

ITERATIVE DECODING OF BLOCK CODES

A THESIS SUBMITTED TO  
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES  
OF  
ÇANKAYA UNIVERSITY

BY

FATİH GENÇ

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR  
THE DEGREE OF MASTER OF SCIENCE  
IN  
ELECTRONIC AND COMMUNICATION  
ENGINEERING

SEPTEMBER 2010

Title of the Thesis : **Iterative Decoding of Block Codes**

Submitted by **Fatih Genç**

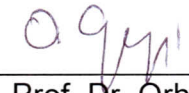
Approval of the Graduate School of Natural and Applied Sciences, Çankaya University

  
Prof. Dr. Taner ALTUNOK  
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

  
Assoc. Prof. Dr. Celal Zaim ÇİL  
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

  
Assist. Prof. Dr. Orhan GAZİ  
Supervisor

**Examination Date** : 01.09.2010

**Examining Committee Members**

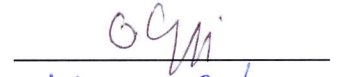
Assoc. Prof. Dr. Celal Zaim ÇİL

(Çankaya Univ.)



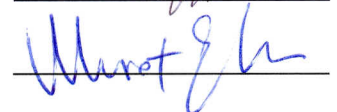
Assist. Prof. Dr. Orhan GAZİ

(Çankaya Univ.)



Assist. Prof. Dr Murat EFE

(Ankara Univ.)



## STATEMENT OF NON-PLAGIARISM PAGE

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : Fatih Genç

Signature : 

Date : 15.10.2010

## **ABSTRACT**

### ITERATIVE DECODING OF BLOCK CODES

Genç, Fatih

M.Sc., Department of Electronic and Communication Engineering

Supervisor : Assist. Prof. Dr. Orhan Gazi

September 2010, 57 Pages

Error control coding is vital for digital communication. With the advent of turbo codes a huge interest on iterative decoding aroused recently. Although iterative decoding reduces bit error rate (BER) significantly, it brings new challenges such as increased complexity and large decoding delays. Turbo like codes can be constructed either using convolutional codes or block codes. In this thesis we construct classical turbo codes with both convolutional codes and block Bose Chaudhuri and Hocquenghem (BCH) codes, and simulate their performances and discuss their advantages and disadvantages.

**Key Words:** BCH Codes, Iterative Decoding, Turbo Code

## ÖZ

### BLOK KODLARDA DÖNGÜLÜ KODÇÖZME

Genç, Fatih

Yükseklisans, Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Tez Yöneticisi : Yrd. Doç. Dr. Orhan Gazi

Eylül 2010, 57 Sayfa

Hata kontrollü kodlama dijital iletişim için hayati öneme sahiptir. Turbo kodların gelişi iteratif çözümlene üzerine büyük bir ilgi uyandırdı. İteratif çözümlene BER'i önemli ölçüde azaltmasına rağmen, artan karmaşıklık ve çözümlene gecikmeleri gibi yeni sorunlar oluşturur. Turbo kodlar katlamalı kodlar veya blok kodlar kullanılarak yapılabilir. Bu tez çalışmasında katlamalı kodlar ve blok BCH kodları kullanılarak klasik turbo kodları inşası ve onların performansları, avantaj ve dezavantajları tartışılmıştır.

**Anahtar Kelimeler:** BCH Kodlar, Döngülü Kod Çözme, Turbo Kod

## **ACKNOWLEDGMENTS**

I am grateful to my supervisor Assist. Prof. Dr. Orhan Gazi for his patience, guidance, encouragement and intellectual support.

I also wish to thank to Assoc. Prof. Dr. Celal Zaim il for his criticisms and comments. He shed light upon my intellectual map for my further studies.

I owe too much to my family, especially; to my grandmother Sultan Uzuner for the life energy she gives to me and to my mother Leman Gen, my father Dr. Ramazan Gen, my brother Dr. Serhat Bahadır Gen, for their emotional support.

## TABLE OF CONTENTS

STATEMENT OF NON-PLAGIARISM .....	iii
ABSTRACT .....	iiiv
ÖZ.....	v
ACKNOWLEDGMENTS .....	vi
TABLE OF CONTENTS.....	vii
LIST OF TABLES.....	ix
LIST OF FIGURES .....	x
CHAPTERS :	
1.INTRODUCTION .....	1
1.1 Coding .....	1
1.2 The AWGN Channel.....	4
1.2 Modulation .....	5
1.4 Entropy and Channel Capacity .....	5
1.5 Thesis Outline.....	6
2.CONVOLUTIONAL CODES.....	7
2.1 Convolutuional Encoder.....	7
2.2 Decoding of Convolotuional Codes .....	9
2.2.1 Hard Decision Decoding Algoritihm.....	9
2.2.2 Soft – Decision Decoding.....	13

2.2.3 Map Decoding Algorithm .....	15
2.2.3.1 Forward Recursive Calculation of the $\alpha_k(s)$ Values .....	21
2.2.3.2 Backward Recursive Calculation of the $\beta_k(s)$ Values .....	21
2.2.3.3 Calculation of the $\gamma_k(s', s)$ Values .....	22
2.2.3.4 Summary of the Map Algorithm.....	23
2.2.4 LOG-MAP Algorithm.....	24
2.3 Turbo Code.....	27
2.3.1 Turbo Encoder.....	28
2.3.2 Iterative Decoding of Turbo Codes .....	30
2.4 Simulation Results of Turbo Codes, MAP, Soft-Hard Decision Algorithms.....	35
3. BLOCK CODE .....	37
3.1 BCH (31, 21, 5) Block Code Encoder .....	38
3.2 Trellis Decoding of Block Codes.....	49
3.3 Simulation Results.....	53
4. CONCLUSION.....	55
REFERENCES .....	57
APPENDICES	
A. State of the BCH (31,21) Code .....	A1
B. Turbo Code .....	B1
CV .....	C1



## LIST OF TABLES

Table 3.1 32 Field Elements .....	40
Table 3.2 Conjugate Elements in $GF(2^5)$ .....	43
Table 3.3 Operation Steps of BCH (7, 4, 3) Encoder.....	48

## LIST OF FIGURES

Figure 1.1 Model of Digital Communication System .....	2
Figure 1.2 Gaussian Channel .....	4
Figure 2.1 A Rate $\frac{1}{2}$ Convolutional Encoder .....	7
Figure 2.2 State Transition Diagram For the Encoder of Figure 2.1.....	8
Figure 2.3 Trellis Diagram for the Encoder of Fig 2.1 .....	9
Figure 2.4 Hamming Distance Calculation of the VA .....	11
Figure 2.5 Branch Metrics of the VA .....	12
Figure 2.6 Winning Path of the VA .....	13
Figure 2.7 Possible Transitions.....	17
Figure 2.8 MAP Decoder Trellis.....	19
Figure 2.9 Recursive Calculation of $\alpha_k(0)$ and $\beta_k(0)$ .....	20
Figure 2.10 Summary of the MAP Algorithm.....	24
Figure 2.11 Trellis Diagram for MAP.....	25
Figure 2.12 Turbo Encoder .....	29
Figure 2.13 Soft-Input / Soft-Output Decoder .....	30

Figure 2.14 Iterative Decoding Procedures with Two Soft-Input / Soft-Output Decoder .....	31
Figure 2.15 Turbo Decoder Schematic .....	33
Figure 2.16 Simulation Result of Turbo Code .....	35
Figure 2.17 Turbo Coding BER Performance Using Different Number of Iterations .....	36
Figure 2.18 Effect of Frame Length on BER Performance of One-Third Rate Turbo Coding .....	36
Figure 3.1 Systematic Encoder for BCH Codes Having $(n-k)$ Shift-Register with $(n-k)$ Cells .....	46
Figure 3.2 Systematic Encoder for BCH(31,21) Code Having $n-k = 10$ Register Stages .....	47
Figure 3.3 Systematic Encoder for BCH (7,4,3) .....	47
Figure 3.4 Binary Representations Of The Encoded Data Bits and Code Bits .....	48
Figure 3.5 State Transition Diagram for BCH(7,4,3) .....	49
Figure 3.6 State Diagram for the BCH(7,4,3) Code .....	50
Figure 3.7 Trellis Diagram for the BCH(7,4,3) Code .....	51
Figure 3.8 BCH Block Turbo Encoder Schematic .....	52
Figure 3.9 BCH Block Turbo Decoder Schematic .....	52
Figure 3.10 Simulation Results of BCH(31,21) Block Turbo Code .....	53
Figure 3.11 Performance of Different Number of Iterations in Turbo BCH(31,21) Code .....	54

## CHAPTER 1

### INTRODUCTION

#### 1.1 Coding

Figure 1.1 illustrates the basic elements found in digital communication which connect a data source to a data user through a channel. The nature of the information source may be either analog or digital, depending on the application. An analog signal; however, must be digitalized before being used in a digital communication system. A digital signal is discrete in time and uses only a finite alphabet. Typically, the data is represented by a sequence of binary digits (*bits*). One of the tasks in coding theory is to detect, or even correct errors. Usually, coding is defined as the source coding and the channel coding. The source-coding theorem is one of the most important theorems introduced by Shannon. The source-coding establishes a fundamental limit on the rate at which the output of an information source can be compressed without causing a large error probability [1, 2, 3]. An example of source coding is the *ASCII* code, which converts each character to a byte of 8 bits. In this work source coding is not concerned. In order to understand the role of error control coding, a model of a general communication system is presented, as shown in Fig.1.1.

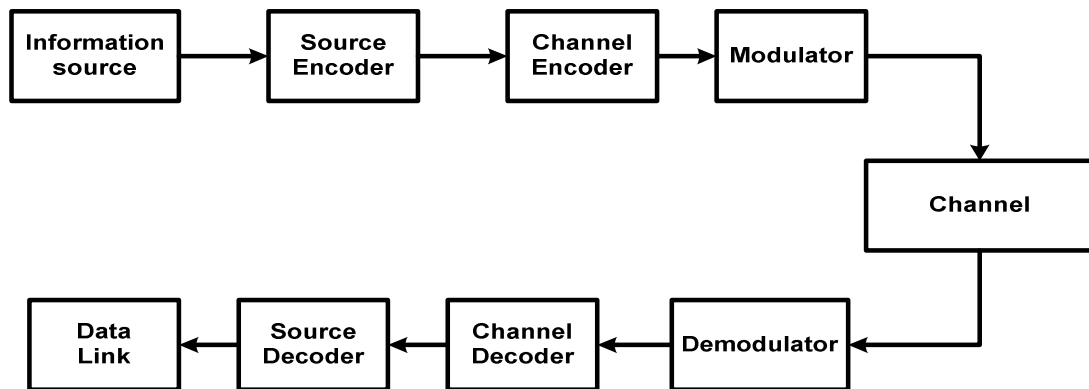


Figure 1.1 Model of Digital Communication System

The task of the transmitter in such a system is to transform the information into a form that can withstand the effect of noise over the transmission medium. An information source generates messages bearing information to be transmitted. The messages can be words, code symbols etc. The output of the information source is converted sequence of symbols from a certain alphabet. The most often binary symbols are transmitted. In general, the output of the information source is not suitable for transmission as it might contain too much redundancy. Due to the efficiency reasons, the source encoder is designed to convert the source output sequence into a sequence of binary digits with minimum redundancy. The number of bits  $r_b$  generated by the source encoder per second is called the data rate. The channel impairments cause errors in the received signal. The channel encoder is incorporated in the system to add redundancy to the information sequence. This redundancy is used to reduce transmission errors. The channel encoder assigns to each message of  $k$  symbols a longer message of  $n$  digits called a codeword. A good error control code generates codewords which are different as possible from one another. This makes the communication system less defenseless to channel errors. Each code is characterized by the ratio  $R = \frac{k}{n} < 1$  called the code rate. The data rate at the output of the

channel encoder is  $r_c = \frac{r_b}{R}$  bps. The primary goal of error control coding is to

maximize the reliability of transmission within the constraints of signal power, system bandwidth and complexity of the circuitry which is achieved by introducing structured redundancy into transmitted signals. This usually results in a lowered data transmission rate of increased channel bandwidth. The output signal of the channel encoder is not normally suitable for transmission. The modulator enables signal transmission over a channel. In the receiver, the demodulator typically generates a binary or analog sequence at its output as the best estimates of the transmitted codewords. The channel decoder makes estimates of the actually transmitted message. The decoder process is based on the encoder rule and the characteristics of the channel. The goal of the decoder is to minimize the effect of channel noise. By proper design of the transmitter - receiver system, it would be possible to reduce or remove the effects of attenuation and distortion and to minimize the noise effects. The impact of noise cannot be totally removed. If the demodulator makes hard decisions, the output is a binary sequence. The subsequent channel decoding process is called Hard-decision decoding. Hard decisions in the demodulator result in some irreversible information loss. An alternative is to quantize the demodulator output to more than two levels or take samples of the analog received baseband signal and pass it to the channel decoder. The subsequent decoding process is called soft decision decoding [4, 5].

The two most frequently used types of codes are block and convolutional codes. In block codes each encoding operation depends on the current input message and is independent on previous encodings. That is, the encoder has no memory of history of past encodings. In contrast, for a convolutional code, each encoder output sequence depends not only on the current input message, but also on a number of past message blocks.

## 1.2 The AWGN Channel

The model that is used in this thesis is the most commonly used model, the *Additive White Gaussian Noise (AWGN)* channel. It is a very simple memoryless channel. The received signal  $Y$  is described by [1]

$$Y = X + Z \quad (1.1)$$

The noise  $Z$  is assumed to be independent of the signal  $X$ . We first analyze a simple suboptimal way to use this channel. Assume that we want to send bit '1' over the channel. Given the power constraint, the best that we can do is to send one of two levels,  $+\sqrt{P}$  or  $-\sqrt{P}$ . The receiver looks at the corresponding  $Y$  received and tries to decide which of the two levels were sent. Assuming that both signals levels are equally likely (this would be the case if we wish to send exactly 1 bit of information), the optimum decoding rule is to decide that '1' was sent if  $Y > 0$  and decide '-1' was sent if  $Y < 0$ .

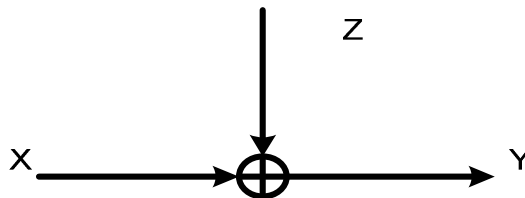


Figure 1.2 Gaussian Channel

The noise  $Z$  is a stationary random process with *Gaussian Probability distribution* given as

$$P = \frac{1}{\sigma \sqrt{2\pi}} \exp\left(\frac{-x^2}{2\sigma^2}\right) \quad , \quad (1.2)$$

Where mean is  $\gamma_n = 0$  and average power of noise is  $\sigma^2$ .

### 1.3 Modulation

The modulation method employed in this thesis is *Binary Phase Shift Keying (BPSK)*. This is a *Memoryless Modulation Technique*. The binary digits '1' and '0' are modulated as '1' and '-1'. In general for binary string  $b$  transmitter signal after modulation operation is  $s = 2b - 1$ .

### 1.4 Entropy and Channel Capacity

Information sources generate any of a set of  $M$  different symbols, which are considered as representatives of a discrete random variable  $X$  that adopts any value in the range  $A = \{x_1, x_2, \dots, x_m\}$ . Each symbol  $X_i$  has the probability  $P_i$  of being emitted and contains information  $I_i$ . The symbol probabilities must be in agreement with the fact that at least one of them will be emitted, which means that,

$$\sum_{i=1}^m P_i = 1 \quad . \quad (1.3)$$

The source symbol probability distribution is stationary, and the symbols are independent and transmitted at a rate of  $r$  symbol per second. This description corresponds to a *Discrete Memoryless Source (DMS)* [1]. Each symbol contains the information  $I_i$  so that the set  $\{I_1 I_2 \dots I_m\}$  can be seen as a discrete random variable with average information

$$H_b(x) = \sum_{i=1}^m P_i I_i = \sum_{i=1}^m P_i \log_b \left( \frac{1}{P_i} \right) \quad (1.4)$$

The function so defined is called the entropy of the source. When base 2 is used, the entropy is measured in bits per symbol:



$$H(x) = \sum_{i=1}^m P_i I_i = \sum_{i=1}^n P_i \log_2 \left( \frac{1}{P_i} \right) \quad \text{bits per symbol} \quad (1.5)$$

The channel capacity of a discrete memoryless channel is equal to

$$c_S = \underset{P(x)}{\text{MAX}} I(x, y) \quad \text{bit per symbol} \quad (1.6)$$

Shannon's "*Noisy Coding Theorem*" states that every channel has a capacity  $C$ , which is the highest rate  $C$  in bits per channel used at which reliable communication is possible. Shannon showed that error free communication is possible at transmission rates below channel capacity employing channel codes. Ever since Shannon proved his noisy coding theorem, the construction of practical capacity-achieving schemes has been the goal of coding theory. The classical approaches to this problem included algebraic block codes and convolutional codes. The field was revolutionized by the introduction of turbo codes by *Berrou, Glavieux* and *Thitimajshima* in 1993. The performance of *Turbo Code* is much closer to capacity than that of any previous codes and has lower complexity.

## 1.5 Thesis Outline

In Chapter 2, we introduce the convolutional codes and some of the decoding methods. First we explain the *Trellis* structure. We consider the *Hard-Decision Viterbi Algorithm (HVA)*, *soft-decision Algorithm (SOVA)* and *Map algorithm-log Map algorithm* and we present the *turbo Convolutional code*. Simulation results are shown and conclusions are added.

In chapter 3, we begin with the *BCH Block code* and present *turbo BCH* codes considering two trellis decoding methods, which are the *Maximum A-Posteriori (MAP)* and the *soft-output Viterbi Algorithm (SOVA)* and simulation results, are given.

Finally conclusions are given in Chapter 4.

## CHAPTER 2

### CONVOLUTIONAL CODES

#### 2.1 Convolutional Encoders

An important technique in error-control coding is that convolutional coding. *Convolutional* codes were invented in 1954 by *P. Elias* [6]. They constitute a family of error correcting codes. In this type of coding the encoder output is not in block form, but is in the form of continuous bit stream. The convolutional encoding operation involves past and present bits and it is a continuous operation. The convolutional encoding operation can be performed using *Finite State Machines (FSMs)*.

In this thesis we will use the convolutional encoder shown in Fig. 2.1. Let  $k$  and  $n$  denote the number data and encode bits [7].

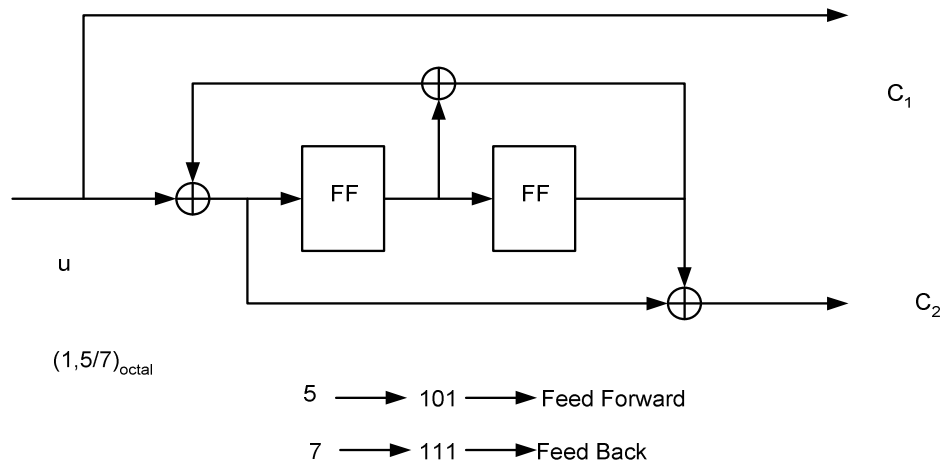


Figure 2.1 A Rate  $\frac{1}{2}$  Convolutional Encoder

Then the rate of the code is defined as;

$$R_c = \frac{k}{n} \quad . \quad (2.1)$$

7

For the convolutional encoder in Fig.2.1  $k=1$ ,  $n=2$   $L=3$ . If the number of cells in convolutional encoder register is  $\gamma$ , the constraint length is  $L$ . Therefore, the convolutional encoder is defined as  $L=\gamma+1$ . The total number of states in Fig.2.1 is  $2^{(L-1)} = 4$ . The convolutional encoder in Fig.2.1 can be represented by a state transition diagram. Fig.2.2 shows the state transition diagram for the convolutional encoder in Fig.2.1. In this state-transition diagram, each state of the convolutional encoder is represented by a box and transitions between states are denoted by lines connecting these boxes.

