ADDIS ABABA UNIVERSITY SCHOOL OF GRADUATE STUDIES

ELECTRICAL AND COMPUTER ENGINEERING DEPARTMENT

COMPARATIVE STUDY OF BURST ERROR CORRECTING CAPABILITY OF CYCLIC AND CONVOLUTIONAL CODES ON DIFFERENT CHANNELS

BY

GENETU DESALEGN

ADVISOR: Dr. -Ing. HAILU AYELE

A Thesis submitted to the School of Graduate Studies of Addis Ababa University In partial Fulfillment of the Requirements for the Degree of Masters of Science In Electrical and Computer Engineering Department.

Addis Ababa, June 2005

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

"Comparative study of Burst Error Correcting capability of Cyclic and Convolutional Codes on Different Channels"

By

Genetu Desalegn

Faculty of Technology

Approval by Board Examiners

Chairman, Department Graduate Committee	Signature
Advisor	Signature
Internal Examiner	Signature
External Examiner	Signature

Acknowledgements

I would like to express my sincere and deepest gratitude to my advisor **Dr.- Ing Samuel Tadesse** for all his limitless efforts in guiding, supervising throughout my research work and for providing me useful reference materials.

I am very grateful to Dr. Messele Haile and Dr. J.N. Mukabi for their positive critical comments and giving me relevant reference documents. Also I am grateful to Ato Sintayehu Hunde and Ato Henok Fikre for their critical comments and supervision.

I express my sincere gratitude to Ato Abiye G/Amanuel for his encouragement to do my research work on the specified project and allow me getting to know helpful persons. Also I am very grateful to Ato Efrem Gebre Egziabher be with me during site supervision, his guidance to show places for lateritic soils, giving me valuable comments, documents, data, advices and encouragements.

I am very grateful to Ato Fekadu Haile, Ato Thomas G/Selassie, Ato Tesfaye Werkineh, Ato Yemane Shiferaw, Ato Dessalegn kebede, Ato Sisay Yimer, Ato Daniel Sheferaw and Ato Fikreyohannes Yadessa for their respective contribution for this research work to come to reality. In addition to AACRA laboratory workers for their restless efforts to execute almost all laboratory tests and their supervision even on public holidays.

Above all I am very thankful to **Almighty God** who is always with me in each and every step of my life. I thank God for everything He did and made education compatible with the rest of my life. Finally I would like to express my deepest gratitude to my parents, my friends Belayneh B., Buruk M., Kitaw Z. and all who contributed to this research work in one way or another.

Table of Contents pages

List of Tables **Pages**

List of Figures **pages**

List of Appendices **pages**

Abstract

Environmental interference and physical defects in the communication medium can cause errors during data transmission. Coding is a method of detecting and correcting these errors to ensure that information is transferred intact from its source to its destination.

The error detecting and correcting capability of a particular coding scheme is correlated with its code rate and complexity. A high code rate means information content is high and coding overhead is low. However, the fewer bits used for coding redundancy, the lesser error protection is provided. A tradeoff must be made between bandwidth availability and the amount of redundancy added for error protection.
There are many types of error correcting codes. 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.

The type of errors that tend to occur on the communication channel determines the choice of the coding scheme for error protection. A channel with burst errors will tend to have cluster of bit errors that occur during one transmission and thus different from channels with random errors. The prime objective of this work is to compare burst error correcting capability of cyclic and convolutional code on different channels.

To achieve the objective, (15,9) cyclic and (3,2,8) convolutional codes with burst length, l=3 , (21,9) cyclic and (3,2,13) convolutional codes with burst length, l=6 and (21,3) cyclic and (3,2,18) convolutional codes with burst length, l=9 are selected .Probability of error , P(E) is important parameter used to measure and compare the performance of the three selected codes .

1. INTRODUCTION

1.1 Motivation

The ever increasing demand for efficient and reliable data transmission and storage systems motivates the need for designing good codes and coding algorithms, so that large amount of data can be stored and transmitted efficiently. Proper encoding / decoding of information reduces errors induced by a noisy channel to any desired level without sacrificing the rate of information transmission or storage. Recent developments have contributed towards achieving the reliability required by today's high speed digital systems, and the use of coding for error control has become an integral part in the design of modern communication systems and digital computers. There are coding techniques for channels on which transmission errors occur independently in digit positions that is; each transmitted digit is affected independently by noise (Random errors). On the other hand, there are also communication channels which are affected by disturbances that cause transmission errors in to bursts (Burst errors). Codes for correcting random error are not efficient for correcting burst errors. Therefore, it is desirable to design codes specifically for correcting burst errors.

1. 2 Background

Active research is going on in this field of providing reliable transmission of data over wireless fading channels. Cyclic and convolutional codes are very effective for burst error correction. Many effective cyclic codes for correcting burst error have been discovered for the past 20 years. Cyclic codes for burst error correction were first studied by Abramson [2]. In an effort to generalize Abramsons results, Fire discovered a large class of burst error correcting cyclic

Codes [3].

Convolutional codes for correcting burst errors were first constructed by Hagel barger [4]. More efficient codes of the same type were later constructed by Iwadare [5] and Massey [6]. These codes require shorter guard space than Hagel barger codes for the same burst error correcting capability.

1.3 Error correcting codes

The theory of error detecting and correcting codes is the branch of engineering and mathematics which deals with the reliable transmission and storage of data. Information media are not hundred percent reliable in practice, in the sense that noise or any form of interference frequently causes data to be distorted. To deal with this undesirable but inevitable situation, some form of controlled redundancy is incorporated in the original data. With this redundancy, even if errors are introduced up to some tolerance level, the original information can be recovered, or at least the presence of errors can be detected. Error correcting codes do exactly this: they add redundancy to the original message in such a way that it is possible for the receiver to detect the error and correct it, recovering the original message.

The code rate, R, of a code is defined as

$$\mathbf{R = k/n} \qquad \text{…….. (1)}$$

Where,

k = message length (actual data to be sent by the user)

n = length of a block message with redundancy added to it.

Since $n \geq k$ and $k > 0$, then $0 < R \leq 1$.

Redundancy in digital communications takes the form of parity bit on the information bits.

The amount of redundancy is equal to the number of bits transmitted minus the original length of the message,

$$\mathbf{n - K = \text{parity cheek bits}} \qquad \text{…… (2)}$$

Note that the maximum code rate is 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 there are 2 bits being transmitted. A low code rate (¼ to ½) is used for channels that have many errors and require that the encoder include considerable redundancy. A high code rate (1/2 to 1) is used for channels that have few errors and do not require that the encoder include very much redundancy. Adding redundancy to a massage effectively reduces bandwidth utilization of the channel. Multiplying the bandwidth of the channel by the code rate gives the effective bandwidth after error control coding:

Bandwidth x R = (Effective Bandwidth) ……. (3)

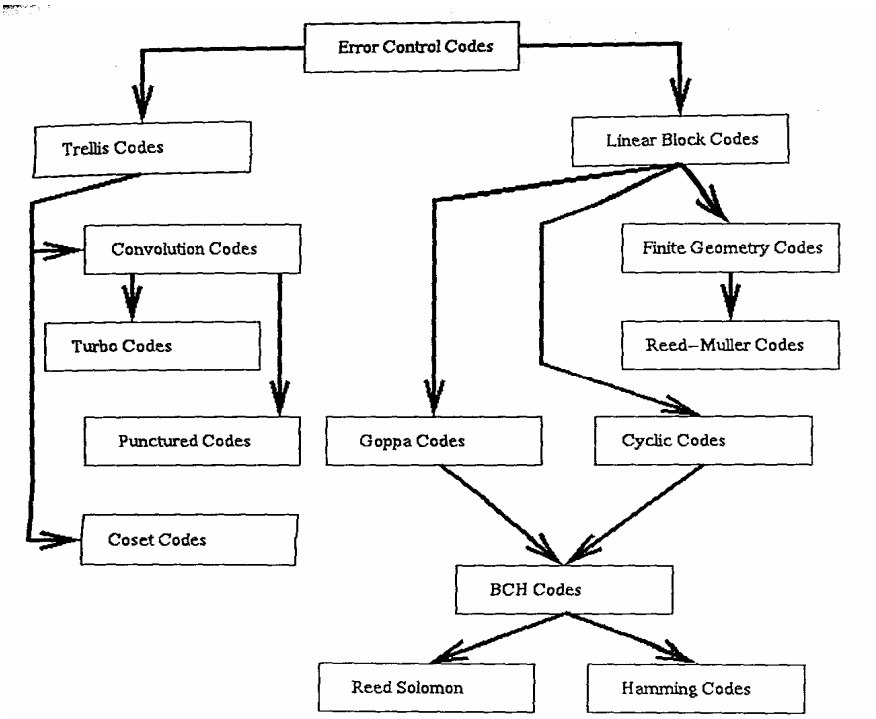


Figure-1: Classification of error correcting codes

Figure 1 shows the classification of error correcting codes. 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. All codes discussed in this work are cyclic and convolutional codes. The motivation to choose these two codes is based on

- Application of cyclic coding for error control in data storage system like magnetic drums, photo digital storage system etc.
- Application of convolutional coding for error control in HF radio, telephone line terminals etc.

1.4 Types of Errors

There are two basic types of errors.

Random-bit Error

Errors that have no relation to each other occur singly and are easily corrected. Here individual bits are corrupted and transmission errors occur randomly in received sequence. Good example random error channels are the deep-space channel and many satellite channel [3]. Most line of site transmission facilities are affected primarily by random errors.

Burst Error

These are errors where a sequence of bits is corrupted. Burst error usually results in data and redundant data loss. Correction is difficult - e.g. a long scratch in a CD. Example of burst error channels are radio channels, where the burst errors are caused by Signal fading due to multi path transmission, wire and cable transmission which are affected by impulsive switching noise and cross talk, and magnetic recording.

1.5 Stages in Error Control Coding

The block diagram shown in figure-2 represents a typical transmission system of digital

Communication that involves coding.

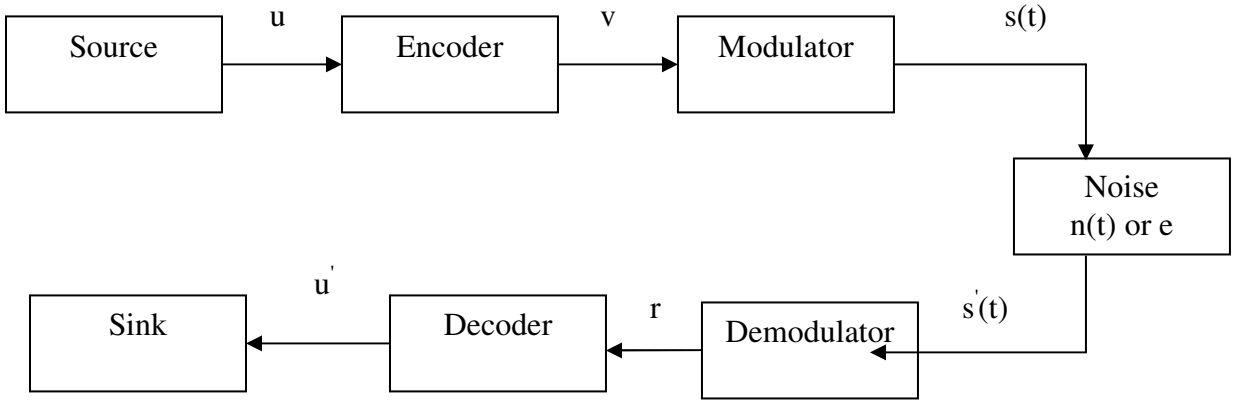


Figure- 2: Error control coding path

u = message

The source generates sequence of binary bits called massage sequence, designated by u. The binary massage can be expressed as a tuple or a polynomial.

v = codeword

The encoder transforms the information sequence **u** into discrete encoded sequence **v** called code word. Code words can also be expressed as a tuple or a polynomial. Code words include the information bits (may be 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 one.

S (t) = transmitted signal

The discrete symbols are not suitable for transmission over physical channel. Hence, the modulator converts each output symbols of the channel encoder into a waveform **s(t)** of duration T seconds. The transmitted signal is then, the modulated encoded information.

n (t) = noise signal

There are many types of noise that can be present in a communication channel, but all channels are subject to Additive White Gaussian Noise (AWGN). The waveform s (t) that enters the communication channel may be corrupted by noise, **n (t).**

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, if the transmitted codeword v has an error, then an error pattern e is added to it. The receiver would then pick up **v + e** with an error in every position where **e** has a one.

S' (t) = received signal

This represents a received modulated signal which is corrupted by noise, **n (t)**.

R= received sequence

The received sequence is the demodulated received signal. The received sequence is equal to the transmitted codeword **v** if there are no errors added by the channel. If **r** does not equal **v**, then **e** must not contain all zeros and therefore an error has occurred.

U'=decoded received sequence

If there were no errors or whatever errors did occur were correctable, **u'** will equal **u** However, if there was an undetectable error or a decoder error, **u'** will be different from **u**.

1.6 Effectiveness of Codes

The following figure shows the effectiveness of a coded signal over an uncoded signal, where P(E) is probability of occurrence of out put error [9]. The difference between the signal to noise ratios (SNR) of this plot at a particular probability of out 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 1.42 dB .

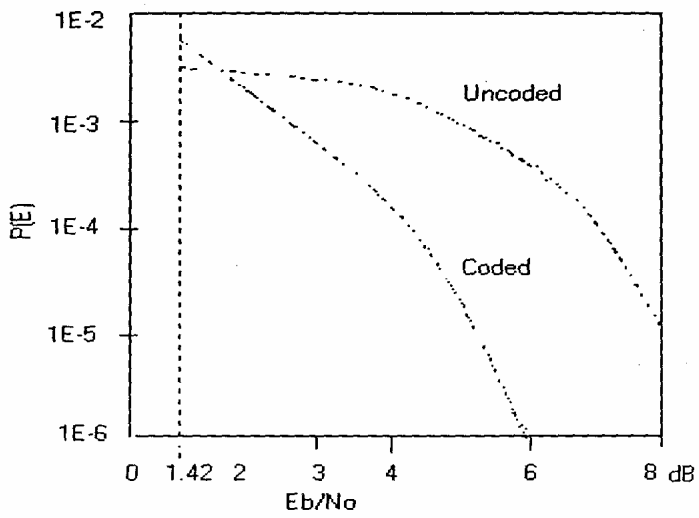


Fig-3 Error Code Efficiency: P(E) versus E_b/No

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 1.42 dB. While no known codes achieve this limit. Another concern in the effectiveness of coding is over encoding and under encoding of signal.

- Over encoding occurs when a channel does not have enough errors to justify a code rate as low as the one being used.
- Under encoding occurs when a 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 over time and different codes with different code rates need be used.

1.7 Error control strategies

To increase the apparent quality of a communication channel, there exist two basic categories of error control techniques.

Forward Error Correction (FEC)

Error control for one-way system most be accomplished using forward Error Correction (FEC), that is, by employing error- correcting codes that automatically correct errors detected at the receiver. Examples are magnetic tape storage systems, in which the information recorded on tape may be replayed weeks or even months after it is recorded and deep-space communication systems, where the relatively simple encoding equipment can be placed a board the spacecraft, but the much more complex decoding procedure must be performed on earth.

Automatic Repeat Request (ARQ)

In some cases, a transmission system can be two-way; that is, information can be sent in both direction and the transmitter also acts as a receiver and vice versa. Error control for a two-way system can be accomplished using error detection and detection capability is provided and no attempt to correct any packets received in error is made; instead it is requested that the packets received in error be retransmitted. Examples of two-way systems are telephone channel and some satellite communication system.

2. BASIC CONCEPTS OF LINEAR BLOCK CODES

In this chapter basic definition of linear block code with emphasis on encoding and decoding schemes of linear cyclic block code and some concepts of binary symmetric channels are covered.

2.1 Definition of Linear Block Codes [1]

A binary block code is a set consisting of 2^k code words, such that modulo-2 sum of any two code words is also a codeword, where k is any positive integer greater than or equal to 1. In block coding the output of information source, sequence of binary digits, “0” or “1 ‘is segmented into message

blocks of fixed length. Each message block denoted by u, consists of k information digits. There are 2 possible messages that correspond to the 2 code words of the block code.

In the development of algebraic properties of a block code the components of code vector,

$V = (v_0, v_1 \ldots, v_{n-1})$ is treated as the coefficients of a polynomial $V(X) = (v_0 + v_1X + \ldots + v_{n-1}X^{n-1}$. V(X) is called the code polynomial of v. Each code vector corresponds to a polynomial of degree n-1 or less. If $v_{n-1} \neq 0$, the degree of v(X) is n-1. If $v_{n-1} = 0$, the degree of v(X) is less than n-1. The correspondence between the vector v and the polynomial v(X) is one-to-one.

