Study and Performance Analysis
of
Error Control Codes

By :

SALAHUDDIN MUHAMMAD SALIM ZABIR

Roll No. : 921825P

Supervised By :

Dr. M. KAYKOBAD

Professor, CSE Department, BUET

SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE & ENGINNERING, BANGLADESH UNIVERSITY OF ENGINEERING & TECHNOLOGY, DHAKA, IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF MASTER OF SCIENCE IN ENGINEERING IN COMPUTER SCIENCE & ENGINEERING.

September, 1997

Department Of Computer Science And Engineering
Bangladesh University Of Engineering And Technology
Dhaka, Bangladesh

STUDY AND PERFORMANCE ANALYSIS OF ERROR CONTROL CODES

BY

Salahuddin Muhammad Salim Zabir

Accepted as satisfactory for partial fulfilment of the requirements for the degree of Master of Science in Engineering (Computer Science and Engineering), Bangladesh University of Engineering and Technology

Examiners

1. *Dr. M. Kaykobad*
Professor & Head,
Department of Computer Science & Engineering,
BUET, Dhaka.

Chairman
&
Supervisor

6.10.97

2. *Dr. Chowdhury Mofizur Rahman*
Assistant Professor,
Department of Computer Science & Engineering,
BUET, Dhaka.

Member

M Masroor Ali 6.10.1997.

3. *Dr. Muhammad Masroor Ali*
Assistant Professor,
Department of Computer Science & Engineering,
BUET, Dhaka.

Member

Md. Zafar Iqbal 6/10/97

4. *Dr. Muhammed Zafar Iqbal*
Professor & Head,
ECS Department ,
Shahajalal University of Science & Technology, Sylhet.

Member
(External)

ACKNOWLEDGEMENT

All praises are for the Almighty.

The author likes to take the opportunity to express his sincere gratitude to all who contributed in completing this work.

The author is especially indebted to Dr. M. Kaykobad, Professor, Department of Computer Science and Engineering, BUET, under whose kind supervision, encouragement, invaluable assistance and suggestion at all stages of the work, that have made it possible to complete this complex task. The services he offered personally was of great help.

Special thanks are due to Mr. Ehtesham Rafiq and Ms. Suraiya Khan both Lecturers of CSE Department, BUET who have helped sincerely at the hours of need. Grateful acknowledgement are also made to Ms. Zakia Bilkis, Lecturer The University of the Asia Pacific, for her cooperation during the course of the work.

Finally, the author thanks all other teachers and the employees of the Department of CSE, BUET for their supports are services.

S.M.S. ZABIR

ABSTRACT

Data integrity has been a major concern in cases where data handling is an essential operation. Also wherever data is to be used, there is every possibility that some errors would occur. As such, Error Control Codes have an important role to play in today's information age. Error Control Codes belong to several different families. The current work deals with codes belonging to three main families namely linear block codes, cyclic codes and convolutional codes. The underlying concepts behind the formation and construction of codes belonging to these groups, their representation, error-correction capabilities and encoding and decoding algorithms along with some other related topics have been studied during this work. Linear codes and cyclic codes belong to the wider family of block codes. Linear codes are constructed directly from algebraic formulation. Several families of linear codes have been discussed here. Cyclic codes have properties similar to linear codes. But they have an additional feature of the ability to undergo cyclic shift and still remain as a cyclic code. The convolutional codes are non block codes. Here a continuous sequence of information symbols is encoded into a continuous sequence of output. For any code, encoding and decoding are important basic operations. The current work has tried to discover a relationship between block length, information length, error correcting capacity and operations required in encoding and decoding algorithms. Computer programs have been developed and the results show that smaller block length or information length simplifies the operation of encoding or decoding along with increased error correcting ability. However channel capacity or channel bandwidth has to be sacrificed while using smaller block length or information length codes.

CONTENTS

Page

CHAPTER ONE : Introduction

1.1 : Necessity of Error Control Codes. 1
1.2 : History of Error Control Codes. 2
1.3 : Families of Error Control Codes. 2
1.4 : Scope of the current work. 3
1.5 : Organization of the thesis. 4
1.6 : Survey of Literature. 4

CHAPTER TWO : Linear Block Codes

2.1 : Introduction. 6
2.2 : Algebraic background. 8
2.3 : Important theorems, concepts & codes. 19

CHAPTER THREE : Cyclic Codes

3.1 : Introduction. 69
3.2 : Polynomial representation of cyclic codes. 70
3.3 : Matrix representation of cyclic codes. 85
3.4 : Cyclic codes for correcting single errors. 89
3.5 : Cyclic codes for correcting double errors. 91
3.6 : Cyclic codes for correcting burst errors. 93
3.5 : Some important cyclic codes. 96

CHAPTER FOVR : Convolutional Codes

4.1 : Introduction. 102
4.2 : Concepts about convolutional codes. 106
4.3 : Parity check convolutional encoder. 107
4.4 : Code tree, trellis & state diagram. 114
4.5 : Catastrophic convolutional encoder. 121
4.6 : Viterbi decoding. 128
4.7 : Minimum free distance. 137

CHAPTER FIVE : Experimental Results and Recommendations

5.1 : Design of experiments 139
5.2 : Effect of blocklength on encoding. 139
5.3 : Effect of blocklength on decoding. 141
5.4 : Recommendation for future works in related fields. 144
BIBLIOGRAPHY 145

CHAPTER ONE : INTRODUCTION

§ 1.1 : NECESSITY OF ERROR CONTROL CODES:

Data integrity is the most important concern for any system dealing with any sort or volume of data. The reason is, in order to be useful, data has to remain in its correct form. But in practical situations, this desired target is often deviated from owing to several reasons. Data may get corrupted during transmission from one place to another due to unpredicted disturbances in the media or intentional interference by unauthorized persons. Again data stored in different media may also get changed owing to problems in the media itself and its surroundings or unauthorized access. Therefore one has to devise some mechanism so that the undesired change in data can be undone and its original form is restored. Error Correction Codes or Error Control Codes (ECC) thus come into being.

1.1.1 : Causes of Errors In The Transmission Media:

When data Transmission takes place, errors may be introduced into the data, causing errors at receiver's end. The reasons that contribute to errors thus introduced may be:

(1) Noise : In electrical sense noise is an unwanted energy tending to interfere with the easy and correct reception and reproduction of the wanted signal. There are two types of noises -

External(environmental) noise : This noise is generated outside the receiver. Some causes of external noise are -

- Interference from AC mains and coupling between signal lines('Cross Talk').
- Transient and fluctuations on the AC mains.
- Noise of mechanical or acoustic origin and transmission line reflections.
- Relay contacts, thunderstorms etc.

Internal(fundamental) noise : This noise is generated by any of the active or passive devices in the receiver(internal to the receiver). Some internal noises are-

- Thermal noise - Vibration of electrons at temperature higher than normal or room temperature may introduce thermal noise.
- Shot noise – Electrons are emitted from active devices in receiver. This may cause introduction of random noise.
- Flicker noise - This is due to the random fluctuation in the current at low frequency (below 10 khz).

(2) Attenuation : It is loss in signal power in transmission media.

(3) Intersymbol Interference : Some symbols of the transmitted data may overlap into adjacent time slots(symbols), causing intersymbol interference. It has wide presence particularly in telephone lines.

(4) Signal Fading : In this case, due to imperfect response of the transmission system, the received signal amplitude is found to fluctuate randomly.

(5) Multipath Effect : In this case, radiated transmitter energy for one signal symbol, following several alternative paths to the receiver appears at the receiver as a sequence of received symbols.

1.1.2 : Applications of Error Control Codes :

From above discussion, it is clear that the main reason for using Error Control Codes or Error Control Codes are to correct errors in information when the information is transmitted from one place to another. Some of the other reasons are -

- To protect data against intentional enemy interference.
- Error Control Codes are an excellent way to reduce power needs, because massage received weakly at their destinations can be recovered correctly with the aid of the code.

Using Error Control Codes, difference between low and high levels can be kept smaller, thus a reduction in average signal power requirement is possible. For this, though the probability of occurrence of errors becomes higher as a consequence of reducing the difference between lower and higher levels, however, using Error Control Codes the errors can be corrected to certain extent.

§ 1.2 : HISTORY OF ERROR-CONTROL CODES :

The history of error-control codes began in 1948 with the publication of a famous paper by Claude Shannon. Shannon showed that associated with any communication channel is a number *C* (measured in bits per second), called the capacity of the channel, which has the following significance. Whenever the transmission rate *R* (in bits per second) required of a communication system is less than *C*, it is possible to design for the channel a communication system using error-correcting codes, whose probability of output error is as small as desired. Throughout the 1950s, much effort was devoted to find explicit constructions for classes of codes that would produce the promised arbitrarily small probability of error. As such, in the 1960s, coding research began to settle down to two main avenues-

1) The first avenue has a strong algebraic flavor and is concerned primarily with block codes. The *block codes* were introduced in 1950, when Hamming described a class of single-error correcting block codes.

2) The second avenue of coding research has a more probabilistic flavor. This attempt led to the notion of sequential decoding. Sequential decoding required the introduction of a class of *nonblock codes* of indefinite length, which can be represented by a tree and can be decoded by algorithms for searching the tree.

During the decade of the 1970s, these two avenues of research began to combine again. During the decade of the 1980s, encoders and decoders began to appear frequently in newly designed digital communication systems and digital storage systems.

§ 1.3 : FAMILIES OF ERROR CONTROL CODES :

From the above discussion, it is clear that there are two main types of error-correcting codes-one has strong algebraic structure, called the block code and another has no strong algebraic structure, called the tree code. The presence of strong algebraic structure is important-because in such case encoding , decoding and computation is efficient as generalized algorithm can be applied.