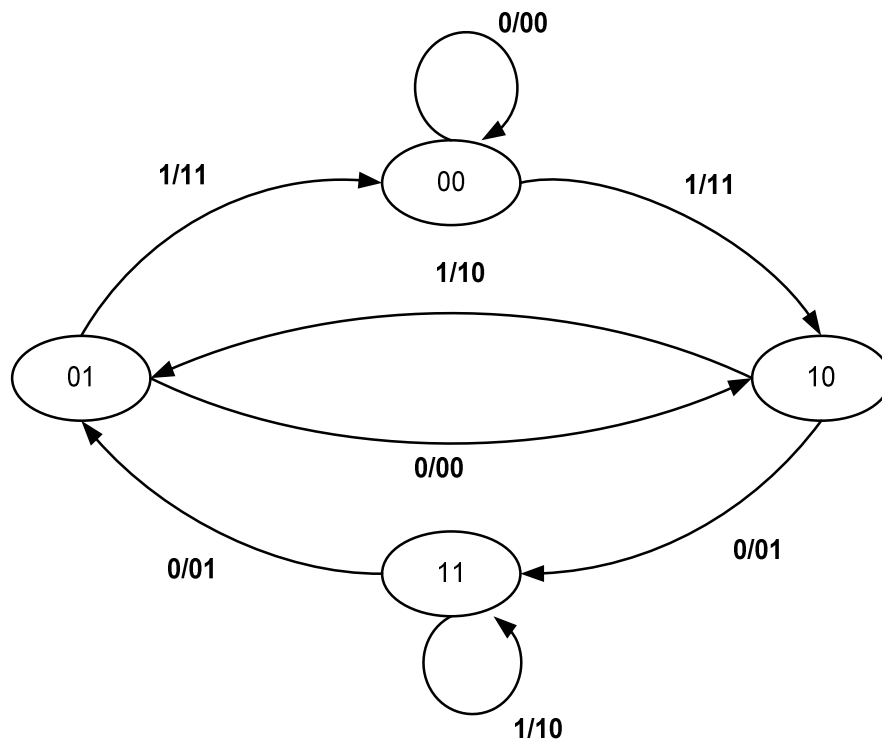


Figure 2.2 State Transition Diagram for the Encoder of Fig 2.1

Convolutional codes can also be described using trellis diagrams. The trellis diagram shows the change in states as time passes. Usually horizontal axis is used for time and vertical axis is used for state transitions.

### Example 2.1

Using state transition diagram 10101 can be encoded as 11 01 10 01 11. For the encoding process first we start with the 00 state. Notice that when the input bit is 0 the encoded bits are 00 which is shown like 0/00. After that when the input bit is 1 the encoded bits are 11 and the state becomes to 10 state.

Fig.2.3 shows the trellis diagram for the convolutional encoder which was shown in Fig.2.1.

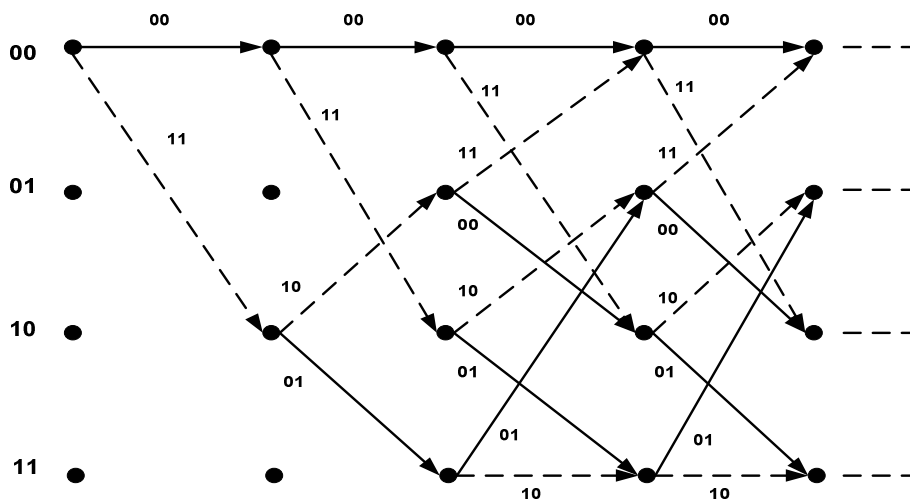


Figure 2.3 Trellis Diagram for the Encoder of Fig 2.1

The trellis diagram starts with the 00 state and the all states merge with the proper next states. The dashed line denotes the input bits which are 0 and the other flat lines denote the input bits 1. The other bits are shown at the top of the all lines which are the output encoded bits.

## 2.2 Decoding of Convolutional Codes

### 2.2.1 Hard Decision Decoding Algorithm

The *Viterbi Algorithm (VA)* performs *maximum likelihood* decoding. It is applied to the trellis of a convolutional code whose properties are conveniently used to implement this algorithm. Such that the code word generated from path in hard-decision decoding the trellis technique denoted

by  $c$ , is a path through the trellis is chosen at minimum *Hamming* distance from the quantized received sequence  $y$ . The *Hamming* distance between  $c$  and  $y$  is therefore.

$$d(c, y) = \sum_{i=1}^m d(c_i, y_i) \quad (2.2)$$

In the other word  $c_i$  are the encoded bits and the  $y_i$  are the received bits. The binary distance was found with  $c_i$  and  $y_i$  by the *Hamming* distance calculation.

*Viterbi* algorithm can be summarized as with [8]:

- 1- Find the *Hamming* distance of the  $i^{th}$  subsequence of the received sequence to all branches which are connecting  $i^{th}$  stage states to the  $(i+1)$  stage states shown in Fig. 2.4. Here  $i=1,2\dots n$  and  $n$  is the number of the input bits.
- 2- Add these distances to the metrics of the  $i^{th}$  stage states to obtain the metric candidates for the  $(i+1)$  stage states. For each state of the  $(i+1)$  stage there are 2 metrics candidate.
- 3- For each state at the  $(i+1)$  stage, choose the minimum one of the metric candidates. Label the branch corresponding to this minimum value as the survivor and assign the minimum of the metric candidates as the metrics of the  $(i+1)$  stage states which is shown in Fig. 2.5.
- 4- Starting with the minimum value state at the final stage. Go back through the trellis along the survivors to reach the initial all-zero state. This path is the optimal path and is called *Wining Path*.

### Example 2.2

Consider the convolutional code of Fig.2.2 whose trellis is seen in Fig.2.3 for the received sequence is  $S_r = 1101110111$ .

In example 2.1 the message sequence 10101 is encoded to 11 01 10 01 11. We will use Trellis Diagram with *Viterbi* Algorithm to decode the encoded 11 01 10 01 11 sequence.

The first step in the application of this algorithm is to determine the *Hamming* distance between the received sequences. This is shown in Fig 2.4 all the codewords generated by the Trellis Diagram.

Message Sequence : 1 0 1 0 1

Code Sequence : 11 01 10 01 11

Received Sequence : 11 01 11 01 11

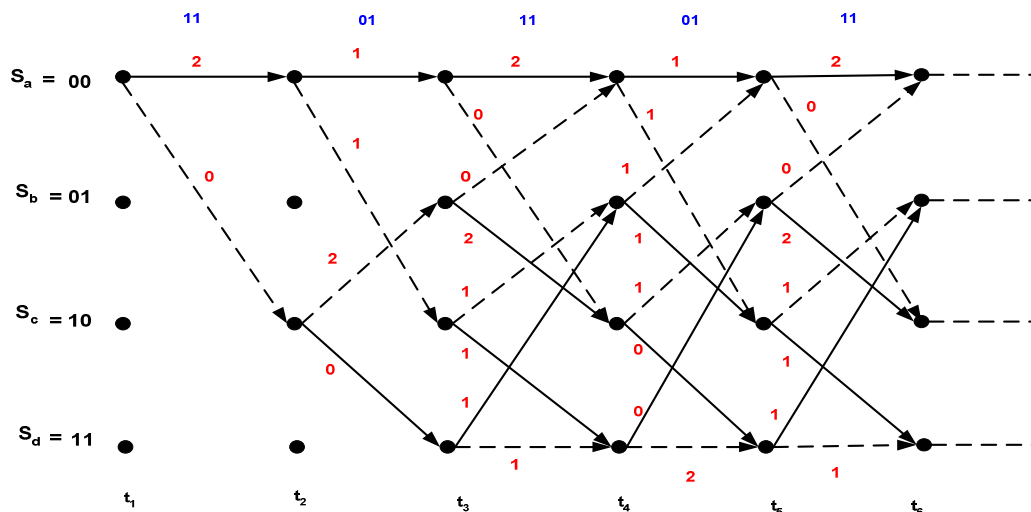


Figure 2.4 Hamming Distance Calculation of the VA

At each shape of the decoding operations from the all-zero state, we compute the *Hamming* distance of the received bit pairs. For example, for the first two-bits “11”, the associated *Hamming* distances are 2 and 0 with respect to both the “00” and “11”. We also note these *Hamming* distances in the trellis diagram of Fig.2.4. These Hamming distances are known in the context of *Viterbi* decoding as the *branch metric*. The power of the *Viterbi* decoding algorithm accrues from the fact that it carries out maximum likelihood sequence estimation. The branch metrics will be accumulated over a number of consecutive trellis stages before a decision as to the most likely encoder path and information sequence can be released. Proceeding to the

next received two-bit symbol namely “01” that is the *Hamming* distance between the encoded symbols of all four legitimate paths and the received symbol is computed. These distances yield the new branch metrics associated with the second trellis stage. By now the encoded symbols of two original input bits have been received and this is why there are now four possible trellis states which the decoder may reside. The branch metrics computed for these four legitimate transitions from top to bottom are 1, 1, 2 and 0. These are now added to the previous branch metrics of 1 in order to generate the *path metrics* of 3, 2, 3 and 0 which were accumulated *Hamming* distance in Fig2.5.

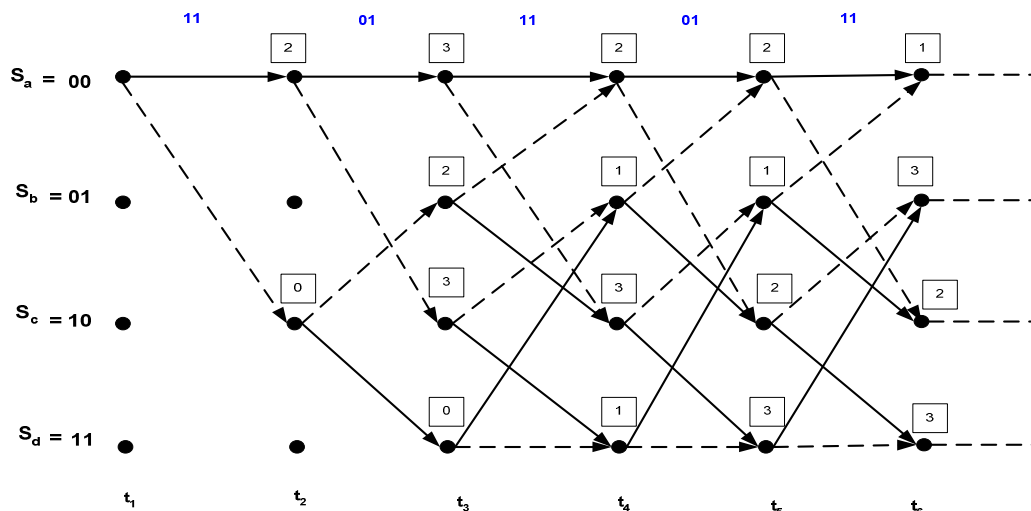


Figure 2.5 Branch Metrics of the VA

A low *Hamming* distance indicates a high similarity between the received sequence and the encoded sequence. If we continue at trellis stage  $t_3$  the received sequence of “11” is compared to the four legitimate two-bit encoded symbols. Notice that there is an error bit that changes the Hamming distance of each path so the lower metric will always remain the more likely encoder path. This is respected in Fig.2.6 by referring to the path exhibiting the lower metric as the *survivor path*.

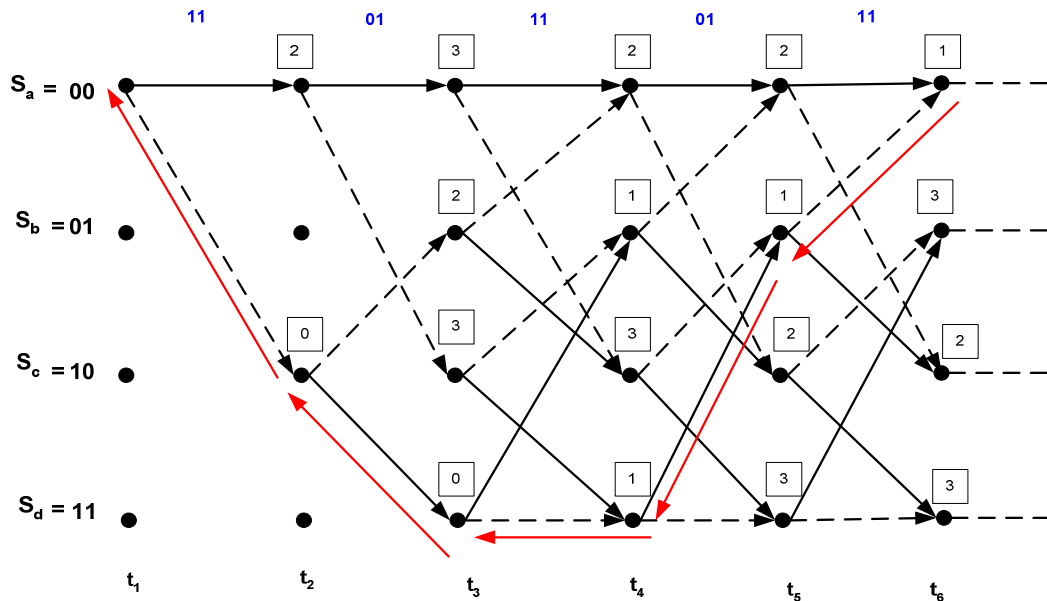


Figure 2.6 Winning Path of the VA

In our example the received bit sequence does not contain any more transmission errors, and so it is plausible that the *winning path* remains the one at the top of Fig.2.6. The corresponding winning path was drawn in bold in the Fig.2.6.

### 2.2.2 Soft – Decision Decoding

In *Hard* decision decoding operation *Hamming* distance are used. On the other hand in *Soft-decision* decoding operation the probability values are used. This is the main differences of *Hard* and *Soft* decision decoding. Let's first review *maximum likelihood* criteria. It can be stated as follows [9,10]:

$$\begin{aligned}
 P(s_1/y) \geq P(s_0/y) & \text{ The decoder decides for hypothesis } H_1 \\
 P(s_0/y) \geq P(s_1/y) & \text{ The decoder decides for hypothesis } H_0
 \end{aligned}
 \tag{2.3}$$

Hypothesis  $H_1$  corresponds to the transmission of symbol '1', and hypothesis  $H_0$  corresponds to the transmission of symbol '0'. Equation (2.3) can be written as;



$$\begin{aligned}
P(y/s_1)P(s_1) &\geq P(y/s_0)P(s_0) \\
P(y/s_0)P(s_0) &\geq P(y/s_1)P(s_1)
\end{aligned} \tag{2.4}$$

Where for the first set the decoder decides for hypothesis  $H_1$  and for the second set the decoder decides for  $H_0$ . If the transmitted symbols are equally likely, then

$$\frac{P(y/s_1)}{P(y/s_0)} > 1 \text{ The decoder decides for hypothesis } H_1$$

$$\frac{P(y/s_1)}{P(y/s_0)} < 1 \text{ The decoder decides for hypothesis } H_0 \tag{2.5}$$

Assume that the transmitted signals are  $s_1$  and  $s_2$ . The received signals are  $y_1 = s_1 + n$  and  $y_2 = s_0 + n$ . If the transmission is over an AWGN channel, the probability density functions is given of

$$P(y|s) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2} \frac{(y-s_i)^2}{\sigma^2}} \quad i = 0,1 \tag{2.6}$$

Using eqn.(2.4) and (2.6) the likelihood ratio can also be expressed in terms of these probability density functions for each of the transmitted symbols as ;

$$\frac{\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2} \left(\frac{y-s_1}{\sigma}\right)^2}}{\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2} \left(\frac{y-s_0}{\sigma}\right)^2}} \underset{H_0}{>} \underset{H_1}{<} \frac{P(s_0)}{P(s_1)} \tag{2.7}$$

The decoder decides for hypothesis  $H_1$  and  $H_0$  which is simplified as;

$$e^{-\frac{1}{2}\left[\left(\frac{y-b_1}{\sigma}\right)^2 - \left(\frac{y-b_0}{\sigma}\right)^2\right]} \underset{H_0}{\overset{H_1}{>}} \frac{P(s_0)}{P(s_1)} \quad (2.8)$$

If natural logarithm of both sides of eqn.(2.7) is taken the decision criteria to  $(y-b_1)^2 \underset{H_0}{\overset{H_1}{>}} (y-b_0)^2$ . This means contrary to Hamming distance in *Hard decision decoding* and *Euclidean* distance are used in Soft decision decoding.

### 2.2.3 Map Decoding Algorithm

In hard decision algorithm estimated bits for transmitted sequence is given to the channel decoder. Thus, any information about the reliability of the received sequence is lost. In soft decision decoding algorithm the probability values for transmitted sequence are used at decoder. Soft decision decoding algorithm achieves better performance than hard decision decoding. In this chapter we introduce *Maximum A Posteriori (MAP)* decoder algorithm that both accepts and delivers soft values. The *Log Likelihood Ratio (LLR)* of a data bit  $u_k$  is denoted as  $L(u_k)$  and is defined to be merely the log of the ratio of the probability of the bit taking its two possible values [11], i.e.,

$$L(u_k) \triangleq \ln \left( \frac{P(u_k = +1)}{P(u_k = -1)} \right) \quad (2.9)$$

The *BPSK* modulated values for the bits  $u_k$  are taken +1 and -1. This approach slightly simplifies the mathematics in the derivations. The sign of the *LLR*,  $L(u_k)$  will indicate whether the bit is more likely to be +1 or -1 and the magnitude of the  $L(u_k)$  gives an indication of how likely it is that the sign of  $L(u_k)$  gives the correct value  $u_k$ . When  $L(u_k) \approx 0$ , It results in  $P(u_k = +1) \approx P(u_k = -1) \approx 0.5$ , and we can not be certain about the value of  $u_k$ . Conversely, when  $L(u_k) \gg 0$ , we get  $P(u_k = +1) \gg P(u_k = -1)$  and we can be

almost certain that  $u_k = +1$ . As well as the  $L(u_k)$  based on the a-priori probabilities  $P(u_k = \pm 1)$ , LLR can also be defined using a-posteriori probabilities as:

$$L(u_k / y_k) \triangleq \ln \left( \frac{P(u_k = +1 | y_k)}{P(u_k = -1 | y_k)} \right) \quad (2.10)$$

Where  $y$  is the noise added transmitting sequence. The conditional probabilities  $P(u_k = \pm 1 | \underline{y})$  are known as the a-posteriori probabilities of data bit  $u_k$ . Assume that  $y_k$  is the received symbol.

Then

$$P(u_k | y_k) = P(u_k)P(y_k | u_k) \quad (2.11)$$

where  $P(u_k | y_k)$  can be computed as

$$P(y / u_k) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y_k - u_k)^2}{2\sigma^2}} \quad (2.12)$$

Where  $\sigma^2$  is the variance of noise. For a sequence of received symbols  $\underline{y}$  the LLR is defined as:

$$L(u_k / \underline{y}) \triangleq \ln \left( \frac{P(u_k = +1, \underline{y})}{P(u_k = -1, \underline{y})} \right) \quad (2.13)$$

In Fig.2.7 a section of trellis diagram for  $(1,5/7)_{octal}$  convolutional encoder is given that if the previous state  $S_{k-1}$  and the present state  $S_k$  it is seen from Fig.2.7, then the value of the input bit  $u_k$  that causes the transition between these two states, will be known. Hence the probability that  $u_k = +1$  is equal to the probability that the transition from the previous state  $S_{k-1}$  to the present

state  $S_k$  is one of the set of four possible transitions that can occur when  $u_k = +1$  (i.e. those transitions shown with broken lines). This set of transitions is mutually exclusive (i.e. only one of them could have occurred at the encoder). Using Bayes' Rule Equation (2.12) can be re-written as;

$$L(u_k / \underline{y}) = \left( \frac{\sum_{\substack{(s',s) \Rightarrow \\ u_k = +1}}^n P(S_{k-1} = s', S_k = s, \underline{y})}{\sum_{\substack{(s',s) \Rightarrow \\ u_k = -1}}^n P(S_{k-1} = s', S_k = s, \underline{y})} \right). \quad (2.14)$$

Where  $(s',s)$  with  $u_k = +1$  is the set of transitions from the previous state  $S_{k-1} = s'$  to the present state  $S_k = s$  and similarly for  $(s',s)$  with  $u_k = -1$  for brevity we shall write  $P(S_{k-1} = s', S_k = s, \underline{y})$  as  $P(s', s, \underline{y})$ .