One of the most important subclass of the linear block code is cyclic code. It possesses an algebraic structure that makes the encoding and decoding scheme of this code easier than the other codes.

An (n, k) linear block code C is called a cyclic code if every cyclic shift of a code vector in C is also another code vector in C. Another important subclass of linear block code, which is frequently encountered in practice, is hamming code. It is the first class of linear codes devised for error correction. Hamming code comprises a class of codes with property that block length of a codeword $n = 2^m - 1$, and information length $k= 2^m - m -1$ where m any positive integer greater than or equal to three and equals the number of parity check digits.

Hamming code with these parameters is capable of correcting one error per message block. The following subsections present some terms and concepts that will help us to understand linear block codes.

Hamming weight

Let $V = (vo, v_1, v_2 \ldots, v_{n-1})$ be a binary n-tuple. The Hamming weight of V, denoted by w(V), is defined as the number of non-zero components of V.

Hamming weight, w (v) = number of non-zero components of V **(4)**

Hamming distance

Let **v** and **w** be two n-tuples. The Hamming distance between v and w, denoted by

d (**v ,w**) is defined as the number of places where they differ. This implies,

d (v, w) = w (v+w) **(5)**

Minimum distance

Given a block code C, the minimum distance d_{min} is defined as

d_{min} = Min {d (v,w):v,w ∈ C, v≠ w} . If C is a linear block code, the sum of two vectors is also a code vector.

Hence,

$$d_{min} = \text{Min } \{w (v+w): v,w \in C, \neq w \}$$

$$= \text{Min } \{w(X): X \in C, X \neq 0\}$$

$$= w_{min} \quad \ldots. \quad (6)$$

This implies, the minimum distance of a linear block code is equal to the minimum weight of its non-zero codewords.

Random-Error-Detecting Capabilities

When a code vector v is transmitted over a noisy channel, an error pattern of l errors may result in a received vector r, which then differs from the transmitted vector v in l places. If the minimum distance of a block code C is d_{min}, any two distinct code vectors of C differ in at least d_{min} places; no error pattern of d_{min} -1 or fewer errors can change one code vector into another.

When the receiver detects that the received vector is not a codeword then it is said that errors are detected. Hence, a block of code with minimum distance d_{min} is capable of detecting all error patterns of d_{min} -1 or fewer errors.

Random-Error-Detecting Capability = d_{min} - 1 (7)

An (n, k) linear code is capable of detecting $2^n - 2^k$ error patterns of length n. There are $2^k - 1$ undetectable error patterns. For large n, $2^k - 1$ is much smaller than 2^n. Therefore, the undetectable error patterns represent only small fraction of the total error patterns.

Random-Error-Correcting Capabilities

A block code with minimum distance d_{min} guarantees correcting all error patterns of

$$t = [(d_{min} - 1)/2] \quad \ldots\ldots\ldots (8)$$

or fewer errors, where [d_{min}-1/2] denotes the largest integer not greater than (d_{min} -1)/2.

The parameter t is called the random-error-correcting capability of the code. A t-error correcting (n, k) linear code is capable of correcting a total of 2 error patterns, including those with t or fewer errors.

From the above discussion we see that the minimum distance of a block code determines error detecting capabilities and error correcting capabilities of the code. Clearly for a given n and k one would like to construct a code with a minimum distance as large as possible.

2.2 Encoding and Decoding Schemes of Linear Block Codes [1]

Various methods are available for encoding and decoding of linear block codes. For the purpose of this work we concentrate only on the schemes for linear cyclic codes, since the work is based on one of the variations of cyclic codes.

Encoding and decoding schemes of cyclic codes based their mathematical formulation on generator polynomials and systematic structure of codewords. Definitions of these two concepts are given below.

Generator Polynomial

In an (n, k) cyclic code there exists a minimum degree code polynomial, g(X), known as generator polynomial of the code. Every code polynomial is a multiple of g(X) and every linear polynomial of degree n-1 or less that is a multiple of g(X) is a code polynomial.

An (n, k) cyclic code is completely specified by this non-zero code polynomial. Hence, we say that g(X) generates the cyclic code. The degree of g(X) is equal to the number of parity check digits of the code and has the following form:

$$g(X) = 1 + g\ X + \ldots + g_{n-k=1} X^{n-k-1} + X^{n-k} \quad \ldots\ldots \qquad (9)$$

2. 2.1 Encoding Schemes of Linear Cyclic Block Codes [1]

Every code polynomial v(X) in an (n, k) cyclic code can be expressed as

$$\mathbf{V(X) = u(X)\ g(X)}$$

$$= (u_0 + u_1 X + \ldots + u_{k-1} X^{k-1}) g(X) \qquad \mathbf{(10)}$$

Where the coefficients of u(X) are the k information digits to be encoded, g(X) is generator polynomial of the code and v(X) is the corresponding code polynomial. Hence, encoding of cyclic codes can be achieved by multiplying the message, u(X) by generator polynomial, g(X). However, the codewords obtained in this method do not have the required systematic structure. To put the codewords in systematic structure additional steps are required. Given the generator polynomial g(X) of an (n, k) cyclic code, the codewords can be encoded in systematic form by using the following three steps:

Step 1: Pre-multiply the message u(x) by X^{n-k}

Step-2: Obtain a remainder b(X) from dividing $X^{n-k}u(X)$ by the generator polynomial, g(X). The remainder b(X) is the parity - check polynomial.

Step-3: Combine b(X) and $X^{n-k}u(x)$ to obtain the desired code polynomial,

$$v(X) = b(X) + X^{n-k}u(x) \qquad \textbf{(11)}$$

All these three steps can be accomplished with a division circuit shown in figure 4. It is a linear (n-k)-stage shift register with feedback connections based on the coefficients of the generator polynomial.

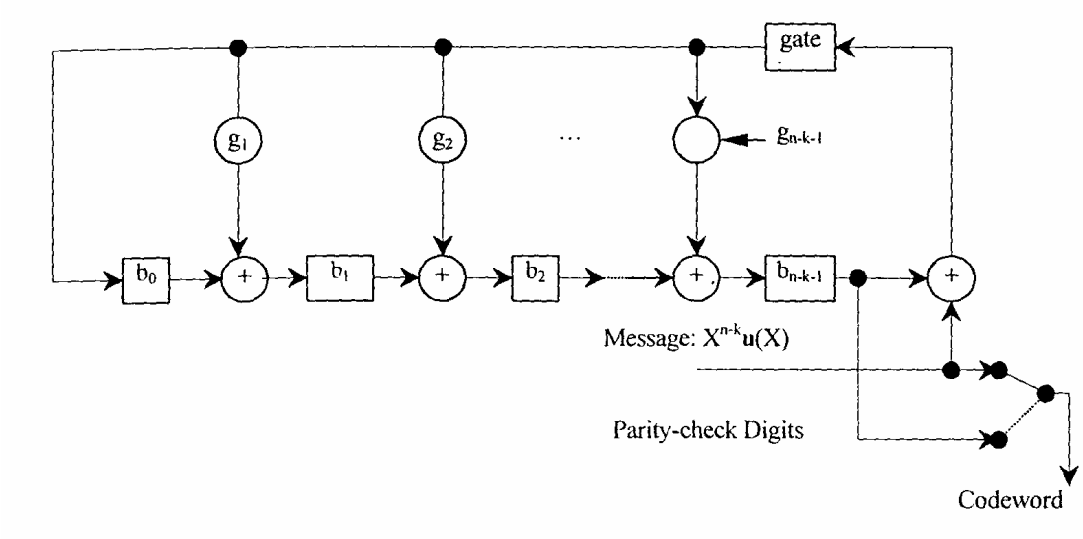


Figure-4: Encoding Circuit for an (n, k) cyclic code

The encoding operation is carried out as follows:

Step-1: With the gate turned on, the k information digits $u_0 + u_1, +...u_{k-1}$ or $U(X) = u_0 + u_1 x + ... + u_{k-1}x^{k-1}$ in polynomial form are shifted into the circuit and

simultaneously into the communication channel. Shifting the message u(X) into the circuit from the front end is equivalent to pre-multiplying u(X) by $X^{n-k.}$ As soon as the complete message has entered the circuit, the n-k digits in the register form the remainder b(X) and thus they are the parity check digits.

Step-2: Break the feedback connection by turning off the gate.

Step-3: Shift the parity-check digits out and send them into the channel. These n-k parity-check digits $b_{0,}\, b_{1\ldots}, b_{n-k-1}$ together with the k information digits, form a complete code vector.

2.2. 2 Decoding Schemes of Linear Cyclic Block Codes [1]

One of the major parts of decoding scheme of linear cyclic block code is computation of syndrome of a received vector. For a linear systematic cyclic code, the syndrome is the vector sum of the received parity digits and the parity-check digits recomputed from the received information digits.

For a cyclic code in systematic form, the syndrome can be computed by dividing r(X) by the generator polynomial g(X),

$$\mathbf{r(X) = a(X)g(X)+s(X)} \qquad \mathbf{(12)}$$

The remainder s(X) is a polynomial of degree n-k-1 or less. The n-k coefficients of s(X) form the syndrome s. The syndrome s(X) computed from the received vector r depends only on the error pattern e(X) and not on the transmitted code polynomial.

Let v(X) be the transmitted codeword and let, $e(X) = e_0 + e_1X + \ldots + e_{n-1}X^{n-1}$ be the error pattern introduced by the channel . Then the received polynomial is

$$\mathbf{r(X) = v(X)+e(X)} \qquad (13)$$

Since v(X) is multiple of the generator polynomial g(X), it can be expressed as

$$\mathbf{V(X) = b(X)\, g(X)} \qquad (14)$$

By substituting (12) and (14) in (13) we obtain Solving for e(X),

$$e(X) = a(X)g(X) + s(X)$$

$$= [a(X) + b(X)]g(X) + s(X)$$

This shows that the syndrome is equal to the remainder resulting from dividing the error pattern e(X) by the generator polynomial g(X). The syndrome computation can be accomplished with division circuit shown in Figure-5.

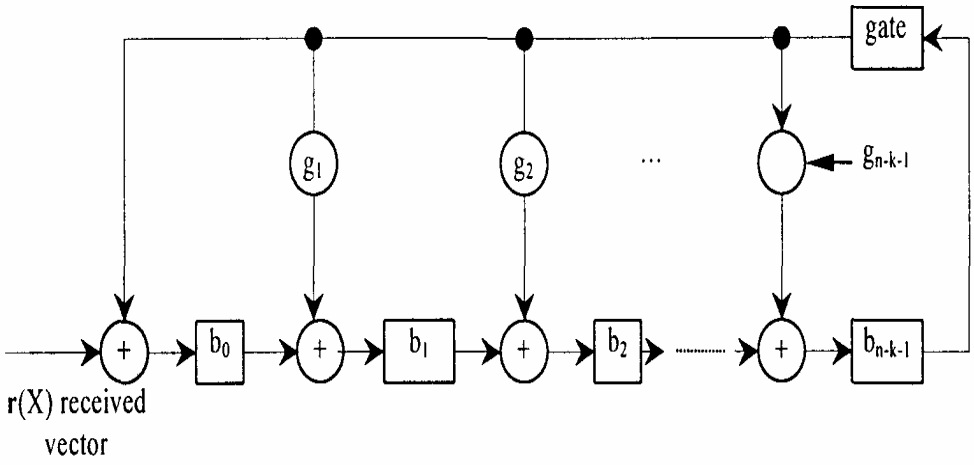


Figure-5: An (n-k) -Stage Syndrome Circuit of a Cyclic Code

As soon as the entire r(X) has been shifted into the register, the content in the register form the syndrome, s(X).

The following paragraphs present a scheme for decoding linear block codes.

Let C be an (n, k) cyclic code. Let $v_1, v_2, ...v_{2^k}$ be the code vectors of C. No matter which code vector is transmitted over a noisy channel the received vector r may be any of the 2^n n-tuple over GF (2). Any decoding scheme used at the receiver is a rule to partition the 2^n possible received vectors into 2^k disjoint subsets such that the code vector v_i is contained in only one of the subsets. Thus, each such subset has a one-to-one correspondence to a code vector. If the received vector r is found in

the subset that corresponds to the actual code vector transmitted, it will be correctly decoded as a valid code vector. Otherwise it will be wrongly decoded.

We recall that the syndrome of an n-tuple is (n-k)-tuple and there are 2^{n-k} distinct n-k syndrome tuples. Using these tuples, we can form a decoding table that consists of correctable error patterns and their corresponding syndromes. This table is either stored or wired in the receiver. Hence, the decoding of a received vector consists of the following three steps:

Step-1: Compute the syndrome, s.

Step-2: Locate the correctable error pattern e_l, whose syndrome equal to s. Then e_l assumed to be the error pattern caused by the channel.

Step-3: Decode the received vector r into the code vector v = r + e_l

This decoding scheme is called table-lookup decoding. In principle, table lookup decoding can be applied to any (n, k) linear code. It results in minimum decoding delay and minimum error probability [1]. However, for large n-k, the implementation of this decoding scheme becomes impractical, as it requires a large storage or complicated logic circuitry.

2.3. Well known Block Codes [8]

Block codes used in practical applications today belong to the class of linear cyclic codes, since these codes lend themselves to easier implementations. Hamming codes, Golay code, BCH code and Reed-Solomon (RS) codes are the widely used linear cyclic codes.

2.3.1 Hamming Codes

Hamming codes are a simple class of block codes characterized by the structure

$$(n, k) = (2^m - 1, 2^m - 1 - m) \quad \ldots\ldots \qquad (15)$$

Where m = 2, 3 … .These codes have a minimum distance of 3 and thus, from Equation (8), it is capable of correcting all single errors or detecting all combinations of two or fewer errors within a block. Syndrome decoding is especially suited for Hamming codes. In fact, the syndrome can be formed to act as a binary pointer to identify the error location [8]. Although Hamming codes are not very powerful, they belong to a very limited class of block codes known as perfect codes, Assuming hard decision decoding, the bit error probability can be written,

$$p_B \approx \frac{1}{n}\sum_{j=2}^{n} j\binom{n}{j} p^j (1-p)^{n-j} \qquad (16)$$

Where p is the channel symbol error probability (transition probability on the binary symmetric channel).

2.3.2 Extended Golay Code

One of the more useful block codes is the binary (24, 12) extended Golay code, which is formed by adding an overall parity bit to the perfect (23, 12) code, known as the Golay code.

This added parity bit increases the minimum distance $d_{\min}$ from 7 to 8 and produces a rate1/2 code, which is easier to implement (with regard to system clocks) than the rate 12/23 original Golay code. Extended Golay codes are considerably more powerful than the Hamming codes described in the preceding section. The price paid for the improved performance is a more complex decoder, a lower code rate, and hence a larger bandwidth expansion.

Since $d_{\min}$ = 8 for the extended Golay code, we see from Equation (8) that the code is guaranteed to correct all triple errors. The decoder can additionally be designed to correct some but not all four-error patterns. Since only 16.7% of the four-error patterns can be corrected, the decoder, for the sake of simplicity, is usually designed to only correct three-error patterns [8].

Assuming hard decision de coding, the bit error probability for the extended Golay code can be written as follows:

$$p_B \approx \frac{1}{24}\sum_{j=4}^{24} j\binom{24}{j}p^j(1-p)^{24-j} \qquad (17)$$

2.3.3 BCH Codes

Bose-Chadhuri-Hocquenghem (BCH) codes are a generalization of Hamming codes that allow multiple error correction. They are a powerful class of cyclic codes that provide a large selection of block lengths, code rates, alphabet sizes, and error- correcting capability.

BCH codes form a large class of powerful random error correcting cyclic codes. It is a generalization of Hamming codes for multiple-error correction.

They are characterized by the following parameters

Block Length: $n = 2^m - 1$

Parity Check Digits: $n - k \le mt$ (18)

Minimum Distance: $d_{\min} \ge 2t + 1$

For any positive integers m ($m \ge 3$) and t ($t < 2^{m-1}$). These codes are capable of correcting any combination of t or fewer errors in a block of n digits, and hence they are known as t-error-correcting BCH codes.

