Performance of Error Correcting Codes for Next Generation Communication

Major Project Report

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics and Communication Engineering

(Communication Engineering)

By

Sachin R. Makavana

(08MECC09)

Department of Electronics & Communication Engineering,

Institute of Technology,

Nirma University,

Ahmedabad-382 481

May 2010

Performance of Error Correcting Codes for Next Generation Communication

Major Project Reprot

Submitted in partial fulfillment of the requirements

for the degree of

Master of Technology

in

Electronics and Communication Engineering

(Communication Engineering)

By

Sachin R. Makavana

(08MECC09)

Under the Guidance of

Prof. Dhaval Pujara

Department of Electronics & Communication Engineering,

Institute of Technology,

Nirma University,

Ahmedabad-382 481

May 2010

Declaration

This is to certify that

i) The thesis comprises my original work towards the degree of Master of Technology in Communication Engineering at Nirma University and has not been submitted elsewhere for a degree.

ii) Due acknowledgement has been made in the text to all other material used.

Sachin R. Makavana

Certificate

This is to certify that the Major Project entitled **"Performance of Error Correcting Codes for Next Generation Communication"** submitted by **Sachin R. Makavana (08MECC09)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Electronics & Communication Engineering (Communication) of Nirma University, Institute of Technology, Ahmedabad is the record of work carried out by him under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of our knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Ahmedabad

Internal Guide

(Prof. Dhaval Pujara)
Sr. Asso. Professor, EC Dept.

External Guide

(Mr. Ankush Shrivastava)
RF Manager
Nokia Siemens Network, Ahmedabad

HOD

(Prof. A. S. Ranade)
Professor, EC

Director

(Dr. K. Kotecha)
Director, IT, NU

Acknowledgements

I would like to express my gratitude and sincere thanks to Prof. A. S. Ranade, Head of Electrical Engineering Department and Dr. D. K. Kothari Coordinator-M.Tech. Communication Engineering program for allowing me to undertake this thesis work and for his guidelines during the review process.

I would like to express my gratitude and sincere thanks to Nokia Siemens Network for giving me an opportunity to work under to guidance of renowned people in the field of communications and for providing all the resources for the project development.

I would like to express my endless thanks to my internal guide Prof. Dhaval Pujara (Nirma University, Ahmedabad) and external guide Mr. Ankush Shrivastava (RF manager, Nokia Siemens Networks (NSN)) for their constant guidance and motivation. They devoted significant amount of valuable time for discussion and planning of the project work.

I would like to thank all the engineers and staff members of NSN, I also would like to thank all my classmate for their support and co-operation.

Last, but not the least, no words are enough to acknowledge constant support and sacrifices of my family members because of whom I am able to complete the degree program successfully.

- Sachin R. Makavana

08MECC09

Abstract

The error correcting codes have gained popularity due to its several well known properties, like the distance property and the block length. The convolution code and the turbo code are used as error correcting codes which are able to come very close to the theoretical limit, the Shannon's limit. The convolution code and the turbo codes are used in the 3G (third Generation) mobile technologies and the standards defined in the IEEE 802.16 (WiMax).

The thesis describes the evaluation and the performance of the convolution code based on the code rate, BER, Eb/No, and different modulation techniques. Secondly the study, analysis and implementation of the turbo codes with encoding and decoding methods have been carried out using AWGN channel with BPSK modulation technique. The turbo code encoder is built using a parallel concatenation of two recursive systematic convolution (RSC) codes with interleaver and the associated decoder, using Soft Output Viterbi Algorithm (SOVA).

Contents

Declaration iii

Certificate iv

Acknowledgements v

Abstract vi

List of Figures ix

List of Tables xi

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Representation of Digital Communication System . . . 2
- 1.3 Types of Channel Codes . . . 4
- 1.4 Problem Statement . . . 5
- 1.5 Outline of the Thesis . . . 6

2 Convolutional Codes 8
- 2.1 Introduction . . . 8
- 2.2 Encoder Structure . . . 8
- 2.3 Encoder Representations . . . 10
  - 2.3.1 Generator Representation . . . 10
  - 2.3.2 Trellis Diagram Representation . . . 12
  - 2.3.3 State Diagram Representation . . . 14
- 2.4 The Viterbi Algorithm . . . 15
- 2.5 Distance Properties of Convolutional Code . . . 17

3 Turbo Codes 18
- 3.1 Introduction . . . 18
- 3.2 Turbo Code Encoder . . . 19
  - 3.2.1 Recursive Systematic Convolutional (RSC) Encoder . . . 20
  - 3.2.2 Concatenation of Codes . . . 21

3.2.3 Interleaver Design 22

4 System Model for Turbo Code Decoder **26**
4.1 Principle of the Soft-Output Viterbi Decoder 26
4.2 Reliability of the General SOVA Decoder 27
4.3 Introduction to SOVA for Turbo Codes 33
4.3.1 Log-Likelihood Algebra 33
4.3.2 Soft Channel Outputs 37
4.4 SOVA Decoder for a Turbo Code 39
4.5 SOVA Iterative Turbo Code Decoder 47

5 Performance Analysis **50**
5.1 Simulation Setup 50
5.2 Simulation Results 51
5.3 Simulation Analysis 68

6 Conclusion and Future Scope **70**
6.1 Conclusion 70
6.2 Contribution 72
6.3 Future Scope 72

References **73**

List of Figures

1.1 Simplified models of digital communication system (a) Coding and modulation performed separately. (b) Coding and modulation combined [1]. 3

2.1 Convolutional encoder with k = 1, m = 2, R = 1/2 and K = 3 [2]. . . 9
2.2 Trellis for the convolutional encoder of Figure 2.1 [2]. 13
2.3 **(a)** A portion of the central part of the trellis for the encoder of Figure 2.1 **(b)** State diagram of the convolutional encoder of Figure 2.1 [2]. . 15

3.1 Fundamental turbo code encoder 19
3.2 The RSC encoder obtained from Figure 2.1 with R = 1/2 and K =3 [2]. 20
3.3 Serial concatenated code. 21
3.4 Parallel concatenated code. 22
3.5 The interleaver increases the code weight for RSC encoder 2 as compared to RSC encoder 1. 23
3.6 An illustrative example of an interleavers capability. 24
3.7 Block interleaver [2]. 25
3.8 Random interleaver with L=8 [6]. 25

4.1 A concatenated SOVA decoder where y represents the received channel values, u represents the hard decision output values, and L represents the associated reliability values. 27
4.2 Example of survivor and competing paths for reliability estimation at time t [11]. 28
4.3 Example that shows the weakness of reliability assignment using metric values directly [11]. 30
4.4 Updating process for time t-2 ($MEM_{low} = 2$) [11]. 32
4.5 Updating process for time t-4 ($MEM_{low} = 4$) [11]. 32
4.6 System model for SOVA derivation. 33
4.7 SOVA component decoder. 39
4.8 Source reliability for SOVA metric computation. 42
4.9 Example of SOVA survivor and competing paths for reliability estimation [12]. 44
4.10 SOVA iterative turbo code decoder [13]. 47

5.1 Convolutional code (rate (R)-1/2) with Different Modulation in AWGN channel 52
5.2 Convolutional code with Different Coding Rate in BPSK and AWGN channel 53
5.3 Performance of turbo code with different code rate 55
5.4 Performance of turbo code with different iteration 56
5.5 Performance of turbo code with different frame size 57
5.6 Performance of turbo code with different constraint length 58
5.7 Comparison of turbo code BER performance for (FS = 64, iter = 1, R = 1/2, K = 3) [14] 59
5.8 Comparison of turbo code BER performance for (FS = 192, iter = 1, R = 1/2, K = 5) [14] 60
5.9 Comparison of turbo code BER performance for (FS = 192, iter = 5, R = 1/2, K = 5) [14] 61
5.10 Comparison of turbo code BER performance for (FS = 2048, iter = 1, R = 1/2, K = 5) [14] 62
5.11 Comparison of turbo Code BER Performance for (FS = 2048, iter = 5, R = 1/2, K = 5) [14] 63
5.12 Comparison of turbo Code BER Performance for (FS = 1024, iter = 1, R = 1/2, K = 5) [15] 64
5.13 Comparison of turbo Code BER Performance for (FS = 1024, iter = 8, R = 1/2, K = 5) [15] 65
5.14 Comparison of turbo Code BER Performance for (FS = 1024, iter = 8, R = 1/2, K = 3) [16] 66
5.15 Comparison of turbo Code BER Performance for (FS = 1024, iter = 8, R = 1/2, K = 5) [16] 67

List of Tables

2.1 State table for convolutional encoder of Figure 2.1 13

3.1 Input and output sequences for encoder of Figure 3.8 24

4.1 Outcome of adding two binary random variables u_1 and u_2 33
4.2 Characteristics of the Log-likelihood Ratio L(u) 34

5.1 Result of convolutional code with BPSK modulation of Figure 5.1 . . 52
5.2 Result of convolutional code with QPSK modulation of Figure 5.1 . . 52
5.3 Result of convolutional code with 8-PSK modulation of Figure 5.1 . . 53
5.4 Result for Rate-$\frac{1}{2}$ soft decision convolutional code of Figure 5.2 53
5.5 Result for Rate-$\frac{1}{2}$ hard decision convolutional code of Figure 5.2 . . . 54
5.6 Result for Rate-$\frac{1}{3}$ soft decision convolutional code of Figure 5.2 54
5.7 Result for Rate-$\frac{2}{3}$ (with puncture) soft decision convolutional code of Figure 5.2 54
5.8 Comparison of turbo code BER performance for different rate (FS = 256, iter =1) 55
5.9 Comparison of turbo code BER performance for different iter (FS = 256, R = 1/3,K = 3) 56
5.10 Comparison of turbo code BER performance for different FS (iter = 8, R = 1/3,K = 3) 57
5.11 Comparison of turbo code BER performance for different constraint length (FS = 256, iter =1) 58
5.12 Comparison of turbo code BER performance for (FS = 64, iter = 1, R = 1/2, K = 3) [14] 59
5.13 Comparison of turbo code BER performance for (FS = 192, iter = 1, R = 1/2, K = 5) [14] 60
5.14 Comparison of turbo code BER performance for (FS = 192, iter = 5, R = 1/2, K = 5) [14] 61
5.15 Comparison of turbo code BER performance for (FS = 2048, iter = 1, R = 1/2, K = 5) [14] 62
5.16 Comparison of turbo Code BER Performance for (FS = 2048, iter = 5, R = 1/2, K = 5) [14] 63

5.17 Comparison of turbo Code BER Performance for (FS = 1024, iter = 1, R = 1/2, K = 5) [15] 64
5.18 Comparison of turbo Code BER Performance for (FS = 1024, iter = 8, R = 1/2, K = 5) [15] 65
5.19 Comparison of turbo Code BER Performance for (FS = 1024, iter = 8, R = 1/2, K = 3) [16] 66
5.20 Comparison of turbo Code BER Performance for (FS = 1024, iter = 8, R = 1/2, K = 5) [16] 67

Chapter 1

Introduction

1.1 Motivation

In recent years, there has been an increasing demand for efficient and reliable digital data transmission and storage systems. This demand has been accelerated by the emergence of large-scale, high-speed data networks for the exchange, processing, and storage of digital information in-the military, governmental, and private spheres. A merging of communications and computer technology is required in the design of these systems. The task facing the designer of a digital communication system is that of providing a cost-effective facility for transmitting information from one end of the system at a rate and a level of reliability and quality that are acceptable to a user at the other end. The two key system parameters available to the designer are transmitted signal power and channel bandwidth. These two parameters, together with the power spectral density of receiver noise, determine the signal energy per bit-to-noise power spectral density ratio Eb/No. This ratio uniquely determines the bit error rate for a particular modulation scheme [1] . Practical considerations usually place a limit on the value that we can assign to Eb/No. Accordingly, in practice, one can often arrive at a modulation scheme and find that it is not possible to provide acceptable data quality (i.e., low enough error performance). For a fixed Eb/No, the only practical

option available for changing data quality from problematic to acceptable is to use error-control coding.

The channel coding theorem states that if a discrete memoryless channel has capacity C and a source generates information at a rate R, less than C, then there exist a coding technique such that the output of the source may be transmitted over the channel with an arbitrary low probability of symbol error.(i.e., by proper encoding of the information, errors introduced by a noisy channel or storage medium can be reduced to any desired level without sacrificing the rate of information transmission or storage.) [1]. For the special case of a binary symmetric channel, the theorem tells us that if the code rate Ro, is less than the channel capacity C, then it is possible to find a code that achieves error-free transmission over the channel. Conversely, it is not possible to find such a code if the code rate Ro is greater than the channel capacity C.

Another practical motivation for the use of coding is to reduce the required Eb/No for a fixed bit error rate. This reduction in Eb/No may, in turn, be exploited to reduce the required transmitted power or reduce the hardware costs by requiring a smaller antenna size in the case of radio communications.

1.2 Representation of Digital Communication System

The transmission and storage of digital information have much in common as both transfer data from an information source to a destination (or user). Error control for data integrity may be exercised by means of forward error correction (FEC). Figure 1.1(a) shows the model of a digital communication system using such an approach. The discrete source generates information in the form of binary symbols. The channel encoder in the transmitter accepts message bits and adds redundancy according to

a prescribed rule, thereby producing encoded data at a higher bit rate. The channel decoder in the receiver exploits the redundancy to decide which message bits were actually transmitted. The combined goal of the channel encoder and decoder is to minimize the effect of channel noise. That is, the number of errors between the channel encoder input (derived from the source) and the channel decoder output (delivered to the user) are minimized. For a fixed modulation scheme, the addition of redundancy in

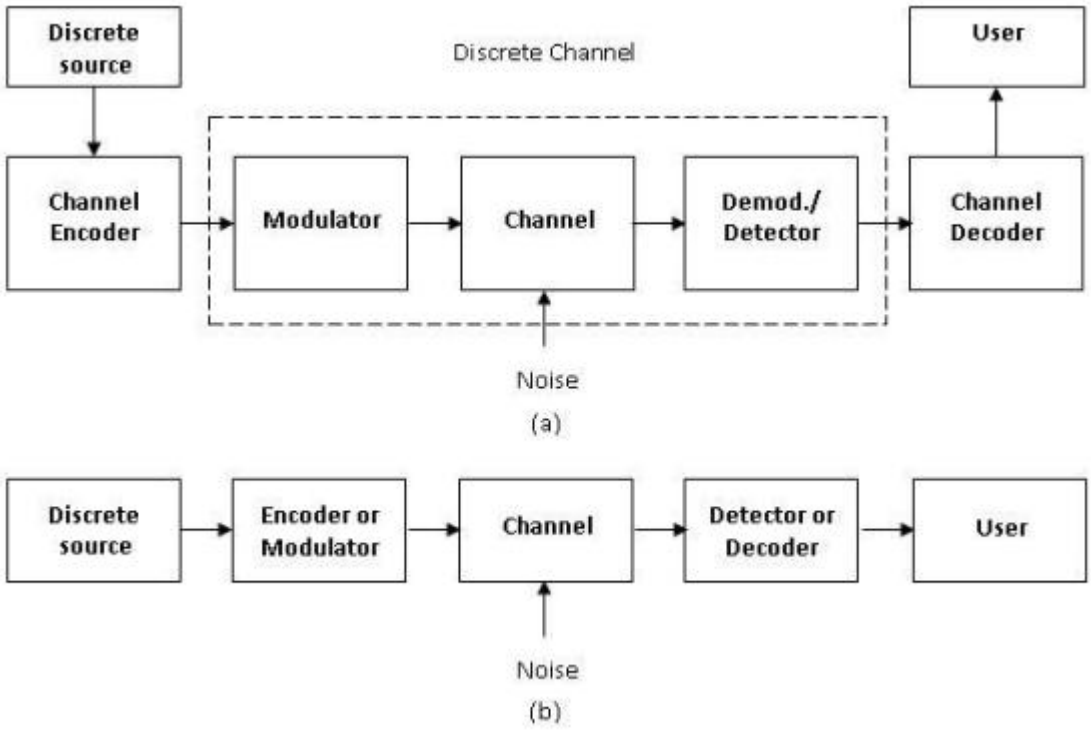

Figure 1.1: Simplified models of digital communication system (a) Coding and modulation performed separately. (b) Coding and modulation combined [1].

the coded messages implies the need for increased transmission bandwidth. Moreover, the use of error-control coding adds complexity to the system, especially for the implementation of decoding operations in the receiver. Thus, the design trade-offs in the use of error-control coding to achieve acceptable error performance includes considerations of bandwidth and system complexity.

1.3 Types of Channel Codes

There are many error-correcting codes, with roots in diverse mathematical disciplines that are used. Historically, these codes have been classified into block codes and convolutional codes. The distinguishing feature for this particular classification is the presence or absence of memory in the encoders for the two coding systems.

Block codes are based on finite field arithmetic and abstract algebra. It can be used to either detect or correct errors. Block codes accept a block of k information bits and produce a block of n coded bits. Commonly, these codes are referred to as (n,k) block codes. To generate an (n,k) block codes, the channel encoder accepts information in successesive k- bit blocks; for each block, it adds n-k redundant bits that are algebraically related to the k message bits, there by producing an overall encoded block of n bits, where n > k. The n-bit block is called a code word, and n is called the block length of the code.

The channel encoder produces bits at the rate Ro = (n/k) Rs, where Rs is the bit rate of the information source. The dimensionless ratio R =k/n is called the code rate, where $0 < r < 1$. The bit rate Ro, coming out of the encoder, is called the channel data rate. Thus, the code rate is a dimensionless ratio, whereas the data rate produced by the source and the channel data rates are both measured in bits per second. Some of the commonly used block codes are Hamming Codes, Golay Codes, BCH Codes, and Reed Solomon Codes [2].

Convolutional codes are one of the most widely used channel code in practical communication systems. These codes are developed with a separate strong mathematical structure and are primarily used for real time error correction. Convolutional codes convert the entire data stream into one single code word. The encoded bits depend not only on the current k input bits but also on past input bits in a convolutional

code, the encoding operation may be viewed as the discrete time convolution of the input sequence with the impulse response of the encoder. The duration of the impulse response equals the memory of the encoder. Accordingly, the encoder for a convolutional code operates on the incoming message sequence, using a "sliding window" equal in duration to its own memory. This, in turn, means that in a convolutional code, unlike a block code, the channel encoder accepts message bits as a continuous sequence and thereby generates a continuous sequence of encoded bits at a higher rate.

The main decoding strategy for convolutional codes is based on the widely used Viterbi algorithm [2]. Since convolutional codes are widely accepted codes, there have been many advances to extend and improve the basic coding and decoding schemes. This advances have resulted in two new coding schemes, namely, trellis coded modulation (TCM) and turbo codes [1].

Turbo code is a near channel capacity error correcting code. This error correcting code is able to transmit information across the channel with arbitrary low (approaching to zero) bit error rate. This code is a parallel concatenation of two component convolutional codes separated by a interleaver. The performance of a turbo code is partly due to the interleaver used to give the turbo code a random appearance. However, one big advantage of a turbo code is that there is enough code structure (from the convolutional codes) to decode it efficiently.

1.4 Problem Statement

The objective of the project is to study the various error correcting codes which involves the study of the widely used codes like the convolution codes and turbo codes. The simulations will be carried out in MATLAB and the outcomes will be compared with the published results.

1.5 Outline of the Thesis

Chapter 2: Presents the fundamentals of convolutional code, the encoder structure and its many representations. Also, it discusses the primary decoding algorithm for convolutional code, namely the Viterbi algorithm.