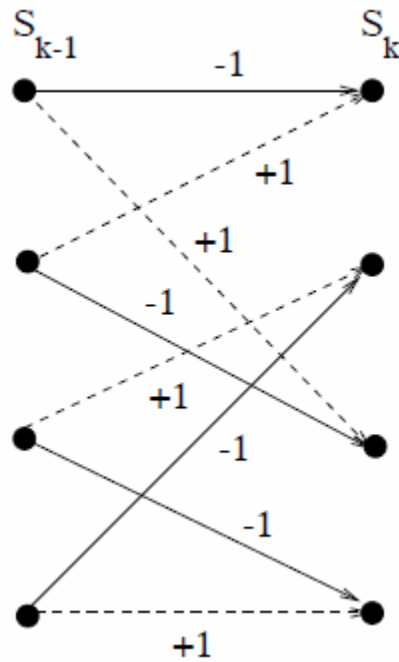


Figure 2.7 Possible Transitions

The individual probabilities can be written as  $P(s', s, \underline{y})$  :

$$P(s', s, \underline{y}) = P(s', s, \underline{y}_{j < k}, \underline{y}_k, \underline{y}_{j > k}) \quad (2.15)$$

Using *Bayes' Rule*  $P(a, b) = P(a | b)P(b)$  and the fact that if we assume that the channel is memoryless, then the future received sequence  $\underline{y}_{j > k}$  will depend only on the present state  $s$  and not on the previous state  $s'$  or the present and previous received channel sequences  $\underline{y}_k$  and  $\underline{y}_{j < k}$ . Thus, Equation (2.14) can be written as;

$$\begin{aligned} P(s', s, \underline{y}) &= P(\underline{y}_{j > k} | \{s', s, \underline{y}_{j < k}\}) P(s', s, \underline{y}_{j < k}, \underline{y}_k) \\ &= P(\underline{y}_{j > k} | s) P(s', s, \underline{y}_{j < k}, \underline{y}_k) \end{aligned} \quad (2.16)$$

Again, using *Bayes' rule* and the assumption that the channel is memoryless, Equation 2.15 can be expanded as follows:

$$\begin{aligned} P(s', s, \underline{y}) &= P(\underline{y}_{j > k} | s) P(s', s, \underline{y}_{j < k}, \underline{y}_k) \\ &= P(\underline{y}_{j > k} | s) P(\{\underline{y}_k, s\} | \{s', \underline{y}_{j < k}\}) P(s', \underline{y}_{j < k}) \\ &= P(\underline{y}_{j > k} | s) P(\{\underline{y}_k, s\} | s') P(s', \underline{y}_{j < k}) \\ &= \beta_k(s) \gamma_k(s', s) \alpha_{k-1}(s') \end{aligned} \quad (2.17)$$

Where;

$$\alpha_{k-1}(s') = P(S_{k-1} = s', \underline{y}_{j < k}) \quad (2.18)$$

which is the probability that the Trellis is in state  $s'$  at time  $k-1$  and the received channel sequence up to this point is  $\underline{y}_{j < k}$ , as visualized in Figure

2.8.

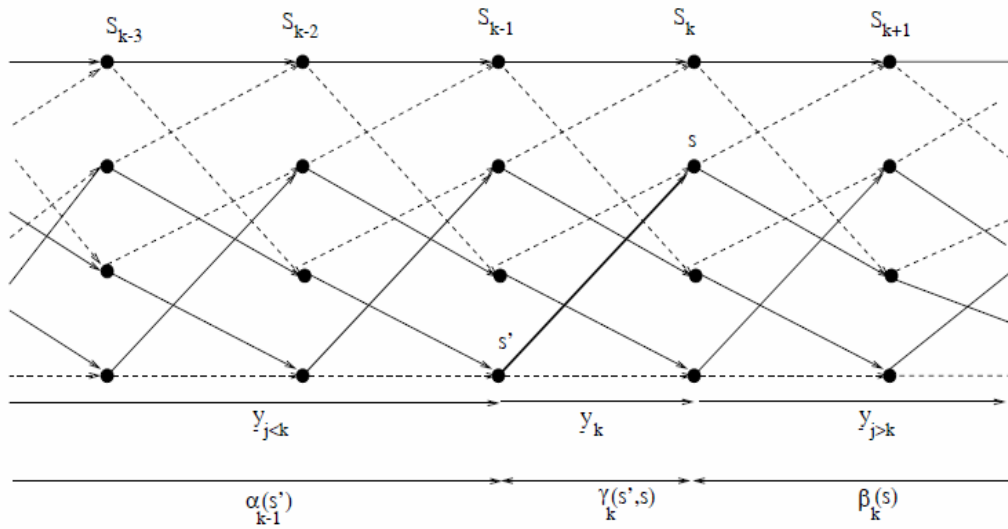


Figure 2.8 MAP Decoder Trellis

In Equation (2.17)

$$\beta_k(s) = P(\underline{y}_{j>k} | S_k = s) \quad (2.19)$$

is the probability that given the trellis is in state  $s$  at time  $k$  the future received channel sequence will be  $\underline{y}_{j>k}$ , and finally:

$$\gamma_k(s', s) = P((\underline{y}_k, S_k = s) | S_{k-1} = s') \quad (2.20)$$

is the probability that given the trellis was in state  $s'$  at time  $k-1$ , and it moves to state  $s$  and the received channel sequence for this transition is  $\underline{y}_k$ .

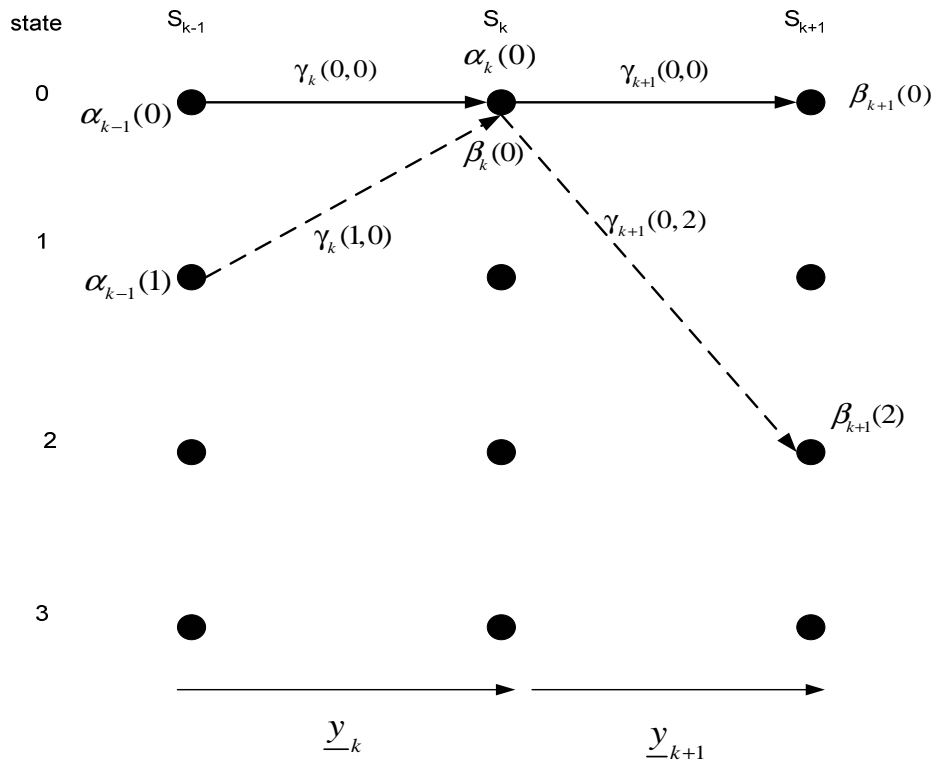


Figure.2.9 Recursive Calculation of  $\alpha_k(0)$  and  $\beta_k(0)$

Equation (2.17) shows that the probability  $P(s', s, \underline{y})$  can be split into the product of three terms:  $\alpha_{k-1}(s')$ ,  $\gamma_k(s', s)$  and  $\beta_k(s)$ . The meaning of these three probability terms is shown in Figure 2.9. The MAP algorithm finds  $\alpha_k(s)$  and  $\beta_k(s)$  for all states  $s$  throughout the trellis, i.e. for  $k = 0, 1, \dots, N-1$ , and  $\gamma_k(s', s)$  for all possible transitions from state  $S_{k-1} = s'$  to state  $S_k = s$ . These values are then used to find the probabilities  $P(S_{k-1} = s', S_k = s, \underline{y})$  which are then used in Equation (2.13) to compute the  $L(u_k | y)$  for each  $u_k$ . Additionally, it can be described that how the values  $\alpha_k(s)$ ,  $\beta_k(s)$  and  $\gamma_k(s', s)$  can be calculated.

### 2.2.3.1 Forward Recursive Calculation of the $\alpha_k(s)$ Values

The  $\alpha_k(s)$  may be computed recursively as;

$$\alpha_k(s) = \sum_{s'} \gamma_k(s', s) \alpha_{k-1}(s') \quad (2.21)$$

Since;

$$\begin{aligned} \alpha_k(s) &= P(s, \underline{y}_1^k) \\ &= \sum_{s'} P(s', s, \underline{y}_1^k) \\ &= \sum_{s'} P(s, y_k | s', \underline{y}_1^{k-1}) P(s', \underline{y}_1^{k-1}) \\ &= \sum_{s'} P(s, y_k | s') P(s', \underline{y}_1^{k-1}) \\ &= \sum_{s'} \gamma_k(s, s') \alpha_{k-1}(s') , \end{aligned} \quad (2.22)$$

Thus,  $\alpha_k(s)$  may be recursively computed as  $\alpha_k(s) = \sum_{s'} \gamma_k(s', s) \alpha_{k-1}(s')$  with the initial condition given as;

$$\begin{aligned} \alpha_0(S_0 = 0) &= 1 \\ \alpha_0(S_0 = s) &= 0 \quad \forall s \neq 0. \end{aligned} \quad (2.23)$$

### 2.2.3.2 Backward Recursive Calculation of the $\beta_k(s)$ Values

Using Equation (2.19); thus,  $\beta_{k-1}(s')$  can be computed recursively;

$$\beta_{k-1}(s') = \sum_s \beta_k(s) \gamma_k(s', s) .$$



$$\begin{aligned}
\beta_{k-1}(s') &= P(\underline{y}_k | s') \\
&= \sum_s P(\underline{y}_k, s | s') \\
&= \sum_s P(\underline{y}_{k+1} | s', s, \underline{y}_k) P(s, y_k | s') \\
&= \sum_s P(\underline{y}_{k+1} | s) P(s, y_k | s') \\
&= \sum_s \beta_k(s) \gamma_k(s', s) \quad .
\end{aligned}
\tag{2.24}$$

As an example, in Figure 2.9;

$$\begin{aligned}
\beta_k(0) &= \beta_{k+1}(0) \gamma_{k+1}(0, 0) + \beta_{k+2}(2) \gamma_{k+1}(0, 2) \\
\alpha_k(0) &= \alpha_{k-1}(0) \gamma_k(0, 0) + \alpha_{k-1}(1) \gamma_k(1, 0) \quad .
\end{aligned}
\tag{2.25}$$

### 2.2.3.3 Calculation of the $\gamma_k(s', s)$ Values

$\alpha_{k-1}(s), \beta_k(s)$  are the state probabilities.  $\gamma_k(s', s)$  is the transition probability among states, these are also called *branch metrics*. Additionally, it is considered that how the  $\gamma_k(s', s)$  values in Equation (2.22) can be calculated from the received signal sequence using the definition of  $\gamma_k(s', s)$  from computation Equation (2.26) and *Bayes' Rule* of

$$\begin{aligned}
\gamma_k(s', s) &= P(s, \underline{y}_k | s') \\
&= \frac{P(s', s, \underline{y}_k)}{P(s')} \\
&= \frac{P(s', s, \underline{y}_k)}{P(s')} \frac{P(s', s)}{P(s', s)} \\
&= \frac{P(s', s)}{P(s')} \frac{P(s', s, \underline{y}_k)}{P(s', s)} \\
&= P(s | s') P(\underline{y}_k | s', s) \quad .
\end{aligned}
\tag{2.26}$$

The probability  $P(s',s)$  equals to  $P(u_k)$ . Since going from  $s'$  to  $s$  depends on input data. Hence,  $P(s'|s) = P(u_k)$ . Thus,

$$\begin{aligned}\gamma_k(s'|s) &= P(u_k)P(y_k|s',s) \\ &= P(u_k)P(y_k|c_k) \quad ,\end{aligned}\tag{2.27}$$

Where  $c_k(u_k, p_k)$ ,  $y_k(y_{k1}, y_{k2})$  and  $P(y_k|c_k)$  is;

$$\begin{aligned}P(y_k|c_k) &= \frac{1}{2\pi\sigma^2} \exp\left[-\frac{1}{2\sigma^2} \left[ (y_{k1} - u_k) + (y_{k2} - p_k)^2 \right]\right] \\ &= \frac{1}{2\pi\sigma^2} \exp\left[-\frac{1}{2\sigma^2} \|y_k - c_k\|^2\right]\end{aligned}\tag{2.28}$$

#### 2.2.3.4 Summary of the MAP Algorithm

Forward recursion from Equation (2.22) can be used to calculate  $\gamma_k(s',s)$ . Once all the channel values have been received, and  $\gamma_k(s',s)$  are calculated for all  $k=0,1,\dots,N$ , the forward state probabilities  $\alpha_k(s',s)$  and the state probabilities  $\beta_k(s',s)$  are computed. Finally, all the calculated values of  $\alpha_k(s',s)$ ,  $\beta_k(s',s)$  and  $\gamma_k(s',s)$  are in

$$\begin{aligned}L(u_k|y) &\triangleq \frac{\sum_{\substack{(s',s) \Rightarrow \\ u_k=+1}} P(S_{k-1}=s', S_k=s, \underline{y})}{\sum_{\substack{(s',s) \Rightarrow \\ u_k=-1}} P(S_{k-1}=s', S_k=s, \underline{y})} \\ &\triangleq \ln \frac{\sum_{\substack{(s',s) \Rightarrow \\ u_k=+1}} \alpha_{k-1}(s') \cdot \gamma_k(s',s) \cdot \beta_k(s)}{\sum_{\substack{(s',s) \Rightarrow \\ u_k=-1}} \alpha_{k-1}(s') \cdot \gamma_k(s',s) \cdot \beta_k(s)}\end{aligned}\tag{2.29}$$

to decide decoded bits. These operation are summarized in Fig.2.10.

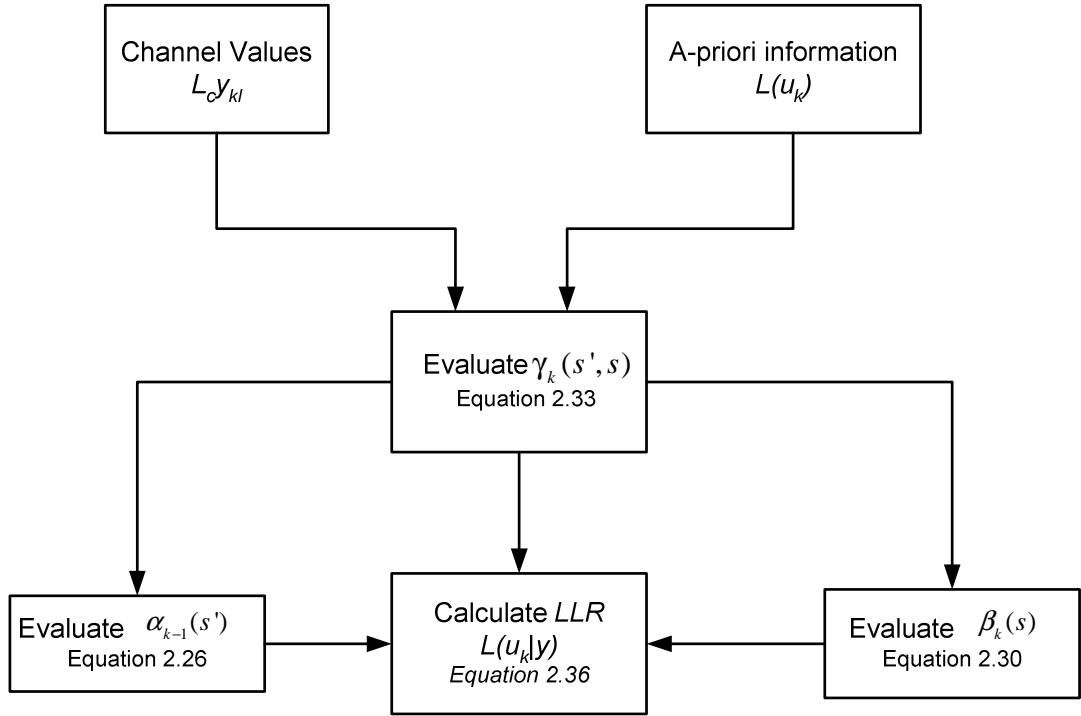


Figure 2.10 Summary of the MAP algorithm

## 2.2.4 LOG – MAP Algorithm

The *MAP* decoding algorithm requires large memory and a large number of operation involving exponential value computations and multiplications. One way of simplifying computation amount is to work with the logarithms of  $\gamma_k(s_1, s)$ ,  $\alpha_k(s')$  and  $\beta_{k+1}(s)$ , denoted by  $\tilde{\gamma}_k(s', s)$ ,  $\tilde{\alpha}_k(s')$  and  $\tilde{\beta}_{k+1}(s)$ . If log values are used  $\tilde{\alpha}_k(s) = \log \alpha_k(s)$ ; so,  $\alpha_k(s) = e^{\tilde{\alpha}_k(s)}$ . Using Equation (2.23);

$$\Rightarrow x_1 = e^{\tilde{\alpha}_k(s)} = e^{\tilde{\alpha}_{k-1}(s')} e^{\tilde{\gamma}_k(s', s)} + e^{\tilde{\alpha}_{k-1}(s')} e^{\tilde{\gamma}_k(s', s)}$$

$$\Rightarrow x_2 = e^{\tilde{\alpha}_k(s')} = e^{\tilde{\alpha}_{k-1}(s') + \tilde{\gamma}_k(s', s)} + e^{\tilde{\alpha}_{k-1}(s') + \tilde{\gamma}_k(s', s)}$$

Using log of  $x_1$  and  $x_2$ ;

$$\Rightarrow \tilde{\alpha}_k(s) = \log(e^{\tilde{\alpha}_{k-1}(s')} + e^{\tilde{\alpha}_{k-1}(s) + \tilde{\gamma}_k(s',s)}) \quad (2.30)$$

Now consider *MAP*;

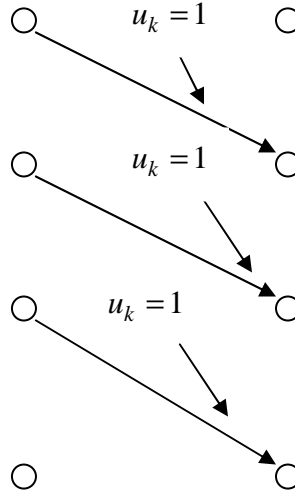


Figure 2.11 Trellis Diagram for MAP

$$P(u_k = +1 | y) = \sum_{\substack{s',s \\ u_k=1}} \alpha_k(s') \gamma_k(s',s) \beta_{k+1}(s) \quad (2.31)$$

In log domain;

$$\tilde{P}(u_k = 1 | y) = \log \sum_{\substack{s',s \\ \gamma_k=1}} \left( e^{\tilde{\alpha}_k(s') + \tilde{\gamma}_k(s',s) + \tilde{\beta}_{k+1}(s)} \right) \quad (2.32)$$

$$\gamma_k(s',s) = P(u_k) P(y_k | c_k)$$

$$\tilde{\gamma}_k(s',s) = \log P(u_k) + \log P(y_k | c_k)$$

$$= L(u_k) + \log P(y_k | c_k)$$

(2.33)

Now consider  $P(y_k | c_k)$

$$P(y_k | c_k) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[\frac{y_{k1} - c_{k1}}{2\sigma^2}\right] \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\left(\frac{y_{k2} - c_{k2}}{2\sigma^2}\right)^2\right] \quad (2.34)$$

$$\log P(y_k | c_k) = \log \frac{1}{2\pi\sigma^2} + \frac{1}{2\sigma^2} \left[ (y_{k1} - c_{k1})^2 + (y_{k2} - c_{k2})^2 \right] \quad (2.35)$$