2.3.4 Reed-Solomon Codes [8]

Reed-Solomon (R-S) codes are non binary cyclic codes with symbols made up of m-bit sequences, where m is any positive integer having a value greater than 1. R-S (n, k) codes on m-bit symbols exist for all n and k for which $0 < k < 2^m + 2$ (19)

Where k is the number of data symbols being encoded, and n is the total number of code symbols in the encoded block. For the most conventional R-S (n, k) code,

$$(n, k) = (2^m - 1, 2^m - 1 - 2t) \quad (20)$$

Where t is the symbol-error correcting capability of the code, and n - k = 2t is the number of parity symbols. An extended R-S code can be made up with n = 2^m or n = 2^m + 1, but not any further.

Reed-Solomon (R-S) codes achieve the largest possible code minimum distance for any linear code with the same encoder input and output block lengths.

For non binary codes, the distance between two codewords is defined (analogous to Hamming distance) as the number of symbols in which the sequences differ. For Reed-Solomon codes the code minimum distance is given by

$$d_{\min} = n - k + 1 \quad (21)$$

The code is capable of correcting any combination of t or fewer errors, where t obtained from Equation (8), can be expressed as

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor = \left\lfloor \frac{n - k}{2} \right\rfloor \quad (22)$$

Where [x] means the largest integer not to exceed x. Equation (22) illustrates that for the case of R-S codes, correcting t symbol errors requires no more than 2t parity symbols.

Equation (22) lends itself to the following intuitive reasoning. One can say that the decoder has n - k redundant symbols “to spend,” which is twice the amount of correctable errors. For each error, one redundant symbol is used to locate the error, and another redundant symbol is used to find its correct value.

The erasure-correcting capability of the code is

$$\rho = d_{\min} - 1 = n - k \qquad (23)$$

Simultaneous error-correction and erasure-correction capability can be expressed by the requirement that

$$2\alpha + \gamma < d_{\min} < n - k \qquad (24)$$

Where α is the number of symbol error patterns that can be corrected, and γ is the number of symbol erasure patterns that can be corrected.

An advantage of non binary codes such as a Reed-Solomon code can be seen by the following comparison. Consider a binary (n, k) = (7, 3) code. The entire n-tuple space contains $2^n = 2^7 = 128$ n-tuples, of which $2^k = 2^3 = 8$ (or 1/16 of the n-tuples) are codewords. Next consider a non binary (n, k) = (7, 3) code where each symbol comprises m = 3 bits. The n-tuple space amounts to $2^{nm} = 2^{21} =$ 2,097,152 n-tuples, of which $2^{km} = 2^9 = 512$ (or 1/4096 of the n-tuples) are codewords.

When dealing with non binary symbols, each made up of m bits, only a small fraction (i.e., 2^{km} of the large number 2^{nm}) of possible n-tuples are codewords. This fraction decreases with increasing values of m. The important point here is that, when a small fraction of the n-tuple space is used for codewords, a large $d_{\min}$ can be created.

Any linear code is capable of correcting n - k symbol erasure patterns if the n - k erased symbols all happen to lie on the parity symbols. However, R-S codes have the remarkable property that they are able to correct any set of n -k symbol erasures within the block.

R-S codes can be designed to have any redundancy. However, the complexity of a high speed implementation increases with redundancy. Thus, the most attractive R-S codes have high code rates (low redundancy).

The Reed-Solomon (R-S) codes are particularly useful for burst-error correction; that is, they are effective for channels that have memory. Also, they can be used efficiently on channels where the set

of input symbols is large. An interesting feature of the R-S code is that as many as two information symbols can be added to an R-S code of length n without reducing its minimum distance. This extended R-S code has length n + 2 and the same number of parity check symbols as the original code.

The R-S decoded symbol error probability, p_E, in terms of the channel symbol probability p, as

$$P_E \approx \frac{1}{2^m - 1} \sum_{j=t+1}^{2^m-1} j \binom{2^m - 1}{j} p^j (1-p)^{2^m-1-j} \qquad (25)$$

3. CONVOLUTIONAL CODES [8]

This chapter deals with convolutional coding. Chapter 2 and 3 describe the fundamentals of linear block codes, which are described by two integers, n and k, and a generator matrix or polynomial. The

integer k is the number of data bits that form an input to a block encoder. The integer n is the total number of bits in the associated codeword out of the encoder. A characteristic of linear block codes is that each codeword n-tuple is uniquely determined by the input message k-tuple. The ratio k/n is called the rate of the code-a measure of the amount of added redundancy. A convolutional code is described by three integers, n, k, and K, where the ratio k/n has the same code rate significance (information per coded bit) that it has for block codes; however, n does not define a block or codeword length as it does for block codes. The integer K is a parameter known as the constraint length; it represents the number of k-tuple stages in the encoding shift register. An important characteristic of convolutional codes, different from block codes, is that the encoder has memory, the n-tuple emitted by the convolutional encoding procedure is not only a function of an input k-tuple, but is also a function of the previous K - 1 input k-tuples. In practice, n and k are small integers and K is varied to control the capability and complexity of the code.

3.1 Encoding of convolutional codes

In Figure 2 we presented a typical block diagram of a digital communication system. A modified version of this functional diagram, focusing primarily on the convolutional encode/decode and modulate/demodulate portions of the communication link, is shown in Figure 6. The input message source is denoted by the sequence $\mathbf{m} = m_1, m_2, m, \ldots$, where each **m**, represents a binary digit (bit), and i is a time index. To be precise, one should denote the elements of **m** with an index for class membership (e.g., for binary codes, 1 or 0) and an index for time. However, in this chapter, for simplicity, indexing is only used to indicate time (or location within a sequence). We shall assume that each **m,** is equally likely to be a one or a zero, and independent from digit to digit. Being independent, the bit sequence lacks any redundancy; that is, knowledge about bit **m** gives no

information about $\mathbf{m}_j$ (i ≠j). The encoder transforms each sequence m into a unique codeword sequence **U = G (m).** Even though the sequence **m** uniquely defines the sequence **U,** a key feature of convolutional codes is that a given k-tuple within m does not uniquely define its associated n-tuple within **U** since the encoding of each k-tuple is not only a function of that k-tuple but is also a function of the K - 1 input k-tuples that precede it. The sequence **U** can be partitioned into a sequence of branch words: **U** = U_1 ,U_2, …,U_i, …. Each branch word **U** is made up of binary code symbols, often called channel symbols, channel bits, or code bits; unlike the input message bits the code symbols are not independent.

In a typical communication application, the codeword sequence U modulates a waveform s(t). During transmission, the waveform s(t) is corrupted by noise, resulting in a received waveform (t) and a demodulated sequence **Z** = Z_1, Z_2 . . . , as indicated in Figure 6. The task of the decoder is to produce an estimate $\hat{m} = \hat{m}_1, \hat{m}_2, ..., \hat{m}_i$ of the original message sequence, using the received sequence **Z** together with a priori knowledge of the encoding procedure.

A general convolutional encoder, shown in Figure 7, is realized using a k-K-stage shift register and n modulo-2 adders, where K is the constraint length.

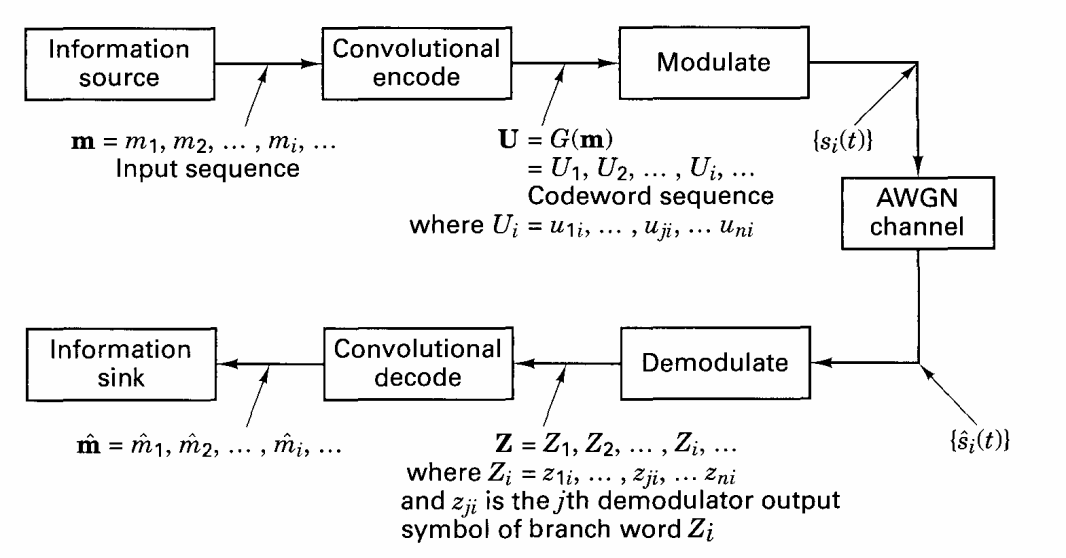
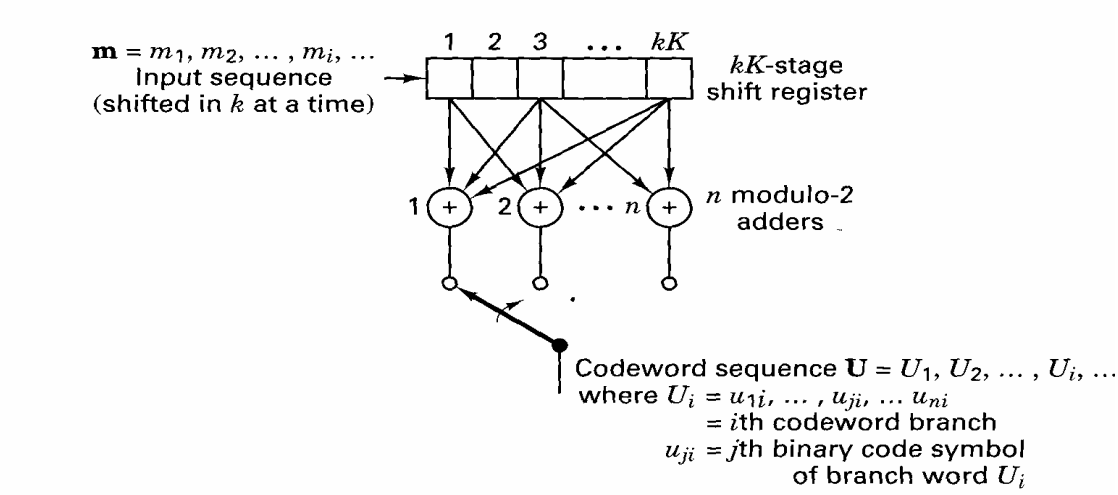


Figure 6 Encode /Decode and modulate /demodulate portion of a communication link



.

Figure 7. Convolutional encoder with constraint length K and rate k/n.

The constraint length represents the number of k-bit shifts over which a single in formation bit can influence the encoder output. At each unit of time, k bits are shifted into the first k stages of the register; all bits in the register are shifted k stages to the right, and the outputs of the n adders are sequentially sampled to yield the binary code symbols or code bits.

These code symbols are then used by the modulator to specify the waveforms to be transmitted over the channel. Since there are n code bits for each input group of k message bits, the code rate is k/n message bit per code bit, where k < n.

We shall consider only the most commonly used binary convolutional encoders for which k = 1-that is, those encoders in which the message bits are shifted into the encoder one bit at a time, although generalization to higher order alphabets is straightforward [1,2]. For the k= 1 encoder, at the i^{th} unit of time, message bit, is shifted into the first shift register stage; all previous bits in the register are

shifted one stage to the right, and as in the more general case, the outputs of the n adders are sequentially sampled and transmitted. Since there are n code bits for each message bit, the code rate is 1/n. The n code symbols occurring at time comprise the i^{th} branch word, U_i, = $u_{1i}, u_{2i}, ...u_{ni}$, where u_{ji} (j = 1, 2, . . . , n) is the j^{th} code symbol belonging to the i^{th} branch word. Note that for the rate 1/n encoder, the k-K-stage shift register can be referred to simply as a K-stage register, and the constraint length K, which was expressed in units of k-tuple stages, can be referred to as constraint length in units of bits.

3. 2 Convolutional Encoder Representation

To describe a convolutional code, one needs to characterize the encoding function G(**m**), so that given an input sequence **m**, one can readily compute the output sequence **U**. Several methods are used for representing a convolutional encoder, the most popular being the connection pictorial, connection vectors or polynomials, the state diagram, the tree diagram, and the trellis diagram. They are each described below.

3. 2.1 Connection Representation

We shall use the convolutional encoder, shown in Figure 8, as a model for discussing convolutional encoders. The figure illustrates a (2, 1) convolutional en coder with constraint length K = 3. There are n = 2 modulo-2 adders; thus the code rate k/n is 1/2. At each input bit time, a bit is shifted into the leftmost stage and the bits in the register are shifted one position to the right. Next, the output switch samples the output of each modulo-2 adder (i.e., first the upper adder, then the lower adder), thus forming the code symbol pair making up the branch word associated with the bit just inputted. The

sampling is repeated for each input bit. The choice of connections between the adders and the stages of the register gives rise to the characteristics of the code. Any change in the choice of connections re suits in a different code. The connections are, of course, not chosen or changed arbitrarily. The problem of choosing connections to yield good distance properties is complicated and has not been solved in general; however, good codes have been found by computer search for all constraint lengths less than about 20 [8]. Unlike a block code that has a fixed word length n, a convolutional code has no particular block size. However, convolutional codes are often forced into a block structure by periodic truncation. This requires a number of zero bits to be appended to the end of the input data sequence, for the purpose of clearing or flushing the encoding shift register of the data bits. Since the added zeros carry no information, the effective code rate falls below k/n. To keep the code rate close to k/n, the truncation period is generally made as long as practical.

One way to represent the encoder is to specify a set of n connection vectors, one for each of the n modulo-2 adders. Each vector has dimension K and describes the connection of the encoding shift register to that modulo-2 adder.

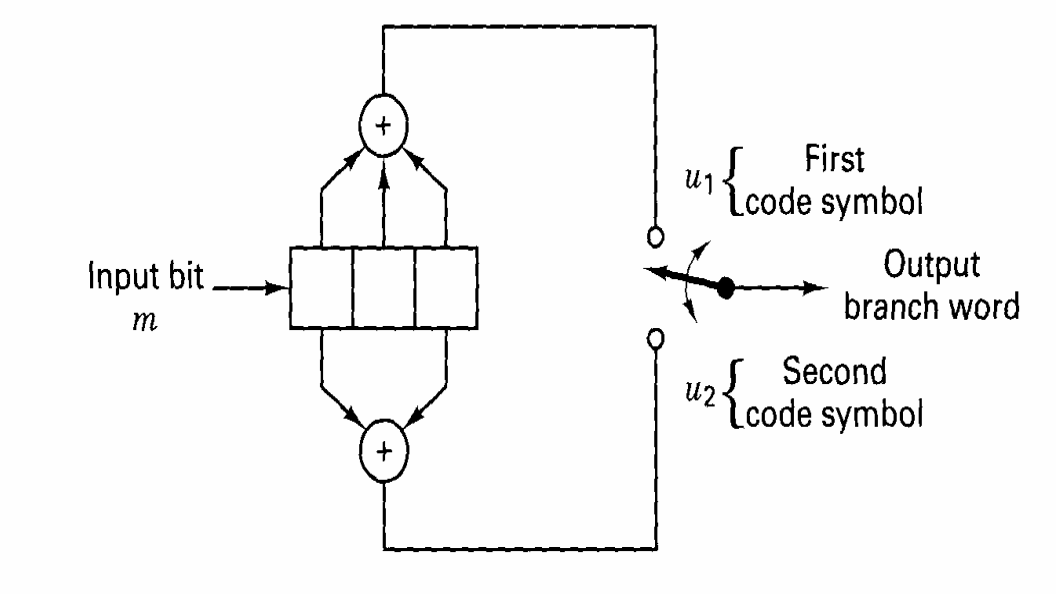


Figure 8. Convolutional encoder (rate ½, K=3)

A one in the i^{th} position of the vector indicates that the corresponding stage in the shift registers is connected to the modulo-2 adder, and a zero in a given position indicates that no connection exists between the stage and the modulo-2 adder.

3 . 2. 2 State Representation and the State Diagram

A convolutional encoder belongs to a class of devices known as finite-state machines, which the general name is given to machines that have a memory of past signals. The adjective finite refers to the fact that there are only a finite number of unique states that the machine can encounter. What is meant by the state of a finite-state machine? In the most general sense, the state consists of the smallest amount of information that, together with a current input to the machine, can predict the output of the machine. The state provides some knowledge of the past signaling events and the restricted set of possible outputs in the future. A future state is restricted by the past state.