Chapter 3: Introduces the basic turbo code encoder. The turbo code encoder is a parallel concatenation of two recursive systematic convolutional (RSC) codes, separated by an interleaver. This chapter shows the construction of a RSC encoder from a non-recursive nonsystematic (conventional) convolutional encoder. It discusses the similarities and differences between the conventional and RSC encoders in terms of their intrinsic properties. Furthermore, this chapter describes different types of interleaver and discusses the function of the interleaver that is suitable for the turbo code encoder.

Chapter 4: This chapter first describes the drawbacks of decoding multistage concatenated convolutional codes with the basic Viterbi algorithm. A reliability (soft output) measure is introduced to combat these drawbacks. Also, integration of this reliability value with the basic Viterbi algorithm is presented. The resulting Viterbi algorithm is known as the soft output Viterbi algorithm (SOVA). The SOVA needs to be further modified for turbo code decoding. To understand this modification, the concepts of log-likelihood ratio and soft channel outputs are introduced. From these concepts, the metric for the SOVA is derived. The SOVA component decoder for turbo code is then introduced. A simple and efficient implementation of the SOVA component decoder is presented. This chapter concludes with a description of the SOVA iterative turbo code decoder.

Chapter 5: Discuses the performance of convolutional codes and turbo codes through extensive computer simulation. Many simulation results are then presented to show

the important characteristics of both codes.

Chapter 6: Summarize the important findings for both codes, and concludes the thesis.

Chapter 2

Convolutional Codes

2.1 Introduction

Convolutional codes have been widely used in applications such as space and satellite communication, cellular mobile, digital video broadcasting etc. Their popularity stems from simple structure and availability of easily implementable maximum likelihood soft decision decoding methods. Convolutional codes were first introduced by Elias [3]. The ground work on the mathematical theory of convolutional codes was studied and introduced by Forney [4]. In this chapter the basic structure of convolutional codes needed in the design of turbo codes is presented. The areas of particular importance are encoder structure and decoder structure. The encoder will be represented in many different but equivalent ways. Since Turbo codes are a new class of convolutional codes, a full understanding of convolutional codes is an important prerequisite to the understanding of turbo codes.

2.2 Encoder Structure

A convolutional code is generated by passing the information sequence to be transmitted through a linear finite-state shift register. In general, the shift register consists of K (k-bit) stages and linear algebraic function generators, as shown in Figure 2.1.

The input data to the encoder, which is assumed to be binary, is shifted into and

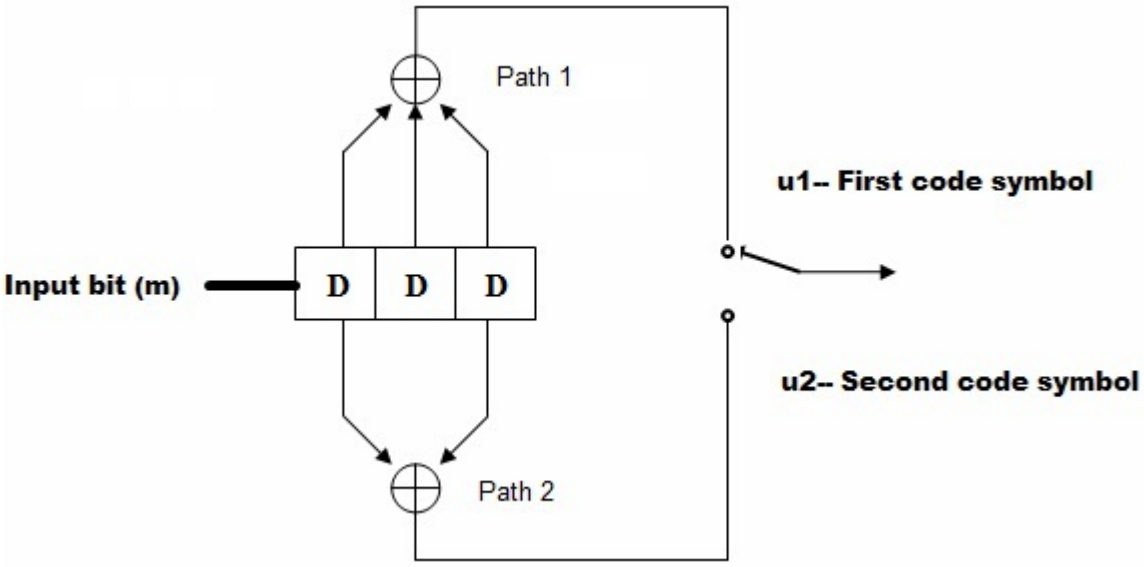

Figure 2.1: Convolutional encoder with k = 1, m = 2, R = 1/2 and K = 3 [2].

along the shift register k bits at a time. The number of output bits for each k-bit input sequence is n bits. Consequently , the code rate is defined as Rc = k/n, consistent with the definition of the code rate for a block code, where k is the number of parallel input information bits and n is the number of parallel output encoded bits in one time interval. The parameter K is called the constraint length of the convolution code and defined as

$$K = m + 1 \tag{2.1}$$

where m is the maximum number of stages (memory size) in any shift register. The shift registers store the state information of the convolutional encoder and the constraint length relates the number of bits upon which the output depends. For the convolutional encoder shown in Figure 2.1, the code rate R = 1/2, the memory size m = 2,and the constraint length K = 3, Where D represents delay.

2.3 Encoder Representations

There are four alternative methods that are often used to represent a convolutional code. From those a tree diagram representation is not widely used to represent encoder, so only three methods are discussed so far.
These are:-

1. Generator Representation
2. Trellis Diagram Representation
3. State Diagram Representation

2.3.1 Generator Representation

Generator representation shows the hardware connection of the shift register taps to the modulo-2 adders. A generator vector represents the position of the taps for an output. A "1" represents a connection and a "0" represents no connection. Each path connecting the output to the input of a convolutional encoder may be characterized in terms of its impulse response, defined as the response of that path to a symbol 1 applied to its input, with each flip-flop in the encoder set initially to the zero state. Equivalently, we may characterize each path in terms of a generator polynomial, defined as the unit-delay transform of the impulse response. To be specific, let the generator sequence $(g_0^{(i)}, g_1^{(i)}, g_2^{(i)}.....g_M^{(i)})$ denote the impulse response of the i^{th} path, where the coefficients $(g_0^{(i)}, g_1^{(i)}, g_2^{(i)}.....g_M^{(i)})$ equal 0 or 1. Correspondingly, the generator polynomial of the i^{th} path is defined by

$$g^{(i)}(D) = g_0^{(i)} + g_1^{(i)}\ D + g_2^{(i)}\ D^2 + ... + g_M^{(i)}\ D^M \tag{2.2}$$

Where D denotes the unit-delay variable, Traditionally, different variables are used for the description of convolutional and cyclic codes, with D being commonly used for convolutional codes and X for cyclic codes.

Here in the Figure 2.1 of the convolutional encoder has two paths numbered 1 and 2 for convenience of reference. The impulse response of path 1 (i.e., upper path) is (1, 1, 1). Hence, the corresponding generator polynomial is given by

$$g^{(1)}(D) = 1 + D + D^2 \tag{2.3}$$

The impulse response of path 2 (i.e., lower path) is (1, 0, 1). Hence the corresponding generator polynomial is given by

$$g^{(2)}(D) = 1 + D^2 \tag{2.4}$$

For the message sequence (10011), the polynomial representation is

$$m(D) = 1 + D^3 + D^4 \tag{2.5}$$

As with Fourier transforms, convolution in the time domain is transformed into multiplication in the D-domain. Hence, the output polynomial of path 1 is given by

$$\begin{aligned} C^{(1)}(D) &= g^{(1)}(D)m(D) \\ &= (1 + D + D^2)(1 + D^3 + D^4) \\ &= 1 + D + D^2 + D^3 + D^6 \end{aligned} \tag{2.6}$$

From this we immediately deduce that the output of path 1 is (1111001). Similarly, the output polynomial of path 2 is given by

$$\begin{aligned} C^{(2)}(D) &= g^{(2)}(D)m(D) \\ &= (1 + D^2)(1 + D^3 + D^4) \\ &= 1 + D^2 + D^3 + D^4 + D^5 + D^6 \end{aligned} \tag{2.7}$$

The output sequence of path 2 is therefore (1011111). Finally, multiplexing the two output sequences of paths 1 and 2, and the encoded sequence is

$$c = (11, 10, 11, 11, 01, 01, 11) \tag{2.8}$$

Note that the message sequence of length L = 5 bits produces an encoded sequence of length $n(L + K - 1) = 14$ bits. Note also that for the shift register to be restored to its zero initial state, a terminating sequence of K 1 = 2 zeros, called the tail of the message, is registered [1].

2.3.2 Trellis Diagram Representation

A trellis is a tree like structure with remerging of branches. It is more instructive than a tree in that it brings out explicitly the fact that the associated convolutional encoder is a finite-state machine. The convention used in figure 2.2 to distinguish between input symbols 0 and 1 is as follows: a code branch produced by an input 0 is drawn as a solid line, whereas a code branch produced by an input 1 is drawn as dashed line. Each input (message) sequence corresponds to a specific path through the trellis. For example, we readily see from Figure 2.2 that the message sequence (10011) produces the encoded output sequence $(11, 10, 11, 11, 01)$.

The state of a convolutional encoder of rate $1/n$ where $(K - 1)$ message bits are stored in the encoder's shift register is defined.. At time instant j, the portion of message sequence containing the most recent K bits is written as $(\mathbf{m_{j-K+1}}, ..\mathbf{m_{j-1}}, \mathbf{m_j})$ where, m_j is the current bit. The $(K - 1)$-bit state of the encoder at time j is therefore written simply as $(\mathbf{m_{j-1}}..\mathbf{m_{j-K+2}}, \mathbf{m_{j-K+1}})$. In the case of simple convolutional encoder of Figure 2.1 has $(K - 1) = 2$. Hence, the state of this encoder can assume any one of four possible values, as described in Table 2.1. The trellis contains $(L+K)$ levels, where L is the length of the incoming message sequence, and K is the constraint length of the code. The levels of the trellis are labeled as $j = 0, 1, , L+K-1$ in Figure

State	*Binary Description*
a	00
b	01
c	10
d	11

Table 2.1: State table for convolutional encoder of Figure 2.1

2.2 for $K = 3$. Level j is also referred to as depth j; both terms are used interchangeably. The first (K-1) levels correspond to the encoder's return to the state a. Clearly, not all the states can be reached in these two portions of the trellis. However, in the central portion of the trellis, for which the level j lies in the range $K - 1 < j < L$, all the states of the encoder are reachable. Note also that the central portion of the trellis exhibits a fixed periodic structure [2].

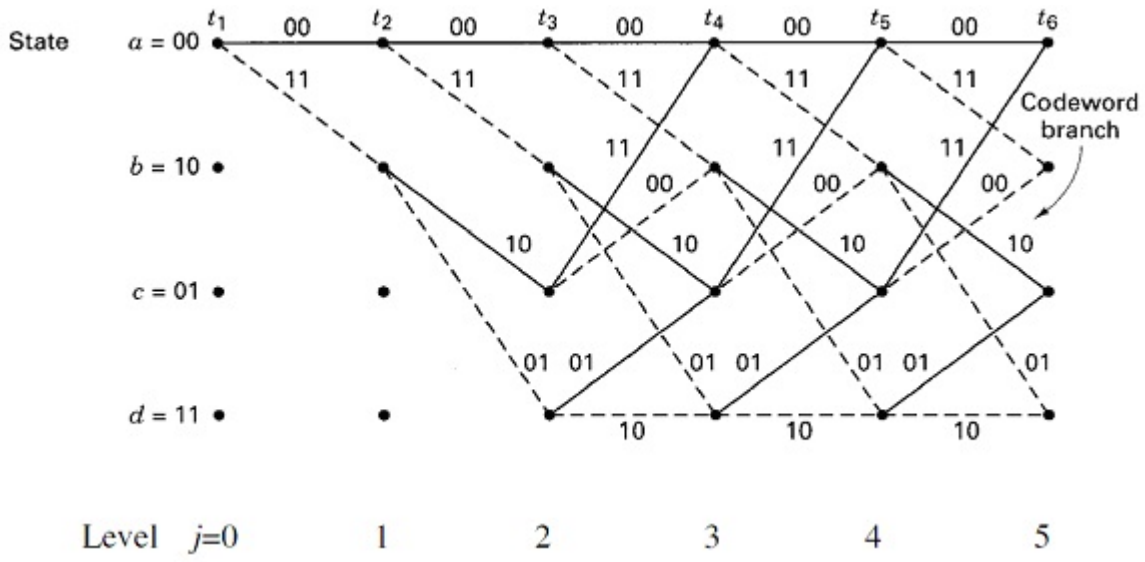

Figure 2.2: Trellis for the convolutional encoder of Figure 2.1 [2].

2.3.3 State Diagram Representation

Consider next a portion of the trellis corresponding to times j and j + 1. It's assume that $j \geq 2$ for the example at hand, so that it is possible for current state of the encoder to be a, b, c, or d. For convenience of presentation, the portion of the trellis in figure 2.3(a) is reproduced. The left nodes represent the four possible current states of the encoder, whereas the right nodes represent the next states. The left and right nodes are combined which gives the state diagram of the encoder, shown in figure 2.3(b). The state diagram shows the state information of a convolutional encoder.

The state information of a convolutional encoder is stored in the shift registers. The nodes of the figure represent the four possible states of the encoder, with each node having two incoming branches and two outgoing branches. A transition from one state to another in response to input 0 is represented by a solid branch, whereas a transition in response to input 1 is represented by a dashed branch. The binary label on each branch represents the encoder's output as it moves from one state to another. Suppose, for example, the current state of the encoder is (01), which is represented by node c. The application of input 1 to the encoder of Figure 2.1 results in the state (10) and the encoded output (00), accordingly, with the help of this state diagram, one can readily determine the output of the encoder of Figure 2.1 for any incoming message sequence. For simplicity start at state a, the all-zero initial state, and walk through the state diagram in accordance with the message sequence. One has to follow a solid branch if the input is a 0 and a dashed branch if it is a 1.

Here, for example, the message sequence (10011). For this input the path abcabd is followed, and therefore output of the sequence (11,10,11,11,01), which agrees exactly with our previous result. Thus, the input-output relation of a convolutional encoder is also completely described by its state diagram[2].

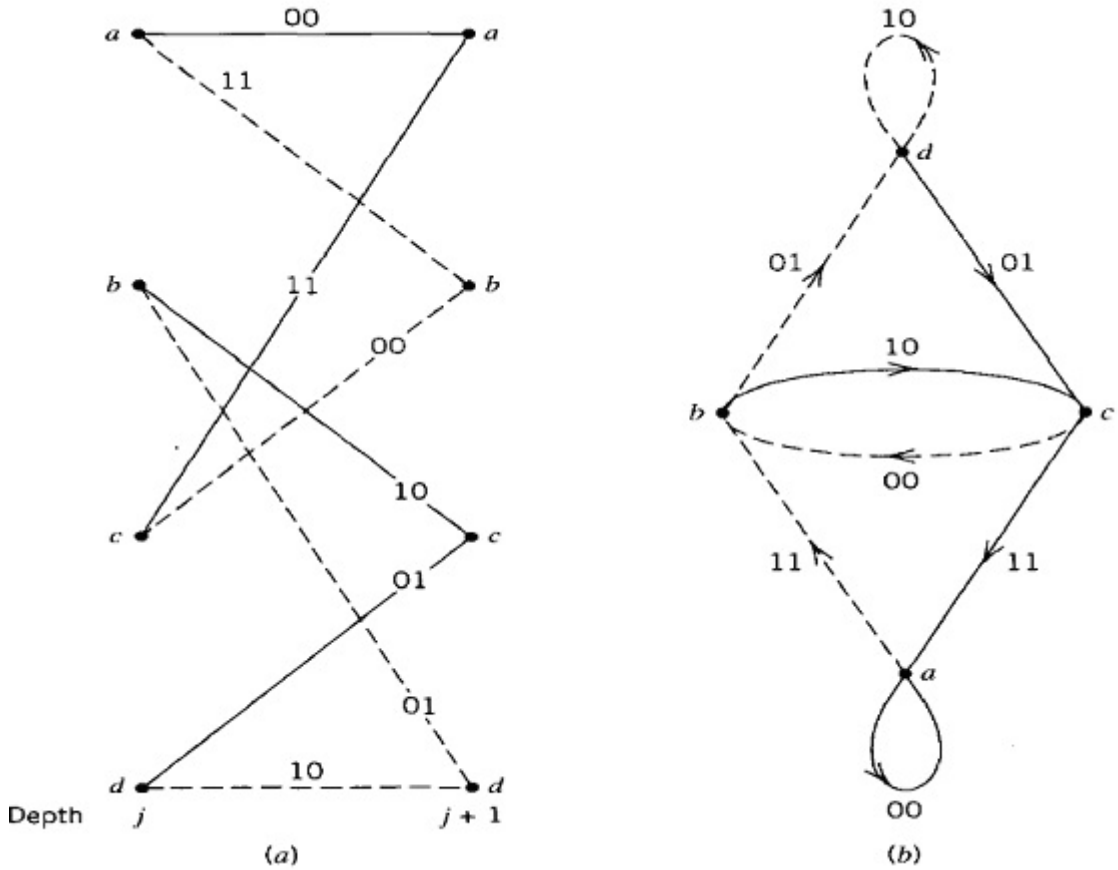

Figure 2.3: **(a)** A portion of the central part of the trellis for the encoder of Figure 2.1 **(b)** State diagram of the convolutional encoder of Figure 2.1 [2].

2.4 The Viterbi Algorithm

The equivalence between maximum likelihood decoding and minimum distance decoding for a binary symmetric channel implies that may be decode a convolutional code by choosing a path in the code tree whose coded sequence differs from the received sequence in the fewest number of places. Since a code tree is equivalent to a trellis, the choice to find the possible paths in the trellis representation of the code are bounded. The reason for preferring the trellis over the tree is that the number of nodes at any level of the trellis does not continue to grow as the number of incoming message bits increases; rather, it remains constant at 2^{K-1}, where K is the constraint

length of the code.

Here, for example, the trellis diagram of Figure 2.2 for a convolutional code with rate R = 1/2 and constraint length K = 3. It's observe that at level j = 3, there are two paths entering any of the four nodes in the trellis. Moreover, these two paths will be identical from that point onward. Clearly, a minimum distance decoder may make a decision at that point as to which of those two paths to retain, without any loss of performance. A similar decision may be made at level j = 4, and so on. This sequence of decisions is exactly what the Viterbi algorithm does as it walks through the trellis. The algorithm operates by computing a metric or discrepancy for every possible path in the trellis. The metric for a particular path is defined as the Hamming distance between the coded sequence represented by that path and the received sequence. Thus, for each node (state) in the trellis of Figure 2.2 the algorithm compares the two paths entering the node. The path with the lower metric is retained, and the other path is discarded. This computation is repeated for every level j of the trellis in the range $M \leq j \leq L$, where $M = K - 1$ is the encoders memory and L is the length of the incoming message sequence. The paths that are retained by the algorithm are called survivor or active paths. For a convolutional code of constraint length K= 3, for example, no more than $2^{K-1} = 4$ survivor paths and their metrics will ever be stored. This list of 2^{K-1} paths is always guaranteed to contain the maximum-likelihood choice.

A difficulty that may arise in the application of the Viterbi algorithm is the possibility that when the paths entering a state are compared, their metrics are found to be identical. In such a situation, select one randomly (i.e., simply make a guess), which may be correct or incorrect [1]. Another method for decoding of convolutional codes is using syndrome decoding [5].

