but add redundancy in a quasi-continuous manner. Basically, convolution code is with memory and the block code is without memory. This is because the convolution code needs some register in encoder or decoder to encode. Convolution code accepts message bits as a continuous sequence and thereby generates a continuous sequence of encoded bits at higher rate.

The first 2 codes' main problem is the reduction in spectral efficiency. Since nowadays need to transmit more bits, in a limited bandwidth, we can only transmit limited data. If we need to transmit more data, then the check bits will become larger, then the bandwidth also need requirement becomes larger. This problem can be avoided by using trellis coded modulation (TCM). In TCM, modulation and coding is treated as

combined entity. The key to this integrated modulation and coding approach is to devise an effective method for mapping the coded bits into signal point such as that the minimum Euclidean distance is maximized. So that, a simplistic approach to solve the spectral efficiency is to add parity check bits to data bit and modulation the symbols together. However, this modulation is not give good results. This is because this disallowing some symbol sequences in this enlarged signal space. The important aspect is the encoding and modulation is joint process so that allow shows higher resilience to noise that uncoded systems with the same spectral efficiency.

A very long codes can approach the Shannon limit. So that, the brute force decoding of such long codes need spent a lot of time and more complex. Turbo codes were the first used codes that came close to the Shannon limit using reasonable effort. The codes are interleaved and the vital trick lies in the decoder because the code need combination a lot short codes, the decoder can also be divide into several simple decoders that exchange information about the decoded bits and arrive at a solution. The random interleaver let the total codeword has very little structure and this come with some advantages such as increases the effective code length of the combined code and make decoding possible with an effort that is essentially determined by the length of the constituent codes.

## **1.2 Aims and Objectives**

Turbo codes is the one of error control codes apply in the current communication system. Researcher needed to know and try to improve the following things:

- Basic turbo codes encoder structures and to understand each parameters performance;
- To understand and research the generator polynomial functions to get the perfect result;
- To study the interleaver, puncturing and multiplexing performance and to change the parameters to get the optimum result in simulation.



### **1.3 Thesis Outline**

#### **1.3.1 Chapter Two: Literature Review**

Chapter two gives an overview explains how the turbo code system work in digital communication system in general. It starts with an introduction of the history of the communication system, followed by the explanation of the error control code include linear block code, convolutional code and so on. The turbo code system including the algorithm and structure of both turbo code encoder and turbo code decoder are described here. RSC encoder and the generator polynomial are explained in detail.

#### **1.3.2 Chapter Three: Methodology**

Chapter three explains the design and implementation of the source code in MATLAB. The simulation process in shown clearly and systematically. “Turbo code distance spectrum calculator” software is also briefly explained in this chapter.

#### **1.3.3 Chapter Four: Discussion**

Chapter four presents various simulations results of the turbo code. The simulation results are compared and analyzed here. Finally, discussion, conclusions and future works are pointed out in Chapter five.

#### **1.3.4 Chapter Five: Conclusion**

Chapter five point out the conclusions and future works.

## **CHAPTER 2**

### **LITERATURE REVIEW**

#### **2.1 Communication System**

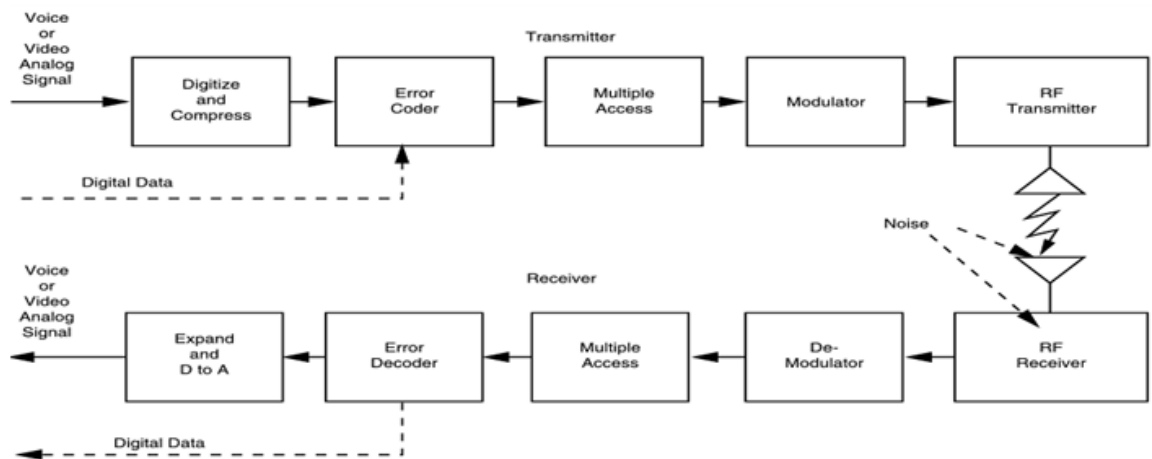
The remaining components in the overall system block diagram are digital processors to convert the analog voice and video signal into digital signals, to apply error correction bits, and to optimize the number of digital channels that can be transmitted within the RF system bandwidth.

The analog signals from the voice microphone and the video camera are shown entering the transmitter part of the communication system from the left-hand side of Figure 1. The signals are digitized, and the digital signal representing data cannot be compressed.

The digital signals representing voice, video, and data are then sent to an error coder, which adds extra bits to allow transmission errors to be corrected. The digital signals to be transmitted from an individual user occupy a small fraction of the service provider's licensed bandwidth. To accommodate many users the digital signals from each user are assigned a specific frequency slot and time slot, or a coding sequence. This is accomplished in the multiple access block of the overall system.

As the digital signal leaves the multiple access block, it is just a series of digital bits and system control signals. These bits are next modulated onto an IF subcarrier, which is at a frequency below the RF band where the modulation can easily be accomplished with a digital processor. The modulated IF subcarrier is then shifted to the RF frequency in the up converter block of the RF transmitter.

In the communication system receiver shown in the lower row of boxes in Figure 2.1, the demodulator, multiple access. Error decoder, expander, and DAC simply perform in the inverse functions of their counterparts in the transmitter. (Scott, 2008)



**Figure 2.1: Block diagram of communication system. (Korhonn, 2003)**

## 2.2 Error Control Code

### 2.2.1 Linear Block Code

Linear block code is the basic code in the error control coding (ECC). For a linear block code is transmit as a block by block and the message block is the fixed length. Each block got parity bits and data bits. Each bits is represented by 0 or 1. Linear block code is designed in term of generator and parity- check matrices.

Normally linear block encoder can distribute by 3 elements. The formula as following (Glavieux, 2007):

$$x = m \times G \quad (2.1)$$

where

$x$  = encoder message

$m$  = message

$G$  = generator polynomial matrix

The generator matrix  $G$  is used in the encoding operation at the transmitter, while the parity check matrix  $H$  is used in the decoding operation at the receiver.

If  $y$  represents the received vector, then

$$y = x + e \quad (2.2)$$

where

$x$  = encoder message

$e$  = error vector

$y$  = received vector

The  $e$  is represents the error vector. Here, we will get one definition is syndrome. The syndrome vector is  $S$ .

$$S = y \times H^T \quad (2.3)$$

where

$S$  = syndrome vector

$H^T$  = parity check matrix

$y$  = received vector

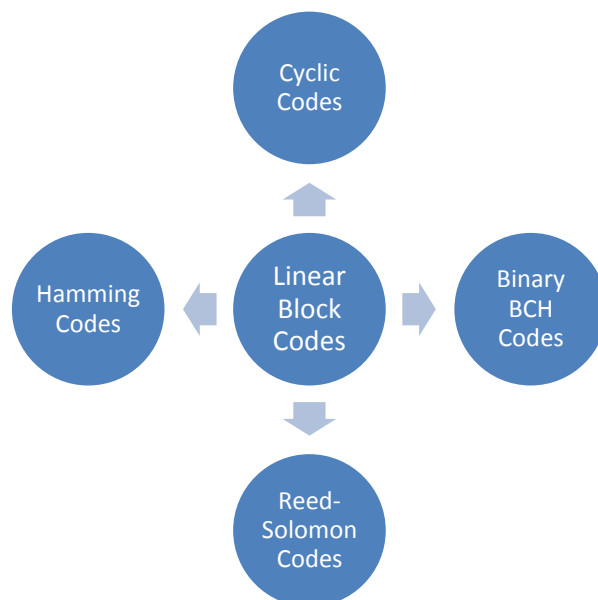
For a block code with  $2^k$  codewords and length  $n$ . The consists of  $k$  is represented information digits so that got  $2^k$  of codewords. A desirable structure for a

block code to possess is linearity because want to reduce the complexity of the encoding.

Here is some definition of the linear block codes:

- (i) Hamming weight: the hamming weight of a code vector  $x$  is defined as the number of non-zero components of code vector  $x$ .
- (ii) Hamming distance: the hamming distance between two vectors is defined as the number of components in which they differ.
- (iii) Minimum distance: the minimum distance of a block code is the smallest distance between any pair of codewords in the code.

Nowadays the linear block code can be distributed as Hamming codes, Cyclic codes, Binary BCH codes and Reed-Solomon codes shows in Figure 2.2. Each code got difference property, but all the linear block code property can fulfill this all codes.



**Figure 2.2: Linear block codes (Korhonn, 2003)**

### 2.2.2 Hamming Code

The hamming code are extremely simple to design and construct because when a single error occurs, the syndrome of the received vector is equal to the  $H^T$ . Hence if

we choose the  $n$  rows of the  $n \times (n-k)$  matrix  $H^T$  to be distinct, then the syndrome of all single errors will be distinct and we can correct single error.

For any positive integer  $m \geq 3$ , then this codes will exists Hamming code with the following property (Scott, 2008):

$$\text{Code length} : n = 2^m - 1, \quad (2.4)$$

$$\text{Number of information symbols: } k = 2^m - m - 1, \quad (2.5)$$

$$\text{Number of parity-check symbols: } n - k = m, \quad (2.6)$$

$$\text{Error-correcting capability} : t = 1 \text{ (} d_{\min} = 3 \text{)}. \quad (2.7)$$

### 2.2.3 Cyclic Codes

The cyclic codes got one important property that is the each code shifter by one and can get another code words. Here got a special factor denoted by  $g(D)$ , is called the generator polynomial of the code. The  $g(D)$  is equivalent to the generator matrix  $G$  as a description of the code. The encoder formula is  $x(D) = b(D) + D^{n-k}m(D)$ . The  $b(D)$  is the message  $m(D)$  divide by  $g(D)$  of the remainder.

### 2.2.4 BCH Codes

Binary BCH codes are one of the most important and powerful cyclic codes. For any positive integers  $m \geq 3$  and  $t < (2^m - 1)/2$  there exists a binary BCH code with the following parameters:

$$\text{Block length: } n = 2^m - 1,$$

$$\text{Number of message bits: } k \geq n - mt,$$

$$\text{Minimum distance: } d_{\min} \geq 2t + 1.$$

The BCH codes provide a large selection of block length, code rates, alphabet sizes, and error correcting capability.