For a rate 1/n convolutional encoder, the state is rep resented by the contents of the rightmost K - 1 stage (see Figure 8). Knowledge of the state together with knowledge of the next input is necessary and sufficient to determine the next output. Let the state of the encoder at time t be defined as $X_i = m_{i-1}, m_{i-2}, ..., m_{i-k+1}$ The i^{th} codeword branch U is completely determined by state X and the present input bit m thus the state X represents the past history of the encoder in determining the encoder output. The encoder state is said to be Markov, in the sense that the probability P($X_{i+1} / X_i, X_{i-1}, ...X_0$) of being in state X_{i+1}, given all previous states, depends only on the most recent state X_i that is, the probability is equal to P(X_{i+1} / X_i).

One way to represent simple encoders is with a state diagram; such a representation for the encoder in Figure 8 is shown in Figure 9. The states, shown in the boxes of the diagram, represent the possible contents of the right most K -1 stages of the register, and the paths between the states represent the

output branch words resulting from such state transitions. The states of the register are designated a = 00, b = 10, c = 01, and d = 11; the diagram shown in Figure 9 illustrates all the state transitions that are possible for the encoder in Figure 8. There are only two transitions emanating from each state, corresponding to the two possible input bits. Next to each path between states is written the output branch word associated with the state transition. In drawing the path, we use the convention that a solid line denotes a path associated with an input bit, zero, and a dashed line denotes a path associated with an input bit, one. Notice that it is not possible in a single transition to move from a given state to any arbitrary state. As a consequence of shifting-in one bit at a time, there are only two possible state transitions that the register can make at each bit time.

For example, if the present encoder state is 00, the only possibilities for the state at the next shift are 00 or 10.

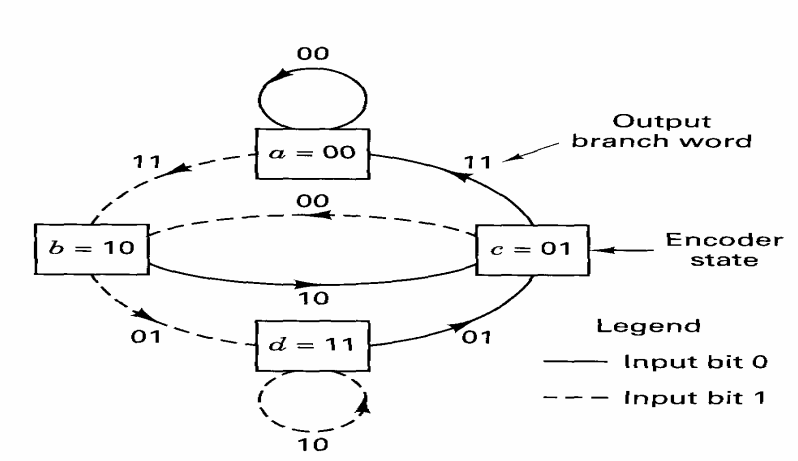


Figure 9. Encoder state diagram (rate1/2, k = 3)

3. 2.3 The Tree Diagram

Although the state diagram completely characterizes the encoder, one cannot easily use it for tracking the encoder transitions as a function of time since the diagram cannot represent time history. The tree

diagram adds the dimension of time to the state diagram. The tree diagram for the convolutional encoder shown in Figure 8 is illustrated in Figure 10. At each successive input bit time the encoding procedure can be described by traversing the diagram from left to right, each tree branch describing an output branch word. The branching rule for finding a codeword sequence is as follows: If the input bit is a zero, its associated branch word is found by moving to the next rightmost branch in the upward direction. If the input bit is a one, its branch word is found by moving to the next rightmost branch in the down ward direction.

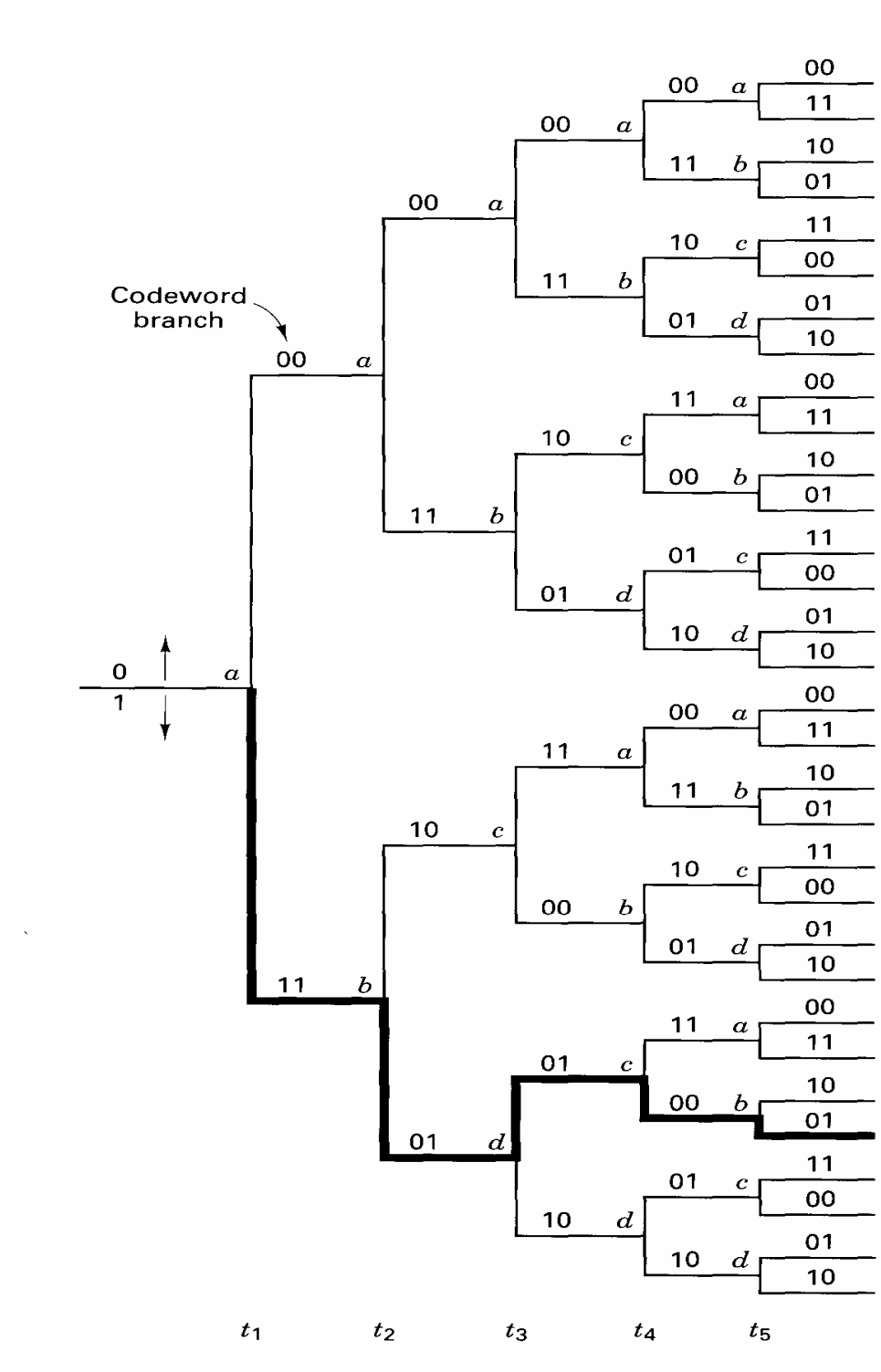


Figure 10. Tree representation of encoder (rate ½, K=3)

Assuming that the initial contents of the encoder is all zeros, the diagram shows that if the first input bit is a zero, the output branch word is 00 and, if the first input bit is a one, the output branch word is 11. Similarly, if the first input bit is a one and the second input bit is a zero, the second output branch word is 10. Or, if the first input bit is a one and the second input bit is a one, the second output branch word is 01. Following this procedure we see that the input sequence 11011 traces the heavy line drawn on the tree diagram in Figure 10. This path corresponds to the output codeword sequence 1101010001.

The added dimension of time in the tree diagram (compared to the state diagram) allows one to dynamically describe the encoder as a function of a particular input sequence. However, can you see one problem in trying to use a tree diagram for describing a sequence of any length? The number of branches increases as a function of 2 where L is the number of branch words in the sequence. You would quickly run out of paper, and patience.

3.2.4 **The Trellis Diagram**

Observation of the Figure 10 tree diagram shows that for this example, the structure repeats itself at time t_4, after the third branching (in general, the tree structure repeats after K branching, where K is the constraint length). We label each node in the tree of Figure 10 to correspond to the four possible states in the shift register, as follows: a = 00, b = 10, c = 01 and d= 11. The first branching of the tree structure, at time t produces a pair of nodes labeled a and b. At each successive branching the number of nodes double. The second branching, at time t results in four nodes labeled a, b, c, and d. After the third branching, there are a total of eight nodes: two are labeled a, two are labeled b, two are labeled c, and two are labeled d.

We can see that all branches emanating from two nodes of the same state generate identical branch word sequences. From this point on, the upper and the lower halves of the tree are identical. The reason for this should be obvious from examination of the encoder in Figure 8. As the fourth input bit enters the encoder on the left, the first input bit is ejected on the right and no longer influences the output branch words. Consequently, the input sequences 100xy ... and 000xy … where the leftmost bit is the earliest bit, generate the same branch words after the (K= 3) rd branching. This means that any two nodes having the same state label at the same time t can be merged, since all succeeding paths will be indistinguishable. If we do this to the tree structure of Figure 10, we obtain another diagram, called the trellis diagram. The trellis diagram, by exploiting the repetitive structure, pro vides a more manageable encoder description than does the tree diagram. The trellis diagram for the convolutional encoder of Figure 8 is shown in Figure 11.

In drawing the trellis diagram, we use the same convention that we introduced with the state diagram- a solid line denotes the output generated by an input bit zero, and a dashed line denotes the output generated by an input bit one. The nodes of the trellis characterize the encoder states; the first row nodes correspond to the state a = 00, the second and subsequent rows correspond to the states b = 10, c = 01, and d = 11. At each unit of time, the trellis requires 2^{k-1} nodes to represent the 2^{k-1} possible encoder states.

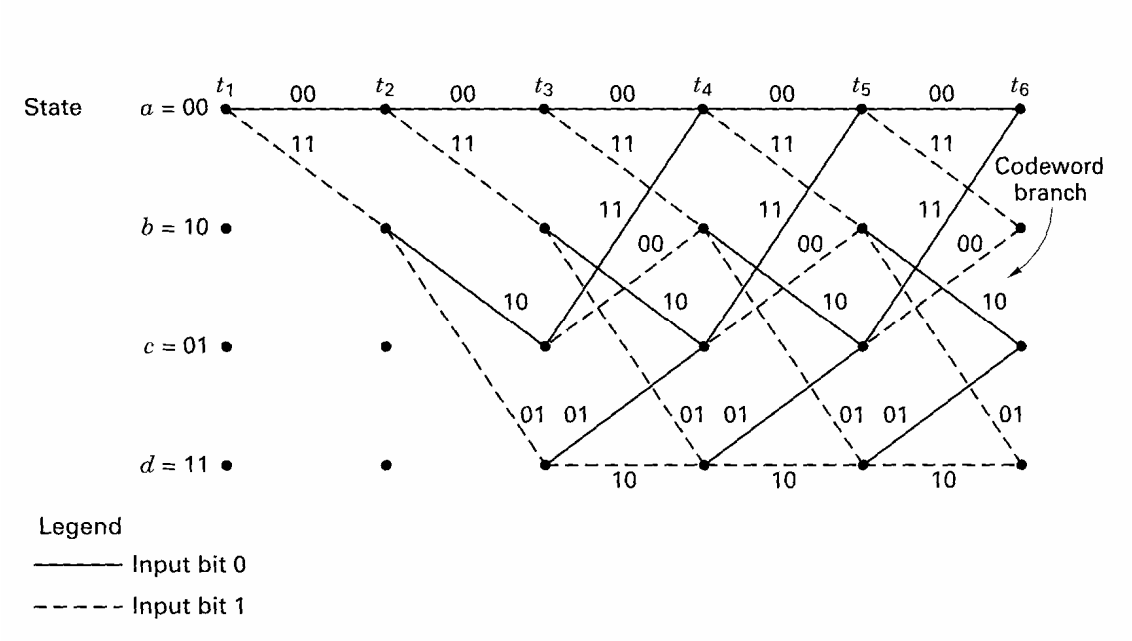


Figure 11. Encoder trellis diagram (rate1/2 , K= 3).

The trellis in our example assumes a fixed periodic structure after trellis depth 3 is reached (at time t In the general case, the fixed structure prevails after depth K is reached. At this point and there after, each of the states can be entered from either of two preceding states. Also, each of the states can transition to one of two states. Of the two outgoing branches, one corresponds to an input bit zero and the other corresponds to an input bit one. On Figure 11 the output branch words corresponding to the state transitions appear as labels on the trellis branches.

One time-interval section of a fully-formed encoding trellis structure completely defines the code. The only reason for showing several sections is for viewing a code-symbol sequence as a function of time. The state of the convolutional en coder is represented by the contents of the rightmost K -1 stage in the encoder register. Some authors describe the state as the contents of the leftmost K - 1 stage. Which description is correct? They are both correct in the following sense. Every transition has a starting state and a terminating state.

The rightmost K-1 stages describe the starting state for the current input, which is in the leftmost stage (assuming a rate 1/n encoder). The leftmost K-1 stages represent the terminating state for that transition. A code-symbol sequence is characterized by N branches (representing N data bits) occupying N intervals of time and associated with a particular state at each of N + 1 times (from start to finish). Thus, we launch bits at times $t_1, t_2, ...t_n$ and are interested in state metrics at times $t_1, t_2, ...t_{n+1}$. The convention used here is that the current bit is located in the leftmost stage (not on a wire leading to that stage), and the rightmost K -1 stages start in the all-zeros state. We refer to this time as the start time and label it t_1. We refer to the concluding time of the last transition as the terminating time and label it t_{n+1}.

3.3 Maximum Likelihood Decoding of Convolutional Codes

If all input message sequences are equally likely, a decoder that achieves the minimum probability of error is one that compares the conditional probabilities, also called the likelihood functions $p(Z/U^{(m)})$ where Z is the received sequence and U(m) is one of the possible transmitted sequences, and chooses the maximum. The decoder chooses U (m') if

$$p(Z/U^{(m')} = \max_{\text{Over all } U^{(m)}} p(Z/U^{(m)}) \qquad \ldots\ldots\ldots\ldots \qquad (26)$$

The maximum likelihood concept, as stated in Equation (26), is a fundamental development of decision theory ; it is the formalization of a “common-sense” way to make decisions when there is statistical knowledge of the possibilities. In the binary demodulation treatment , there were only two equally likely possible signals, s or s that might have been transmitted.

Therefore, to make the binary maximum likelihood decision, given a received signal, meant only to decide that s was transmitted if $p(z/s_1) > p(z/s_2)$ otherwise, to decide that s was transmitted. The parameter z represents z (fl, the receiver pre detection value at the end of each symbol duration time t = T. How ever, when applying maximum likelihood to the convolutional decoding problem, we observe that the convolutional code has memory (the received sequence represents the superposition of current bits and prior bits). Thus, applying maximum likelihood to the decoding of convolutionally encoded bits is performed in the con text of choosing the most likely sequence, as shown in Equation (26). There are typically a multitude of possible codeword sequences that might have been transmitted. To be specific, for a binary code, a sequence of L branch words is a member of a set of 2 possible sequences. Therefore, in the maximum likelihood context, we can say that the decoder chooses a particular U (m') as the transmitted sequence if the likelihood $p(Z/U^{(m')})$ is greater than the likelihoods of all the other possible transmitted sequences. Such an optimal decoder, which minimizes the error probability (for the case where all transmitted sequences are equally likely), is known as a maximum likelihood decoder. The likelihood functions are given or computed from the specifications of the channel.

3.4 Gilbert Elliott (GE) Channel Model

The Gilbert Elliott (GE) model [10,11] was chosen to model the fading channel in the simulations. In the model, the channel is a binary symmetric channel (BSC) with memory determined by a two state Markov chain. This model is shown schematically in Fig.12.

A channel has two states, a good (G) state and a bad (B) state with transition probabilities α and β as shown in the figure. In either state the channel is represented as a binary symmetric channel (BSC) with the probability of bit error given by p_g in the good and p_b in the bad state.