2.5 Distance Properties of Convolutional Code

The error probability performance of convolutional codes depends on their distance properties. There are two types of distances. For hard decision decoding, the decoder operates with binary symbols and the code performance is measured by Hamming distances. A soft decision decoder receives quantized or analog signals from the demodulator and the decoding operation is based on Euclidian distance. The minimum free distance, denoted by d_{free}, of a convolutional code is defined as the minimum Hamming distance between any two code sequences in the code. Since convolutional codes are linear, the Hamming distance between two code sequences is equal to the weight of modulo-2 sum, which is another code sequence. Therefore, the minimum free distance is the minimum weight of all non-zero code sequences. In other words, the all-zero sequence can be used as the reference sequence in the determining the minimum free distance.

The minimum free distance, d_{free}, is defined as

$$d_{free} = min[d\ (\mathbf{y_1}, \mathbf{y_2})|\mathbf{y_1} \neq \mathbf{y_2}] \tag{2.9}$$

$$= min[w(\mathbf{y})|\mathbf{y} \neq \mathbf{0}] \tag{2.10}$$

where $d(\mathbf{y_1}, \mathbf{y_2})$ is the Hamming distance between a pair of convolutional codewords and $w(\mathbf{y})$is the Hamming distance between a convolutional codeword and the all-zero codeword (the weight of the codeword). The minimum free distance corresponds to the ability of the convolutional code to estimate the best decoded bit sequence. As d_{free} increases, the performance of the convolutional code also increases. This characteristic is similar to the minimum distance for block codes [6].

Chapter 3

Turbo Codes

3.1 Introduction

It is well known that a good trade-off between coding gain and complexity can be achieved by serial concatenated codes proposed by Forney [4]. A serial concatenated code is one that applies two levels of coding, an inner and an outer code linked by an interleaver. This approach has been used in space communications, with convolutional codes as the inner code and low redundancy Reed Solomon codes as the outer code. The primary reason for using a concatenated code is to achieve a low error rate with an overall decoding complexity lower than that required for a single code of the corresponding performance. The low complexity is attained by decoding each component code separately. As the inner decoder generates burst errors an interleaver is typically incorporated between the two codes to decorrelate the received symbols affected by burst errors. In decoding these concatenated codes, the inner decoder may use a $soft - input/soft - output$ decoding algorithm to produce soft decisions for the outer decoder. Turbo codes exploit a similar idea of connecting two codes and separating them by a long interleaver [7]. The difference between turbo and serial concatenated codes is that in turbo codes two identical systematic component codes are connected in parallel. The primary reason for using a long interleaver in turbo

coding is to generate a concatenated code with a large block length which leads to a large coding gain. The performance of turbo codes is also improve by using multifold coding technique [8].

3.2 Turbo Code Encoder

A turbo encoder is formed by parallel concatenation of two recursive systematic convolutional (RSC) encoders separated by a random Interleaver [7]. The encoder structure is called parallel concatenation because the two encoders operate on the same set of input bits, rather than one encoding the output of the other. Thus turbo codes are also referred to as parallel concatenated convolutional codes (PCCC). A block diagram of a rate 1/3 turbo encoder is shown in Figure 3.1. The generator matrix of a component RSC code can be represented as:

$$\mathbf{G(D)} = [\mathbf{1}, \frac{\mathbf{g_2(D)}}{\mathbf{g_1(D)}}] \tag{3.1}$$

Where $g_1(D)$ and $g_2(D)$ are a feedback and feed forward polynomials respectively

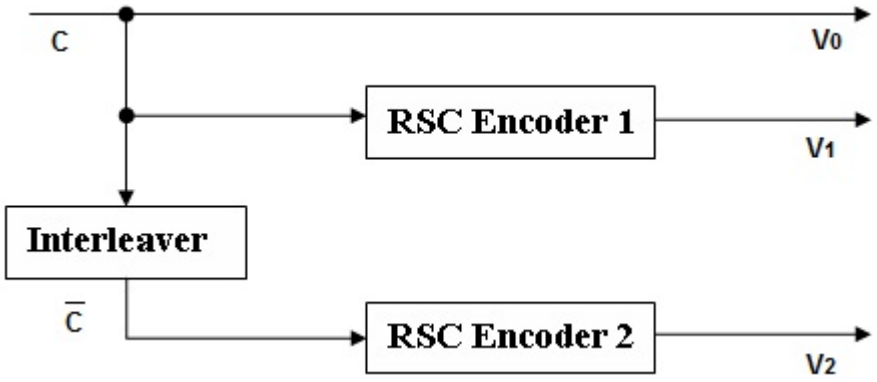

Figure 3.1: Fundamental turbo code encoder

with degree v. In the encoder, the same information sequence is encoded twice but in a different order. The first encoder operates directly on the input sequence, denoted

by $\mathbf{C}$, of length $\mathbf{N}$. The first component encoder has two outputs. The first output, denoted by $\mathbf{V_0}$, is equal to the input sequence since the encoder is systematic. The other output is the parity check sequence, denoted by $\mathbf{V_1}$. The interleaved information sequence at the input of the second encoder is denoted by $\bar{\mathbf{C}}$. Only the parity check sequence of the second encoder, denoted by $\mathbf{V_2}$, is transmitted. The information sequence $\mathbf{V_0}$ and the parity check sequences of the two encoders, $\mathbf{V_1}$ and $\mathbf{V_2}$ are multiplexed to generate the turbo code sequence. The overall code rate is 1/3.

3.2.1 Recursive Systematic Convolutional (RSC) Encoder

The recursive systematic convolutional (RSC) encoder is obtained from the non recursive nonsystematic (conventional) convolutional encoder by feeding back one of its encoded outputs to its input. The conventional encoder is represented by the generator sequences g_1=[111] and g_2=[101] and can be equivalently represented in a more compact form as G=[g_1,g_2]. The RSC encoder of this conventional encoder is represented as G=[1,g_2/g_1] where the first output (represented by g_1) is fed back to the input. In the above representation, 1 denotes the systematic output, g_2 denotes the feed forward output, and g_1 is the feedback to the input of the RSC encoder. Figure 3.2 shows the resulting RSC encoder [2].

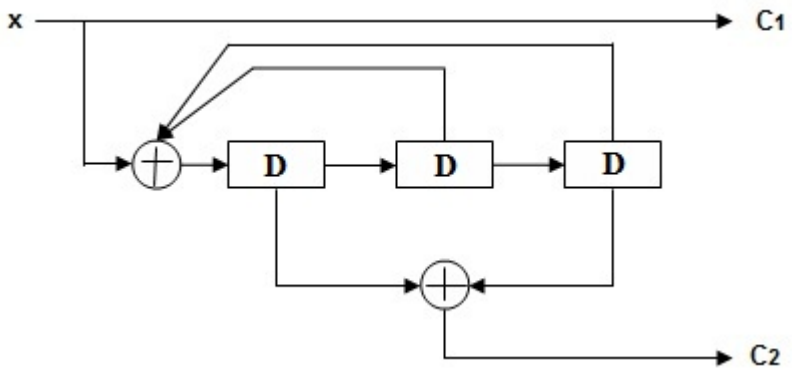

Figure 3.2: The RSC encoder obtained from Figure 2.1 with R = 1/2 and K =3 [2].

3.2.2 Concatenation of Codes

The power of FEC codes increase with length k and approaches the Shannon bound only at very large k. However the decoding complexity increases very rapidly with k. This suggests that it would be desirable to build a long, complex code out of much shorter component codes, which can be decoded much more easily. Concatenation provides a very straightforward means of achieving this. A concatenated code is composed of two separate codes that are combined to form a larger code. There are two types of concatenation, namely serial and parallel concatenations. Figure 3.3 shows the serial concatenation scheme for transmission [1]. The total code rate for

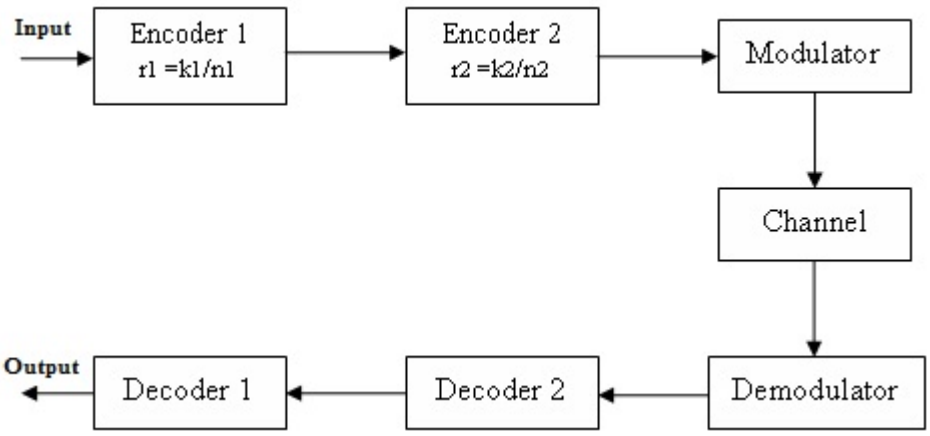

Figure 3.3: Serial concatenated code.

serial concatenation is

$$R_{total} = \frac{k_1 k_2}{n_1 n_2} \tag{3.2}$$

which is equal to the product of the two code rates.

Figure 3.4 shows the parallel concatenation scheme for transmission.

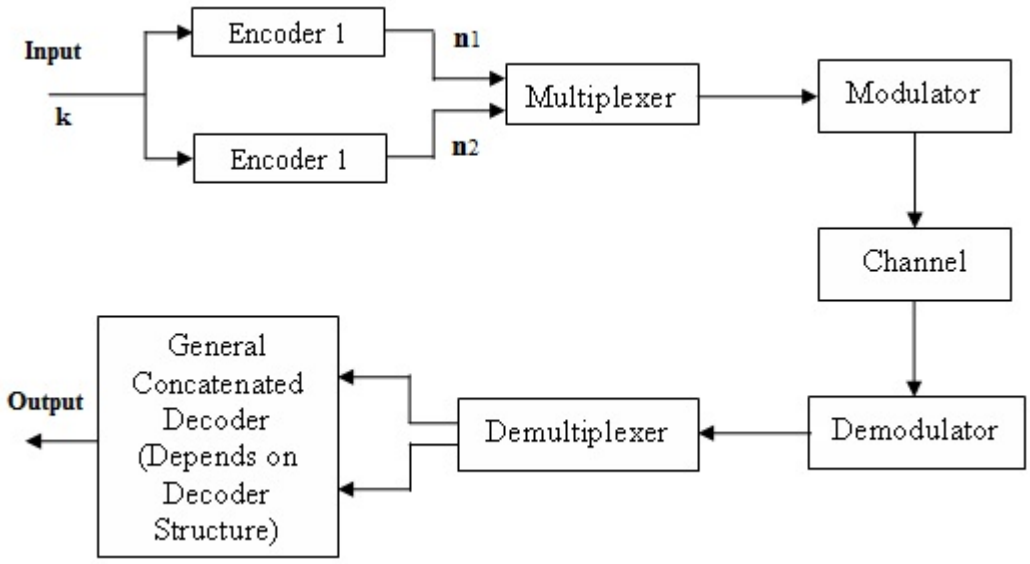

Figure 3.4: Parallel concatenated code.

The total code rate for parallel concatenation is

$$R_{total} = \frac{k}{n_1 + n_2} \tag{3.3}$$

The serial concatenated scheme suffers from a number of drawbacks, the most significant of which is error propagation. If a decoding error occurs in a code word, it usually results in a number of data errors. When these are passed on to the next stage, it may be overwhelming the ability of that code to correct the errors. The performance of the outer decoder might be improved if these errors were distributed among a number of separate code words. This can be achieved by using an interleaved.

3.2.3 Interleaver Design

An interleaver is an input-output mapping device that permutes the ordering of a sequence of symbols from a fixed alphabet in a completely deterministic manner; that

is, it takes the symbols at the input and produces identical symbols at the output but in a different temporal order. For turbo codes, an interleaver is used between the two component encoders. The interleaver is used to provide randomness to the input sequence, generate a long block code from small memory convolutional codes; it decorrelates the inputs to the two decoders. If the input sequences to the two component decoders are decorrelated there is a high probability that after correction of some of the errors in one decoder some of the remaining errors should become correctable in the second decoder. Also, it is used to increase the weights of the codewords as shown in Figure 3.5 [9]. From Figure 3.5, the input sequence **u** pro-

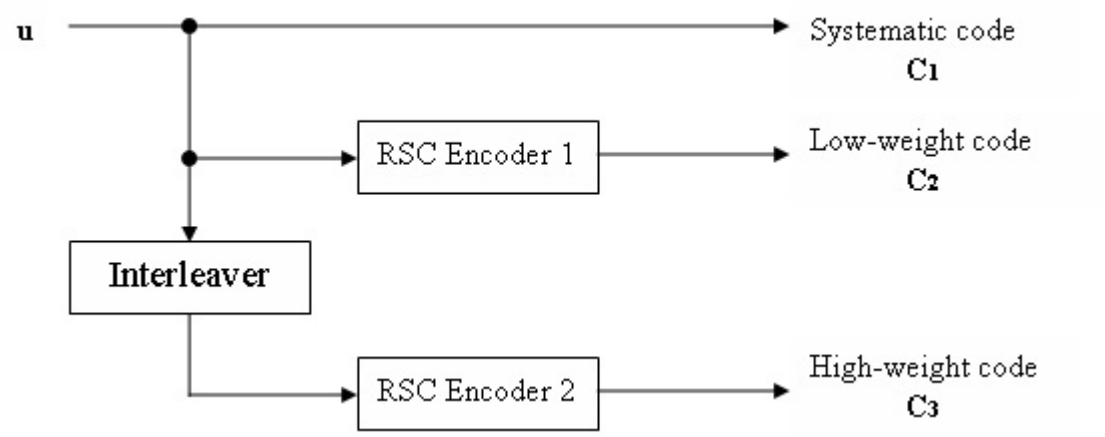

Figure 3.5: The interleaver increases the code weight for RSC encoder 2 as compared to RSC encoder 1.

duces a low-weight recursive convolutional code sequence $\mathbf{C_2}$ for RSC encoder 1. To avoid having RSC encoder 2 produce another low-weight recursive output sequence, the interleaver permutes the input sequence **u** to obtain a different sequence that hopefully produces a high-weight recursive convolutional code sequence $\mathbf{C_3}$. Thus, the turbo code's code weight is moderate, combined from encoder 1's low-weight code and encoder 2's high-weight code. Figure 3.6 shows an illustrative example. From Figure 3.6, the input sequence $\mathbf{u_i}$ produces output sequences $\mathbf{C_{1i}}$ and $\mathbf{C_{2i}}$ respectively. The input sequences $\mathbf{u_1}$ and $\mathbf{u_2}$ are different permuted sequences of $\mathbf{u_0}$. Table 3.1

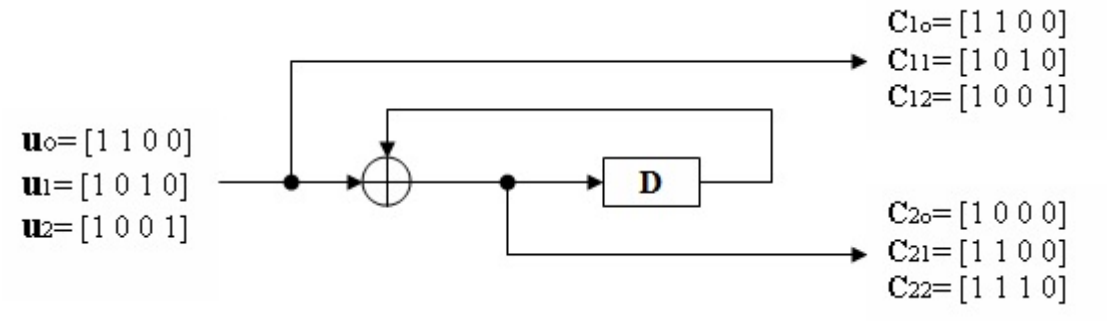

Figure 3.6: An illustrative example of an interleavers capability.

	Input Sequence $\mathbf{u_i}$	Output Sequence $\mathbf{C_{1i}}$	Output Sequence $\mathbf{C_{2i}}$	Codeword Weight **i**
i=0	1 1 0 0	1 1 0 0	1 0 0 0	3
i=1	1 0 1 0	1 0 1 0	1 1 0 0	4
i=2	1 0 0 1	1 0 0 1	1 1 1 0	5

Table 3.1: Input and output sequences for encoder of Figure 3.8

shows the resulting codewords and weights. As it can be seen from Table 3.1, the codeword weight can be increased by utilizing an interleaver. The interleaver affects the performance of turbo codes because it directly affects the distance properties of the code. By avoiding low-weight codewords, the BER of a turbo code can improve significantly. Thus, much research has been done on interleaver design. The following subsections shows different types of interleaver commonly used in turbo code design.

Block Interleaver

The block interleaver is the most commonly used interleaver in communication systems. It writes in column wise from top to bottom and left to right and reads out row wise from left to right and top to bottom. Figure 3.7 shows a block interleaver. From Figure 3.7, the interleaver writes in [**00**...**101**...**0**...**1**...**101**...**01**] and reads out [**01**...**100**...**1**...**1**...**000**...**11**] [2].

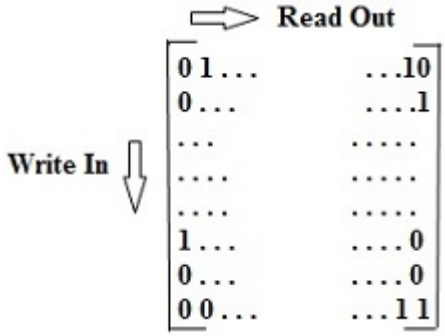

Figure 3.7: Block interleaver [2].

Random Interleaver

The random interleaver uses a fixed random permutation and maps the input sequence according to the permutation order. The length of the input sequence is assumed to be L. Figure 3.8 shows a random interleaver with L=8. From Figure 3.8, the interleaver writes in [**01101001**] and reads out [**01011001**] [6].

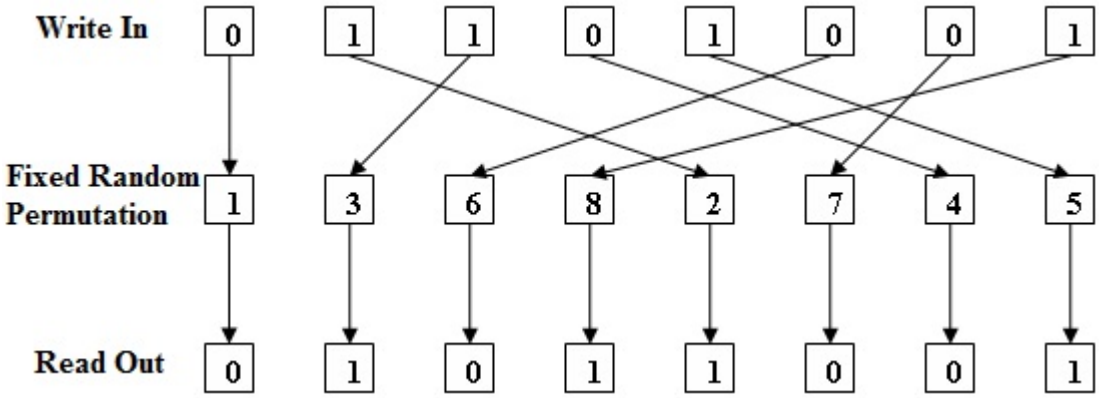

Figure 3.8: Random interleaver with L=8 [6].