### 2.2.5 Reed-Solomon code

Reed-Solomon code is the important of the nonbinary BCH codes. Reed-Solomon code work on symbols rather than individual bits. A t-error correcting Reed-Solomon code has the following parameters:

Block length:  $n=2^m-1$ ,

Message size:  $k$ ,

Parity –check size:  $n-k=2t$ ,

Minimum distance:  $d_{\min}=2t+1$ ,

Number of correctable errors:  $t=0.5(d_{\min}-1)$

## 2.3 Convolutional Codes

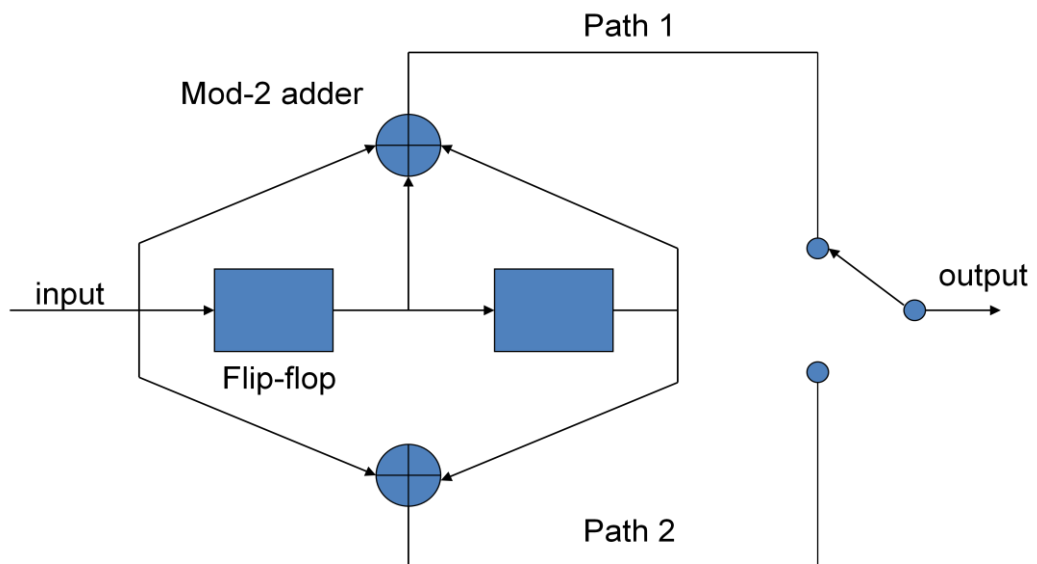
Convolutional code is the one with memory code (tree code) and the linear block code is without memory code( block code). Convolutional code structure is a tree code and it is fundamentally difference from the structure of a block code. Convolutional code unlike the linear block codes, large minimum distance and low error probabilities are achieved not by increasing the  $k$  and  $n$  but need increase the memory order  $m$ .

Convolutional code can have many generator polynomial  $g(D)$ . All convolutional code can be realized using a linear feed forward shift register encoder of this type. Normally the convolutional codes are preferred in many telecommunication application. Hence, here will introduce the terminology of convolutional codes shown in Table 2.1

**Table 2.1: Terminology of Convolutional Codes**

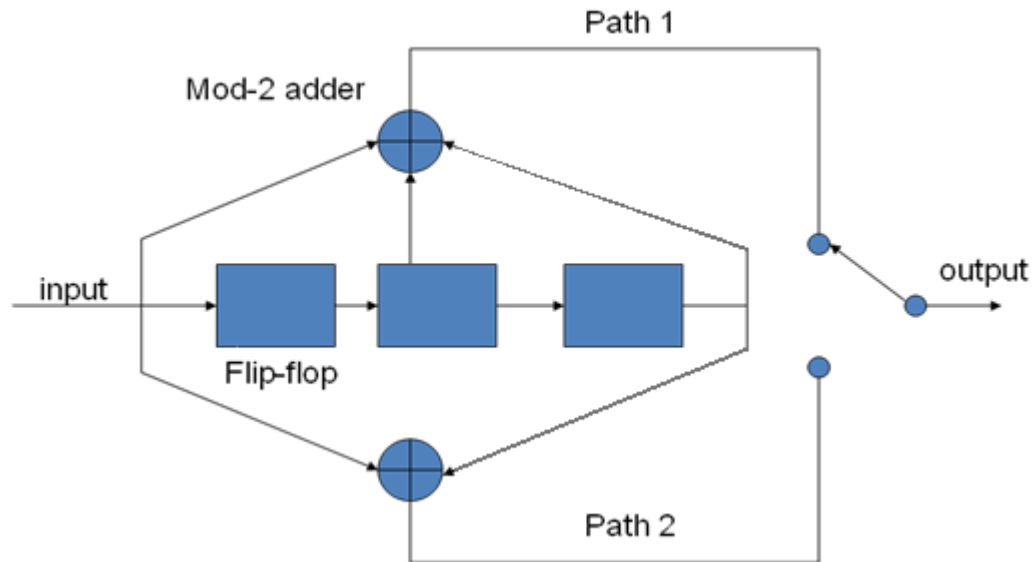
Input Frame, $k$	• Number input bits taken into the encoder at once
Output Frame, $n$	• Number output bits produced from the encoder at once
Memory order, $M$	• Maximum number of shift register stage in the path to any output bit.
Memory constraint length, $m$	• Total number of shift registers in the encoders.
Input constraint length, $K = m+k$	• Total number of bits involved in the encoding operation

Here is the example of the convolutional codes in figure 3. This diagram shows that the convolutional codes can have more than one generator polynomial for difference output. The path 1 generator polynomial is  $g_1=(111)$  and the path 2 generator polynomial is  $g_2=(101)$ . And the other this one need 2 registers to store the memory. From this diagram, we can get 3 difference diagram to easy understand and encoder. The 3 difference way to represent the encoder is code tree, state diagram and trellis diagram.



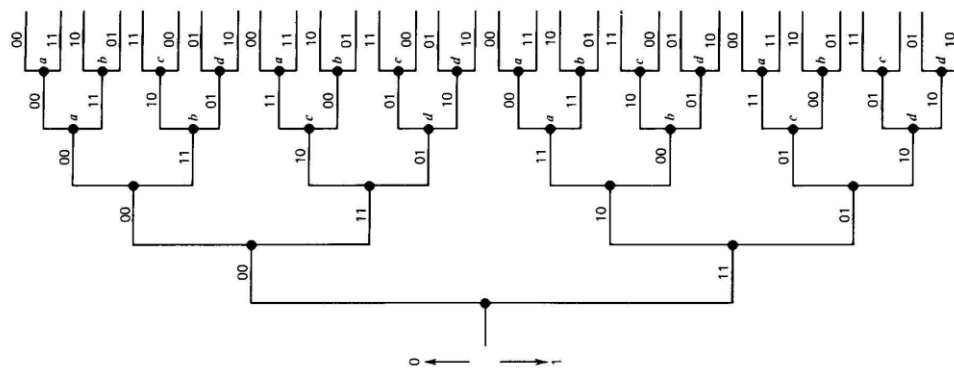
**Figure 2.3: Convolutional encoder with constraint length  $K=3$ , rate  $r=1/2$**   
 (Berrou C. , 2010)





**Figure 2.4: Convolutional encoder with constraint length  $K=4$ , rate  $r=1/2$  (Berrou C. , 2010)**

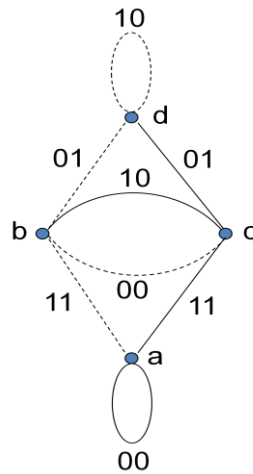
The tree diagram shown in Figure 2.5 is simple to represent the convolutional code output and the flip-flop of register memory by its code tree. There are only 2 ways to go from a starting point. When the input is '0', then it will go upward (left). On the other hand, when the input is '1', then it will go downward (right). From this diagram, all the 2 digits in the line represent the output state. For the alphabets, it is represented by the flip-flop memory.



**Figure 2.5: Code Tree for the Convolutional Code (Berrou C. , 2010)**

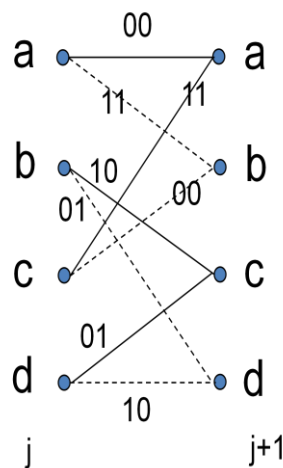
Convolutional code is a code with memory, so that the previous input will affect the next state. For this state diagram, the current output of the convolutional

encoder from Figure 2.6 is depends on previous state. So that, the code output or next state can be form as a circle. The alphabets still represented same things as the code tree is the flip-flop memory. The digit is the output. The solid branch is represent the input '0' and the dashed branch is represent the response to input '1'.



**Figure 2.6: State diagram of the convolutional codes (Berrou C. , 2010)**

This trellis diagram show in Figure 2.7 left nodes represent the four possible flip-flop current state same as previous figure and the right node is represent the next state of the register memory. Same thing the input '0' is represented by a solid branch and input '1' is represented by a dashed branch.



**Figure 2.7: Portion of Trellis diagram for convolutional encoder with constraint length  $K=3$ , rate  $r=1/2$  (Berrou C. , 2010)**

This 3 diagram result is get from the first diagram of the convolutional codes. For decoding the convolutional code we can use the viterbi algorithm. The viterbi algorithm is a method commonly used for decoding bit stream encoded by convolutional encoder. Viterbi algorithm realized that not all paths needs to be considered, if the channel errors are random then non-optimal paths at this stage can never become optimal path in the future. This type algorithm only keep one of the paths reaching each node and at end state only  $2^m$  paths need to be retained.

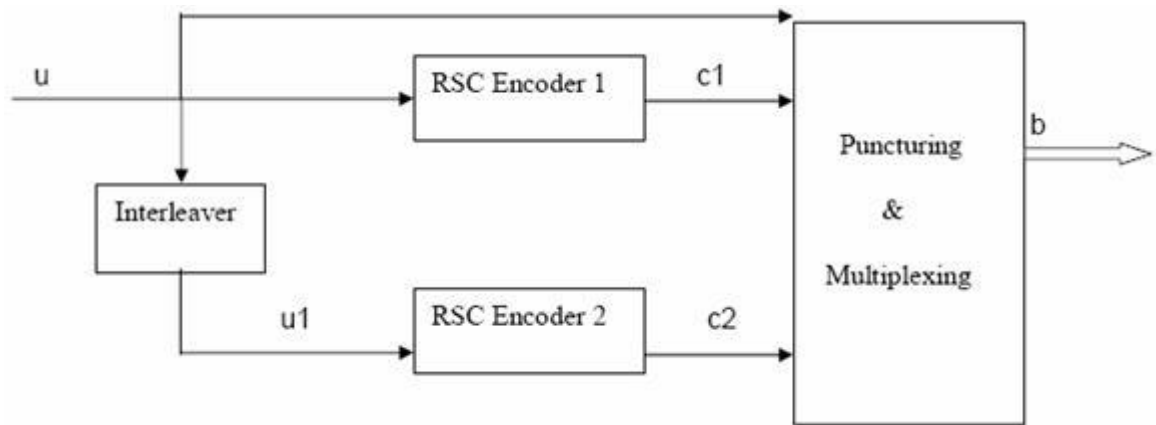
## **2.4 Turbo Codes**

Turbo code was introduced in 1993 by Berrou, Glavieux, and Thitimajshima, and reported in (Berrou, Glavieux, & Thitimajshima, 1993). In the field of forward-error correction channel coding are a recent developmental by turbo codes. This codes got three simple ideas to make use: parallel concatenation of codes to allow simpler decoding, interleaving provide a better weight distribution and soft decoding to increase decoder decisions and maximize the gain from decoder interaction.