Otherwise, tabular decoding technique should be used and maintaining table is impractical for large block length code of arbitrary form (no strong algebraic structure).

Block codes are used, when information is dealt with as blocks, while tree codes are used, when information come in a sequence, not in blocks. Codes belonging to these two major families may further be classified into

1) Linear Block Codes : In block coding the encoder accepts *k* message symbol and generates a codeword with *n* symbols. Thus codewords are produced on a block by block basis. Linear block codes are so called as every codeword of a code can be produced by adding (linear operation) other codewords of the code. Clearly provisions must be made in the encoder to buffer an entire message block before generating the codeword. There are applications , however , where the message bits come serially rather than in large block , in which case the use of a buffer may be undesirable . In such situations , the use of convolutional coding may be a preferred method . Linear block codes follow strong algebraic structure.

2) Cyclic codes : Cyclic codes belong to an important subclass of linear block codes. Cyclic codes are very easy to encode. They possess a well-defined mathematical structure which has led to the development of very efficient decoding schemes for them. They are used not only for error detection, but also for error correction, since they are easily implementable.

The Bose, Chaudhuri and Hocquenghem (BCH) codes, an important family belonging to cyclic codes were discovered by Hocquenghem (1959) and independently by Bose and Chaudhuri(1960). BCH codes have capabilities for multiple (burst) error detection and correction. Hardware (Circuit) implementation of BCH codes is easy and practically feasible.

3) Convolutional codes : The most useful tree codes are highly structured codes called convolutional codes. A convolutional encoder operates on the incoming message sequence continuously in a serial manner. Decoding of convolutional codes using probabilistic concepts and methods is rather difficult a job -but it is employed in situations where no other coding has better performance.

§ 1.4 : SCOPE OF THE CURRENT WORK :

In the current work –'Study and Performance Analysis of Error Control Codes', Error Correcting Codes or Error Control Codes have been studied under three main groups namely the Linear Block Codes, Cyclic Codes and Convolutional Codes. The study has incorporated a detailed investigation of the underlying concepts behind the formation and construction of codes belonging to these groups, their representation, error-correction capabilities and encoding and decoding algorithms along with some other related topics. Some special cases or codes, have also been studied in this purpose.

In the performance analysis part, these groups of Error Control Codes have been considered from encoding and decoding point of view. For example, the cyclic codes, which has been treated as a different family from linear codes can be dealt with methods applicable for linear block codes. Some computer programs have then been developed using high level programming languages like Pascal and C to consider the encoding and decoding performances of these codes. The performances are measured in terms of timing requirements for these operations. As such, upon observation, some recommendations have been made.

§ 1.5 : ORGANIZATION OF THE THESIS :

The thesis has been organized to cover its scope in a chronological manner suitable for a clear understanding of the ideas and concepts presented. Each group of the Error Correcting Codes or Error Control Codes covered by this work has been discussed in a separate chapter. Three chapters beginning from chapter 2 have dealt with three groups of Error Control Codes namely Linear Bloc Codes, Cyclic Codes and Convolutional codes. The theoretical backgrounds behind these families of Error Control Codes, the formation and construction of the codes, their representation, error-correction capabilities and encoding and decoding algorithms along with some other related topics have therefore been discussed in each of these chapters.

The next chapter deals with the performance study of some Error Control Codes. This follows development of some computer programs and consideration of the output thus generated. The timing requirements for encoding or decoding have been carefully examined from a maximum yield point of view. As such, some recommendations have also been made to ease the way of working with these codes. Lastly, some suggestions have been made to act as an aid for further work in related fields.

§ 1.6 : SURVEY OF LITERATURE :

The design of any project must be aided with sufficient references- the documents of previous work in the related field. The initiation of this work and its development in later stages has therefore been aided by different books and journals. Among them, the ones which influenced the course of this work are to be mentioned in this section.

Error Control Coding has long been a consideration of information science and engineering or data communications. Therefore, literatures dealing with these fields have often focused on Error Control Coding. Again, some publications extensively deal with Error Control Codes.

The book of Richard E. Blahut [1], that deals with Error Control Codes from theoretical point of view along with their practical implications has been used all throughout to guide the current work at times of need. Here, the underlying concepts and assumptions behind different families of codes have been presented in a comprehensive manner. As such, the theoretical study during the current work has been aided greatly. In discussing different groups of codes, the concepts presented in [1] has been employed in several places.

Other than [1], study of linear block codes have also followed the book of Michelson et al [3]. The formation of encoding and decoding matrices in different cases has been presented there to provide sufficient insight into the matter. The works of Fossorier et al [10] and G. Caire et al [11] has been helpful in studying linear block codes.

Working with cyclic codes has been aided again to a great extent by [1]. The basics of cyclic codes along with their polynomial representations, matrix form etc. are covered in this. Lucky, Salz and Weldon [6] has also been helpful in this regard. Cyclic property, cyclic shifting, decoding etc. are presented here which have been employed in forming concepts of the current work.

The work on convolutional codes has made extensive use of Robert Gallager [7]. Here the basic construction of such codes, their polynomial equivalents, matrix fromation etc. are covered thoroughly. Simon Haykin[2], has also served important purpose in this regard. Important constructs for convolutional codes like tree, signal flow graph or state diagram and trellis diagram have been presented here. Decoding of convolutional codes using Viterbi algorithm and relation of decoding ability with minimum distance has also been available here.

Many other works and publications have been consulted at times of need during the current work. They have been acknowledged in the reference.

CHAPTER TWO : LINEAR BLOCK CODES

D

§ 2.1 : INTRODUCTION :

A *block code* represents a block of k information symbols by an n symbol codeword .

A code is said to be *linear* if any two code words in the code can be added (in modulo 2 arithmetic -obviously for binary system) to produce a third code word of the code . This basic property of linear block code is called *closure* .

2.1.1 : Parity Check Codes :

Linear codes are parity check codes and almost all the block codes used in error control system belongs to this class .

Parity check block codes can be defined as follows:

The encoder accepts k information digits from the information source and appends a set of parity check digits, which are derived from the information digits in accordance with a prescribed encoding rule. The encoding rule determines the mathematical structure of the code. The information and parity digits are transmitted as a block of $n = k+r$ digits on the communication channel.

It is customary to call the code an (n,k) block code. The n bit block is called the code block or codeword, and n is called the block length of the code. For code rate $R=\frac{k}{n}$, a parity check code is a particular type of mapping from binary sequences of length k into sequences of some longer length n.

2.1.2 : Systematic Code :

A *systematic code* is one that starts each code word with the information symbols unmodified. The remaining symbols are parity check symbols. For applications requiring both error detection and error correction , the use of systematic block codes simplifies implementation of the decoder. The use of non-systematic block code is generally avoided , since encoding and decoding involve the steps of converting the information vectors to and from the nonsystematic code vectors. But , in case of systematic code in the receiving side the information vector can be easily extracted from the codeword without reordering the symbols in the codeword. But for convolution code though the systematic code has in general smaller hamming distance , it is preferred over nonsystematic code as it is non catastrophic .

Let , m_0 , m_1 , m_2 , ... , m_{k-1} constitute a block of k arbitrary message bits. Thus we have 2^k distinct message blocks. Let this sequence of message bits be applied to a linear block encoder , producing an n bit code word whose elements are denoted by c_0 , c_1 , ..., c_{n-1}. Let b_0 , b_1 , ... , b_{n-k-1} denote the $n-k$ parity bits in the codeword. For the code to possess a systematic structure , a codeword is divided into two parts , one of which is occupied by the message bits and the other by parity bits. The message bits of a codeword can be sent before the parity bits or after the parity bits . The later option is used here .

$b_0 , b_1 , \dots , b_{n-k-1}$	$m_0 , m_1 , m_2 , \dots , m_{k-1}$
Parity Symbols	Message Symbols

Parity bits are also called check bits or parity check bits .

According to the representation of this figure , the $(n-k)$ leftmost bits of a code are identical to the corresponding parity bits , and the k rightmost bits of the code word are identical to the corresponding message bits . We may therefore write

$$c_i = b_i \quad \text{for } i = 0,1, \dots , n-k-1$$
$$= m_{i+k-n} \quad \text{for } i = n-k , n-k+1 , \dots , n-1 \qquad (2.1)$$

The $(n-k)$ parity bits are linear sums of the k message bits as shown by the generalized relation

$$b_i = p_{0i} . m_0 + p_{1i}\, m_1 + \dots + p_{k-1,i}\, m_{k-1} \qquad (2.2)$$

Where the coefficients are defined as follows

$$p_{ji} = 1 \quad \text{if parity bit } b_i \text{ depends on message bit } m_j$$
$$= 0 \quad \text{otherwise} \qquad (2.3)$$

The coefficients p_{ji} are chosen in such a way that the rows of the generator matrix are linearly independent and the parity equations (defined by 2.2) are unique that is no two or more b_i for different i's are exactly alike .

Equations (2.1) and (2.2) defines the mathematical structure (mathematical description of codeword bits and parity bits) of linear block code .

§ 2.2 : ALGEBRAIC BACKGROUND :

Three algebraic systems that are fundamental to the theory of error detecting and correcting codes,

they are groups , fields and vector spaces . A *linear block code* can be defined as a set of vectors in an n dimensional vector space over a finite field .This description is related to the generator matrix and parity check matrix of a code .

The number of possible received sequences are exponentially increasing function (for a q-ary system it is q^n) of n (block length) . Thus for large n , it is impractical to store all the possible codewords in the encoder and to store the mapping from received sequences to messages in the decoder . To avoid this storage problems algorithms are provided to generate codewords from messages (**c=iG**) and messages from received sequences using generator matrix **G** in encoding and parity check matrix **H** in decoding .