This can be simplified as;

$$\log P(y_k | c_k) = -\log 2\pi\sigma^2 - \frac{1}{2\sigma^2} \left[ y_{k1}^2 + y_{k2}^2 + c_{k1}^2 + c_{k2}^2 - 2y_{k1}c_{k1} - 2y_{k2}c_{k2} \right] \quad (2.36)$$

$$= \log 2\pi\sigma^2 - \frac{1}{2\sigma^2} (y_{k1}^2 + y_{k2}^2 + c_{k1}^2 + c_{k2}^2) + \frac{1}{\sigma^2} [y_{k1}c_{k1} + y_{k2}c_{k2}] \quad (2.37)$$

Where the first and second term can be dropped due to non dependently on  $u_k$ , and the following can be obtained;

$$\log P(y_k | c_k) = \frac{1}{\sigma^2} (y_{k1}c_{k1} + y_{k2}c_{k2})$$

$$\begin{array}{ccc} \downarrow & & \downarrow \\ u_k & & P_k \end{array} \quad (2.38)$$

Hence,

$$\widetilde{\gamma}_k(s', s) = L(u_k) + \frac{1}{\sigma^2} (y_{k1}c_{k1} + y_{k2}c_{k2})$$

$$\begin{array}{ccc} \downarrow & & \downarrow \\ u_k & & P_k \end{array} \quad (2.39)$$

Using the log values of  $\alpha_k(s)$ ,  $\gamma_k(s', s)$  and  $\beta_k(s)$  it can be written as  $P(\widetilde{u}_k | y) = 1$ ;

$$\widetilde{P}(u_k = 1 | y) = \log \sum_{\substack{s', s \\ u_k=1}} \exp \left[ \alpha_k(s) + L(u_k) + \frac{1}{\sigma^2} (y_{k1}c_{k1} + y_{k2}c_{k2}) + \widetilde{\beta}_{k+1}(s) \right] \quad (2.40)$$

This can be simplified as;

$$\widetilde{P}(u_k = 1 | y) = \log \sum_{\substack{s', s \\ u_k=1}} \exp \left[ \widetilde{\alpha}_k(s) + L(u_k) + \frac{1}{\sigma^2} y_{k1} + \frac{1}{\sigma^2} y_{k2}c_{k2} + \beta_{k+1}(s) \right] \quad (2.41)$$

Also, using  $\log(e^{a+b+c} + e^{a+d+f}) = a + \log(e^{b+c} + e^{d+f})$  property, Equation (2.41) can be further simplified of;

$$\widetilde{P}(u_k = 1 | y) = \frac{1}{\sigma^2} y_{k1} + L(u_k) + \log \sum_{s', s} \exp \left[ \widetilde{\alpha}_k(s) + \frac{1}{\sigma^2} y_{k2}c_{k2} + \beta_{k+1}(s) \right] \quad (2.42)$$

Where;

$$\frac{1}{\sigma^2} y_{k1} \rightarrow \text{Channel Value } L_c y_k \quad L_c \text{ is the channel reliability value}$$

$$L(u_k) \rightarrow a \text{ - priori provided by the other decoder}$$

$$\log \sum_{s',s} \exp \left[ \widetilde{\alpha}_k(s) + \frac{1}{\sigma^2} y_{k2} c_{k2} + \widetilde{\beta}_{k+1}(s) \right] \rightarrow L_e(u_k) \quad \text{Extrinsic information}$$

$$\widetilde{P}(u_k = 1 | y) \text{ is } a\text{-posteriori log-likelihood ration } L(\hat{u}) .$$

### 2.3 Turbo Code

Berrou, Glavieux and Thitimajshima [11] introduced in 1993 a novel and apparently revolutionary error-control coding technique, which they called turbo coding. This coding technique consists essentially of a parallel concatenation of two binary convolutional codes. These codes obtain an excellent bit error rate (*BER*) performance by making use of three main components. They are constructed using two systematic convolutional encoders usually known as recursive systematic convolutional (*RSC*) encoders, which are concatenated in parallel. In this parallel concatenation, a random inter-leaver plays a very important role as the randomizing constituent part of the coding technique. Interleaving procedure is designed to make the encoder output sequences be statistically independent from each other. This coding scheme is decoded by means of an iterative decoder that makes the resulting *BER* performance close to the Shannon limit.

In parallel concatenation, so that each input element is encoded twice, but the input to the second encoder passes first through a random inter-leaver. Turbo codes can be iteratively decoded using soft decision decoding algorithms. The decoders operate in a soft-input–soft-output mode; that is, both the input applied to each decoder, and the resulting output generated by the decoder, should be soft decisions or estimates. Both decoders operate by utilizing what is called *a priori* information, and together with the channel information provided by the samples of the received sequence, and information about the structure of the code, they produce an estimate of the message bits. They are also able to produce an estimate called the extrinsic

information, which is passed to the other decoder, information that in the following iteration will be used as the *a priori* information of the other decoder. Thus the first decoder generates extrinsic information that is taken by the second decoder as its *a priori* information. This procedure is repeated in the second decoder. The first decoder then takes the received information as its *a priori* information for the new iteration, and operates in the same way as described above, and so on.

The iterative passing of information between the first and the second decoders continues until a given number of iterations are reached. With each iteration, the estimate of the message bits improves and they usually converge to a correct estimate. The number of errors corrected increases as the number of iterations increases. However, the improvement of the estimates does not increase linearly, and so, in practice, it is enough to utilize a reasonable small number of iterations to achieve acceptable performance [12 13 14].

One of the most suitable decoding algorithms that performs soft – input – soft - output decisions is a maximum *a posteriori* (*MAP*) algorithm known as the *BCJR* (*Bahl, Cocke, Jelinek, Raviv, 1974*) algorithm [15]. Further optimizations of this algorithm lead to lower complexity algorithms, like *SOVA* (*soft-output Viterbi algorithm*), and the *LOG MAP* algorithm, which is basically the *BCJR* algorithm with logarithmic computation [9].

### **2.3.1 Turbo Encoder**

A turbo encoder structured using two *RSC encoders* arranged in parallel, and combined with a random *inter-leaver*, together with a multiplexing and puncturing block, is depicted in Figure 2.12.

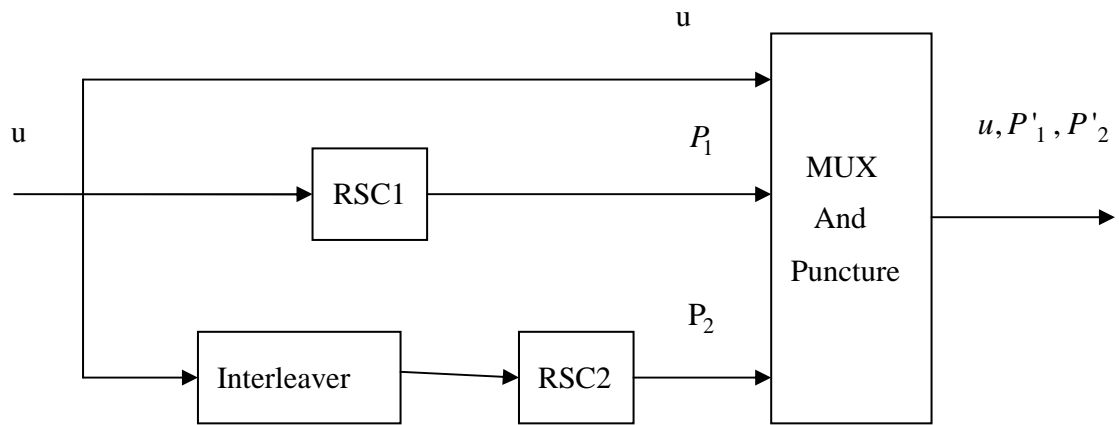


Figure 2.12 Turbo Encoder

In the traditional structure of a turbo encoder, the encoders are usually *RSC* encoders of rate  $R_c = \frac{1}{2}$ , such that  $P'_1 = P_1$ ,  $P'_2 = P_2$ , and the lengths of the sequences  $u$ ,  $P_1$  and  $P_2$ ,  $P'_1$  and  $P'_2$  are all the same. Then the overall turbo code rate is  $R_c = \frac{1}{3}$ . Puncturing [2, 9, 16] is a technique very commonly used to improve the overall rate of the code. The puncturing selection process is performed by periodically eliminating one or more of the outputs generated by the constituent *RSC* encoders. During puncturing, the parity bits generated by these two encoders are alternately eliminated so that the redundant bit of the first encoder is first transmitted, eliminating that of the second decoder, and in the following time instant the redundant bit of the second encoder is transmitted, eliminating that of the first. In this way, the lengths of  $P'_1$  and  $P'_2$  are half the lengths of  $P_1$  and  $P_2$ , respectively, and the resulting overall rate becomes  $R_c = \frac{1}{2}$ . Puncturing is not preferred for message (systematic) bits, due to *BER* performance loss. The use of interleaver has a major influence on the *BER* performance of a turbo code, especially its length and structure. The excellent *BER* performance of these codes is enhanced when the length of the inter-leaver is significantly large and its structure have pseudo random nature. The interleaving block and its corresponding de-inter-leaver in the decoder do not much increase the complexity of a turbo scheme. However, it does introduce a significant delay



in the system, which in some cases can be a strong drawback, depending on the application. The *RSC-generated* Convolutional Codes are comparatively simple, but offer excellent performance when iteratively decoded using *soft-input–soft-output algorithms*.

### 2.3. 2 ITERATIVE DECODING of TURBO CODES

Principle of the Iterative Decoding Algorithm in Figure 2.13, Soft-Input / Soft-Output decoder structure is shown [12].

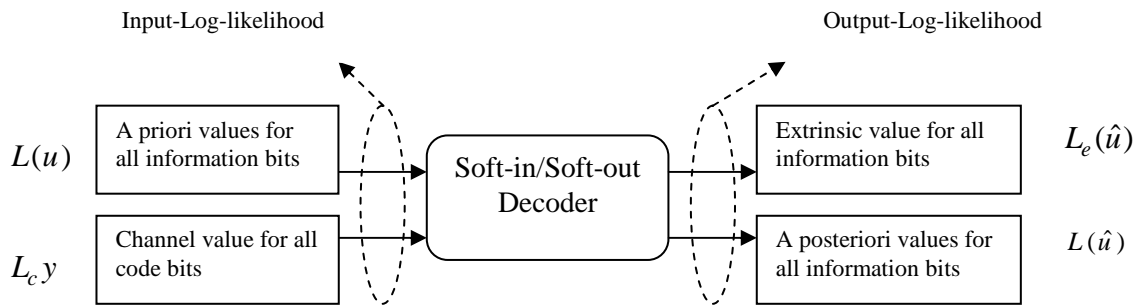


Figure 2.13 Soft-Input / Soft-Output Decoder

The output of the “symbol-by-symbol” **Maximum a posteriori Probability (MAP)** decoder is defined as the **a posteriori log-likelihood ratio**, that is, the logarithm of the ratio of the probabilities of a given bit being “+1” or “-1” given the observation  $y$ , and is given as

$$L(\hat{u}) = L(u_k / y) \triangleq \ln \left( \frac{P(u_k = +1 | \underline{y})}{P(u_k = -1 | \underline{y})} \right). \quad (2.43)$$

Such a decoder uses *a priori* values  $L(u)$  for all information bits  $u$ , if available, and the channel values  $L_c y$  for all coded bits. It also delivers soft outputs  $L(\hat{u})$  on all information bits and an extrinsic information  $L_e(\hat{u})$ , which contains the soft output information from all the other coded bits in the code sequence and is not influenced by the  $L(u)$  and  $L_c y$  values of the current bit

[10,17]. For systematic codes, the soft output for the information bit  $u$  will be represented as the sum of three terms

$$L(\hat{u}) = L_c y + L(u) + L_e(\hat{u}) \quad (2.44)$$

This means that three independent estimates exist for the log-likelihood ratio of the information bits; the channel values  $L_c y$  the *a priori* values  $L(u)$  and the values  $L_e(\hat{u})$  by a third independent estimator utilizing the code constraint. The whole procedure of iterative decoding with two “Soft-in/Soft-out” decoders is shown in Figure (2.14). In the first iteration of the iterative decoding algorithm, decoder 1 computes

The extrinsic information;

$$L_e^1(\hat{u}) = L^1(\hat{u}) - [L_c y + L(u)] \quad (2.45)$$

Assume that equally likely information bits; thus, it is initialized that  $L(u) = 0$  for the first iteration. This extrinsic information from the first decoder is passed to the decoder 2, which uses  $L_e^1(\hat{u})$  as the *a priori* value in place of  $L(u)$  to compute  $L^2(u)$ . Hence, the extrinsic information value computed by Decoder2 is;

$$L_e^2(\hat{u}) = L^2(\hat{u}) - [L_c y + L_e^1(\hat{u})] \quad (2.46)$$

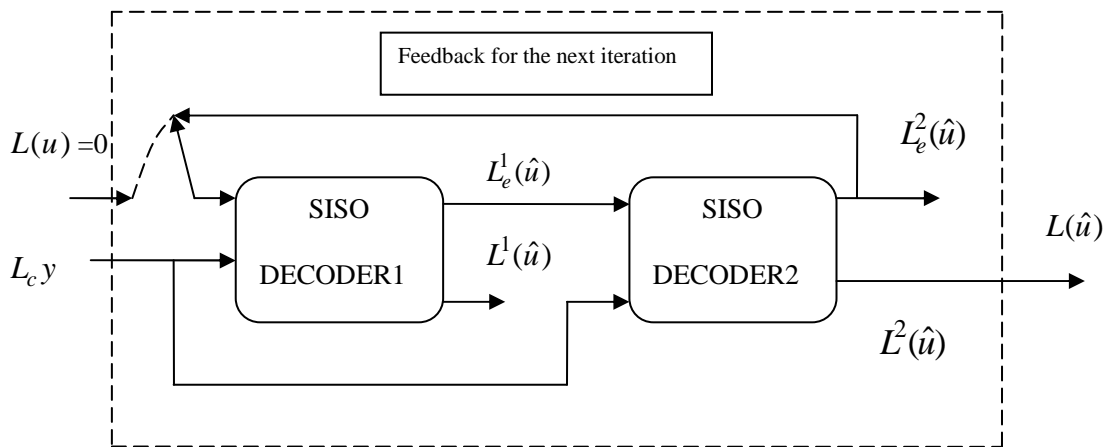


Figure 2.14 Iterative Decoding Procedures with Two “Soft-Input/Soft-Output Decoder

Then, Decoder 1 will use the extrinsic information values  $L_e^2(\hat{u})$  as *a priori* information in the second iteration. The computation is repeated every iteration. The iterative process is usually terminated after a predetermined number of iterations, when the soft-output value  $L^2(\hat{u})$  stabilizes and changes little between successive iterations. In the final iteration, Decoder 2 combines both extrinsic information values in computing the soft-output values of;

$$L^2(\hat{u}) = L_c y + L_e^1(\hat{u}) + L_e^2(\hat{u}) \quad (2.47)$$

and makes decision on decoded bits.

It was summarized below what is meant by the terms a-priori, extrinsic and a-posteriori information, which will be used throughout this thesis [9].

**A priori**  $L(u_k)$  : The a-priori information related to a bit is information known before decoding commences, from a source other than the received sequence or the code constraints. It is also sometimes referred to as intrinsic information for contrasting it with the extrinsic information to be described next.

**Extrinsic**  $L_e(u_k)$  : The *extrinsic* information related to a bit  $u_k$  is the information provided by a decoder based on the received sequence and on the a-priori information, but *excluding* the received systematic bit  $y_u$  and the a-priori information  $L(u)$  related to the bit  $u_k$ . Typically the component decoder provides this information using the constraints imposed on the transmitted sequence by the code used. It processes the received bits and the a-priori information surrounding the systematic bit  $u_k$ , and uses this information and the code constraints for providing information about the value of the bit  $u_k$ .

**A posteriori**  $L(u_k/y)$  : The *a-posteriori* information related to a bit is the information that the decoder generates by taking into account *all* available sources of information concerning  $u_k$ . It is the a-posteriori *LLR*, i.e.  $L(u_k/y)$

that the *MAP algorithm* generates as its output. A more detailed explanation of Figure 2.14 is shown in Figure 2.15 for  $R_c = \frac{1}{3}$  turbo decoders.

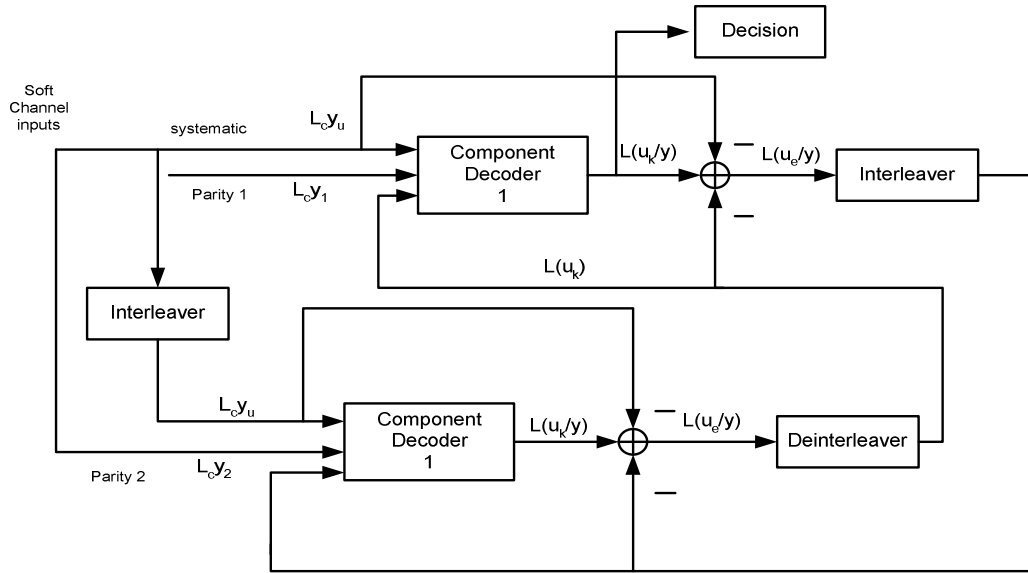


Figure 2.15 Turbo Decoder Schematic

In Figure 2.15, decoder the first decoder receives the channel sequence  $L_c y_1$  containing the received versions of the transmitted systematic bits,  $L_c y_u$ , and the parity bits,  $L_c y_2$  [10,17,28]. Usually, to obtain a half-rate code, half of these parity bits will have been punctured at the transmitter, and so the turbo decoder must insert zeros in the soft channel output  $L_c y_u$  for these punctured bits. The first component decoder can then process the soft channel inputs and produce its estimate  $L_{11}(u_k / y)$  of the conditional *LLRs* of the data bits  $u_k, k = 1, 2, \dots, N$ . In this notation the subscript 11 in  $L_{11}(u_k / y)$  indicates that this is the a-posteriori *LLR* in the first iteration from the first component decoder. Note that in this first iteration the first component decoder will have no a-priori information about the bits, and hence  $L(u_k)$  in Equation (2.34) giving  $\gamma_k(s', s)$  will be zero, corresponding to a-priori probability of 0.5. Next, the second component decoder comes into

operation. It receives the channel sequence  $L_c y_2$  containing the *interleaved* version of the received systematic bits, and the parity bits from the second encoder. However now in addition to the received channel sequence  $L_c y_2$ , the decoder can use the conditional *LLR*  $L_{11}(u_k/y)$  provided by the first component decoder to generate posteriori to be used by the second component decoder. Ideally these a-posteriori information would be completely independent from all the other information used by the second component decoder. As can be seen in Fig.2.16 in iterative turbo decoders the extrinsic information  $L_e(u_k)$  from the other component decoder is used as the a-priori after being interleaved. Again, according to eqn.2.42, the reason for the subtraction paths shown in Fig.2.16 is that the a-posteriori *LLRs* from one decoder have the systematic soft channel inputs  $L_c y_u$  and the a-priori *LLRs*  $L(u_k)$  (if any were available) subtracted to yield the extrinsic information  $L_e(u_k)$  which are then used as a-priori *LLRs* for the other component decoder. The second component decoder thus uses the received channel sequence  $L_c y_2$  and the a-priori *LLRs*  $L(u_k)$  to produce its a-posteriori *LLRs*  $L_{12}(u_k/y)$ . This is the end of the first iteration. For the second iteration the first component encoder again processes its received channel sequence  $L_c y_1$ , but now it has updated a-priori  $L(u_k)$  provided by the second component encoder, and hence it can produce an improved a-posteriori *LLRs*  $L_{21}(u_k/y)$ . The second iteration then continues with the second component decoder using the improved a-posteriori  $L_{21}(u_k/y)$  from the first encoder to derive, through Equation (2.42), improved a-priori *LLRs*  $L(u_k)$  which it uses in conjunction with its received channel sequence  $L(u_k)$  to calculate  $L_{22}(u_k/y)$ . This iterative process continues and the each iteration on the average the *BER* of the decoded bits will fall. However, the improvement in performance for the each additional iteration carried out falls as the number of iterations increases. Hence for complexity reasons usually only about between 8 and 12 iterations are carried out, as no significant

improvement in performance is obtained with a higher number of iterations. This is the arrangement of 8 iterations which was used in the simulations, i.e. the decoder carries out a fixed number of iterations.