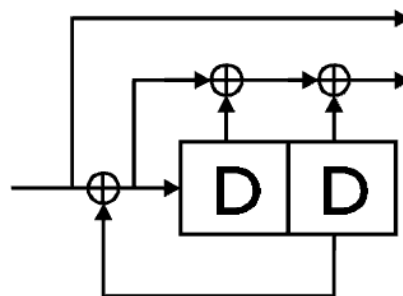
### **2.4.1 Turbo Codes Structure**

The basic ides of turbo codes is use two convolutional codes in parallel and the second encoder input bits will process to interleaving just go through the encoder 2. The operation of Turbo encoding is based on the use of a pair of encoders, separated by the interleave, and iterative detection involving the use of feedback around a pair of decoders separated by a deinterleaver and an interleave. The turbo code can be regard as a large block code. The encoder user a parallel FEC encoding scheme in which the information is systematically encoded by two separate identical encoders. The performance depends on the weight distribution. So that, the input patterns giving low weight words from first encoder will to be interleaved to the more height

weight distribution for the second encoder. Turbo codes provide significant improvements in the quality of data transmission over a noisy channel. Figure 2.8 illustrates the functional block diagram of a turbo encoder.



**Figure 2.8: Structure of a Turbo Encoder** (Rekh, rani, & A.Shanmugam, 2000)



**Figure 2.9: Structure of the RSC Encoder** (Rekh, rani, & A.Shanmugam, 2000)

In figure 2.9 generator polynomial is  $g=(1,(1+D+D^2/(1+D^2)))$  and we have shown an encoder on the recursive systematic form. From the diagram we can find out that the turbo codes is a systematic codes because the first output is original bits and not process to any encoder. The rest of output is the parity bits. Notice that the fact that the codes are systematic is just a coincidence, although it turns out to be very convenient for several reasons. One of these is that the bit error rate (BER) after decoding of a systematic code can not exceed the BER on the channel. Imagine that the received parity symbols were completely random, then the decoder would of

course stick to the received version of the information. If the parity symbols at least make some sense we would gain information on the average and the BER after decoding will be below the BER on the channel.

### **2.4.2 Puncturing**

One thing is important concerning the systematic property, though. If we transmit the systematic part from both encoders, this would just be a repetition, and we know that we can construct better codes than repetition codes. The information part should only be transmitted from one of the constituent codes, so if we use constituent codes with rate  $1/2$  the final rate of the turbo code becomes  $1/3$ . If more redundancy is needed, we must select constituent codes with lower rates. Likewise we can use puncturing after the constituent encoders to increase the rate of the turbo codes. The information bits and the parity bits generated by the two encoders are the multiplexed and punctured in a repeating pattern to increase the code rate for transmission.

### **2.4.3 Interleaving**

Interleaving is a process to rearranging the input data stream of sequence in a one to one deterministic format. In turbo coding, interleaving is used before the second encoder want to input the information data to encoder. The basic role of an interleaving is to construct the long block code from small memory convolutional codes, as long as can approach the Shannon capacity limit. The final role of the interleaver is to break low weight input sequences, and hence increase the code free Hamming distance or reduce the number of code words with small distances in the code distance spectrum. The size and structure of interleavers play a major role in the performance of turbo codes. Here got one of interleavers, which can be implemented.

The random interleaver uses a fixed random permutation and maps the input sequence according to the permutation order. In this turbo code, we recommend use

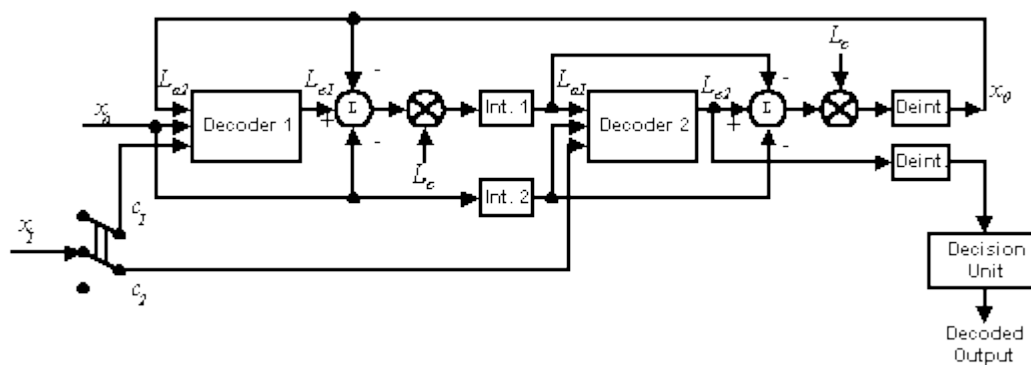
the random interleaver. The random interleave, also called turbo interleave, enables the second encoder to reorder the information bits prior to encoding. Table 2.2 shows a 3 x 5 matrix positions after the matrix is interleaved. This is the example shows that if the input is 15 bits, then will permutation to randomly.

**Table 2.2: Equivalent index position for original matrix index in diagonal interleaver** (Rekh, rani, & A.Shanmugam, 2000)

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Interleaver(i)	0	6	12	1	7	13	2	8	14	3	9	10	4	5	11

#### 2.4.4 Turbo Decoding

It is proposed that an iterative decoding scheme should be used. The decoding algorithm is similar to Viterbi algorithm in the sense that it produces soft outputs. While the Viterbi algorithm outputs either 0 or 1 for each estimated bit, the turbo code decoding algorithm outputs a continuous value of each bit estimate. While the goal of the Viterbi decoder is to minimize the code word error by finding a maximum likelihood estimate of transmitted code word, the soft output decoding attempts to minimize bit error by estimating the posterior probabilities of individual bits of the code word. We called the decoding algorithm Software Decision Viterbi Decoding.



**Figure 2.10: Structure of Turbo Decoding** (Proakis, 1995)

The turbo decoder consists of  $M$  elementary decoders - one for each encoder in turbo encoding part. Each elementary decoder uses the Software Decision Viterbi Decoding to produce a software decision for each received bit. After an iteration of the decoding process, every elementary decoder shares its soft decision output with the other  $M - 1$  elementary decoders.

In theory, as the number of these iterations approaches infinity, the estimate at the output of decoder will approach the maximum a posteriori (MAP) solution.

#### **2.4.5 Turbo Decoding MAP Algorithm Overview**

In MPSK or QAM demodulation process, decision about a bit is made by looking at what decision region the phase or the amplitude falls in. This way to make decision is called the Maximum Likelihood Detection(MLD).

A similar but better rule is based on knowing the priory probability of the signal. If a -1 bit has a probability of 80%, then if the signal falls in the negative decision range, the MLD will decide it as a +1. However, it is clear that the priory probability as here of 80% should be taken into account. This decision method that on this conditional probability is called the Maximum Aposteriori Probability(MAP). (Langton, 2006)

## **CHAPTER 3**

### **METHODOLOGY**

#### **3.1 Introduction**

MATLAB has been used in this project. MATLAB is user friendly and most efficient to simulation turbo codes in this section. The MATLAB is a high-performance language for computing, simulating and programming in a convenient environment. Thus, MATLAB is researcher first choice to simulation turbo codes. After simulation, researcher collect simulation result bit error rate (BER) versus signal to noise ratio ( $E_b/N_0$ ) are plotted using Microsoft Office excel 2007.

#### **3.2 Simulation Setup**

The Turbo code m-files developed in this project were based on original work by Yufer Wu (YuFei, 2005). The matlab Turbo Code structure used by Wu is based upon the structure described in the original paper by Berrou. (Berrou, Glavieux, & Thitimajshima, 1993)

This simulation is based on the classical parallel concatenated convolutional codes (PCCC) system. In this structure got two components of RSC (Recursive Systematic Convolutional) encoders with code rate 1/3 (not puncture) are designed. First encoder is terminated with tails bits. Next, both the info bits and the tail bits are scrambled and passed to the second encoder, while second encoder is left open

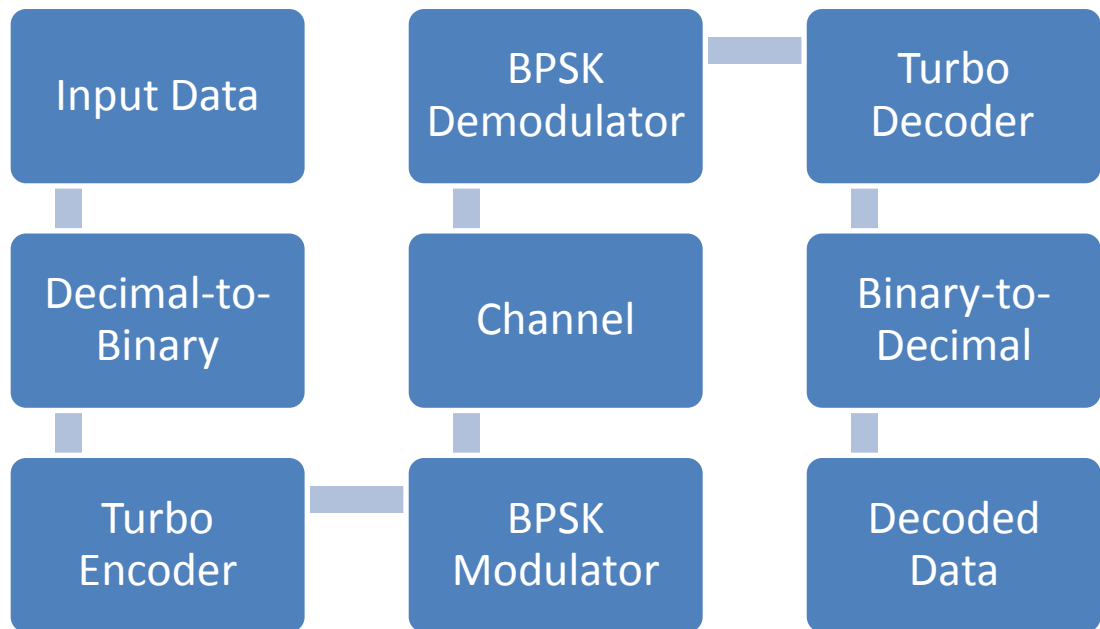


without tail bits. Random information bits are modulated into  $+1/-1$ , and transmitted through an AWGN channel. Interleavers are randomly generated for each frame. Log-MAP algorithm without quantization or approximation is used. Below is the list of functions used to call the main program:

- i. Function `bin_state`: To convert a vector of integer into a matrix
- ii. Function `demultiplex`: To get the codeword of each encoder at the receiver end.
- iii. Function `encode_bit`: This function takes as an input a single bit to be encoded, as well as the coefficients of the generator polynomials and the current state vector. It returns as output  $n$  encoded data bits, where  $1/n$  is the code rate.
- iv. Function `encoderm`: This function interleaves input of the second encoder, if unpunctured, produces a rate  $1/3$  output of fixed length, if punctured, produces a rate  $1/2$  output. Multiplexer chooses odd check bits from RSC1, even check bits from RSC2, and determine the constraint length ( $K$ ), memory ( $m$ ) and number of information bits plus tail bits. Also, performs BPSK Antipodal modulation:  $+1/-1$ .
- v. Function `int_state`: This function converts a row vector of  $m$  bits into a integer (base 10)
- vi. Function `logmapo`: To compute the soft output, log-likelihood ratio of symbols in the frame using Log MAP algorithm.
- vii. Function `rsc_encode`: To generate codeword.
- viii. Function `trellis`: To set up the trellis for the given ,  $G(D)$

### 3.3 Interpretation of the Simulation Model

The standard description of the system is the block diagram, where each block represents a signal processing operation (Balamuralithara, 2005), and it shown in Figure 3.1. WCDMA standard parameters are used for this investigation and the output bits of the turbo encoder are then modulated using a Binary Phase Shift Keying (BPSK) modulator.