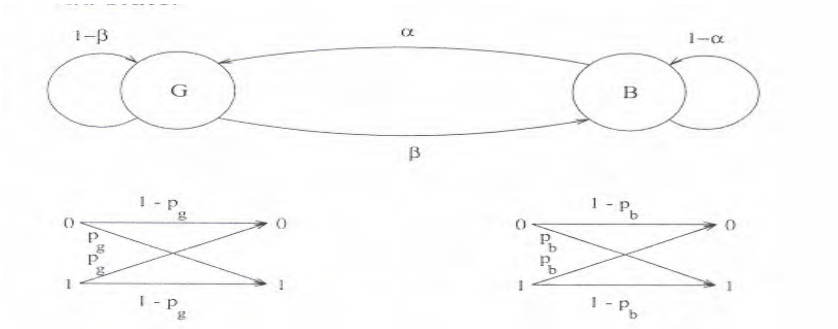


Figure 12: The Gilbert Elliott Channel Model.

Assuming that the channel fades slowly with respect to a bit interval the parameters of the model can be related to various physical quantities. To obtain such a relation, note that Raleigh fading results in an exponentially distributed multiplicative distortion of signal.

From GE (Gilbert-Elliott) channel model, the expression of probability of error (PE) for (n, k) burst error correcting cyclic or (n, k, m) convolutional code capable of correcting all cyclic or convolutional burst with burst length (BL) ≤ l is obtained as,

$$P(E) \equiv P(n, l, p) = 1 - \text{pr(No error)} - \sum_{i=1}^{l} pr(BL < l \text{ and } i \quad errors)$$

Since on BSC all error patterns with i errors are equi-probable,

$$\Pr(\quad BL \quad < \quad l \mid i \quad errors \quad) \quad = \quad \frac{n \binom{l-1}{i-1}}{\binom{n}{i}}$$

Using Bayes' rule, it follows that

$$P(E) = 1 - (1 - p)^n - \sum_{i=1}^{l} n\binom{(l-1)}{(i-1)}(1 - p)^{n-i}\, p^i. \qquad (27)$$

3.5 Binary Symmetric Channel

In a communication channel if the output of the channel depends only on the transmitted signal in that interval, and not on any previous transmission, the channel is said to be memoryless. A memoryless channel in which binary modulation is employed, amplitude distribution of the noise is symmetric and the output is quantized to two levels, is called binary symmetric channel (BSC). A BSC is completely described by a set of transition probabilities, p,

p =P (j/i), i=0 or 1, and j=0 or 1 (28)

Where i: represents a transmitted symbol,

j: represents a received symbol, and

P (j/i): is the probability of receiving j given that i was transmitted

The transition probability diagram for a BSC channel is show in Fig 13.

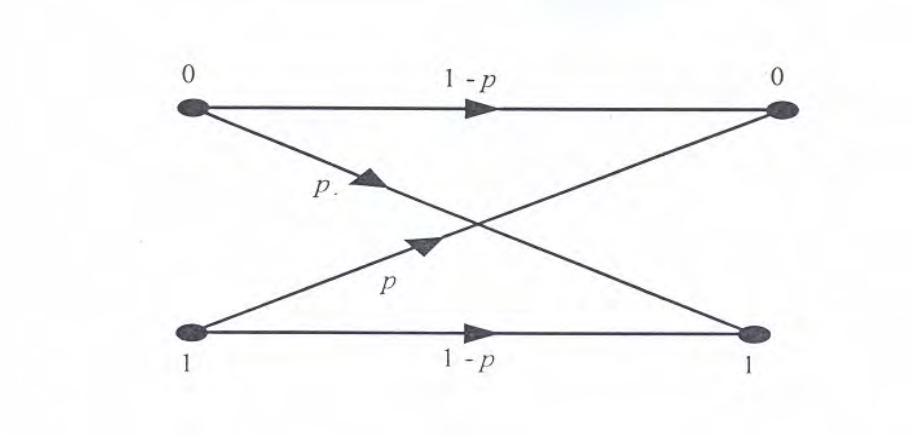


Figure 13. Transition Probability Diagram of BSC

Transition probabilities can be calculated from knowledge of the signal used the probability distribution of the noise and the output quantization threshold. When Binary phase shift Keyed (BPSK) modulation is used on AWGN channel with binary output quantization, transition probability is given by

$$P = Q\left(\sqrt{2\frac{E_b}{N_o}}\right) \tag{29}$$

Where

$$Q(x) = \int_x^{\infty}\left(\frac{1}{\sqrt{2\pi}}\right)e^{-\frac{1}{2}y^2}\,dy \tag{30}$$

$$Q(x) \le \frac{1}{2}e^{-\frac{1}{2}x^2}, \quad for \quad x \ge 0. \tag{31}$$

4. BURST ERROR CORRECTING CODES

There are communication channels which are affected by disturbance that causes transmission errors to cluster into bursts. For example, on telephone lines, a stroke of lightening or a human-made electrical disturbance frequently affects many adjacent transmitted digits. Therefore, it is desirable to design codes specifically for correcting burst errors. Codes of this kind are called burst error correcting codes [1].

4.1 Burst Error Correcting Cyclic Codes [2, 3]

A burst of length l is defined as a vector whose nonzero components are confined to l consecutive digit positions, the first and the last of which are nonzero. For example, the error vector e = (000010110100000) is a burst of length 6. A linear code that is capable of correcting all error burst of length l or less but not all error bursts of length l+1 is called an l-burst error correcting code. For given code length n and burst error correcting capability l, we design to construct an (n, k) code with as small a redundancy n-k as possible.

We now establish certain restriction on n-k for a given l, or restriction on l for given n - k.

- A necessary condition for an (n, k) linear code to be able to correct all burst errors of length l or less is that no burst of length 2l or less can be a code vector.
- The number of parity-check digits of an (n, k) linear code that has no burst of length b or less as a code vector is at least b (i.e., $n - k \geq b$).
- The number of parity-check digits of an l-burst-error-correcting code must be at least 2l, that is $n - k \geq 2l$

4.2 Error-trapping Decoder

In principle, the general method of Meggit's applies to any cyclic code, but due to its easy practical implementation, fast decoding speed, better burst error correcting capability, error trapping decoder is selected for this work.

An error-trapping decoder for an l-burst-correcting cyclic code is shown in Figure 14, where the received vector is shifted into the syndrome register from the left end.

The decoding procedure is described in the following steps:

Step 1. The received vector r(x) is shifted in to the syndrome and buffer registers simultaneously. (If we do not want to decode the received parity-check digits, the buffer register needs only k

stages). As soon as r(x) has been shifted into the syndrome register, the syndrome s(x) is formed.

Step 2. The syndrome register starts to shift with gate 2 on. As soon as it's n - k - l leftmost stages contain only zeros, its l rightmost stages contain the burst-error pattern. The error correction begins. There are three cases to be considered.

Step 3. If the n - k -l leftmost stages of the syndrome register contains all zeros after the i^{th} shift for 0 ≤ i ≤ n - k - l, the errors of the burst e(x) are confined to the parity-check positions of r(x). In this event, the k received information digits in the buffer register are error-free. Gate 4 is then activated and the k error-free information digits in the buffer are shifted out to the data sink. If the n - k - l leftmost stages of the syndrome register never contain all zeros during the first n - k - l shifts of the syndrome register, the error burst is not confined to the n - k parity-check positions of r(x).

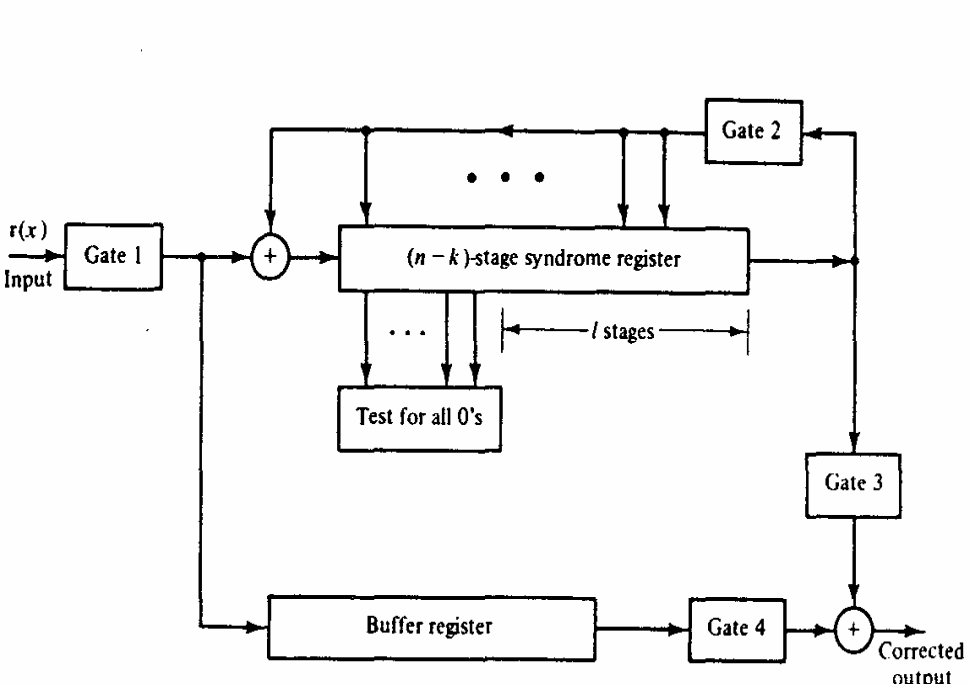


Figure 14. Error trapping decoder

Step 4. If the n - k - l leftmost stages of the syndrome register contain all zeros after the $(n - k - l + i)^{th}$ shift of the syndrome register for $l \leq i \leq l$, the error burst is confined to positions x^0, x^1, x^2,…x^{l-i-1} of r(x). (This is an end-around burst). In this event, the l-i digits contained in the l-i rightmost stages of the syndrome register match the errors at parity-check positions, x^0, x^1, x^2,…x^{l-i-1} of r(x), and the i digits contained in the next i stages of the syndrome register match the errors at the positions x^{n-i},…, x^{n-2}, x^{n-1} of r (x). At this instance a clock starts to count from (n - k -l) + i+1. The syndrome register is then shifted (in step with the clock) with gate 2 turned off. As soon as the clock has counted up to n-k, the i rightmost digits in the syndrome register match the errors at the positions x^{n-i},…, x^{n-2}, x^{n-1} of r(x). Gates 3 and 4 are then activated. The received information digits are read out of the buffer register and corrected by the error digits shifted out from the syndrome register.

Step 5. If n - k - l leftmost stages of the syndrome register never contains all zeros by the time that the syndrome register has been shifted n - k times, the received information digits are read out of the buffer register one at a time with gate 4 activated. At the same time, the syndrome register is shifted with gate 2 activated. As soon as the n- k - l leftmost stages of the syndrome register contain all zeros, the digits in the l rightmost stages of the syndrome register match the errors in the next l received information digits to come out of the buffer register. Gate 3 is then activated and the erroneous information digits are corrected by the digits coming out from the syndrome register with gate 2 disabled.

If the n - k - l stages of the syndrome never contain all zeros by the k information digits have been read out of the buffer, an uncorrectable burst errors has been detected. With the decoder described above, the decoding process takes 2n clock cycles; the first n clock cycles are required for syndrome computation and the next n clock cycles are needed for error trapping and error correction. Clock cycles for syndrome computation are concurrent with the reception of received vector from the channel; no time delay occurs in this operation. The second n clock cycles for error trapping and correction represent decoding delay.

4.3 Burst-Error-Correcting Convolutional Codes [4-7]

A sequence of error bits $e_{l+1,}\, e_{l+2...}\ e_{l+b}$ is called a burst of length b relative to a guard space of length g if:

1. $e_{l+1} = e_{l+b} = 1$
2. The g bits preceding e_{l+1}, and the g bits following e_{l+b} are all 0's
3. The b bits from e_{l+1}, through e_{l+b}, contain no subsequence of g 0's.

Consider the error sequence = (... 0000001001111, 00001101, 00010011011, 00000 …). This sequence contains a burst of length b = 28 relative to a guard space of length g = 6. Alternatively, it contains two bursts, one of length 7 and the other of length 16, relative to a guard space of length 4, or three bursts, of lengths 7, 6, and 8, relative to a guard space of length 3. This example illustrates that the length of a burst is always determined relative to some guard space, and that the two cannot be specified independently.

Gallager [12] has shown that for convolutional code of rate R that corrects all bursts of length b or less relative to a guard space of length g,

$$g/b \geq 1+R/1-R \quad \text{..... (32)}$$

Where, g = guard space

b = burst length

R = Rate

The bound is known as the bound on complete burst-error correction. Massey [7] has also shown that if we allow a small fraction of the bursts of length b to be decoded incorrectly, the guard space requirements can be reduced significantly. In particular, for a convolutional code of rate R that corrects all but a fraction e of bursts of length b or less relative to a guard space of length g.

4.4 The Viterbi Algorithm

In order to understand Viterbi's decoding algorithm, it is convenient to expand the state diagram of the encoder in time (i.e., to represent each time unit with a separate state diagram). The resulting structure is called a trellis diagram, and is shown in Figure 15 for the (3, 1, 2) code with G (D) = [1 +D, 1 +D^2, 1 +D+D^2] and an information sequence of length

L = 5.

The trellis diagram contains L + m+1 time units or levels, and these are labeled from 0 to L + m in Figure 8. Assuming that the encoder always starts in state S_o and returns to state So the first m time unit correspond to the encoder's departure from state S_o, and the last m time units correspond to the encoder's return to state S_o.

It follows that not all states can be reached in the first m or the last m time units. However, in the center portion of the trellis, all states are possible, and each time unit contains a replica of the state diagram. There are two branches leaving and entering each state. The upper branch leaving each state

at time unit i represents the input $u_i = 1$, while the lower branch represents $u_i = 0$. Each branch is labeled with the n corresponding outputs V_i, and each of the 2^L code words of length $N = n(L + m)$ is represented by a unique path through the trellis. For example, the code word corresponding to the information sequence u = (1 1 1 0 1) is shown highlighted in Figure 15. In the general case of an (n, k, m) code and an information sequence of length kL, there are 2^K branches leaving and entering each state, and 2^{KL} distinct paths through the trellis corresponding to the 2^{KL} code words.

Now assume that an information sequence $u = (u_{0,} \ldots u_{L-1})$ of length kL is encoded into a code word $V = (v_{0,\ldots,} v_{L+m-1,})$ of length $N = n(L + m)$, and that a Q-ary sequence $r = (r_{o,} \ldots r_{L+m-1})$ is received over a binary input, Q-ary output discrete memoryless channel .

Alternatively, these sequences can be written as

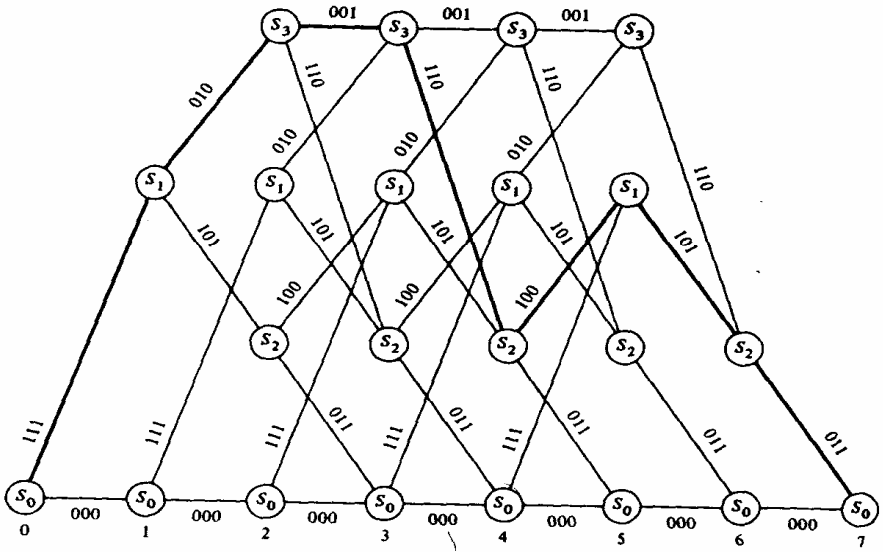


Figure 15.Trellis diagram (3, 2, 1) convolutional codes

$u = (u_{o,} u_{1,} \ldots, u_{KL-1})$, $V = (v_o v_{1,} \ldots v_{N-1,})$, and $r = (r_o, r_{1,} \ldots r_{N-0})$ where the subscripts now simply represent the ordering of the symbols in each sequence. As discussed above, the decoder must produce an estimate 9 of the code word **v** based on the received sequence **r**.

A maximum likelihood decoder (MLD) for a DMC chooses **v** as the code word v which maximizes the log-likelihood function log P(r /v).