Chapter 4

System Model for Turbo Code Decoder

The turbo code decoder is based on modified Viterbi algorithm that incorporates reliability values to improve decoding performance. The Viterbi algorithm (VA) is modified to deliver not only the most likely path sequence in a finite-state Markov chain, but either the a-posteriori probability for each bit or reliability value. With this reliability indicator the modified VA produces soft decisions to be used in decoding of outer codes.

In order to design and implement the decoding algorithm, first the concept of reliability for Viterbi decoding and the metric that will be used in the modified Viterbi algorithm for turbo code decoding is described.

4.1 Principle of the Soft-Output Viterbi Decoder

The Viterbi algorithm produces the ML output sequence for convolutional codes. This algorithm provides optimal sequence estimation for one stage convolutional codes. For concatenated (multistage) convolutional codes, there are two main drawbacks to conventional Viterbi decoders. First, the inner Viterbi decoder produces bursts of bit

errors which degrade the performance of the outer Viterbi decoders [10]. Second, the inner Viterbi decoder produces hard decision outputs which prohibit the outer Viterbi decoders from deriving the benefits of soft decisions [10]. Both of these drawbacks can be reduced and the performance of the overall concatenated decoder can be significantly improved if the Viterbi decoders are able to produce reliability (soft-output) values [11].

The reliability values are passed on to subsequent Viterbi decoders as a priori information to improve decoding performance. This modified Viterbi decoder is referred to as the Soft-Output Viterbi Algorithm (SOVA) decoder. Figure 4.1 shows a concatenated SOVA decoder.

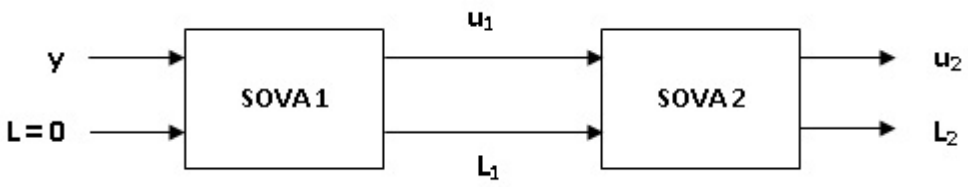

Figure 4.1: A concatenated SOVA decoder where y represents the received channel values, u represents the hard decision output values, and L represents the associated reliability values.

4.2 Reliability of the General SOVA Decoder

The reliability of the SOVA decoder is calculated from the trellis diagram as shown Figure 4.2. In Figure 4.2, a 4-state trellis diagram is shown. The solid line indicates the survivor path (assumed here to be part of the final ML path) and the dashed line indicates the competing (concurrent) path at time t for state 1. For the sake of brevity and clarity, survivor and competing paths for other nodes are not shown. The label $S_{1,t}$ represents state 1 and time t. Also, the labels 0,1 shown on each path

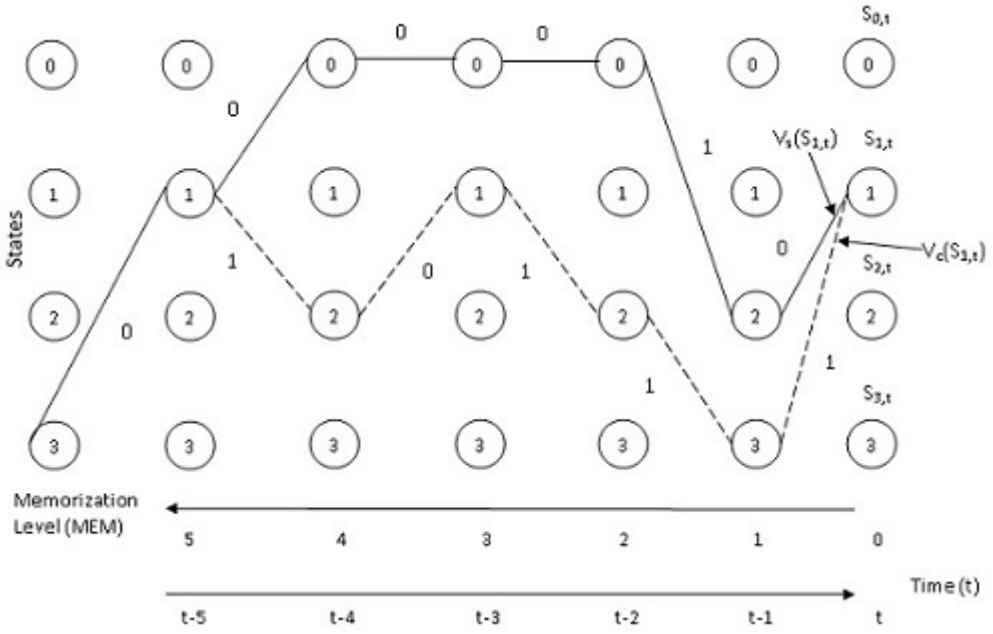

Figure 4.2: Example of survivor and competing paths for reliability estimation at time t [11].

indicate the estimated binary decision for the paths. The survivor path for this node is assigned an accumulated metric $V_s(S_{1,t})$ and the competing path for this node is assigned an accumulated metric $V_c(S_{1,t})$.

To assigning a reliability value L(t) to node $S_{1,t}$'s survivor path is the absolute difference between the two accumulated metrics, L(t) = | $V_s(S_{1,t})$ - $V_c(S_{1,t})$ | [11]. The greater this difference, the more reliable is the survivor path. Because the two paths give two opposite binary decisions about d(t), $V_s(S_{1,t})$ - $V_c(S_{1,t})$ can be directly assigned as a first estimation of the weight of the decision given by the survivor about d(t), conditional to the choice of node m. Where d(t) is the output of the encoder memory. For this reliability calculation, it is assumed that the survivor accumulated metric is always "better" than the competing accumulated metric. Furthermore, to reduce complexity, the reliability values only need to be calculated for the ML sur-

vivor path (assume it is known for now) and are unnecessary for the other survivor paths since it will be discarded later.

To illustrate the concept of reliability, two examples are given below. In these examples, the Viterbi algorithm selects the survivor path as the path with the smaller accumulated metric. In the first example, assume that at node $S_{1,t}$ the accumulated survivor metric $V_s(S_{1,t}) = 50$ and that the accumulated competing metric $V_c(S_{1,t}) = 100$. The reliability value associated with the selection of this survivor path is L(t) = | 50 - 100 |=50. In the second example, assume that the accumulated survivor metric does not change, $V_s(S_{1,t})$=50, and that the accumulated competing metric $V_c(S_{1,t})$=75. The resulting reliability value is L(t) = | 50 - 75 | = 25. Although in both of these examples the survivor path has the same accumulated metric, the reliability value associated with the survivor path is different. The reliability value in the first example provides more confidence (twice as much confidence) in the selection of the survivor path than the value in the second example. Figure 4.3 illustrates a problem with the use of the absolute difference between accumulated survivor and competing metrics as a measure of the reliability of the decision. In Figure 4.3, the survivor and competing paths at $S_{1,t}$ have diverged at time t-5. The survivor and competing paths produce opposite estimated binary decisions at times t, t-2, and t-4 as shown in bold labels. For the purpose of illustration, let us suppose that the survivor and competing accumulated metrics at $S_{1,t}$ are equal, $V_s(S_{1,t}) = V_c(S_{1,t}) = 100$. This means that both the survivor and competing paths have the same probability of being the ML path.

Furthermore, let us assume that the survivor accumulated metric is "better" than the competing accumulated metric at time t-2 and t-4 as shown in Figure 4.3. To reduce the Figure complexity, these competing paths for times t-2 and t-4 are not shown. From this argument, it can be seen that the reliability value assigned to the survivor path at time t is L(t)=0, which means that there is no reliability associated with the

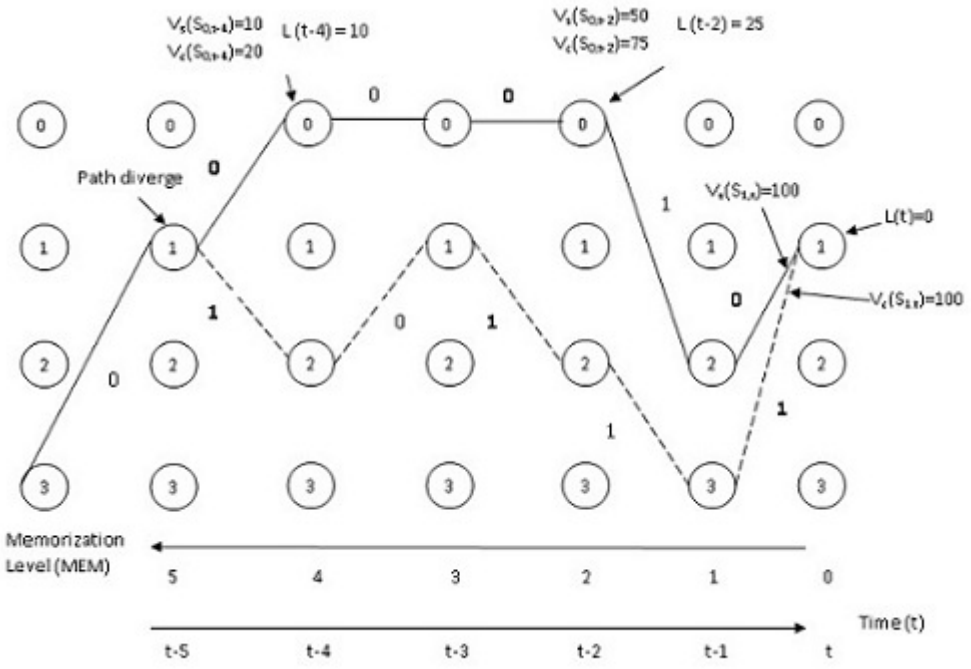

Figure 4.3: Example that shows the weakness of reliability assignment using metric values directly [11].

selection of the survivor path. At times t-2 and t-4, the reliability values assigned to the survivor path were greater than zero (L(t-2)=25 and L(t-4)=10) as a result of the "better" accumulated metrics from the survivor path. However, at time t, the competing path could also have been the survivor path because both have the same metric. Thus, there could have been opposite estimated binary decisions at times t, t-2, and t-4 without reducing the associated reliability values along the survivor path.

To improve the reliability values of the survivor path, a trace back operation to update the reliability values has been suggested [10]. This updating procedure is integrated into the Viterbi algorithm as follows [10]: For node $S_{k,t}$ in the trellis diagram (corresponding to state k at time t):

1. Store L(t) = | $V_s(S_{k,t})$ - $V_c(S_{k,t})$ |. If there is more than one competing path,

then multiple reliability values must be calculated and the smallest reliability value is then set to L(t).

2. Initialize the reliability value of $S_{k,t}$ to $+\infty$ (most reliable).

3. Compare the survivor and competing paths at $S_{k,t}$ and store the memorization levels (MEMs) where the estimated binary decisions of the two paths differ.

4. Update the reliability values at these MEMs with the following procedure:

   - Find the lowest $MEM > 0$, denoted as MEM_{low}, whose reliability value has not been updated.
   - Update MEMlows reliability value L(t-MEM_{low}) by assigning the lowest reliability value between MEM = 0 and MEM = MEM_{low}.

Continuing from the example, the opposite bit estimations between the survivor and competing bit paths for $S_{1,t}$ are located and stored as $MEM = 0, 2, 4$. With this MEM information, the reliability updating process is accomplished as shown in Figure 4.4 and Figure 4.5. In Figure 4.4, the first reliability update is shown. The lowest $MEM > 0$, whose reliability value has not been updated, is determined to be MEM_{low}=2. The lowest reliability value between MEM=0 and MEM=MEM_{low}=2 is found to be L(t) = 0. Thus, the associated reliability value is updated from L(t-2) = 25 to L(t-2) = L(t)=0. The next lowest $MEM > 0$, whose reliability value has not been updated, is determined to be MEM_{low}=4. The lowest reliability value between MEM=0 and MEM= MEM_{low} =4 is found to be L(t)=L(t-2)=0. Thus, the associated reliability value is updated from L(t-4)=10 to L(t-4)=L(t)=L(t-2)=0. Figure 4.5 shows the second reliability update. It has been suggested that the final reliability values should be normalized or logarithmically compressed before passing to the next concatenated decoder to offset possible defects of this updating operation [11].

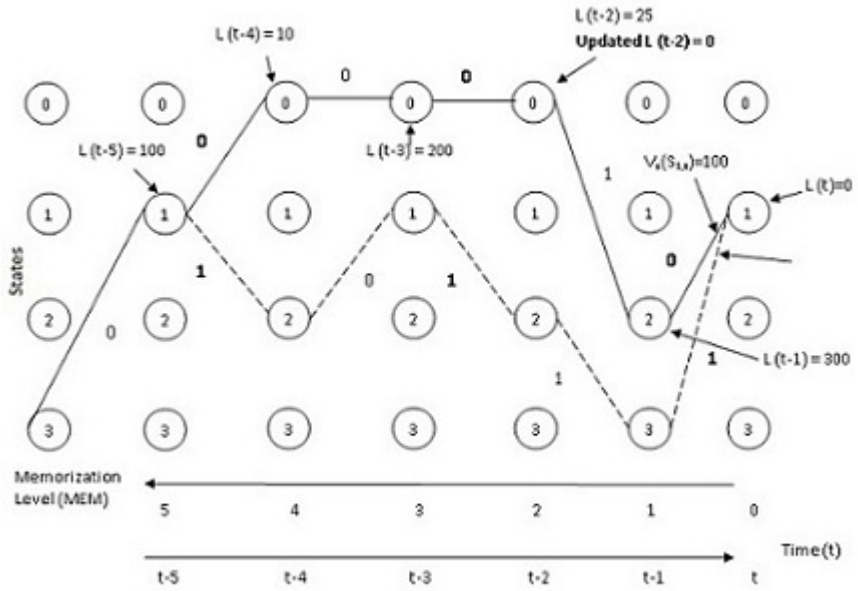

Figure 4.4: Updating process for time t-2 ($MEM_{low} = 2$) [11].

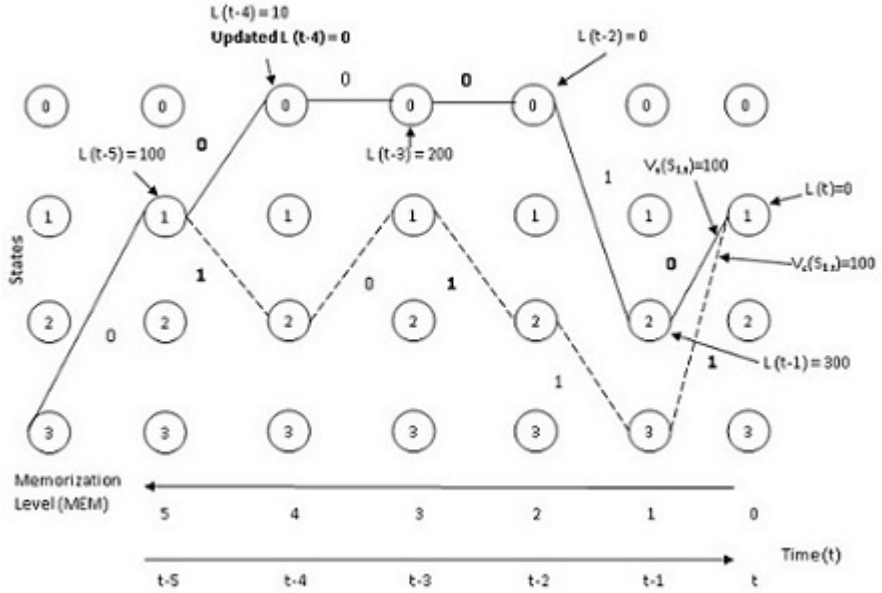

Figure 4.5: Updating process for time t-4 ($MEM_{low} = 4$) [11].

4.3 Introduction to SOVA for Turbo Codes

The SOVA for turbo codes is implemented with a modified Viterbi metric. A close examination of log-likelihood algebra and soft channel outputs is required before attempting to derive this modified Viterbi metric. Figure 4.6 shows the system model that is used to describe the above concepts.

Figure 4.6: System model for SOVA derivation.

4.3.1 Log-Likelihood Algebra

The log-likelihood algebra used for SOVA decoding of turbo codes is based on a binary random variable u in GF(2) with elements +1, -1, where +1 is the logic 0 element (null element) and -1 is the logic 1 element under $\oplus$ (modulo 2) addition. Table 4.1 shows the outcome of adding two binary random variables under these governing factors. The log-likelihood ratio L(u) for a binary random variable u is defined to be

$\mathbf{u_1 \oplus u_2}$	$\mathbf{u_2 = +1}$	$\mathbf{u_2 = -1}$
$u_1 = +1$	+1	-1
$u_1 = -1$	-1	+1

Table 4.1: Outcome of adding two binary random variables u_1 and u_2

$$L(u) = \ln \frac{p(u = +1)}{p(u = -1)} \tag{4.1}$$

L(u) is often denoted as the soft value or L-value of the binary random variable u. The sign of L(u) is the hard decision of u and the magnitude of L(u) is the reliability of this

decision. Table 4.2 shows the characteristics of the log-likelihood ratio L(u). Clearly

P(u = +1)	**P(u = -1)**	**L(u)**
1	0	$+\infty$
0.9	0.1	2.1972
0.5	0.5	0
0.1	0.9	-2.1972
0	1	$-\infty$

Table 4.2: Characteristics of the Log-likelihood Ratio L(u)

from Table 4.2, as L(u) increase toward $+\infty$, the probability of u=+1 also increases. Furthermore, as L(u) decreases toward $-\infty$, the probability of u=-1 increases. As it can be seen, L(u) provides a form of reliability for u. This will be exploited for SOVA decoding as will be described later. The probability of the random variable u may be conditioned on another random variable z. This forms the conditioned log-likelihood ratio L(u|z) and is defined to be

$$L\left(u|z\right) = \ln \frac{p(u = \ +1|z)}{p(u = \ -1|z)} \tag{4.2}$$

The probability of the sum of two binary random variables, say p $(u_1 \ \oplus \ u_2 = \ +1)$, is found from

$$p(u_1 \oplus u_2 = +1) = p(u_1 = \ +1)\ p(u_2 = \ +1) + \ p(u_1 = \ -1)\ p(u_2 = \ -1) \tag{4.3}$$

With the following relation,

$$p\ \left(u = \ -1\right) = 1 - \ p\ \left(u = \ +1\right) \tag{4.4}$$

the probability, p $(u_1 \ \oplus \ u_2 = \ +1)$,becomes

$$p(u_1 \oplus u_2 = +1) = p(u_1 = +1)p(u_2 = +1) + (1 - p\left(u_1 = +1\right))(1 - p\left(u_2 = +1\right)) \tag{4.5}$$

Using the following relation shown in [12].

$$p\ (u = \ +1) = \ \frac{e^{L(u)}}{1 + e^{L(u)}} \tag{4.6}$$

it can be shown that

$$p(u_1\ \oplus\ u_2 = \ +1) = \frac{1 + \ e^{L(u_1)}e^{L(u_2)}}{(1 + e^{L(u_1)})\,(1 + \ e^{L(u_2)})} \tag{4.7}$$

The probability p $(u_1\ \oplus\ u_2 = \ -1)$ can then be calculated as

$$p(u_1\ \oplus\ u_2 = \ -1) \ = \ 1 - \ p\ (u_1\ \oplus\ u_2 = \ +1) \tag{4.8}$$

$$= \ \frac{e^{L(u_1)}\ +\ e^{L(u_2)}}{(1 + e^{L(u_1)})\,(1 + \ e^{L(u_2)})} \tag{4.9}$$