## 2.4 Simulation Results of Turbo Codes, MAP, Soft - Hard Decision Algorithms

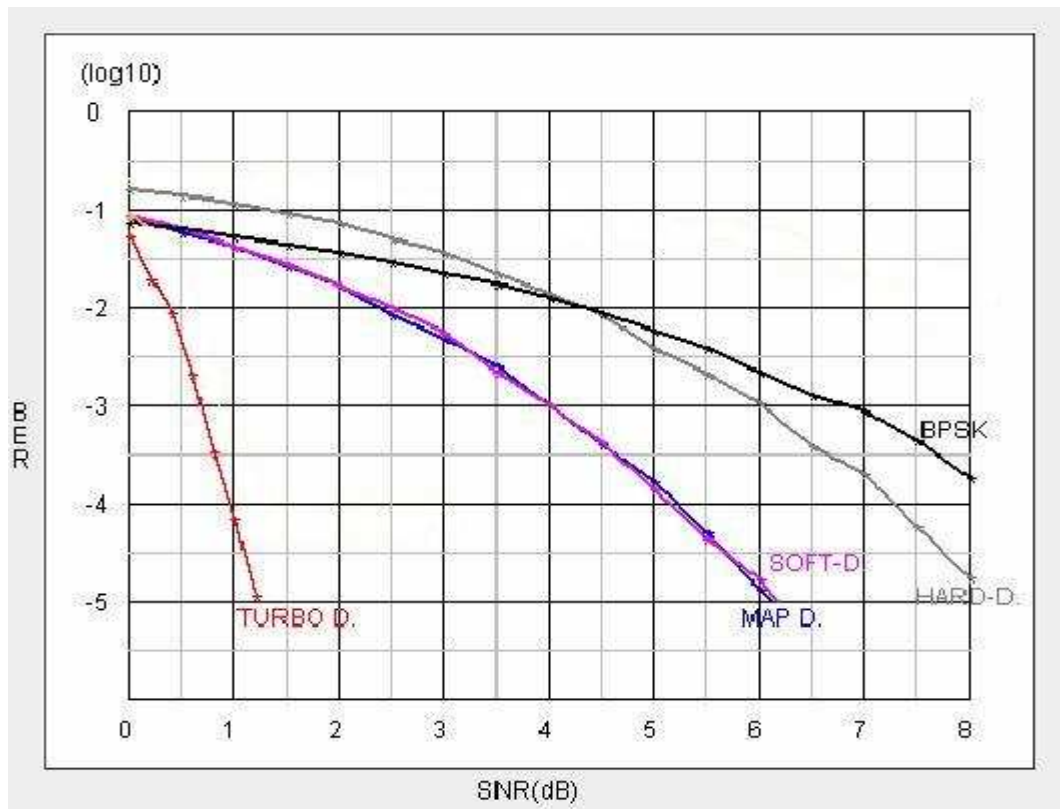


Figure.2.16 Simulation Result of Turbo Code

$(1,5/7)_{\text{octal}}$  RSCs were used for turbo encoders. The rate of the code is  $R_c = 1/3$  and an inter-leaver of length 1024 is used. S-Random inter-leaver is used with  $s=10$ . Iteration number is chosen as 8. First the data sequence is *BPSK* modulated and it is passed through an *AWGN* channel with noise variance  $\sigma^2$ . The simulation results are shown in Figure 2.16 where performance graphs for Map decoder, Turbo decoder, Hard and Soft Decision *Viterbi* Algorithm are depicted. It is clear from Figure 2.16 that Turbo Decoder achieves the best performance and is 5 dB better than MAP

decoder.

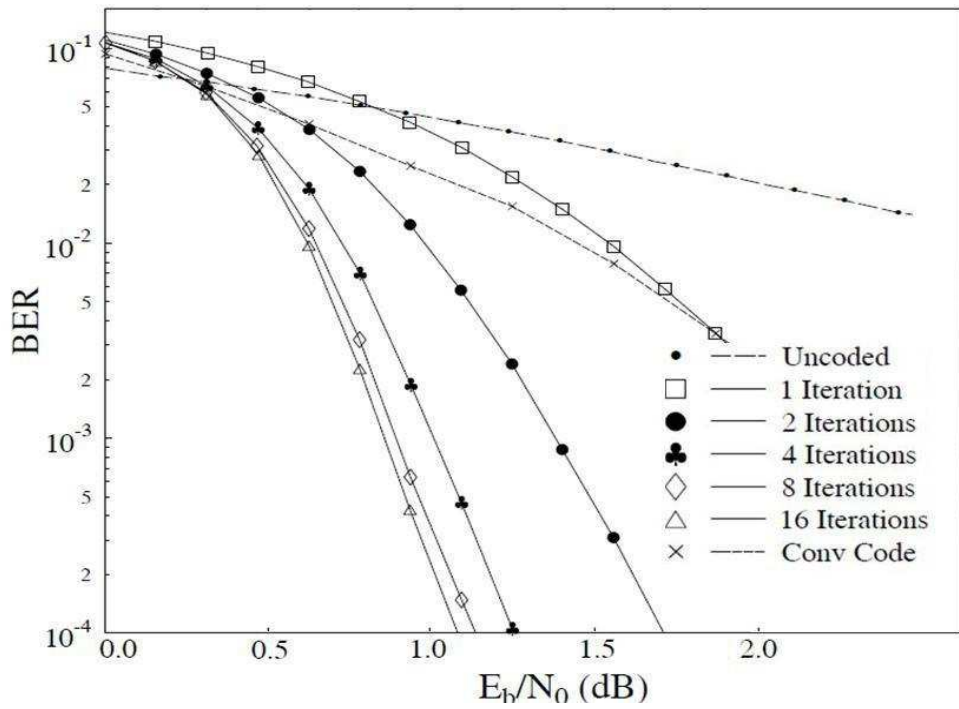


Figure 2.17 Turbo Coding BER Performance Using Different Number of Iterations

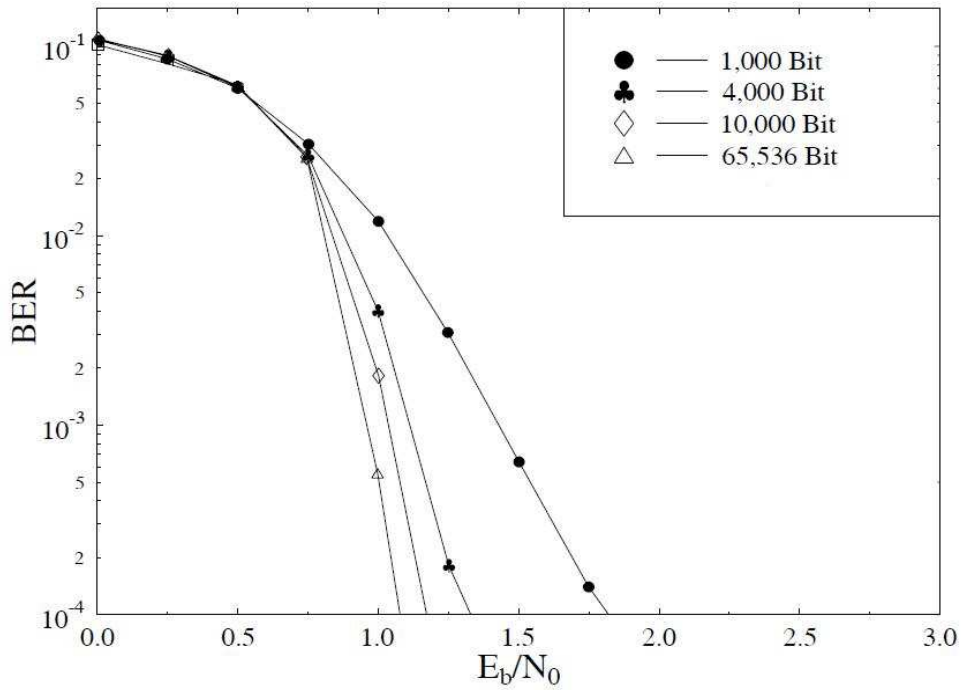


Figure 2.18 Effect of Frame Length on BER Performance of One-Third Rate Turbo Coding [9]

## CHAPTER 3

### BLOCK CODES

A *block code* is set of vectors that have a well defined mathematical property of structure and each vector is a sequence of a fixed number of bits. The vectors belonging to a block code are called *code words*. A code word both contains information bits and parity-check bits which are used for error correction and detection purposes. An  $n$ -bit code has code words which have  $k$  information bits  $r$  parity bits  $n = k + r$ . Such a code referred as an  $(n, k)$  *block code*,  $n$  and  $k$  are respectively named as the *block length* and *information word length*.

A codeword whose information bits are kept together is defined as in a systematic form otherwise the codeword is referred to as in *non-systematic* form. A block code whose code words are in systematic form is referred to as a *systematic* code. Systematic codes are normally preferred to *nonsystematic* codes.

In this thesis *BCH (31,21) Bose Chaudhuri and Hocquenghem*, block codes will be used *BCH* Codes which is a subclass of cyclic code. *BCH* Code parameters are given as;

$$n = 2^m - 1 \tag{3.1}$$

$$n - k \leq mt \tag{3.2}$$

$$d_{\min} = 2t + 1 \tag{3.3}$$



Where  $t$  is the number of errors to be corrected,  $n$  is the block length,  $k$  is the information word length, and  $m$  is an integer. For *BCH (31,21)* code the parameters are  $n = 31, k = 21, m = 5, t = 2$  and  $d_{\min} = 5$ .

In general the minimum distance of a code is the minimum *Hamming distance* between any two different code words, i.e.

$$d_{\min} = \min_{\substack{c_i, c_j \\ i \neq j}} d(c_i, c_j) \quad (3.4)$$

Where the Hamming distance between two code words  $c_i$  and  $c_j$  is the number of components at which the two code words differ, and is denoted by  $d(c_i, c_j)$ . Using  $d_{\min}$  *BCH* code can also be shown as *BCH (n, k, d<sub>min</sub>)*, for *BCH(31,21,5)*.

### 3.1 *BCH (31, 21, 5)* Block Code Encoder

*BCH* codes are defined over the mathematical structure of *finite fields*; for this reason, their construction will be briefly considered. The mathematical framework of the *BCH* code is *Galois field* and these codes defined in the binary field  $GF(2)$ . Galois fields containing more than two elements can be constructed using  $GF(2)$ . Construction of  $GF(2^m)$  can be performed using the primitive element  $\alpha$  which satisfies  $\alpha^{2^m-1} = 1$  and all the other elements of  $GF(2^m)$  can be generated from primitive element. The elements of the extension field  $GF(q=2^m)$  can be written as  $\{0, 1, \alpha, \alpha^2, \alpha^3, \dots, \alpha^{q-2}\}$  and their polynomial representation is given by the remainder of  $x^n + 1$  upon division by the prime polynomial  $p(x)$ :

$$\alpha^n = \text{Remainder} \left\{ (x^n + 1) / p(x) \right\}. \quad (3.5)$$

For the construction of *BCH (31,21)* code, first, the finite field  $GF(2^5)$  will be constructed. The primitive polynomial for *BCH (31,21)* can be chosen as

$p(x) = \alpha^5 + \alpha^2 + 1$ . The field  $GF(2^5)$  can be generated from the newly defined element  $\alpha$  which satisfies

$$\alpha^5 + \alpha^2 + 1 = 0 \quad . \quad (3.6)$$

Consider addition and multiplication of  $\alpha$  with the binary numbers 0 and 1, the binary numbers 0 and 1 have the form additive and multiplicative elements respectively, so

$$\begin{aligned} \alpha + 0 &= \alpha \\ \alpha * 1 &= \alpha . \end{aligned} \quad (3.4)$$

The additive inverse of  $\alpha$  is  $\alpha$  itself, as can be easily shown

$$\alpha + \alpha = 1\alpha + 1\alpha = (1+1)\alpha = 0\alpha = 0 \quad (3.5)$$

And;

$$\alpha + \alpha = 0, \quad (3.6)$$

This gives;

$$\alpha = -\alpha \quad . \quad (3.7)$$

After these properties rearranging (3, 7), the following can be obtained;

$$\alpha^5 = \alpha^2 + 1. \quad (3.8)$$

When constructing higher powers of  $\alpha$  Equation (3.13) is used to reduce field elements to their lowest power. Power of  $\alpha$  other than 5 can be obtained as:

$$\begin{aligned}
\alpha^5 &= \alpha^2 + 1 \\
\alpha^6 &= \alpha\alpha^5 = \alpha(\alpha^2 + 1) = \alpha^3 + \alpha \\
\alpha^7 &= \alpha\alpha^6 = \alpha(\alpha^3 + \alpha) = \alpha^4 + \alpha^2 \\
\alpha^8 &= \alpha\alpha^7 = \alpha(\alpha^4 + \alpha^2) = \alpha^5 + \alpha^3 = \alpha^3 + \alpha^2 + 1.
\end{aligned}
\tag{3.9}$$

This is continued in a similar manner up to  $\alpha^{30} = \alpha^4 + \alpha$  and finally;

$$\alpha^{31} = \alpha\alpha^{30} = \alpha(\alpha^4 + \alpha) = \alpha^5 + \alpha^2 = \alpha^2 + 1 + \alpha^2 = 1
\tag{3.10}$$

Constructing more powers of  $\alpha$  other than 31 will always give existing field elements; for instance,

$$\begin{aligned}
\alpha^{33} &= \alpha^{31}\alpha^2 = \alpha^2 \\
\alpha^{45} &= \alpha^{31}\alpha^{14} = 1\alpha^{14} = \alpha^4 + \alpha^3 + \alpha^2 + 1
\end{aligned}
\tag{3.11}$$

Table 3.1 32 Field Elements

0	$\alpha^{15} = \alpha^4 + \alpha^3 + \alpha^2 + \alpha + 1$
1	$\alpha^{16} = \alpha^4 + \alpha^3 + \alpha + 1$
$\alpha$	$\alpha^{17} = \alpha^4 + \alpha + 1$
$\alpha^2$	$\alpha^{18} = \alpha + 1$
$\alpha^3$	$\alpha^{19} = \alpha^2 + \alpha$
$\alpha^4$	$\alpha^{20} = \alpha^3 + \alpha^2$
$\alpha^5 = \alpha^2 + 1$	$\alpha^{21} = \alpha^4 + \alpha^3$
$\alpha^6 = \alpha^3 + \alpha$	$\alpha^{22} = \alpha^4 + \alpha^2 + 1$
$\alpha^7 = \alpha^4 + \alpha^2$	$\alpha^{23} = \alpha^3 + \alpha^2 + \alpha + 1$
$\alpha^8 = \alpha^3 + \alpha^2 + 1$	$\alpha^{24} = \alpha^4 + \alpha^3 + \alpha^2 + \alpha$
$\alpha^9 = \alpha^4 + \alpha^3 + \alpha$	$\alpha^{25} = \alpha^4 + \alpha^3 + 1$
$\alpha^{10} = \alpha^4 + 1$	$\alpha^{26} = \alpha^4 + \alpha^2 + \alpha + 1$
$\alpha^{11} = \alpha^2 + \alpha + 1$	$\alpha^{27} = \alpha^3 + \alpha + 1$
$\alpha^{12} = \alpha^3 + \alpha^2 + \alpha$	$\alpha^{28} = \alpha^4 + \alpha^2 + \alpha$
$\alpha^{13} = \alpha^4 + \alpha^3 + \alpha^2$	$\alpha^{29} = \alpha^3 + 1$
$\alpha^{14} = \alpha^4 + \alpha^3 + \alpha^2 + 1$	$\alpha^{30} = \alpha^4 + \alpha$

In Table 3.1, the 32 field elements are tabulated.

Complex roots of equations with real coefficients always occur in pairs of complex conjugates. If  $p + jq$  is a root of an equation with real coefficients then its complex conjugate  $p - jq$  is also a root. The roots of a polynomial with binary coefficients likewise occur in conjugates, not necessarily in pairs but in groups or sets of conjugates [19,20,21]. Given that  $\beta$  is a field element of  $GF(2^m)$  then the conjugates of  $\beta$  are;

$$\beta, \beta^2, \beta^4, \beta^8, \dots, \beta^{2^{r-1}}$$

Where  $r$  is the smallest integer such that  $\beta^{2^r} = \beta$ . For example consider the conjugates of  $\alpha$  in  $GF(2^5)$

$$(\alpha)^2 = \alpha^2$$

$$(\alpha)^4 = \alpha^4$$

$$(\alpha)^8 = \alpha^8$$

$$(\alpha)^{16} = \alpha^{16}$$

$$(\alpha)^{32} = \alpha^{31}\alpha = \alpha \quad [\text{recall that } \alpha^{31} = 1 \text{ in } GF(2^5)]$$

Therefore, in  $GF(2^5)$   $\alpha$  has four conjugate  $\{\alpha^2, \alpha^4, \alpha^8, \alpha^{16}\}$ . In a similar manner conjugate classes of  $\alpha^3, \alpha^5, \alpha^7, \alpha^{11}, \alpha^{15}$  can be found as;

$$(\alpha^3)^2 = \alpha^6$$

$$(\alpha^3)^4 = \alpha^{12}$$

$$(\alpha^3)^8 = \alpha^{24}$$

$$(\alpha^3)^{16} = \alpha^{48} = \alpha^{31}\alpha^{17} = \alpha^{17}$$

$$(\alpha^3)^{32} = \alpha^{96} = \alpha^{31}\alpha^{31}\alpha^{31}\alpha^3 = \alpha^3 \quad ,$$

The conjugates class of  $\alpha^3$  is  $\{\alpha^3, \alpha^6, \alpha^{12}, \alpha^{17}, \alpha^{24}\}$ .

$$\begin{aligned}
(\alpha^5)^2 &= \alpha^{10} \\
(\alpha^5)^4 &= \alpha^{20} \\
(\alpha^5)^8 &= \alpha^{40} = \alpha^{31}\alpha^9 = \alpha^9 \\
(\alpha^5)^{16} &= \alpha^{80} = \alpha^{31}\alpha^{31}\alpha^{18} = \alpha^{18} \\
(\alpha^5)^{32} &= \alpha^{160} = \alpha^{31}\alpha^{31}\alpha^{31}\alpha^{31}\alpha^{31}\alpha^5 = \alpha^5
\end{aligned}$$

The conjugates class of  $\alpha^5$  is  $\{\alpha^9, \alpha^{10}, \alpha^{18}, \alpha^{20}\}$ ,

$$\begin{aligned}
(\alpha^7)^2 &= \alpha^{14} \\
(\alpha^7)^4 &= \alpha^{28} \\
(\alpha^7)^8 &= \alpha^{56} = \alpha^{31}\alpha^{25} = \alpha^{25} \\
(\alpha^7)^{16} &= \alpha^{112} = \alpha^{93}\alpha^{19} = \alpha^{19} \\
(\alpha^7)^{32} &= \alpha^{224} = \alpha^7
\end{aligned}$$

The conjugates class of  $\alpha^7$  is  $\{\alpha^{14}, \alpha^{19}, \alpha^{25}, \alpha^{28}\}$ .

$$\begin{aligned}
(\alpha^{11})^2 &= \alpha^{22} \\
(\alpha^{11})^4 &= \alpha^{44} = \alpha^{31}\alpha^{13} = \alpha^{13} \\
(\alpha^{11})^8 &= \alpha^{88} = \alpha^{31}\alpha^{31}\alpha^{26} = \alpha^{26} \\
(\alpha^{11})^{16} &= \alpha^{176} = (\alpha^{31})^5\alpha^{21} = \alpha^{21} \\
(\alpha^{11})^{32} &= \alpha^{11}
\end{aligned}$$

The conjugates class of  $\alpha^{11}$  is  $\{\alpha^{13}, \alpha^{21}, \alpha^{26}\}$ .

$$\begin{aligned}
(\alpha^{15})^2 &= \alpha^{30} \\
(\alpha^{15})^4 &= \alpha^{60} = \alpha^{29} \\
(\alpha^{15})^8 &= \alpha^{120} = \alpha^{27} \\
(\alpha^{15})^{16} &= \alpha^{23} \\
(\alpha^{15})^{32} &= \alpha^{15}
\end{aligned}$$

The conjugates class of  $\alpha^{15}$  is  $\{\alpha^{23}, \alpha^{27}, \alpha^{29}, \alpha^{30}\}$ . All the conjugate classes are depicted in Table 3.2

Table 3.2 Conjugate Elements in  $GF(2^5)$ .