**Figure 3.1: Block Diagram of the simulation model**

Each block contains the algorithm and equations needed to implement the block functions within the simulation (Balamuralithara, 2005).

#### 3.3.1 Input Data

Researcher are using a randomly values for frame size. In this simulation, researcher using  $N=500$  and  $N=1200$  for frame size values. The line below is used in the source code to generate random values:

$$x = \text{round}(\text{rand}(1, L_{\text{total}} - m))$$

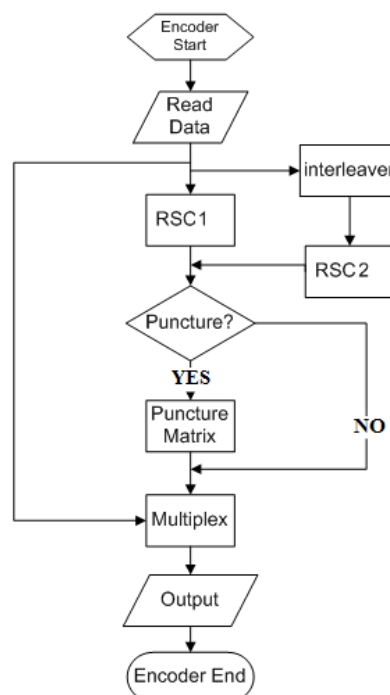
### 3.3.2 Decimal-to-binary

The input data all in decimal form but in the encoder and modulator process all in binary form. So, the function of this module is to transform the source from the decimal value to binary value.

### 3.3.3 Turbo encoder

Turbo code encoder is already been description in Chapter 2. The simulation turbo code encoder is composed of two identical RSC component encoders. A random interleaver separates these two encoders and this is a random permutation of bit order in a bit stream.

In this simulation, researcher basically using code rate of  $1/3$  (unpunctured) and some of the journals are using  $1/2$  as is achieved by puncturing the coded bit streams of the turbo code. Figure 3.2 illustrate the flow chart of the turbo encoder in this simulation model.



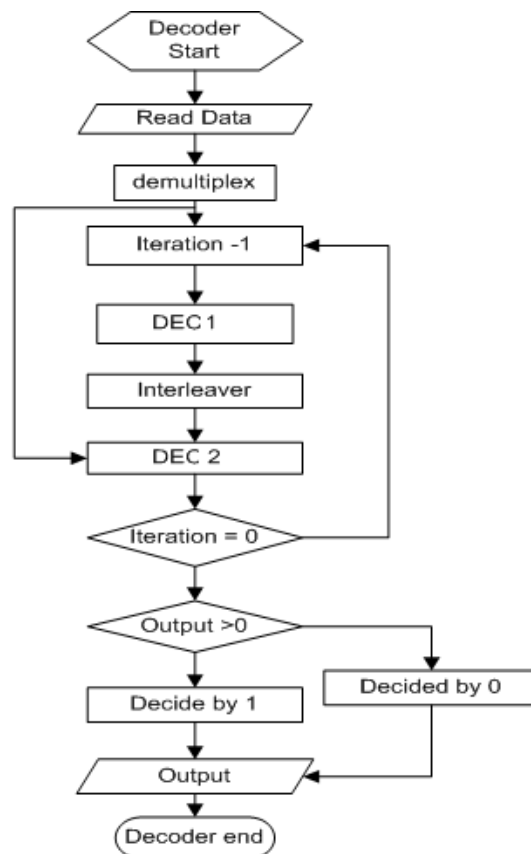
**Figure 3.2 : The encoder flow chart**

### 3.3.4 Channel

In this simulation, the Additive White Gaussian Noise (AWGN) channel model is used because it is useful to simulation the realized underlying behaviour of a system and AWGN channel is also a good approximation for many satellite and deep space communication links. The Gaussian noise easy to construct from the Gaussian distribution with mean of zero and standard deviation of one.

### 3.3.5 Turbo Decoder

The turbo code decoder system have 2 type decoder: soft input soft output (SOVA) and the log maximum a posterior algorithm (log-MAP). In these simulation, researcher are using the log-Map algorithm to investigation. The flow chart of the turbo decoder simulation is illustrated in figure 3.3.



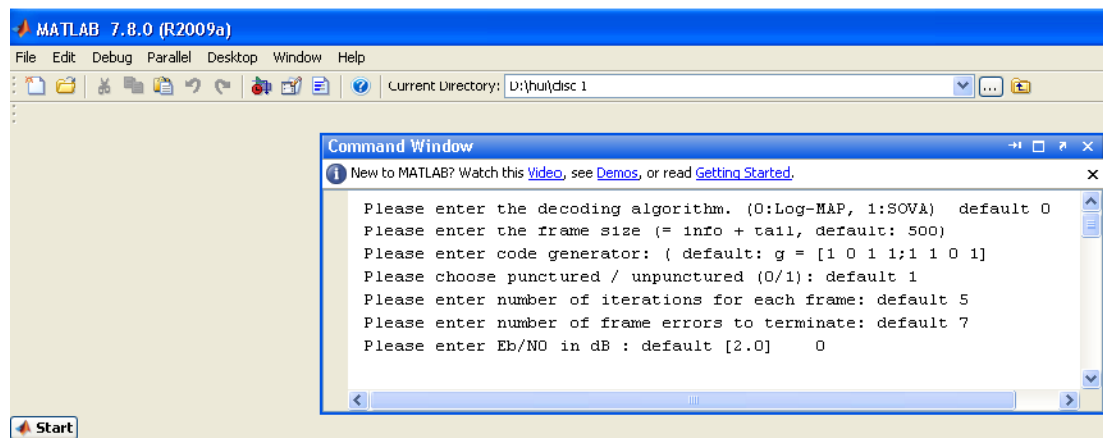
**Figure 3.3 : The turbo decoder flow chart**

### 3.3.6 Binary-to-decimal

The output of the decoder is binary but the system read it as decimal value, so must transform the binary number to decimal number and recover it to the input of the encoder.

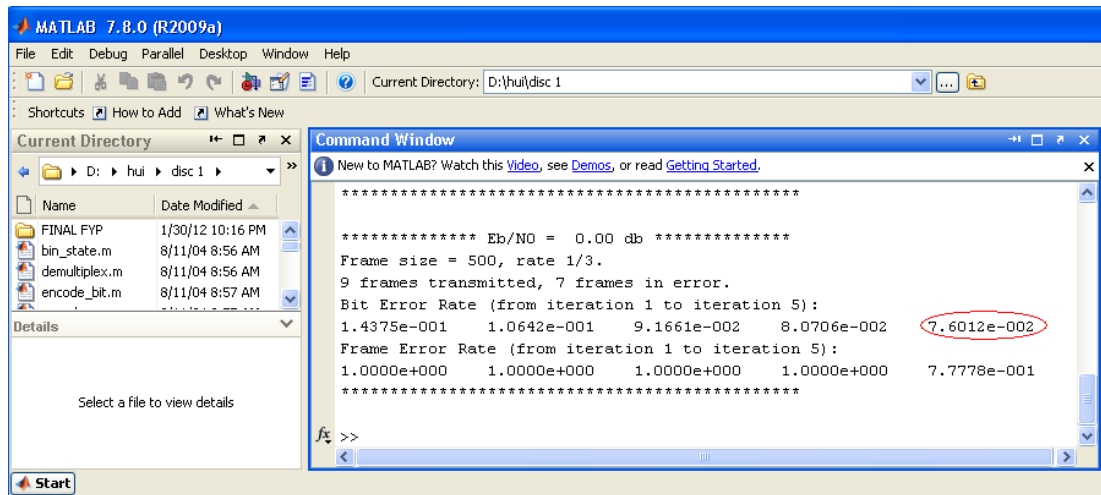
## 3.4 Step procedure in Matlab

Matlab programme takes in all the design parameter and passes it over to the functions. All the design parameter is defined in the main function itself. Figure 3.4 display a print screen of the matlab simulation interface where user can choose the decoding algorithm and input the value for the frame size, punctured, iterations number, frame errors to terminate and the SNR.



**Figure 3.4 : User interface of the Turbo Code simulator**

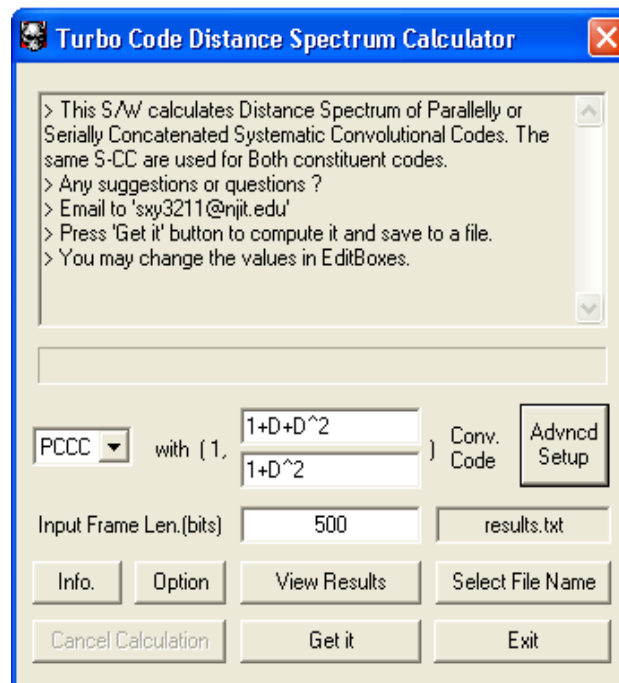
After simulation finish, researcher can view all the result. Figure 3.5 shows that researcher need copy Bit Error Rate (BER) to the MS-Excel to plot the Bit Error Rate (BER) against the channel condition that is Signal to Noise Ratio (SNR). The plotted graphs are shown in detail in next chapter.