Since for a DMC:

$$P\ (\ r\ /\ v\ )\ =\ \prod_{i\ =\ 0}^{L\ +\ m\ -\ 1} P\ (\ r_{\ i}\ v_{\ i}\ )\ =\ \prod_{i\ =\ 0}^{N\ -\ 1} P\ (\ r_{\ i}\ /\ v_{\ i}\ ) \tag{33}$$

$$\text{Log}\qquad \text{P(r/v)}\qquad = \sum_{i=0}^{L+m-1} LogP(r_i\ /V_i) = \sum_{i=0}^{N-1} LogP(r_i\ /v_i) \tag{34}$$

Where $P(r_i/v_i)$ is a channel transition probability. This is a minimum error probability decoding rule when all code words are equally likely. The log-likelihood function log P(r /v) is called the metric associated with the path v, and is denoted M(r /v). The terms log $P(r_i\ /v_i)$ in the sum of (34) are called branch metrics, and are denoted M(r/v), whereas the terms log P(r/v) are called bit metrics, and are denoted $M(r_i/v_i)$. Hence, the path metric M(r/v) can be written as

$$\mathbf{M(r/v)}\ \ =\ \ \sum_{i=0}^{L+m-1} M(r_i\ /v\ ) = \sum_{i=0}^{N-1} M(r_i\ /v_i) \tag{35}$$

A partial path metric for the first) branches of a path can now be expressed as

$$\mathbf{M\ ([r/v]_j)}\ \ =\ \ \sum_{i=0}^{j-1} M\ (r_{\ i}\ /\ v_{\ i}\ ) \tag{36}$$

The following algorithm, when applied to the received sequence **r** from a DMC, finds the path through the trellis with the largest metric (i.e., the maximum likelihood path). The algorithm processes **r** in an iterative manner. At each step, it compares the metrics of all paths entering each state, and stores the path with the largest metric, called the survivor, together with its metric.

The Viterbi Algorithm

Step 1. Beginning at time unit j = m, compute the partial metric for the single path entering each state. Store the path (the survivor) and its metric for each state.

Step 2. Increase j by 1. Compute the partial metric for all the paths entering a state by adding the branch metric entering that state to the metric of the connecting survivor at the preceding time unit. For each state, store the path with the largest metric (the survivor), together with its metric, and eliminate all other paths.

Step 3. If $j < L + m$, repeat step 2. Otherwise, stop.

5. IMPLEMENTATION OF BURST ERROR CORRECTING CAPABILITY

5.1 Codes used for implementation

The following cyclic and convolution codes were chosen for comparison.

1. (15, 9) cyclic and (3, 2, 8) convolutional codes

   The cyclic code has block length, n = 15 and the convolutional code with the block length, n = 3. Both codes are capable of correcting all error bursts having burst length, l = 3 or less.

2. (21, 9) cyclic and (3, 2, 13) convolutional codes

   Here the cyclic code has block length, n = 21 and the convolutional code with the block length, n = 3. Both codes are capable of correcting all error bursts having burst length, l = 6 or less.

3. (21, 3) cyclic and (3, 2, 18) convolutional codes

   Similarly cyclic code has block length, n = 21 and convolutional code with block length, n = 3. Both codes are capable of correcting all error bursts having burst length, l = 9 or less.

From GE (Gilbert-Elliott) channel model, the expression of probability of error (PE) for (n, k) burst error correcting cyclic or (n, k, m) convolutional code capable of correcting all cyclic or convolutional burst with burst length (BL) ≤ l is obtained as,

$$P(E) \equiv P(n, l, p) = 1 - pr(\text{No error}) - \sum_{i=1}^{l} pr(BL < l \text{ and } i \quad errors)$$

Since on BSC all error patterns with i errors are equi-probable,

$$\Pr(\quad BL \quad < \quad l \mid i \quad errors \quad) \quad = \quad \frac{n \binom{l-1}{i-1}}{\binom{n}{i}}$$

Using Bayes' rule, it follows that

$$P(E) = 1 - (1-p)^n - \sum_{i=1}^{l} n \binom{l-1}{i-1} (1-p)^{n-i} p^i .$$ This is

given in equation (27) and using BPSK modulation scheme on channel with output quantization, transition probability p is calculated for given E_b/N_o by equation (29), P(E) for both codes are calculated analytically using C^{++} Programming language and some sample values of P(E) are tabulated for the corresponding E_b/N_o and the graph P(E) versus E_b/N_o is plotted.

5.2 Result obtained

Using the channel model in above equation (27) probability of error, P(E) is calculated analytically with c++ programming language and sample values of p(E) are tabulated for each values of the transition probability p obtained from the above equation (29) by inserting the corresponding E_b/N_o .

Table 1: sample values of P(E) for (15, 9) cyclic and (3, 2, 8) convolutional codes for burst length, l = 3

Eb/No[dB]	Transition probability (P)	sample values of P(E)	
		(15 , 9)cyclic code	(3 , 2 , 8) convolutional code
-3	1.58×10^{-1}	0.623249	0.0709477
0	7.86×10^{-2}	0.265637	0.0180483
3	2.28×10^{-2}	0.0331436	0.00154767
6	2.38×10^{-3}	0.000417679	1.69797×10^{-5}
9	3.34×10^{-5}	8.36471×10^{-8}	3.34664×10^{-9}
12	9.00×10^{-9}	5.54389×10^{-15}	3.19523×10^{-16}

Fig. 16 Error probability for (15,9) cyclic and (3,2,8) conv. Codes for burst length l = 3

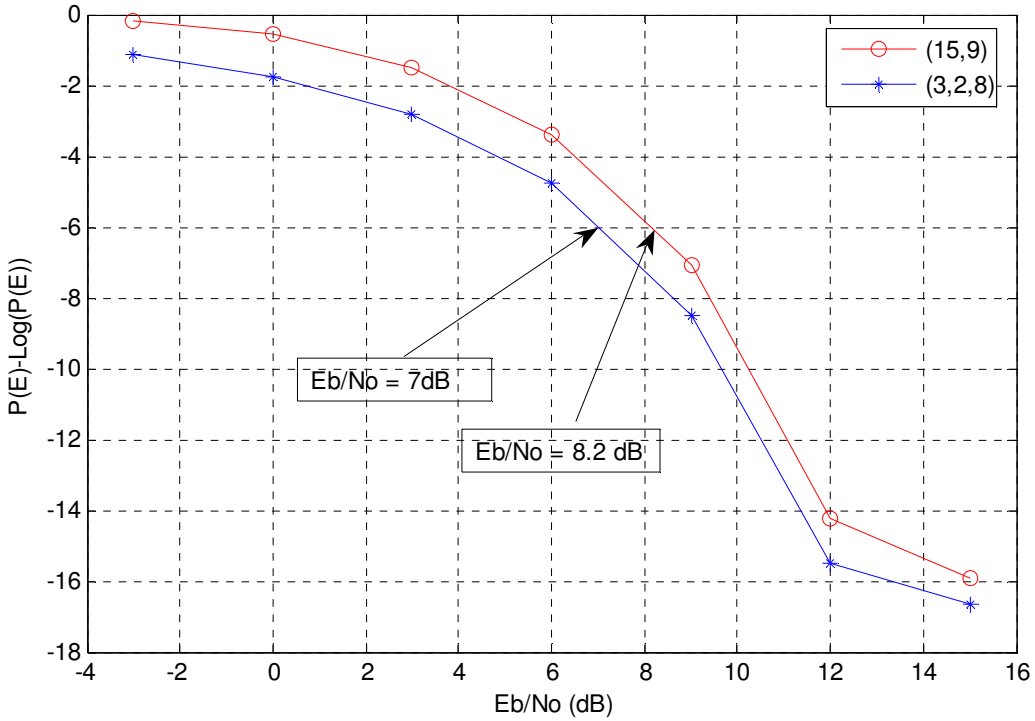


Table 2: sample values of P(E) for (15, 9) cyclic and (3, 2, 8) convolutional codes for burst length l = 4.

Eb/No[dB]	Transition probability(P)	sample values of P(E)	
		(15 , 9)cyclic code	(3 , 2 , 8) convolutional code
-3	1.58×10^{-1}	0.566776	0.159893
0	7.86×10^{-2}	0.227978	0.0381632
3	2.28×10^{-2}	0.0270933	0.00314357
6	2.38×10^{-3}	0.000334911	3.40135×10^{-5}
9	3.34×10^{-5}	6.69198×10^{-8}	6.69343×10^{-9}
12	9.00×10^{-9}	4.32889×10^{-15}	5.62523×10^{-16}

Fig. 17 Error probability for (15,9) cyclic and (3,2,8) conv. Codes for burst length l=4

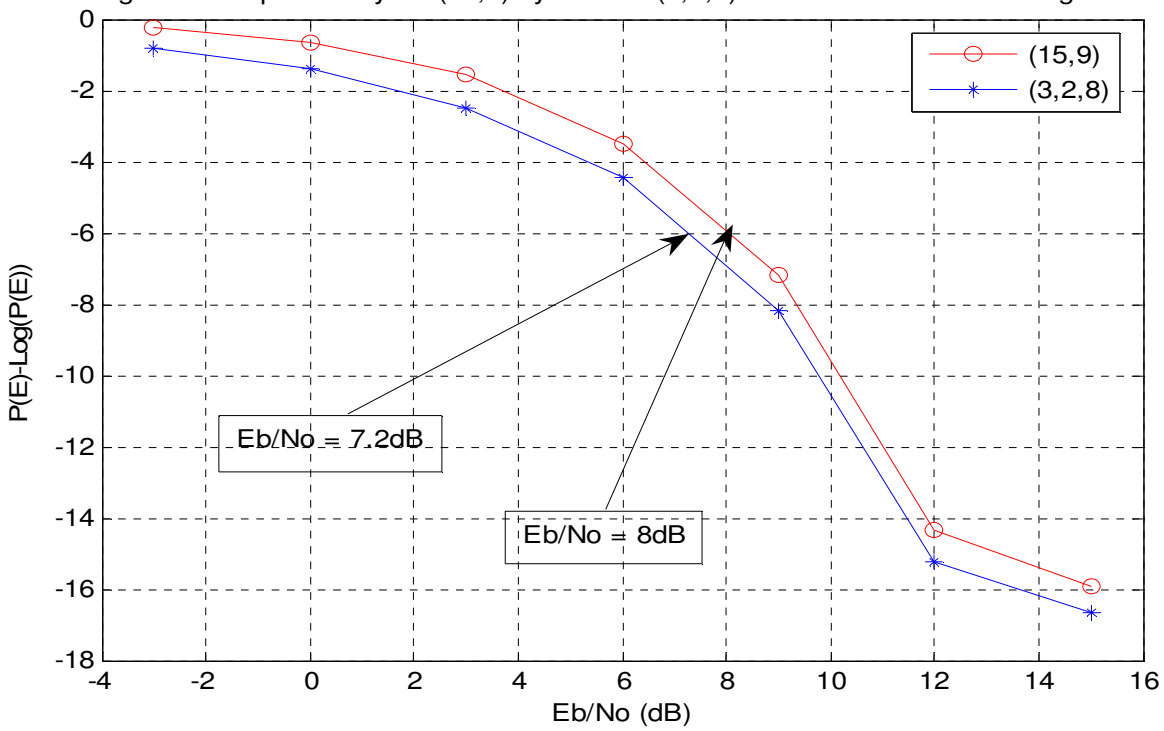


Table 3: sample values of P(E) for (15, 9) cyclic and (3, 2, 8) convolutional codes for burst length l = 5.

Eb/No[dB]	Transition probability (P)	sample values of P(E)	
		(15 , 9) cyclic code	(3 , 2 , 8) convolutional code
-3	1.58×10^{-1}	0.499706	0.265529
0	7.86×10^{-2}	0.187106	0.059994
3	2.28×10^{-2}	0.0209018	0.00477672
6	2.38×10^{-3}	0.000251946	5.10878×10^{-5}
9	3.34×10^{-5}	5.0192×10^{-8}	1.00403×10^{-8}
12	9.00×10^{-9}	3.11389×10^{-15}	8.05523×10^{-16}

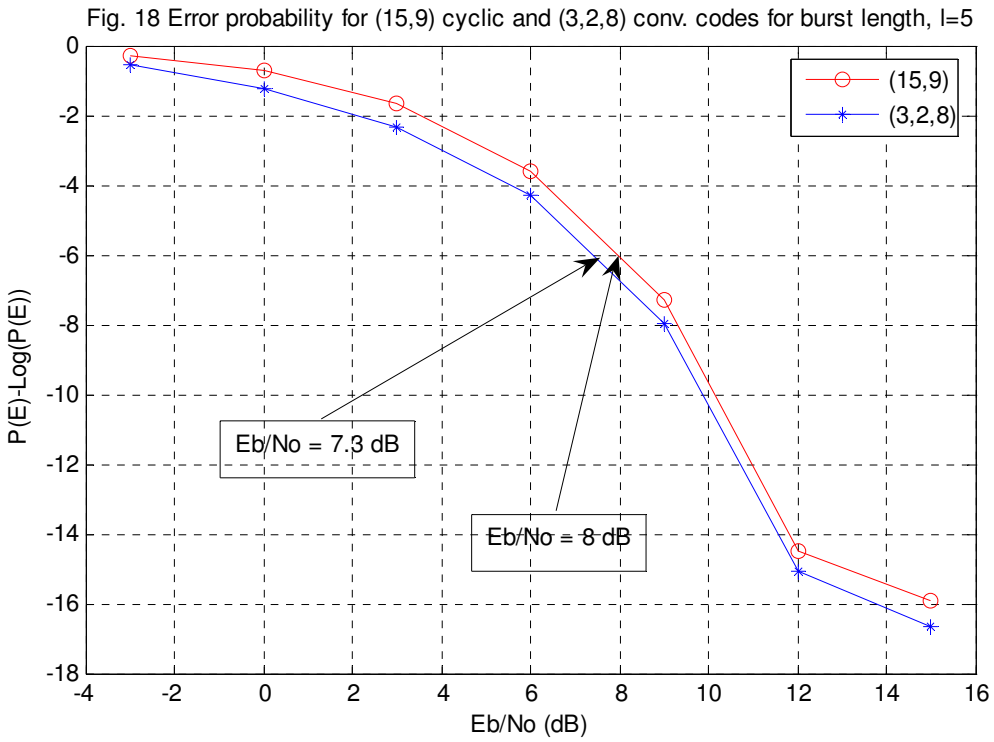


Table 4: sample values of P(E) for (21, 9) cyclic and (3, 2, 13) convolutional codes for burst length, l = 6

Eb/No[dB]	Transition Probability (P)	sample values of P(E)	
		(21 ,9) cyclic code	(3 , 2 , 13) convolutional code
-3	1.58×10^{-1}	0.721477	0.390987
-0	7.86×10^{-2}	0.337304	0.0836871
3	2.28×10^{-2}	0.0451272	0.00644796
6	2.38×10^{-3}	0.00058308	6.82029×10^{-5}
9	3.34×10^{-5}	1.17101×10^{-7}	1.33874×10^{-8}
12	9.00×10^{-9}	7.84624×10^{-15}	1.04852×10^{-15}

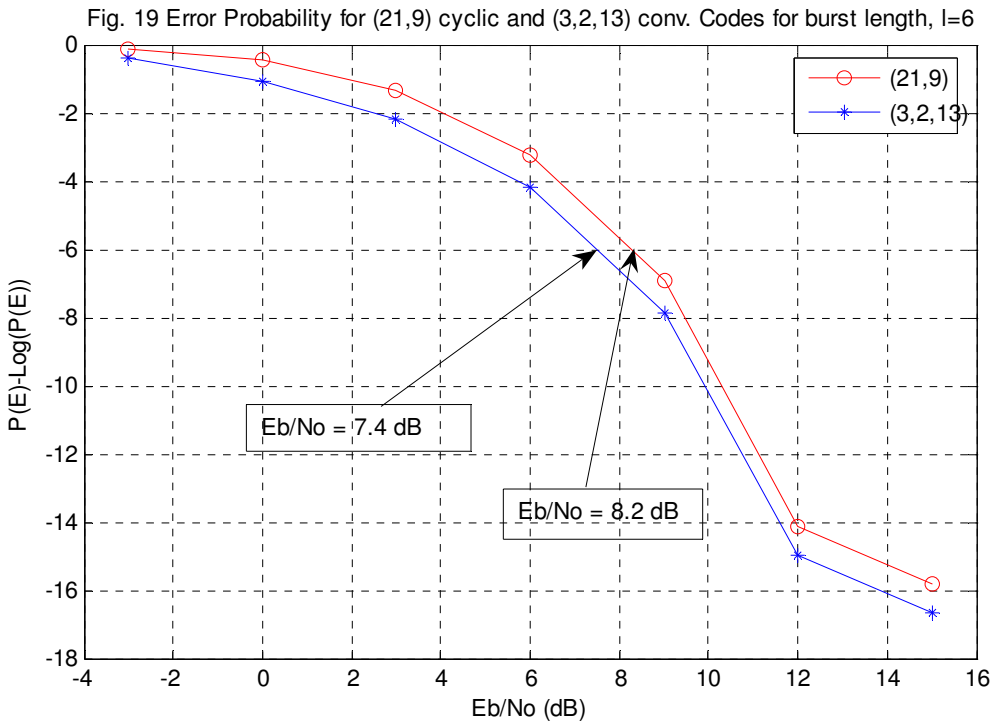


Table 5: sample values of P(E) for (21, 9) cyclic and (3, 2, 13) convolutional codes for burst length, l = 7

