Efficient Error Correcting Codes for Wireless Networks: TURBO CODES

Developed by

Syed Khaldoon Khurshid
& Muhammad Imran Herl

Supervised by

Dr. Muhammad Sher

Department of Computer Science
International Islamic University, Islamabad
(2004)

بِسمِ اللهِ الرَّحمٰنِ الرَّحِيم

In the name of Almighty ALLAH,
The most Beneficent, the most
Merciful.

International Islamic University, Islamabad

Dated: July 02 2004

Final Approval

It is certified that we have read the project report, titled "***Efficient Error Correcting Codec for Wireless Networks: TURBO CODES***" submitted by Khaldoon and Imran. It is our judgment that this project is of sufficient standard to warrant its acceptance by the International Islamic University, Islamabad, for the Degree, **Master of Science**.

Committee

External Examiner
Dr. Abdul Sattar
Consultant, Faculty of Science
Allama Iqbal Open University, Islamabad.

Internal Examiner
Dr. S. Tauseef-ur-Rahman
Head
Department of Telecommunication &
Department of Computer Engineering
FAS, International Islamic University
Islamabad.

Supervisor
Dr. Muhammad Sher
Asstt. Professor,
Department of Computer Sciences
FAS, International Islamic University
Islamabad.

30/06/04

Dedication

Dedicated to our loving parents, siblings and teachers. We offer our humblest thanks to them, who made this project possible by their love and guidance and obviously ALLAH, who has helped us, whenever we needed him the most.

A dissertation submitted to the

Department of Computer Science,

International Islamic University, Islamabad

as a partial fulfillment of the requirements

for the award of the degree

Master of Science

Declaration

Some of the research in this thesis has been published in publications, conference proceedings and journals. These publications are listed at the end of the thesis.

All work presented in this thesis as being original is so, to the best knowledge of the authors. References and acknowledgements to other researchers have been given as appropriate.

Syed Khaldoon Khurshid
03-CS/MS/01
Muhammad Imran Herl
41-CS/MS/01

Acknowledgements

First of all we pay our humble thanks to **Almighty Allah** who knows all the things hidden or evident in this universe, even the things which pass through our hearts, who created us and gave us the courage to complete this project. It is said in the Holly Quran:

"Does man think that he will be left uncontrolled, (without purpose)? Was he not once a drop of ejected semen? Then he became a clot, so He created and fashioned him and made him into two sexes, male and female. Is He who does this not able to bring the dead to life?"

[Surah alQiyama: 36-40]

Without His help and blessings, we were unable to complete the project.

We also offer our humblest thanks to **The Holy Prophet Muhammad** (Allah's grace and peace be upon him), who made us aware of our creator and guided us to the track, which leads to the success, who is a symbol of love and affection for all the creatures of Allah.

We feel highly privileged in taking opportunity to express our profound gratitude and sense of devotion to our supervisor **Dr. Muhammad Sher**, and **Dr. Muhammad S. Tauseef ur-Rahman**, International Islamic University, Islamabad for their inspiring guidance, consistent encouragement, sympathetic attitude and dynamic supervision.

We also express our cordial and humble thanks to all our friends and teachers for their untiring help and cooperation in completing this project.

We cordially regard the inspiration, prays, encouragement and financial support of our loving and affectionate **Parents and Siblings** for their motivation in every aspect of our studies enabling us to complete this project.

Khaldoon & Imran

Project in Brief

Project Title:	*Efficient Error Correcting Codes for Wireless Networks: TURBO CODES*
Objective:	**To completely grasp the concept of Turbo Codes, i.e. Encoder and Decoder working, Understanding Decoding Algorithm (SOVA). Enhancing our understanding by developing simulation and discussing interleaver design complexity issue.**
Undertaken By:	**Khaldoon & Imran**
Supervised By:	**Dr. Muhammad Sher** DCS, International Islamic University, Islamabad
Technologies Used:	**Mat Lab 6.1**
System Used:	**Pentium® III**
Operating Systems Used:	**Microsoft® Windows® 2000 Professional**
Date Started:	**June, 2003**
Date Completed	**June, 2004**

Abstract

The correct transmission and reception of data has been a main concern for researchers since the advent of the communication era. In the presence of noise generating factors like frequency interference, weather conditions; in the open media, it is impossible to remove errors in the wireless media, where the vulnerability of data is higher than in any other media. In a noisy environment, it is often not possible to reduce the bit error rate to acceptable levels. Doing so may require raising the signal power beyond practical limits. Alternately lowering the error rate might require communicating at an unacceptably slow rate. There is another available option to improve the performance of digital communication systems such as 'error control coding' can be used to provide a structure for error tolerant communication. The structure is such that errors can be recognized at the receiver. The error control is accomplished in two steps: detection of error and its correction. Error detection is the process of providing enough structure so that receiver knows when the error occurs. If the added structure is sufficiently detailed to allow pinpoint the location of these errors, the code is an error correcting code, and it is possible to correct errors at the receiver without requesting additional information from the transmitter (e.g. a retransmit request).This process is known as Forward Error Correction. Forward error correction normally requires adding the redundancy to the signal; more bits are sent than required. There is no possibility of the error free communication over the noisy channel when messages are encoded with zero redundancy. Various coding techniques are developed to detect and correct the errors at the receiver/destination. These techniques are called error correcting codes and the most efficient one is 'Turbo Codes'. Considering the efficiency, Parallel Concatenated Convolutional Codes (the other name for Turbo Codes) perform remarkably better as compared to any version of other coding techniques.

Claude Shannon has shown that it is possible to achieve error free communication by adding sufficient redundancy. Shannon limit was not achieved by the convolutional codes but Turbo Codes have achieved the signal to noise ratio near (E_b / N_o = -1.6 db). Amongst the encoders for turbo codes, Recursive Systematic Convolutional (RSC) Encoder is exclusively better than the Non-recursive and Non-systematic ones. While the decoding algorithm for turbo codes, SOVA (Soft Output Viterbi Algorithm) outperforms its counterparts like MAP in terms of complexity and efficiency. This thesis is unique having complete understanding of the SOVA decoding algorithm, step by step.

The concept of interleaver has expanded in Turbo codes, which not only remove burst errors but also help in code weight distribution. It feeds the encoders with permutations so that the generated redundancy sequence can be assumed independent. All the functions of the interleaver have a significant effect on the performance of the turbo code, thus making the design of the interleaver a critical issue.

This thesis puts different issues concerning Turbo Codes at one place describing the working of decoding algorithm SOVA and, design and complexity features of interleaver which is a vital part in turbo codes construction. However this research focuses on the existing literature and techniques rather than on generating and testing new theories.

Key Words: Error Correcting Codes, Turbo Codes, Recursive Systematic Convolutional (RSC) Encoder, SOVA (Soft Output Viterbi Algorithm), Interleaver Design

TABLE OF CONTENTS

Chapter No. *Contents* *Page No.*

1. INTRODUCTION

2. LINEAR BLOCK AND CONVOLUTIONAL CODES

3. TURBO CODES.

1. Introduction

Error free communication systems are needed to guard against loss or damages of data and control information. In all communication systems, there is a potential for error. Transmitted signals are distorted to some extent before reaching their destination. Error detection and correction are required in circumstances where error cannot be tolerated This is usually the case with data processing systems. Typically, error control is implemented as two separate functions. error detection and retransmission. To achieve error detection, the sender inserts an error-detection code in the transmitted PDU (Protocol Data Unit), which is a function of the other bits in the PDU. The receiver checks the value of the code in the incoming PDU. If an error is detected, the receiver discards the PDU. Upon failing to receive an acknowledgment from the reciever in a reasonable time, the sender retransmits the PDU. Some protocols also employ an error correction code which enables the receiver not only to detect errors but, in some cases to correct them as well. We will talk about these protocols in the thesis.

1.1 Error Induction

When data is being transmitted between two DTEs, it is very common (especially if the transmission lines are in an electrically noisy environment such as telephone network) for the electrical signals representing the transmitted bit stream to be changed by electromagnetic interference induced in the lines by neighboring electrical devices This means that signals representing a binary 1 may be interpreted by the receiver as a binary 0 and vice versa. To ensure that information received by a destination DTE has a high probability of being the same as that transmitted by the sending DTE, there must be some way for the induction to a high probability when received information contains errors Furthermore, should errors be detected, a mechanism is needed to obtain error free data. Error free communication systems are needed to provide secure transmission of data over the network. The growing traffic over the networks, band width limitations and nature of data make it essential to have error free communication systems. Growing use of satellites in communication systems make it vital to have some reliable error free communication mechanism.

1.2 Error Control

There are various approaches to achieve forward error control in which each transmitted character or frame contains additional (redundant) information so that the receiver can not only detect when errors are present but also determine where in the received stream the errors are. The correct data is then obtained by inverting these bits.Feedback (backward) error control in which each character or frame includes only sufficient additional information to enable the receiver to detect when error are present and not their location. A retransmission control scheme is then used to request a retransmit.

In practice the number of additional bits required to achieve reliable forward error control increases rapidly as the number of information bits increases, hence feedback error control is the predominant method used in the types of data communication and networking systems. Feedback error control can be divided into two parts: firstly the techniques that are used to achieve reliable error detection, secondly the control algorithms that are available to perform the associated retransmission control schemes This section is concerned with the most common error detection techniques currently in use. The two factors that determine the type of error detection scheme used are the bit error rate (BER) of the line or circuit and the type of errors (that is whether the error occurs as random single-bit errors or as burst errors). The BER shows the probability *P* of a single bit being corrupted in a defined time interval. For example if 125 characters per block each of eight bits per frame is to be sent and the probability of a block (frame) containing an error is approximately 1. This means that on average every block will contain an error and must be retransmitted. Clearly this length of frame is too long for this type of line and must be reduced to obtain an acceptable throughput. The type of errors present is important since as we will see different types of error detection schemes are used to detect different types of error. Also the number of bits used in some schemes determine the burst lengths that are detected. The three most widely used schemes are parity bit check, block sum check and cyclic redundancy check.

1.3 Probability of Errors

The probability that a frame arrives with no bit errors decreases when the probability of a single bit error increases, as you would expect. Also, the probability that a frame arrives with no bit errors decreases with increasing frame length. Longer the frame, the more bits it has and higher the probability that one of these is in error.

1.4 Error Detection

Following error detection techniques are commonly used. Each technique has its own mechanism to work and its drawbacks which are discussed in brief in the following.

1.4.1 Parity Bit Check

The most common method used for detecting bit errors with asynchronous and character-oriented synchronous transmission is the parity bit method. With this scheme the transmitter adds an additional bit - the parity bit to each transmitted character prior to transmission. The parity bit used is a function of the bits that make up the character being transmitted. Hence, on receipt of each character, the receiver can perform a similar function on the received character and compare the result with the received parity bit. If they are equal, no error is assumed, but if they are different, then a transmission error is assumed to have occurred. To compute the parity bit for a character, number of 1 bits in the code for the character are added together (modulo-2) and the parity bit is then chosen so that the total number of 1 bit (including the parity bit itself) is either even- even parity - or odd - odd parity.

Method will only detect single (or an odd number of) bit error and that two (or an even number of) bit errors will go undetected. The circuitry used lo compute the parity bit for each character comprises a set of exclusive-OR (XOR) gates connected. The XOR gate is also known as a modulo-2 adder since the output of the Exclusive-OR operation between two binary digits is the same as the addition of the two digits without a carry bit. The least significant pair of bits is first XORed together and the output of this gate is then XORed with the next (more significant) bit, and so on. The output of the final gate is the required parity bit which is loaded into the transmit register prior to transmission of the character. Similarly, on receipt, the recomputed parity bit is compared with the received parity bit. If it is different, this indicates that a transmission error has been detected.

1.4.2 Block sum check

When blocks of characters are being transmitted, there is an increased probability that a character (and hence the block) will contain a bit error hence an extension to the error detection capabilities obtained by the use of a single parity bit per character (byte) can be achieved by using an additional set of parity bits computed from the complete block of characters (bytes) in the frame. With this method each character (byte) in the frame is assigned a parity bit (as before column or row parity). In addition an extra bit is computed for each bit position (longitudinal or column parity in the complete frame. the resulting set of parity bits for each column is referred to as the block (sum) check character. It uses (odd parity) for the row parity bits and even parity for the column parity bits. It assumes that the frame contains printable characters

It can be deduced that although two bit errors in a character will escape the row parity check they will be detected by the corresponding column parity check. This is true of course only if no two bit errors occurring will be much less than the probability of two bit in a single character occurring hence use of a block sum check significantly improves the error detection.

1.4.3 Cyclic Redundancy Check (CRC)

One of the most common, and most powerful, error-detecting codes is the cyclic redundancy check (CRC), which can be described as follows. Given a k-bit block, or message, the transmitter generates an n-bit sequence, known as a frame check sequence

(FCS), so that the resulting frame, consisting of k + n bits, is exactly divisible by some predetermined number. The receiver then divides the incoming frame by that number and if there is no remainder, assumes there was no error.

1.5 Drawbacks of the techniques

The use of parity bit check is not foolproof, as noise impulses are often long enough to destroy more than one bit, particularly at high data rate. As parity check only encounters odd number of bit errors and even number of bit patterns are overhead. Block sum fails when there is a burst error in the column and rows as well. So the final parity bit cannot verify the burst errors in block sum.

1.6 Introduction to Error Control Coding

Over the years, there has been a tremendous growth in digital communication especially in the fields of cellular/PCS, satellites, and computer communication. In these communication systems, the information is represented as a sequence of binary bits. The binary bits are then mapped (modulated) to analog signal waveforms and transmitted over a communication channel. The communication channel introduces noise and interference to corrupt the transmitted signal. At the receiver, the channel corrupted transmitted signal is mapped back to binary bits. The received binary information is an estimate of the transmitted binary information. Bit errors may result due to the communication channel and the number of bit errors depends on the amount of noise and interference in the communication channel.Channel coding is often used in digital communication systems to protect the digital information from noise and interference and reduce the number of bit errors. Channel coding is mostly accomplished by selectively introducing redundant bits into the transmitted information stream. These additional bits will allow detection and correction of bit errors in the received data stream and provide more reliable information transmission. The cost of using channel coding to protect the information is a reduction in data rate or an expansion in bandwidth. In 1948, Claude Shannon showed that controlled redundancy in digital communications allows for the existence of communications at arbitrarily low bit error rate (BER).

1.6.1 Shannon Theory

The field of Information Theory, of which Error Control Coding is a part, is founded upon a paper by Claude Shannon. Shannon calculated a theoretical maximum rate at which data could be transmitted over a channel perturbed by additive white Gaussian noise (AWGN) with an arbitrarily low bit error rate. This maximum data rate, the channel capacity C, was shown to be a function of the average received signal power, S, the average noise power, N, and the bandwidth of the system, W. This function, known as the Shannon-Hartley Capacity Theorem, can be stated as:

$$C = W * \log_2(1+S/N) \qquad (1.5)$$

If W is in Hz, then the channel capacity C is in bits/s. Shannon stated that it is theoretically possible to transmit data with a data *rate R* . C with an arbitrarily small error probability by use of a sufficiently complicated coding scheme.

Error control coding (ECC) uses this controlled redundancy to detect and correct errors. When ECC is used to correct errors this is called Forward Error Correction (FEC). The method of error control coding used, depends on the requirements of the system (e.g. data, voice, and video) and the nature of the channel (e.g. wireless, mobile, high interference). The obstacle in error control coding research is to find a way to add redundancy to the channel so that the receiver can fully utilize that redundancy to detect and correct the errors and to improve the coding gain -- the effective lowering of the power required or improvement in through put. Codes have varying degrees of efficiency and only a few limited cases actually make use of all of the redundancy in the channel. Different codes also work better in certain conditions like high bit error rates (BER), high throughput, or bursty channel. Shannon's discoveed the theoretical limit on the lowest signal to noise ratio required to achieve throughput.He also proved that channels have a maximum capacity C, that can not be surpassed regardless of how good the code is. However, he proved that codes exist such that by keeping the code rate, R, (explained below) less than C, one could control the error rate based simply on the code structure without having to lower the effective information throughput. Four decades of research have lead to codes that come very close to matching this theoretical limit.

Since adding redundancy to the channel effectively reduces the bandwidth of the channel, there is a strong impetus to come as close to the theoretical limit. The code rate R is defined as follows:

k = (length of message block)

n = (length of the transmission block with redundancy added)

R = k / n {because n >= k and k > 0, 0 < R <= 1}

Redundancy in digital communications (what this paper deals with) takes the form of parity bits on the information bits. The amount of redundancy is equal to the number of bits transmitted minus the original length of the message or:

n - k = parity check bits

Note that a code rate of one -- the maximum and n = k -- means that there is no redundancy in the channel. A code rate of 1/2 means that for every 1 bit of information their are 2 bits being transmitted. A low code rate (1/4 to 1/2) is used for channels that have many errors and require that the encoder includes considerable redundancy. A high code rate (1/2 to 1) is used for channels that have few errors and don't require that the encoder include much redundancy. Multiplying the bandwidth or capacity of the channel with the code rate gives the effective bandwidth after ECC:

*(Effective Bandwidth) = (Bandwidth) * R*

Error control coding bases its mathematical foundation on linear algebra. All codes discussed are linear codes. The linear codes are divided into two categories: trellis codes and block codes. Figure 1.1 shows the lineage of the best known codes.

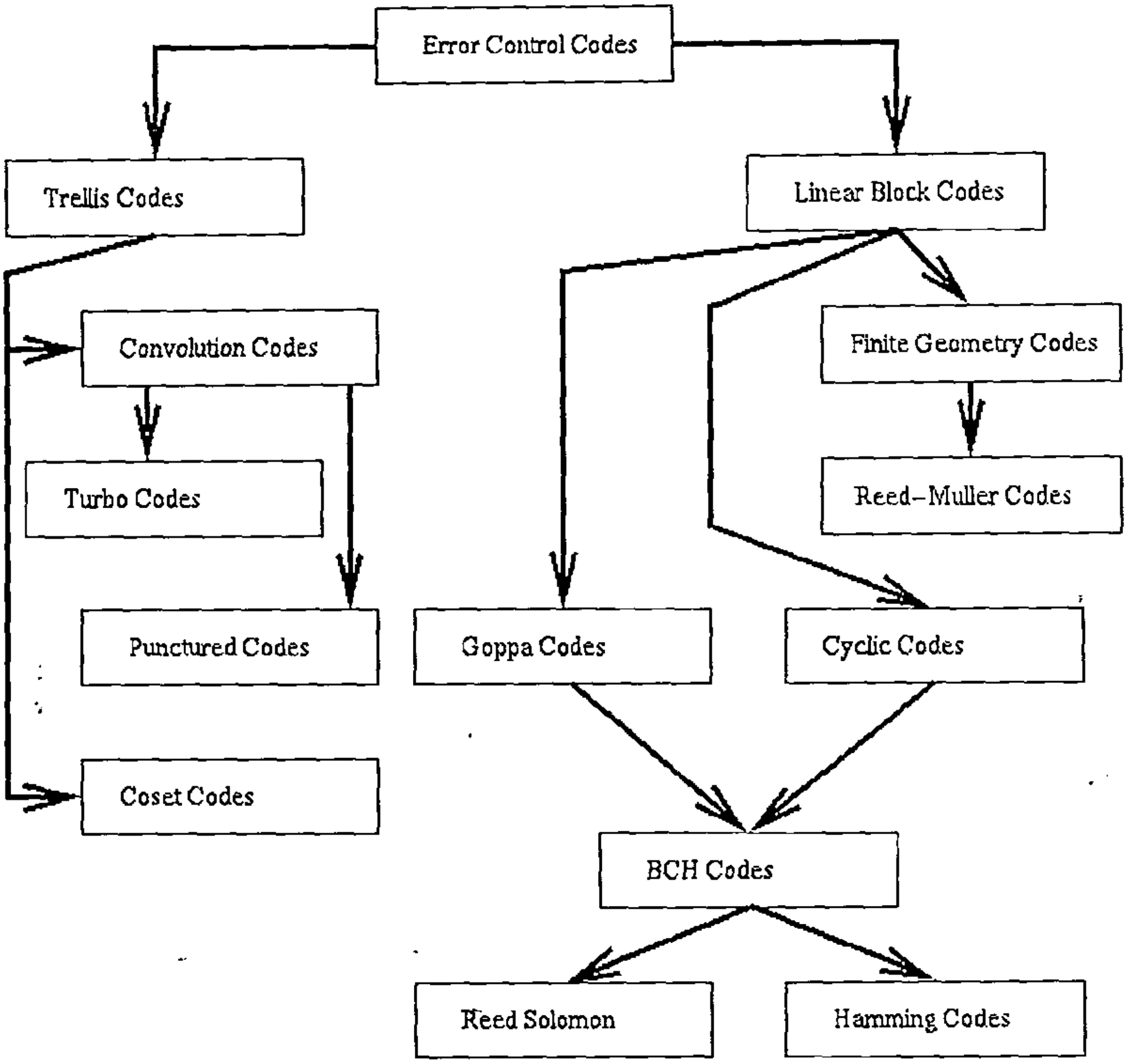


Figure 1.1: Code Lineage

Each code is distinguished by the method used to add redundancy and how much of this redundancy is added to the information going out of the transmitter. Below are several terms that are used to differentiate and understand different codes.

- **Weight** The *weight* of a tuple of bits is equal to the number of ones in the tuple. The weight of (0 0 1 1 0 1 0) is three. The weight of (1 1 1 1 1 0) is five. The weight of the *zero codeword* or all zero codeword (0 0 0 0 0 0) is zero.
- **Minimum Distance** The *minimum distance* d_{min} of a code is the minimum number of bits that must change in any codeword to produce another codeword.

The greater the minimum distance, the more effective the code can be. The *hamming distance* (also known just as the *distance*) between two tuples is defined to be the number of bits that are different between two tuples. For example, the *hamming distance* between the two tuples, (0 0 0 0) and (1 1 1 0), is equal to three, because there are three bits those are different. The hamming distance between any two tuples v and w is denoted as d (v, w).

1.6.2 How the Redundancy is used to Correct and Detect Errors

Error control coding takes a packet or sequence of data. Typically this takes the form of a tuple of bits. This tuple represents the message. Redundancy is added to each message in a systematic way so that the resultant codewords have a one to one relation (match) to the messages. This means that the codewords are larger than the messages (because of the redundancy), but for every valid codeword there is one and only one message. For example:

You take an 8 bit message and add 1 bit of redundancy in the form of an even parity bit:

(0 1 1 0 1 0 0 0) becomes (0 1 1 0 1 0 0 0 1)

This implies that the tuple (0 1 1 0 1 0 0 0 0) is *not* a valid codeword since it does not satisfy even parity although it is 9 bits long just like the other codeword. The resultant codeword bit-tuple supports 2^9 possible bit patterns, but only 2^8 of those patterns are valid.