2.2.1 : Matrix Notation :

The system of equations (2.1) (code word bit definition) and (2.2) (parity bit definition) may be written in a compact form using matrix notation . Let us define a 1 by k message or information vector **m** or **i** , the 1 by (n-k) parity vector **b** , and the 1 by n code vector **c** as follows :

$$\mathbf{m} = \mathbf{i} = [\ m_0, m_1, \dots, m_{k\text{-}1}\ ] = [\ i_0, i_1, \dots, i_{k\text{-}1}\ ] \qquad (2.4)$$

$$\mathbf{b} = [\ b_0, b_1, \dots, b_{n\text{-}k\text{-}1}\ ] \qquad (2.5)$$

$$\mathbf{c} = [\ c_0, c_1, \dots, c_{n\text{-}1}\ ] \qquad (2.6)$$

All the above three vectors are row vectors . Column vector forms can also be used . But row vector form is most commonly used. Equation (2.2) defining the parity bits can be written in compact matrix form

$$\mathbf{b} = \mathbf{m} \,.\, \mathbf{p} \qquad (2.7)$$

Where with the help of equation (2.1) and (2.2) **p** is the k by (n-k) coefficient matrix defined by

$$\mathbf{p} = \begin{bmatrix} p_{00} & p_{01} & \cdots & p_{0,n-k-1} \\ p_{10} & p_{11} & \cdots & p_{1,n-k-1} \\ \cdots & \cdots & \cdots & \cdots \\ \mathrm{p}_{k-1,0} & \mathrm{p}_{k-1,1} & \cdots & p_{k-1,n-k-1} \end{bmatrix} \qquad (2.8)$$

Where p_{ij} is 1 or 0.

From the definitions given in equations (2.4) to (2.6) , we see that **c** may be expressed as a partitioned row vector in terms of the vector **m** and **b** as follows :

$$\mathbf{c} = [\mathbf{b} \mid \mathbf{m}] \tag{2.9}$$

Substituting equation (2.7) in equation (2.9) and factoring out the common message vector **m** , we get

$$\begin{aligned}\mathbf{c} &= [\mathbf{mp} \mid \mathbf{m}] \\ &= \mathbf{m}[\mathbf{p} \mid \mathbf{I}_k]\end{aligned} \tag{2.10}$$

Where $\mathbf{I}_k$ is the k by k identity matrix :

$$\mathbf{I}_k = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & 0 \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{2.11}$$

Let , $\mathbf{G} = [\mathbf{p} \mid \mathbf{I}_k]$ (2.12)

This k by n matrix **G** is called the generator matrix . The generator matrix **G** is said to be in *echelon canonical form* in that its k rows are linearly independent ; that is it is not possible to express any row of the matrix **G** as a linear combination of the remaining rows , which is clear from the presence of identity matrix in **G** . Using the definition of the generator matrix **G** , we may simplify equation (2.10) as

$$\mathbf{c} = \mathbf{mG} = \mathbf{iG} \tag{2.13 }$$

The full set of codewords referred to simply as the code , is generated in accordance with equation (2.13) by letting the message vector **m** or **i** range through the set of all 2^k binary k tuples (1 by k vectors).

2.2.2 : Group :

The most basic algebraic structure used in specifying the properties of error control codes is a group.

A *group G* is a set of elements and an operation (*) defined on pair of elements in the set specifying four properties :

Closure : The set G is closed under the operation $*$. That is , if a and b are in G then $a*b$ is also in G.

Associativity : The associative law of ordinary arithmetic holds for $*$. That is , for any a, b , c in G $(a*b)*c=a*(b*c)$.

Identity : G has an identity element e that satisfies $a*e=e*a=a$, for every a in G.

Inverses : If a is in the set , then there is some element b in the set called an inverse of a such that $a*b=b*a=e$.

Here the operation * can be '+' (addition) or '•' (Multiplication). But one has to note that '+' and '•' are not ordinary addition and multiplication. A special form of arithmetic called modulo arithmetic is employed for this purpose.

For '+' the identity element is called 0 .
For '•' the identity element is called 1 .
For '+' the inverse element of a is written as $-a$, therefore , $a+(-a)=(-a)+a=0$.
For '•' The inverse element of a is written as a^{-1} , so that $a \bullet a^{-1} = a^{-1} \bullet a = 1$.

A group may have finite or infinite number of elements. If G has a finite number of elements then it is called a *finite group* .The number of elements in G is called the *order* of G .

Some groups satisfy the additional property for all a , b in the group $a*b=b*a$. This is called the commutative property . Groups with this additional property are called *commutative groups* or *abelian groups*.

A subset S of elements in a group G is called a *subgroup* of G if S itself is a group with respect to the operation defined on G .

If h is element of G, then h is said to form a cyclic subgroup of G if the elements of the cyclic subgroup are powers of h, i.e. h^1, h^2, h^3, h^4,........, $h^c=1$. Then c is called the period of the cyclic subgroup.

2.2.3 : Field :

A field is described simply as a set of elements with two operations defined , addition (+) and multiplication (•) .Two further operations , subtraction and division , are implied by the existence of inverse elements under each of the defining operations.

The elements in a field F, taken together with the operations + and • , must satisfy the following conditions :

1. F is closed under the two operations , that is the sum or product of any two elements in F is also in F.

2. For each operation , the associative and commutative laws of ordinary arithmetic holds , as that for any elements u,v, and w in F , $(u+v)+w=u+(v+w)$, $u+v=v+u$, $(u \bullet v) \bullet w=u \bullet (v \bullet w)$, $u \bullet v=v \bullet u$.

3. Connecting the two operations , the distributive law of ordinary arithmetic holds , so that $u \bullet (v+w)=u \bullet v+u \bullet w$,for any u,v, w in F.

4. F contains a unique additive identity element 0 and a unique multiplicative identity , different from 0 and written as 1 , such that $u+0 = u$, $u \bullet v = u$, For any element u in F , the two identity elements are the minimum that any field must contain .This two elements are simply called zero and unity .

5. Each element u in the field has a unique additive inverse , denoted by $-u$, such that $u+(-u)=0$. And , for $u \neq 0$ a unique multiplicative inverse denoted by u^{-1}, such that $u \bullet u^{-1}=1$, from the inverse operations subtraction (-) and division(÷) are defined by

$u-v = +(-v)$, for any u,v in F
$u \div v = u \bullet (v^{-1})$, $v \neq 0$.

Where $-v$ and v^{-1} are the additive and multiplicative inverses , respectively ,of v .

A field is an abelian (commutative) group under addition .The field is closed under multiplication and the set of nonzero elements is an abelian group under multiplication.

The number of elements in a field , called the order of the field , may be finite or infinite . A field having a finite number of elements is called a finite field or *Galois field* and is denoted by $GF(q)$, where q means the number of elements in the field and GF means Galois field .

The simplest finite field is $GF(2)$, which contains only the zero (0) and unity element (1) .

2.2.4 : Vector Spaces :

Let F be a field. The elements of F will be called *scalars*. A set V is called a *vector space* over a field F with a set of elements, called vectors, if there is defined an operation called *vector addition* (denoted by +) on pairs of elements from V, and an operation called *scalar multiplication* (denoted by juxtaposition) on an element from F and an element from V to produce an element from V provided the following axioms are satisfied.

1. V is an abelian group under vector addition.
2. The distributive laws apply , so that $a(\mathbf{u} + \mathbf{v})=a\mathbf{u}+a\mathbf{v}$ and $(a+b)\mathbf{u}=a\mathbf{u} + b\mathbf{u}$ .where **u** and **v** are vectors and *a and b* are scalars.
3. The associative law applies , so that $(ab)\mathbf{u}= a(b\mathbf{u})$.
4. For the multiplicative identity element 1 in F , $1\mathbf{u}=\mathbf{u}$.

There are two identity element to deal with here. Since the vectors in V forms a commutative group under addition , there is an identity element , called the zero vector and written as **0** . such that $\mathbf{v}+\mathbf{0}=\mathbf{0}+\mathbf{v}$ for all vectors **v** in V. This is not same as 0 , the zero scalar element in the field F.

Let us consider an ordered set of n elements u_1 , u_2 , ..., u_n , where each element u_i belongs to the field F . This is called an n-tuple (vector) over F . Let us , define the addition of any two n tuples as the element by element addition

$$(u_1, u_2,..,u_n) + (v_1 . v_2 ,..., v_n) = (u_1+v_1, u_2+v_2, ...,u_n+v_n)$$

Where each addition u_i+v_I is performed if F.

Due to the closure property of F , the addition of two n-tuples over F produces another n-tuples over F.

Now , the multiplication of an element from F by an n-tuple over F is defined as the element by element multiplication

$$a(u_1, u_2,...,u_n)=(au_1 , au_2 , ... , au_n)$$

Where each multiplication au_i is done in F . The result is again an n-tuple over F . And if a =1 the result of the multiplication is simply the original n-tuple itself .

The addition and multiplication rules for vectors satisfy the distributive and associative laws (condition 1 and 2) , and the set of all n- tuples over F , taken together with the operations of addition and multiplication in F , constitutes vector space over F , with the all -zeros n-tuple or zero vector being the additive identity in the additive group of the vector space .

We will consider *vector spaces over finite field* , where the *scalars* represent *code symbols* and *vector* (*n* -*tuples*) represent *codewords* .

2.2.5 : Linear Operations in a Vector Space Over a Field :

2.2.5.1 : Linear combination :

If $\mathbf{v}_1 , \mathbf{v}_2 , \dots , \mathbf{v}_k$ are vectors in a vector space V over a field F , then a linear combination of $\mathbf{v}_1 , \mathbf{v}_2 , \dots , \mathbf{v}_k$ is any sum of the form

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \dots + a_k\mathbf{v}_k \qquad (2 .14)$$

Where each a_i is in the field F . The given set of vectors { $\mathbf{v}_i$ } is said to span the vectors in V if any vector in V can be generated by a linear combination of them .