From the definition of log-likelihood ratio (4.1), it follows directly that

$$L(u_1 \oplus\ u_2) = \ln \frac{p(u_1\ \oplus\ u_2 = \ +1)}{p(u_1\ \oplus\ u_2 = \ -1)} \tag{4.10}$$

Using (4.7) and (4.9), L$(u_1 \oplus\ u_2)$ is found to be

$$L(u_1 \oplus\ u_2) = \ln \frac{1 + \ e^{L(u_1)}e^{L(u_2)}}{e^{L(u_1)}\ +\ e^{L(u_2)}} \tag{4.11}$$

This result is approximated in [12] as

$$L(u_1 \oplus\ u_2) \approx sign\ \ (L\,(u_1))\ \ sign\,(L\,(u_2)) \min(|L\,(u_1)|\,, |L(u_2)|) \tag{4.12}$$

The addition of two soft or L-values is denoted by [+] and is defined as

$$L\,(u_1)\,[+]\ \ L\,(u_2) = \ L\ (u_1 \oplus\ u_2) \tag{4.13}$$

with the following three properties

$$L(u)\,[+]\;\infty = L(u) \tag{4.14}$$

$$L(u)\,[+]\,(-\infty) = L(u) \tag{4.15}$$

$$L(u)\,[+]\;0 = 0 \tag{4.16}$$

By induction, it can be shown that

$$\underset{\substack{[+]\\ j=1}}{\overset{J}{\sum}} L(u_j) = L\left(\overset{J}{\underset{\substack{\oplus\\ j=1}}{\sum}} u_j\right) \tag{4.17}$$

$$= \ln \frac{p\left(\sum_{\substack{\oplus\\ j=1}}^{J} u_j = +1\right)}{p\left(\sum_{\substack{\oplus\\ j=1}}^{J} u_j = -1\right)} \tag{4.18}$$

$$= \ln \frac{j = \prod_{j=1}^{J}(e^{L(u_j)} + 1) + \prod_{j=1}^{J}(e^{L(u_j)} - 1)}{\prod_{j=1}^{J}(e^{L(u_j)} + 1) - j = \prod_{j=1}^{J}(e^{L(u_j)} - 1)} \tag{4.19}$$

By using the relation

$$\tanh\left(\frac{x}{2}\right) = \frac{e^x - 1}{e^x + 1} \tag{4.20}$$

The induction can be simplified to

$$\sum_{\substack{[+] \\ j=1}}^{J} L\left(u_j\right) = \ln \frac{1+\prod_{j=1}^{J}\tanh(\frac{L(u_j)}{2})}{1-\prod_{j=1}^{J}\tanh(\frac{L(u_j)}{2})} \tag{4.21}$$

$$= 2\tanh^{-1}\left(\prod_{j=1}^{J}\tanh(\frac{L(u_j)}{2})\right) \tag{4.22}$$

This value is very tedious to compute. Thus, it can be approximated as (following eqn.4.12) before to

$$\sum_{\substack{[+] \\ j=1}}^{J} L\left(u_j\right) = L\left(\begin{array}{c} \sum^{J} u_j \\ \oplus \\ j=1 \end{array}\right) \tag{4.23}$$

$$\sum_{\substack{[+] \\ j=1}}^{J} L\left(u_j\right) \approx \left(\prod_{j=1}^{J}\text{sign}(L(u_j))\right) \min_{j=1,\ldots,J}\{|L(u_j)|\} \tag{4.24}$$

It can been seen from (4.24) that the reliability of the sum of soft or L-values is mainly determined by the smallest soft or L-value of the terms.

4.3.2 Soft Channel Outputs

From the system model in Figure 4.6, the information bit u is mapped to the encoded bits x. The encoded bits x are transmitted over the channel and received as y. From

this system model, the log-likelihood ratio of x conditioned on y is calculated as

$$L\left(x|y\right) = \ln \frac{p(x = \ +1|y)}{p(x = \ -1|y)} \tag{4.25}$$

By using Bayes Theorem, this log-likelihood ratio is equivalent to

$$L\left(x|y\right) = \ln \left(\frac{p\left(y \mid x = \ +1\right) \ p(x = \ +1)}{p\left(y \mid x = \ -1\right) \ p(x = \ -1)} \right) \tag{4.26}$$

$$= \ln \frac{p\left(y \mid x = \ +1\right)}{p\left(y \mid x = \ -1\right)} + \ln \frac{p(x = \ +1)}{p(x = \ -1)} \tag{4.27}$$

The channel model is assumed to be flat fading with gaussian noise. By using the Gaussian pdf f(z),

$$f\left(z\right) = \ \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{(z-m)^2}{2\sigma^2}} \tag{4.28}$$

where m is the mean and σ^2 is the variance, it can be shown that

$$\ln \frac{p\left(y \mid x = \ +1\right)}{p\left(y \mid x = \ -1\right)} = \ln \frac{e^{-\ \frac{E_b}{N_o}(y-a)^2}}{e^{-\ \frac{E_b}{N_o}(y+a)^2}} \tag{4.29}$$

$$= \ln \frac{e^{\frac{E_b}{N_o}\ 2ay}}{e^{-\ \frac{E_b}{N_o}\ 2ay}} \tag{4.30}$$

$$= \ 4\ \frac{E_b}{N_o}\ ay \tag{4.31}$$

where Eb/No is the signal to noise ratio per bit (directly related to the noise variance) and a is the fading amplitude. For non fading Gaussian channel, a=1. The log-likelihood ratio of x conditioned on y, L(x/y), is equivalent to

$$L\left(x \mid y\right) = \ L_c\ y + L(x) \tag{4.32}$$

where L_c is defined to be the channel reliability $L_c = 4\ \frac{E_b}{N_o}\ a$
Thus, L(x/y) is just the weighted received value ($L_c y$) added with the log-likelihood value of x (L(x)).

4.4 SOVA Decoder for a Turbo Code

The SOVA component decoder estimates the information sequence using one of the two encoded streams produced by the turbo code encoder. Figure 4.7 shows the inputs and outputs of the SOVA component decoder. The SOVA component decoder

Figure 4.7: SOVA component decoder.

processes the (log-likelihood ratio) inputs $\mathbf{L(u)}$ and $\mathbf{L_c y}$, where $\mathbf{L(u)}$ is the a-priori sequence of the information sequence $\mathbf{u}$ and $\mathbf{L_c y}$ is the weighted received sequence. The sequence $\mathbf{y}$ is received from the channel. However, the sequence $\mathbf{L(u)}$ is produced and obtained from the preceding SOVA component decoder. If there is no preceding SOVA component decoder then there are no a-priori values. Thus, the $\mathbf{L(u)}$ sequence is initialized to the all-zero sequence. A similar concept is also shown at the beginning of the chapter in Figure 4.1. The SOVA component decoder produces $\hat{\mathbf{u}}$ and L($\hat{\mathbf{u}}$) as outputs where $\hat{\mathbf{u}}$ is the estimated information sequence and L($\hat{\mathbf{u}}$) is the associated log-likelihood ratio (soft or L-value) sequence.

The SOVA component decoder operates similarly as the Viterbi decoder except the ML sequence is found by using a modified metric. This modified metric, which incorporates the a-priori value, is derived below [13][12].

The fundamental Viterbi algorithm searches for the state sequence $S^{(m)}$ or the information sequence $u^{(m)}$ that maximizes the a-posteriori probability $p(S^{(m)}|y)$. For binary (k=1) trellises, m can be either 1 or 2 to denote the survivor and the competing paths respectively. By using Bayes Theorem, the a-posteriori probability can be expressed as

$$p\left(S^{(m)} \mid y\right) = p\left(y \mid S^{(m)}\right) \frac{p\left(S^{(m)}\right)}{p\left(y\right)} \tag{4.33}$$

Since the received sequence $\mathbf{y}$ is fixed for metric computation and does not depend on m, it can be discarded. Thus, the maximization results to

$$\max_m p\left(y \mid S^{(m)}\right)\ p\left(S^{(m)}\right) \tag{4.34}$$

The probability of a state sequence terminating at time t is $p(S_t)$. This probability can be calculated as

$$\begin{aligned} p\left(S_t\right) &= p\left(S_{t-1}\right) p\left(S_t\right) && (4.35)\\ &= p\left(S_{t-1}\right) p\left(u_t\right) && (4.36) \end{aligned}$$

where $p(S_t)$ and $p(u_t)$ denote the probability of the state and the bit at time t respectively. The maximization can then be expanded t

$$\max_m p\left(y \mid S^{(m)}\right)\ p\left(S^{(m)}\right) = \max_m \left\{ \prod_{i=0}^{t} p\left(y_i \;\middle|\; S_{i-1}^{(m)},\ S_i^{(m)} p\left(S_t^{(m)}\right)\right)\right\} \tag{4.37}$$

where, (S_{i-1}^m, S_i^m)denotes the state transition between time i-1 and time i and $\mathbf{y_i}$ denotes the associated received channel values for the state transition.

After substituting and rearranging,

$$\max_m p\left(y \mid S^{(m)}\right) p\left(S^{(m)}\right) = \max_m \left\{ p\left(S_{t-1}^{(m)}\right) \prod_{i=0}^{t-1} p\left(y_i | S_{i-1}^{(m)},\ S_i^{(m)}\right) p\left(u_t^m\right) p\left(y_t | S_{t-1}^{(m)}, S_t^{(m)}\right) \right\} \quad (4.38)$$

Not that

$$p\left(y_t | S_{t-1}^{(m)},\ S_t^{(m)}\right) = \prod_{j=1}^{N} p\left(y_{t,j} \,\middle|\, x_{t,j}^{(m)}\right) \quad (4.39)$$

Thus the maximization becomes

$$\max_m \left\{ p\left(S_{t-1}^{(m)}\right) \prod_{i=0}^{t-1} p\left(y_i | S_{i-1}^{(m)},\ S_i^{(m)}\right)\ p\left(u_t^m\right) \prod_{j=1}^{N} p\left(y_{t,j} \,\middle|\, x_{t,j}^{(m)}\right) \right\} \quad (4.40)$$

This maximization is not changed if logarithm is applied to the whole expression, multiplied by 2, and two constants that are independent of m are added. This leads to

$$\max_m \left\{ M_t^{(m)} \right\} = \max_m \left\{ M_{t-1}^{(m)} + \left[2 \ln p\left(u_t^{(m)}\right) - C_u \right] + \sum_{j=1}^{N} \left[2 \ln p\left(y_{t,j} | x_{t,j}^{(m)}\right) - C_y \right] \right\} \quad (4.41)$$

where

$$\frac{M_{t-1}^{(m)}}{2} = \ln\left(p\left(S_{t-1}^{(m)}\right) \prod_{i=0}^{t-1} p\left(y_i | S_{i-1}^{(m)},\ S_i^{(m)}\right) \right) \quad (4.42)$$

and for convenience, the two constants are

$$C_u = \ln P\left(u_t = +1\right) + \ln P\left(u_t = -1\right) \quad (4.43)$$

$$C_y = \ln\left(P\left(y_{t,j} \mid x_{t,j} = +1\right)\right) + \ln\left(P\left(y_{t,j} \mid x_{t,j} = -1\right)\right) \quad (4.44)$$

After substitution for these two constants, the SOVA metric is obtained as

$$M_t^{(m)} = M_{t-1}^{(m)} + \sum_{j=1}^{N} x_{t,j}^{(m)} \ln \frac{P\left(y_{t,j} \mid x_{t,j} = +1\right)}{P\left(y_{t,j} \mid x_{t,j} = -1\right)} + u_t^{(m)} \ln \frac{P(u_t = +1)}{P(u_t = -1)} \quad (4.45)$$

and is reduced to

$$M_t^{(m)} = M_{t-1}^{(m)} + \sum_{j=1}^{N} x_{t,j}^{(m)} L_c y_{t,j} + u_t^{(m)} L(u_t) \quad (4.46)$$

For systematic codes, this can be modified to become

$$M_t^{(m)} = M_{t-1}^{(m)} + u_t^{(m)} L_c y_{t,1} + \sum_{j=2}^{N} x_{t,j}^{(m)} L_c y_{t,j} + u_t^{(m)} L(u_t) \quad (4.47)$$

As seen from (4.46) and (4.47), the SOVA metric incorporates values from the past metric, the channel reliability, and the source reliability (a-priori value). Figure 4.8 shows the source reliability as used in SOVA metric computation.

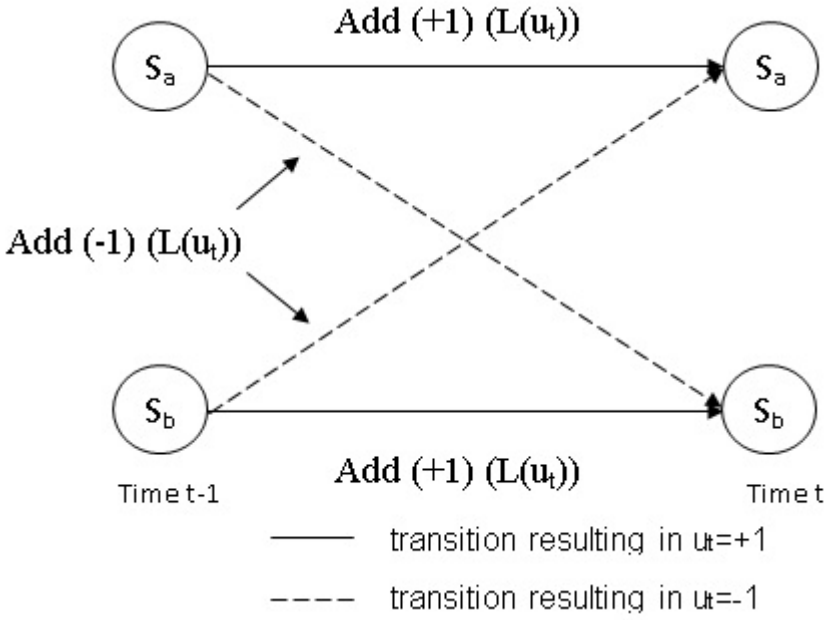

Figure 4.8: Source reliability for SOVA metric computation.

Figure 4.8 shows a trellis diagram with two states S_a and S_b and a transition period between time t-1 and time t. The solid line indicates that the transition will produce an information bit u_t =+1 and the dashed line indicates that the transition will produce an information bit u_t =-1. The source reliability L(u_t), which may be either a positive or a negative value, is from the preceding SOVA component decoder. The add on value is incorporated into the SOVA metric to provide a more reliable decision on the estimated information bit. For example, if L(u_t) is a large positive number, then it would be relatively more difficult to change the estimated bit decision from +1 to -1 between decoding stages (based on assigning $\max_m \left\{ M_t^{(m)} \right\}$ to the survivor path). However, if L(u_t) is a small positive number, then it would be relatively easier to change the estimated bit decision from +1 to -1 between decoding stages. Thus, L(u_t) is like a buffer which tries to prevent the decoder from choosing the opposite bit decision to the preceding decoder.

At time t, the reliability value (magnitude of the log-likelihood ratio) assigned to a node in the trellis is determined from

$$\triangle_t^0 = |M_t^{(1)} - M_t^{(2)}| \tag{4.48}$$

where, $\triangle_t^{MEM}$ denotes the reliability value at memorization level MEM relative to time t. This notation is similar to the notation L(t-MEM) as used earlier and is shown in Figure 4.9 for discussion. The probability of path m at time t and the SOVA metric are stated in [13] to be related as

$$p\left(path\left(m\right)\right) = p\left(S_t^{(m)}\right) \tag{4.49}$$

$$= e^{\frac{M_t^{(m)}}{2}} \tag{4.50}$$

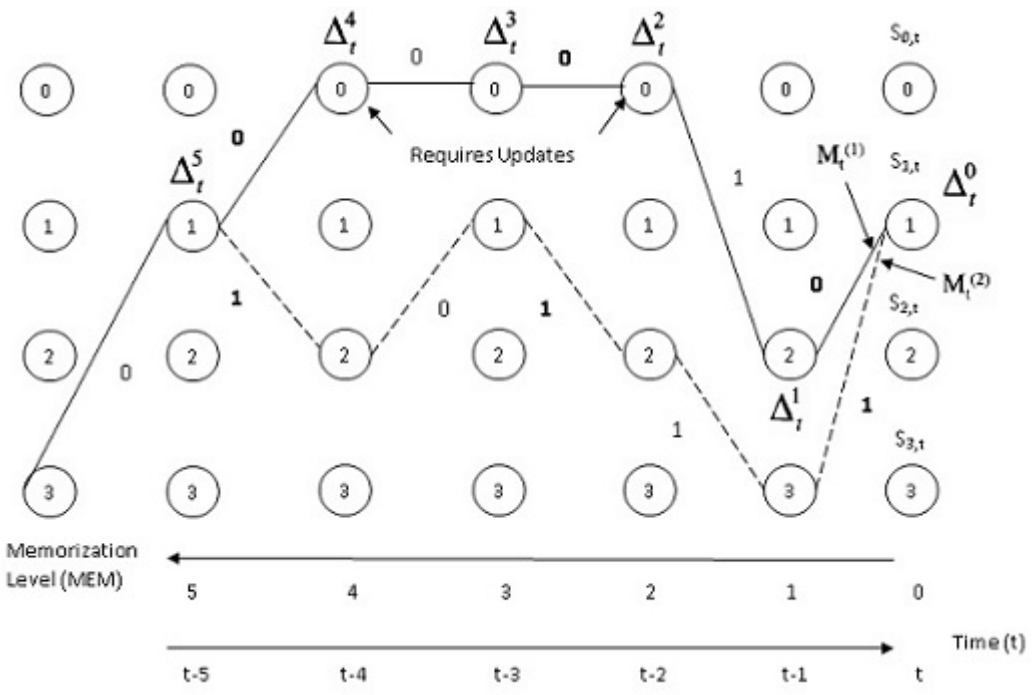

Figure 4.9: Example of SOVA survivor and competing paths for reliability estimation [12].

At time t, let us suppose that the survivor metric of a node is denoted as $M_t^{(1)}$ and the competing metric is denoted as $M_t^{(2)}$. Thus, the probability of selecting the correct survivor path is

$$p\left(correct\right) = \frac{p\left(path\left(1\right)\right)}{p\left(path\left(1\right) + path\left(2\right)\right)} \tag{4.51}$$

$$= \frac{e^{\frac{M_t^{(1)}}{2}}}{e^{\frac{M_t^{(1)}}{2}} + e^{\frac{M_t^{(2)}}{2}}} \tag{4.52}$$

$$= \frac{e^{\triangle_t^0}}{1 + e^{\triangle_t^0}} \tag{4.53}$$

The reliability of this path decision is calculated as

$$\log \frac{p\,(correct)}{1-p\,(correct)} = \log \frac{\frac{e^{\triangle_t^0}}{1+\ e^{\triangle_t^0}}}{1-\ \frac{e^{\triangle_t^0}}{1+\ e^{\triangle_t^0}}} \tag{4.54}$$

$$= \triangle_t^0 \tag{4.55}$$

The reliability values along the survivor path for a particular node at time t are denoted as $\triangle_t^{MEM}$, where MEM = 0,,t. For this node at time t, if the bit on the survivor path at MEM=k (or equivalently at time t-MEM) is the same as the associated bit on the competing path, then there would be no bit error if the competing path was chosen. Thus, the reliability value at this bit position remains unchanged. However, if the bits differ on the survivor and competing path at MEM=k, then there is a bit error. The reliability value at this bit error position must then be updated using the same updating procedure as described at the beginning of the chapter. As shown in Figure 4.9, reliability updates are required for MEM=2 and MEM=4.