1
$\alpha \alpha^2 \alpha^4 \alpha^8 \alpha^{16}$
$\alpha^3 \alpha^6 \alpha^{12} \alpha^{17} \alpha^{24}$
$\alpha^5 \alpha^9 \alpha^{10} \alpha^{18} \alpha^{20}$
$\alpha^7 \alpha^{14} \alpha^{19} \alpha^{25} \alpha^{28}$
$\alpha^{11} \alpha^{13} \alpha^{21} \alpha^{22} \alpha^{26}$
$\alpha^{15} \alpha^{23} \alpha^{27} \alpha^{29} \alpha^{30}$

One of the properties of conjugates is that they provide a mechanism for going from an extension field to its base field. Consider the pair of complex conjugates  $z = p + jq$  and  $z^* = p - jq$ , their product gives the real number

$$zz^* = p^2 + q^2$$

Taking the product of the two factors  $(x - z)$  and  $(x - z^*)$  likewise gives a real expression

$$(x - z)(x - z^*) = x^2 - 2px + p^2 + q^2.$$

In finite field sets of conjugate elements perform the same operation. Consider  $\alpha^3$ ; belonging to  $GF(2^5)$ , and its conjugates  $\alpha^6, \alpha^{12}, \alpha^{17}$  and  $\alpha^{24}$ , let

$$M(x) = (x + \alpha^3)(x + \alpha^6)(x + \alpha^{12})(x + \alpha^{17})(x + \alpha^{24})$$

Performing the multiplication on the right side it is obtained that;

$$\begin{aligned}
& (x+\alpha^3)(x+\alpha^6)(x+\alpha^{12})(x+\alpha^{17})(x+\alpha^{24}) \\
&= [x^2+x(\alpha^6+\alpha^3)+\alpha^9][x^2+x\alpha^{14}+\alpha^{29}](x+\alpha^{24}) \\
&= [x^4+x^3\alpha^{14}+x^2\alpha^{29}+x^3\alpha+x^2\alpha^{15}+x\alpha^{30}+x^2\alpha^9+x\alpha^{23}+\alpha^7](x+\alpha^{24}) \\
&= [x^4+x^3(\alpha^{14}+\alpha)+x^2(\alpha^{29}+\alpha^{15}+\alpha^9)+x(\alpha^{30}+\alpha^{23})+\alpha^7](x+\alpha^{24}) \\
&\quad \quad \quad \downarrow \quad \quad \quad \downarrow \quad \quad \quad \downarrow \\
&\quad \quad \quad \alpha^{15} \quad \quad \quad \alpha^{20} \quad \quad \quad \alpha^{14}
\end{aligned}$$

we use table 3.1 for addition  
of  $\alpha$  in  $GF(2^5)$

$$\begin{aligned}
&= x^5 + x^4\alpha^{24} + x^4(\alpha^{14} + \alpha) + x^3(\alpha^{14} + \alpha)\alpha^{24} + x^3(\alpha^{29} + \alpha^{15} + \alpha^9) + x^2(\alpha^{29} + \alpha^{15} + \alpha^9)\alpha^{24} \\
&+ x^2(\alpha^{30} + \alpha^{23}) + x(\alpha^{30} + \alpha^{23})\alpha^{24} + x\alpha^7 + 1 \\
&= x^5 + x^4(\alpha^{24} + \alpha^{15}) + x^3(\alpha^8 + \alpha^{20}) + x^2(\alpha^{13} + \alpha^{14}) + x(\alpha^7 + \alpha^7) + 1 \\
&\quad \quad \quad \downarrow \quad \quad \quad \downarrow \quad \quad \quad \downarrow \quad \quad \quad \downarrow \\
&\quad \quad \quad 1 \quad \quad \quad 1 \quad \quad \quad 1 \quad \quad \quad 0
\end{aligned}$$

$$M(x) = x^5 + x^4 + x^3 + x^2 + 1$$

Hence,  $M(x) = x^5 + x^4 + x^3 + x^2 + 1$  which is referred to as the *minimal polynomial* of conjugate class of  $\{\alpha^3, \alpha^6, \alpha^{12}, \alpha^{17}$  and  $x^{24}\}$ . It is the binary polynomial of smallest degree that has  $\alpha^3, \alpha^6, \alpha^{12}, \alpha^{17}$  and  $\alpha^{24}$  as roots. Let  $M_i(x)$  denote the minimal polynomial of  $\alpha^i$ , then  $M_i(x)$  is defined to be the smallest degree polynomial in  $GF(2)$  that has  $\alpha^i$  as a root, and; so,

$$M_i(\alpha^i) = 0. \tag{3.12}$$

The minimal polynomial  $M_i(x)$  is also the minimal polynomial of the conjugates of  $\alpha^i$  and; therefore,

$$M_3(x) = M_6(x) = M_{12}(x) = M_{17}(x) = M_{24}(x) = x^5 + x^4 + x^3 + x^2 + 1. \tag{3.13}$$

In a similar manner  $M_1(x), M_2(x), M_4(x), M_8(x)$  and  $M_{16}(x)$  can be calculated as;

$$x^5 + x^2 + 1 \quad . \quad (3.14)$$

Using the minimal polynomial, generator polynomial of  $t$ -error-correcting binary  $BCH$  code can be computed as;

$$g(x) = LCM [M_1(x), M_2(x), M_3(x), \dots, M_{2t}(x)] \quad . \quad (3.15)$$

where LCM is the *Least Common Multiple (LCM)* operation. For the  $BCH$  (31,21) double-error-correcting  $t=2$  code with block length  $n=31$  over  $GF(2^5)$ . The generator polynomial is;

$$g(x) = LCM [M_1(x), M_2(x), M_3(x), M_4(x)] \quad . \quad (3.16)$$

Where,

$$\begin{aligned} M_1(x) &= x^5 + x^2 + 1 \\ M_2(x) &= M_1(x) \\ M_3(x) &= x^5 + x^4 + x^3 + x^2 + 1 \\ M_4(x) &= M_2(x) \quad . \end{aligned}$$

Hence,

$$\begin{aligned} g(x) &= M_1(x)M_3(x) \\ &= (x^5 + x^2 + 1)(x^5 + x^4 + x^3 + x^2 + 1) \\ &= x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1 \quad . \end{aligned}$$

A  $t$ -error-correcting  $BCH$  code has guaranteed minimum distance of  $d = 2t + 1$ . Therefore this is the (31,21,5) double-error-correcting binary  $BCH$  code.

Since  $BCH$  Codes are cyclic codes, their encoders can be implemented using shift-register circuits [3,9]. The codes can be encoded either systematically or non-systematically. Systematic codes perform slightly better than their non-systematic counterparts. For systematic codes, the generator polynomial  $g(x)$  is written as follows;



$$g(x) = g_0 + g_1x + g_2x^2 + \dots + g_{n-k-1}x^{n-k-1} + g_{n-k}x^{n-k} . \quad (3.17)$$

The generator polynomial  $g(x)$  formulates  $n$  codeword bits by appending  $(n-k)$  parity bits to the  $k$  information data bits. The encoder employs a shift register having  $(n-k)$  stages as depicted in Fig.3.1, where  $\otimes$  represents multiplication and  $\oplus$  is modulo-2 addition. The parity bits are computed from the information bits according to the rules imposed by the generator polynomial [3,9].

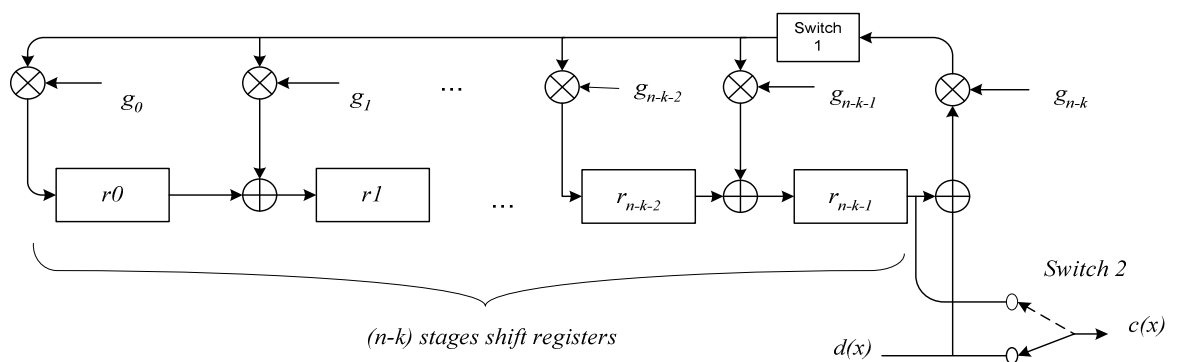


Figure 3.1 Systematic Encoder for *BCH* Codes Having  $(n-k)$  Shift-Register with  $(n-k)$  Cells

The following steps describe the systematic procedures [9]:

- 1) *Switch1* is closed during the first  $k$  shifts, in order to allow the information data bits,  $d(x)$  to shift into the  $n-k$  stages of the shift register.
- 2) At the same time, *Switch 2* is in the down position to allow the data bits,  $d(x)$ , to be copied directly to the codeword,  $c(x)$ .
- 3) After  $k^{\text{th}}$  shifts, *Switch 1* is opened and *Switch 2* is moved to the upper position.
- 4) The remaining  $n-k$  shifts clear the shift register by appending the parity bits to the codeword,  $c(x)$ .

Fig.3.2 shows the specific encoder, which is a derivative of Fig.3.1. Observe that all the multipliers illustrated in Fig.3.1 are absent in Fig.3.2. Explicitly, if the generator polynomial coefficient is 1, the multiplier is replaced by a direct

hard-wire connection as shown in Fig.3.2, whereas if the coefficient is 0, no connection is made.

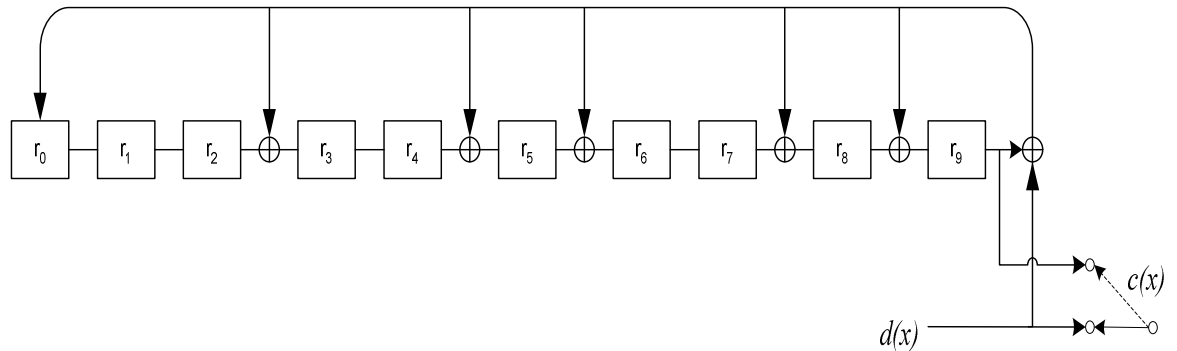


Figure 3.2 Systematic Encoder for  $BCH(31,21)$  Code Having  $n - k = 10$  Register Stages

The shift registers must be reset to zero before the encoding process starts. After the  $k$  (information bits) shift, Switch 1 is opened and Switch 2 is moved to the upper position. The parity bits contained in the shift register are appended to the codeword. Let's consider an example to be more specific.

### Example 3.1

Consider the  $BCH(7,4,3)$  code. The generator polynomial is  $g(x) = x^3 + x^2 + 1$

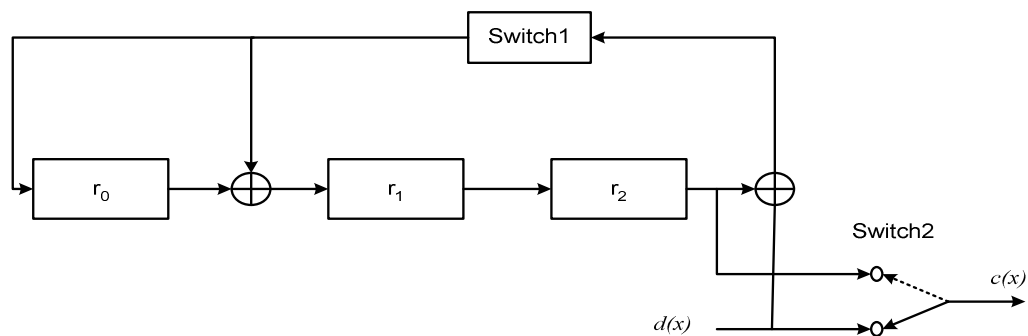


Figure 3.3 Systematic Encoder for  $BCH(7,4,3)$

Let use the shift register shown in Figure 3.3 for encoding four ( $k=4$ ) information data bits,  $d=1011$   $d(x)=1+x^2+x^3$ . The operational steps are given in Table 3.3;

Table 3.3 Operation Steps of *BCH (7,4,3) Encoder*

<i>Input queue</i>	<i>Shift index</i>	<i>Shift register</i> $r_0r_1r_2$	<i>Codeword</i> $c_0c_1c_2c_3c_4c_5c_6$
1011	0	000	-----
101	1	110	-----1
10	2	101	----- 11
1	3	100	----- 011
-	4 *	100	--- 1011
-	5	010	--0 1 011
-	6	001	-0 0 1011
-	7	000	1 0 0 1011

The codeword is  $c=1001011$ . The binary representation of both  $d(x)$  and  $c(x)$  is shown in Figure 3.4.

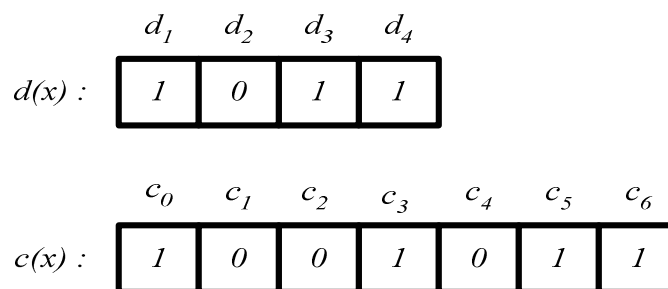


Figure 3.4 Binary Representations of the Encoded Data Bits and Code Bits

In the example above, there are a few points worth noting [9]:

- The encoding process always starts at the all-zero state and ends at the all zero state.
- The number of the output bits is always one following a clock pulse.

- For the first  $k$  (which is four for this example) shifts, the output bit is the same as the input bit.
- After the  $k^{\text{th}}$  shift, the parity bits of the shift register are shifted to the output.
- The number of states is equal to  $2^{n-k}$  increasing exponentially,  $n-k$  increases.

By this information, it can be introduced that the Trellis structure of *Block* code to use in *Viterbi* algorithm. At the next topic, this option will be the focus point.

### 3.2 Trellis Decoding of Block Codes

For the previous example for the *BCH* (7,4,3) code,  $n-k=3$  and the total number of encoder states is  $2^3=8$ . By using the shift register shown in Figure 3.2, it can be found that all the subsequent states when the register is in a particular state. Figure 3.5 shows all possible state transitions at any encoder state for the *BCH*(7,4,3) code [9,22].

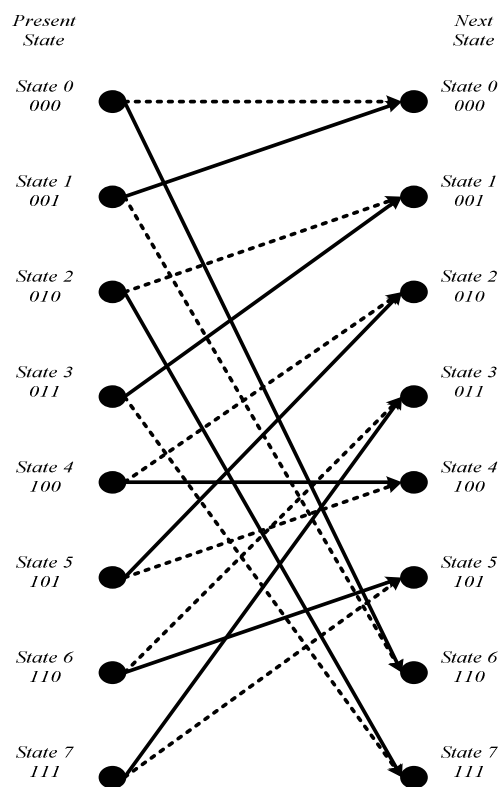


Figure 3.5 State Transition Diagram for *BCH* (7,4,3) .

The branch emanating from the present state to the next state indicates the state transition. The broken line branch is the transition initiated by a data bit of logical 0, whereas the continuous branch is due to the data bit being logical 1. The number of branches emanating from the present state is 2, which corresponds to the number of possible input bits. State transition diagram can also be illustrated via state diagram the state diagram of Figure 3.5 is shown in Figure 3.6. By using the state diagram in Figure 3.6, data bits  $d = 1011$  can be encoded, without using the shift register shown in Figure 3.3. The first data bit is a logical 1, hence the state changes from 000 to 110, as illustrated by the solid branch emanating from state 000 in Figure 3.6. The encoder output is the same as the input data bit, which is a logical 1. At the next instant, the present state becomes 110 and the data bit is logical 1. This causes the state transition from 110 to 101. The encoding cycle is repeated for subsequent data bits, which change the states. By following the change of states throughout the first  $k$  cycles of the encoding process, a particular path associated with states  $000 \rightarrow 110 \rightarrow 101 \rightarrow 100 \rightarrow 100$  can be observed [9, 22].

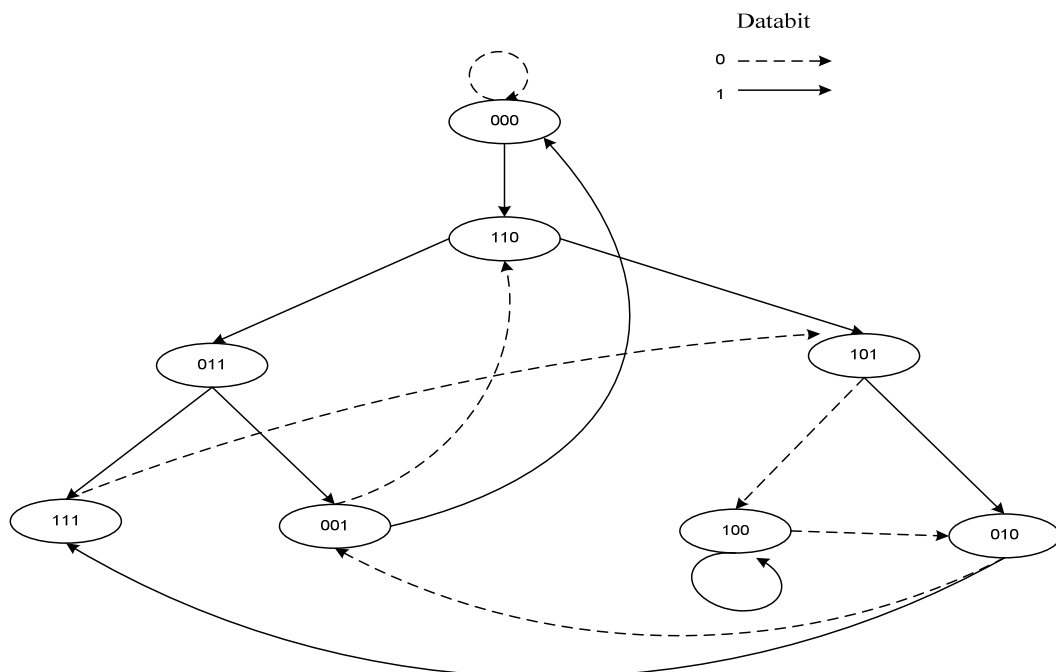


Figure 3.6 State Diagram for the  $BCH (7,4,3)$  code

After the  $k^{th}$  cycle, the state changes correspond to shifting out the parity bits from the shift register. In our example, the parity bits are 100 at the  $k^{th}$  cycle. In the following cycle, the parity bits are shifted to the right. The rightmost bit of the parity bits is shifted out to become the output bit and the leftmost bit is filled with logical 0. As a result, the state changes are  $100 \rightarrow 010 \rightarrow 001 \rightarrow 000$ . The whole encoding process can be associated with state transitions of  $000 \rightarrow 110 \rightarrow 101 \rightarrow 100 \rightarrow 100 \rightarrow 010 \rightarrow 001 \rightarrow 000$ .