**Figure 3.5: The matlab workspace with the result**

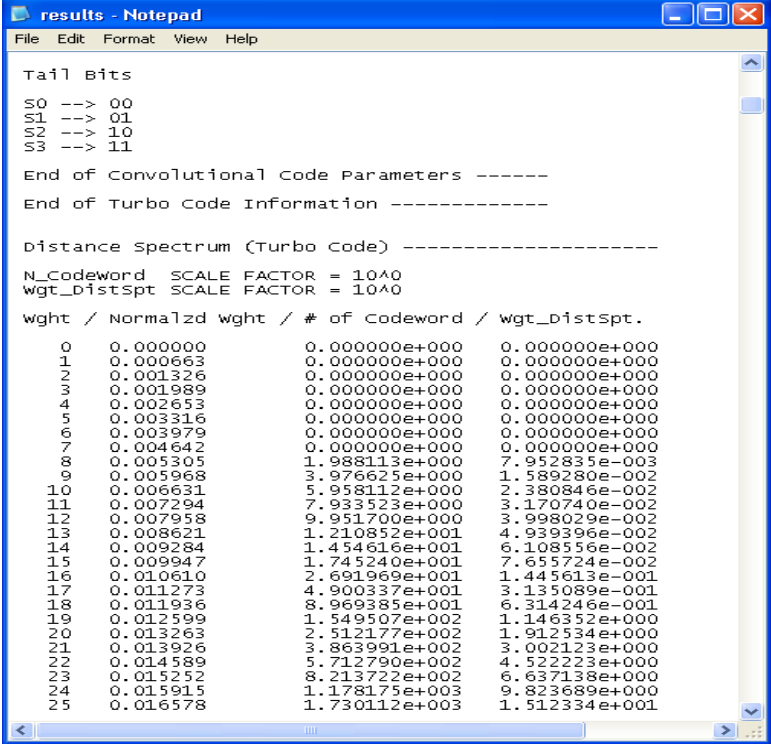
### 3.5 Turbo Code Distance Spectrum Calculator

This software computes the distance spectrum of parallel or serially concatenated convolutional code (overall rate of  $1/3$  for PCCC and  $1/4$  for SCCC) where the same  $1/2$  rate systematic convolutional code is used for both the constituent codes. (Yoon, 2009). Researcher can test with various generator polynomials. It will create a test file containing the BER performance of the turbo code. A print Screen of the Turbo Code Distance Spectrum Calculator is shown in figure 3.6.



**Figure 3.6 : Turbo Code Distance Spectrum Calculator**

Distance Spectrum is equal to the number of codewords that has certain Hamming weight, averaged over all possible permutations (Uniform Interleaver Assumption). It is for computaion of Codeword Error probability bound. While, the weighted Distance Spectrum is for computation of Bit Error Probability Bound. After software finish analysis, the result is shown in text file. Researcher can analysis the code weight and compare the result from the matlab simulations result to verify conclusions and justification. A screen shot of the text file obtained from the simulation is presented in figure 3.7.



```

results - Notepad
File Edit Format View Help

Tail Bits
S0 --> 00
S1 --> 01
S2 --> 10
S3 --> 11

End of Convolutional Code Parameters -----
End of Turbo Code Information -----

Distance Spectrum (Turbo Code) -----
N_Codeword SCALE FACTOR = 10^0
wgt_distSpt SCALE FACTOR = 10^0

wght / Normalzd wght / # of Codeword / wgt_distSpt.
0 0.000000 0.000000e+000 0.000000e+000
1 0.000663 0.000000e+000 0.000000e+000
2 0.001326 0.000000e+000 0.000000e+000
3 0.001989 0.000000e+000 0.000000e+000
4 0.002653 0.000000e+000 0.000000e+000
5 0.003316 0.000000e+000 0.000000e+000
6 0.003979 0.000000e+000 0.000000e+000
7 0.004642 0.000000e+000 0.000000e+000
8 0.005305 1.988113e+000 7.952835e-003
9 0.005968 3.976625e+000 1.589280e-002
10 0.006631 5.958112e+000 2.380846e-002
11 0.007294 7.933523e+000 3.170740e-002
12 0.007958 9.951700e+000 3.998029e-002
13 0.008621 1.210852e+001 4.939396e-002
14 0.009284 1.434616e+001 6.108556e-002
15 0.009947 1.745240e+001 7.655724e-002
16 0.010610 2.691969e+001 1.445613e-001
17 0.011273 4.900337e+001 3.135089e-001
18 0.011936 8.969385e+001 6.314246e-001
19 0.012599 1.549507e+002 1.146352e+000
20 0.013263 2.512177e+002 1.912534e+000
21 0.013926 3.863991e+002 3.002123e+000
22 0.014589 5.712790e+002 4.522223e+000
23 0.015252 8.213722e+002 6.637138e+000
24 0.015915 1.178175e+003 9.823689e+000
25 0.016578 1.730112e+003 1.512334e+001

```

**Figure 3.7 : The screen shot of the notepad results obtained from simulation**



## **CHAPTER 4**

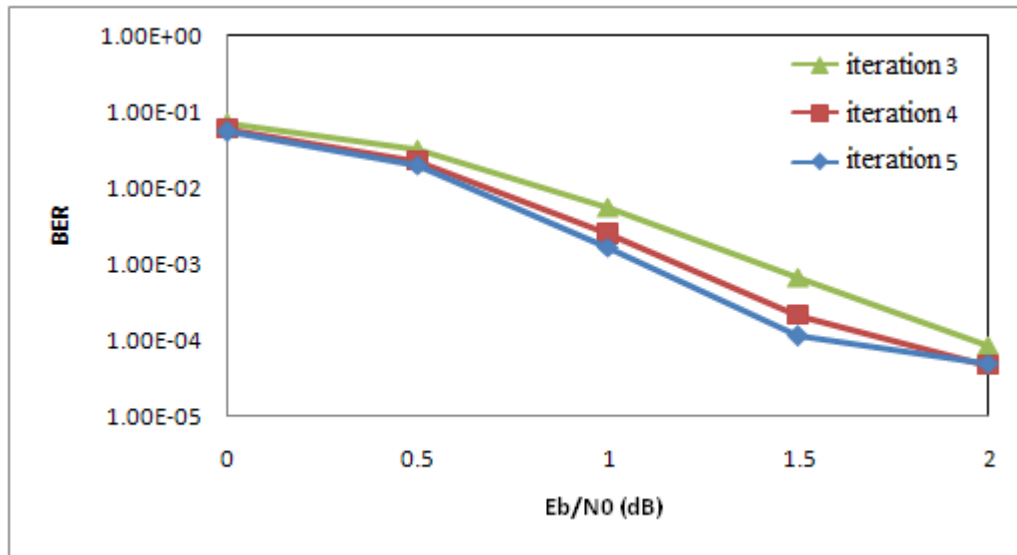
### **DISCUSSION**

#### **4.1 Introduction**

This chapter presents simulation results for the implementation of classical turbo codes in Matlab simulation platform. Researcher need to observe the effects of the generator polynomial on the turbo code performance in various scenarios. The effect of generator polynomials with constraint length,  $K=3, 4$  and  $5$  are investigated for the frame length of  $N=500$  for different iteration and for the decoding algorithm used Log-Map and the code rate was fixed at  $1/3$ .

#### **4.2 Effects of number of decoding iterations**

When the number of iteration increases will cause the performance of the turbo code increases. However, the improvement of the BER will not be significant at certain regions of  $E_b/N_0$ . But, there will increase processing time delay as the iteration increase. This is because this action will increase the complexity and reduce the system efficiency. So those, researcher need to justify the number of iterations to get the optimum performance result. Figure 4.1 shows turbo code performance due to effects of number of iterations.

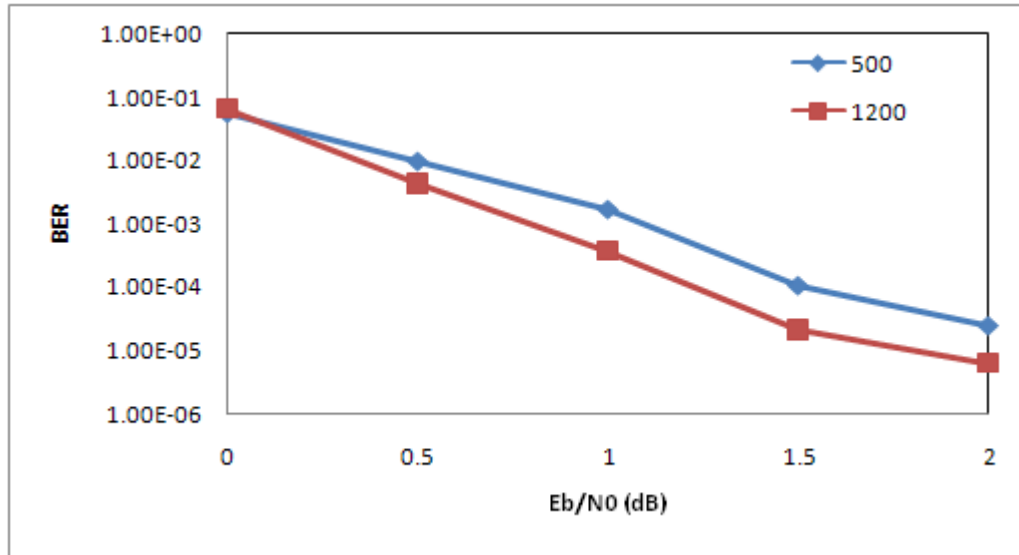


**Figure 4.1: Turbo code performance for frame length,  $N=500$  for iteration 3, 4 and 5 with  $g(D)=[7,5]_{10}$ , code rate  $=1/3$  under AWGN channel**

From figure 4.1, we can find out that the five iterations are good enough to get the good performance results for the frame size of 500 bits. So, we chose the five iterations for simulation of frame size of 500 bits under AWGN.

### 4.3 Frame size effects

Frame size will effect the performance of turbo code. Large frame size means that the larger distance between each frame can be interleaved and the decoder can get better performance because the correlation between two adjacent bits will becomes smaller. But, if increase the frame size will also increase the processing time as turbo code is a one type of block code, it need to wait for the completion of the decoding of the whole block before getting the decoded output. So that, we need to certain limit the frame size to avoid higher processing delay due to larger amount of transmitted data (Balamuralithara, 2005). From the figure 4.2 shows the simulation result verified this conclusion. In this simulation, we are using  $N=500$  and  $N=1200$ , and we can see that the turbo code with larger frame size has better performance.



**Figure 4.2 : Comparison of frame length,  $N=500$  and  $1200$  for bit error rate versus  $E_b/N_0$  with  $g(D)=[7,5]$ , code rate= $1/3$ , number of iteration 7.**

#### **4.4 Effects of generator polynomial**

Generator Polynomials are used for this simulation are shown in table 4.1. In this simulation, we only use up to constraint length,  $K=5$  with the study of seven combinations of the generator polynomials as the complexity of the simulation. All the display generator polynomial value is decimal.