By making the possible number of bit patterns in the codeword much larger than the number of possible messages, the number of invalid codeword bit patterns becomes larger. If bit patterns are chosen within the space of possible codewords that maximize the hamming distance between the valid codewords, an effective code can be created. A message is encoded into its matching codeword. This codeword has a certain amount of *distance* from all the other valid codewords. Even if a few errors occur, as long as the number of errors is less than the distance to any other code, the resulting codeword with errors will always be invalid. At this point, the receiver has to decide to fix the error or signal that is has found an error and drop the packet. In the latter case, the receiver is done. In the former case, there is a difficult question to be answered. If a valid codeword has had errors added to it, the receiver must choose the closest valid codeword to map the invalid codeword to. One very important assumption that is made in error control coding is that when the decoder has a choice between two codewords, it will always choose the codeword that is the closest to the received codeword according to hamming distance {minimize d(possible valid codeword, received erroneous codeword)} -- the least number of errors is predicted to be the most likely. Note that if a codeword receives enough errors in the right places, it could be picked up at the receiver and "look" like a valid codeword, or be closer to a another valid codeword than the one that was originally transmited. In either case, errors get through undetected. In the latter case, the receiver, in the process of "correcting" errors, effectively adds errors to the data. For instance, if the repetitive code (3,1) is used where:

n = 3 (three bits transmitted in each code word)

k = 1 (one bit of information is transmitted)

If the information bit k is 0, then the codeword n is (0 0 0). If the information bit k is 1, then the codeword n is (1 1 1).If no errors occur, then the only two codewords that can be received at the receiver (figure 1.2 below) are (0 0 0) and (1 1 1). The receiver can then decode these into 0 and 1 respectively. However, if errors were introduced by the channel then we can theoretically receive anything. This code can correct one error. Because the receiver does not know what the transmitted codeword is and the receiver also knows that errors can be introduced by the channel, it will look for the closest match between the received codeword and one of the two valid codewords (0 0 0) and (1 1 1).

Received codeword r	The most probable codeword v transmitted	Hamming distance to (0 0 0)	Hamming distance to (1 1 1)
(0 0 0)	(0 0 0)	0	3
(0 0 1)	(0 0 0)	1	2
(0 1 0)	(0 0 0)	1	2
(1 0 0)	(0 0 0)	1	2
(1 1 0)	(1 1 1)	2	1
(0 1 1)	(1 1 1)	2	1
(1 0 1)	(1 1 1)	2	1
(1 1 1)	(1 1 1)	3	0

Table 1.1: The decoding table for a (3, 1) repetitive code

Note that if the repetitive code (2,1) had been chosen the errors could not have been corrected because the hamming distance between the two possible code words would be equal. This code can detect one error.

Received codeword r	The most probable codeword v transmitted	Hamming distance to (0 0)	Hamming distance to (1 1)
(0 0)	(0 0)	0	2
(0 1)	Error Detected*	1	1
(1 0)	Error Detected*	1	1
(1 1)	(1 1)	2	0

Table 1.2: The decoding table for a (2, 1) repetitive code

* Error was detected, but could not be correct because the minimum distances to the closest code words were equal.

1.6.3 Stages in Error Control Coding

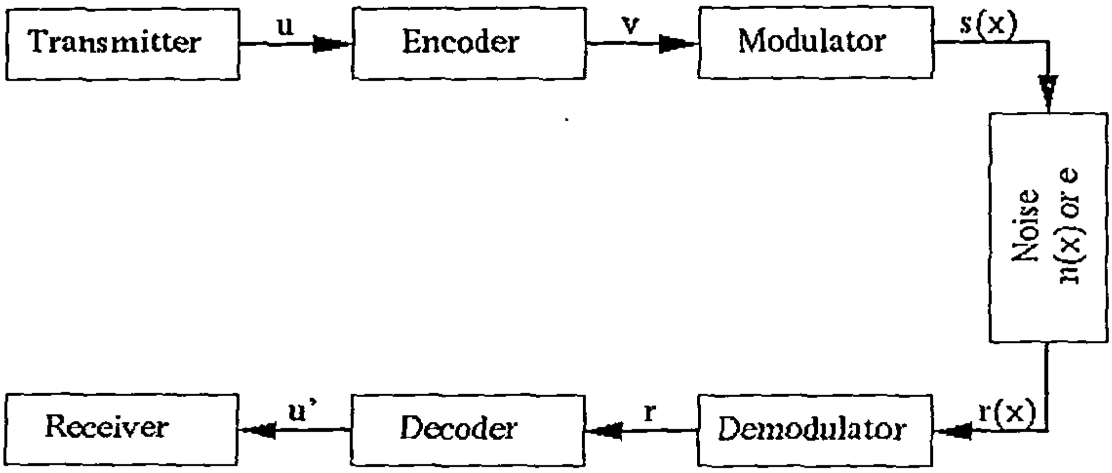


Figure 1.2: Error control coding path

- **u** = message: The binary message can be expressed as a tuple (1 0 0 1 1 0 0 0). The bit on the left (denoted u_0) is *the first to enter the encoder*. The bits are entered from left to right. The last bit to enter is u_n where n = 7 in this case.

- v = codeword: Code words can be expressed in the same way as information bits. Code words include the information bits (maybe in a slightly altered form) plus the redundancy in the form of parity checks. v will always be greater in length than u if the code rate is below 1. Using the above tuple and a one bit even parity, v would equal (1 0 0 1 1 0 0 0 1).

- s(x) = transmitted signal: The transmitted signal is the modulated encoded information.

- n(x) = noise signal: The noise signal , added to the transmitted signal s(x).

- e = error pattern: This is the counterpart of the noise signal in the binary tuple domain. Error patterns can be expressed as tuples or polynomials as well. For example, the transmitted codeword v has the error pattern e added to it. The receiver would pick up v + e with an error in the every position in e that has a 1. In the graph above, we have assumed that the error channel is equivalent to an additive white Gaussian noise channel. There are many types of noise that can be present in a communications channel, but all channels are subject to AWGN.

- **r(x)** = received signal: **r(x) = s(x) + n(x)**

- **r** = received sequence: The received sequence is equal to v + e as shown above. It is the demodulated received signal r. The received sequence r is equal to the transmitted codeword v if there are no errors. If r does not equal v, then e must not contain all zeros and therefore an error has occurred. For example, if v = (1 0 0 1 1 0 0 0 1) and e = (0 0 0 0 1 0 0 0 0) then r = (1 0 0 1 0 0 0 0 1) with an error in the fifth bit.

- **u'** = decoded received sequence: If there were no errors or whatever errors that did occur were correctable u' will equal u. However, if there was an undetectable error or decoder error, u' will not equal u.

1.6.4 Effectiveness of Codes

Error-correcting codes operate in general by introducing redundancy to combat errors introduced by the noise in the channel. Thus by adding redundancy we have reduced the rate of information transfer from 1 bit per channel use, to a fraction R = (k/n) bits per channel use, where k is number of information bits (before adding redundancy), and n is number of bits after adding redundancy. The ratio R is called the code rate. In order to introduce the concept of Code Performance, we need to introduce two definitions. Bit Error Rate (BER) is the probability of any particular bit being in error within a transmission.The signal to noise ratio (SNR, expressed as Eb/No) is the ratio of the channel power to the noise power. Bit Error Rate (BER) and Signal to Noise Ratio

(SNR) of the transmission determine channel performance. As shown below in Figure 1.3, low bit error rates are attainable with all channels, coded or uncoded. The difference, however, is how much power (SNR) is necessary to achieve a low BER.

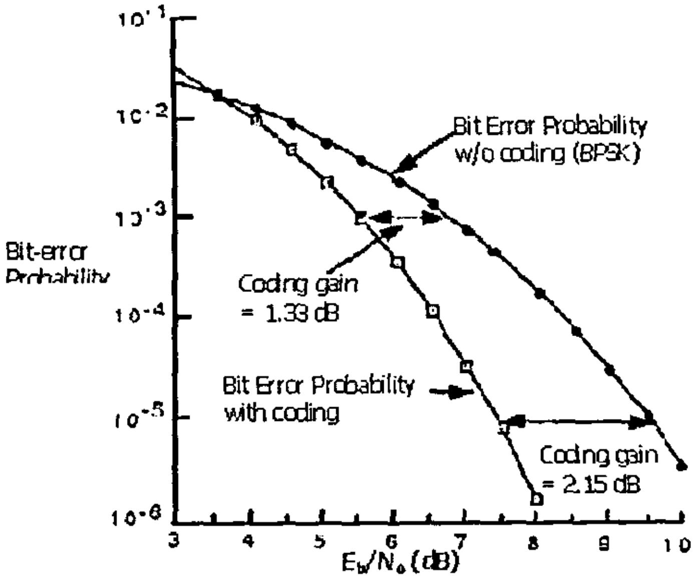


Figure 1.3: Error Code Efficiency: BER versus E_b/N_0

For a fixed BER, the required Eb/No is reduced with error-control coding by coding gain. The plot in Figure 1.3 shows the coding gains available with the Golay rate (12/23) code having block length 23. At a bit-error probability equal to 10-6, the available coding gain using hard decisions is 2.15dB. . The difference between the two signals to noise ratios (SNR) at a particular probability of output error is equal to the coding gain. As long as the coded channel requires less SNR then the code is effective. Shannon's Law states that no matter what code is used nothing can be gained below -1.6 [dB]. Since there are an infinite number of codes that will produce a positive coding gain, the objective is to choose the most effective code. This would be a line along Shannon's limit at -1.6 [dB]. While no known codes achieve this limit, recent codes (e.g. turbo codes) are starting to get close to the Shannon limit. Another concern is, not to over-encode or under-encode your channel. *Over-encoding* occurs when the channel does not have enough errors to justify a code rate as low as the one being used. *Under-encoding* occurs when the channel has too many errors and the code rate is too high. In static situations, the types and probability of errors will be known. However, in dynamic situations like wireless mobile applications the quality of the channel (number of errors) will be changing versus time and different codes with different code rates should be used. The repetitive codes shown above are very inefficient and would probably never be used. The method of how redundancy in the form of parity checks is added can get very complicated and efficiency of the code must be proven mathematically, because the brute-force method of trying every codeword could take centuries or longer. Once a code has been proven effective, a method of application must also be found that is within the space, time, and monetary requirements of the system.

The goal of channel coding is to reduce the number of errors caused by transmission in a, power-limited environment. Theoretically, the best possible performance any channel can accomplish is called the Shannon limit A code with Shannon limit performance is ideal, but so far, has not been achieved in practice. The only practical code that comes close to the Shannon limit is the RSC Turbo code.

1.6.5 Types of Channel Codes

There are two main types of channel codes, namely block codes and convolutional codes. There are many differences between block codes and convolutional codes. Block codes 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. By predetermined rules, n-k redundant bits are added to the k information bits to form the n coded bits. Commonly, these codes are referred to as (n,k) block codes. Some of the commonly used block codes are Hamming codes, Golay codes, BCH codes, and Reed Solomon codes (uses non-binary symbols).There are many ways to decode block codes and estimate the k information bits. These decoding techniques will not be discussed here but can be studied in courses on Coding Theory [Wic95].

Convolutional codes are one of the most widely used channel codes in practical communication systems. These codes are developed with a strong mathematical structure and are primarily used for real time error correction. In convolutional codes the encoded bits depend not only on the current k input bits but also on past input bits. The main decoding strategy for convolutional codes is based on the widely used viterbi algorithm. As a result of the wide acceptance of convolutional codes, there have been many advances to extend and improve this basic coding scheme. This advancement resulted in two new coding schemes, namely, trellis coded modulation (TCM) and turbo codes. TCM adds redundancy by combining coding and modulation into a single operation (as the name implies). The unique advantage of TCM is that there is no reduction in data rate or expansion in bandwidth as required by most of the other coding schemes. A near channel capacity error correcting code called **turbo code** was also introduced. This error correcting code is able to transmit information across the channel with arbitrary low (approaching zero) bit error rate [Pro95]. This code is a parallel concatenation of two component convolutional codes separated by a random interleaver. It has been shown that a turbo code can achieve performance within 1 dB of channel capacity [BER93]. Random coding of long block lengths may also perform close to channel capacity, but this code is very hard to decode due to the lack of code structure. Without a doubt, the performance of a turbo code is partly due to the random 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.

There are two primary decoding strategies for turbo codes. They are based on a maximum a posteriori (MAP) algorithm and a soft output Viterbi algorithm (SOVA). Regardless of which algorithm is implemented, the turbo code decoder requires the use of two (same algorithm) component decoders that operate in an iterative manner. In this thesis, the SOVA will be examined, because it is much less complex than MAP and it provides comparable performance results. Furthermore, SOVA is an extension of the Viterbi algorithm, and thus has an implementation advantage over MAP.

1.7 Wireless Communication

Guglielmo Marconi invented the wireless telegraph in 1896. In 1901, he sent telegraphic signals acrooss the Atlanic Ocean from Cornwall to St. John's Newfoundland; adistance of 1800 miles. His invention allowed two parties to communicate by sending each other alphanumeric characters encoded in a analog signal. Over the last century, advances in the wireless technology has led to the radio,the television, the mobile telephone, and the communication satellites. All types of the inforamtion can now be sent to any corner of the world. Recently, a great deal of attention has been focused on satellite communications, wireless networking and cellular technology.

Communication satellites were first launched in 1960s. Those first satellites could handle 240 voice circuits. Today, satellites carry about one third of the voice traffic and all of the television signals between the countries[EVAN98].Modern satdlites typically introduce a quarter-second propagation delay to the signals they handle. Newer satellites in the lower orbits, with less inherent signal delay, are deployed to provide a data services such as internet access.

Wireless networking is allowing businesses to develop WANs, MANs, and LANs without a cable plant. The IEEE has developed 802.11 as a standard for wireless LANs. The Bluetooth industry consortium is also working to provide a seamless wireless networking technology.

The celluar or mobile telephone is the modern equivalent of Marconi's wireless telegraph, offering two-party, two way communication. The first generation wireless phones used analog technology. These devices were heavy and coverage was patchy, but they successfully demonstrated the inherent convenience of mobile communications. The current generation of wireless devices are built using digital technology istead of analog. Digital networks can carry much more traffic and provide better reception and security than analog networks. In addition digital technology has made possible value-added services such as caller identification. Now a days devices are connecting with internet using new frequency ranges at higher information rates. The impact of the wireless communication has been and will continue to be profound. Very few invention have been able to "Shrink" the world in such a manner. The standards that define how wireless communication interact are quickly converging and will soon allow the creation of global wireless network that will deliver a wide variety of services.

As higher and higher speeds are used in wireless applications, error correction continues to pose major design challenge. Recently, a new class of codes, called turbo codes, has emerged as a popular choice for third-generation wirless systems.The figure 1.4 provides an overview of the processing of speech signals for transmission over a logical traffic channel. This example figure has similarity to genernal figure 1.2 which illustrates the similarity between the functionality.

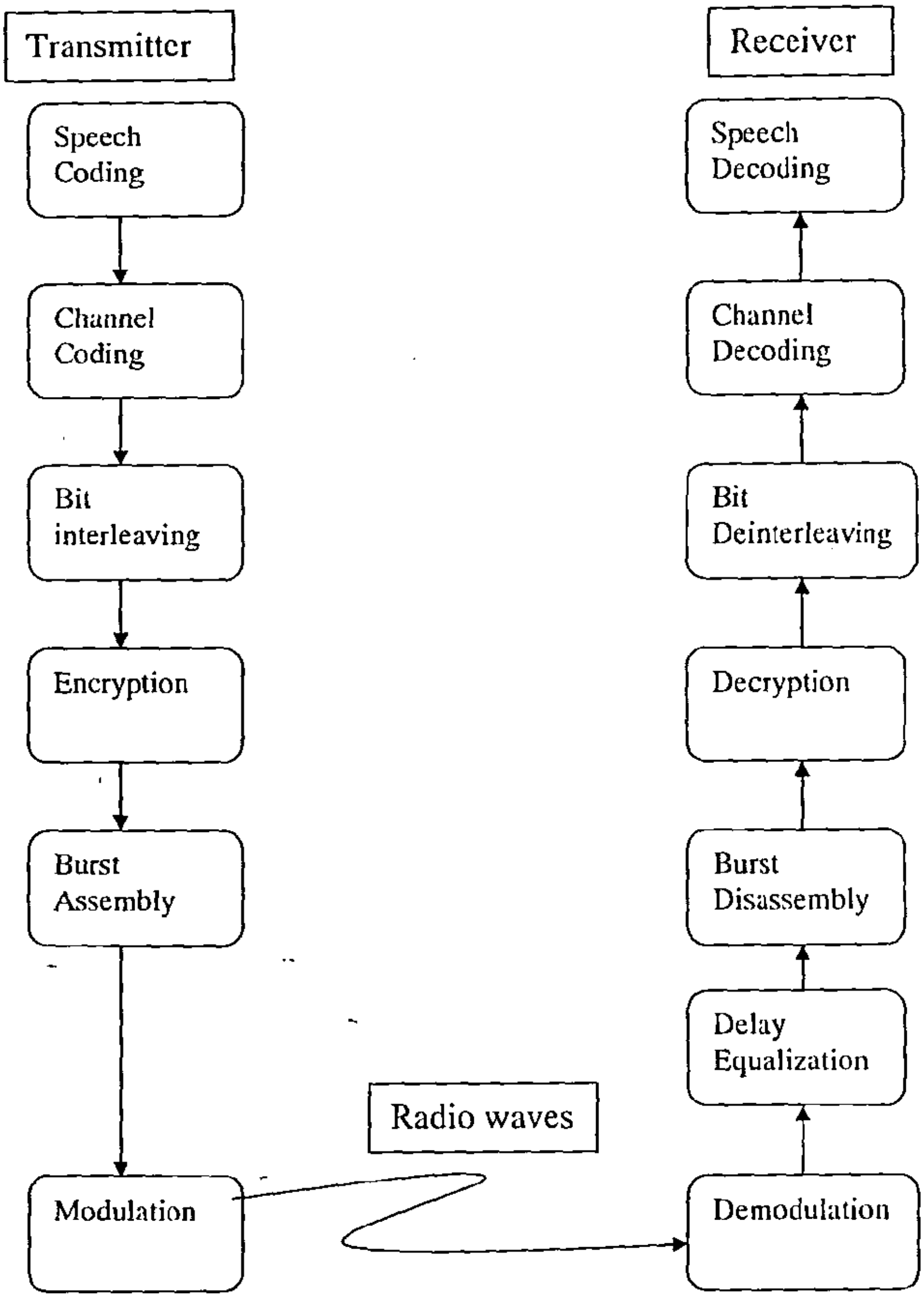


Figure1.4 GSM Speech Signal Processing

1.8 Outline of Thesis

In this thesis, a special class of convolutional codes, called turbo codes, is presented. Since turbo codes are an extension of convolutional codes, the basic concepts of convolutional codes are required. Chapter 2 presents the fundamentals of linear block & convolutional codes. Linear block code are explained in detail in this chapter. This chapter discusses the encoder structure and its many representations. Also, it discusses the primary decoding algorithm for convolutional codes, namely the viterbi algorithm. Both hard and soft-decision viterbi algorithms are presented in Chapter 2. Furthermore, performance issues related to convolutional codes are discussed. Chapter 3 introduces the basic turbo codes encoder. The turbo codes encoder is a parallel concatenation of two recursive systematic convolutional (RSC) encoders, separated by an interleaver. This chapter shows the construction of a RSC encoder from a non-recursive nonsystematic (conventional) convolutional encoder. The chapter explains the working of the SOVA algorithm in detail. Furthermore, this chapter describes many different types of interleaver that are suitable for the turbo code encoder. Chapter 4 describes the analysis, design and implementation issues of the simulation. Chapter 5 investigates the performance of turbo codes through extensive computer simulation testing. Many simulation results are then presented to show the important characteristics of turbo codes. Furthermore, this chapter summarizes the important findings about turbo codes, and concludes the thesis.

2. Linear Block and Convolutional Codes

2.1 History of Error Control Coding

Year	Who	Event or Discovery
1948	Shannon	Shannon's Limit
1949	Golay	Golay Codes
1950	Hamming	Hamming codes (first linear error correcting codes)
1954	Reed	First majority logic decoder
1954	Muller	Reed-Muller codes
1955	Elias	Convolutional codes
1957	Prange	Cyclic codes
1959	Hocquenghem	BCH codes
1959	Fire	Fire codes
1959	Hagelbarger	Convolutional burst error correcting codes
1960	Bose	BCH codes independently discovered
1960	Chaudhuri	BCH codes independently discovered
1960	Peterson	Cyclic structure of BCH proven
1960	Peterson	Simple decoding method of binary BCH codes
1960	Reed & Solomon	Reed-Solomon Codes
1961	Wozencraft	Sequential decoding for convolutional codes
1961	Meggitt	Meggitt decoder
1961	Kasami	Error trapping decoder with Kasami algorithm
1961	Mitchell	Error trapping decoder discovered independently
1962	Rudolph	Error trapping decoder discovered independently
1963	Massey	Threshold decoding for convolutional codes
1964	Chien	Chien searching algorithm for BCH codes
1965	Berlekamp	Berlekamp's iterative algorithm for BCH codes
1967	Viterbi	Viterbi (Maximum Likelihood) decoding for convolutional codes
1967	Rudolph	Finite geometry codes
1979	Cain *et all*	Punctured cc
1993	Berrou *et all*	Turbo codes

Table 2.1: History of Error Correcting Codes

2.2 Linear Block Codes

Block codes are not necessarily linear, but in general all block codes used in practice are linear. Linear codes are divided into two categories: block codes and convolutional codes. In a block code, the transmitted sequence is always a multiple of n-bit long blocks. This is because information is transmitted in blocks of data that are n bits long (hence the name). For an (n,k) code, C, the blocks have a length of n, k information bits encoded in them, and n-k parity bits. Taking the repetitive code (3,1) again we have

n = 3 bits in a block

k = 1 message bit

n - k = 2 parity bits

Figure 2.1 shows the information bit and the parity bits.

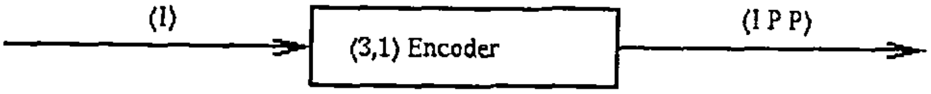


Figure 2.1: The encoding process showing information and parity bits for the (3, 1) repetitive code.

If each block is n bits long, there are 2^n possible combinations. However, linear block codes consist of 2^k possible codewords. These 2^k possibilities form a subset of the 2^n set. These are called valid codewords. The transmitter only outputs valid codewords, but the receiver receives the same codeword with an error pattern added to it:

u(x) = (1 1 0 1 0 0 0)

e(x) = (0 0 1 0 0 0 0)

---

r(X) = (1 1 1 1 0 0 0)

If no errors occurred, then e(x) would have been equal to (0 0 0 0 0 0 0). Since an error did occur in position 3 (counted from left), the receive bit in position 3 was inverted. The receiver would know that (1 1 1 1 0 0 0) was an error, because it did not fall in the list of valid codewords. The receiver would then signal that an error had occured for error detection. For error correction it would choose the codeword that was closest to the r(x) pattern, hopefully (1 1 0 1 0 0 0), and pass this to the receiver.