Eb/No[dB]	Transition robability(P)	sample values of P(E)	
		(21 , 9)cyclic code	(3 , 2 , 13) convolutional code
-3	$1.58x10^{-1}$	0.674281	0.539987
0	$7.86x10^{-2}$	0.296062	0.109401
3	$2.28x10^{-2}$	0.037223	0.0081582
6	$2.38x10^{-3}$	0.00046803	$8.53589x10^{-5}$
9	$3.34x10^{-5}$	$9.36854x10^{-8}$	$1.67345x10^{-8}$
12	$9.00x10^{-9}$	$6.14524x10^{-15}$	$1.29152x10^{-15}$

Fig. 20 Error probability for (21,9) cyclic and (3,2,13) conv. Codes for burst length, l = 7

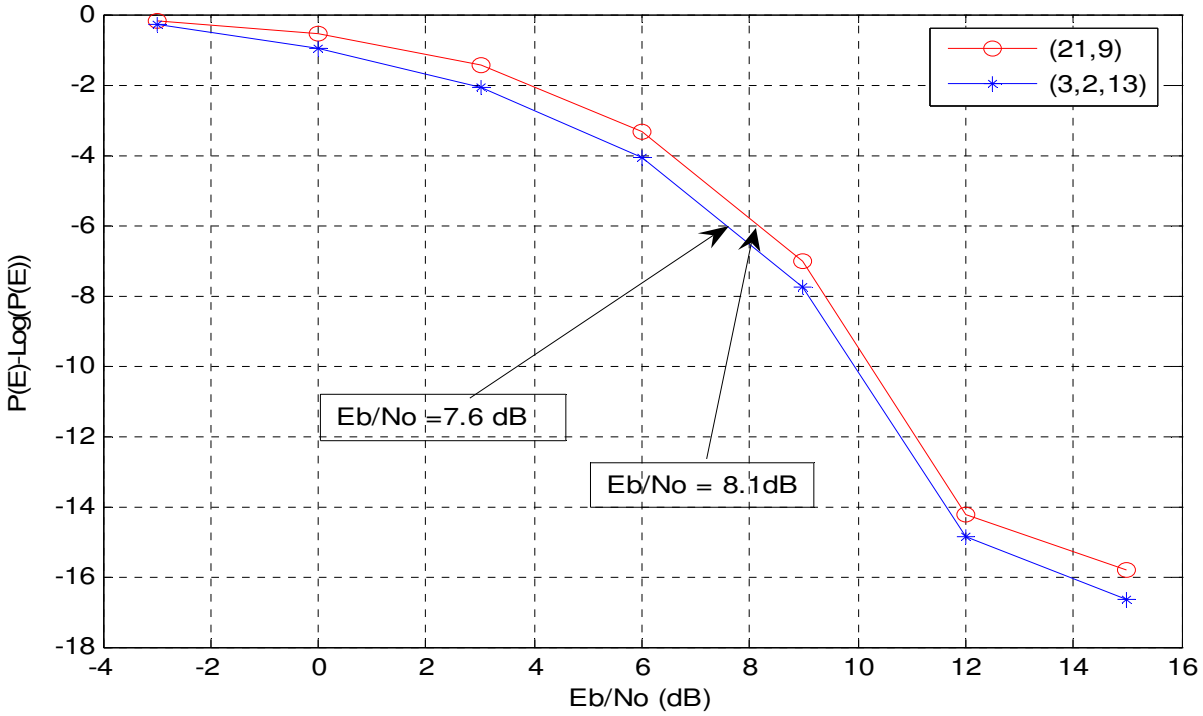


Table 6: sample values of P(E) for (21, 3) cyclic and (3, 2,18) convolutional codes for burst length, l = 9

Eb/No[dB]	Transition probability(P)	sample values of P(E)	
		(21 , 3) cyclic code	(3,2,18) convolutional code
-3	1.58×10^{-1}	0.551659	0.427114
0	7.86×10^{-2}	0.202724	0.167598
3	2.28×10^{-2}	0.020857	0.0116993
6	2.38×10^{-3}	0.000237107	0.000119794
9	3.34×10^{-5}	4.68514×10^{-8}	2.34291×10^{-8}
12	9.00×10^{-9}	2.74324×10^{-15}	1.77752×10^{-15}

Fig. 21 Error probability for (21,3) cyclic and (3,2,18) conv. Codes for burst length, l=9

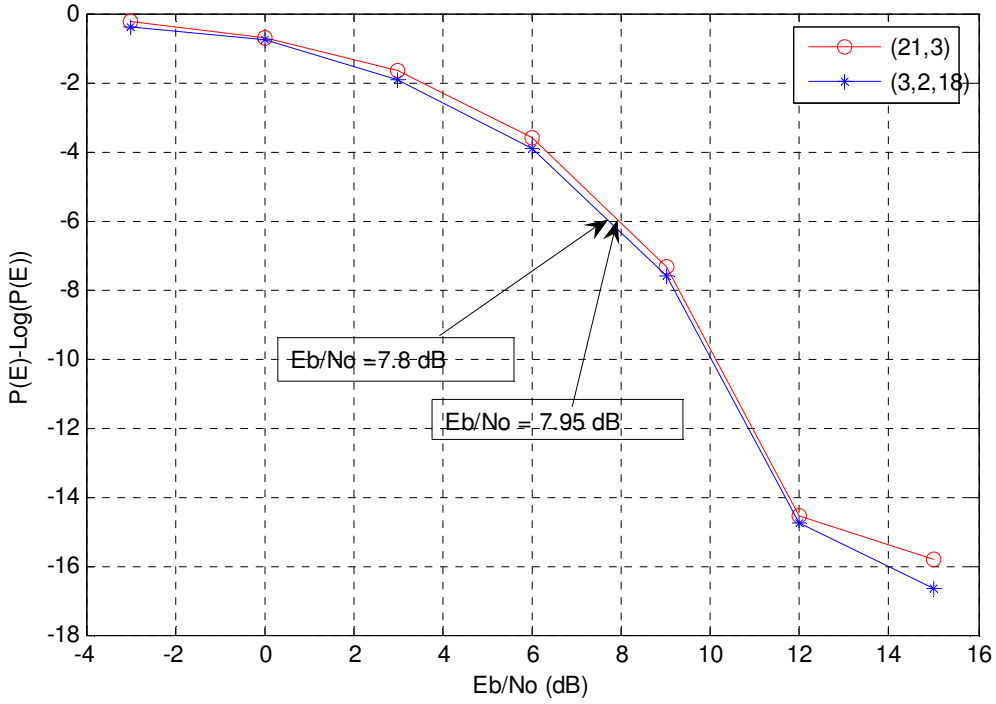


Table 7: Sample values of P(E) for simulated (15,9) and (3,2,8) convolutional codes for burst length, l = 3

Eb/No[dB]	Transition probability (P)	sample values of P(E)	
		(15 , 9) cyclic code	(3 , 2 , 8)convolutional code
-3	1.58×10^{-1}	0.643563	0.252430
0	7.86×10^{-2}	0.380876	0.031077
3	2.28×10^{-2}	0.123520	0.001758
6	2.38×10^{-3}	0.001419	6.6280×10^{-5}
9	3.34×10^{-5}	5.4×10^{-4}	4.3384×10^{-9}
12	9.00×10^{-9}	5.54389×10^{-15}	4.6656×10^{-16}

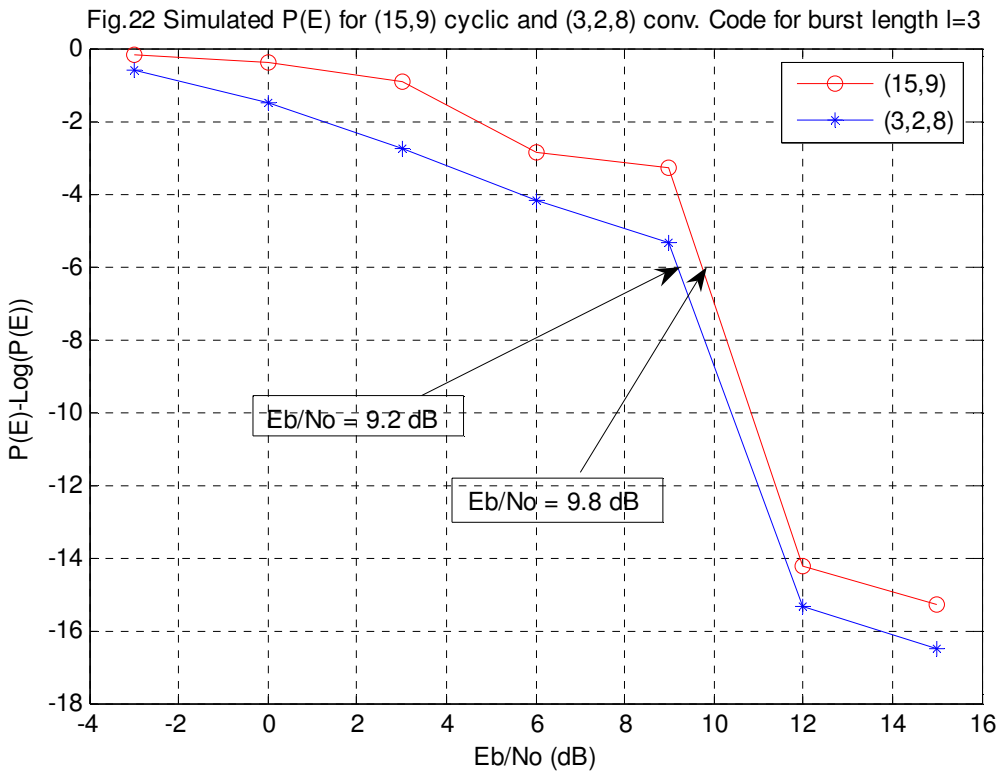


Table 8: Sample values of P(E) for simulated (15,9) and (3,2,8) convolutional codes for burst length, l = 4

Eb/No[dB]	Transition probability (P)	sample values of P(E)	
		(15 , 9) cyclic code	(3 , 2 , 8) convolutional code
-3	1.58×10^{-1}	0.52346	0.36520
0	7.86×10^{-2}	0.27068	0.12567
3	2.28×10^{-2}	0.02348	0.01678
6	2.38×10^{-3}	0.000529	5.4280×10^{-5}
9	3.34×10^{-5}	4.2×10^{-4}	3.4524×10^{-9}
12	9.00×10^{-9}	4.5428×10^{-15}	3.4456×10^{-16}

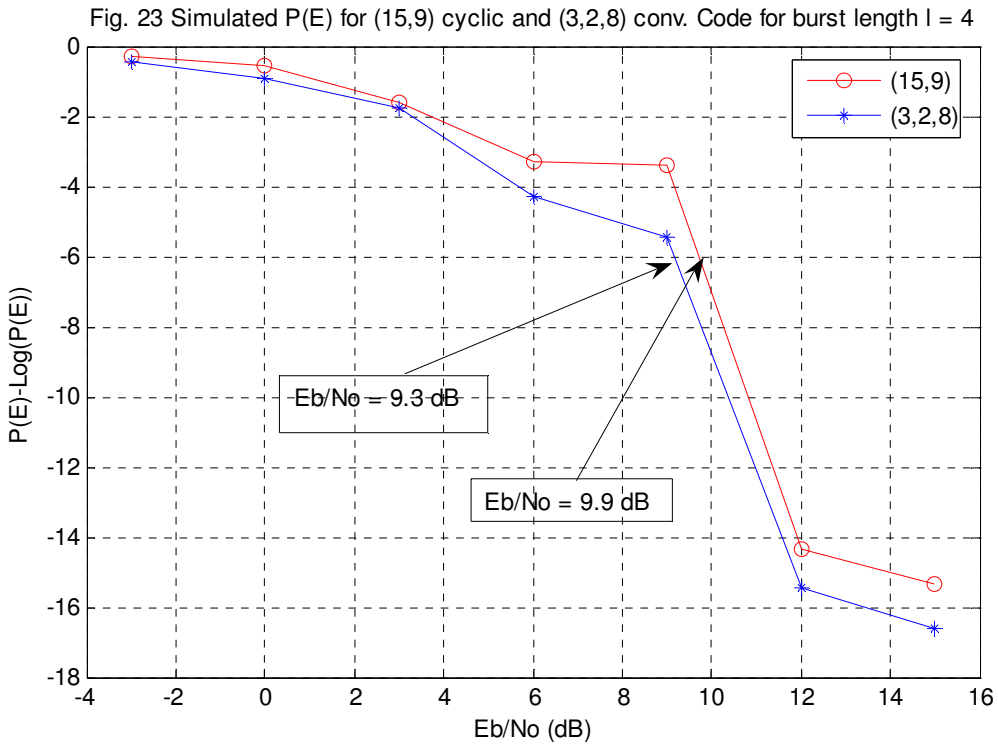


Table 9: Sample values of P(E) for simulated (15,9) and (3,2,8) convolutional codes for burst length, l = 5

Eb/No[dB]	Transition probability (P)	sample values of P(E)	
		(15 , 9) cyclic code	(3,2,8) convolutional code
-3	1.58×10^{-1}	0.42564	0.38620
0	7.86×10^{-2}	0.18956	0.15678
3	2.28×10^{-2}	0.01425	0.013642
6	2.38×10^{-3}	0.000423	6.834×10^{-5}
9	3.34×10^{-5}	3.4×10^{-4}	5.6432×10^{-9}
12	9.00×10^{-9}	3.6520×10^{-15}	4.6320×10^{-16}

Fig. 24 Simulated P(E) for (15,9) cyclic and (3,2,8) conv. Codes for burst length l = 5

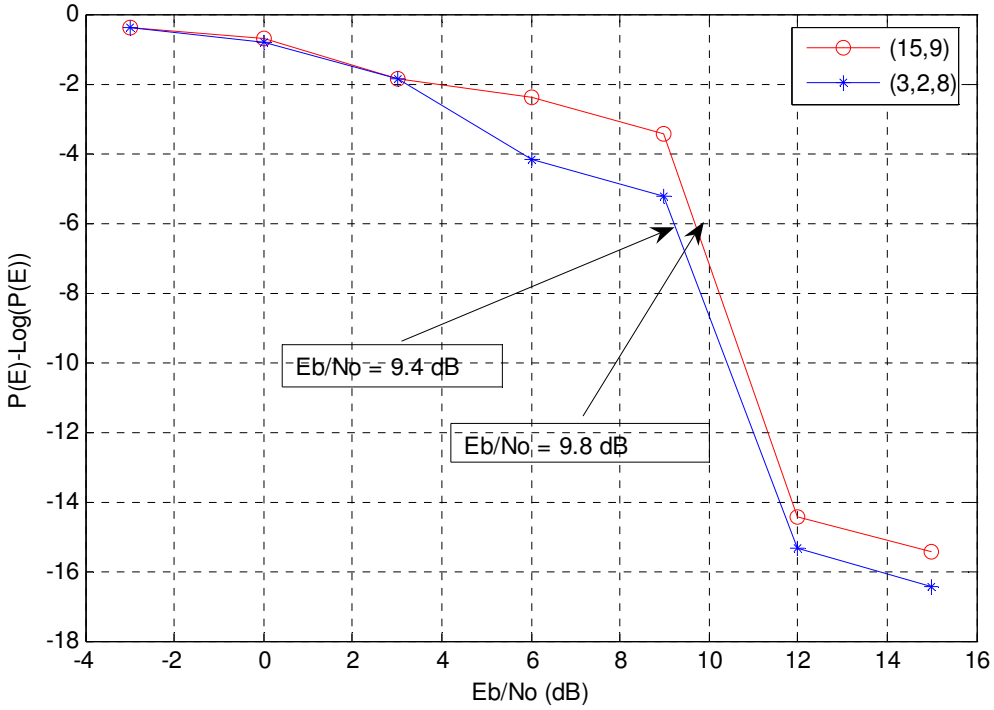


Table 10: sample values of P(E) for simulated (21,9) and (3,2,13) convolutional codes for burst length, l = 6