2.2.5.2 : Linear independence :

A set of k vectors $\mathbf{v}_1 , \mathbf{v}_2 , \dots , \mathbf{v}_k$ is said to be linearly independent if no set of scalars $a_1 , a_2 , \dots, a_k$ (except all $a_i = 0$) exists such that

$$a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \dots + a_k\mathbf{v}_k = \mathbf{0} \qquad (2.15)$$

0 denotes zero vector .

If equation (2.15) can be satisfied for at least one set of scalars not all equal to zero , the vectors $\mathbf{v}_1 , \mathbf{v}_2 , \dots , \mathbf{v}_k$ are said to be linearly *dependent* .

For example , the vectors (1,0,0) , (0,1,0) , and (0,0,1) are linearly independent over any field . However the vectors (1,1,0) , (0,1,1) and (1,0,1) are linearly dependent over GF(2) because they sum to the zero vector .

2.2.5.3 : Vector subspace or subspace :

A subset S containing at least one vector in a vector space V is called the subspace of V if S itself has all the properties of a vector space with respect to the operations of addition and scalar multiplication in V . For example , if we take subset of vectors $\mathbf{v}_1 , \mathbf{v}_2 , \dots , \mathbf{v}_r$ from V over F , the set S of all vectors formed by linear combinations of $\mathbf{v}_1 , \mathbf{v}_2 , \dots , \mathbf{v}_r$ over F constitutes a vector subspace .

proof :

1. Closure property is satisfied , that is for any vector $\mathbf{v}_a$ and $\mathbf{v}_b$ in S , $\mathbf{v}_a$ + $\mathbf{v}_b$ is in S . We let

$$\mathbf{v}_a = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + \dots + a_r\mathbf{v}_r \text{ and}$$
$$\mathbf{v}_b = b_1\mathbf{v}_1 + b_2\mathbf{v}_2 + \dots + b_r\mathbf{v}_r \quad \text{so that}$$
$$\mathbf{v}_a + \mathbf{v}_b = (a_1+b_1)\mathbf{v}_1 + (a_2+b_2)\mathbf{v}_2 + \dots + (a_r+ b_r)\mathbf{v}_r$$

$= c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + ... + c_r\mathbf{v}_r$.

Because of the closure property of F under addition , the scalars c_1 , c_2 , ..., c_r are in F , and therefore $\mathbf{v}_a + \mathbf{v}_b$ is in S .

2. For any b in F and $\mathbf{v}$ in S , $b\mathbf{v}$ is in S . Let , $\mathbf{v} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + ... + a_r\mathbf{v}_r$, then using distributive and associative laws , $b\mathbf{v} = ba_1\mathbf{v}_1 + ba_2\mathbf{v}_2 + ... + ba_r\mathbf{v}_r = c\mathbf{v}_1 + c\mathbf{v}_2 + ... + c_r\mathbf{v}_r$. Because of the closure property of F under multiplication $c_1 , c_2 , ... , c_r$ in F and therefore $b\mathbf{v}$ is in S .

$0\mathbf{v}=\mathbf{0}$ for all $\mathbf{v}$ in V.
$a\mathbf{0}=\mathbf{0}$ for all a in F.
So, S (a vector subspace) must contain the zero vector .

Each vector $\mathbf{v} = a_1 \mathbf{v}_1 + a_2\mathbf{v}_2 + ... + a_r\mathbf{v}_r$ in S has an additive inverse given by $-\mathbf{v} = (-a_1)\mathbf{v}_1 + (-a_2)\mathbf{v}_2 + ... + (-a_r)\mathbf{v}_r$, where $-a_i$ is the additive inverse of a_i in S .

The commutative , associative , and the distributive laws carry over from V into S , since the vectors in S are contained in V . So , S is a subspace of V .

The set of vectors that generate a vector space V by linear combinations is said to span V . The same terminology applies to any subspace S of V . For example , the vectors $\mathbf{v}_1$, $\mathbf{v}_2$, ... , $\mathbf{v}_r$ are said to span the subspace S .

Example :

Let us consider the three binary vectors (1,0,0) , (0,1,0) and (0,0,1). Let these vectors span the vector space V_3 , composed of all eight binary vectors , (0,0,0) , (0,0,1) , ... , (1,1,1) formed by the linear combinations of the three spanning vectors over GF(2) . If we use the two vectors (1,0,0) and (0,1,0) to form linear combinations over GF(2) , the resulting vectors (0,0,0) , (1,0,0) , (0,1,0) and (1,1,0) constitute vector space , let V_2 , which is a subspace of V_3. The vectors (1,0,0) and (0,1,0) are said to span V_2.

2.2.5.4 : Basis of a vector space :

In any vector space there is at least one set of linearly independent vectors that spans the space . Any such set is said to be a basis of the vector space . in the example of V_3 , the spanning vectors (1,0,0) , (0,1,0) and (0,0,1) are linearly independent and therefore constitute a basis of V_3 .

A set of vectors having 1 in one position and 0 in all other positions , no two vectors having 1 in the same position - such vectors are called unit vectors , and they are linearly independent and can form a basis. however the unit vectors are not the only basis vectors . It can be shown for V_3. For example (1,0,0) , (0,1,0) , and (0,1,1) also form a basis of V_3. These three vectors are linearly independent and they generates all vectors in V_3 (or span V_3) .V_2 has basis (1,0,0) and (0,1,0) . The vectors (1,0,0) and (1,1,0) form an alternative basis of the subspace V_2.

Vector (1,1,1) is a basis for a subspace , say V_1 , containing only the vectors (0,0,0) and (1,1,1) .

2.2.5.5 : Dimension :

All bases of a given vector space contain the same number of vectors , the number is called the *dimension* of the vector space . Therefore the *dimension* of a vector space is the number of vectors in any linearly independent set of vectors that can be used to generate the space by forming linear combinations (spans the space). In the above examples V_3 , V_2 , V_1 have dimensions 3,2,1 respectively .

So a vector space composed of *n*-tuples need not have dimension *n* , but may instead have dimensions less than *n* . For example , the vector space generated by the three 7 tuples (0,0,0,1,1,1,1) , (0,1,1,0,0,1,1) and (1,0,1,0,1,0,1) over GF(2) can be stated . The linear independence of the vectors is easily verified by noting that the elements in the first , second, and fourth positions of any linear combination cannot be all zero unless all three vectors are given the scalar multiplier zero . Therefore the three vectors generate a vector space of dimension 3 and this is a subspace of the vector space of all 7 tuples over GF(2) .

2.2.5.6 : Matrix representation of a vector space :

It is convenient to write the linear combinations of basis vectors in a shorthand matrix notation .Matrices can be added and multiplied . This operations obey the distributive and associative laws of ordinary arithmetic , however matrix multiplication is not commutative in general .

A linear combination **c** of the three 7 tuples is written as

$$\mathbf{c} = a_1(0,0,0,1,1,1,1) + a_2(0,1,1,0,0,1,1) + a_3(1,0,1,0,1,0,1)$$

$$= (a_1\ a_2\ a_3)\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \quad \text{(each } a_i \text{ is a scalar)}$$

We write this more compactly as **c=aT**

The matrix multiplication of the 1 × 3 row vector **a** times the 3 ×7 matrix **T** yields a 1 ×7 row vector **c** . This matrix multiplication constitutes a linear transformation of a 3-tuple **a** into a 7-tuple **c** , where the nature of the transformation is determined by the elements of **T** . That is the matrix **T** maps 3 -tuples into 7-tuples . Since the rows of **T** are basis vectors (as linearly independent) in a space of dimension 3 (3 rows ,so three basis vectors , so dimension is 3), if the three tuples **a** are allowed to take on all eight (2^3) binary combinations , the transformation generates the entire vector space spanned by the rows of **T** (that is ,vector space formed by the linear combinations of the rows (vector) of **T**) . To generalize this , a $k \times n$ matrix **T** having k linearly independent rows generates a k - dimensional vector space C_k , where the vectors in C_k forms a subspace (contains some of the 2^n combinations of n symbols over the scalar field) of the vector space of all n-tuples over the scalar field .

2.2.6 : General Mathematical Framework For The Binary Block Codes Or Block Codes Over GF(2) :

An (n,k) linear block code over a field F is a k dimensional vector subspace of the space of all n-tuples over F , where $k < n$. The term linear refers to the formation of a vector space by linear combinations over GF(2) of k linearly independent basis vectors $\mathbf{g}_1$, $\mathbf{g}_2$, ... , $\mathbf{g}_k$ (each of these k vectors is an n-tuple) is a binary (n,k) linear block code C (vector subspace) . That is a vector **c** is a codeword (vector or a n-tuple) in C if and only if it lies or contained in the k dimensional vector space spanned by the $\{\mathbf{g}_i\}$, equivalently , if the $\{\mathbf{g}_i\}$ are arranged as rows of a $k \times n$ matrix **G** (generator matrix), a codeword **c** can be expressed as

$$\mathbf{c} = (i_1 , i_2, ..., i_k)\begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \vdots \\ \mathbf{g}_k \end{bmatrix} = \mathbf{iG} \,. \tag{2.16}$$

Where **i** is a k bit information vector and **G** is called the generator matrix of the code . The full set of codewords , referred to simply as the code , is generated by letting **i** range through all 2^k binary k tuples .