Message	Cooresponding codeword	Hamming Distance to (1 1 1 1 0 0 0)
(0 0 0 0)	(0 0 0 0 0 0 0)	4
(1 0 0 0)	(1 1 0 1 0 0 0)	1
(0 1 0 0)	(0 1 1 0 1 0 0)	3
(1 1 0 0)	(1 0 1 1 1 0 0)	2
(0 0 1 0)	(1 1 1 0 0 1 0)	2
(1 0 1 0)	(0 0 1 1 0 1 0)	3
(0 1 1 0)	(1 0 0 0 1 1 0)	5
(1 1 1 0)	(0 1 0 1 1 1 0)	4
(0 0 0 1)	(1 0 1 0 0 0 1)	3
(1 0 0 1)	(0 1 1 1 0 0 1)	2
(0 1 0 1)	(1 1 0 0 1 0 1)	4

(1 1 0 1)	(0 0 0 1 1 0 1)	5
(0 0 1 1)	(0 1 0 0 0 1 1)	5
(1 0 1 1)	(1 0 0 1 0 1 1)	4
(0 1 1 1)	(0 0 1 0 1 1 1)	6
(1 1 1 1)	(1 1 1 1 1 1 1)	3

Table 2.2: Linear block code with k = 4 and n = 7

The mapping of the information sequence or message to its cooresponding codeword by the encoder is one to one. Notice, that each of the codewords in the code C above have at least 3 bits or more different from other codewords. Therefore the minimum distance (the minimum number of bits that must be changed to end up with another codeword) is 3. This could be written as for any codewords v and w in C the d(v,w) is >= 3. Therefore, the r(x) pattern (1 1 1 1 0 0 0) would be mapped to u'(x) = (1 1 0 1 0 0 0) because it is the codeword with the closest hamming distance to r(x). An important feature of linear codes, is that the zero codeword is always a member of the valid codewords.

Theorem 2.1

The minimum distance of a linear code is equal to the lowest weightof a nonzero codeword.

t is the number of errors that can be corrected. s is the number of errors that can be detected. Because a decoder must be able to detect an error before it corrects an error, s is always more than or equal to t.

$s >= t$

$$d_{min} >= s + t + 1 \qquad (2.1)$$

$$d_{min} >= 2t + 1 \text{ (for error correction only code)} \qquad (2.2)$$

s = 4	t = 0
s = 3	t = 1
s = 2	t = 2

The possibilities of t and s with d_{min} = 5

A linear code maps k-bit messages (2^k possible messages) to n-bit codewords. Only 2^k of the 2^n possibilities are used, leaving 2^n - 2^k illegal bit sequences. The next step in error correction is assigning each of these illegal bit sequences to their nearest codeword. The standard array demonstrates how this is done:

Message	Coset Leaders						
(0 0 0 0 0 0 0)	0000001	0000010	0000100	0001000	0010000	0100000	1000000
(1 1 0 1 0 0 0)	1101001	1101010	1101100	1100000	1111000	1001000	0101000
(0 1 1 0 1 0 0)	0110101	0110110	0110000	0111100	0100100	0010100	1110100
(1 0 1 1 1 0 0)	1011101	1011110	1011000	1010100	1001100	1111100	0011100
(1 1 1 0 0 1 0)	1110011	1110000	1110110	1111010	1100010	1010010	0110010

(0 0 1 1 0 1 0)	0011011	0011000	0011110	0010010	0001010	0111010	1011010
(1 0 0 0 1 1 0)	1000111	1000100	1000010	1001110	1010110	1100110	0000110
(0 1 0 1 1 1 0)	0101111	0101100	0101010	0100110	0111110	0001110	1101110
(1 0 1 0 0 0 1)	1010000	1010011	1010101	1011001	1000001	1110001	0010001
(0 1 1 1 0 0 1)	0111000	0111011	0111101	0110001	0101001	0011001	1111001
(1 1 0 0 1 0 1)	1100100	1100111	1100001	1101101	1110101	1000101	0100101
(0 0 0 1 1 0 1)	0001100	0001111	0001001	0000101	0011101	0101101	1001101
(0 1 0 0 0 1 1)	0100010	0100001	0100111	0101011	0110011	0000011	1100011
(1 0 0 1 0 1 1)	1001010	1001001	1001111	1000011	1011011	1101011	0001011
(0 0 1 0 1 1 1)	0010110	0010101	0010011	0011111	0000111	0110111	1010111
(1 1 1 1 1 1 1)	1111110	1111101	1111011	1110111	1101111	1011111	0111111

Table 2.3: Standard Array for (7, 4) code

The left most column represents the valid codewords. The top row represents the possible error patterns that are to be corrected (coset leaders). All the sequences in row one are mapped to (0 0 0 0 0 0 0), all the sequences in row two are mapped to (1 1 0 1 0 0 0), etc. Since every valid codeword (same as table 2.2) has a hamming distance of atleast three from any other codeword ($d_{min} = 3$), a bit sequence v' created by changing one bit in any codeword v will satisfy the following:

d(v',v) = 1

d(v', any other codeword) >= 2

Therefore, any codeword with only one error will always be closer (smaller hamming distance) to the correct codeword than any other codeword. However, if two errors occur, then:

d(v',v) = 2

d(v', any other codeword) >= 1

and looking at the standard array, there is a 100% chance of a decoding error. A decoding error in forward error correction happens when the decoder chooses the wrong codeword. This normally produces more errors into the codeword (in the process of "correcting" errors that don't exist the decoder actually adds errors). The probablility that the decoder will produce an error is dependent on the type of code, the decoder used, and the probability of a bit error in the channel. This value is of primary concern when choosing an error controlling code. The difficulty of discovering a useful code lies in being able to prove that it has a sufficient minimum distance and that it can be decoded and encoded quickly. For example Cyclic, BCH, Reed Solomon, Reed-Mullerc codes, all have convenient structures that not only allow the minimum distance to be found, but also provide an easy way to encode and decode. The qualification of a "linear block" code does not provide much in way of quick encoding and decoding (aside from a memory decoder which becomes prohibitive for larger n). However, it does provide a structural foundation for other codes to build upon. A linear code C satisfies the following conditions:

1) For any codeword u and v in C, u + v produces another codeword in C. Therefore the code must be closed under addition.

2) For any codeword u in C and element a in GF(2^m), a * u produces another codeword in C.

3) (Distributive law) For any element a in GF(2^m) and any two:

$$a * (u + v) = a * u + a * v$$

4) (Associative Law) For any element a and b in GF(2^m) and any vector v in V,

$$a * (b * v) = (a * b) * v$$

5) Let 1 be the unit element in GF(2^m). Then for any vector v in V, 1 * v = v.

Since each of the 2^k codewords in C is contained in the set of 2^n possible received sequences and satisfies the above properties, the codeword C forms a subspace in V_n. There also exists a dual space of the codewords which produces another code C′ called the dual code of C. Every linear code has a zero codeword since the addition of v + v equals 0.

2.2.1 Generator and Parity Check Matrices and Syndromes

The generator matrix for a linear block code is one of the basis of the vector space of valid codewords. The generator matrix defines the length of each codeword n, the number of information bits k in each codeword, and the type of redundancy that is added; the code is completely defined by its generator matrix. The generator matrix is a k x n matrix that is the row space of V_k. One possible generator matrix for a typical (7,4) linear block code: **Table 2.4: A (7, 4) linear block code generator matrix**

$$G = \begin{array}{|c|c|c|c|c|c|c|} \hline 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ \hline 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ \hline 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ \hline 1 & 0 & 1 & 0 & 0. & 0 & 1 \\ \hline \end{array}$$

Multiplying the k-bit information sequence u(x) by the (n,k) generator matrix gives the n-bit codeword.

$u(x) = (u_0\ u_1\ ...\ u_{k-1})$ $u(x) * G = ((u_0g_{00} + u_1g_{10} + ... + u_{k-1}g_{k-1,0})\ (u_0g_{01} + u_1g_{11} + ... + u_{k-1}g_{k-1,1})\ ...\ (u_0g_{0,n-1} + u_1g_{1,n-1} + ... + u_{k-1}g_{k-1,n-1}))$

For example, using the matrix in table 3.5 and u(x) = (1 0 1 1)

$u(x) * G = ((1*1 + 0*0 + 1*1 + 1*1)$

$(1*1 + 0*1 + 1*1 + 1*0)$

$(1*0 + 0*1 + 1*1 + 1*1)$

$(1*1 + 0*0 + 1*0 + 1*0)$

$(1*0 + 0*1 + 1*0 + 1*0)$

$(1*0 + 0*0 + 1*1 + 1*0)$

$(1*0 + 0*0 + 1*0 + 1*1)) = (1\ 0\ 0\ 1\ 0\ 1\ 1)$

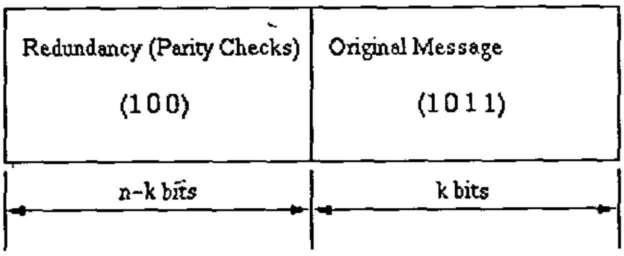


Figure 2.2: Systematic Code

Note: The generator matrix in table 3.5 will always produce a codeword with the last four bits being equal to the information sequence. When the information bits are passed into the codeword unchanged and parity bits are added this is called a systematic code (see figure 2.2). If the information bits are not directly represented then the code is called a nonsystematic code. Typically, systematic codes have every advantage of their nonsystematic counterparts plus the encoder only needs to produce (n-k) bits instead of the entire codeword. There are other advantages that can be gained by using systematic codes. Nonetheless, we could manipulate the generator matrix in table 3.5 with elementary row operations (adding the second row to the first and the fourth to the third and second) to produce:

G =

1	0	1	1	1	0	0
1	1	0	0	1	0	1
0	1	0	0	0	1	1
1	0	1	0	0	0	1

Table 2.5: A (7, 4) linear block code generator matrix in nonsystematic form

The information sequence u(x) = (1 0 1 1) would now give:

u(x) * G = ((1*1 + 0*1 + 1*0 + 1*1)

(1*0 + 0*1 + 1*1 + 1*0)

(1*1 + 0*0 + 1*0 + 1*1)

(1*1 + 0*0 + 1*0 + 1*0)

(1*1 + 0*1 + 1*0 + 1*0)

(1*0 + 0*0 + 1*1 + 1*1)

(1*0 + 0*1 + 1*1 + 1*1)) = (1 1 0 1 1 0 0)

Any generator matrix for a linear block code can be manipulated into reduced row echelon form (RREF) with elementary row operations. By swapping columns (which changes the code, but none of its fundamental properties -- called an equivalent code), we can change any RREF matrix into a systematic matrix. This means that all linear block codes are equivalent to a systematic code with all the same properties. Once the message has been encoded (converted into a codeword through a one to one relationship), the codeword is transmitted. The receiver then picks up the received sequence r(x). r(x) consists of the original codeword plus the error pattern.

$$r(x) = v(x) + e(x) \qquad (2.3)$$

The error pattern is an n-tuple with ones cooresponding to transmission errors and zeros to bits that were transmitted correctly. Manipulating the above expression we find that

$$e(x) = r(x) + v(x) \qquad (2.4)$$

Note: The only way an undetected error can occur is if r(x) is another codeword. The receiver would then have no way of telling if the error pattern were zero or non-zero. If r(x) is another codeword and the code is linear then e(x) must also be a codeword. When r(x) equals v(x), e(x) is equal to the zero codeword. The syndrome is the receive sequence multiplied by the transposed parity check matrix H.

$$s(x) = r(x) * [\text{dual space of } g(x) \text{ in } V_n]^T$$

$$s(x) = r(x) * H^T$$

$$s(x) = (v(x) + e(x)) * H^T \qquad (2.5)$$

Because H is the null space of G.

$$s(x) = e(x) * H^T \qquad (2.6)$$

The syndrome is a (n-k)-tuple that has a one to one correspondence with the correctable error patterns. The syndrome depends only on the error pattern and is independent of the transmitted codeword.

Most codes do not use *all* of the redundancy that has been added for error correction. The only two codes known to do this are Hamming (2^m - 1, 2^m - m - 1) and Golay (23, 12) codes. These codes are called perfect codes.

2.2.2 Memory Decoder

A memory decoder for a linear block code requires $2^{n-k} + 2^k$ bits of storage. This is reasonable for codes with smaller values on n, however, as n gets larger the cost of memory becomes prohibitive. Since most linear block codes are more efficient at large values of n, memory decoding is not an option. Memory decoding can be easily implemented in software.

The received sequence r(x) is read in and the syndrome is calculated through a series of multipliers and adders. There are 2^{n-k} possibilities for syndromes and each memory location must store s^k bits for the error pattern. Since the generator matrix will be in systematic form the only received bits that we need the error pattern for are the bits that coorespond with message bits.

2.2.3 Codeword Structure

Block codes accept a block of k message bits and produce a block of n coded bits. By predetermined rules, n-k redundant bits are added to the k message bits to form the n coded bits. These codes are referred to as (n, k) block codes.

Message	Codeword	Weight	Message	Codeword	Weight
0000	0000000	0	1000	1101000	3
0001	1010001	3	1001	0111001	4
0010	1110010	4	1010	0011010	3
0011	0100011	4	1011	1001011	4
0100	0110100	3	1100	1011100	4
0101	1100101	4	1101	0001101	3
0110	1000110	3	1110	0101110	4
0111	0010111	4	1111	1111111	7

Table 2.6: Linear block code with k = 4 and n = 7

Block codes in which the message bits are included in unaltered form are called systematic codes. Figure 2.3 shows codeword structure for systematic codes.

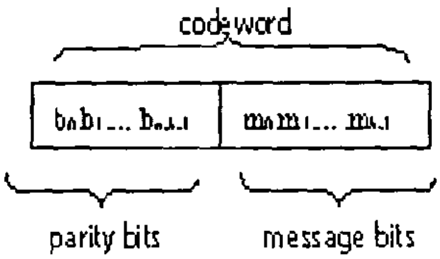


Figure 2.3: Codeword structure

2.2.4 Generator Matrix G and Encoding Operation

Parity bits are linear sum of the message bits:

bi = p0im0 + plim1 + ... + pk-1,i mk-1 (+: modulo-2 addition).

It can also be written as:

$$P = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0,n-k-1} \\ p_{10} & p_{11} & \cdots & p_{1,n-k-1} \\ \vdots & \vdots & \ddots & \\ p_{k-1,0} & p_{k-1,1} & & p_{k-1,n-k-1} \end{bmatrix} \begin{array}{l} \\ \\ \mathrm{m} = [m_0 \quad m_1 \quad \cdots \quad m_{k-1}] \\ \mathrm{b} = [b_0 \quad b_1 \quad \cdots \quad b_{n-k-1}] \end{array} \quad \mathrm{b} = \mathrm{mP} \qquad (2.7)$$

Generator Matrix is a k´n matrix defined as

$$\mathrm{G} = [\mathrm{P} \vdots \mathrm{I}_k], \qquad \text{where } \mathrm{I}_k : k \times k\, identity matrix \qquad (2.8)$$

Encoding operation is defined as

$$\mathrm{c} = \mathrm{mG} \qquad (2.9)$$

Parity Check Matrix H and Error Correction Decoding

Parity Check Matrix is a (n-k) ´n matrix defined as

$$\mathrm{H} = [\mathrm{I}_{n-k} \vdots \mathrm{P}^{T}], \text{ where } \mathrm{I}_{n-k} : (n-k) \times (n-k)\, identity matrix \qquad (2.10)$$

and x is a code vector generated by matrix G is, if and only if, $\mathrm{xH}^T = 0$.

Suppose vector x was sent over a noisy channel. Let y be the received vector which can be expressed as y = x + e, where e is the error vector (or pattern). To determine the error vector e, a 1´(n-k) vector s, which is referred to as the syndrome, is computed.

$$s = y\mathrm{H}^T = (x + e)\mathrm{H}^T = \mathrm{eH}^T \qquad (2.11)$$

For any error vector e, there are 2k distinct vectors: e + xi, i = 0,1, ..., 2k-1, which is known as coset and e is the coset leader. Each coset has a unique syndrome. Once the syndrome is computed, the error vector (which is the coset leader) is known. The corrected vector is then computed: x = y + e.

2.3 Convolutional Codes

A firm understanding of convolutional codes is an important prerequisite to the understanding of turbo codes.

2.3.1 Encoder Structure

A convolutional code introduces redundant bits into the data stream through the use of linear shift registers as shown in Figure 2.4

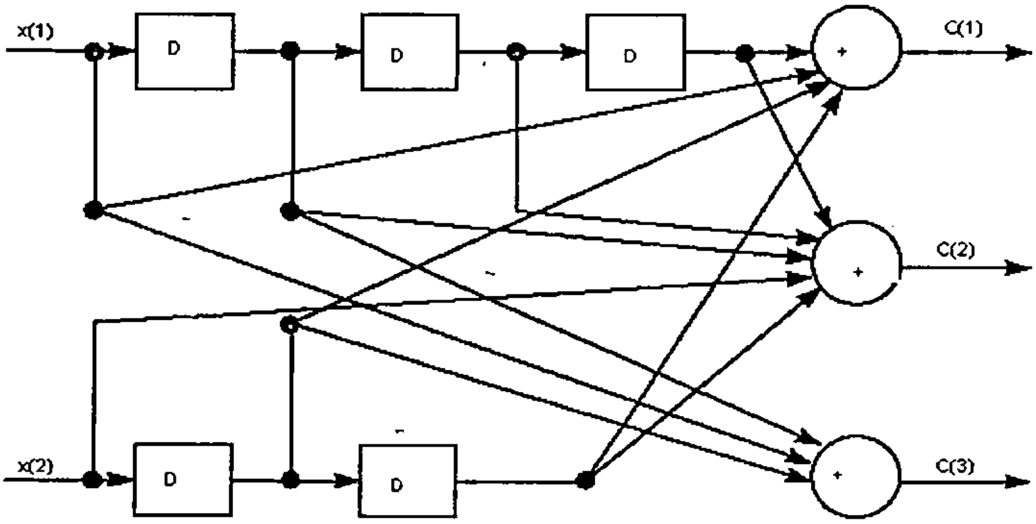


Figure 2.4: Convolutional encoder

where

x(i) is an input information bit stream and c(i) is an output encoded bit stream

The information bits are input into shift registers and the output encoded bits are obtained by modulo-2 addition of the input information bits and the contents of the shift registers. The connections to the modulo-2 adders were developed heuristically with no algebraic or combinatorial foundation.

The code rate r for a Convolutional code is defined as:

$$r = k/n \tag{2.12}$$

where k is the number of parallel input information bits and n is the number of parallel output encoded bits at one time interval. The constraint length K for a convolutional code is defined as:

$$K = m + 1 \tag{2.13}$$

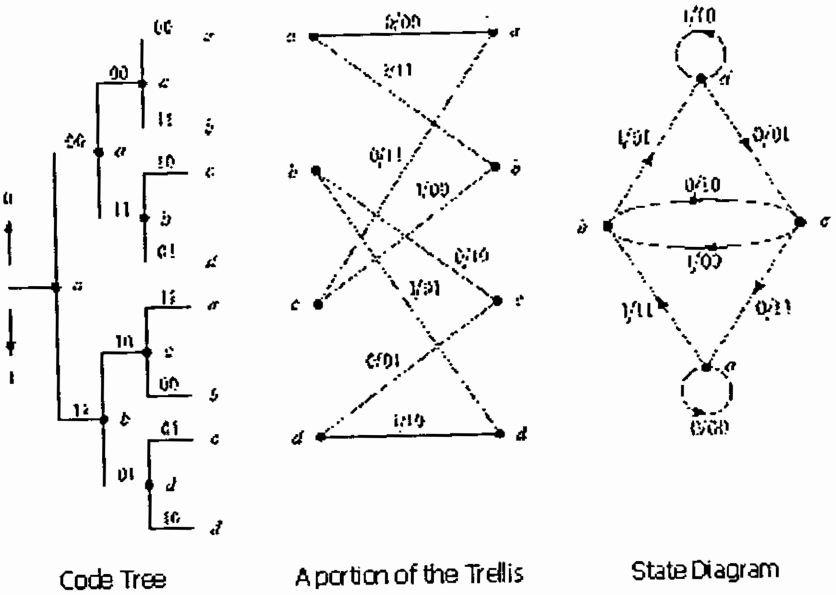
2.3.2 Encoder Representations

The encoder can be represented in several ways:

1. Tree Diagram Representation

2. State Diagram Representation

3.Trellis Diagram Representation



Code Tree A portion of the Trellis State Diagram

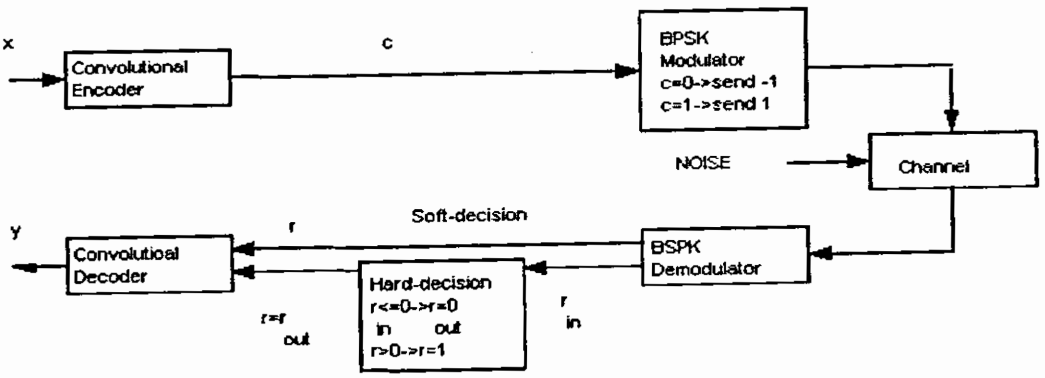
State	Register Content
a	00
b	10
c	01
d	11

Figure 2.5 Encoder Representation

2.3.3 Convolutional Decoding

2.3.3.1 Hard-Decision and Soft-Decision Decoding

Hard-decision and soft-decision decoding refer to the type of quantization used on the received bits. Hard-decision decoding uses 1-bit quantization on the received channel values. Soft-decision decoding uses multi-bit quantization on the received channel values. For the ideal soft-decision decoding (infinite-bit quantization), the received channel values are directly used in the channel decoder.