The reliability updates are performed to improve the soft or L-values. It is shown in [13] that the soft or L-value of a bit decision is

$$L\,(u_{t-MEM}) = \ u_{t-MEM} \underset{[+]}{\overset{MEM}{\sum_{k=0}}} \triangle_t^k \tag{4.56}$$

and can be approximated by (4.24) to become

$$L\,(u_{t-MEM}) \approx \ u_{t-MEM} \min_{k=0,\ldots,MEM} \left\{\triangle_t^k\right\} \tag{4.57}$$

The soft output Viterbi algorithm (along with its reliability updating procedure) can now be implemented as follows:

1. (a) Initialize time t = 0.
   (b) Initialize $M_0^{(m)} = 0$ only for the zero state in the trellis diagram and all other states to - ∞.

2. (a) Set time t = t + 1.
   (b) Compute the metric $M_t^{(m)} = M_{t-1}^{(m)} + u_t^{(m)} L_c y_{t,1} + \sum_{j=2}^{N} x_{t,j}^{(m)} L_c y_{t,j} + u_t^{(m)} L(u_t)$ for each state in the trellis diagram where, m denotes allowable binary trellis branch/transition to a state (m = 1,2).

   $M_t^{(m)}$ is the accumulated metric for time t on branch m.
   $u_t^{(m)}$ is the systematic bit (1^{st} bit on N bits) for time t on branch m.
   $x_{t,j}^{(m)}$ is the j-th bit on N bits for time t on branch m ($2 \leq j \leq N$).
   $y_{t,j}^{(m)}$ is the received value from the channel corresponding to $x_{t,j}^{(m)}$.
   $L_c = 4 \frac{E_b}{N_o}$ is the channel reliability value.
   $L(u_t)$ is the a-priori reliability value for time t. This value is from the preceding decoder. If there is no preceding decoder, then this value is set to zero.

3. Find $\max_m M_t^{(m)}$ for each state. For simplicity, let $M_t^{(1)}$ denote the survivor path metric and $M_t^{(2)}$ denote the competing path metric.

4. Store $M_t^{(1)}$ and its associated survivor bit and state paths.

5. Compute $\triangle_t^0 = \frac{1}{2}|M_t^{(1)} - M_t^{(2)}|$

6. Compare the survivor and computing paths at each state for time t and store the MEMs where the estimated binary decisions of the two paths differ.

7. Update $\triangle_t^{MEM} \approx \min_{k=0,\ldots,MEM} \left\{\triangle_t^k\right\}$ for all MEMs from smallest to largest MEM.

8. Go back to Step (2) until the end of the received sequence.

9. Output the estimated bit sequence and its associated Soft or L-value sequence.

$L(\hat{u}) = \hat{u} * \triangle$, where,* operator defines element by element multiplication operation and $\triangle$ is the final updated reliability sequence. $L(\hat{u})$ is then processed (to be discussed later) and passed on as the a-priori sequence L(u) for the succeeding decoder.

4.5 SOVA Iterative Turbo Code Decoder

The iterative turbo code decoder is composed of two concatenated SOVA component decoders [13] [12]. Figure 4.10 shows the turbo code decoder structure. The turbo

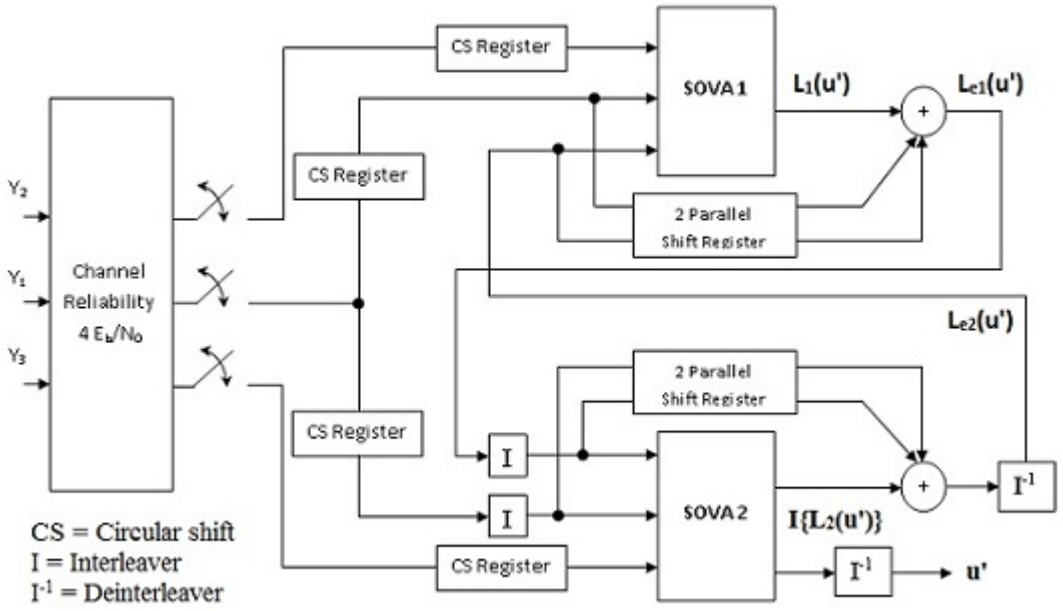

Figure 4.10: SOVA iterative turbo code decoder [13].

code decoder processes the received channel bits on a frame basis. As shown in Figure 4.10, the received channel bits are demultiplexed into the systematic stream y_1 and two parity check streams y_2 and y_3 from component encoders 1 and 2 respectively. These bits are weighted by the channel reliability value and loaded on to the CS registers. The registers shown in the figure are used as buffers to store sequences. The switches are placed in the open position to prevent the bits from the next frame

from being processed until the present frame has been processed.

The SOVA component decoder produces the soft or L-value $L(u'_t)$ for the estimated bit u'_t (for time t). The soft or L- value $L(u'_t)$ can be decomposed into three distinct terms as stated in [12].

$$L\left(u'_t\right) = L\left(u_t\right) + \ L_c y_{t,1} + \ L_e\left(u'_t\right) \tag{4.58}$$

$L(u_t)$ is the a-priori value and is produced by the preceding SOVA component encoder.
$L_c y_{t,1}$ is the weighted received systematic channel value.
$L_e(u'_t)$ is the extrinsic value produced by the present SOVA component decoder.
The information that is passed between SOVA component decoders is the extrinsic value

$$L_e\left(u'_t\right) = \ L\left(u'_t\right) - \ L\left(u_t\right) - \ L_c y_{t,1} \tag{4.59}$$

The a-priori value$L(u_t)$ is subtracted out from the soft or L -value $L(u'_t)$ to prevent passing information back to the decoder from which it was produced. Also, the weighted received systematic channel value $L_c y_{t,1}$ is subtracted out to remove common information in the SOVA component decoders.

Figure 4.10 shows that the turbo code decoder is a closed loop serial concatenation of SOVA component decoders. In this closed loop decoding scheme, each of the SOVA component decoders estimates the information sequence using a different weighted parity check stream. The turbo code decoder further implements iterative decoding to provide more dependable reliability/a-priori estimations from the two different weighted parity check streams, hoping to achieve better decoding performance.

The iterative turbo code decoding algorithm for the n-th iteration is as follows:

1. The SOVA1 decoder inputs sequences 4 $\frac{E_b}{N_o}y_1$ (systematic),4 $\frac{E_b}{N_o}y_2$ (parity check), and $L_{e2}(u')$ and outputs sequence $L_1(u')$ .For the first iteration, sequence $L_{e2}(u') = 0$ because there is no initial a-priori value (no extrinsic values from SOVA2).

2. The extrinsic information from SOVA1 is obtained by
   $L_{e1}(u') = L_1(u') - L_{e2}(u') - L_c y_1$, where $L_c = 4\ \frac{E_b}{N_o}$

3. The sequences 4 $\frac{E_b}{N_o}y_1$ and $L_{e1}(u')$ are interleaved and denoted as $I\left\{4\ \frac{E_b}{N_0}y_1\right\}$ and $I\ \{L_{e1}(u')\}$

4. The SOVA2 decoder inputs sequences $I\left\{4\ \frac{E_b}{N_0}y_1\right\}$ (systematic), $I\left\{4\ \frac{E_b}{N_0}y_3\right\}$ (parity check that was already interleaved by the turbo code encoder), and $I\ \{L_{e1}(u')\}$ (a-priori information) and outputs sequences $I\ \{L_2(u')\}$ and $I\ \{u'\}$.

5. The extrinsic information from SOVA2 is obtained by
   $I\ \{L_{e2}(u')\} = I\ \{L_2(u')\} - I\ \{L_{e1}(u')\} - I\ \{L_c y_1\}$

6. The sequences $I\ \{L_{e2}(u')\}$ and $I\ \{u'\}$ are deinterleaved and denoted as $L_{e2}(u')$ and u'. $L_{e2}(u')$ is fed back to SOVA1 as a-priori information for the next iteration and u' is the estimated bits output for the n-th iteration.

Chapter 5

Performance Analysis

This chapter illustrates the performance of the turbo codes and convolutional codes through computer simulation. MATLAB is used to construct the computer code, and the simulations were run in MATLAB environment.

5.1 Simulation Setup

The simulation setup is composed of three distinct parts, namely the encoder, the channel, and the decoder. The simulated convolutional encoder is use modulo-2 adder and shift register or constraint length (K) with code memory, of size m. In the simulation, the Additive White Gaussian Noise (AWGN) channel is used and Viterbi decoder is used as convolutional decoder for hard decision as well as soft decision.

Second, the simulation for the Turbo code encoder is composed of two identical RSC component encoders. These two component encoders are separated by a random interleaver. The random interleaver is a permutation of bit order in a bit stream as shown in chapter 3. This permutation of bit order is stored so that the interleaved bit stream can be deinterleaved at the decoder. The output of the turbo code encoder is described by three streams, one systematic (uncoded) bit stream and two coded bit streams (parity bits), which is shown in chapter 3 and SOVA decoder is used shown

in chapter 4.

In its basic form, the turbo code encoder is rate $\frac{1}{3}$ and convolutional encoder is rate $\frac{1}{2}$. However, in many journal papers, the published computer simulations of above codes often use rate $\frac{1}{2}$, $\frac{1}{3}$, $\frac{2}{3}$ and $\frac{3}{4}$. This is accomplished by puncturing the coded bit streams of the above code.

5.2 Simulation Results

Simulation results for a convolutional code are based on bit error rate (BER) performance over a range of Eb/No. The BER is simply the ratio of incorrect data bits divided by the total number of data bits transmitted. The SNR is computed by dividing the energy per received data bit Eb by the single-sided noise spectral density No of the channel. For simulation rate $\frac{1}{2}$, $\frac{2}{3}$ and $\frac{1}{3}$ convolutional codes and rate $\frac{1}{3}$ turbo codes using two $\frac{1}{2}$ RSC encoder and interleaver is used.

First, the simulation result are shown for convolutional code using different modulation technique (BPSK, QPSK and 8-PSK) with rate $\frac{1}{2}$ and different rate ($\frac{1}{2}$, $\frac{1}{3}$ and $\frac{2}{3}$) using one modulation technique (BPSK) in AWGN channel. The simulation is carried out on the basis of BER improvement over uncoded BER given by equation:

$$\mid \frac{\mathbf{BER}_{uncoded} - \mathbf{BER}_{Coded}}{\mathbf{BER}_{uncoded}} \mid \times 100\ \%$$

Second, the performance of rate $\frac{1}{3}$ turbo code encoder is shown with RSC encoder and SOVA decoder.

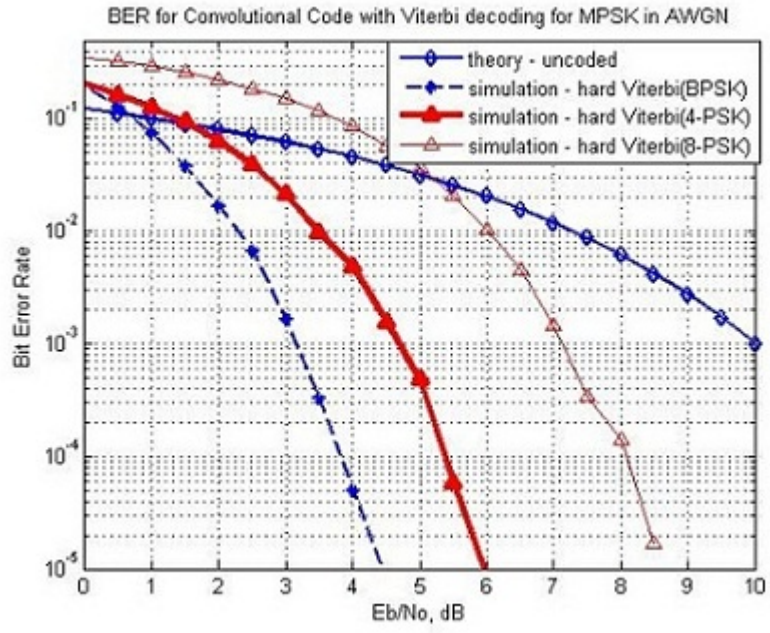

Figure 5.1: Convolutional code (rate (R)-1/2) with Different Modulation in AWGN channel

Eb/No (dB)	**Uncoded (BER)**	**BPSK (BER)**	**BER improvement over Uncoded**
1.5	4.46×10^{-2}	4.0×10^{-2}	13.8 %
2.5	2.9×10^{-2}	6.1×10^{-3}	79.46 %
3.5	1.7×10^{-2}	3.1×10^{-4}	98.16 %

Table 5.1: Result of convolutional code with BPSK modulation of Figure 5.1

Eb/No (dB)	**Uncoded (BER)**	**QPSK (BER)**	**BER improvement over Uncoded**
3	2.29×10^{-2}	2.16×10^{-2}	5.44 %
4	1.25×10^{-2}	4.0×10^{-3}	68.27 %
5	6.0×10^{-3}	3.33×10^{-4}	94.4 %

Table 5.2: Result of convolutional code with QPSK modulation of Figure 5.1

Eb/No (dB)	Uncoded (BER)	8-PSK (BER)	BER improvement over Uncoded
6	2.05×10^{-2}	1.08×10^{-2}	47.11 %
7	1.2×10^{-2}	1.6×10^{-3}	86.34 %
8	6.2×10^{-3}	1.16×10^{-4}	98.11 %

Table 5.3: Result of convolutional code with 8-PSK modulation of Figure 5.1

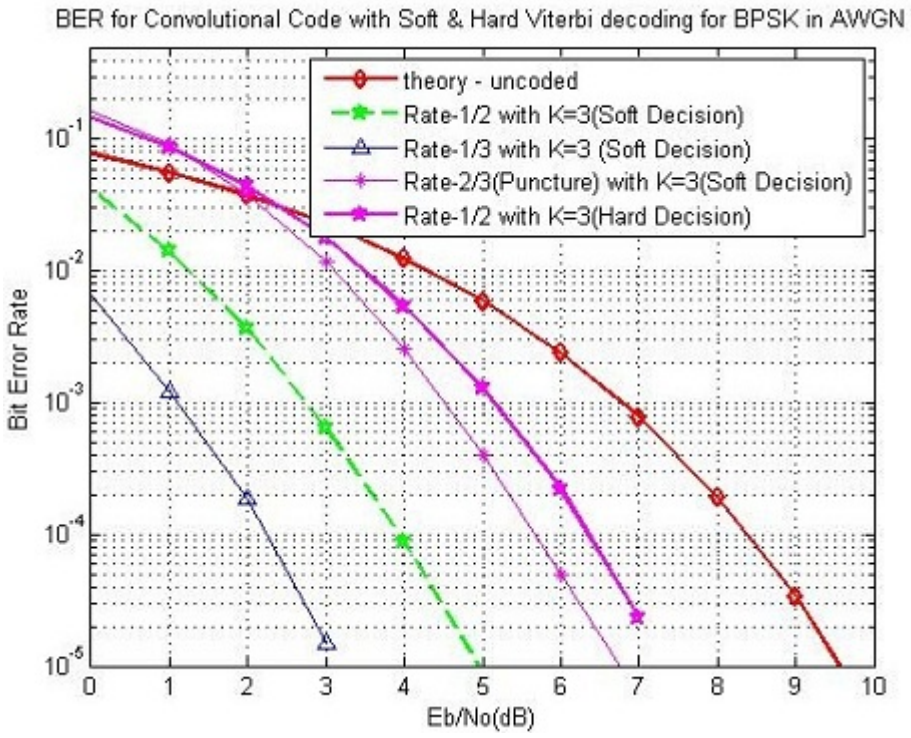

Figure 5.2: Convolutional code with Different Coding Rate in BPSK and AWGN channel

Eb/No (dB)	Uncoded (BER)	Rate-1/2 (BER) (Soft Decision)	BER improvement over Uncoded
3	2.29×10^{-2}	6.88×10^{-4}	96.99 %
4	1.25×10^{-2}	5.8×10^{-5}	99.54 %
5	6.0×10^{-3}	1.8×10^{-5}	99.70 %

Table 5.4: Result for Rate-$\frac{1}{2}$ soft decision convolutional code of Figure 5.2

Eb/No (dB)	**Uncoded (BER)**	**Rate-1/2 (BER) (Hard Decision)**	**BER improvement over Uncoded**
6	2.4×10^{-3}	2.19×10^{-4}	90.83 %
7	7.72×10^{-4}	3.1×10^{-5}	95.99 %
8	1.9×10^{-4}	2.0×10^{-6}	98.95 %

Table 5.5: Result for Rate-$\frac{1}{2}$ hard decision convolutional code of Figure 5.2

Eb/No (dB)	**Uncoded (BER)**	**Rate-1/3 (BER) (Soft Decision)**	**BER improvement over Uncoded**
1	5.63×10^{-2}	1.3×10^{-3}	97.78 %
2	3.75×10^{-2}	2.07×10^{-4}	99.45 %
3	2.29×10^{-2}	1.7×10^{-5}	99.93 %

Table 5.6: Result for Rate-$\frac{1}{3}$ soft decision convolutional code of Figure 5.2

Eb/No (dB)	**Uncoded (BER)**	**Rate-2/3 (BER) (Hard Decision)**	**BER improvement over Uncoded**
5	6.0×10^{-3}	4.24×10^{-4}	92.88 %
6	2.4×10^{-3}	5.2×10^{-5}	97.82 %
7	7.72×10^{-4}	3.0×10^{-6}	99.61 %

Table 5.7: Result for Rate-$\frac{2}{3}$ (with puncture) soft decision convolutional code of Figure 5.2

Eb/No (dB)	**Uncoded**	**R = 1/3,K = 3**	**R = 1/2,K = 3**
0.5	6.9×10^{-2}	9.81×10^{-2}	8.16×10^{-2}
1	5.7×10^{-2}	7.92×10^{-2}	5.98×10^{-2}
1.5	4.6×10^{-2}	3.79×10^{-2}	3.95×10^{-2}
2	3.9×10^{-2}	2.65×10^{-2}	2.83×10^{-2}
2.5	2.8×10^{-2}	1.14×10^{-2}	1.3×10^{-2}
3	2.4×10^{-2}	3.42×10^{-3}	1.18×10^{-2}
3.5	1.7×10^{-2}	2.43×10^{-3}	5.36×10^{-3}
4	1.3×10^{-2}	6.23×10^{-4}	4.088×10^{-3}

Table 5.8: Comparison of turbo code BER performance for different rate (FS = 256, iter =1)