We know vector space (example : code) is a set of elements called vectors (each codeword is also a vector in this space) and vector space is an abelian group (commutative group) under vector addition . Since the code is a vector space , by definition (given above) the 2^k codewords constitute a group under vector addition .This is clear in coset decomposition and standard array . In fact it is possible to define a binary (n,k) linear code as a subgroup (under vector addition ,as the code is a vector subspace, and a vector subspace is a group) of order 2^k (number of codewords in a code) taken from the group of all binary n-tuples , therefore binary linear block codes are sometimes called group codes .

Certain obvious properties of linear block codes follow from the group property : the sum of two codewords is a codeword (closure) and every linear code as a vector subspace contains a zero vector , the all zero n-tuples (from part 2 of the proof related to vector subspace).

These properties are readily seen from equation (2.16) . Since **0.G=0** and , for any two codewords $\mathbf{c}_j$ and $\mathbf{c}_k$ in C (code) the sum of $\mathbf{c}_j$ and $\mathbf{c}_k$ is $\mathbf{c}_j + \mathbf{c}_k = \mathbf{i}_j\mathbf{G} + i_kG = (\mathbf{i}_j + \mathbf{i}_k)\mathbf{G} = \mathbf{i}_l\mathbf{G}$. The modulo 2 sum of $\mathbf{c}_j$ and $\mathbf{c}_k$ represents a new codevector (code word) .

The row space of a matrix **A** is the set of all linear combinations of the row vectors of **A**, the dimension of the row space is called the row rank - that is the number of rows of **A** which are linearly independent .Let

$$\mathbf{i} = [\,1 \;\; 1 \;\; 1\,]$$

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{a_0} \\ \mathbf{a_1} \\ \mathbf{a_2} \end{bmatrix}$$

$$\begin{aligned} \therefore\ \mathbf{i\,A} &= 1(1,0,0) + 1(0,1,0) + 1(0,0,1) \\ &= [\,(1+0+0) \;\; (0+1+0) \;\; (0+0+1)\,] \\ &= [1 \;\; 1 \;\; 1\,] \\ &= \text{one member of the row space of } \mathbf{A}. \end{aligned}$$

$\mathbf{a}_0$, $\mathbf{a}_1$,$\mathbf{a}_2$ are row vectors of the matrix **A** . Here , all the 3 row vectors of **A** are linearly independent , that is the dimension (number of basis vectors) of the row space of **A** is 3 , i.e. the row rank of **A** is 3 . The vector space consisting of all vectors formed by **iG** (by varying **i**) is called the row space (if all rows of **G** are linearly independent and **i** has *k* symbols then the row space of the matrix **G** has 2^k elements) .

$$\mathbf{c} = \mathbf{iG} = \mathbf{i}[\mathbf{P}:\mathbf{I}_k] \tag{2.17}$$

P determines how the parity check bits are related to the information bits .This is one representation of **c**.

The generator matrix

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_1 \\ \mathbf{g}_2 \\ \vdots \\ \mathbf{g}_k \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1 & 1 & 0 & 0 & \cdots & 0 \\ \mathbf{p}_2 & 0 & 1 & 0 & \cdots & 0 \\ \mathbf{p}_3 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{p}_k & 0 & 0 & 0 & \cdots & 1 \end{bmatrix} = [\mathbf{P} \mid \mathbf{I}_k] \qquad \text{(stated before hand)}$$

is a generator matrix for an arbitrary (n,k) systematic (because the matrix has the form in which the existence of $k \times k$ identity matrix is readily obvious) linear block code , where $\mathbf{p}_j$ denotes the *j* th row in the $k\times(n\text{-}k)$ matrix **P** . Since every row in *G* is in the vector space that defines the code , every row of *G* is itself a codeword . It is seen that the *j* th row of *G* is the codeword that corresponds to an information vector having a one in the j th position and zero elsewhere . It is clear from the form of encoding transformation c=iG that a codeword with an arbitrary k bit information vector is obtained by forming a linear combination of the vectors $\{g_i\}$. Specifically the codeword associated with an information vector i is the sum of the rows of G whose row numbers correspond to positions of ones in i .

The relationship of parity and information bits can be written as in (2.17) but can be written in another useful way . For example , let $r = n-k$ and consider the $r\times n$ matrix **H** given by $\mathbf{H} = [\mathbf{I}_r : -\mathbf{P}^T]$ where $\mathbf{P}^T$ is the transpose of the parity matrix **P** . $\therefore$ $\mathbf{P}^T$ is an $r\times k$ matrix , that is an $(n-k)\times k$ matrix . $\mathbf{I}_r$ is the $r \times r$ or $(n-k) \times (n-k)$ identity matrix .

By multiplication of partitioned matrices $\mathbf{GH}^T = [\mathbf{P} : \mathbf{I}_k]\begin{bmatrix}\mathbf{I}_r \\ -\mathbf{P}\end{bmatrix} = \mathbf{P}- \mathbf{P} = \mathbf{0}$. For GF(2) **P** = **-P** . Here **0** represents the $k \times r$ or $k \times (n-k)$ all zero matrix . Therefore we have ,

$$\mathbf{cH}^T = \mathbf{iGH}^T = \mathbf{0} . \tag{2.18}$$

As $\mathbf{GH}^T = \mathbf{0}$. The matrix **H** is the parity check matrix of the code . The set of all vectors **v** such that $\mathbf{Av}^T = \mathbf{0}$ for matrix **A** , is called the null space of the matrix **A** . And the equations specified by equation (2.18) are the parity check equations . $\mathbf{cH}^T = \mathbf{0}$. And as $\mathbf{c}^T$ and **H** can be multiplied Considering dimension of them it is clear . By transposing $(\mathbf{cH}^T)^T = \mathbf{Hc}^T = \mathbf{0}$, this zero matrix is an $r\times k$ or $(n-k) \times k$ matrix. So, we can say that the code vectors or codewords in a code are in the null space of the parity check matrix H. And as the code generated by **G** is the row space of **G** (described before hand) , we can say that the row space of **G** is in the null space of **H** .

Information vectors may be transformed into codewords by the operation **c=iG** (encoding rule - where **c** is the code word for message or information vector **i** and **G** is the generator matrix for the code) , which forms the sum of rows of the generator matrix .

Techniques (c=iG) equivalent to the use of G are usually employed for encoding , while the parity check relationships defined by parity check matrix H are typically used in decoding .

§ 2.3 : IMPORTANT THEOREMS, CONCEPTS, AND CODES :

2.3.1 Theorem : If a vector space V is spanned by a finite set of k vectors $A = \{ \mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_k \}$ and V contains a set of m linearly independent vectors (V may have more than k vectors but in a vector set of linearly independent vectors in V has m vectors) $B = \{ \mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_{m-1} \}$ then $k \geq m$.

proof :

We will describe how to construct a sequence of sets $A_0, A_1, ..., A_m$ in such a way that each set spans V. Each set has k elements chosen from A and B and the set A_r contains $u_1, ..., u_r$. Consequently, A_m contains $u_1, ..., u_m$ among its k elements, and thus $k \geq m$. All vectors in A are not necessarily linearly independent, and also B may not contain all the linearly independent vectors in V.

Because no nonzero linear combination of vectors of B is equal to zero, no element of B can be expressed as a linear combination of the other elements of B. If the set A_{r-1} does not contain u_r , and it spans V, then it must be possible to express u_r as a linear combination of elements of A_{r-1} including at least one vector of A (say $\mathbf{v}_j$) not in B. The equation describing the linear combination can be solved to express $\mathbf{v}_j$ as a linear combination of $\mathbf{u}_r$, and the other elements of A_{r-1}, so that afterward we can eliminate $\mathbf{v}_j$ from A_{r-1} and can place $\mathbf{u}_r$ to form A_r.

The construction is as follows. Let , $A_0 = A$, that is no elements of B are in A. If A_{r-1} contains $\mathbf{u}_r$ then let $A_r = A_{r-1}$. Otherwise, $\mathbf{u}_r$ does not appear in A_{r-1} but can be expressed as a linear combination of the elements of A_{r-1} involving some element $\mathbf{v}_j$ of A not in B. We form A_r from A_{r-1} by replacing $\mathbf{v}_j$ by $\mathbf{u}_r$. Any vector of V that is a linear combination of vectors in A_{r-1} and contains $\mathbf{v}_j$ in the linear combination can also be expressed as the linear combination of vectors of A_r (though A_r does not contain $\mathbf{v}_j$) as in the linear combination $\mathbf{v}_j$ can be expressed by linear combination of $\mathbf{u}_r$ and other vectors of A_r.Therefore as A spans V , A_r that is a different form of A also spans V. Therefore we starts with a set A with no elements from B, and sequentially at first,we replace one element of A that is not in B but related to $\mathbf{u}_1$, by $\mathbf{u}_1$ and we get A_1, then we replace an element of A_1, that is not in B and can be expressed in terms of $\mathbf{u}_2$ of B by $\mathbf{u}_2$, thus we get A_2. In these ways we can construct A_m, with m elements among k elements of A replaced by m elements of B. Thus the proof is complete and $k \geq m$.

2.3.2 Theorem :Two linearly independent sets of vectors that span the same finite dimensional vector space have the same number of vectors :

proof:

Let the two linearly independent sets of vectors are B and A as stated in theorem 2.3.1. So, B spans V. Let B has m vectors and A has k vectors. By theorem 2.3.1 $k \geq m$. As both the sets span V. Changing role of A and B we get by the same theorem that $m \geq k$. As $m \geq k$ and $k \geq m$, and to satisfy both of these conditions $m = k$.

2.3.3 Theorem : Any nonsystematic linear block code is equivalent to a systematic code :

proof:

A set of vectors can be generated from the linear combinations of the rows of generator matrix **G.** If two generator matrices have same row space (set of vectors formed by linear combination of rows) then they produce the same set of codewords (because set of codewords is nothing but the row space of the generator matrix) though with different mapping (for the same information vector it varies depending on the rows of **G**) from information vector to codeword vector. Such generator matrices are called equivalent.