The selected metric represents the survivor path and the remaining metrics represent the non survivor paths. The survivor paths are stored while the non survivor paths are discarded in the trellis diagram. The Viterbi algorithm selects the single survivor path left at the end of the process as the ML path. Trace-back of the ML path on the trellis diagram would then provide the ML decoded sequence.

2.3.3.1.1 Hard Decision Decoding

The hard-decision Viterbi algorithm (HDVA) can be implemented as follows [Rap96], [Wic95]:

$S_{(k,t)}$ is the state in the trellis diagram that corresponds to state S_k at time t. Every state in the trellis is assigned a value denoted $V(S_{k,t})$.

1. (a) Initialize time t = 0.

   (b) Initialize $V(S_{0,0}) = 0$ and all other

2. $V(S_{k,t}) = +.$.

3. (a) Set time t = t+1.

4. (b) Compute the partial path metrics for all paths going to state S_k at time t. First, find the t^{th} branch metric

5. $M(r_t|y_t) =. \ {}^{n}{}_{j=1} M(r_t^{(j)}|\, y_t^{(j)})$

6. This is calculated from the Hamming distance . ${}^{n}{}_{j=1} M\, |r_t^{(j)} - y_t^{(j)}|$. Second, compute the tth partial path metric.

7. $M^t(r|\, y) =. \ {}^{t}{}_{i=0} M(r_i|\, y_i^{)}$. This is calculated from $V(S_{k,t-1}) + M(r_t|y_t)$

8. (a) Set $V(S_{k,t})$ to the "best" partial path metric going to state S_k at time t.

Conventionally, the "best" partial path metric is the partial path metric with the smallest value.

9. (b) If there is a tie for the "best" partial path metric, then any one of the tied partial path metric may be chosen.

Store the "best" partial path metric and its associated survivor bit and state paths.

10. If t < L+m-1, return to Step 2

2.3.3.1.2 Soft-Decision Viterbi Algorithm

There are two general methods of implementing a soft-decision Viterbi algorithm. The first method (Method 1) uses Euclidean distance metric instead of Hamming distance metric. The received bits used in the Euclidean distance metric are processed by multi-bit quantization. The second method (Method 2) uses a correlation metric where its received bits used in this metric are also processed by multi-bit quantization.

Method 1

In soft-decision decoding, the receiver does not assign a zero or a one (single-bit quantization) to each received bit but uses multi-bit or infinite-bit quantized values [Wic95]. Ideally, the received sequence $\tilde{r}$ is infinite-bit quantized and is used directly in the soft-decision Viterbi decoder. The soft-decision Viterbi algorithm is similar to its hard-decision algorithm except that squared Euclidean distance is used in the metric instead of Hamming distance.

The soft-decision Viterbi algorithm (SDVA1) can be implemented as follows

$S_{(k,t)}$ is the state in the trellis diagram that corresponds to state S_k at time t. Every state in the trellis is assigned a value denoted $V(S_{k,t})$.

1. (a) Initialize time $t = 0$.

   (b) Initialize $V(S_{0,0}) = 0$ and all other

2. $V(S_{k,t}) = +.$.

3. (a) Set time $t = t+1$.

   (b) Compute the partial path metrics for all paths going to state S_k at time t. First, find the t^{th} branch metric

4. $M(r_t|y_t) =. \ {}^{n}{}_{j=1} M(r_t^{(j)}| y_t^{(j)})$

This is calculated from the Euclidean distance . ${}^{n}{}_{j=1} M(r_t^{(j)} | y_t^{(j)})$. Second, compute the t^{th} partial path metric

$$M^t(r| y) =. \ {}^{t}{}_{i=0} M(r_i | y_i) \qquad (2.14)$$

This is calculated from

$$V(S_{k,t-1}) + M(r_t |y_t) \qquad (2.15)$$

5. (a) Set $V(S_{k,t})$ to the "best" partial path metric going to state S_k at time t

   Conventionally, the "best" partial path metric is the partial path metric with the smallest value.

   (b) If there is a tie for the "best" partial path metric, then any one of the tied partial path metric may be chosen.

6. Store the "best" partial path metric and its associated survivor bit and state paths.

7. If $t < L+m-1$, return to Step 2

Method 2

The second soft-decision Viterbi algorithm (SDVA2) is developed below. The likelihood function is represented by a Gaussian probability density function

$$P\,(r_t^{(j)} \mid y_t^{(j)}) = 1/.\,.\,No * e^{-(r_t^{(j)} - y_t^{(j)} . E_b)^2/N_o} \qquad (2.16)$$

Where E_b is received energy per code-word bit and N_o is one sided noise spectral density. The received bit is Gaussian random variable with mean $y_i^{(j)}$. E_b and variance $N_o/2$.

Bit metric is defined by:

$$M(r_t^{(j)} \mid y_t^{(j)}) = r_t^{(j)} y_t^{(j)} \qquad (2.17)$$

The remaining steps are the same of SDVA1 algorithms except in the second step the tth branch metric is calculated from the correlation of $r_t^{(j)}$ and $y_t^{'(j)}$, . $^n{}_{j=1}\, r_i^{(j)} y_i^{(j)}$.

In the third step (a) part the best partial path metric is the partial path metric with the largest value.Generally with soft-decision decoding, approximately 2dB of coding gain over hard-decision decoding can be obtained in Gaussian channels.

2.4 Performance Analysis of Convolutional Code

The performance of convolutional codes can be quantified through analytical means or by computer simulation. The analytical approach is based on the transfer function of the convolutional code which is obtained from the state diagram. The process of obtaining the transfer function and other related performance measures are described below.

2.4.1 Transfer Function of Convolutional Code

The analysis of Convolutional codes is generally difficult to perform because traditional algebraic and combinatorial techniques cannot be applied. These heuristically constructed codes can be analyzed through their transfer functions. By utilizing the state diagram, the transfer function can be obtained. With the transfer function, code properties such as distance properties and the error rate performance can be easily calculated.

To obtain the transfer function, the following rules are applied:

1. Break the all-zero (initial) state of the state diagram into a start state and an end state. This will be called the modified state diagram.

2. For every branch of the modified state diagram, assign the symbol D with its exponent equal to the Hamming weight of the output bits.

3. For every branch of the modified state diagram, assign the symbol J.

4. Assign the symbol N to the branch of the modified state diagram, if the branch transition is caused by an input bit 1.

The modified state diagram is shown.

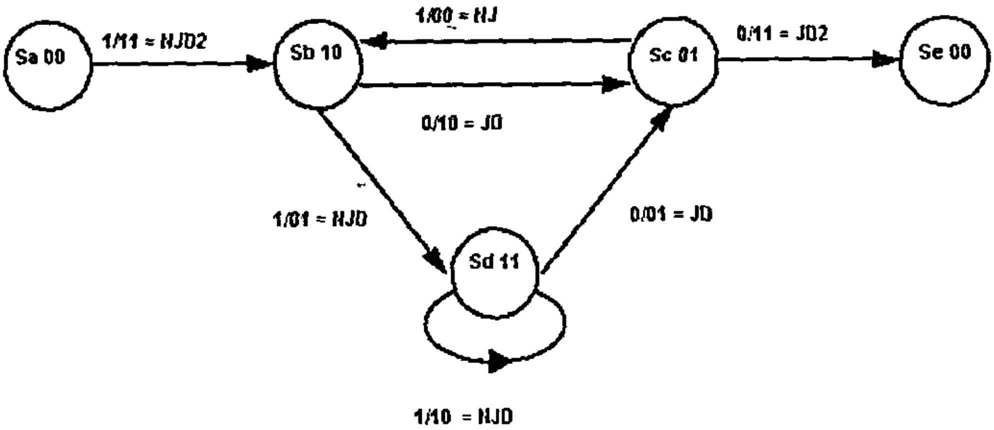


Figure 2.7: The modified state diagram

where

“Sa” is the start state and “Se” is the end state.

Nodal equations are obtained for all the states except for the start state in Figure 2.7.

These results are

$$Sb = NJD^2Sa+ NJSc \quad (2.18)$$

$$S_c = JDSb + JDSd \quad (2.19)$$

$$S_d = NJDSb + NJDS\ Sd \quad (2.20)$$

$$S_e = JD^2SC \quad (2.21)$$

The transfer function is defined to be

$$T\ (D, N, J) = S_{end}\ (D,N,J) / S_{start}\ (D,N,J) \quad (2.22)$$

and for Figure 2.7,

$$T\ (D, N, J) = S_e / S_a \quad (2.23)$$

By substituting and rearranging,

$$T\ (D, N, J) = NJ^3D^5/1\text{-}(NJ + NJ^2\ D) \quad (2.24)$$

(closed form)

$$T(D,N,J) = NJ^3D^5 +(N^2J^4 +N^2J^5)D^6+(N^3J^5+ 2N^3J^6+ N\ ^3J^7\ )D^7+\ldots \quad (2.25)$$

(expanded polynomial form)

2.4.2 Distance Properties

The free distance between a pair of convolutional codeword is the hamming distance between the pair of code words. The minimum free distance, d free, is the minimum hamming distance between all pairs of complete convolutional code words and is defined as

$$d_{free} = \min \{d(y_1, y2) | y_1 . y_2\} \text{ [Wic95]} \quad (2.26)$$

$$= \min \{w(y) | y . 0\} \text{ [Wic95]} \quad (2.27)$$

where d (·, ·) is the Hamming distance between a pair of Convolutional code word and w(·) 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. From the transfer function, the minimum free distance is identified as the lowest exponent of D. From the above transfer function for Figure,

$d_{free} = 5$.

Also, if N and J are set to 1, the coefficients of D^{i}'s represent the number of paths through the trellis with weight D^{i}. More information about the codeword is obtained from observing the exponents of N and J. For a codeword, the exponent of N indicates the number of 1s in the input sequence, and the exponent of J indicates the length of the path that merges with the all-zero path for the first time [Pro95].

2.4.3 Error Probabilities

There are two error probabilities associated with Convolutional codes, namely first event and bit error probabilities. The first event error probability, P_e, is the probability that an error begins at a particular time. The bit error probability, P_b, is the average number of bit errors in the decoded sequence. Usually, these error probabilities are defined using the Chernoff Bounds and are derived in [PrX95], [Rhe89], [Wic95].

For hard-decision decoding, the first event error and bit error probabilities are defined as

$$P_e < T(D,N,J)|_{D=.\,4P(1-P),\,N=1,\,J=1} \tag{2.28}$$

and

$$P_b < (dT(D,N,J)/dN)|_{D=.\,4P(1-P),\,N=1,\,J=1} \tag{2.29}$$

Where

$$P = Q(.\ 2rE_b/N_o) \tag{2.30}$$

and

$$Q(x) = f1/.\ 2.\ *\ e^{-u2/2}du \tag{2.31}$$

For soft-decision decoding, the first event error and the bit error probabilities are defined as:

$$P_e < T\,(D, N, J)\,|_{D=e^{-rEb/No},\,N=1,\,J=1} \tag{2.32}$$

and

$$P_b < (dT\,(D, N, J)/dN)|_{D=e^{-rEb/No},\,N=1,\,J=1} \tag{2.33}$$

Two other factors also determine the performance of the Viterbi decoder. They are commonly referred to as the deciding depth and the degree of quantization of the received signal.

2.4.4 Decoding Depth

The deciding depth is a window in time that makes a decision on the bits at the beginning of the window and accepts bits at the end of the window for metric computations. This scheme gives up the optimum ML decoding at the expense of using less memory and smaller decoding delay. It has been experimentally found that if the decoding depth is 5 times greater than the constraint length K then the error introduced by the decoding depth is negligible [Pro95].

2.4.5 Degree of Quantization

For soft-decision Viterbi decoding, the degree of the quantization on the received signal can affect the decoder performance. The performance of the Viterbi decoder improves with higher bit quantization. It has been found that an eight-level quantize degrades the performance only slightly with respect to the infinite bit quantized case [Wic95].

2.4.6 Decoding Complexity for Convolutional Codes

For a general Convolutional code, the input information sequence contains k*L bits where k is the number of parallel information bits at one time interval and L is the number of time intervals. Thus results in L+m stages in the trellis diagram. There are exactly 2^{k*L} distinct paths in the trellis diagram, and as a result, an exhaustive search for the ML sequence would have a computational complexity on the order of O[2^{k*L}]. The Viterbi algorithm reduces this complexity by performing the ML search one stage at a time in the trellis. At each node (state) of the trellis, there are 2^{k} calculations. The number of nodes per stage in the trellis is 2m. Therefore, the complexity of the Viterbi algorithm is on the order of O[(2k)(2m)(L+m)]. This significantly reduces the number of calculations required to implement the ML decoding because the number of time intervals L is now a linear factor and not an exponent factor in the complexity. However, there will be an exponential increase in complexity if either k or m increases.

3. Turbo codes

Lately in [4], the proposal of Parallel-Concatenated Convolutional Codes *(PCCC)*, called turbo codes, was introduced.The introduction of turbo codes has increased the interest in the coding area since these codes give most of the gain promised by the channel-coding theorem. Turbo codes have an astonishing performance of bit error rate (BER) at relatively low E_b/N_0.To give an idea of how powerful turbo codes are, for a frame size of 256×256=65536 bits we can achieve a BER=10^{-5} over AWGN channel at only E_b/N_0=0.7dB, which is very close to Shannon limit [4].

In the following sections the structure of both encoder and the decoder are explained. The effects of interleaving on the code performance are also discussed.

3.1 Turbo encoder

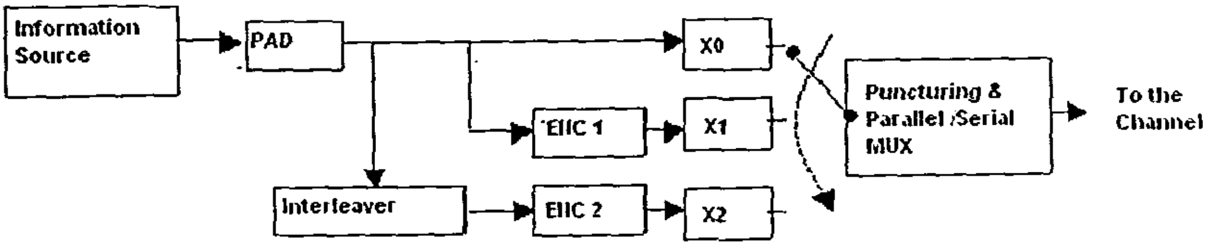


Figure 3.1: Simplified Turbo Encoder

In a simplified turbo encoder, there are two convolutional encoders in parallel. The information bits are scrambled before entering the second encoder. Input bits are appended to the convoluted output - i.e. the code is *systematic*-followed by the parity check bits from the first encoder and then the parity bits from the second encoder, as depicted in Figure 3.1.

The simplified turbo code block diagram shows only two branches. In general, one can have multiple turbo encoders with more than two branches. The Convolutional code at every branch is called the *constituent code (CC)*. The CCs can have similar or different generator functions. We will discuss the usual configuration with two branches having the same CC. A PAD is shown in the Figure 3.1 to append the proper sequence of bits to terminate all the encoders to the all-zero state. This is because a convolutional code may be used to generate a block code if we use beginning and tail bits. If we have one then the required tail is a sequence of zeros with length equal to the memory order m. The problem of terminating both encoders simultaneously seems to be difficult because of the interleaver. However, it is still possible to do with m tail bits only [6].

In general we can have another interleaver before the first encoder but usually it is replaced with a delay line to account for the interleaver delay and keep the branches working simultaneously.*Puncturing* can be introduced to increase the rate of the convolutional code beyond that resulting from the basic structure of the encoder. Some codes are called recursive since the state of the internal shift register depends on the past outputs. Figure 3.2 illustrates a non-recursive and non- systematic convolutional code with its corresponding recursive systematic code. X_0 and $X_1$are the check bits. Note that for the systematic Convolutional encoder, one of the outputs, X_0, is exactly the input

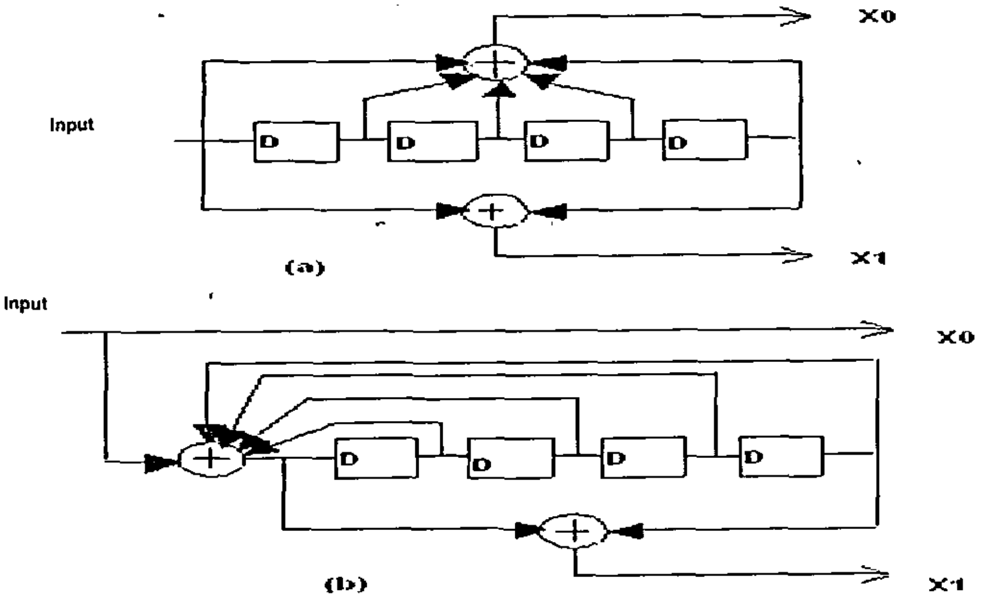


Figure 3.2: (a) Classical Non-Recursive and Non-Systemic Code (b) Recursive Systemic Code

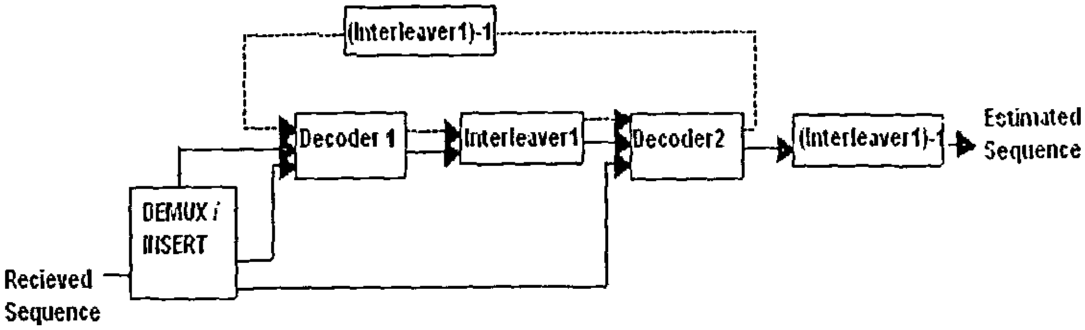


Figure 3.3: Block Diagram of Turbo Codes

An RSC encoder can be obtained from a non-systematic & non-recursive encoder by setting one of the outputs equal to the input (if we have one input) and using a feed back. The trellis and the free distance (d_{free}) also will be the same for both codes [4].Of course; the output sequence does not correspond to the same input in the two codes because the two generator functions are not the same.

3.2 Turbo decoder

The decoder works in an iterative way. Figure 4 shows a block diagram of a turbo decoder. The iteration stage is shown with doted lines to differentiate it from the initialization stage. Only one loop is performed at a time. In practice the number of iterations does not exceed 18, and in many cases 6 iterations can provide satisfactory performance [4].Actually, the term turbo code is given for this iterative decoder scheme with reference to the turbo engine principle. The first decoder will decode the sequence and then pass the hard decision together with a reliability estimate of this decision to the next decoder. Now, the second decoder will have extra information for the decoding; a priori value together with the sequence. The interleaver in-between is responsible for making the two decisions uncorrelated and the channel between the two decoders will seem to be memory less due to interleaving. The exact procedure in what information is passed to the next decoder or next iteration stage is a subject of research. In the next section, we describe a widely accepted decoding algorithm, which is the modified version of Viterbi algorithm.

3.3 Decoding Algorithm: Soft Output Viterbi Algorithm (SOVA)

Algorithms used in decoding convolutional codes can be modified to be used in decoding turbo code.In the original paper on turbo codes [4], a modified BAHL et algorithm was proposed for the decoding stage. This algorithm is based on Maximum A-posteriori Probability (MAP). The problem with this algorithm is the inherent complexity and time delay. MAP algorithm was originally developed to minimize the bit-error probability instead of the sequence error probability. The algorithm, although optimal, seem less attractive due to the increased complexity.

Viterbi algorithm is an optimal decoding method that minimizes the probability of sequence error for convolutional codes. A modified version of Viterbi algorithm, called SOVA (Soft Output Viterbi Algorithm), which uses soft outputs, is introduced in [8] [9].

3.3.1 Understanding SOVA

It is proposed that an iterative decoding scheme should be used. The decoding algorithm is similar to Viterbi algorithm in the sense that it produces soft outputs. While the Viterbi algorithm outputs either 0 or 1 for each estimated bit, the turbo code decoding algorithm outputs a continuous value of each bit estimate. While the goal of the Viterbi decoder is to minimize the code word error by finding a maximum likelihood estimate of transmitted code word, the soft output decoding attempts to minimize bit error by estimating the posterior probabilities of individual bits of the code word. We called the decoding algorithm Soft Decision Viterbi Decoding. The turbo decoder consists of M elementary decoders - one for each encoder in turbo encoding part. Each elementary decoder uses the Soft Decision Viterbi Decoding to produce a soft decision for each received bit. After an iteration of the decoding process, every elementary decoder shares its soft decision output with the other$M$-1 elementary decoders.

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

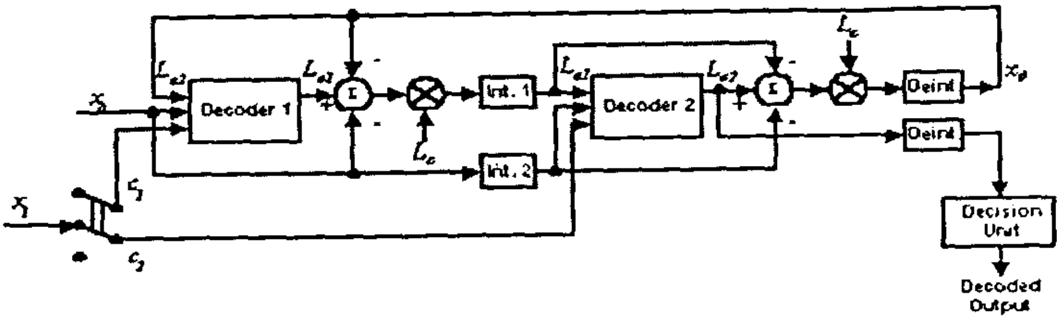


Figure 3.4: Turbo Decoder structure

3.3.2 The Overall Schema:

The turbo-code decoder is described in the figure 3.4. Assuming zero decoder delay in the turbo-decoder, the Decoder 1 computes a soft-output from the systematic data (x_0), code information of Encoder 1 (y_1) and a-priori information (La_2). From this output the systematic data (x_0) and a-priori information (La_2) are subtracted. The result is multiplied by the scaling factor called channel reliability L_c to compensate the distortion. The result is uncorrelated with x_k and is denoted as Le_1,for extrinsic data from Decoder 1.