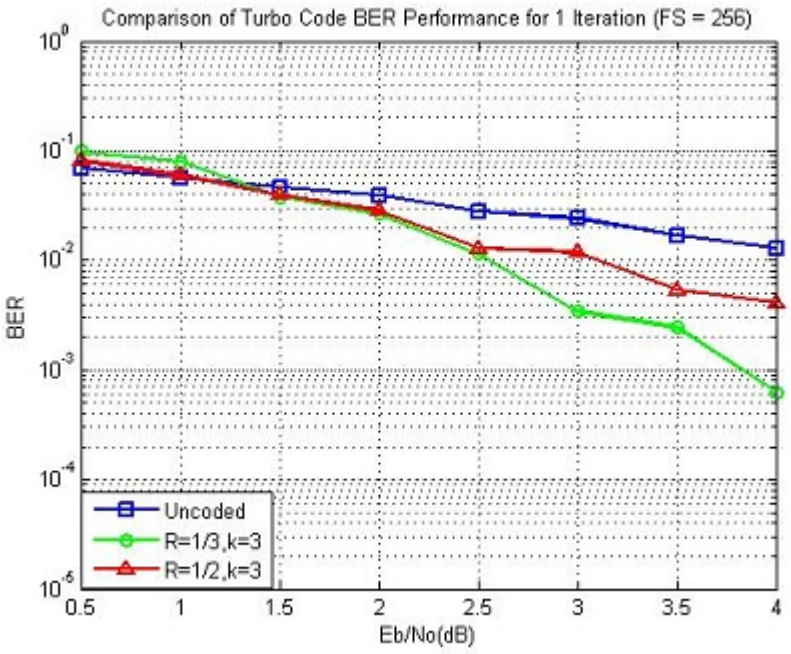

Figure 5.3: Performance of turbo code with different code rate

Eb/No (dB)	**Uncoded**	**iter = 1**	**iter = 8**
0.5	6.9×10^{-2}	9.81×10^{-2}	5.17×10^{-2}
1	5.7×10^{-2}	7.92×10^{-2}	2.4×10^{-2}
1.5	4.6×10^{-2}	3.79×10^{-2}	3.31×10^{-3}
2	3.9×10^{-2}	2.65×10^{-2}	7.82×10^{-4}
2.5	2.8×10^{-2}	1.14×10^{-2}	3.69×10^{-4}
3	2.4×10^{-2}	3.42×10^{-3}	8.92×10^{-5}
3.5	1.7×10^{-2}	2.43×10^{-3}	2.81×10^{-5}
4	1.3×10^{-2}	6.23×10^{-4}	1.22×10^{-5}

Table 5.9: Comparison of turbo code BER performance for different iter (FS = 256, R = 1/3,K = 3)

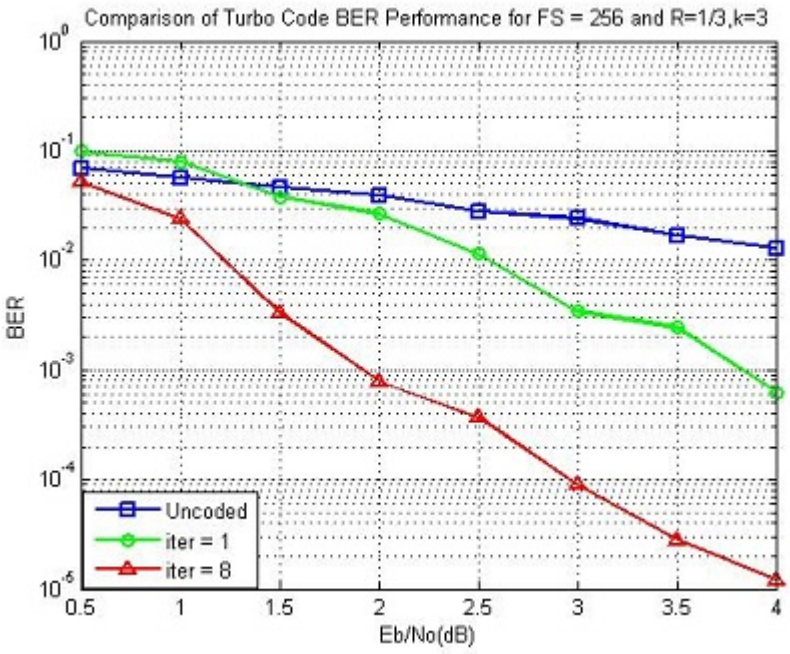

Figure 5.4: Performance of turbo code with different iteration

Eb/No (dB)	**Uncoded**	**Frame Size = 256**	**Frame Size = 1024**
0.5	6.9×10^{-2}	5.17×10^{-2}	3.68×10^{-2}
1	5.7×10^{-2}	2.4×10^{-2}	3.33×10^{-3}
1.5	4.6×10^{-2}	3.31×10^{-3}	2.87×10^{-4}
2	3.9×10^{-2}	7.82×10^{-4}	7.51×10^{-5}
2.5	2.8×10^{-2}	3.69×10^{-4}	3.69×10^{-5}
3	2.4×10^{-2}	8.92×10^{-5}	1.67×10^{-5}
3.5	1.7×10^{-2}	2.81×10^{-5}	5.58×10^{-6}
4	1.3×10^{-2}	1.22×10^{-5}	2.15×10^{-6}

Table 5.10: Comparison of turbo code BER performance for different FS (iter = 8, R = 1/3,K = 3)

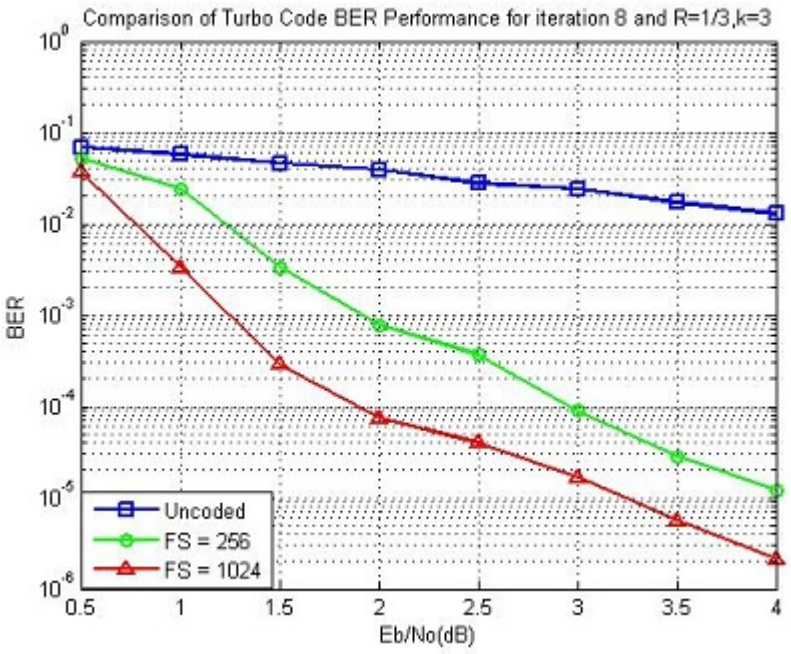

Figure 5.5: Performance of turbo code with different frame size

Eb/No (dB)	**Uncoded**	**R = 1/3,K = 3**	**R = 1/3,K = 4**	**R = 1/3,K = 5**
0.5	6.9×10^{-2}	9.81×10^{-2}	1.40×10^{-1}	1.27×10^{-1}
1	5.7×10^{-2}	7.92×10^{-2}	6.99×10^{-2}	1.01×10^{-1}
1.5	4.6×10^{-2}	3.79×10^{-2}	4.65×10^{-2}	5.7×10^{-2}
2	3.9×10^{-2}	2.65×10^{-2}	2.53×10^{-2}	1.44×10^{-2}
2.5	2.8×10^{-2}	1.14×10^{-2}	6.64×10^{-3}	5.68×10^{-3}
3	2.4×10^{-2}	3.42×10^{-3}	2.47×10^{-3}	8.07×10^{-4}
3.5	1.7×10^{-2}	2.43×10^{-3}	4.76×10^{-4}	3.1×10^{-4}
4	1.3×10^{-2}	6.23×10^{-4}	1.22×10^{-4}	8.13×10^{-5}

Table 5.11: Comparison of turbo code BER performance for different constraint length (FS = 256, iter =1)

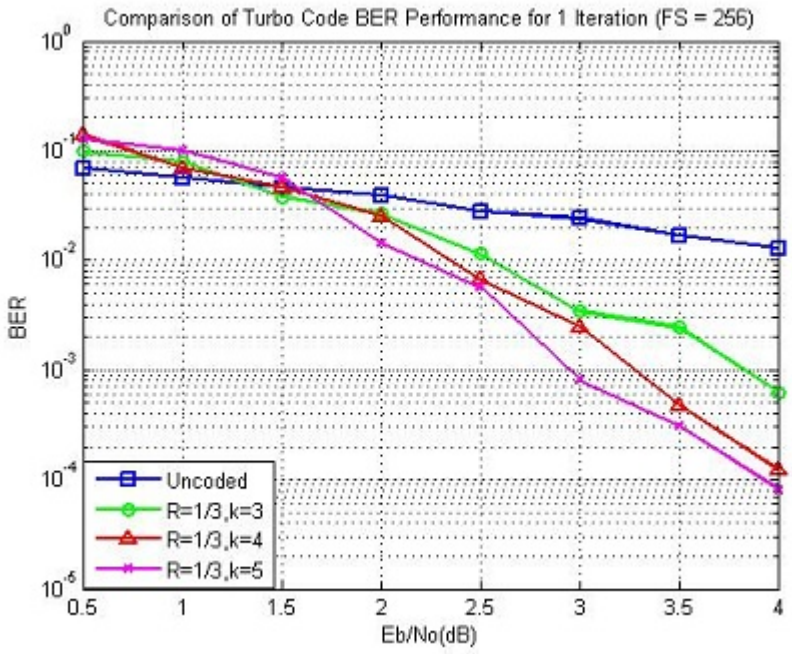

Figure 5.6: Performance of turbo code with different constraint length

Eb/No (dB)	Uncoded	Published	Simulated
0	8.12×10^{-2}	1.1×10^{-1}	8.81×10^{-2}
1	5.7×10^{-2}	6×10^{-2}	6.88×10^{-2}
2	3.9×10^{-2}	1.7×10^{-2}	1.95×10^{-2}
3	2.4×10^{-2}	4×10^{-3}	7.27×10^{-3}
4	1.3×10^{-2}	9.12×10^{-4}	3.37×10^{-3}
5	5.67×10^{-3}	1.12×10^{-4}	7.91×10^{-4}
6	2.33×10^{-3}	1.1×10^{-5}	8.62×10^{-5}

Table 5.12: Comparison of turbo code BER performance for (FS = 64, iter = 1, R = 1/2, K = 3) [14]

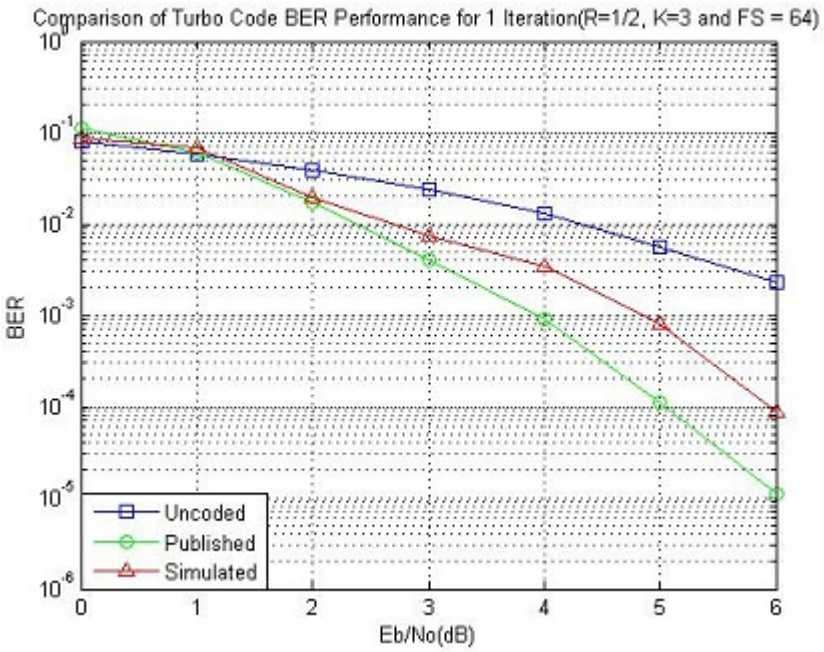

Figure 5.7: Comparison of turbo code BER performance for (FS = 64, iter = 1, R = 1/2, K = 3) [14]

Eb/No (dB)	Uncoded	Published	Simulated
0	8.12×10^{-2}	2.2×10^{-1}	1.64×10^{-1}
1	5.7×10^{-2}	1.08×10^{-1}	1.07×10^{-1}
2	3.9×10^{-2}	8.15×10^{-2}	2.27×10^{-2}
3	2.4×10^{-2}	2.03×10^{-2}	3.87×10^{-3}
4	1.3×10^{-2}	3.2×10^{-3}	2.75×10^{-4}
5	5.67×10^{-3}	2.24×10^{-4}	3.13×10^{-5}

Table 5.13: Comparison of turbo code BER performance for (FS = 192, iter = 1, R = 1/2, K = 5) [14]

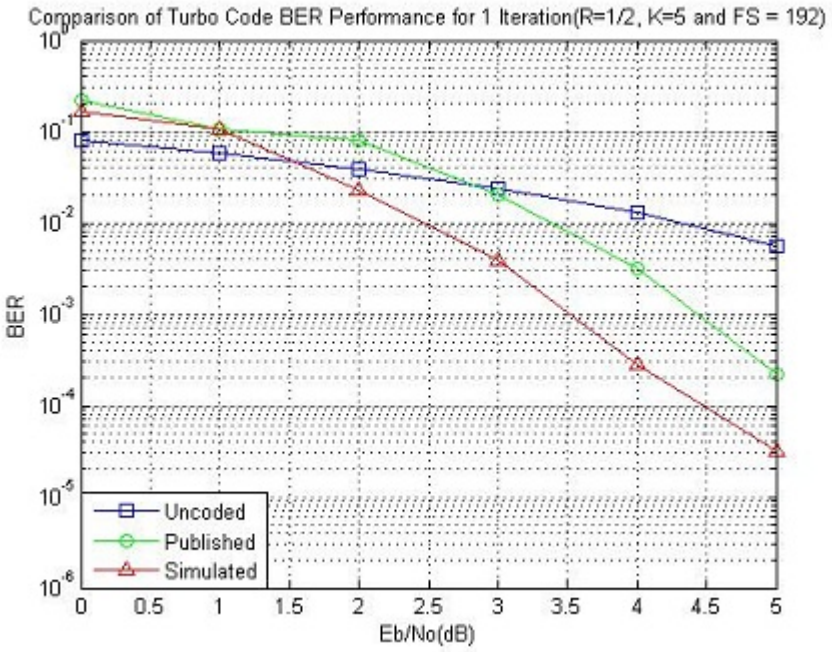

Figure 5.8: Comparison of turbo code BER performance for (FS = 192, iter = 1, R = 1/2, K = 5) [14]

Eb/No (dB)	Uncoded	Published	Simulated
0	8.12×10^{-2}	1.8×10^{-1}	1.60×10^{-1}
1	5.7×10^{-2}	9×10^{-2}	6.89×10^{-2}
2	3.9×10^{-2}	3.84×10^{-2}	1.63×10^{-2}
3	2.4×10^{-2}	7.08×10^{-3}	5.03×10^{-4}
4	1.3×10^{-2}	7.84×10^{-3}	4.24×10^{-5}

Table 5.14: Comparison of turbo code BER performance for (FS = 192, iter = 5, R = 1/2, K = 5) [14]

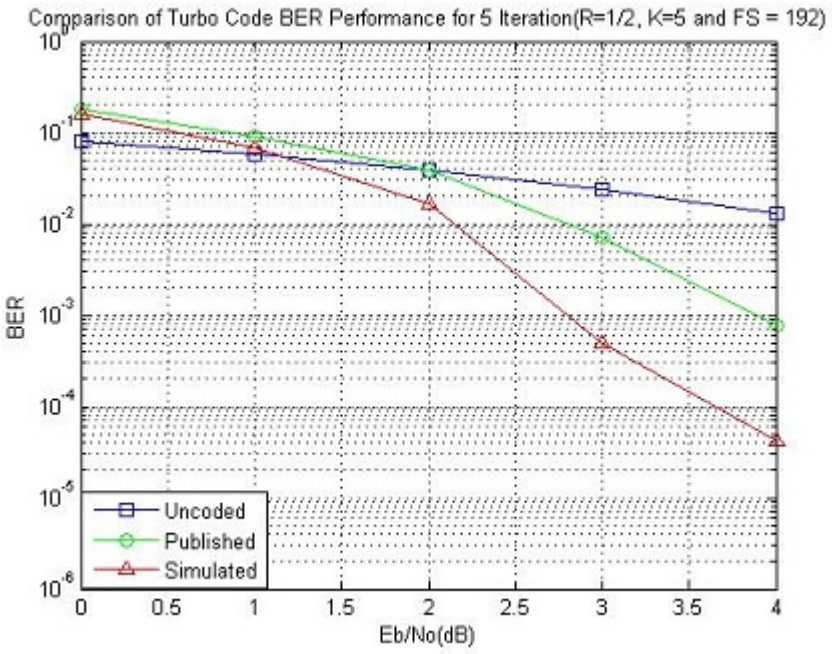

Figure 5.9: Comparison of turbo code BER performance for (FS = 192, iter = 5, R = 1/2, K = 5) [14]

Eb/No (dB)	Uncoded	Published	Simulated
0	8.12×10^{-2}	2.08×10^{-1}	1.8×10^{-1}
1	5.7×10^{-2}	1.15×10^{-1}	1.08×10^{-1}
2	3.9×10^{-2}	7.89×10^{-2}	2.28×10^{-2}
3	2.4×10^{-2}	2.25×10^{-2}	1.18×10^{-3}
4	1.3×10^{-2}	9.89×10^{-4}	1.02×10^{-4}
5	5.67×10^{-3}	5.89×10^{-5}	8.73×10^{-6}

Table 5.15: Comparison of turbo code BER performance for (FS = 2048, iter = 1, R = 1/2, K = 5) [14]

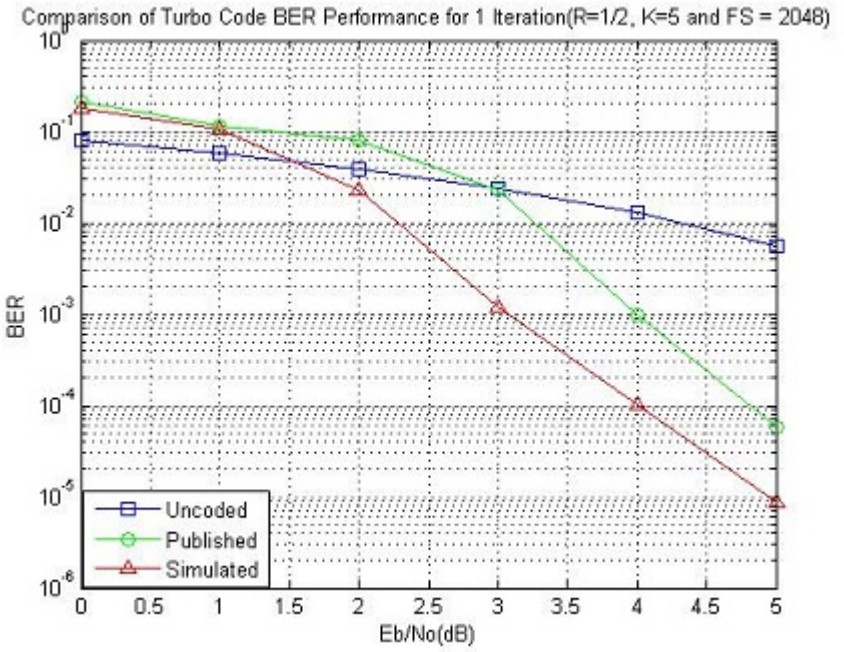

Figure 5.10: Comparison of turbo code BER performance for (FS = 2048, iter = 1, R = 1/2, K = 5) [14]