Given a non-systematic generator matrix $\mathbf{G}''$ for a linear block code, it is possible to convert $\mathbf{G}''$ to a systematic generator matrix **G** (The rows of $\mathbf{G}''$ are such that systematic generator matrix can be formed from it because as $\mathbf{G}''$ is used to produce code vector, the codewords obtained by using $\mathbf{G}''$ must contain the bits of information vectors though in unordered form, and after ordering, the whole information vector is readily visible in the codeword, and such form of codeword is available only from systematic generator matrix) by a straightforward procedure involving column interchange and elementary row operations on $\mathbf{G}''$. For this purpose we have to consider three types of elementary row operations for matrices :

1. Interchanging any two rows.
2. Multiplying a row by a nonzero scalar.
3. Adding a scalar multiple of one row to another row.

The procedure for converting $\mathbf{G}''$ to **G** is same as procedures for diagonalizing matrices in ordinary linear algebra (because systematic generator matrix contains identity matrix as a part of it, and an identity matrix is a diagonal matrix - i.e. matrix elements other than diagonal elements are all zeros.) If column interchange is not applied, only row operations are applied the same set ($L1$) of codewords are generated, but with a different mapping from information sequence to codeword is produced. If column interchange is also applied, then there will be rearrangement of bit positions in codewords (and the set of codewords thus produced would be different from $L1$ with respect to bit positions).

The code generated by the systematic generator matrix **G** must contain exactly the same set of codewords as does $\mathbf{G}''$, except for a possible rearrangement of bit positions in codewords. This can be seen as follows. Let the conversion from G'' to G be done in two steps G'' to G' and G' to G, where the first step consists of elementary row operations and the second step consists of column interchange. Obviously, the row operations of type 1 and 2 does not change the vector space. (because for type 1 operation , only the rows are interchanged, and as vector addition is commutative, the same set of codewords are produced after multiplying by information sequences. For type two operation it should be mentioned that , from the closure property of a vector space, we know if we multiply a vector by a scalar over the field under consideration, a vector is produced which is in the vector space, and the rows are still linearly independent as the operations are performed in modulo and as the rows of the original generator matrix is modulo. So the vector space is not changed, also as the rows in the new generator matrix after this operation are still linearly independent, the new rows are also in the basis set if the rows of the original matrix are each different and linearly independent, and by linear combination of the basis vectors the same vector space would be generated. Because by a theorem : in a k-dimensional vector space V, any set of k linearly independent vectors is a basis for V, that is, they produce the same vector space. Proof of this can be given as follows : Let $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3,..., \mathbf{v}_k\}$ be any set of k linearly independent vectors in V. If they do not span V, then one can find a vector v in V that is not a linear combination of $\mathbf{v}_1,\mathbf{v}_2, \mathbf{v}_3,..., \mathbf{v}_k$. The set $\{\mathbf{v}, \mathbf{v}_1, \mathbf{v}_2,..., \mathbf{v}_k\}$ is linearly independent and contains k+1 vectors in V, which contradicts the theorem that two linearly independent sets of vectors that span the same finite dimensional vector space have the same number of vectors. Therefore every set of k linearly independent vectors $\{ \mathbf{v}_1, \mathbf{v}_2,..., \mathbf{v}_k \}$ in V is a basis for V. For vectors over binary field and scalars from binary field, type 2 operation keeps the row concerned unchanged).

For type 3 operations, a row constructed in $\mathbf{G}'$ is a linear combination of the rows of $\mathbf{G}''$ (as $\mathbf{G}'$ is formed by elementary row operations on $\mathbf{G}''$) and thus contained in the vector space generated by $\mathbf{G}''$ (due to closure property). As each elementary row operation is inverted by an elementary row operation of the same kind - any type 3 operation (an elementary row operation) used in going from $\mathbf{G}''$ to $\mathbf{G}'$ can be undone by applying a type 3 operation to $\mathbf{G}'$ (that is $\mathbf{G}''$ can be obtained by multiplying $\mathbf{G}'$ by appropriate symbols in information sequence , that is by linear combination of rows of $\mathbf{G}'$, rows of $\mathbf{G}''$ is formed. In this way the rows of $\mathbf{G}''$ is contained in the vector space generated by $\mathbf{G}'$, proving that $\mathbf{G}''$ and $\mathbf{G}'$ generates the same vector space.

The interchanging of columns in going from $\mathbf{G}'$ to $\mathbf{G}$ (systematic generator matrix) can, of course, change the vectors in the vector space, however this is simply a reordering of bits in the codewords, which does not change the error correction power of the code (Because the error correction power depends on the minimum distance of the code, which is not changed by this operation).

2.3.1 : Inner Product And Orthogonal Complement :

Given a field F, the quantity $(a_1, a_2,...,a_n)$, composed of field elements, is called an n-tuple of elements from the field F. Under the operations of componentwise addition and componentwise scalar multiplication, the set of n-tuples of elements from a field F is a vector space and is denoted by the label F^n. Any finite dimensional vector space can be represented as an n-tuple space by choosing a basis $\{ \mathbf{v}_1, \mathbf{v}_2,..., \mathbf{v}_n \}$ and representing a vector $\mathbf{v} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 +...+ a_n\mathbf{v}_n$ by the n-tuple of coefficients $(a_1, a_2,...,a_n)$. Hence, we need consider only vector spaces of n-tuples.

The *inner product* of two n-tuples of F^n (the vector space over the field F) $\mathbf{u} = (a_1, a_2,..., a_n)$ and $\mathbf{v} = (b_1, b_2,..., b_n)$ is a scalar defined as

$$\mathbf{u} \cdot \mathbf{v} = (a_1,..., a_n) \cdot (\mathrm{b}_1,..., \mathrm{b}_n)$$

$= a_1b_1 + a_2b_2 + ... + a_nb_n = 0$ (scalar zero).

If the inner product of two vectors is zero they are said to be *orthogonal*. It is possible for a nonzero vector over GF(q) to be orthogonal to itself. As for example over GF(2), a vector or a two-tuple 11 is orthogonal to itself ,because $(1,1) \cdot (1,1)$ =(1+1) mod 2=0. A vector orthogonal to every vector in a set is said to be orthogonal to the set. The set of vectors orthogonal to the vector space W is said to be *orthogonal complement* of W and is denoted by $W^{\perp}$·

2.3.1.1 Theorem : A vector orthogonal to every vector of a set that spans (that is basis for) W is in the orthogonal complement of W.

proof:

In language description the basis vectors form the vectors of W by linear combination. So, a vector orthogonal to every vector of the basis set is in the orthogonal complement of W.

Let us assume that, the set { $\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_n$ } spans W. A vector $\mathbf{w}$ in the vector space W can be written in the form $\mathbf{w} = c_1\mathbf{w}_1 + c_2\mathbf{w}_2 + ... + c_n\mathbf{w}_n$, Then

$$\begin{aligned} \mathbf{w} \cdot \mathbf{u} &= (c_1\mathbf{w}_1 + c_2\mathbf{w}_2 + ... + c_n\mathbf{w}_n) \cdot \mathbf{u} \\ &= c_1\mathbf{w}_1 \cdot \mathbf{u} + c_2\mathbf{w}_2 \cdot \mathbf{u} + ... + c_n\mathbf{w}_n \cdot \mathbf{u} \end{aligned}$$

If $\mathbf{u}$ is orthogonal to every $\mathbf{w}_i$,
$\mathbf{w} \cdot \mathbf{u} = 0 + 0 + ... + 0 = \mathbf{0}$.
i.e. $\mathbf{u}$ is orthogonal to every $\mathbf{w}$ in W.

2.3.1.2 Theorem : If W, a subspace of a vector space of n-tuples has dimension k, then $W^{\perp}$, the orthogonal complement of W, has dimension n-k :

proof :

Let { $\mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_k$} be a basis for the vector subspace W, and we define the matrix $\mathbf{G}$ by

$$\mathbf{G} = \begin{bmatrix} \mathbf{g}_1 \\ \vdots \\ \mathbf{g}_k \end{bmatrix}$$

Where the basis vectors appear as rows. (The row space of a matrix **A** is the set of linear combinations of the row vectors of **A.** The dimension of the row space is called the *row rank* . The column space of **A** is the set of all linear combinations of column vectors of **A**, and the dimension of the column space is called the *column rank*. The set of vectors **v** such that $\mathbf{Av}^{\mathrm{T}} = \mathbf{0}$ is called the null space of the matrix **A**. Also a k by n matrix **A** whose k rows are linearly independent has k linearly independent columns, so the matrix has row rank = column rank = k.) The matrix has row rank

k, and the column space of $\mathbf{G}$ has dimension k so column rank is also k. A vector $\mathbf{v}$ is in $W^{\perp}$ (the orthogonal complement of the vector subspace W) if

$$\mathbf{G}\mathbf{v}^{\mathrm{T}} = \mathbf{0}\,.$$

The reason is, if a vector of the vector space W (here row space) produced by rows (here basis vectors) of a matrix (here $\mathbf{G}$) is multiplied with any vector from the orthogonal complement of the vector space W then a scalar zero is produced. Thus if, more than one vectors of the row space (here basis vector) are multiplied by a member of the orthogonal complement set, then a zero vector is produced. For multiplication , transpose of the vector of the orthogonal set is used.)