Decoder 2 takes as input the interleaved version of Le_1 (the a-priori information La1), the code information of second Encoder (y_2) and the interleaved version of systematic data ($\alpha(x_k)$). Decoder 2 generates a soft output, from which the systematic data ($L_c\alpha(x_0)$) and a-priori information (La_1) was subtracted. The result is multiplied by the scaling factor called channel reliability L_c to compensate the distortion. The extrinsic data from Decoder 2 (Le_2) is interleaved to produce La_2, which is fed back to Decoder 1 and the iterative process continues.

3.3.3 Steps of SOVA:

Step 1: Form the Trellis

Inside the decoder, Soft-Output Viterbi Algorithm (SOVA) is used to determine the result with maximum likelihood. The process SOVA is similar to that Viterbi algorithm. A trellis is formed first.

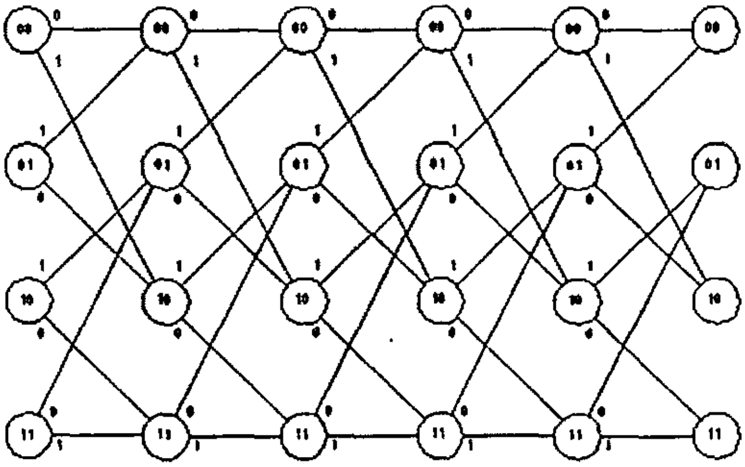


Figure 3.5: Trellis Diagram

The trellis is to show how the codes are output by encoder. The bits in each node the encoder output (decoder input) for all combination of states and inputs of the encoders are summarized as on the next page:

State 1	State 2	Input	Output	Next State 1	Next State 2	Decoder Input
0	0	0	0	0	0	00
0	0	1	1	1	0	11
0	1	0	0	1	0	00
0	1	1	1	0	0	11
1	0	0	1	1	1	01
1	0	1	0	0	1	10
1	1	0	1	0	1	01
1	1	1	0	1	1	10

Table 3.1: Turbo Encoder working

Step 2: Determine the Accumulated Maximum Likelihood for Each State

From the above Table 3.1, each state has its own input bit pattern. When bit streams are inputted to decoder, they can be compared to the input ($x_0 x_1$) to see whether they are matched. The result is represented by likelihood of input:

L_i = -1 If no bits are matched.

...0 if 1 bit is matched.

...1 if both bits are matched.

The overall likelihood of a transition is sum of likelihood of input and a-prior likelihood information (L_p).

$$L = L_i + L_p \tag{3.1}$$

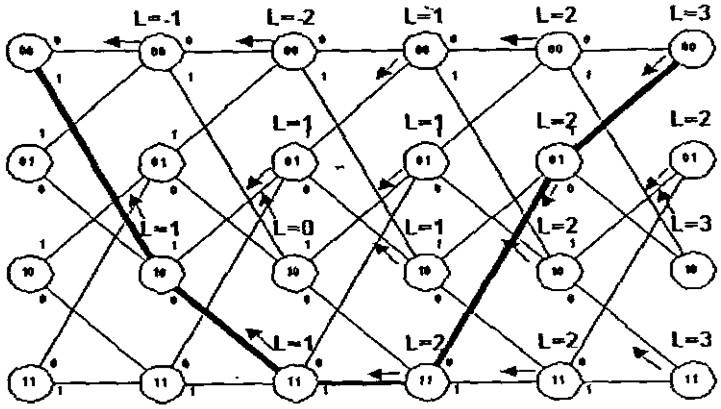
$$L_p = 0.5 \text{ x a-prior information } L_{an} \text{ if } x_0 \text{ are } 1 \tag{3.2}$$

$$\ldots -0.5 \text{ x a-prior information } L_{an} \text{ if } x_0 \text{ are } 0 \tag{3.3}$$

The algorithm is as follows:

Start from state 00, the overall likelihood of each transition is evaluated. The overall likelihood of each node is obtained by the maximum accumulated likelihood. With this algorithm, the trellis in figure 7 will be obtained.

Step 3: Find the Surviving Path



original	11	01	10	01	11
received	11	11	10	00	11

Figure 3.6: Surviving Path in Trellis Diagram

The surviving path is thus derived by tracing back from last stage (stage 5) to the first one (stage 0) and is the results of hard-output Viterbi Algorithm.

Let u= [1 0 1 0 1]

Step 4: Determine the Soft Output

To find the soft-output, non-surviving paths are considered. We consider the non-surviving path the one that is produced by making different decision in one of the stage in tracing back. For example, when tracing back from (last stage) stage 5, one of the non-surviving path is found by tracing back to state 00 instead of state 01 from stage 5 to stage 4. Following the arrows, bit 1 is also 1. We found that bit 1 will have the following results when decision is changed in different stages:

Stage	1	2	3	4	5	
Bit 1	X	X	0	1	1	where X means cannot trace back to state 00 in stage 0
Delta	-	-	3	1	2	

Table 3.2: Non-Surviving paths are considered

Here we define a function delta to describe the tendency to have non-surviving path. It is the difference in overall likelihood when a different decision is possible in a particular stage.The soft-output of bit 1 is evaluated by the following formula:

bit value x *min*(*delta* which make bit 1 change to value other than *bit value*) (3.4)

bit value = 1 for bit output = 1 (3.5)

...-1 for bit output = 0 (3.6)

In bit one, the decision only changes when a state changes in stage 3. The minimum delta is 3. Thus the soft-output of the bit one is 3.

Repeat it for bit two (originally bit two = 0),

Stage	2	3	4	5	
Bit 1	X	1	1	1	where X means cannot trace back to state 00 in stage 0
Delta	-	3	1	2	

Table 3.3: Non-Surviving paths are considered

From the above results, in bit two, the decision changes when a state changes in stage 3, 4 and 5. The minimum delta is 1. Thus the soft-output of the bit two is -1.Using this algorithm, the soft-output will become

[3 -1 1 -1 2]

Step 5: Feeding Data to another Decoder

After the soft-output is evaluated by SOVA decoder, the data will be passed to the second decoder for further decoding. Before passing to data to the second decoder, two processes are performed to the decoder output:

d_1	3	-1	1	-1	2
La_2	0	0	0	0	0
x_0	1	1	1	-1	1
Le_1	2	-2	0	0	1

Table 3.4: The a-prior information (La_2) and the systematic data (x_0) are subtracted

The result is multiplied by the scaling factor called channel reliability L_c. The reason of the factor is because the SOVA algorithm suffers a major distortion which is caused by over-optimistic soft outputs. The factor is used to compensate this distortion.

$$L_c = \text{mean}(Le_1) \times 2 / \text{var}(Le_1) \qquad (3.7)$$

Step 6: Iterative Decoding

The data from first decoder L_cL_{e1}, together with the systematic data α (x_k) and code information from second encoder (y_2), are then fed into the second decoder for decoding; the decoding algorithm is the same as the first one. After decoding, the output of second decoder is processed in the same way and fed back to the first decoder. The process continues. The number of iterations depends on the designer. Usually, the larger the iteration, the more accurate the data but the longer the time its takes for decoding.

Step 7: Decision of Output

After iterations of decoding, the decoding results are the sign of the soft-output of the last decoder. Take the example, for the results of first decoder, the output become:

decoder output	3	-1	1	-1	2
Result	1	0	1	0	1

Table 3.5: The decoder outputs

Which is the same as the input bit stream *u*. i.e., the error can be recovered. [7]

SOVA has only twice the complexity of Viterbi algorithm. The new algorithm will make it possible to integrate both the encoder and the decoder in a single silicon chip with unmatched performance at the present time [4].

3.4 Interleaving

In many turbo codes the same component encoder is used for the first encoder as the second. Recall that the minimum distance is an important measure of a codes error correcting capability. So a problem arises when a low-weight sequence is interleaved to produce another low-weight sequence thus effectively limiting the codes resistance to interference. It is for this reason that there is significant research into the design of suitable interleavers for turbo codes.

An *interleaver* is a device that rearranges the ordering of sequence of symbols in a deterministic manner. Associated with the interleaver is a *deinterleaver* that applies the inverse permutations to restore the original sequence [9].

According to [1] the most critical part in the design of a turbo code is the interleaver. The two main issues in the interleaver design are the interleaver size and the interleaver map. The size of the interleaver plays an important role in the trade off between performance and time (delay) since both of them are directly proportional to the size. On the other hand, the map of the interleaver plays an important role in setting the code performance. Conventionally, interleaving is used to spread out the errors occurring in burst. For turbo codes, the interleaver has more functions. Interleaving is used to feed the encoders with permutations so that the generated redundancy sequences can be assumed independent. The validity of the assumption that the generated redundancy sequences are independent is a function of the particular interleaver used. This will exclude a number of interleavers, which generate regular sequences such as cyclic shifts [1].

Another key role of the interleaver is to shape the weight distribution of the code, which ultimately controls its performance. This is so because the interleaver will decide which word of the second encoder will be concatenated with the current word of the first encoder, and hence what weight the complete codeword will have [1]. So the aim of the designer is to produce (by manipulating the weights of the second redundancy part through interleaver mapping) whole code words with the overall weights as large as possible [10].

Turbo codes, unlike convolutional codes, make the distribution of the weight more important than the minimum distance [2]. Another issue that is worth considering in the design of the interleaver is the termination of the trellis of both convolutional encoders. By properly designing the map of the interleaver, it is possible to force the two encoders to the all-zero state with only *m* bits (where *m* is the memory length of the convolutional encoder assuming the same convolutional code is used in both encoders). To achieve that a condition on the interleaver is proposed by [6] and demonstrated by [11].

3.4.1 Uniform Interleaver

There have been many attempts to characterize the effects of the interleaver on the performance of turbo codes. To overcome the difficulty of representing the interleaver map or the difficulty of enumerating all the permutations the authors in [2] introduced an abstract interleaver called *uniform interleaver*, defined as follows:

A uniform interleaver of length *k* is a probabilistic device, which maps a given input word of weight *w* into all distinct

$$C^{k}_{w} \qquad (3.8)$$

permutations of it with equal probability

$$1/C^{k}_{w}. \qquad (3.9)$$

The uniform interleaver can not be used in practice since one is confronted with deterministic interleavers. However, it has been shown that for each value of signal to noise ratio, the performance obtained with the uniform interleaver is achievable by at least one deterministic interleaver [2].

The concept of *uniform interleaver* was further used in the design and evaluation of turbo codes. In [Per96] an asymptotic bound on the performance was given as a function of the interleaver length and some other code parameters.

In [2] it was shown through a bound that random interleavers offer performance close to the average ones, independent, to a large extent, of the particular interleaver used. It was also shown that the beneficial effect of increasing the interleaver length tends to decrease at high k (interleaver length). Actually the effect of interleaver length should be considered in conjunction with the memory span of the CCs.

Dolinar and Divsalar [11] compared the difference in performance between random and non-random interleavers. The authors discussed a partial separation of the problem of picking good permutations and that of picking good component codes.

Researchers are still working to develop design guidelines and to relate the interleaver parameters to the code performance. There are different designs of the interleavers:

3.4.2 Different Types of Interleavers:

3.4.2.1 Row-Column Interleaver (Block Interleaver):

Data is written into a block by row and read out by column. Due to its structure, the row-column interleaver is not a good choice for a turbo code as the code appears less random and it is more likely that a low-weight input sequence will get interleaved to produce another low-weight input sequence. [5]

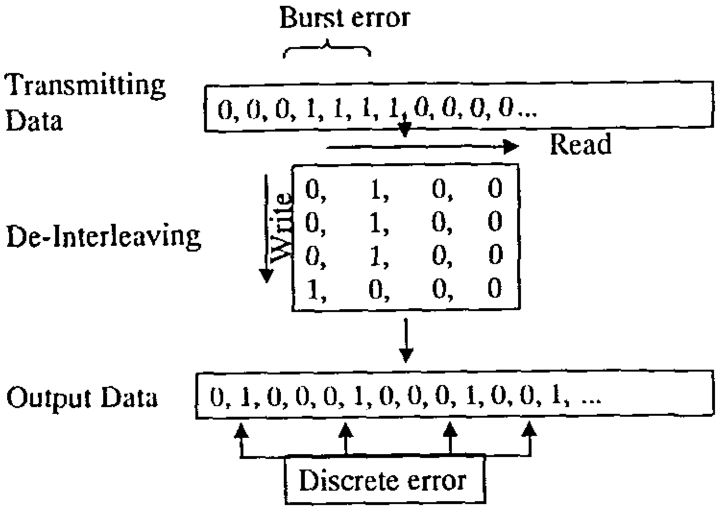


Figure 3.7: Interleaving (Example: Block Interleaver)

3.4.2.2 Pseudo-Random Interleaver:

The pseudo-random interleaver scrambles data in a random fashion. Hall suggests that for an AWGN channel, a pseudo-random interleaver will perform better than a block-interleaver for frame sizes greater than 1000 bits this is in contrast to Andersen who says that a well designed block-interleaver will outperform a pseudorandom interleaver.[5]

3.4.2.3 Helical Interleaver:

A helical interleaver is a block interleaver where data is written in by row and read out diagonally. Hall suggests the use of a Helical Interleaver in situations where the frame size is less than 1000 bits. The helical interleaver has long been used for channel interleaving in convolutional codes to reduce the effect of busty interference. [5]

3.4.2.4 Andersen Interleaver

It was introduced by Jacob Andersen and he does not give it a name. The Andersen approach generally requires large block sizes due to its dependence on prime numbers.The Andersen interleaver is a P by L block interleaver with P and L being prime numbers however the rows are permuted with permutations of the form

$$i \rightarrow a * i \bmod P \qquad (3.10)$$

Here, the a values are Prime and are different for each row. The a values are usually matched to the component code generator by computer search. We are interested in using a less rigorous approach by using Andersen's permutations on an unmatched interleaver with a smaller block size than that of Andersen.

3.4.2.5 Random Interleaver

A *Random* interleaver is one with a randomly generated mapping between input and output positions. That is, for an interleaver of length N, the input sequence i = 0...(N-1). is scrambled according to a pseudo-random number set to form output sequence . (i)= 0...(N-1). T he advantage of random interleavers is their ease of generation, but the disadvantage is that it is not possible to guarantee the minimum spreading properties and therefore BER performance of the interleaver. Random interleavers tend to have very low s-parameters and very high dispersions. In this sense, they are conceptually the 'opposite' of a block interleaver.

3.4.2.6 S-Random Interleaver

The S-Random (Semi-Random) interleaver is a variation on the Random interleaver which is constructed in such a way as to guarantee a high s-parameter, s. The interleaver was first described in [12]. For each successive input position i, an output position . (i) is chosen at random. This value is compared to s, previously selected output positions. If any of these s, positions lies within a distance of s, of the currently selected position, then the current position is rejected. This process is repeated until all N positions in the interleaver are selected. So we have an interleaver which is basically random in structure, but which guarantees a minimum spreading of error pairs. The time taken to find each new suitable position increases disproportionately with increasing s, and the technique may not always find positions which satisfy a particular value of s. It was found in [12] that choosing s<. N/2 usually produces a solution in reasonable time. This practical value of s has been found here to decrease from . N/2 markedly as N increases. A key disadvantage of this interleaver is the time consuming nature of the search method described above for generating good S-Random interleavers.

4.1 Analysis

At a technical level, software engineering begins with a series of modeling tasks that lead to a complete specification of requirements and a comprehensive design representation for the software development. Over the years many modeling methods have been proposed for analysis modeling. However two now dominate the analysis modeling landscape. The first, structured analysis is a classical modeling method and the other approach is object oriented method. We have usedthe former modeling technique for the analysis.

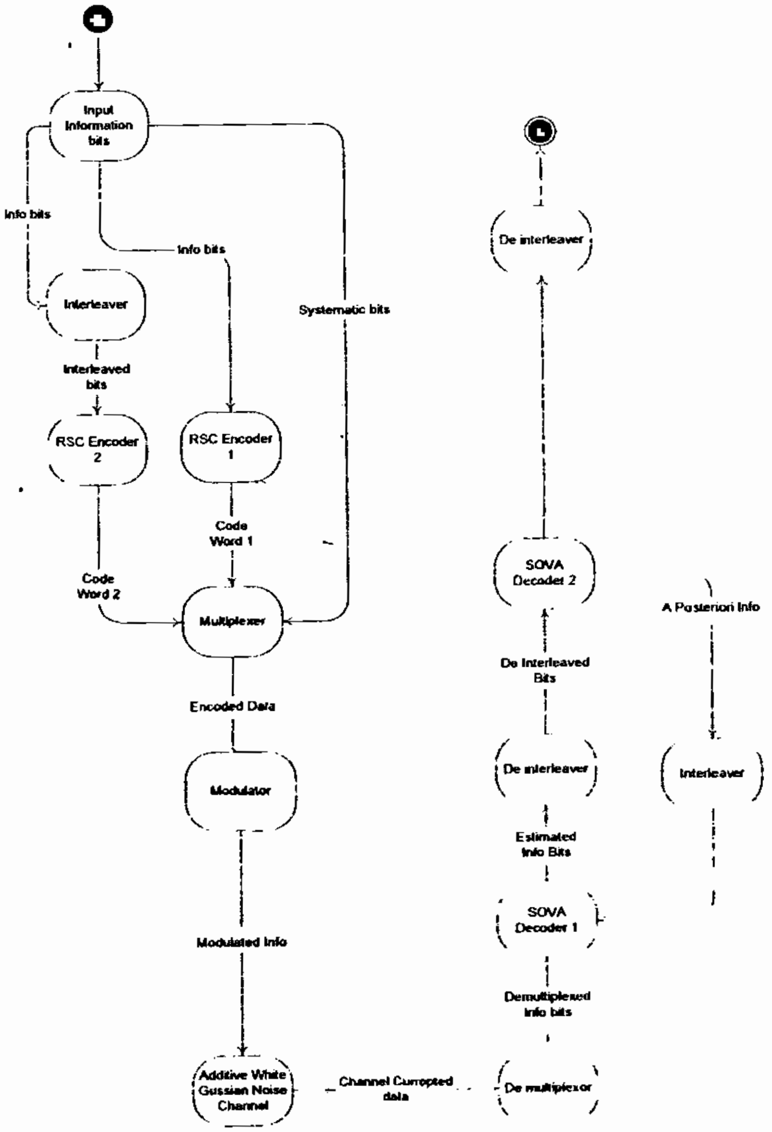
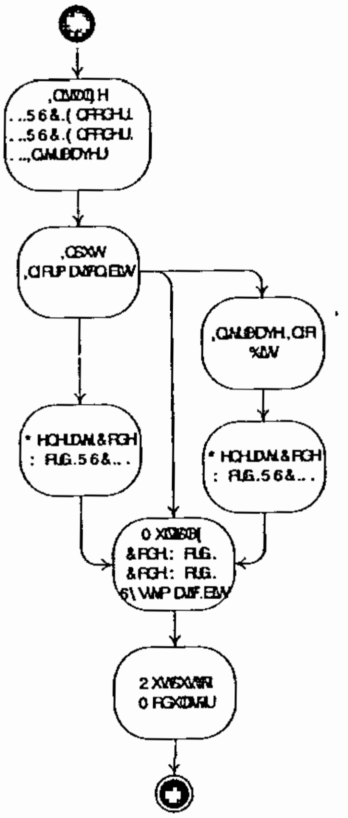


Figure 4.1 Turbo Encoding Decoding System



) LJXUH. ..
7XUER (QFRGLQJ 3 URFHVV

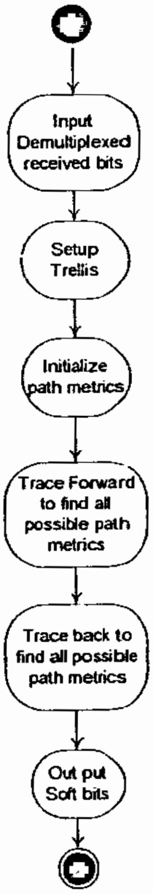


Figure 4.3
SOVA Decodina Process

4.2 Design

During an iterative development cycle it is possible to move to a design phase, once the modeling is complete. During this step a logical solution based upon the structured analysis is developed. The designer's goal is to produce a model or representation of a structure that will later be built. The process by which the model is developed is based on intuition and judgment based on experience in building similar structures., a set of principles and/ or heuristics that guide the way in which the model evolves, a ultimately leads to a final design representation.

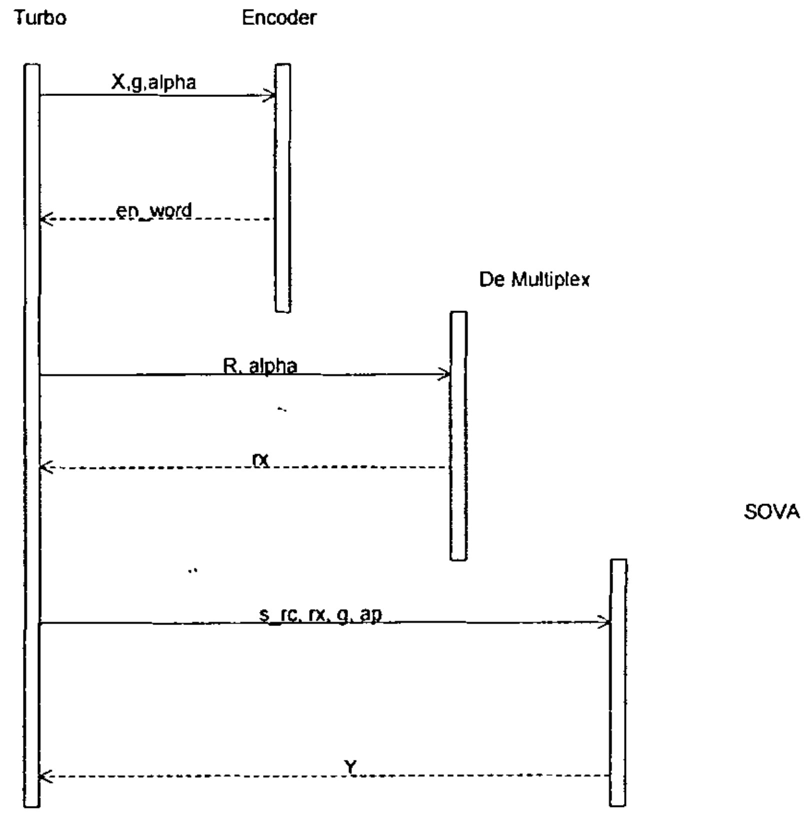


Figure 4.4
Sequence Diagram for Turbo Encoding Decoding Process

Turbo: Main function that takes input from the user calculates and outputs results to the user.