**Table 4.1 : Various combination of generator polynomials**

Constraint Length, $K$	Generator Polynomial, $g(D) = [g_1, g_2]_{10}$
$K=3$	$[5, 7]_{10}$
	$[7, 5]_{10}$
$K=4$	$[9, 11]_{10}$
	$[9, 13]_{10}$
	$[9, 15]_{10}$
	$[11, 9]_{10}$
	$[11, 13]_{10}$
	$[11, 15]_{10}$
	$[13, 9]_{10}$
	$[13, 11]_{10}$
	$[13, 15]_{10}$
	$[15, 9]_{10}$
	$[15, 11]_{10}$
	$[15, 13]_{10}$
$K=5$	$[19, 17]_{10}$
	$[21, 17]_{10}$
	$[23, 17]_{10}$
	$[25, 17]_{10}$
	$[27, 17]_{10}$
	$[29, 17]_{10}$
	$[31, 17]_{10}$

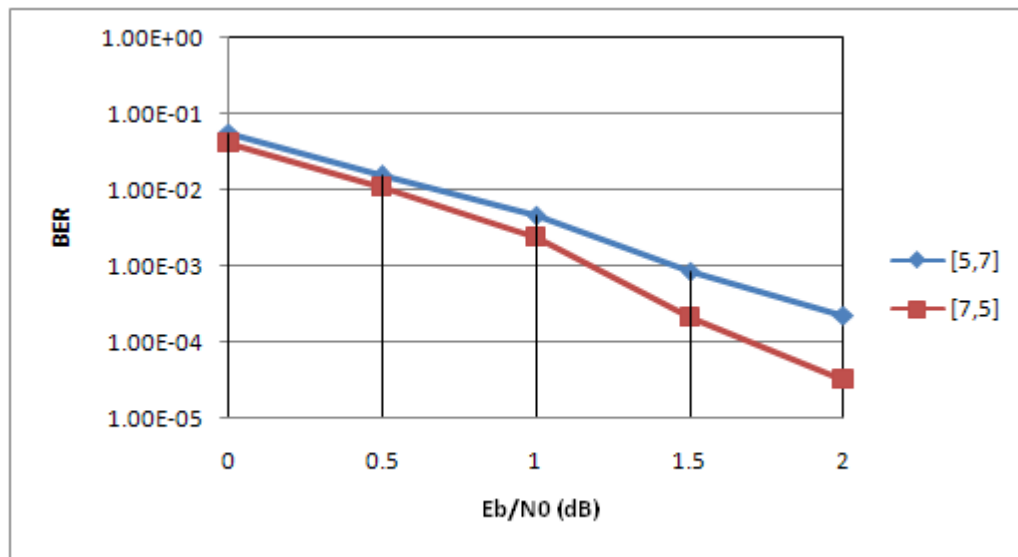
#### 4.4.1 Effects of Generator Polynomial

The default parameters in this case simulation are shown in Table 4.2.

**Table 4.2: Default parameters in simulation**

Channel	=AWGN Channel
Decoder	=Log-MAP decoder
Frame Size	=500 bits
Code Rate	=1/3
Iteration number	=5
Terminate frame errors	=7

##### 4.4.1.1 Simulation results for constraint length, $K=3$



**Figure 4.3 : Turbo code performance for constraint length,  $K=3$**

Figure 4.3 plotted that the BER performance of turbo code with  $g(D)=[7,5]$  and  $g(D)=[5,7]$ . From the graph, we can find out that the  $g(D)=[5,7]$  is not good enough compare with  $g(D)=[7,5]$ . At BER  $10^{-3}$  the approximated coding gain is 0.3dB.

In this case, this phenomena is caused by the codewords. The  $g(D) = [7,5]$  has fewer codewords with relatively low codeweights than  $g(D)=[5,7]$ . So that, we can assume

that  $g(D) = [7,5]$  has higher number of high weighted codewords and this property can be prove and understand by using the software “Turbo Code Distance Spectrum Calculator”.

wght /	Normalzd wght /	# of Codeword	wght /	Normalzd wght /	# of Codeword
0	0.000000	0.000000e+000	0	0.000000	0.000000e+000
1	0.000663	0.000000e+000	1	0.000663	0.000000e+000
2	0.001326	0.000000e+000	2	0.001326	0.000000e+000
3	0.001989	0.000000e+000	3	0.001989	0.000000e+000
4	0.002653	0.000000e+000	4	0.002653	0.000000e+000
5	0.003316	0.000000e+000	5	0.003316	0.000000e+000
6	0.003979	0.000000e+000	6	0.003979	0.000000e+000
7	0.004642	0.000000e+000	7	0.004642	1.197601e-002
8	0.005305	1.988113e+000	8	0.005305	1.517261e-004
9	0.005968	3.976625e+000	9	0.005968	8.150451e-002
10	0.006631	5.958112e+000	10	0.006631	1.997816e+000
11	0.007294	7.933523e+000	11	0.007294	2.853925e-001
12	0.007958	9.951700e+000	12	0.007958	3.993583e+000
13	0.008621	1.210852e+001	13	0.008621	6.871884e-001
14	0.009284	1.454616e+001	14	0.009284	6.093278e+000
15	0.009947	1.745240e+001	15	0.009947	1.394761e+000
16	0.010610	2.691969e+001	16	0.010610	8.655047e+000
17	0.011273	4.900337e+001	17	0.011273	2.795989e+000
18	0.011936	8.969385e+001	18	0.011936	1.234132e+001
19	0.012599	1.549507e+002	19	0.012599	6.458152e+000
20	0.013263	2.512177e+002	20	0.013263	2.421541e+001
21	0.013926	3.863991e+002	21	0.013926	1.640342e+001
22	0.014589	5.712790e+002	22	0.014589	5.241066e+001
23	0.015252	8.213722e+002	23	0.015252	4.055875e+001

(a)  $g(D)=[5,7]$ (b)  $g(D)=[7,5]$ 

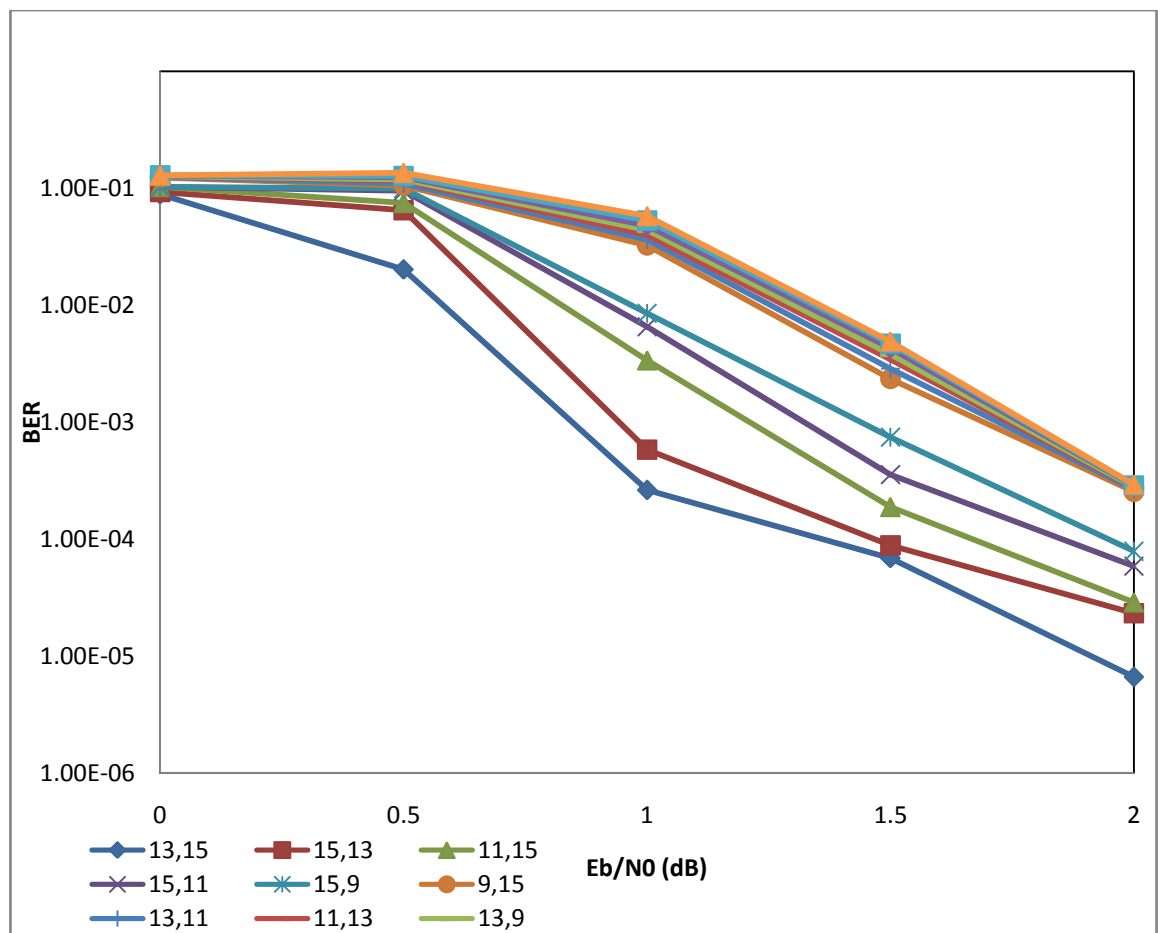
**Figure 4.4: The simulation results of number of codewords and weight distribution**

As this software, we can calculate the code weight of the encoded bits. Distance Spectrum is equal to the number of codewords that has certain Hamming weight, averaged over all possible permutations. It is for computation of Codeword Error probability bound. The print screen this software and shown in the figure 4.4 (a) and (b), we only shows the first 23 codeweight out of 1508. Codewords with weight 1-20 considered very low codeweights.

From the figure 4.4, we can observe that the figures 4.4(a) only at codeweight 0-12 are negligible, and the figure 4.4(b), we can negligible codeweight 0-19. So that, we can prove that the  $g(D)=[7,5]$  has fewer codewords with relatively low code weights than  $g(D)=[5,7]$ . In other way, we can observe the codeweight at 10, figure 4.4(a) have number of codewords is 6 (5.958112e+000). Figure 4.4 (b) have number of codewords at codeweight is 10 are 2 (1.997816e+000).

Hence, turbo code in constraint length  $K=3$  and  $g(D)=[7,5]$  were sought that would allow for more easier decoding compare with  $g(D)=[5,7]$  because one way of making the task of the decoder easier is using a code with mostly high-weight code words. High-weight code words can be distinguished more easily (K äper, 2006).

#### 4.4.1.2 Simulation results for constraint length, $K=4$

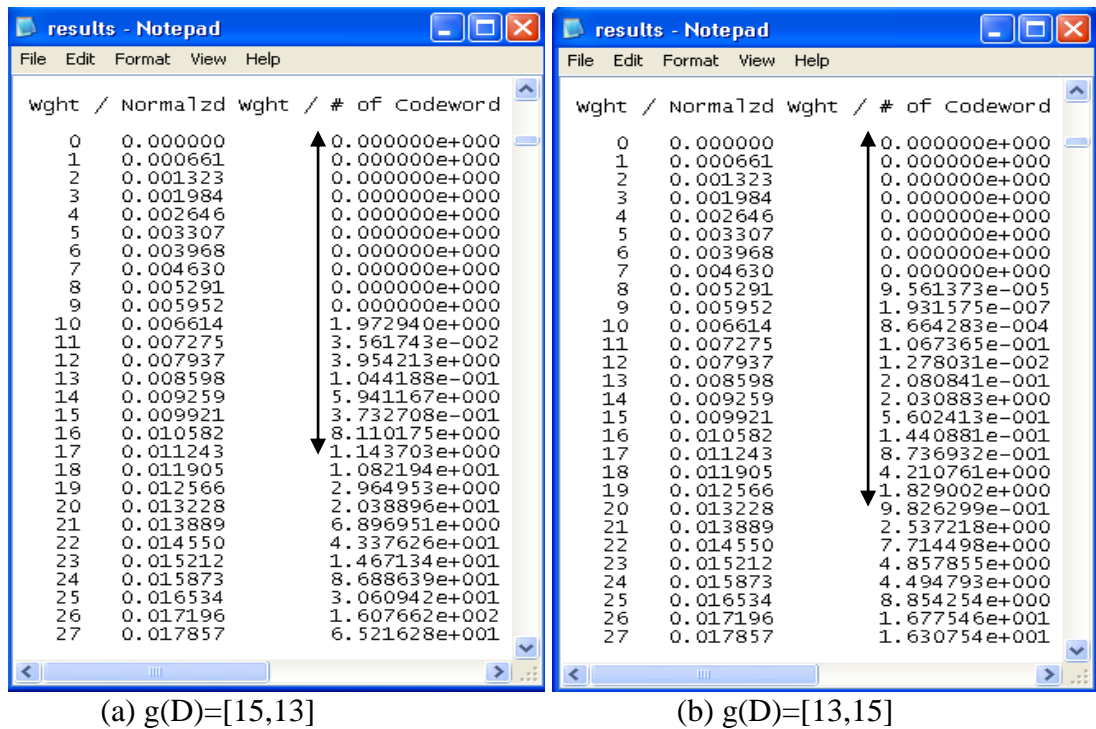


**Figure 4.5 : Turbo code performance for constraint length,  $K=4$**

Figure 4.5 shows that the generator polynomial  $g(D)=[13,15]_{10}$  given the best performance in turbo code system for constraint length  $K=4$ . From the graph, researcher conclude that  $g(D)=[13,15]_{10}$  better than  $g(D)=[15,13]_{10}$  as it shows a better BER performance. At BER  $10^{-3}$  the approximated coding gain is 0.2dB.