Let $\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_r\}$ be a basis for $W^{\perp}$. Let us extend this to a basis for the whole space $\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_r, \mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{n-r}\}$. Linear combinations of the columns of $\mathbf{G}$ (so every vector $\mathbf{u}$ in the column space of $\mathbf{G}$) is expressible as

$$\mathbf{u} = \mathbf{G}\mathbf{b}^{\mathrm{T}}$$

where vector $\mathbf{b}$ is a linear combination of of the basis vectors of the column space (k number of independent columns in $\mathbf{G}$, so k basis vectors for the column space). Where the scalar coefficient (for making linear combination) is each row scalar of the column vector $\mathbf{u}$. As $\mathbf{b}$ is also an n tuple, it is a vector in the vector space of n-tuples or a vector formed by linear combination of the basis vectors for the whole space $\{\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3, \ldots, \mathbf{h}_r, \mathbf{f}_1, \ldots, \mathbf{f}_{n-r}\}$ (note A).

$$\begin{aligned}\mathbf{u} &= \mathbf{G}\mathbf{b}^{\mathrm{T}}\\ &= \mathbf{G}(c_1\mathbf{h}_1^{\mathrm{T}} + c_2\mathbf{h}_2^{\mathrm{T}} + \ldots + c_r\mathbf{h}_r^{\mathrm{T}} + a_1\mathbf{f}_1^{\mathrm{T}} + a_2\mathbf{f}_2^{\mathrm{T}} + \ldots + a_{n-r}\mathbf{f}_{n-r}^{\mathrm{T}})\end{aligned}$$

As all $\mathbf{h}_i$ are in $W^{\perp}$, so if we multiply $\mathbf{h}_i$ with a matrix $\mathbf{G}$ having rows formed with vectors over W, the multiplication would produce a all zero vector $\mathbf{G}\mathbf{h}_i^{\mathrm{T}} = \mathbf{0}$. Then from (note A) the set $\{\mathbf{0}, \mathbf{0}, \ldots, \mathbf{0}, \mathbf{G}\mathbf{f}_1^{\mathrm{T}}, \mathbf{G}\mathbf{f}_2^{\mathrm{T}}, \ldots, \mathbf{G}\mathbf{f}_{n-r}^{\mathrm{T}}\}$ spans the column space of $\mathbf{G}$. And also from the above equation every vector in the column space is formed by linear combination of $\{\mathbf{G}\mathbf{f}_1^{\mathrm{T}}, \mathbf{G}\mathbf{f}_2^{\mathrm{T}}, \ldots, \mathbf{G}\mathbf{f}_{n-r}^{\mathrm{T}}\}$. As the vectors in the basis set for whole space $\{\mathbf{h}_1, \mathbf{h}_2, \ldots, \mathbf{h}_r, \mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{n-r}\}$ are linearly independent, so $\{\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_{n-r}\}$ are linearly independent. To prove that $\{\mathbf{G}\mathbf{f}_1^{\mathrm{T}}, \mathbf{G}\mathbf{f}_2^{\mathrm{T}}, \ldots, \mathbf{G}\mathbf{f}_{n-r}^{\mathrm{T}}\}$ is a basis for the column space, we have to show

1. $\{\mathbf{G}\mathbf{f}_1^{\mathrm{T}}, \mathbf{G}\mathbf{f}_2^{\mathrm{T}}, \ldots, \mathbf{G}\mathbf{f}_{n-r}^{\mathrm{T}}\}$ spans the column space.
2. they are linearly independent.

   1 is already shown.

To prove, 2

$$\text{If}\quad a_1(\mathbf{G}\mathbf{f}_1^{\mathrm{T}}) + a_2(\mathbf{G}\mathbf{f}_2^{\mathrm{T}}) + \ldots + a_{n\text{-}r}(\mathbf{G}\mathbf{f}_{n\ -r}^{\mathrm{T}}) = \mathbf{0} \qquad (2.19)$$

Then $\mathbf{G}(a_1\mathbf{f}_1^{\mathrm{T}}) + \mathbf{G}(a_2\mathbf{f}_2^{\mathrm{T}}) + \ldots + \mathbf{G}(a_{n\text{-}r}\mathbf{f}_{n\ -r}^{\mathrm{T}}) = \mathbf{0}$. That is $\mathbf{G}(a_1\mathbf{f}_1^{\mathrm{T}} + a_2\mathbf{f}_2^{\mathrm{T}} + \ldots + a_{n\text{-}r}\mathbf{f}_{n\ -r}^{\mathrm{T}}) = \mathbf{0}$. But, we have already stated $\mathbf{f}_i$'s are linearly independent. So, $a_1\mathbf{f}_1^{\mathrm{T}} + a_2\mathbf{f}_2^{\mathrm{T}} + \ldots + a_{n\text{-}r}\mathbf{f}_{n\ -r}^{\mathrm{T}} = \mathbf{0}$, can be zero only when all a_i's are 0.

So from all linear combinations of $\mathbf{f}_i$'s only one produces the zero vector and all other are nonzero, So from equation (2.19) the $\mathbf{G}\mathbf{f}_i^{\mathrm{T}}$'s are linearly independent. So, n-r is the dimension of the column space, but the dimension of the column space is k, therefore n-$r = k$ or n-$k = r$. But, r is the dimension of the null space. So, it is proved that if the dimension of a space is k then the dimension of the null space is n-k.

2.3.2 : Dual Codes :

There is a relationship between the generator matrix **G** and parity check matrix **H**, $\mathbf{G}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$, another important relationship between the generator matrix **G** and the parity check matrix **H** can be illustrated by transposing the two sides of the matrix equation $\mathbf{G}\mathbf{H}^{\mathrm{T}} = \mathbf{0}$, which yields $\mathbf{H}\mathbf{G}^{\mathrm{T}} = \mathbf{0}^{\mathrm{T}}$, we have transposed the zero matrix simply to emphasize the fact that since **0** has k rows and $(n$-$k)$ columns, $\mathbf{0}^{\mathrm{T}}$ has $(n$-$k)$ rows and k columns.

H is a $(n$-$k) \times n$ matrix.
G is a $k{\times}n$ matrix.

We can let **i** be an $(n$-$k)$ element binary vector or an $(n$-$k)$ tuple. The product $\mathbf{c}' = \mathbf{i}\mathbf{H}$ represents linear combination of rows of **H**. By letting **i** range through all $2^{n\text{-}k}$ binary combinations, we generate all the $2^{n\text{-}k}$ linear combination of rows of **H**. $\therefore$ $\mathbf{c}'$ is in the row space of **H**. Also, $\mathbf{c}'\mathbf{G}^{\mathrm{T}} = \mathbf{i}(\mathbf{H}\mathbf{G}^{\mathrm{T}}) = \mathbf{i}.\mathbf{0} = \mathbf{0}$, **0** is a $1\times k$ matrix.Transposing, we get $\mathbf{G}(\mathbf{c}')^{\mathrm{T}} = \mathbf{0}$, this **0** is a $k \times 1$ matrix. So, $\mathbf{c}'$ is in null space of **G**, or all the linear combinations of the rows of **H** (as $\mathbf{c}'$ is a linear combination of rows of **H**) are in the null space of **G**. This suggests that here **H** is serving as a generator matrix and **G** as a parity check matrix for a code with $2^{n\text{-}k}$ codewords. (Just opposite of the case discussed in section 2.2.6 - the role of **H** and **G** is changed)

If $\mathbf{i}_j\mathbf{H} = \mathbf{v}_j$ and $\mathbf{i}_k\mathbf{H} = \mathbf{v}_k$, $\mathbf{v}_j\mathbf{G}^{\mathrm{T}} = (\mathbf{i}_j\mathbf{H})\,\mathbf{G}^{\mathrm{T}} = \mathbf{0}$, $\mathbf{v}_k\mathbf{G}^{\mathrm{T}} = \mathbf{0}$, $\therefore$ $\mathbf{v}_j$ and $\mathbf{v}_k$ are both in null space of parity check matrix **G**, we have to show , $\mathbf{v}_j + \mathbf{v}_k$ is also in the null space of **G** , (that is if $\mathbf{v}_j$ and $\mathbf{v}_k$ is in the code, then $\mathbf{v}_j + \mathbf{v}_k$ is also in the code (as the new code should be a linear code) that is $\mathbf{v}_j + \mathbf{v}_k$ is also a codeword. $(\mathbf{v}_j + \mathbf{v}_k)\mathbf{G}^{\mathrm{T}} = \mathbf{v}_j\,\mathbf{G}^{\mathrm{T}} + \mathbf{v}_k\mathbf{G}^{\mathrm{T}} = \mathbf{0} + \mathbf{0} = \mathbf{0}$ (using distributive law), which says that $\mathbf{v}_j + \mathbf{v}_k$ is also in the null space of **G**, proving that the vectors $\mathbf{v}_l$ $(l = 0,1,\ldots, 2^{n\text{-}k}\text{-}1)$ generated by **iH** are all codewords and constitutes a vector space (satisfies closure property, distributive property, addition is commutative, and so on).