Encoder: This function is called by TURBO. It takes as input, the Generator matrix, raw information bits and Interleaver map. This function uses other sub-functions to accomplish the task. The out put of this function is an n bit encoded and modulated codeword ready to be transmitted. (Noise is added to the info during transmission.)

De-Multiplexer: Next function to be called by TURBO is that of demultiplexing. It is needed to separate the received information bits for both the decoders. The out put of this function is scaled to input to the SOVA decoders.

SOVA: The decoding process is performed using two component SOVA decoders in parallel. The out put of second decoder is used as a priori information for the first decoder. Input for this function is the generator matrix, scaled received bits, the de multiplexed information bits and interleaver map. This function returns the estimated bit values for the original information transmitted.

Comparison of originally transmitted bits and out put bits of SOVA decoder is compared to analyze the "Bit Error Rate".

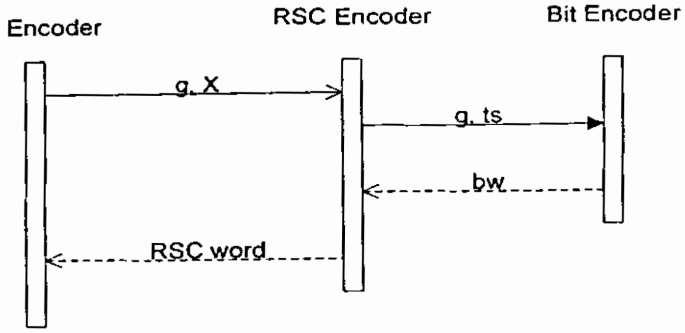
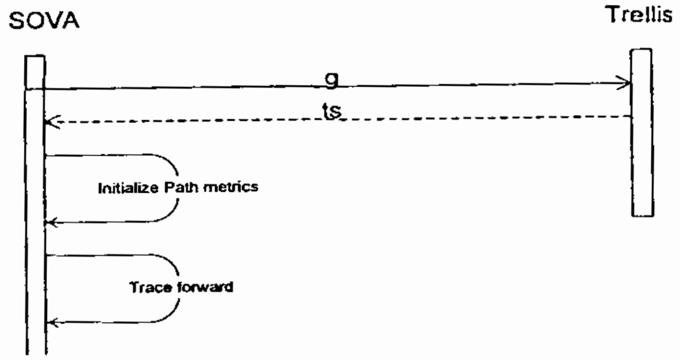


Figure 4.5
Sequence Diagram for Encoding Process



RSC Encoder: The function takes generator matrix and information bits as input and outputs the Convolutional encoded word embedded with systematic bits.

Bit Encoder: Task of this function is to generate n bit code word out put for the k bits of the input. It requires trellis structure and generator matrix as input.

Trellis: This function is called by SOVA to generate Trellis structure by taking generator matrix as input.

Data Exchanged between the processes.

X: For a simulation purpose a randomly generated information bits matrix is used as input to the encoder. This matrix is further compared with the received information bits to analyze the efficiency of the codec.

g: The generator matrix. This is polynomial representation of the convolutional encoder structure.

alpha: Interleaver mapping. It describes the interleaver structure.

en_word: The encoded word returned from the encoder to the main program. The word is not only encoded rather modulated too, to be transmitted over the channel.

bw: The bit encoder function generates a word of length 'n' for each input bit.

RSC_ word: Recursive systematic convolutionally encoded word. The RSC encoder generates the code word and embeds the systematic bits to it.

R: Received information bits from the channel. These information bits have been corrupted by the channel noise.

rx: De-multiplexed information bits at the receiver end. Further these bits are scaled for the input to the decoders.

s_rc: Scaled received bits. The information bits ready to be processed by the SOVA decoder0.

ap: A priori information. The out put of the second decoder, this information is used by the first decoder to decode next coming information bits.

ts : Trellis structure. It is generated by Trellis function and used to generate the trace forward and trace back paths to find maximum likelihood.

5. Performance Analysis of Turbo Codes

The turbo code, as described in Chapters 3, is a very complex channel coding scheme. The turbo code encoder is a parallel concatenation of two recursive systematic convolutional (RSC) codes. The turbo code decoder is an iterative serial concatenation of two soft output Viterbi algorithm (SOVA) decoders. In addition, the presence of interleaver in both the encoder and the decoder further complicates this coding scheme. The two main methods to evaluate the performance of turbo codes are theoretical analysis and computer simulation. Theoretical analysis of a turbo code is very difficult due to the structure of the coding scheme. A few journal papers have analyzed turbo codes with this theoretical approach; however, their results do not match closely (too optimistic) to computer simulation results. Also, the theoretical analyses presented in these papers are not clearly described and thus are difficult to follow. Furthermore, the theoretical approach often requires tremendous computer run time to find the complete weight distribution of the turbo code words. In this thesis, due to the difficulties presented in the theoretical analysis, the performance of turbo codes is evaluated through computer simulation. MATLAB is used to construct the computer code of a turbo code, and the simulations were carried out Pentium Machine.

5.1 Simulation Setup

The simulation setup is composed of three distinct parts, namely the encoder, the channel, and the decoder. The simulation of the turbo code encoder is based on its description in Chapter 3. The simulated 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 random permutation of bit order in a bit stream. This random permutation of bit order is stored so that the interleaved bit stream can be de interleaved at the decoder.

In the literature, the termination aspects of a turbo code are not very well described. At the turbo code encoder, the RSC encoders need to be properly terminated by returning the code memory, of size m, to the all zero state. To perform this task, the systematic code stream is augmented by the addition of m tail bits. These m tail bits cannot be easily predetermined and depend on the code memory. The output of the turbo code encoder is described by three streams, one systematic (un-coded) bit stream and two coded bit streams. The systematic bit stream can only have one set of m tail bits from one of the two recursive encoders. Figure 5.1 shows the two possible scenarios for these m tail bits.

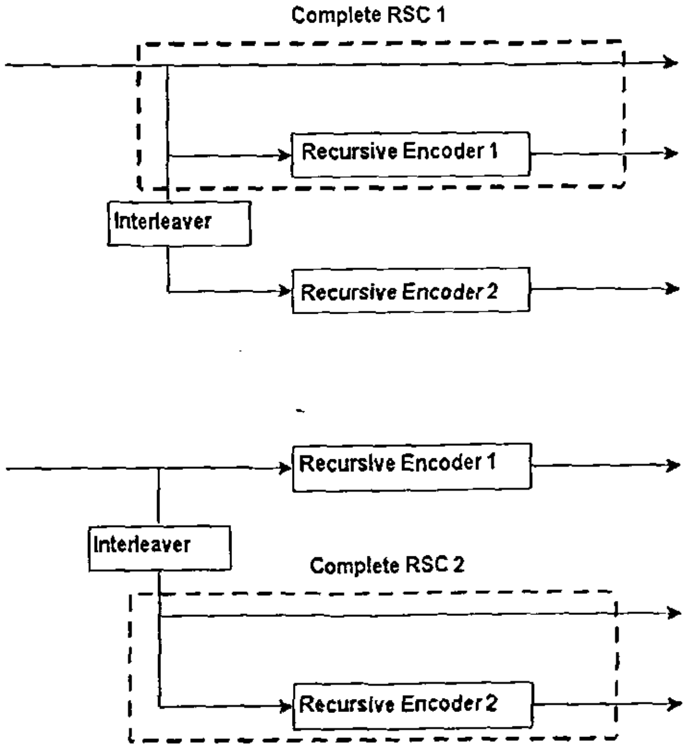


Figure 5.1: Termination of the systematic bit stream is associated with recursive Encoder 1 and Encoder 2.

In the literature, the termination scheme of Figure 5.1 is commonly used. However, the literature also suggests acquisition of the decoded bit decisions at the second component decoder of the turbo code decoder. This is not advisable because the m tail bits for the systematic bit stream are associated with recursive encoder 1. As a result, it is possible that the final path chosen in the second component decoder is not the Maximum Likelihood (ML) path due to the erroneous m tail bits of the systematic bit stream. Thus, if the encoder structure of Figure 5.1 is used, then the decoded bit decisions should be acquired from the first component decoder. Also, if the encoder structure of Figure 5.1 is used, then the decoded bit decisions should be acquired from the second component decoder. In the simulation, the encoder structure of Figure 5.1 is used. In its basic form, the turbo code encoder is rate 1/3. However, in many journal papers, the published computer simulations of turbo codes often use rate 1/2. This is accomplished by puncturing the coded bit streams of the turbo code. The puncturing pattern is that for one coded bit stream, the odd bits are punctured out, and for the other coded bit stream, the even bits are punctured out. In the simulation we have used the basic form.

In the simulation, the Gaussian channel model is used because it is a fairly good model for different transmission mediums. The Gaussian channel (Gaussian noise) is fairly easy to construct from the basic Gaussian distribution with mean of zero and standard deviation of one. In order to use this model, the turbo code encoder output bit streams must be mapped from {0,1} to {-1,+1} domain. The simulation of the turbo code decoder is based on its description in Chapter 3.

If the encoder structure of Figure 5.1 is used, then the systematic bit stream must be de interleaved before passing to the first component decoder of the turbo code decoder. Also, the initial (first iteration) a priori values for the first component decoder are set to zero because there are no extrinsic information available from the second component decoder.

It has been found that the turbo code (SOVA) decoder, implemented as described in the literature, did not perform up to the published results. The cause of this performance degradation has been traced to the extrinsic/a priori information that is passed between the component decoders. After many computer simulation runs, it has been determined that the extrinsic/a priori information must be limited (reduced) before passing to the next component decoder. The two main strategies that were devised to reduce the extrinsic/a priori information are:

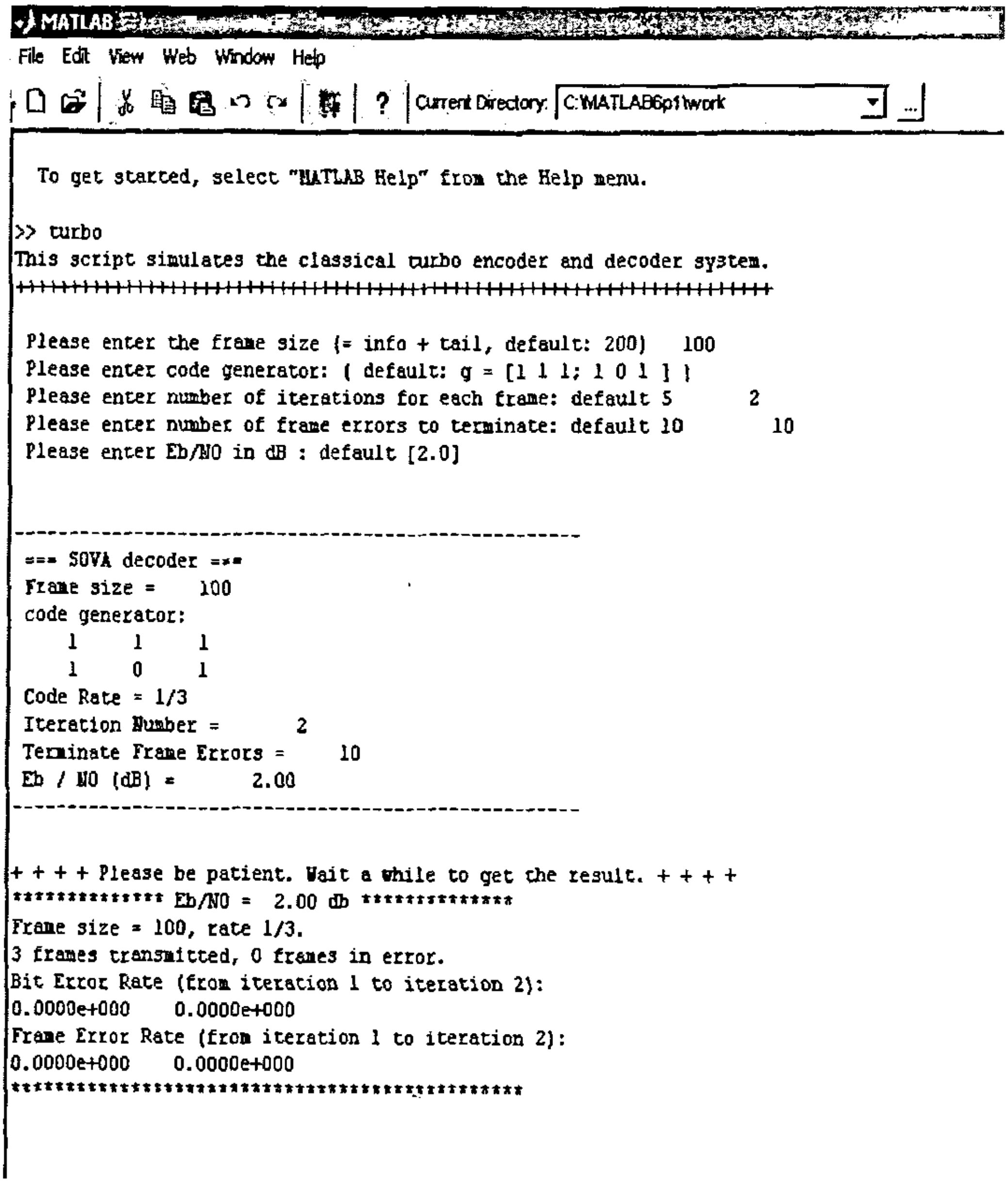
1. Introduce a limiter on the extrinsic/a priori information so that the values do not go over a predetermined bound.

2. Introduce a "numerical factor" to scale down the extrinsic/a priori information after extensive trials and tests, it has been determined that the most effective reduction technique was to introduce a numerical factor Eb/No (signal to noise ratio per bit) to scale down the extrinsic/a priori information. This factor is idealistically reasonable in the sense that for low Eb/No, the extrinsic/a priori values are relatively small so no drastic reduction is required. However, for high Eb/No, the extrinsic/a priori values are relatively large so greater reduction is required. In the simulation, the scale down factor of E_b/N_o is used on the extrinsic/a priori information.

5.2 Simulation

The simulation designed is implemented in Matlab 6.1.0.45 Release 12.1, and run on a pentium III, 450 MHz machine. The figure 4.6 depicts a running of the simulation. The user inputs the frame size to be transmitted, the generator matrix, the number of iterations for each frame, the limit of frame reeor to stop the simulation and finally the E_b/N_o for which the code performance is to be tested. Code rate for the simulation is predefined as 1/3 .

The simulation outputs on each iteration the number of frames transmitted and points out the number of frames with error. It also displays the bit error rate for each iteration and the frame error rate.


MATLAB
File Edit View Web Window Help
Current Directory: C:\MATLAB6p1\work

```
 To get started, select "MATLAB Help" from the Help menu.

>> turbo
This script simulates the classical turbo encoder and decoder system.
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

 Please enter the frame size (= info + tail, default: 200)   100
 Please enter code generator: ( default: g = [1 1 1; 1 0 1 ] )
 Please enter number of iterations for each frame: default 5      2
 Please enter number of frame errors to terminate: default 10       10
 Please enter Eb/N0 in dB : default [2.0]



----------------------------------------------------
 === SOVA decoder ===
 Frame size =    100
 code generator:
     1     1     1
     1     0     1
 Code Rate = 1/3
 Iteration Number =       2
 Terminate Frame Errors =     10
 Eb / N0 (dB) =       2.00
----------------------------------------------------

+ + + + Please be patient. Wait a while to get the result. + + + +
************** Eb/N0 =  2.00 db **************
Frame size = 100, rate 1/3.
3 frames transmitted, 0 frames in error.
Bit Error Rate (from iteration 1 to iteration 2):
0.0000e+000    0.0000e+000
Frame Error Rate (from iteration 1 to iteration 2):
0.0000e+000    0.0000e+000
**********************************************
```

Figure 5.2 MATLAB Simulation

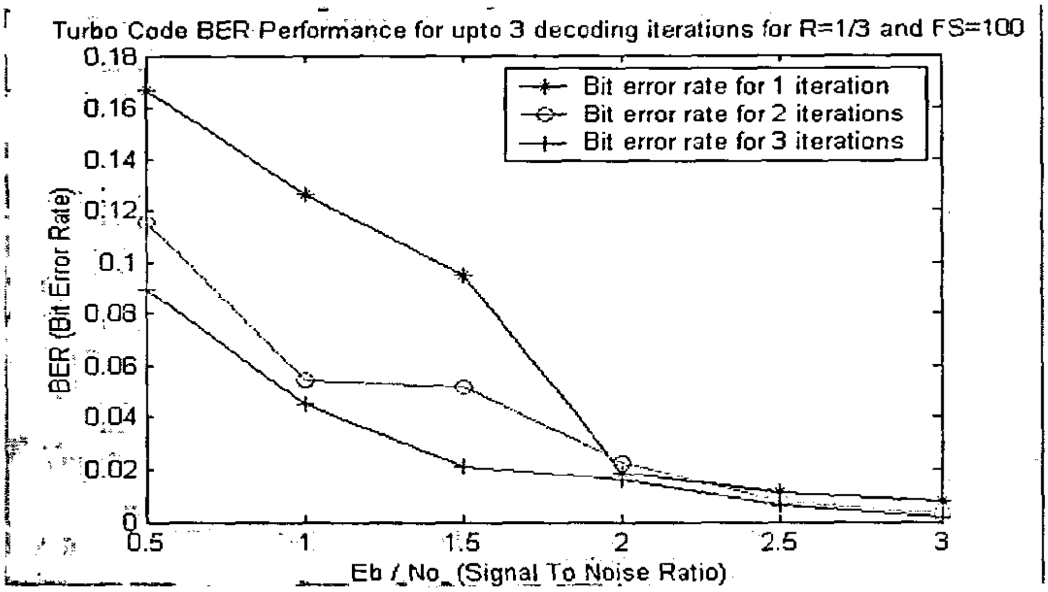
5.3 Simulation Results

$E_b/$ N_0	Average BER			Total Frames Transmitted		
	Iteration 1	Iteration 2	Iteration 3	Iteration 1	Iteration 2	Iteration 3
0.5	0.1667	0.1156	0.0899	3	3	7
0.1	0.1259	0.0544	0.0452	3	3	10
1.5	0.0952	0.0520	0.0214	3	10	18
2.0	0.0187	0.0230	0.0159	6	14	32
2.5	0.0115	0.0078	0.0061	8	28	69
3.0	0.0081	0.0033	0.0019	29	87	256

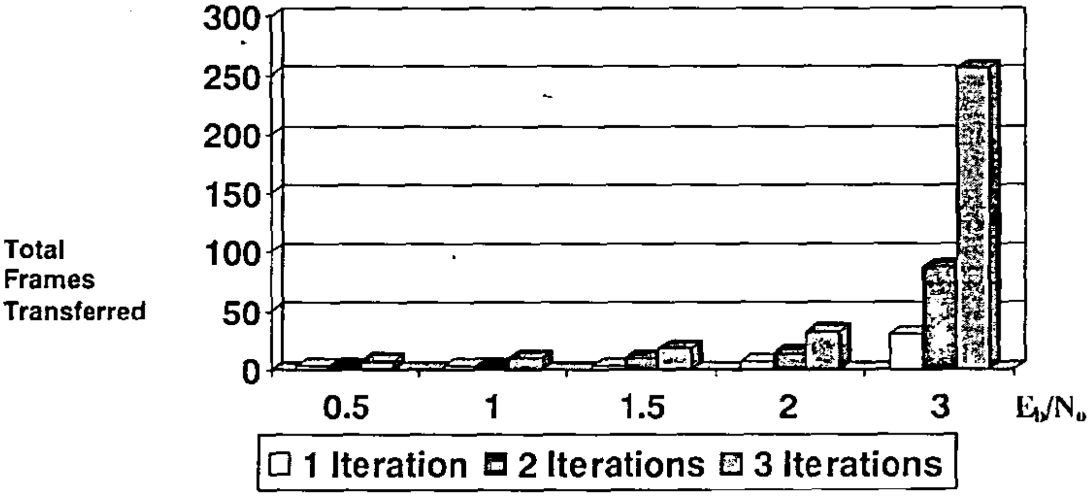
Table 5.1: E_b/N_0 vs. Average BER

The above table shows the results of a Rate 1/3 Turbo codec simulation with the frame size of 100 bits for 1, 2 and 3 iterations. The data between signal to Noise ratio (E_b/N_o) and Bit Error Rate (BER) clearly explains efficiency of turbo codec for wireless communication. As we increase the number of iterations and E_b/N_o the performance of the turbo code increases. On the basis of above data we can create following graphs.

5.3.1 E_b/N_0 vs. Average BER



5.3.2 E_b/N_0 vs. Total number of frames transmitted



5.6 Conclusions

With the detailed description of the turbo code encoder and decoder presented in Chapter 3 .This Chapter investigates the performance of turbo code through extensive computer simulation. To validate the turbo code simulation, comparisons were made between the simulated and published bit error rate (BER) results. These differences are believed to be caused by two independent factors, namely, the numerical inaccuracies introduced by the workstations and the lack of "critical" details about the SOVA decoding algorithm. The BER performance for turbo codes is investigated for two cases.

1. Increase number of Iterations.
2. Increase of E_b/N_0 value.

As it is known by now, turbo code decoding can become computationally intensive. As a result, most of the simulated performance results are for high code rates, short constraint lengths, and small frame sizes.

5.7 Future Work

There are many directions in the future for research in turbo codes. As shown in this thesis, some detailed work needs to be done on the aspect of information transfer between the SOVA component decoders. Presently, this issue is not very well understood. As for further improvements in turbo code, research should be focused on the joint issues of improving decoder performance and reducing decoder complexity. Also, developing simple analytical bounds for the performance of SOVA decoding is important. Furthermore, for feasibility concerns, issues involved in DSP implementation of the SOVA turbo code decoder will be important when turbo code are implemented in real systems.