In this case, this phenomena is caused by the codewords. The  $g(D) = [13,15]$  has fewer codewords with relatively low codeweights than  $g(D)=[15,13]$ . So that, we can assume that  $g(D) = [13,15]$  has higher number of high weighted codewords and this property can be prove and understand by using the software “Turbo Code Distance Spectrum Calculator”.

The print screen this software and shown in the figure 4.6 (a) and (b), we only shows the first 23 codeweight out of 1508. Codewords with weight 1-20 considered very low codeweights.



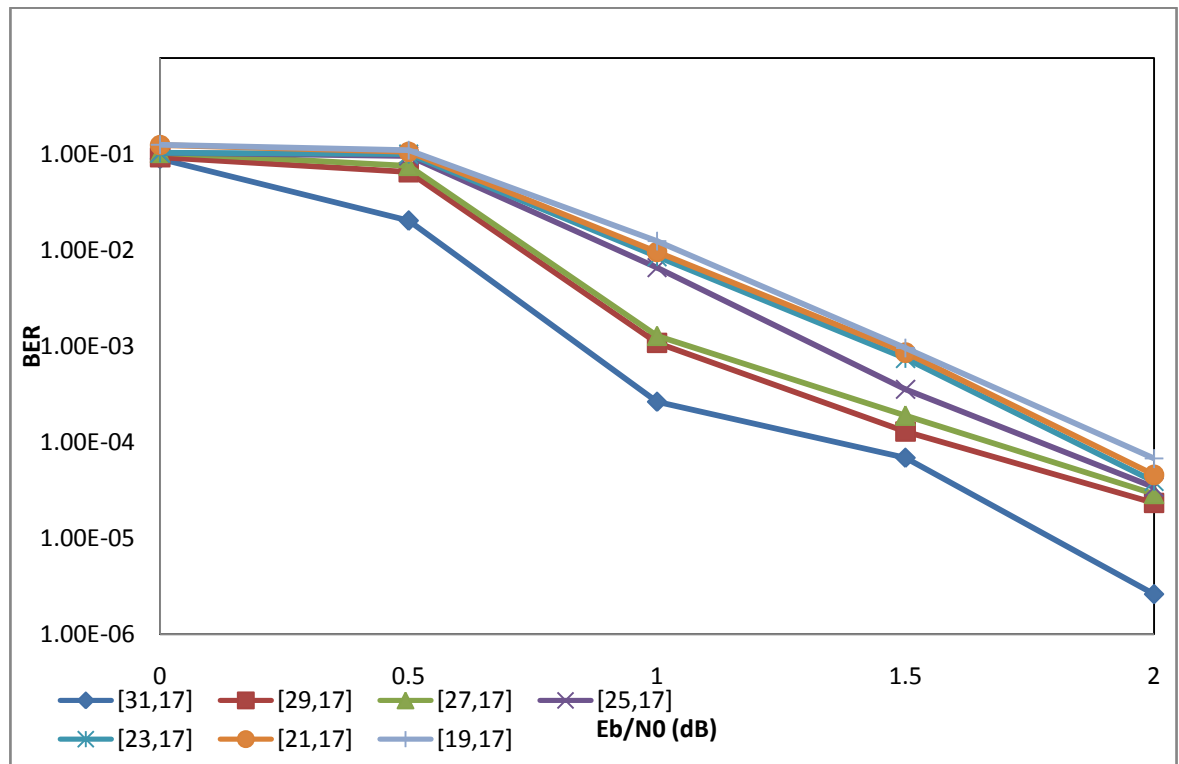
**Figure 4.6: The simulation result of No. of codewords and weight distribution for constraint length  $K=4$**

From the figure 4.6, we can observe that the figures 4.6(a) only at codeweight 0-16 are negligible, and the figure 4.6(b), we can negligible codeweight 0-20. So that, we can prove that the  $g(D)=[13,15]$  has fewer codewords with relatively low code weights than  $g(D)=[15,13]$ .



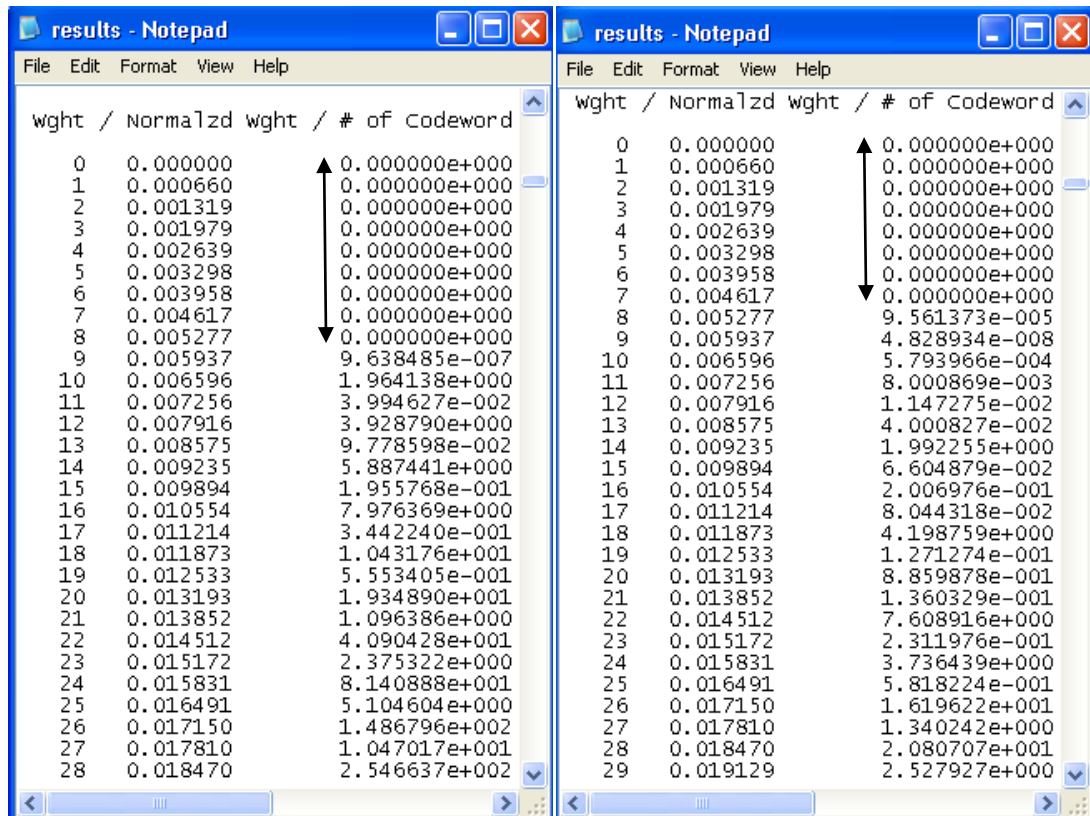
Hence, turbo code in constraint length  $K=4$  and  $g(D)=[13,15]$  were sought that would allow for more easier decoding compare with  $g(D)=[15,13]$  because one way of making the task of the decoder easier is using a code with mostly high-weight code words. High-weight code words can be distinguished more easily (K äper, 2006).

#### 4.4.1.3 Simulation results for constraint length, $K=5$



**Figure 4.7: Turbo code performance for constraint length,  $K=5$**

Figure 4.7 only shown some generator polynomial of BER performance for constraint length  $K=5$ . From the graph, researcher can observe the best performance are given by the  $g(D)=[31,17]_{10}$ . From figure 4.8, this graph clearly to prove the phenomena is correct.



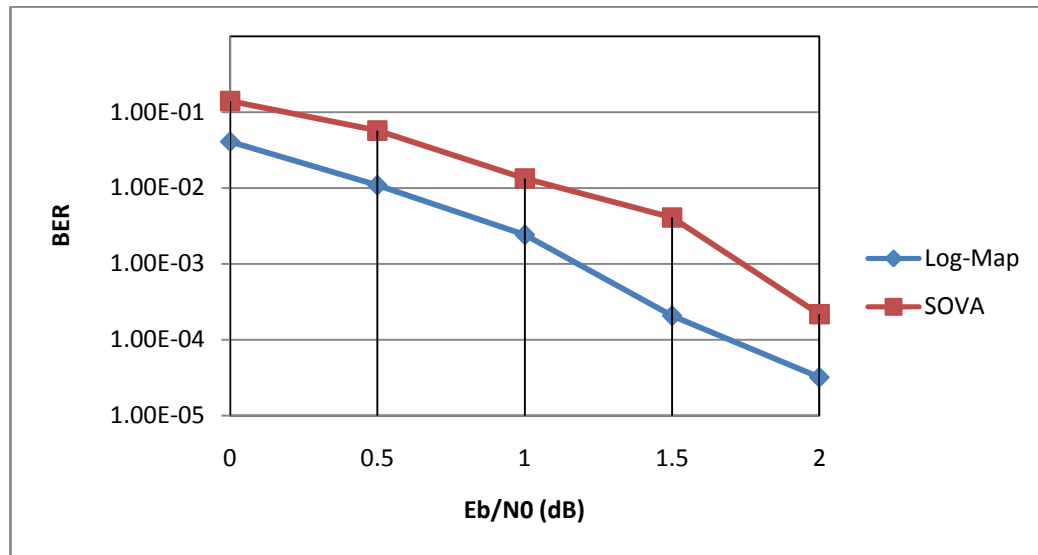
(a)  $g(D)=[31,17]$  (b)  $g(D)=[29,17]$

**Figure 4.8: The simulation result of No. of codewords and weight distribution from “Turbo Code Distance Spectrum Calculator”**

Figure 4.8 show that the  $g(D)=[31,17]_{10}$  has higher number of high weighted codewords. Researcher can observe that the figures 4.8(a) only at codeweight 9 just start have codewords, and the figure 4.8(b), researcher can observe at codeweight 8 start have codewords. So that, we can prove that the  $g(D)=[31,17]$  has fewer codewords with relatively low code weights than  $g(D)=[31,17]$ .

Hence, turbo code in constraint length  $K=5$  and  $g(D)=[31,17]$  were sought that would allow for more easier decoding compare with other generator polynomials for  $K=5$  because one way of making the task of the decoder easier is using a code and can be distinguished more easily with mostly high-weight code words (Käper, 2006).