Notice that if the binary value of the state number becomes input, the parity bits provide by the final  $n-k$  bits of the codeword. Therefore, it can be found that all of the *BCH (7,4,3)* code's states with the *MATLAB* code in Appendix A. Also we can use this code for *BCH (31,21,5)* code which has  $2^{n-k} = 2^{31-21} = 2^{10} = 1024$  states. This *MATLAB* code is written for our *BCH (31,21,5)* which will be used in this thesis.

Using state transition diagram of Figure 3.5 trellis diagram of *BCH (7,4,3)* can be obtained as in Figure 3.7.

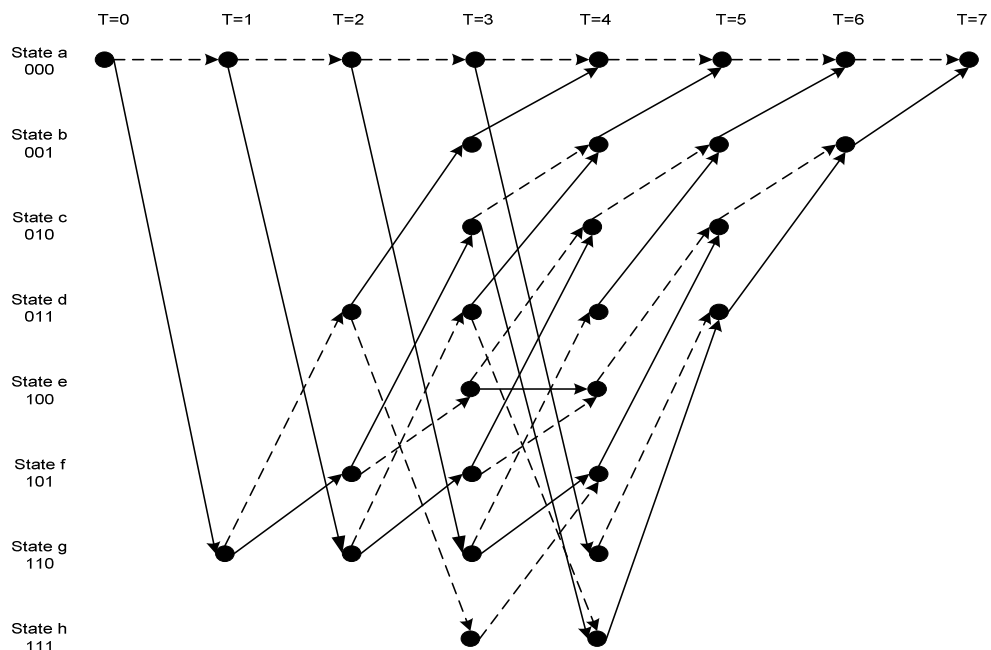


Figure 3.7 Trellis Diagram for the BCH (7,4,3) code

Thus, both *Convolutional* and *Block Codes* have a Trellis Diagram representation. It can be constructed thatf turbo codes whose constituent codes are *BCH* codes. Block turbo encoder and decoder structures constructed with *BCH* codes are shown in Figures 3.8 and 3.9.

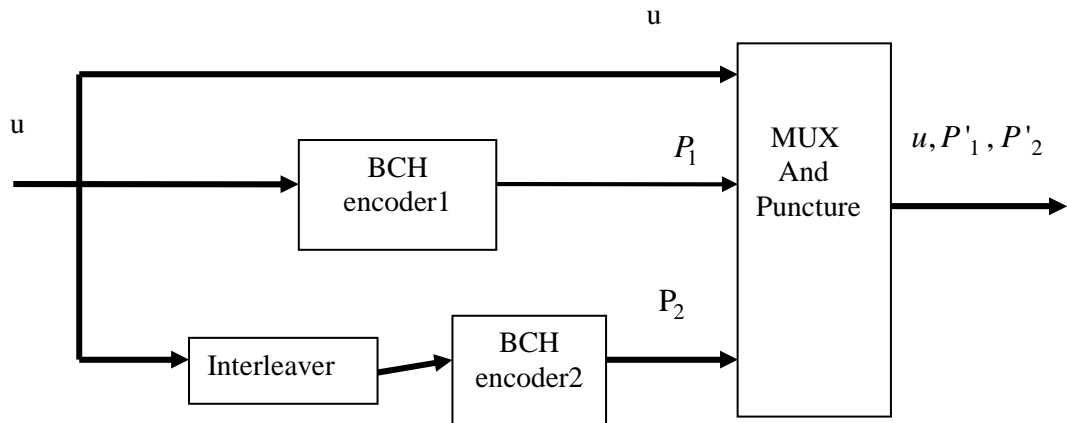


Figure 3.8 *BCH* Block Turbo Encoder Schematic

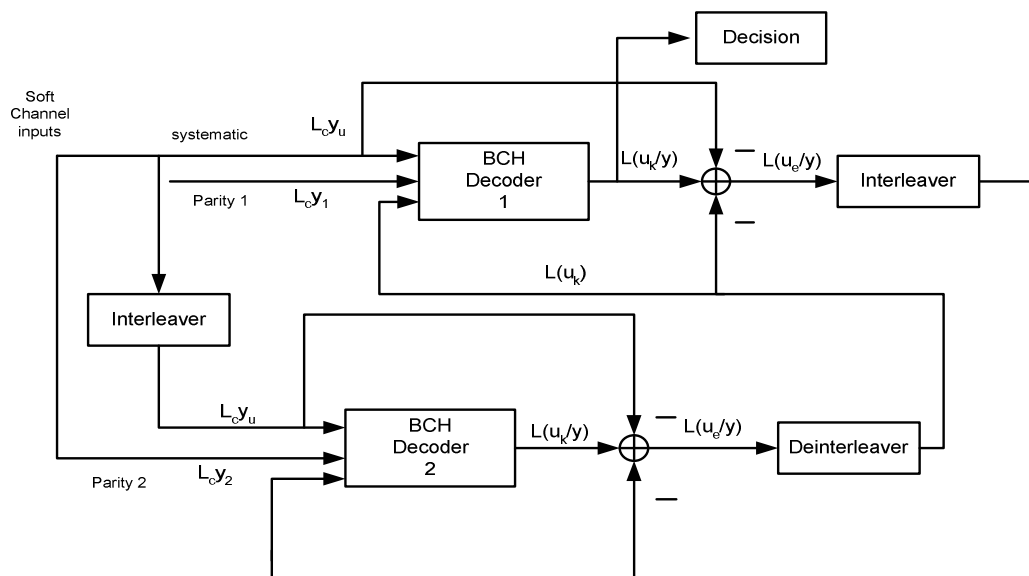


Figure 3.9 *BCH* Block Turbo Decoder Schematic

This chapter presented the idea of *Block Turbo Codes* including the way they are encoded and decoded.

### 3.3 Simulation Results

*BCH (31,21)* codes was used as constituent codes in turbo code encoder. The simulation results are depicted in Figure 3.10. The rate of block turbo code is  $R_c = 0.68$ . The number of iterations is limited to 8. AWGN channel is used. It can be clearly depicted from Figure 3.10 that iterative decoding achieves more than 4.5 dB gain when compared to non-iterative Hard-Viterbi decoding and 2.5 dB gain when compared to non-iterative Soft-Viterbi decoding.

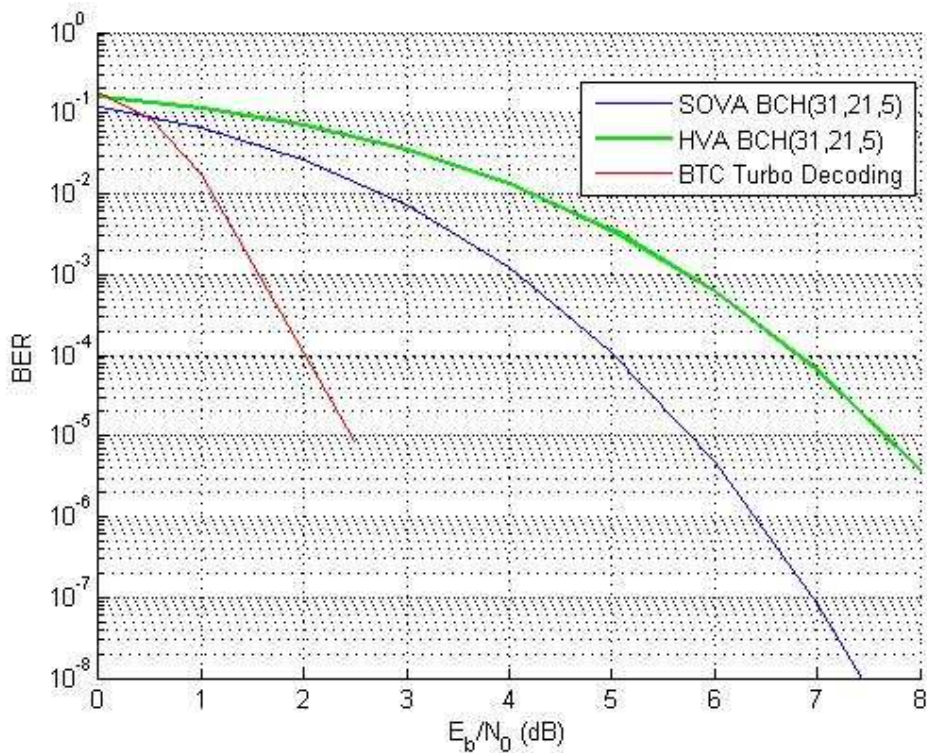


Figure 3.10 Simulation Results of *BCH (31,21)* Block Turbo Code



### BER against $E_b/N_0$

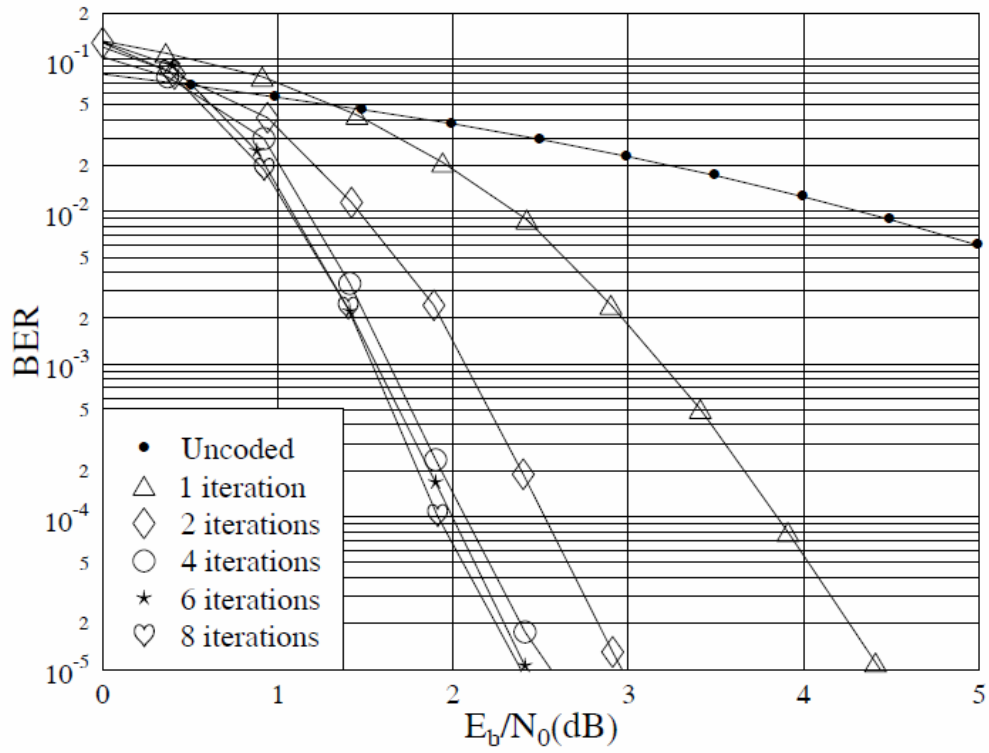


Figure 3.11 Performance of Different Number of Iterations in Turbo BCH (31,21) Code

## CHAPTER 4

### CONCLUSION

This master thesis reports iterative decoding method on block codes. In Chapter 2 the focus point was on those convolutional codes which are main component of turbo encoder. After that the *Viterbi* Algorithm which is the most important topic to understand the Hard-Decision, Soft-Decision and MAP Algorithms are considered. Then they were compared with each other. It was seen that Soft-Decision decoding is 2 dB better than Hard-Decision decoding. After that MAP Decoding was explained. With the modified Log-Map Algorithm iteration decoding of turbo codes are considered next. The simulation results are depicted in Figure 2.16 where it was clear that iterative decoding of convolutional Turbo Codes achieves more than 4.5 dB gain than hard or soft Viterbi decoding methods.

Turbo decoder is about four times as complex as decoding as the same code using a standard Viterbi Algorithm. Because in MAP Algorithm  $\alpha$ ,  $\beta$ ,  $\gamma$  and  $P$  are calculated but in Viterbi algorithm only the distance is considered.

One of the other parameter that affects the performance of Turbo codes is the number of decoding iteration. If the number of iterations used by the Turbo Decoder increases, the Turbo Decoder performs significantly better. However, after eight iterations there is little improvement achieved by using further iterations. For example, it can be seen from Figure 2.17 that using 16 iterations rather than eight gives an improvement of only about 0.1 dB. Again there is little improvement in BER performance of the decoder from using more than eight iterations. Hence for complexity reasons usually only between about 4 and 12 iterations are used.

In the original paper on turbo coding by Berrou et al. [11], and in many of the subsequent papers, impressive results have been presented for coding with very large frame lengths. However, for many applications the large delays inherent in using high frame lengths are unacceptable. Therefore an important area of turbo coding research is achieving as impressive results with short frame lengths like had demonstrated for long frame length systems. Figure 2.18 shows how the performance of turbo codes depends on the frame length  $L$  used in the encoder.

The other parameter is the generator polynomials and constraint lengths of the component codes. The standard RSC component codes are constraint length  $K=3$  codes with generator polynomials  $G_0=7$  and  $G_1=5$  in octal representation which are shown in Fig 2.1. These generator polynomials are optimum in terms of maximizing the minimum free distance of the component codes [23].

In Chapter 3, *Block Codes* were described. The same decoding method was used which was referred before as Chapter 2. Then the block turbo code was constructed and decoded with iterative decoding principle. The simulation results are depicted in Figure 3.10 where it is seen that the advantages of block turbo codes as compared to their counterparts' *Convolutional Turbo Codes* are their high code rate  $R = \frac{k}{2n-k} = \frac{21}{41} = 0.51$  and short frame length

to be decoded with high performance. Short frame length reduces the decoding latency and these results in increased throughput which is a critical issue for high speed communication which is essential especially for multimedia communication, i.e., video, image, music, speed, data, etc. The disadvantages of block turbo code are the large number of the states. For *BCH (31,21)* code the number of states is 1024. So, it increases the complexity of the Viterbi Algorithm but this situation may create a new workspaces. In *Convolutional Turbo Code*, the complexity is  $4 \times 4L \times 1024$  and in block turbo code the complexity is  $1024 \times 4L \times 31$ . The *Block Turbo Code* is about 8 times complex than the *Convolutional Turbo Code*.

One of the other parameter that affects the performance of *BCH Turbo Code* is the number of decoding iteration. As it can be seen in Figure 3.11, optimum iteration number is 8. Other parameter is the frame length while the 1024-bit code would be suitable for video transmission. The larger frame length systems would be useful in data or non-real-time transmission systems.

Finally, *Convolutional* and *Block Turbo Codes* are constructed. *Turbo Codes* are compared with the other coding methods. The main goal is to constitute the Trellis form of the *Block Codes* and applying the *Viterbi Algorithm* for iterative decoding.

## REFERENCES

- [1] **Glavieux, A.** (2007), *Channel Coding in Communication Networks*, Iste Ltd.
- [2] **Moreira, J.C., Farrell, P.G.** (2006), *Essentials of Error-Control Coding*, Wiley & Sons.
- [3] **Lin, S., Costello, D.J.** (1983), *Error Control Coding: Fundamentals and Applications*, Prentice-Hall.
- [4] **Neubauer, A., et. al.** (2007), *Coding Theory Algorithm, Architectures and Applications*.
- [5] **Hoffman, D.G. et. al.** (1991), *Coding Theory the Essentials*, Marcel Dekker.
- [6] **Simon, M.K., Smith J.G.** *Alternate Symbol Inversion for Improved Symbol Synchronization in Convolutionally Coded Systems*, IEEE Trans. Commun., Com-28, pp. 228-237, February 1980.
- [7] **Purser, M.** (1995), *Introduction the Error Control Codes*, Artech House.
- [8] **Proakis, J.G., Salehi, M.** (2002), *Communication System Engineering*, Prentice Hall.
- [9] **Hanzo, L., et. al.** (2002), *Turbo Coding, Turbo Equalization and Space-Time Coding for Transmission over Wireless Channels*, Wiley & Sons.
- [10] **Robert, H., Zaragoza, M.** (2002), *the Art of Error Correcting Codes*, Wiley & Sons.
- [11] **Berrou, C., et. al.** *Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes*, Proc. 1993, IEEE International Conference on Communications, Geneva, Switzerland, pp. 1064-1070, May 1993.
- [12] **Hagenauer, J., et. al.** *Iterative Decoding of Binary Block and Convolutional Codes*, IEEE Trans. Inform. Theory, Vol. 42, No: 2, March 1996, pp. 429-445.
- [13] **Soleymani, M.R., et. al.** (2002), *Turbo Coding for Satellite and Wireless Communication*, Kluwer Academic Publisher.
- [14] **Heegard, C., Wiker, S.** *Turbo Coding*, Kluwer, Massachusetts, 1999.
- [15] **Bahl, L., et. al.** *Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate*, IEEE Trans. Inf. Theory, Vol. I.T-20, pp. 284-287, March 1974.

- [16] **Cain, J.B.**, et. al. *Punctured Convolutional Codes of Rate  $(n-1)/n$  and Simplified Maximum Likelihood Decoding*, IEEE Trans.Inf.Theory, Vol. IT-25, pp. 97-100, January 1979.
- [17] **Huffman, W.C., Pless, V.** (2003), *Fundamental of Error Correcting Codes*, Cambridge Uni. Press.
- [18] **M.R. Soleymani, Yingzi Gao, U. Vilaipornsawai, (2002)**, *Turbo Coding For Satellite and Wireless Communcation*, Kluwer Academic Publisher.
- [19] **Gravano, S.** (2001), *Introduction to Error Control Codes*, Oxford.
- [20] **Ling, S., Xing, C.** (2004), *Coding Theory A First Course*, Cambridge Uni. Press.
- [21] **Stephan, B.W., Kim, S.** (2003), *Fundamentals of Codes, Graphs and Iterative Decoding*, Kluwer Academic Publisher.
- [22] **Schreier, P.J.** (1999), *Iterative Decoding of Parallel Concatenated High Rate Linear Block Codes*, University of Notre Dame, Indiana.
- [23] **Steele, R., Hanzo, L.** (1999), eds., *Mobile Radio Communications*, Ch. 4.4.4, pp. 425-428. Piscataway, NJ, USA: IEEE Press and Pentech Press.