Understanding Decoding Algorithm SOVA

Syed Khaldoon Khurshid, Muhammad Imran Herl and Dr. Muhammad Sher

***Abstract*— The correct transmission of the data has been a main concern for researchers since the advent of the communication era. There are two main errors correcting codes i.e. Linear Block codes and Convolutional codes. Another version of Error correcting Codes i.e. Turbo codes were introduced in 1993. Shannon limit was not achieved by the convolutional codes but Turbo Codes have succeeded by achieving the signal to noise ratio (E_b / N_o) near -1.6db. This paper describes two main components of the Turbo codes namely: Its Encoder and Decoder. Amongst the encoders for turbo codes, Recursive Systematic Convolutional (RSC) Encoder is exclusively better than the Non-Recursive and Non-Systematic ones .While the decoding algorithm for Turbo codes, SOVA (Soft Output Viterbi Algorithm) outperforms its counterparts like MAP in terms of complexity and Time Delay. Working of the SOVA (Soft Output Viterbi Algorithm), decoding algorithm, is explained in a sequential manner. At the end, it investigates the performance and efficiency of Turbo codes with respect to the increasing number, of iterations and frames transmitted, through simulation results.**

***Index Terms*—Convolutional Codes, Recursive Systematic Convolutional (RSC) Encoder, SOVA (Soft Output Viterbi Algorithm), Turbo Codes.**

I. Introduction

Over the years, there has been tremendous growth in digital communications especially in the fields of cellular phones, satellite, and computer communication. In these communication systems, the information is represented as a sequence of binary digits. The binary digits are then mapped (modulated) to analog signal waveforms and transmitted over a communication channel. The communication channel introduces noise and interference to corrupt the transmitted signal. This signal is mapped back to binary digits at the receiver. The received binary information is an estimate of the transmitted binary information. [7]

In a noisy environment, it is often not possible to reduce the bit error rate to acceptable levels. Doing so may require rising the signal power beyond practical limits. Alternatively, lowering the error rate might require communicating at an unacceptably slow rate.

However there is another available option to improve the performance of digital communication system: *Error control coding* can be used to provide a structure for error tolerant communication. The structure is such that errors can be recognized at the receiver. The error control is accomplished in two steps: detection and correction. Error detection is the process of providing enough structure so that the receiver knows when the error occurs. If the added structure is sufficiently detailed to allow pinpoint the location of these errors, the code is an *error correcting code*, and it is possible to correct errors at the receiver without requesting additional information from the transmitter (e.g., a retransmit request). This process is known as *Forward Error Correction*. Forward error correction normally requires adding the redundancy to the signal; more bits are sent than required. [1] There are two main codes, i.e. Linear Block and Convolutional codes, before Turbo codes which are being in error free communications.

The concept of turbo code was first introduced by C. Berrou in 1993. Today, Turbo Codes are considered as the most efficient coding schemes for FEC. When is compared to other codes and almost achieving the Shannon Limit (E_b/N_0 = -1.6 db if $R_b \rightarrow 0$) at modest complexity [10].

Turbo codes have been proposed for low-power applications such as deep-space and satellite communications, as well as for interference limited applications such as third generation cellular, personal communication services, and ad-hoc and sensor networks.

II. Turbo Codes

Lately in [4], the proposal of Parallel-Concatenated Convolutional Codes *(PCCC)*, called turbo codes, was introduced. The introduction of turbo codes has increased the interest in the coding area since these codes give most of the gain promised by the channel-coding theorem. Turbo codes have an astonishing performance of bit error rate (BER) a relatively low signal to noise ratio (E_b/N_0). To give an idea o how powerful turbo codes are, for a frame size o 256×256=65536 bits we can achieve a BER=10^{-5} over AWGN channel at only E_b/N_0=-0.7dB [4].

In the following two sections the structure of both encoder an the decoder of turbo codes are explained

Manuscript received July 1, 2004. This Paper is conducted as a part of MS degree from International Islamic University Islamabad.

Syed Khaldoon Khurshid is student of MS in International Islamic University. (Mobile Phone: 0300-951-2328 and Home number: 0541-9270139; E-mail: skhaldoon@ hotmail.com).

Muhammad Imran Herl is student of MS in International Islamic University. (E-mail: imranherl@isb.pol.com.pk).

Dr. Muhammad Sher is currently Head of the Computer Engineering Department, International Islamic University, Sector H-10, Islamabad. He will persuade his second Ph.D. in Germany (e-mail: msher313@iiu.edu.pk).

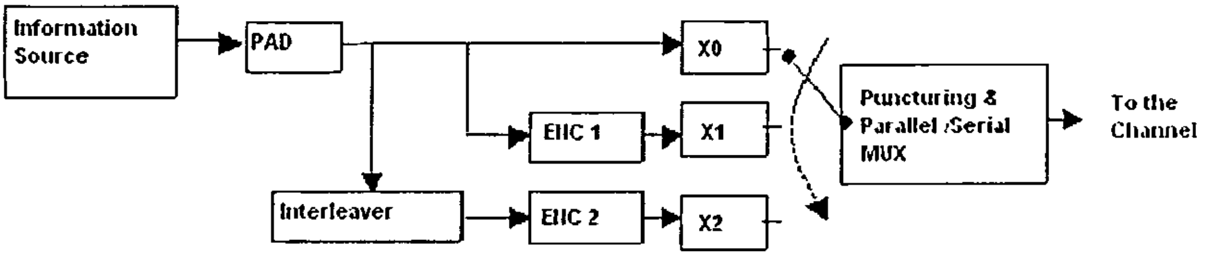


Fig. 1: Simplified Turbo Encoder

III. Turbo Encoder

In a simplified turbo encoder, there are two convolutional encoders in parallel. The information bits are scrambled before entering the second encoder. Input bits are appended to the convoluted output - i.e. the code is systematic-followed by the parity check bits from the first encoder and then the parity bits from the second encoder, as depicted in Fig. 1.

The simplified turbo code block diagram shows only two branches. In general, one can have multiple turbo encoders with more than two branches. The Convolutional code at every branch is called the *constituent code (CC)*. The CCs can have similar or different generator functions. We will discuss the usual configuration with two branches having the same CC. A PAD is shown in the Figure to append the proper sequence of bits to terminate all the encoders to the all-zero state. This is because a convolutional code may be used to generate a block code if we use beginning and tail bits. If we have one then the required tail is a sequence of zeros with length equal to the memory order m. The problem of terminating both encoders simultaneously seems to be difficult because of the interleaver. However, it is still possible to do with m tail bits only [6].

In general we can have another interleaver before the first encoder but usually it is replaced with a delay line to account for the interleaver delay and keep the branches working simultaneously. Puncturing can be introduced to increase the rate of the convolutional code beyond that resulting from the basic structure of the encoder. Some codes are called recursive since the state of the internal shift register depends on the past outputs. Fig. 2 illustrates a non-recursive and non-systematic convolutional code with its corresponding recursive systematic code. X_0 and X_1 are the check bits. Note that for the systematic Convolutional encoder, one of the outputs, X_0, is exactly the input sequence. In turbo codes Recursive Systematic Convolutional (RSC) codes are proved to perform better than the Non-recursive ones [4] [3].

An RSC encoder can be obtained from a non-systematic & non-recursive encoder by setting one of the outputs equal to the input (if we have one input) and using a feed back

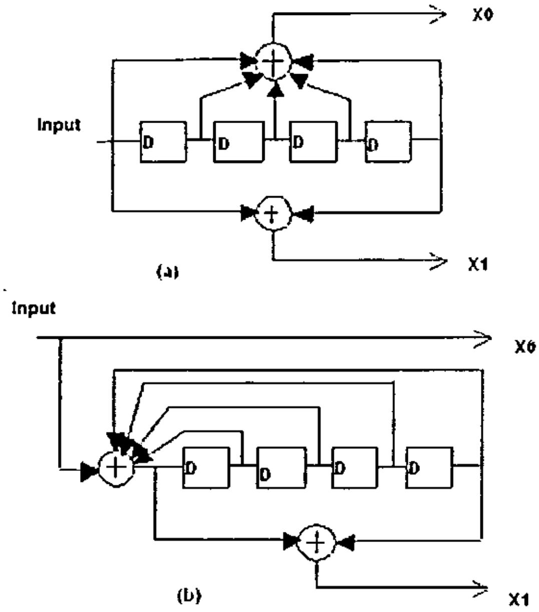


Fig. 2: (a) Classical Non-Recursive and Non-Systemic Code (b) Recursive Systemic Code

. The trellis and the free distance (d_{free}) also will be the same for both codes [4].Of course; the output sequence does not correspond to the same input in the two codes because the two generators are not the same.

IV. Turbo Decoder

The decoder works in an iterative way. Fig. 3 shows a block diagram of a turbo decoder. The iteration stage is shown with doted lines to differentiate it from the initialization stage. Only one loop is performed at a time. In practice the number of iterations does not exceed 18, and in many cases 6 iterations can provide satisfactory performance [4].

Actually, the term turbo code is given for this iterative decoder scheme with reference to the turbo engine principle. The first decoder will decode the sequence and then pass the hard decision together with a reliability estimate of this decision to the next decoder. Now, the second decoder will have extra information for the decoding; a priori value together with the sequence.

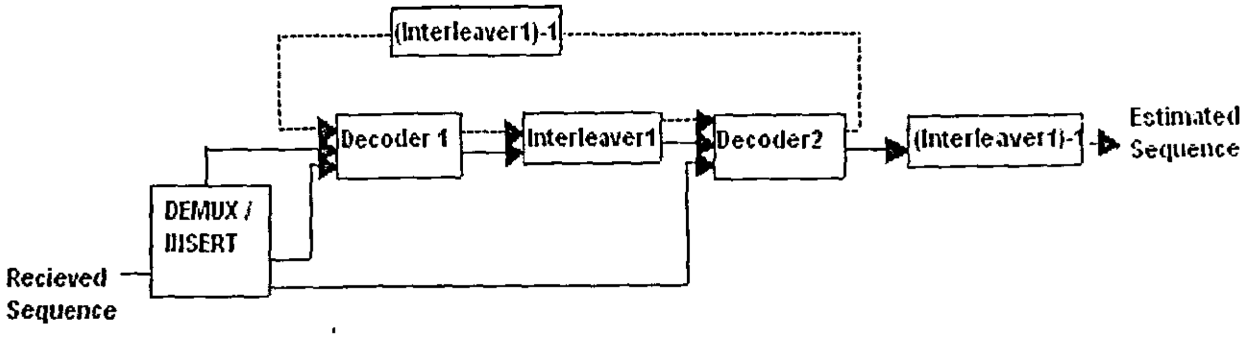


Fig. 3: Block Diagram of Turbo Codes

The interleaver in-between is responsible for making the two decisions uncorrelated and the channel between the two decoders will seem to be memory less due to interleaving. The exact procedure in what information is passed to the next decoder or next iteration stage is a subject of research. In the next section, we describe a widely accepted decoding algorithm, which is the modified version of Viterbi algorithm. [2]

V. Decoding Algorithm: SOVA

Algorithms used in decoding convolutional codes can be modified to be used in decoding turbo code. In the original paper on turbo codes [4], a modified BAHL et algorithm was proposed for the decoding stage. This algorithm is based on Maximum A-posteriori Probability (MAP). The problem with this algorithm is the inherent complexity and time delay. MAP algorithm was originally developed to minimize the bit-error probability instead of the sequence error probability. The algorithm, although optimal, seem less attractive due to the increased complexity.

Viterbi algorithm is an optimal decoding method that minimizes the probability of sequence error for convolutional codes. A modified version of Viterbi algorithm, called SOVA (Soft Output Viterbi Algorithm), which uses soft outputs, is introduced in [8] [9].

A. Understanding SOVA

It is proposed that an iterative decoding scheme should be used. The decoding algorithm is similar to Viterbi algorithm in the sense that it produces soft outputs. While the Viterbi algorithm outputs either 0 or 1 for each estimated bit, the turbo code decoding algorithm outputs a continuous value of each bit estimate. While the goal of the Viterbi decoder is to minimize the code word error by finding a maximum likelihood estimate of transmitted code word, the soft output decoding attempts to minimize bit error by estimating the posterior probabilities of individual bits of the code word. We called the decoding algorithm Software Decision Vitérbi Decoding. The turbo decoder consists of *M* elementary decoders - one for each encoder in turbo encoding part. Each elementary decoder uses the Software Decision Viterbi Decoding to produce a software decision for each received bit. After an iteration of the decoding process, every elementary decoder shares its soft decision output with the other *M*-1 elementary decoders.

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

The turbo-code decoder is described in the fig. 6. Assuming zero decoder delay in the turbo-decoder, the decoder 1 computes a soft-output from the systematic data (x_0), code information of encoder 1 (y_1) and a-priori information (La_2).

From this output the systematic data (x_0) and a-priori information (La_2) are subtracted. The result is multiplied by the scaling factor called channel reliability L_c to compensate the distortion. The result is uncorrelated with x_k and is denoted as Le_1, for extrinsic data from decoder 1.

Decoder 2 takes as input the interleaved version of Le_1 (the a-priori information La_1), the code information of second encoder (y_2) and the interleaved version of systematic data ($\alpha(x_k)$). Decoder 2 generates a soft output, from which the systematic data ($L_c\alpha(x_0)$) and a-priori information (La_1) was subtracted. The result is multiplied by the scaling factor called channel reliability L_c to compensate the distortion. The extrinsic data from decoder 2 (Le_2) is interleaved to produce La_2, which is fed back to decoder1 and the iterative process continues.

B. Steps of SOVA

1) Form the Trellis

Inside the decoder, Soft-Output Viterbi Algorithm (SOVA) is used to determine the result with maximum likelihood. The process SOVA is similar to that Viterbi algorithm. A trellis is formed first.

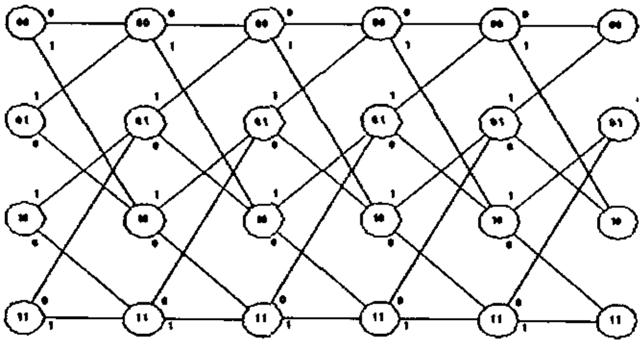


Fig. 4: Trellis Diagram

The trellis is to show how the codes are output by encoder. The bits in each node the encoder output (decoder input) for all combination of states and inputs of the encoders are summarized as follows:

State 1	State 2	Input	Output	Next State 1	Next State 2	Decoder Input
0	0	0	0	0	0	00
0	0	1	1	1	0	11
0	1	0	0	1	0	00
0	1	1	1	0	0	11
1	0	0	1	1	1	01
1	0	1	0	0	1	10
1	1	0 ·	1	0	1	01
1	1	1	0	1	1	10

TABLE 1: TURBO ENCODER WORKING

2) *Determine the Accumulated Maximum Likelihood for Each State*

From the Table 1, each state has its own input bit pattern. When bit streams are inputted to decoder, they can be compared to the input ($x_0\ x_1$) to see whether they are matched. The result is represented by likelihood of input:

L_i = -1 If no bits are matched.
...0 if 1 bit is matched.
...1 if both bits are matched.

The overall likelihood of a transition is sum of likelihood of input and a-prior likelihood information (L_p).

$L = L_i + L_p$ (1)
L_p = 0.5 x a-prior information L_{un} if x_0 are 1
... -0.5 x a-prior information L_{an} if x_0 are 0

The algorithm is as follows:

Start from state 00, the overall likelihood of each transition is evaluated. The overall likelihood of each node is obtained by the maximum accumulated likelihood. With this algorithm, the trellis in figure 5 will be obtained.

3) *Find the Surviving Path*

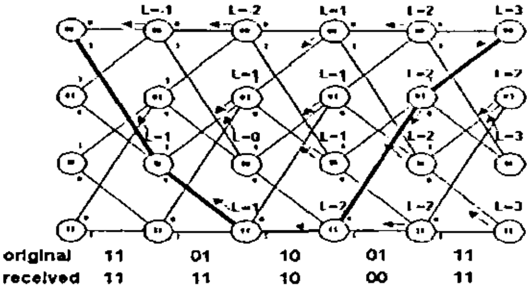


Fig.5: Surviving Path in Trellis Diagram

The surviving path is thus derived by tracing back from last stage (stage 5) to the first one (stage 0) and is the results of hard-output Viterbi Algorithm.

Let u= [1 0 1 0 1]

4) *Determine the Soft Output*

To find the soft-output, non-surviving paths are considered. We consider the non-surviving path the one that is produced by making different decision in one of the stage in tracing back. For example, when tracing back from (last stage) stage 5, one of the non-surviving path is found by tracing back to state 00 instead of state 01 from stage 5 to stage 4. Following the arrows, bit 1 is also 1. We found that bit 1 will have the following results when decision is changed in different stages:

Stage	1	2	3	4	5	
Bit 1	X	X	0	1	1	where X means cannot trace back to state 00 in stage 0
Delta	-	-	3	1	2	

TABLE 2: NON-SURVIVING PATHS ARE CONSIDERED

Here we define a function delta to describe the tendency to have non-surviving path. It is the difference in overall likelihood when a different decision is possible in a particular stage. The soft-output of bit 1 is evaluated by the following formula:

soft-output of bit 1 = bit value x *min* (2)
(*delta* which make bit 1 change to value other than *bit value*)
bit value = 1 for bit output = 1
...-1 for bit output = 0

In bit one, the decision only changes when a state changes in stage 3. The minimum delta is 3. Thus the soft-output of the bit one is 3.

Repeat it for bit two (originally bit two = 0),

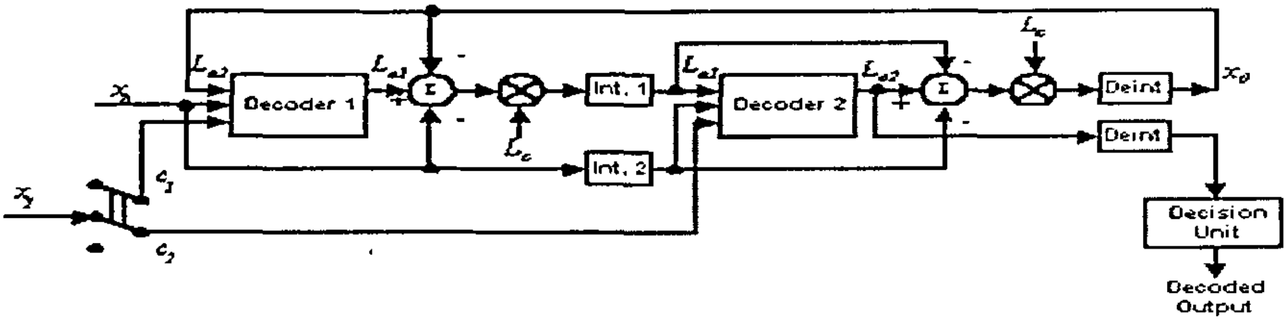


Fig. 6: Turbo decoder structure

Stage	2	3	4	5	
Bit 1	X	1	1	1	where X means cannot trace back to state 00 in stage 0
Delta	-	3	1	2	

TABLE 3: NON-SURVIVING PATHS ARE CONSIDERED

From the above results, in bit two, the decision changes when a state changes in stage 3, 4 and 5. The minimum delta is 1. Thus the soft-output of the bit two is -1.Using this algorithm, the soft-output will become

[3 -1 1 -1 2]

1) *Feeding Data to another Decoder*

After the soft-output is evaluated by SOVA decoder, the data will be passed to the second decoder for further decoding. Before passing to data to the second decoder, two processes are performed to the decoder output:

D_1	3	-1	1	-1	2
La_2	0	0	0	0	0
X_0	1	1	1	-1	1
Le_1	2	-2	0	0	1

TABLE 4: THE A-PRIOR INFORMATION (LA_2) AND THE SYSTEMATIC DATA (X_0) ARE SUBTRACTED

The result is multiplied by the scaling factor called channel reliability L_c. The reason of the factor is because the SOVA algorithm suffers a major distortion which is caused by over-optimistic soft outputs. The factor is used to compensate this distortion.

$$L_c = \text{mean}\,(Le_1) \times 2 \,/\, \text{var}\,(Le_1) \qquad (3)$$

2) *Iterative Decoding*

The data from first decoder L_cL_{e1}, together with the systematic data α (x_k) and code information from second encoder (y_2), are then fed into the second decoder for decoding; the decoding algorithm is the same as the first one. After decoding, the output of second decoder is processed in the same way and fed back to the first decoder. The process continues.

The number of iterations depends on the designer. Usually, the larger the iteration, the more accurate the data but the longer the time it takes for decoding.

3) *Decision of Output*

After iterations of decoding, the decoding results are the sign of the soft-output of the last decoder. Take the example, for the results of first decoder, the output become:

decoder output	3	-1	1	-1	2
Result	1	0	1	0	1

Table 5: The decoder outputs

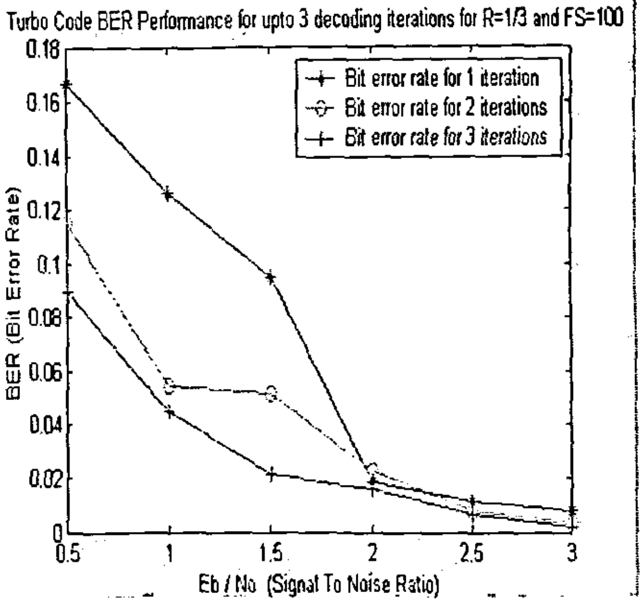
Which is the same as the input bit stream *u*. i.e., the error can be recovered. SOVA has only twice the complexity of Viterbi algorithm. The new algorithm will make it possible to integrate both the encoder and the decoder in a single silicon chip with unmatched performance at the present time [4].

VI. SIMULATION'S RESULTS

E_b/N_0	Average BER			Total Frames Transmitted		
	Iteration 1	Iteration 2	Iteration 3	Iteration 1	Iteration 2	Iteration 3
0.5	0.1667	0.1156	0.0899	3	3	7
1.0	0.1259	0.0544	0.0452	3	3	10
1.5	0.0952	0.0520	0.0214	3	10	18
2.0	0.0187	0.0230	0.0159	6	14	32
2.5	0.0115	0.0078	0.0061	8	28	69
3.0	0.0081	0.0033	0.0019	29	87	256

TABLE 6: THE SIMULATION GENERATED DATA

The above table shows the results of a Rate 1/3 Turbo codec simulation with the frame size of 100 bits for 1, 2 and 3 iterations. The communication is terminated when three bit error occurs. The following graph is produced on the basis of above generated data.