Eb/No (dB)	Uncoded	Published	Simulated
0	8.12×10^{-2}	1.03×10^{-1}	1.76×10^{-1}
1	5.7×10^{-2}	9.89×10^{-2}	1.02×10^{-1}
2	3.9×10^{-2}	6.08×10^{-2}	4.18×10^{-4}
3	2.4×10^{-2}	8.76×10^{-4}	7.1×10^{-6}
4	1.3×10^{-2}	5.67×10^{-6}	2.2×10^{-6}

Table 5.16: Comparison of turbo Code BER Performance for (FS = 2048, iter = 5, R = 1/2, K = 5) [14]

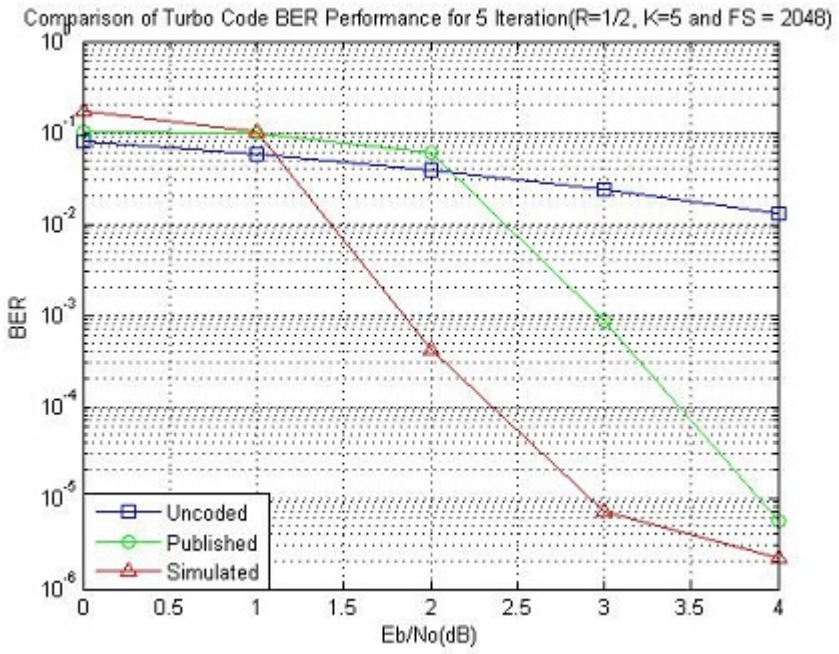

Figure 5.11: Comparison of turbo Code BER Performance for (FS = 2048, iter = 5, R = 1/2, K = 5) [14]

Eb/No (dB)	Uncoded	Published	Simulated
1	5.7×10^{-2}	9.87×10^{-2}	8.83×10^{-2}
1.5	4.6×10^{-2}	4.68×10^{-2}	5.34×10^{-2}
2	3.9×10^{-2}	1.77×10^{-2}	3.44×10^{-2}
2.5	2.8×10^{-2}	8.68×10^{-3}	1.35×10^{-2}

Table 5.17: Comparison of turbo Code BER Performance for (FS = 1024, iter = 1, R = 1/2, K = 5) [15]

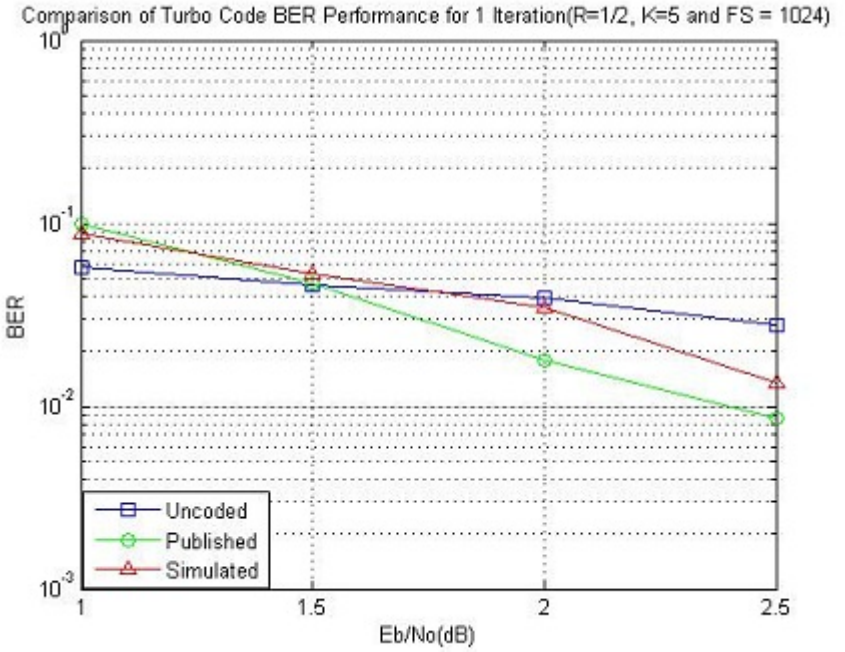

Figure 5.12: Comparison of turbo Code BER Performance for (FS = 1024, iter = 1, R = 1/2, K = 5) [15]

Eb/No (dB)	Uncoded	Published	Simulated
1	5.7×10^{-2}	7.02×10^{-2}	1.017×10^{-1}
1.5	4.6×10^{-2}	9.43×10^{-3}	1.13×10^{-2}
2	3.9×10^{-2}	1.23×10^{-4}	4.408×10^{-4}
2.5	2.8×10^{-2}	9.93×10^{-6}	3.02×10^{-5}

Table 5.18: Comparison of turbo Code BER Performance for (FS = 1024, iter = 8, R = 1/2, K = 5) [15]

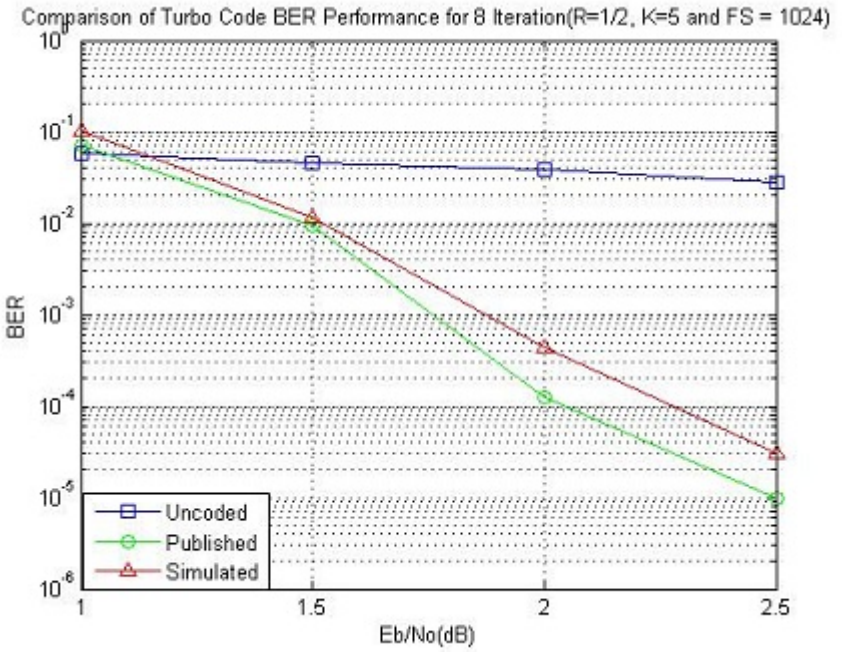

Figure 5.13: Comparison of turbo Code BER Performance for (FS = 1024, iter = 8, R = 1/2, K = 5) [15]

Eb/No (dB)	Uncoded	Published	Simulated
1.5	4.6×10^{-2}	8.09×10^{-3}	1.6×10^{-2}
2	3.9×10^{-2}	7.83×10^{-4}	3.34×10^{-3}
2.5	2.8×10^{-2}	9.15×10^{-5}	8.80×10^{-4}

Table 5.19: Comparison of turbo Code BER Performance for (FS = 1024, iter = 8, R = 1/2, K = 3) [16]

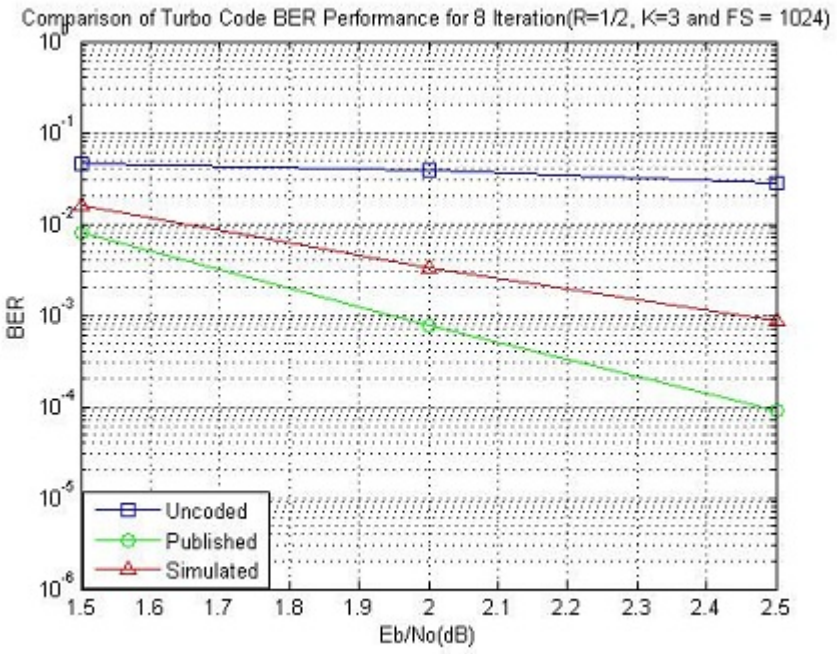

Figure 5.14: Comparison of turbo Code BER Performance for (FS = 1024, iter = 8, R = 1/2, K = 3) [16]

Eb/No (dB)	Uncoded	Published	Simulated
1.5	4.6×10^{-2}	1.01×10^{-2}	1.13×10^{-2}
2	3.9×10^{-2}	2.43×10^{-4}	4.408×10^{-4}
2.5	2.8×10^{-2}	1.89×10^{-5}	3.02×10^{-5}

Table 5.20: Comparison of turbo Code BER Performance for (FS = 1024, iter = 8, R = 1/2, K = 5) [16]

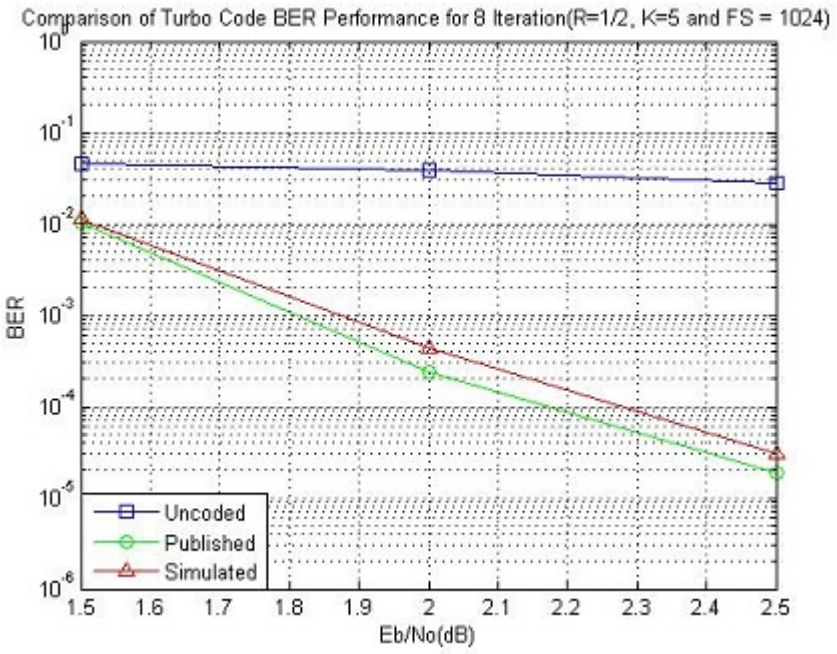

Figure 5.15: Comparison of turbo Code BER Performance for (FS = 1024, iter = 8, R = 1/2, K = 5) [16]

5.3 Simulation Analysis

The simulated performance results of convolutional codes are shown in Figure 4.1 and 4.2.

From Figure 4.1, it shows that when BPSK modulation is used it results in BER improvement over uncoded is 98.16% at 3.5 dB, when QPSK modulation is used it results in BER improvement over uncoded is 94.4% at 5 dB and 8-PSK modulation is used then BER improvement over uncoded is 98.11% at 8 dB.

From Figure 4.2, it shows that when rate 1/2 is used it results in BER improvement over uncoded is 99.70% at 5 dB for soft decision on other hand it gives 98.95% at 8 dB for hard decision and rate 1/3 and punctured rate 2/3 is used it results in BER improvement over uncoded is 99.93% at 3 dB and 99.61% at 7 dB for soft decision.

The simulated performance results of turbo code are shown in Figure 5.3, Figure 5.4, Figure 5.5 and Figure 5.6.

The simulated performance results of turbo codes with fixed frame sizes but different rates are shown in Figure 5.3. From these figures, it can be seen that for a fixed constraint length, a decrease in code rate increases the turbo code performance.

The simulated performance results of turbo codes with fixed frame sizes, fixed constraint length and fixed rate are shown in Figure 5.4. From figure an increase in number of iteration improves the turbo code performance.

The overall iterative (8 iterations) decoding gain for a turbo code with the same constraint length and rates but different frame sizes are shown in Figure 5.5. As

shown in these figures, the overall iterative decoding gain increases as the frame size increases.

The performance of the rate 1/3 turbo code in soft decision Viterbi decoding for different constraint lengths is shown in Figure 5.6. In this figure, it can be seen that as the constraint length increases, the performance of the code also increases, resulting in lower BER. This is the typical characteristic of any convolutional code.

Finally Figure 5.7 to Figure 5.15 present a compression of the results obtained in this study with performance results for various specific cases [14] [15] [16]. This and the above mentioned characteristics of the simulated turbo code performance results indicate that results obtained are consistent with published results.

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

The thesis describes the concept of error correction codes, the convolutional codes and turbo codes. The basic configuration of convolutional codes depends upon the shift registers used in the encoder, the Viterbi decoder and the turbo codes encoder which used two 1/2 rate RSC encoder with interleaver whereas the turbo codes decoder used the Soft Output Viterbi Algorithm (SOVA).

From simulation of convolutional codes,

1. It shows that use of soft decision gives improvement in coding gain and secondly with the decrease in code rate results in the increase of coding gain.

2. Another simulation shows that when channel noise is more, then BPSK modulation gives significant improvement but low bit rate, while on other hand when channel noise is less then QPSK and 8-PSK modulation is used to improve bit rate.

The thesis described the concept of turbo coding, whose basic configuration depends on the parallel concatenation of two component codes (RSC). The three distinct terms

of "Soft" or L-value are reviewed, and these values were used for the information exchange between the two SOVA component decoders.

The BER performance for turbo codes is investigated for many different cases. These different cases are summarized under the following three main categories:

1. Turbo code BER performance of 8 decoding iterations for fixed code rates and constraint lengths but different frame sizes.

2. Turbo code BER performance of 1 decoding iterations for fixed frame sizes but different code rates and constraint lengths.

3. Turbo code BER performance improvement between 1 decoding iteration and 8 decoding iterations for fixed code rates, fixed constraint lengths and fixed frame sizes.

The simulation results showed many interesting properties about turbo codes that are in the same direction with published research work. Some of these important results are listed below:

1. For a fixed turbo code encoder, its performance improves as the frame size increases.

2. For a fixed frame size, the turbo code performance increases under two different conditions. First, for a fixed constraint length, a decrease in code rate improves the performance. Second, for a fixed code rate, an increase in constraint length improves the performance.

3. Considerable decoding gain is observed if more than one decoding iterations is used.

6.2 Contribution

I have studied, analyzed and implemented the convolutional codes and turbo codes. The simulations of the convolutional codes and turbo codes have been carried out in the MATLAB.

6.3 Future Scope

There are many possibilities for future work in turbo codes. As described in the thesis, some detailed work needs to be done on the aspect of information transfer between the SOVA component decoders. For further improvements in turbo code, research should be focused on the joint issues of improving decoder performance and reducing decoder complexity. Also, in addition to Gaussian channel model developing other channel model is important. Furthermore, for better performance of turbo code, SISO module can be implemented by using MAP algorithm instead of SOVA.

References

[1] S. Haykin. *Communication Systems.* John Wiley and Sons, Inc., 4^{th} edition.

[2] B. Sklar. *Digital Communication Fundamentals and Applications.* Prentice Hall, 2^{nd} edition.

[3] P. Elias. Error-free coding. *IRE Trans. Inform.Theory, vol. IT-4*, pp. 29-37, 1954.

[4] Jr. G. D. Forney. Convolutional codes i: Algebraic structure. *IEEE Trans. Inform Theory, vol. IT-16, no.6*, pp. 720-738, 1970.

[5] J. Geldmacher K. Hueske and J. Gotze. Adaptive decoding of convolutional codes. *Adv. Radio Sci., 5*, pp. 209-214, 2007.

[6] J. C. Moreira and P. G. Farrell. *Essential of Error-Control Coding.* John Wiley and Sons, Inc., 2^{nd} edition.

[7] A. Glavieux C. Berrou and P. Thitimajshima. Near shannon limit error-correcting coding and decoding: Turbo codes(1). *in ICC'93*, pp. 1064-1070, 1993.

[8] J. Xu C. Tanriover, B. Honary and S. Lin. Improving turbo code error performance by multifold coding. *IEEE Comm., Letters, vol. 6, no. 5*, pp. 193-195, 2002.

[9] M. Salehi H. R. Sadjadpour, J. A. Sloane and G. Nebe. Interleaver design for turbo codes. *IEEE J.Select. Areas Commun., vol.19, no.5*, pp. 831-837, 2001.

[10] J. Hagenauer and P. Hoeher. A viterbi algorithm with soft-decision outputs and its applications. *in Proc., IEEE Globecom Conj*, pp. 1680-1686, 1989.

[11] Claude Berrou. A low complexity soft-output viterbi decoder architecture. *IEEE Trans. Commun.*, pp. 737-745, 1996.

[12] J. Hagenauer. Iterative decoding of binary block and convolutional codes. *IEEE Trans. Commun., vol. 42*, pp. 429-445, 1996.

[13] J. Hagenauer. Source-controlled channel decoding. *IEEE Trans. Commun., vol. 43*, pp. 2449-2457, 1995.

[14] K. M. S. Soyjaudah and M. T. Russool. Comparative study of turbo codes in awgn channel using map and sova decoding. *Department of Electrical and Electronic Engineering, University of Mauritius.*

[15] S. Lin J. Chen, M. P. Fossorier and C. Xu. Bi-directional sova decoding for turbo-codes. *IEEE Commun. Lett., vol. 4, no. 12*, pp. 405-408, 2000.

[16] E Villebrun L Papke, P Robertson. Improved decoding with the sova in a parallel concatenated (turbo-code) scheme. *Proc. IEEE ICC 96,Vol. 1*, pp. 102-106, 1996.