## APPENDIX A

### STATE OF THE BCH(31,21) CODE

**x=0:1023;** → decimal value of 1024 state

**y=de2bi(x,21,'left-msb');** → binary representation of state numbers

**msg=gf(y);** → generates information bits in  $GF(2)$

**codeword=bchenc(msg,31,21);** → bchen is the  $BCH(31,21)$  encoder

**states=codeword(512:1024,22:31)** → it represents the final  $n-k=10$   
bits which is the states of the  
 $BCH(31,21,5)$  code

## Appendix B

### TURBO CODE

```
public class AWGNGenerator
{
    Random rn1=new Random();
    private int n=3080;
    private double noiseArray[]=new double[n];
    private double uA=0;
    private double uB=0;
    private double s=0;

    public void noiseOlustur(double sigmasquare){

        for(int i=0;i<n;i++){
            s=1;
            while(s>=1){
                uA=1-2*rn1.nextDouble();
                uB=1-2*rn1.nextDouble();
                s=uA*uA+uB*uB;
            }
            noiseArray[i]=uA*Math.sqrt((-
2*sigmasquare*Math.log(s))/s);
        }
        public double getNoise(int i){
            return noiseArray[i];
        }
    }
}

public class BPSKModulator{
    public double Mapping(int x){
        if(x==0)
            return -1;
        else
            return 1;
    }
}
```



```

    }

    public int Demapping(double y){
        if(y>0)
            return 1;
        else
            return 0;
    }
}

```

```

import java.util.Random;
public class encoder
{
    private int state1=0;
    private int state2=0;
    private int A=0;
    private int c1=0;
    private int c2=0;
    private int input[]=new int[1024];
    private int code[]=new int[2052];
    private int count=0;
    public void encode()
    {
        Random sec=new Random();/**/
        for(int i=0;i<1024;i++)
        {
            input[i]=sec.nextInt(2);/**/

            A=makebinary(state1,state2);
            c1=input[i];
            c2=makebinary(makebinary(input[i],state2),A);
            code[count++]=c1;
            code[count++]=c2;
            state2=state1;
            state1=makebinary(A,input[i]);
        }
    }
}

```

```

/*
*Trellis termination
*/
    if(state1==0&&state2==0){
        code[2048]=0;code[2049]=0;
        code[2050]=0;code[2051]=0;
    }
    else if(state1==0&&state2==1){
        code[2048]=1;code[2049]=1;
        code[2050]=0;code[2051]=0;
    }
    else if(state1==1&&state2==0){
        code[2048]=1;code[2049]=0;
        code[2050]=1;code[2051]=1;
    }
    else if(state1==1&&state2==1){
        code[2048]=0;code[2049]=1;
        code[2050]=1;code[2051]=1;
    }
}
}
public int makebinary(int x,int y)
{
    int z=x+y;
    if(z==0||z==2)
        return 0;
    else
        return 1;
}
public int[] getInput()
{
    return input;
}
public int[] getCode()
{
    return code;
}
}

```

```

public class Inter_encoder
{
    okuyucu oku=new okuyucu();
    private int state1=0;
    private int state2=0;
    private int A=0;
    private int c1=0;
    private int c2=0;
    private int input[]=new int[1024];
    private int Inter[]=new int[1024];
    private int Inter_input[]=new int[1024];
    private int code[]=new int[2052];
    private int count=0;

    public void InterlieverDefinition(){
        oku.openFile("Interliever.txt");
        oku.readRecord();
        Inter=oku.getRecords();
        oku.closeFile();
        for(int i=0;i<1024;i++){
            Inter_input[i]=input[Inter[i]];
        }
    }
}

public class TurboEncoder{
    private int[] Turbocode=new int[3080];
    private int[] input=new int[1024];
    public void MakeAll(){
        encoder encoder1=new encoder();
        Inter_encoder IE=new Inter_encoder();
        encoder1.encode();
        input=encoder1.getInput();
        IE.setInput(input);
        IE.InterlieverDefinition();
    }
}

```

```

        IE.encode();
        int[] code1=encoder1.getCode();
        int[] code2=IE.getCode();

        for(int i=0;i<1024;i++){
            Turbocode[3*i]=code1[2*i];
            Turbocode[3*i+1]=code1[2*i+1];
            Turbocode[3*i+2]=code2[2*i+1];
        }
        Turbocode[3072]=code1[2048];
        Turbocode[3073]=code1[2049];
        Turbocode[3074]=code1[2050];
        Turbocode[3075]=code1[2051];
        Turbocode[3076]=code2[2048];
        Turbocode[3077]=code2[2049];
        Turbocode[3078]=code2[2050];
        Turbocode[3079]=code2[2051];
    }
    public int[] getTurbocode(){
        return Turbocode;
    }
    public int[] getInput(){
        return input;
    }
}

```

```

public class MAP1
{
    private double sigmasqr=1;
    private double[] CodeArray=new double[2052];
    private int[] uArray=new int[1026];
    private double[] prioriP0=new double[1026];
    private double[] prioriP1=new double[1026];
    private double[] a_posPr0=new double[1026];
    private double[] a_posPr1=new double[1026];
    private double[] alfa00=new double[1026];
    private double[] alfa01=new double[1026];

```

```

private double[] alfa10=new double[1026];
private double[] alfa11=new double[1026];
private double[] beta00=new double[1027];
private double[] beta01=new double[1027];
private double[] beta10=new double[1027];
private double[] beta11=new double[1027];
private double[] branch0_0=new double[1026];
private double[] branch0_2=new double[1026];
private double[] branch1_0=new double[1026];
private double[] branch1_2=new double[1026];
private double[] branch2_1=new double[1026];
private double[] branch2_3=new double[1026];
private double[] branch3_3=new double[1026];
private double[] branch3_1=new double[1026];

public void setCodes( double[] car){
    CodeArray=car;
    for(int i=0;i<1026;i++){
        prioriP0[i]=Math.log( 0.5 );
        prioriP1[i]=Math.log( 0.5 );
    }
}

public int[] getCikis(){
    return uArray;
}

public void setsigma(double sigma){
    sigmasqr=sigma;
}

public void setPrioriPr0(double[] xxx){
    for(int i=0;i<1024;i++)
        prioriP0[i]=xxx[i];
}

public void setPrioriP1(double[] xxx){
    for(int i=0;i<1024;i++)
        prioriP1[i]=xxx[i];
}

public double[] getP0(){
    double[] P0=new double[1024];
    for(int i=0;i<1024;i++){
        P0[i]=a_posPr0[i]+CodeArray[2*i]/sigmasqr-prioriP0[i];
    }
    return P0;
}

```

```

    }
    public double[] getP1(){
        double[] P1=new double[1024];
        for(int i=0;i<1024;i++){
            P1[i]=a_posPr1[i]-CodeArray[2*i]/sigmasqr-prioriP1[i];
        }
        return P1;
    }

    public double[] getMAPP0(){

        return a_posPr0;
    }
    public double[] getMAPP1(){

        return a_posPr1;
    }

    public double takeDifference(double x1,double x2,double c1,double c2){
        return ( (x1*c1)+(x2*c2) )/(sigmasqr);
    }

    public void fillbranches(){
        for(int i=0;i<1026;i++){
            branch0_0[i]=prioriP0[i]+takeDifference(-1,-
1,CodeArray[2*i],CodeArray[2*i+1]);

            branch0_2[i]=prioriP1[i]+takeDifference(1,1,CodeArray[2*i],CodeArray[2*i
+1]);

            branch1_0[i]=prioriP1[i]+takeDifference(1,1,CodeArray[2*i],CodeArray[2*i
+1]);

            branch1_2[i]=prioriP0[i]+takeDifference(-1,-
1,CodeArray[2*i],CodeArray[2*i+1]);
            branch2_1[i]=prioriP1[i]+takeDifference(1,-
1,CodeArray[2*i],CodeArray[2*i+1]);
            branch2_3[i]=prioriP0[i]+takeDifference(-
1,1,CodeArray[2*i],CodeArray[2*i+1]);
            branch3_1[i]=prioriP0[i]+takeDifference(-
1,1,CodeArray[2*i],CodeArray[2*i+1]);
            branch3_3[i]=prioriP1[i]+takeDifference(1,-
1,CodeArray[2*i],CodeArray[2*i+1]);
        }
        branch1_0[0]=-100;
    }

```

```

        branch1_2[0]=-100;
        branch2_1[0]=-100;
        branch2_3[0]=-100;
        branch3_1[0]=-100;
        branch3_3[0]=-100;
        branch1_0[1]=-100;
        branch1_2[1]=-100;
        branch3_1[1]=-100;
        branch3_3[1]=-100;
    }
    public void fillalfas(){
        alfa00[0]=0;
        alfa01[0]=-100;
        alfa10[0]=-100;
        alfa11[0]=-100;
        alfa00[1]=alfa00[0]+branch0_0[0];
        alfa01[1]=-100;
        alfa10[1]=alfa00[0]+branch0_2[0];
        alfa11[1]=-100;
        alfa00[2]=alfa00[1]+branch0_0[1];
        alfa01[2]=alfa10[1]+branch2_1[1];
        alfa10[2]=alfa00[1]+branch0_2[1];
        alfa11[2]=alfa10[1]+branch2_3[1];
        for(int i=3;i<1026;i++){
            alfa00[i]=Math.log(Math.exp(alfa00[i-1]+branch0_0[i-1])+Math.exp(alfa01[i-1]+branch1_0[i-1]));
            alfa01[i]=Math.log(Math.exp(alfa10[i-1]+branch2_1[i-1])+Math.exp(alfa11[i-1]+branch3_1[i-1]));
            alfa10[i]=Math.log(Math.exp(alfa00[i-1]+branch0_2[i-1])+Math.exp(alfa01[i-1]+branch1_2[i-1]));
            alfa11[i]=Math.log(Math.exp(alfa10[i-1]+branch2_3[i-1])+Math.exp(alfa11[i-1]+branch3_3[i-1]));

//normalization of alfas
            double
            SS=Math.log(Math.exp(alfa00[i])+Math.exp(alfa01[i])+Math.exp(alfa10[i])+Math.exp(alfa11[i]));
            alfa00[i]=alfa00[i]-SS;
            alfa01[i]=alfa01[i]-SS;
            alfa10[i]=alfa10[i]-SS;
            alfa11[i]=alfa11[i]-SS;
        }
    }
    public void fillbetas(){

```

```

        beta00[1026]=0;
        beta01[1026]=-100;
        beta10[1026]=-100;
        beta11[1026]=-100;
        beta00[1025]=beta00[1026]+branch0_0[1025];
        beta01[1025]=beta00[1026]+branch1_0[1025];
        beta10[1025]=-100;
        beta11[1025]=-100;
        beta00[1024]=beta00[1025]+branch0_0[1024];
        beta01[1024]=beta00[1025]+branch1_0[1024];
        beta10[1024]=beta01[1025]+branch2_1[1024];
        beta11[1024]=beta01[1025]+branch3_1[1024];
        for(int i=1023;i>0;i--){

            beta00[i]=Math.log(Math.exp(beta00[i+1]+branch0_0[i])+Math.exp(beta10[
i+1]+branch0_2[i]));

            beta01[i]=Math.log(Math.exp(beta00[i+1]+branch1_0[i])+Math.exp(beta10[
i+1]+branch1_2[i]));

            beta10[i]=Math.log(Math.exp(beta01[i+1]+branch2_1[i])+Math.exp(beta11[
i+1]+branch2_3[i]));

            beta11[i]=Math.log(Math.exp(beta01[i+1]+branch3_1[i])+Math.exp(beta11[
i+1]+branch3_3[i]));

            // Normalization of betas
            double
            SS=Math.log(Math.exp(beta00[i])+Math.exp(beta01[i])+Math.exp(beta10[i])+Math.
exp(beta11[i]));
            beta00[i]=beta00[i]-SS;
            beta01[i]=beta01[i]-SS;
            beta10[i]=beta10[i]-SS;
            beta11[i]=beta11[i]-SS;

        }
    }
    public void decode(){
        fillbranches();
        fillalfas();
        fillbetas();
        for(int i=0;i<1026;i++){

```



```

        a_posPr1[i]=Math.log(Math.exp(alfa00[i]+beta10[i+1]+branch0_2[i])+
Math.exp(alfa01[i]+beta00[i+1]+branch1_0[i])+
Math.exp(alfa10[i]+beta01[i+1]+branch2_1[i])+
Math.exp(alfa11[i]+beta11[i+1]+branch3_3[i]));

        a_posPr0[i]=Math.log(Math.exp(alfa00[i]+beta00[i+1]+branch0_0[i])+
Math.exp(alfa01[i]+beta10[i+1]+branch1_2[i])+
Math.exp(alfa10[i]+beta11[i+1]+branch2_3[i])+
Math.exp(alfa11[i]+beta01[i+1]+branch3_1[i]));
    }
}
}

```

```

public class MAPFrame{
    private int errorcounter=0;
    private int frameerr=0;
    private int framecounter=0;
    private int temple=0;
    private double sigmasquare=0.5;
    public void yeniframe(){
        temple=0;
        TurboEncoder encoder1=new TurboEncoder();
        TurboDecoder decoder1=new TurboDecoder();
        BPSKModulator bpsk=new BPSKModulator();
        AWGNGenerator awgn=new AWGNGenerator();

        encoder1.MakeAll();
        int[] inputArray=encoder1.getInput();
        int[] codes=encoder1.getTurbocode();
        awgn.noiseOlustur(sigmasquare);

        double bpskcodes[]=new double[3080];
        for(int j=0;j<3080;j++){

```

```

        bpskcodes[j]=bpsk.Mapping(codes[j])+awgn.getNoise(j);
    }
    decoder1.setSigma(sigmasquare);
    decoder1.setCodes(bpskcodes);
    decoder1.decode();
    int cikis[]=decoder1.getCikis();
    for(int l=0;l<1024;l++){
        if(inputArray[l]!=cikis[l]){
            errorcounter++;
            temple++;
        }
    }
    if(temple!=0)
        frameerr++;
}
public void Simulation(double Sigma){
    sigmasquare=Sigma;
    errorcounter=0;
    framecounter=0;
    frameerr=0;
    while(frameerr<100)
    {
        yeniframe();
        framecounter++;
    }
}
public int getframecounter(){
    return framecounter;
}
public int geterrorcounter(){
    return errorcounter;
}
public int getframeerror(){
    return frameerr;
}
}

```

```

public class MAPFrameTest
{
    public static void main(String args[])
    {
        MAPFrame fatih=new MAPFrame();
    }
}

```

```

//0dB
fatih.Simulation(1.5);
System.out.printf("Eb/No:0dB(variance:1,500000)   HataliBit:%d
HataliFrame:%d
Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/

//0,2dB
fatih.Simulation(1.432489);
System.out.printf("Eb/No:0,2dB(variance:1,432489)   HataliBit:%d
HataliFrame:%d
Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/

//0,4dB
fatih.Simulation(1.368016);
System.out.printf("Eb/No:0,4dB(variance:1,368016)   HataliBit:%d
HataliFrame:%d
Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/

//0,6dB
fatih.Simulation( 1.306445 );
System.out.printf("Eb/No:0,6dB(variance:1,306445)   HataliBit:%d
HataliFrame:%d
Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/

//0,8dB
fatih.Simulation( 1.247646);
System.out.printf("Eb/No:0,8dB(variance:1,247646)   HataliBit:%d
HataliFrame:%d
Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/

//1dB
fatih.Simulation( 1.191492);
System.out.printf("Eb/No:1dB(variance:1,191492)   HataliBit:%d
HataliFrame:%d
Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/

//1,2dB
fatih.Simulation( 1.137866);
System.out.printf("Eb/No:1,2dB(variance:1,137866)   HataliBit:%d
HataliFrame:%d
Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/

//1,4dB
fatih.Simulation( 1.086654 );

```

```

        System.out.printf("Eb/No:1,4dB(variance:1,086654)  HataliBit:%d
HataliFrame:%d
Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/

        //1,6dB
        fatih.Simulation( 1.037746 );
        System.out.printf("Eb/No:1,6dB(variance:1,037746)  HataliBit:%d
HataliFrame:%d
Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/

        //1,8dB
        fatih.Simulation( 0.991040 );
        System.out.printf("Eb/No:1,8dB(variance:0,991040)  HataliBit:%d
HataliFrame:%d
Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/

        //2dB
        fatih.Simulation( 0.946436 );
        System.out.printf("Eb/No:2dB(variance:0,946436)  HataliBit:%d
HataliFrame:%d
Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/
/*
        //2,2dB
        fatih.Simulation( 0.903839 );
        System.out.printf("Eb/No:2,2dB(variance:0,903839)  HataliBit:%d
HataliFrame:%d
Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/
/*
        //2,4dB
        fatih.Simulation( 0.863160 );
        System.out.printf("Eb/No:2,4dB(variance:0,863160)  HataliBit:%d
HataliFrame:%d
Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/
/*
        //2,6dB
        fatih.Simulation( 0.824311 );
        System.out.printf("Eb/No:2,6dB(variance:0,824311)  HataliBit:%d
HataliFrame:%d
Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/

        //2,8dB
        fatih.Simulation( 0.787211 );
        System.out.printf("Eb/No:2,8dB(variance:0,787211)  HataliBit:%d
HataliFrame:%d

```

```

Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/
/*          //3dB
           fatih.Simulation( 0.751781 );
           System.out.printf("Eb/No:3dB(variance:0,751781)   HataliBit:%d
HataliFrame:%d
Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/
           //3,6dB
           fatih.Simulation( 0.654774 );
           System.out.printf("Eb/No:3,6dB(variance:0,654774)   HataliBit:%d
HataliFrame:%d
Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/
           //4dB
           fatih.Simulation( 0.597161 );
           System.out.printf("Eb/No:4dB(variance:0,597161 )   HataliBit:%d
HataliFrame:%d
Frame:%d\n",fatih.geterrorcounter(),fatih.getframeerror(),fatih.getframecounter());/
**/

    }
}

```

```

import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
import java.lang.IllegalStateException;
import java.util.NoSuchElementException;

```

```

public class okuyucu{
    private Scanner oku;
    private int[] records=new int[1024];
    private int rec=0;
    public void openFile(String name){
        try
        {
            oku=new Scanner(new File(name));
        }
        catch(FileNotFoundException fnfexc)
        {

```

```

        System.err.println("Error opening the file");
        System.exit(1);
    }
}
public void readRecord(){
    try{
        int i=0;
        while(oku.hasNextInt()){
            rec=oku.nextInt();
            records[i++]=rec;
            /*      System.out.printf("okunan bilgiler:\n%s",records);/**/
        }
    }
    catch(NoSuchElementException nseexc){
        System.err.println("File improperly formed");
        oku.close();
        System.exit(1);
    }
    catch(IllegalStateException isexc){
        System.err.println("Error reading from file");
        System.exit(1);
    }
}
public void closeFile()
{
    if(oku != null)
        oku.close();
}
public int[] getRecords(){
    return records;
}
}

```

## APPENDIX C

### CURRICULUM VITAE

#### PERSONAL INFORMATION

Surname, Name : Genç. Fatih  
Nationality : Turkish (TC)  
Date and Place of Birth : 1 January 1984, Ankara  
Marital Status : Single  
Phone : +90 312 437 29 97  
Email : fatihgenc84@gmail.com

#### EDUCATION

Degree	Institution	Year of Graduation
MS	Çankaya Univ. Electronic and Communication Eng.	2010
BS	Çankaya Univ. Electronic and Communication Eng.	2007
BS	Wien Technical Univ. ERASMUS Student Exchange Program (Austria)	2007-2006
High School	Muharrem Hasbi High School, Balıkesir	2002

#### WORK EXPERIENCE

Year	Place	Enrollment
2010-Present	Gazi University	Project Assistant
2009-2010	Çankaya Uni.	Specialist
2007-2008	Oriantel A.Ş Hoşdere	Ar-Ge Eng
2005-2006	Türksat Aş. CMC-Communication Monitoring Center	Summer Trainer
2004-2005	Türktelekom Aş. IT Networks	Summer Trainer

#### FOREIGN LANGUAGES

Advanced English, Basic German

## **HOBBIES**

Swimming, Basketball, Chess, Table Tennis, Watching Movies, Reading Books, Listening Music, Karting Paintball, Latin Dances

## **EXPERIENCE AND PROJECT**

Iterative Decoding of Block Coding (2010) (Master Thesis)

Digital Transmitter and Receiver Theory - Simulation Matlab - FPGA/VHDL (2007)(Senior Project)

Çankaya University Spring Festival (2007)

Karting Competition Activity (2007)

C++ Class Architecture (2007)

Seminar related with Object-Oriented Programming

Remote Controlled Car (2005) (Semester Project)

Token Ring Simulation (2005)

The Project of "Let's Go to School" (2004-2005)

## **TRAINING**

TOSFED Observer Training (2006)

Society Voluntaries White Key Volunteer Certificate (2006)

Society Voluntaries Green Key Training (2006)

Society Voluntaries Red Key Training (2005)

Society Voluntaries Blue Key Training (2005)

Society Voluntaries Yellow Key Training (2004)

## **MEMBER OF GROUPS AND CLUBS**

Present Member of Information Technologies Community of Çankaya University (2003)

Present Member of FRP Community of Çankaya University (2003)

Present Organization committee Go-Kart Community of Çankaya University (2004)



Present Incorporator of Society Voluntaries Community of Çankaya University (2004)

Present Head of Organization Committee and Incorporator of Chess Community of Çankaya University (2005)

Present Member of IEEE Community of Çankaya University (2007)

## **REFERENCES**

Assoc.Prof.Dr Celal Zaim Çil

Chairman of Electronic and Communication Eng. Çankaya University

Phone: (+90) 312 2844500 / 262

Assist. Prof. Dr. Orhan Gazi

Electronic and Communication Eng. Çankaya University

Phone: +90 535 379 08 25

Assist. Professor Dr. Özgür Ertuğ

Electronic and Communication Eng. Gazi University

Phone: +90 (312) 5823320