Graph 1: Average BER vs. E_b/N_0

III. Conclusion:

Two conclusions are drawn from above provided simulation results:

1) As the number of iterations increases, the performance of turbo codes also increases.

2) At higher value of signal to noise ratio (E_b/N_0), efficiency of the turbo codes increases.

The graph between signal to Noise ratio (E_b/N_o) and Bit Error Rate (BER) clearly explains efficiency of turbo codec for wireless communication. When we increase the number of iterations or E_b/N_o the performance of the turbo code increases.

Furthermore, The excellent performance of turbo codes requires careful understanding of the code ingredients. The effects of the code parameters on the performance need further investigation. Proper understanding of the code will help in making the code applicable for real time applications

References

[1] Battail, Gerard, "A Conceptual framework for understanding turbo codes," IEEE Journal on selected Areas in Communications vol.1 No. 2,February 1998 pp 245-254

[2] "*Understanding Turbo Codes*", *Ali .H.* Mugaibel and Maan A.Kusa, King Fahd University of Petroleum and minearnals, is Published.

[3] Benedetto, Sergio and Montorsi, Guido "*Design of parallel concatenated* convolutional codes", IEEE Transactions on the communications, vol. 44, No 5, May 1996, pp.591-600.

[4] Berrou, C., Glavieux, A. and Thitimajhima, P "Near Shannon limits error correcting coding and decoding: tubo-codes," Proc. Of ICC '93, Geneva, May 1993, pp. 1064-1070

[5] The working of the SOVA is explained (which is available at URL:www.ie.cuhk.edu.hk/~chankm6/TurboCode/decoding)

[6] Blackert, W.j.Hall E.K., and Wilson, S.G. "Turbo code termination and interleaver conditions" Electronic Letters vol. 31, No 24 November 1995, pp 1182-1184

[7] Analog to Digital Communications, Error Control Coding, Chapter 9 *Channel Encoding*, Page 393

[8] Hagenauer, J.,Hocher, P., "A Viterbi algorithm with soft-decision outputs and its *applications*," *Proceedings* of GLOBECOM '89 Dallas, Texas, Nov. 1989, pp. 47.11-47.17.- codes," Proceedings of 1995 International Symposium on Information Theory, 17-22 Sep. 1995.p.38.

[9] [9].Hagenauer, J..and Papke, L., "Decoding turbo codes with the soft-output Viterbi algorithm (SOVA)," in Proceedings of International Symposium On Information Theory(Trondheim, Norway, June 1994), p. 164.

[10] R. W. Hamming "Error detecting and correcting codes" in Bell Systems Technical Journal, Vol. XXVI, 1950

Books

Title: "Application of Error-Control Coding" IEEE Trans. Information Theory,
Name: D.J. Costello, J. Hagernauer, Hideki Imai, Stephen B. Wicher
Edition: Vol.44, No. 6, Oct. 1998.

Title: "Modern Digital and Analog Communication Systems"
Name: B.P. Lathi
Edition 1983

Title: "Error Control Coding, Fundamental and Application"
Name: Shu Lin, Daniel J. Lostello, Jr
Edition: Prentice Hall Series 1983

Title : Wireless communications and networks
Name : Willam Stallings
Edition: 1st Edition

References

[1].Battail, Gérard, "A conceptual framework for understanding turbo codes," IEEE Journal on Selected Areas in Communications, vol. 16, No. 2,February 1998, pp. 245-254.

[2] Benedetto, Sergio and Montorsi., Guido "Unveilig turbo-codes: some results on parallel concatenated coding schemes," IEEE Transactions on Information Theory, vol. 42, No. 2, March 1996, pp. 409-428.

[3] Benedetto, Sergio and Montorsi., Guido "Design of parallel concatenated convolutional codes," IEEE Transactions on Communications, vol. 44, No. 5, May 1996, pp. 591-600.

[4] Berrou, C., Glavieux, A. and Thitimajhima, P "Near Shannon limits error correcting coding and decoding: turbo-codes," Proc. Of ICC '93, Geneva, May 1993, pp. 1064-1070

[5]murray.newcastle.edu.au/users/students/1100/c9706299/interleaver

[20] [Div95b] Divsalar, D. and Pollara, F., "Turbo Codes for PCS Applications," *Proceedings of ICC 1995*, Seattle, WA., pp. 54-59, June 1995.

[21] [Dol95] Dolinar, S. and Divsalar, D., "Weight Distributions for Turbo Codes Using Random and Nonrandom Permutations," *JPL TDA Progress Report 42- 122*, Aug. 15, 1995.

[22] [Hag89] Hagenauer, J. and Hoeher, P., "A Viterbi Algorithm with Soft-Decision Outputs and Its Applications," *GLOBECOM 1989,* Dallas, Texas, pp.1680-1686, Nov. 1989.

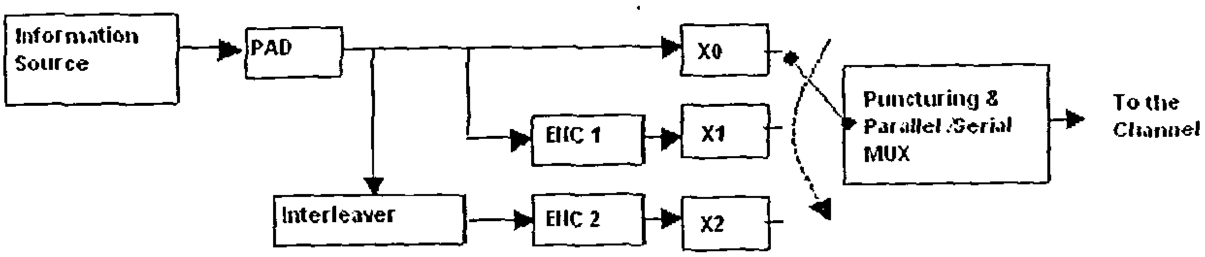


Fig. 1: Simplified Turbo Encoder

III. TURBO ENCODER

In a simplified turbo encoder, there are two convolutional encoders in parallel. The information bits are scrambled before entering the second encoder. Input bits are appended to the convoluted output - i.e. the code is systematic-followed by the parity check bits from the first encoder and then the parity bits from the second encoder, as depicted in Fig. 1.

The simplified turbo code block diagram shows only two branches. In general, one can have multiple turbo encoders with more than two branches. The Convolutional code at every branch is called the *constituent code (CC)*. The CCs can have similar or different generator functions. We will discuss the usual configuration with two branches having the same CC. A PAD is shown in the Figure to append the proper sequence of bits to terminate all the encoders to the all-zero state. This is because a convolutional code may be used to generate a block code if we use beginning and tail bits. If we have one then the required tail is a sequence of zeros with length equal to the memory order m. The problem of terminating both encoders simultaneously seems to be difficult because of the interleaver. However, it is still possible to do with m tail bits only [6].

In general we can have another interleaver before the first encoder but usually it is replaced with a delay line to account for the interleaver delay and keep the branches working simultaneously. Puncturing can be introduced to increase the rate of the convolutional code beyond that resulting from the basic structure of the encoder. Some codes are called recursive since the state of the internal shift register depends on the past outputs. Fig. 2 illustrates a non-recursive and non- systematic convolutional code with its corresponding recursive systematic code. X_0 and X_1 are the check bits. Note that for the systematic Convolutional encoder, one of the outputs, X_0, is exactly the *input sequence.* In turbo codes Recursive Systematic Convolutional (RSC) codes are proved to perform better than the Non-recursive ones [4] [3].

An RSC encoder can be obtained from a non-systematic & non-recursive encoder by setting one of the outputs equal to the input (if we have one input) and using a feed back

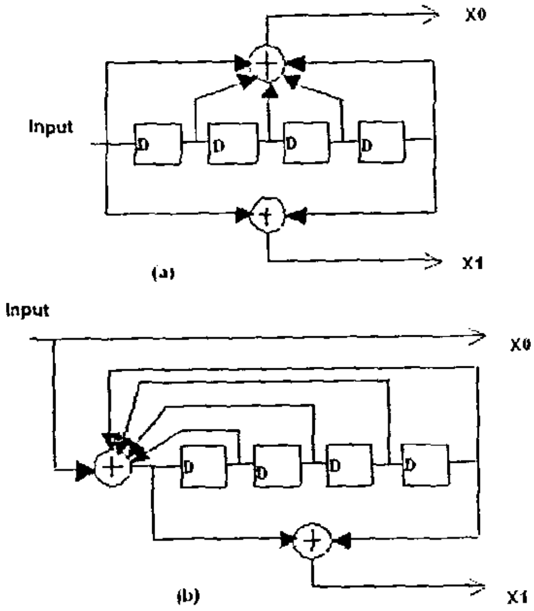


Fig. 2: (a) Classical Non-Recursive and Non-Systemic Code (b) Recursive Systemic Code

. The trellis and the free distance (d_{free}) also will be the same for both codes [4].Of course; the output sequence does not correspond to the same input in the two codes because the two generators are not the same.

IV. TURBO DECODER

The decoder works in an iterative way. Fig. 3 shows a block diagram of a turbo decoder. The iteration stage is shown with doted lines to differentiate it from the initialization stage. Only one loop is performed at a time. In practice the number of iterations does not exceed 18, and in many cases 6 iterations can provide satisfactory performance [4].

Actually, the term turbo code is given for this iterative decoder scheme with reference to the turbo engine principle. The first decoder will decode the sequence and then pass the hard decision together with a reliability estimate of this decision to the next decoder. Now, the second decoder will have extra information for the decoding; a priori value together with the sequence.

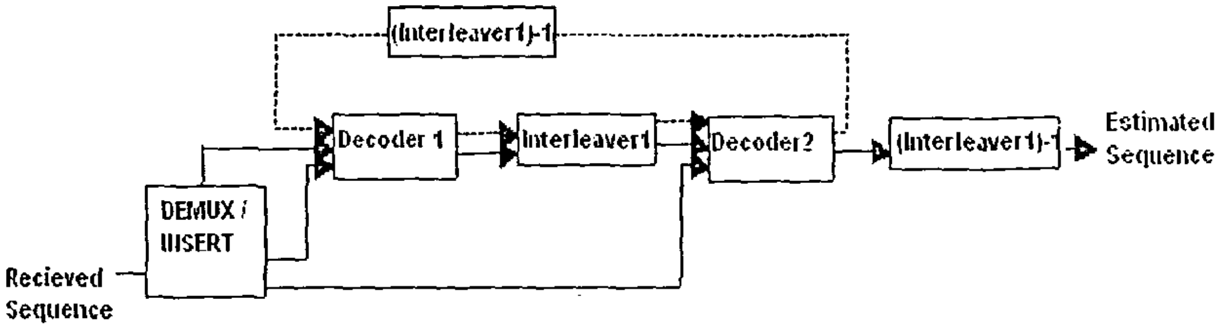


Fig. 3: Block Diagram of Turbo Codes

The interleaver in-between is responsible for making the two decisions uncorrelated and the channel between the two decoders will seem to be memory less due to interleaving. The exact procedure in what information is passed to the next decoder or next iteration stage is a subject of research. In the next section, we describe a widely accepted decoding algorithm, which is the modified version of Viterbi algorithm. [2]

V. Decoding Algorithm: SOVA

Algorithms used in decoding convolutional codes can be modified to be used in decoding turbo code. In the original paper on turbo codes [4], a modified BAHL et algorithm was proposed for the decoding stage. This algorithm is based on Maximum A-posteriori Probability (MAP). The problem with this algorithm is the inherent complexity and time delay. MAP algorithm was originally developed to minimize the bit-error probability instead of the sequence error probability. The algorithm, although optimal, seem less attractive due to the increased complexity.

Viterbi algorithm is an optimal decoding method that *minimizes the* probability of sequence error for convolutional codes. A modified version of Viterbi algorithm, called SOVA (Soft Output Viterbi Algorithm), which uses soft outputs, is introduced in [8] [9].

A. Understanding SOVA

It is proposed that an iterative decoding scheme should be used. The decoding algorithm is similar to Viterbi algorithm in the sense that it produces soft outputs. While the Viterbi algorithm outputs either 0 or *1 for each estimated bit, the turbo* code decoding algorithm outputs a continuous value of each bit estimate. While the goal of the Viterbi decoder is to minimize the code word error by finding a maximum likelihood estimate of transmitted code word, the soft output decoding attempts to minimize bit error by estimating the posterior probabilities of individual bits of the code word. We called the decoding algorithm Software Decision Viterbi Decoding. The turbo decoder consists of *M* elementary decoders - one for each encoder in turbo encoding part. Each elementary decoder uses the Software Decision Viterbi Decoding to produce a software decision for each received bit. After an iteration of the decoding process, every elementary decoder shares its soft decision output with the other *M*-1 elementary decoders.

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

The turbo-code decoder is described in the fig. 6. Assuming zero decoder delay in the turbo-decoder, the decoder 1 computes a soft-output from the systematic data (x_0), code information of encoder 1 (y_1) and a-priori information (La_2).

From *this output* the systematic data (x_0) and a-priori information (La_2) are subtracted. The result is multiplied by the scaling factor called channel reliability L_c to compensate *the distortion*. The result is uncorrelated with x_k and is denoted as Le_1, for extrinsic data from decoder 1.

Decoder 2 takes as input the interleaved version of Le_1 (the a-priori *information* La_1), *the code information of second* encoder (y_2) and the interleaved version of systematic data ($\alpha(x_k)$). Decoder 2 generates a soft output, from which *the* systematic data ($L_c\alpha(x_0)$) and a-priori *information* (La_1) was subtracted. The result is multiplied by the scaling factor called channel reliability L_c to compensate the distortion. The extrinsic data from decoder 2 (Le_2) is interleaved to produce La_2, which is fed back to decoder1 and the iterative process continues.

B. Steps of SOVA

1) Form the Trellis

Inside the decoder, Soft-Output Viterbi Algorithm (SOVA) is used to determine the result with maximum likelihood. The process SOVA is similar to that Viterbi algorithm. A trellis is formed first.

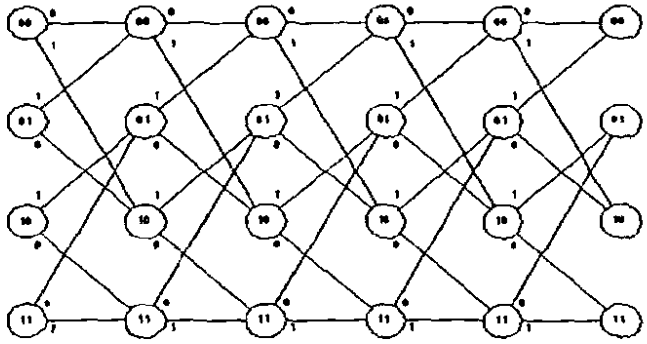


Fig. 4: Trellis Diagram

The trellis is to show how the codes are output by encoder. The bits in each node the encoder output (decoder input) for all combination of states and inputs of the encoders are summarized as follows:

State 1	State 2	Input	Output	Next State 1	Next State 2	Decoder Input
0	0	0	0	0	0	00
0	0	1	1	1	0	11
0	1	0	0	1	0	00
0	1	1	1	0	0	11
1	0	0	1	1	1	01
1	0	1	0	0	1	10
1	1	0	1	0	1	01
1	1	1	0	1	1	10

TABLE 1: TURBO ENCODER WORKING

2) Determine the Accumulated Maximum Likelihood for Each State

From the Table 1, each state has its own input bit pattern. When bit streams are inputted to decoder, they can be compared to the input ($x_0\ x_1$) to see whether they are matched. The result is represented by likelihood of input:

L_i = -1 If no bits are matched.
...0 if 1 bit is matched.
...1 if both bits are matched.

The overall likelihood of a transition is sum of likelihood of input and a-prior likelihood information (L_p).

$L = L_i + L_p$ (1)
L_p = 0.5 x a-prior information L_{un} if x_0 are 1
... -0.5 x a-prior information L_{un} if x_0 are 0

The algorithm is as follows:

Start from state 00, the overall likelihood of each transition is evaluated. The overall likelihood of each node is obtained by the maximum accumulated likelihood. With this algorithm, the trellis in figure 5 will be obtained.

3) Find the Surviving Path

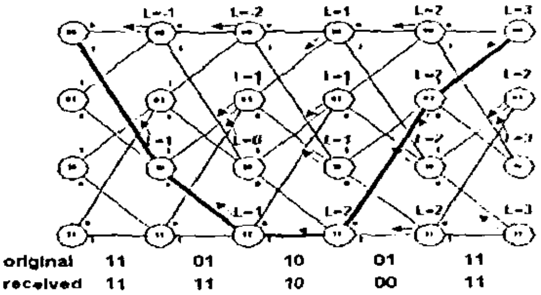


Fig.5: Surviving Path in Trellis Diagram

The surviving path is thus derived by tracing back from last stage (stage 5) to the first one (stage 0) and is the results of hard-output Viterbi Algorithm.

Let u= [1 0 1 0 1]

4) Determine the Soft Output

To find the soft-output, non-surviving paths are considered. We consider the non-surviving path the one that is produced by making different decision in one of the stage in tracing back. For example, when tracing back from (last stage) stage 5, one of the non-surviving path is found by tracing back to state 00 instead of state 01 from stage 5 to stage 4. Following the arrows, bit 1 is also 1. We found that bit 1 will have the following results when decision is changed in different stages:

Stage	1	2	3	4	5	
Bit 1	X	X	0	1	1	where X means cannot trace back to state 00 in stage 0
Delta	-	-	3	1	2	

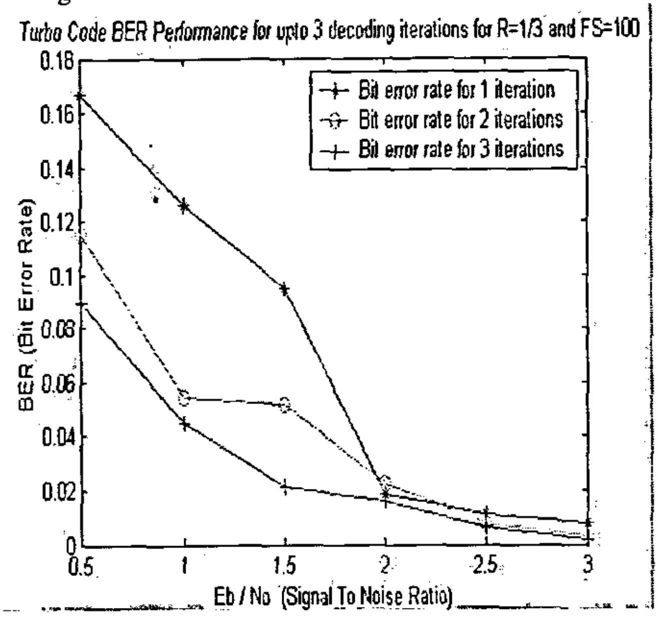
TABLE 2: NON-SURVIVING PATHS ARE CONSIDERED

Here we define a function delta to describe the tendency to have non-surviving path. It is the difference in overall likelihood when a different decision is possible in a particular stage. The soft-output of bit 1 is evaluated by the following formula:

soft-output of bit 1 = bit value x *min* (2)
(*delta* which make bit 1 change to value other than *bit value*)
bit value = *1* for bit output = *1*
...-1 for bit output = 0

In bit one, the decision only changes when a state changes in stage 3. The minimum delta is 3. Thus the soft-output of the bit one is 3.
Repeat it for bit two (originally bit two = 0).

iterations. The communication is terminated when three bit error occurs. The following graph is produced on the basis of above generated data.



Graph 1: Average BER vs. E_b/N_0

III. Conclusion:

Two conclusions are drawn from above provided simulation results:

1) As the number of iterations increases, the performance of turbo codes also increases.

2) At higher value of signal to noise ratio (E_b/N_0), efficiency of the turbo codes increases.

The graph between signal to Noise ratio (E_b/N_o) and Bit Error Rate (BER) clearly explains efficiency of turbo codec for wireless communication. When we increase the number of iterations or E_b/N_o the performance of the turbo code increases.

Furthermore, The excellent performance of turbo codes requires careful understanding of the code ingredients. The effects of the code parameters on the performance need further investigation. Proper understanding of the code will help in making the code applicable for real time applications

References

[1] Battail, Gerard, "A Conceptual framework for understanding turbo codes," *IEEE Journal on selected Areas in Communications* vol.1 No. 2,February 1998 pp 245-254

[2] "Understanding Turbo Codes", Ali .H. Mugaibel and Maan A.Kusa, King Fahd University of Petroleum and minearnals, is Published.

[3] Benedetto, Sergio and Montorsi, Guido "Design of parallel concatenated convolutional codes", IEEE Transactions on the *communications, vol.* 44, No 5, May 1996, pp.591-600.

[4] *Berrou, C., Glavieux, A. and* Thitimajhima, P "Near Shannon limits error correcting coding and decoding: tubo-codes," Proc. Of ICC '93, Geneva, May 1993, pp. 1064-1070

[5] The working of the SOVA is explained (which is available at URL:www.ie.cuhk.edu.hk/~chankm6/TurboCode/decoding)

[6] Blackert, W.j.Hall E.K., and Wilson, S.G. "Turbo code termination and interleaver conditions" Electronic *Letters vol. 31, No 24 November* 1995, pp 1182-1184

[7] *Analog to Digital* Communications, Error Control Coding, Chapter 9 Channel Encoding, Page 393

[8] Hagenauer, J.,Hoeher, P., "A Viterbi algorithm with soft-decision outputs and its applications," Proceedings of GLOBECOM '89 Dallas, Texas, Nov. 1989, pp. 47.11-47.17.- codes," Proceedings of 1995 International Symposium on Information Theory, 17-22 Sep. 1995.p.38.

[9] [9].Hagenauer, *J.and* Papke, L., "*Decoding turbo* codes with the soft-output Viterbi algorithm (SOVA)." in Proceedings of *International* Symposium On Information Theory(Trondheim, Norway, June 1994), p. 164.

[10] R. W. Hamming "Error detecting and correcting codes" in Bell Systems Technical Journal, Vol. XXVI, 1950

Date: Sun, 8 Aug 2004 17:38:57 -0000
From: "Islamabad Journal of Information Technology" <ijit@ah-automations.com>
Reply-to: "Islamabad Journal of Information Technology" <ijit@ah-automations.com>
To: skhaldoon@hotmail.com, "imranherl@isb.pol.com.pk"
Subject: regarding paper

Dear Sir

Your paper titled "Understanding Decoding Algorithm SOVA" has been selected for publication in Islamabad Journal of Information Technology.

We thank you for your participation.

Regards.

Managing Editor
Islamabad Journal of Information Technology
AH Automations
# 600 Street 110
I-8/4 Islamabad.