Eb/No[dB]	Transition Probability (P)	sample values of P(E)	
		(21 ,9) cyclic code	(3,2 ,13) convolutional code
-3	1.58×10^{-1}	0.873017	0.459530
-0	7.86×10^{-2}	0.625563	0.097708
3	2.28×10^{-2}	0.241769	0.0069179
6	2.38×10^{-3}	0.0028189	8.2138×10^{-5}
9	3.34×10^{-5}	4.00726×10^{-4}	3.4210×10^{-8}
12	9.00×10^{-9}	1.08×10^{-7}	1.6796×10^{-15}

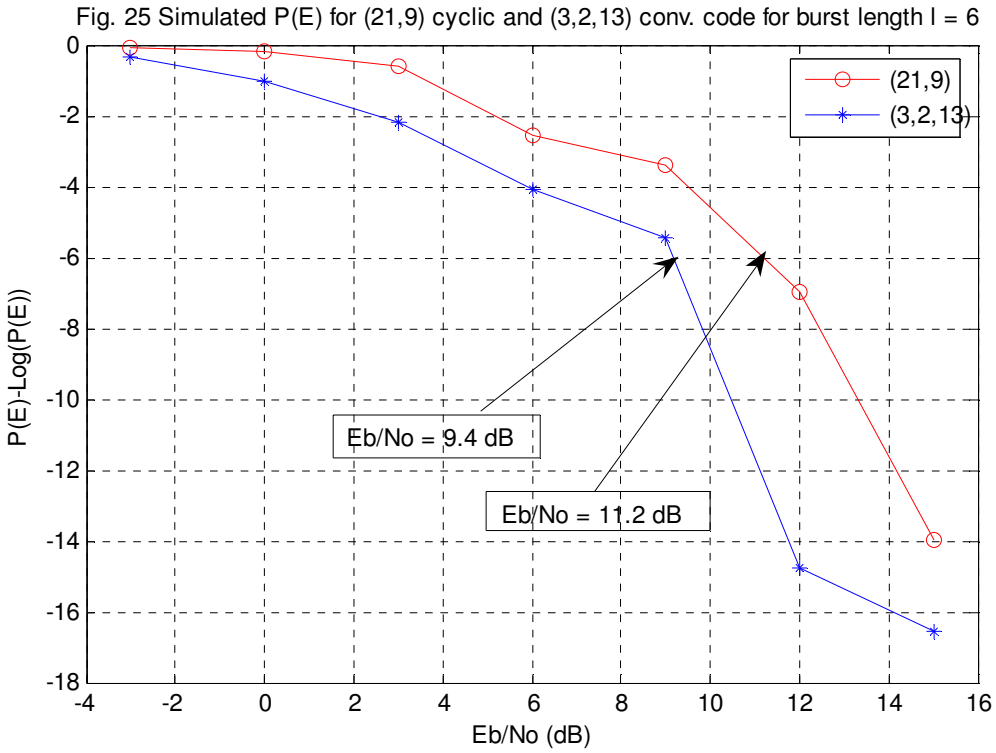


Table 11: sample values of P(E) for simulated (21,9) and (3,2,13) convolutional codes for burst length, l = 7

Eb/No[dB]	Transition Probability (P)	sample values of P(E)	
		(21 ,9) cyclic code	(3,2,13)convolutional code
-3	1.58×10^{-1}	0.76201	0.56978
-0	7.86×10^{-2}	0.54332	0.18760
3	2.28×10^{-2}	0.13259	0.09778
6	2.38×10^{-3}	0.00171	9.6452×10^{-5}
9	3.34×10^{-5}	3.11635×10^{-4}	5.6872×10^{-8}
12	9.00×10^{-9}	1.07×10^{-8}	3.4250×10^{-15}

Fig. 26 Simulated P(E) (21,9) cyclic and (3,2,13) conv. Codes for burst length l = 7

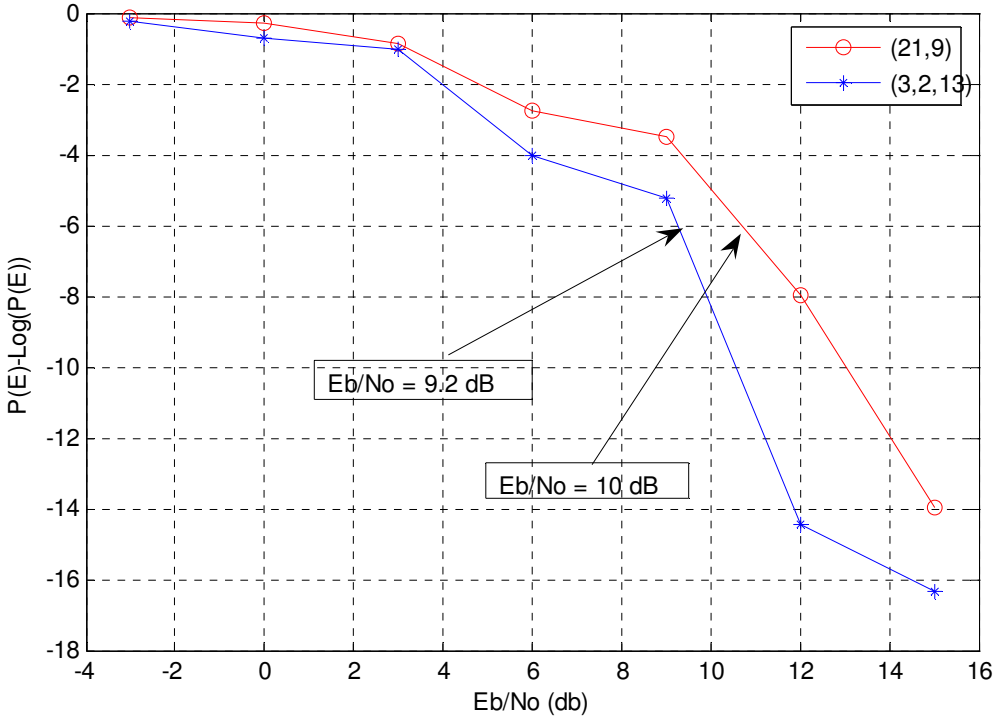
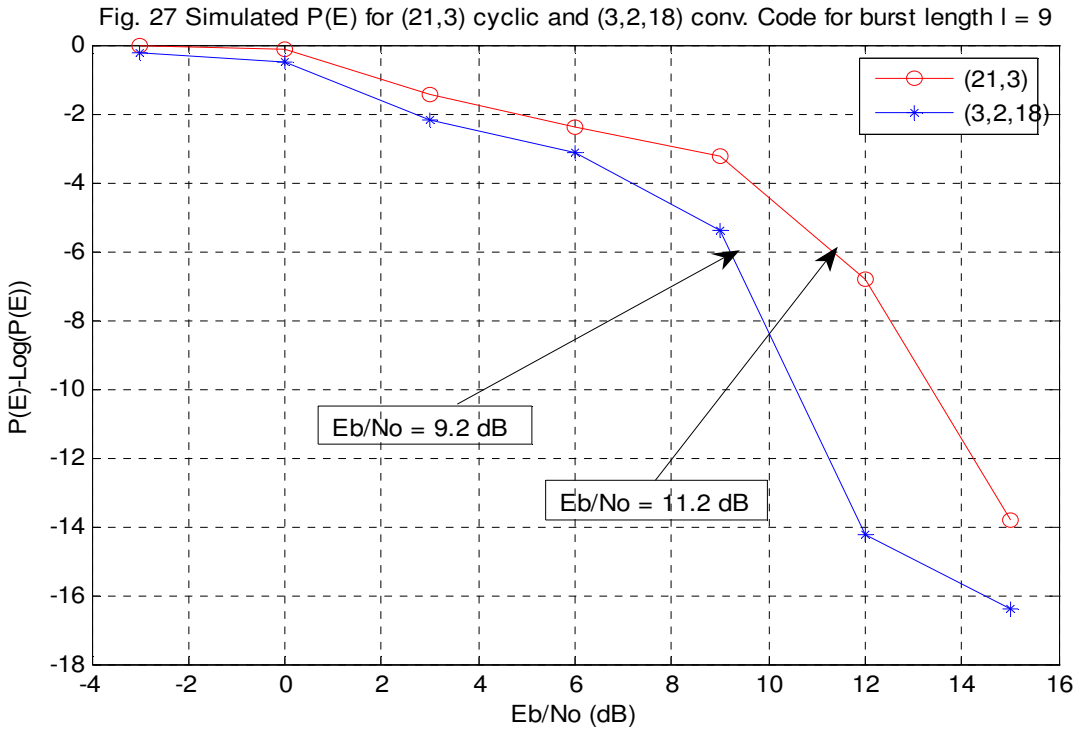


Table 12: sample values of P(E) for simulated (21,3) and (3,2,18) convolutional codes for burst length, l = 9

Eb/No[dB]	Transition probability(P)	sample values of P(E)	
		(21 , 3) cyclic code	(3,2,18)convolutional code
-3	1.58×10^{-1}	0.95475	0.60043
0	7.86×10^{-2}	0.77087	0.30719
3	2.28×10^{-2}	0.033975	0.06309
6	2.38×10^{-3}	0.004198	0.0007819
9	3.34×10^{-5}	6.0102×10^{-4}	4.2563×10^{-8}
12	9.00×10^{-9}	1.62×10^{-7}	6.0466×10^{-15}



Summery for analytically calculated results

From the above results we can summarize the performance and the behavior of the selected codes and draw some conclusions

- From table 1 we can see that (3,2,8) convolutional codes have better burst error capability than (15,9) cyclic code . It is also a faster code as compared to (15,9) cyclic code. About 66.67% (2/3X100) of the total data transmitted over the channel per code word are nformation bits as compared to 60% (9/15X100) cyclic code. Similarly from table 2 and table 3 we can see that still convolutional code has better burst error correcting capability and faster data transmission than cyclic code, when equal burst length is given to both codes.

Assuming that the upper bound of the error probability is $P(E) \leq 10^{-6}$ that is one error in million is acceptable for the types of communication under investigation over a BSC channel, we can draw the following conclusions :

- (15, 9) cyclic code satisfies the criteria for signal level ≥ 8.2dB and (3,2,8) convolutional code satisfies the criteria for signal level ≥ 7dB when burst length, l = 3.
- (15, 9) cyclic code satisfies the criteria for signal level ≥ 8dB and (3,2,8) convolutional code satisfies the criteria for signal level ≥ 7.2dB when burst length, l = 4.
- (15, 9) cyclic code satisfies the criteria for signal level ≥ 8dB and (3,2,8) convolutional code satisfies the criteria for signal level ≥ 7.3dB.

  when burst length, l = 5.
- (21,9) cyclic code satisfies the criteria for signal level ≥ 8.2dB and (3,2,13) convolutional code satisfies the criteria for signal level ≥ 7.4dB when burst length = 6.
- (21,9) cyclic code satisfies the criteria for signal level ≥ 8.1dB and (3,2,13) convolutional code satisfies the criteria for signal level ≥ 7.6dB when burst length, = 7.
- (21, 3) cyclic code satisfies the criteria for signal level ≥7.95dB and (3,2,18) convolutional code satisfies the criteria for signal level ≥ 7.8dB when burst length, l = 9.

Summery for simulated results

- (15, 9) cyclic code satisfies the criteria for signal level ≥ 9.8dB and (3,2,8) convolutional code satisfies the criteria for signal level ≥ 9.2dB when burst length, l = 3.
- (15, 9) cyclic code satisfies the criteria for signal level ≥ 9.9dB and (3,2,8) convolutional code satisfies the criteria for signal level ≥ 9.3dB when burst length, l = 4.
- (15, 9) cyclic code satisfies the criteria for signal level ≥ 9.8dB and (3,2,8) convolutional code satisfies the criteria for signal level ≥ 9.4dB when burst length, l = 5.
- (21,9) cyclic code satisfies the criteria for signal level ≥ 11.2dB and (3,2,13) convolutional code satisfies the criteria for signal level ≥ 9.4dB when burst length, l = 6.
- (21,9) cyclic code satisfies the criteria for signal level ≥ 10dB and (3,2,13) convolutional code satisfies the criteria for signal level ≥ 9.2dB when burst length, l =7.
- (21, 3) cyclic code satisfies the criteria for signal level ≥ 11.2dB and (3, 2, 18) convolutional code satisfies the criteria for signal level ≥ 9.2dB when burst length, l = 9.

Comparison of computed and simulated results

- Computed results for (15, 9) cyclic code satisfies the criteria for the signal level ≥ 8.2 dB and (3, 2, 8) convolutional code satisfies the criteria for the signal level ≥ 7dB when burst length, l = 3. The corresponding simulated result for both

codes has signal level ≥ 9.8dB and ≥ 9.2dB respectively.

- Computed results for (21,9)cyclic code satisfies the criteria for the signal level ≥ 8.2dB and (3,2,13) convolutional code satisfies the criteria for the signal level ≥ 7.4dB when the burst length, l = 6 . The corresponding simulated results for both codes have signal level ≥ 10dB and ≥ 9.2 dB respectively.
- Computed results for (21, 3) cyclic code satisfies the criteria for the signal level ≥ 7.95 dB and (3, 2, 18) convolutional code satisfies the criteria for the signal level ≥ 7.8 dB when burst length, l = 9. The corresponding simulated results for both codes have the signal level

  ≥ 11.2 dB and ≥ 9.2 dB respectively.

From the above analysis one can see that the results for the computed and simulated were consistent inspite of the differences of the signal levels required.

Comparisons based on Robustness

- (15, 9) cyclic code satisfies the criteria for signal level ≥ 8.2dB and (3,2,8) convolutional code satisfies the criteria for signal level ≥ 7dB when burst length, l = 3.
- (15, 9) cyclic code satisfies the criteria for signal level ≥ 8dB and (3,2,8) convolutional code satisfies the criteria for signal level ≥ 7.2dB when burst length, l = 4.
- (15, 9) cyclic code satisfies the criteria for signal level ≥ 8dB and (3,2,8) convolutional code satisfies the criteria for signal level ≥ 7.3dB when burst length, l = 5.

- (21,9) cyclic code satisfies the criteria for signal level ≥ 8.2dB and (3,2,13) convolutional code satisfies the criteria for signal level ≥ 7.4dB when burst length = 6.
- (21, 9) cyclic code satisfies the criteria for signal level ≥ 8.1dB and (3, 2,13) convolutional code satisfies the criteria for signal level ≥ 7.6dB when burst length = 7.

From the discussion above, good results are achieved for burst length greater than the design value but the difference can be seen when more samples or data taken out.

6.3 Conclusions and future work

Conclusions

- Convolutional codes have better burst error correcting capability and fast data transmission than cyclic codes when equal burst length given to both codes on **GE** channel model.
- Error correcting capability of convolutional codes deteriorates when burst length is increasing where as cyclic code shows better performance. Good results for burst length greater than the design values achieved for the above samples and data.
- From the above results, one can see that analytically calculated results found to be consistent inspite of the difference of the signal level required.
- Small error probabilities are achievable with cyclic codes by increasing block length, n.
- Less error probabilities are achievable with convolutional codes by increasing constraint length (memory order, m).

- Less error probabilities are achievable with both convolutional and cyclic codes by increasing redundancy (lowering code rate).
- The price paid for improved performance is a more complex decoder, a lower rate and hence a larger bandwidth expansion.

Future Work

The further works for better performance are followings:

Use of concatenated coding schemes

A concatenated code is one that uses two levels of coding, an inner code and an outer code, to achieve the desired error performance. The inner code is usually configured to correct most of the channel errors. The outer codes usually a high rate code, and then reduces the probability of error to the specific level. The primary reason for using a concatenated code is to achieve a low error rate with an overall implementation complexity which is less than that which would be required by a single coding operation.

Use of interleaving techniques

Interleaving the coded message before transmission and deinterleaving after reception causes bursts of channel errors to be spread out in time and thus to be handled by the decoder as if they were random errors. Since, in all practical cases, the channel memory decreases with time separation, the idea behind interleaving is to separate the code word symbols in time.

The intervening times are similarly filled with by the symbols of other code words. Separating the symbols in time effectively transformers a channel with memory to a memoryless one, and thereby enables the random error correcting codes to be useful in a burst noise channel.