#### 4.4.2 Effects of decoder algorithm



**Figure 4.9: Effects of decoder algorithm**

In this simulation, researcher choose the constraint length  $K=3$  and generator polynomial  $g(D) = [7,5]_{10}$ . The other parameter used in these simulations was the same as detailed in Section 4.2.1. From this graph, clearly show that the Log-Map decoder algorithm is better than SOVA decoder algorithm. The approximated coding gain obtained at BER  $10^{-3}$  is 0.3dB.

Log-Map given better performance than SOVA, but SOVA is less complex (C Chaikalis, 2000). Since for different applications (e.g. video, data transfer) and for different parameters (e.g. performance, complexity) either SOVA or Log-Map is optimum, a reconfigurable SOVA/Log-Map turbo decoder can be used, resulting in lower power consumption (Noras, 2002).

## **CHAPTER 5**

### **CONCLUSION**

#### **5.1 Conclusion**

From this research, the researcher has found that RSC turbo encoder generator polynomial gives the optimum performance under AWGN channel. From the research, high frame size will get better performance in turbo code system compare with the lower frame size. From the optimum performance generator polynomial compare with other generator polynomial the performance improvement around 0.1-0.3dB coding gain in the turbo code system. Generator polynomial is important in designing the Turbo code system where the performance of the system can be improved without any increase in hardware element or complexity of the system. The researcher also find out that in through simulation, Log-MAP decoding algorithm performance is better than SOVA decoding algorithm, which improvement around 0.4dB.

## **5.2 Future Work**

- In this research, the researcher using the symmetrical turbo code system to get simulation result but there also has an asymmetrical turbo code system. Study asymmetrical turbo code system will be necessary in future.
- Apply 3G standard in the simulation at study the performance
- Study Low-density Parity-check codes (LDPC). LDPC codes to be more and more attractive as enhancement of current (beyond 3G) or next generation wireless systems(4G)

## REFERENCES

- Balamuralithara, B. (2005). modifications in Turbo Coding System for Performance Enhancement. *Master Thesis* , 8-65.
- Berrou, C. (2010). *Code and Turbo code*. France: Springer.
- Berrou, C., Glavieux, A., & Thitimajshima, P. (1993). Near Shannon Limit Error Correcting Coding and Decoding: Turbo Codes. *IEEE International Conference on Communications (ICC)* , 1064-1070.
- C Chaikalis, M. S.-K. (2000). Reconfiguration between soft output Viterbi and log maximum a posteriori decoding algorithms. *IEEE* , MARCH.
- D.J.C. Mackay, R. (1997, march 6). Near Shannon limit performance of low density parity check codes.
- Du, K.-L., & M.N.S.SWAMY. (2010). *Wireless Communication Systems: From RF Subsystems to 4G Enabling Technologies*. UK: Cambridge University.
- Glavieux, A. (2007). *Channel Coding in Communication Networks*. United States: ISTE.
- K äsper, E. (2006). *Turbo Codes*.
- Korhonen, J. (2003). *Introduction to 3G Mobile Communication*. United States: Artech House.
- Langton, C. (2006). *Turbo Coding and MAP Decoding*.
- Malarić, K. (2010). *EMI protection for Communication Systems*. MA canton street Norwood , MA: Artech House.
- Noras, J. C. (2002). *Implementation of an improved reconfigurable SOVA/log-MAP turbo decoder in 3GPP*. *IEEE* , MAY

Proakis, J. (1995). *Digital Communications*. New York: McGraw-Hill.

Rekh, S., rani, D. S., & A. Shanmugam, D. (2000). Optimal Choice of Interleaver for Turbo codes. *IEEE*.

Scott, A. (2008). *RF Measurements for Cellular Phones and Wireless Data Systems*. Canada: John Wiley & Sons.

Singal, T. (2010). *Wireless Communications*. New Delhi: Tata McGraw Hill.

Yoon, S. (2009, March). *Turbo Code Distance Spectrum Calculator Version 1.2*. Retrieved from [www.eccpage.com/tcds\\_readme.txt](http://www.eccpage.com/tcds_readme.txt)

YuFei, W. (2005, April). *Turbo Code Demo*. Retrieved from [www.ee.vt.edu/~yufei/turbo.html](http://www.ee.vt.edu/~yufei/turbo.html)

## APPENDICES

### APPENDIX A: TURBO CODE DEMO CODING

```
% This script simulates the classical turbo encoding-decoding system.
% It simulates parallel concatenated convolutional codes.
% Two component rate 1/2 RSC (Recursive Systematic Convolutional) component
encoders are assumed.
% First encoder is terminated with tails bits. (Info + tail) bits are scrambled and
passed to
% the second encoder, while second encoder is left open without tail bits of itself.
%
% Random information bits are modulated into +1/-1, and transmitted through a
AWGN channel.
% Interleavers are randomly generated for each frame.
%
% Log-MAP algorithm without quantization or approximation is used.
% By making use of  $\ln(e^x + e^y) = \max(x, y) + \ln(1 + e^{(-\max(x, y))})$ ,
% the Log-MAP can be simplified with a look-up table for the correction function.
% If use approximation  $\ln(e^x + e^y) = \max(x, y)$ , it becomes MAX-Log-MAP.
%
% Copyright Nov 1998, Yufei Wu
% MPRG lab, Virginia Tech.
% for academic use only

clear all
```



```

% Write display messages to a text file
diary turbo_logmap.txt

% Choose decoding algorithm
dec_alg = input(' Please enter the decoding algorithm. (0:Log-MAP, 1:SOVA)
default 0 ');
if isempty(dec_alg)
    dec_alg = 0;
end

% Frame size
L_total = input(' Please enter the frame size (= info + tail, default: 500) ');
if isempty(L_total)
    L_total = 500;      % information bits plus tail bits
end

% Code generator
g = input(' Please enter code generator: ( default: g = [1 1 1;1 0 1] ');
if isempty(g)
    g = [1 1 1;1 0 1];
end
%g = [1 1 0 1; 1 1 1 1];
%g = [1 1 1 1 1; 1 0 0 0 1];

[n,K] = size(g);
m = K - 1;
nstates = 2^m;

%puncture = 0, puncturing into rate 1/2;
%puncture = 1, no puncturing
puncture = input(' Please choose punctured / unpunctured (0/1): default 1 ');
if isempty(puncture)
    puncture = 1;
end

```

```

% Code rate
rate = 1/(2+puncture);

% Fading amplitude; a=1 in AWGN channel
a = 1;

% Number of iterations
niter = input(' Please enter number of iterations for each frame: default 5 ');
if isempty(niter)
    niter = 5;
end

% Number of frame errors to count as a stop criterior
ferrlim = input(' Please enter number of frame errors to terminate: default 7 ');
if isempty(ferrlim)
    ferrlim = 7;
end

EbN0db = input(' Please enter Eb/N0 in dB : default [2.0] ');
if isempty(EbN0db)
    EbN0db = [2.0];
end

fprintf('\n\n-----\n');
if dec_alg == 0
    fprintf('=== Log-MAP decoder === \n');
else
    fprintf('=== SOVA decoder === \n');
end

fprintf(' Frame size = %6d\n',L_total);
fprintf(' code generator: \n');
for i = 1:n
    for j = 1:K
        fprintf(' %6d', g(i,j));
    end
end

```

```

    end
    fprintf('\n');
end
if puncture==0
    fprintf(' Punctured, code rate = 1/2 \n');
else
    fprintf(' Unpunctured, code rate = 1/3 \n');
end
fprintf(' iteration number = %6d\n', niter);
fprintf(' terminate frame errors = %6d\n', ferrlim);
fprintf(' Eb / N0 (dB) = ');
for i = 1:length(EbN0db)
    fprintf('%10.2f',EbN0db(i));
end
fprintf('\n-----\n\n');

fprintf('+++ Please be patient. Wait a while to get the result. +++\n');

for nEN = 1:length(EbN0db)
    en = 10^(EbN0db(nEN)/10);    % convert Eb/N0 from unit db to normal numbers
    L_c = 4*a*en*rate; % reliability value of the channel
    sigma = 1/sqrt(2*rate*en); % standard deviation of AWGN noise

    % Clear bit error counter and frame error counter
    errs(nEN,1:niter) = zeros(1,niter);
    nferr(nEN,1:niter) = zeros(1,niter);

    nframe = 0; % clear counter of transmitted frames
    while nferr(nEN, niter)<ferrlim
        nframe = nframe + 1;
        x = round(rand(1, L_total-m)); % info. bits
        [temp, alpha] = sort(rand(1,L_total)); % random interleaver mapping
        en_output = encoderm( x, g, alpha, puncture ); % encoder output (+1/-1)
    end
end

```

```

r = en_output+sigma*randn(1,L_total*(2+puncture)); % received bits
yk = demultiplex(r,alpha,puncture); % demultiplex to get input for decoder 1 and
2

% Scale the received bits
rec_s = 0.5*L_c*yk;

% Initialize extrinsic information
L_e(1:L_total) = zeros(1,L_total);

for iter = 1:niter
% Decoder one
    L_a(alpha) = L_e; % a priori info.
    if dec_alg == 0
        L_all = logmapo(rec_s(1,:), g, L_a, 1); % complete info.
    else
        L_all = sova0(rec_s(1,:), g, L_a, 1); % complete info.
    end
    L_e = L_all - 2*rec_s(1,1:2:2*L_total) - L_a; % extrinsic info.

% Decoder two
    L_a = L_e(alpha); % a priori info.
    if dec_alg == 0
        L_all = logmapo(rec_s(2,:), g, L_a, 2); % complete info.
    else
        L_all = sova0(rec_s(2,:), g, L_a, 2); % complete info.
    end
    L_e = L_all - 2*rec_s(2,1:2:2*L_total) - L_a; % extrinsic info.

% Estimate the info. bits
    xhat(alpha) = (sign(L_all)+1)/2;

% Number of bit errors in current iteration
    err(iter) = length(find(xhat(1:L_total-m)~=x));

```

```

% Count frame errors for the current iteration
    if err(iter)>0
        nferr(nEN,iter) = nferr(nEN,iter)+1;
    end
end%iter

% Total number of bit errors for all iterations
    errs(nEN,1:niter) = errs(nEN,1:niter) + err(1:niter);

    if rem(nframe,3)==0 | nferr(nEN, niter)==ferrlim
% Bit error rate
        ber(nEN,1:niter) = errs(nEN,1:niter)/nframe/(L_total-m);
% Frame error rate
        fer(nEN,1:niter) = nferr(nEN,1:niter)/nframe;

% Display intermediate results in process
        fprintf('*****      Eb/N0      =      %5.2f      db      *****\n',
EbN0db(nEN));
        fprintf('Frame size = %d, rate 1/%d. \n', L_total, 2+puncture);
        fprintf('%d frames transmitted, %d frames in error.\n', nframe, nferr(nEN,
niter));
        fprintf('Bit Error Rate (from iteration 1 to iteration %d):\n', niter);
        for i=1:niter
            fprintf('%8.4e  ', ber(nEN,i));
        end
        fprintf('\n');
        fprintf('Frame Error Rate (from iteration 1 to iteration %d):\n', niter);
        for i=1:niter
            fprintf('%8.4e  ', fer(nEN,i));
        end
        fprintf('\n');
        fprintf('*****\n\n');

% Save intermediate results

```

```
        save turbo_sys_demo EbN0db ber fer  
    end  
  
    end          %while  
end            %nEN  
  
diary off
```