We have to also show that the dimensionality of the vector space is (n-k). A theorem of linear algebra states that if the dimension of a subspace of n-tuples (vector with n components, that is number of columns) is m (the number of rows in the generator matrix of the subspace = dimension of the subspace), the dimension of null space is (n-m) = column - dimension. This means that, if a row of a $k \times n$ matrix **G** generates a vector space (a row space) of dimension k, the null space of code vector space, which is also the null space of **G** (parity check matrix) is a vector space of dimension (n-k). But we have already proved that the **v**'s are in the null space of **G** (when **G** is a parity check matrix),so that the vector space composed of all such **v**'s such that $\mathbf{vG}^T=\mathbf{0}$, is a vector space of dimension (n-k), i.e. there are n-k linearly independent rows in the generator matrix (**H**), so, n-k elements in a information vector. So, over GF(2), $2^{n\text{-}k}$ information vectors or message vectors . This vector space is the linear code with $2^{n\text{-}k}$ codewords. In otherway , we can use the theorem to show that **H,** which has a null space of dimension k (k rows of **G**), must generate a vector subspace of dimension n-k (when **H** contains vectors of n tuples). *This subspace is therefore a code generated by matrix H , called the dual of the code generated by G.*

2.3.3 : Properties Of Syndrome :

When a particular code word **c** is transmitted, we receive a word **r** which may differ from **c** in one or more symbol (for binary code a symbol is a bit) positions due to the channel induced errors. That is

$$\mathbf{r}=\mathbf{c}+\mathbf{e} \tag{2.20}$$

where **e,** which is called the error vector or error pattern, has zeros in those positions where **r** and **c** agree and ones in those positions where **r** and **c** disagree. In other words, the ones in **e** mark the positions where errors have occurred. *The decoding problem then is to attempt to determine e so that the intended codeword may be recovered by forming* $\hat{\mathbf{c}}$= r-$\hat{\mathbf{e}}$ *, where* $\hat{\mathbf{e}}$ *is the estimate of the error pattern and* $\hat{\mathbf{c}}$ *is the decoded word . When* $\hat{\mathbf{e}}$ = *e , we have* $\hat{\mathbf{c}}$=*c , which means that decoding successfully reproduces the transmitted codeword .*

The algorithms commonly used for decoding parity check block codes start with the calculation of an (n-k) element vector called the *syndrome*. Syndrome is symptomatic to the error vector that actually occurred. The term symptomatic is used , since the syndrome *does not uniquely specify* the true error vector. What is important though, is that the syndrome can be used to identify the complete set of error vectors that could have occurred (the same syndrome can indicate two or more error vectors). This is clear from the discussion of most likely choice of coset leaders or error vectors and production of same syndrome for same coset in section 2.3.5.1 below the table for (5,2) code. The decoder then made best selection from the set. Let us consider the transmission of a codeword **c** satisfying the matrix equation $\mathbf{cH}^T = \mathbf{0}$, and the reception of the vector **r**. *The syndrome corresponding to r has been defined as* $s=rH^T$. Important general properties of the syndrome for the parity check codes are given below :

Property : 2.3.3.1 : The syndrome depends only on the error pattern, and not on the transmitted codeword.

proof:

$$\begin{aligned} \mathbf{s} &= \mathbf{r}\mathbf{H}^{T} \\ &= (\mathbf{c}+\mathbf{e})\mathbf{H}^{T} \\ &= \mathbf{c}\mathbf{H}^{T} + \mathbf{e}\mathbf{H}^{T} \\ &= \mathbf{e}\mathbf{H}^{T} \qquad (\text{ because } \mathbf{c}\mathbf{H}^{T}= \mathbf{0}\ ) \end{aligned} \tag{2.21}$$

Hence, the parity check matrix of a code permits us to compute the syndrome or syndrome vector s which depends only upon the error pattern **e**.

Property : 2.3.3.2 : All received words that differ by a codeword (that is both have errors in the same bit positions) have the same syndrome.

proof:

For k message bits, there are 2^k distinct code vectors denoted as $\mathbf{c}_i$, i = 0,1,..., 2^k-1. Correspondingly for any error pattern **e**, we define 2^k distinct vectors $\mathbf{e}_i$ as

$$\mathbf{e}_i = \mathbf{e}+\mathbf{c}_i,\ \ i = 0,1,..., 2^k - 1, \tag{2.22}$$

$\mathbf{e}_i$ has same role as received vector **r**. In other words, a coset has exactly 2^k elements that differ at most by a codeword or codevector. Thus, an (n,k) linear block code has 2^{n-k} (number of vectors with n-tuples/ number of vectors in each coset) possibilities of syndrome values.

$$\mathbf{e}_i\mathbf{H}^{T}= \mathbf{e}\mathbf{H}^{T}+\mathbf{c}_i\mathbf{H}^{T}= \mathbf{e}\mathbf{H}^{T} \tag{2.23}$$

which is independent of the index i. Same error means same coset. Accordingly we may state that a coset of a code is characterized by a unique syndrome, but the same syndrome can indicate more than one coset. This is shown in the table of section 2.3.7. As for example both the cosets {001,110} and {110, 001} for code words {000 , 111 } have syndrome 01.

Let , **H** (parity check matrix) have the systematic form
$\mathbf{H} = [\mathbf{I}_{n\text{-}k} \mid \mathbf{P}^{T}]$ (where **P** is the parity matrix from equation (2.8))

$$= \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & p_{00} & p_{10} & \cdots & p_{k-1,0} \\ 0 & 1 & 0 & \cdots & 0 & p_{01} & p_{11} & \cdots & p_{k\text{-}1,1} \\ 0 & 0 & 1 & \cdots & 0 & \cdots & \cdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & p_{0,n-k-1} & p_{1,n-k-1} & \cdots & p_{k-1,n-k-1} \end{bmatrix}$$

we find from equation

$\mathbf{s} = \mathbf{e}\mathbf{H}^T$

$$\Rightarrow \quad [s_0 \;\; s_1 \ldots s_{n\text{-}k\text{-}1}] = \mathbf{e}\mathbf{H}^T$$
$$= [e_0 \;\; e_1 \ldots e_{n\text{-}1}]\, \mathbf{H}^T$$

$$= [e_0 \;\; e_1 \ldots e_{n\text{-}1}] \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 \\ p_{0,0} & p_{0,1} & \cdots & p_{0,n-k-1} \\ p_{10} & p_{11} & \cdots & p_{1,n-k-1} \\ \vdots & \vdots & \cdots & \vdots \\ p_{k-1,0} & p_{k-1,1} & \cdots & p_{k-1,n-k-1} \end{bmatrix}$$

$$\therefore\; s_0 = e_0 + e_{n\text{-}k}p_{00} + e_{n\text{-}k\text{+}1}p_{10} + \ldots + e_{n\text{-}1}p_{k\text{-}1,0}$$
$$s_1 = e_1 + e_{n\text{-}k}p_{01} + e_{n\text{-}k\text{+}1}p_{11} + \ldots + e_{n\text{-}1}p_{k\text{-}1,1}$$
$$\vdots$$
$$s_{n\text{-}k\text{-}1} = e_{n\text{-}k\text{-}1} + e_{n\text{-}k}p_{0,n\text{-}k\text{-}1} + e_{n\text{-}k\text{+}1}p_{1,\; n\text{-}k\text{-}1} + \ldots + e_{n\text{-}1}p_{k\text{-}1,\; n\text{-}k\text{-}1}$$

(2.24)

This set of (n-k) linear equations clearly shows that the syndrome contains information about the error pattern and may therefore be used for error correction or at least error detection (for single parity check code). However , it should be noted that the set of equations be undetermined in that we have more unknowns (n) than equations (n-k) . Accordingly , there is no unique solution for error pattern . Rather for the same value of syndrome, and for GF(2) there are 2^k error patterns (n-(n-k) = k bit values are chosen randomly producing total n-k+k = n bit values for **e** for same **s**, depending on chosen k bits, other n-k bits are fixed, so for k arbitrary values, 2^k possible combinations) that satisfy equation (2.24). Different member of a coset can be chosen as coset leader or error pattern, but the same coset has always same syndrome irrespective of the choice of the coset leader.

But, according to equation (2.23) and property 2.3.3.2, for the same syndrome there are 2^k received vectors in a coset . The information contained in the syndrome **s** about the error pattern is not enough for the decoder to compute the exact value of the transmitted code vector or codeword. Nevertheless, the knowledge of the syndrome **s** reduces the search for the true error pattern **e** from 2^n to 2^{n-k} possibilities of syndrome values . In particular , the decoder has the task of making the best selection of transmitted codeword from the coset corresponding to s . For this most likely error pattern is preferred as coset leader (or occurred error) over several possible choices of error pattern in a coset producing same syndrome .

Property : 2.3.3.3 : The syndrome **s** is the sum of the rows of $\mathbf{H}^T$ corresponding to the bit positions in which the errors have occurred.

proof :

First we writ check matrix in the form

$$\mathbf{H}^T = \quad v \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \\ \vdots \\ \mathbf{h}_n \end{bmatrix} e \text{ the } j \text{ th row of } \mathbf{H}^T .$$

The syndrome then can be written as $\mathbf{s} = \mathbf{e}\mathbf{H}^T = \sum_{j=1}^{n} e_j \mathbf{h}_j$ (2. 25)

Where e_j is the j th element of the error vector **e**. However $e_j = 1$ if an error has occurred in the j th bit position, and $e_j = 0$ otherwise. Therefore, **s** is the summation of those rows of $\mathbf{H}^T$ whose row positions are error location in **e**.

2.3.4 : Hamming Distance And Weight Distribution :

Before considering the error correction power of linear block codes , it is first necessary to know some key concepts .

2.3.4.1 : Hamming distance :

The hamming distance between two vectors (or *n*-tuples) having the same number of elements is defined as the number of positions in which the elements differ . For example , the hamming distance between the vectors (1,0,1) and (0,1,1) is 2 .

2.3.4.2 : Minimum distance :

The minimum distance *d* of a linear block code *C* is the smallest of the hamming distances between pairs of different codewords in the code . It follows that the minimum distance is the smallest hamming weight of $\mathbf{c}_i - \mathbf{c}_j$ where $\mathbf{c}_i$ and $\mathbf{c}_j$ are any two different codewords in *C* . Since the difference (or sum for a binary code) of a pair of codewords is another codeword , the minimum distance of a linear code is the minimum hamming weight of the non zero code words .

2.3.4.3 : Hamming weight :

The hamming weight of a vector is the number of non zero elements in the vector . Stated differently , the hamming weight of a vector is equal to the hamming distance between the vector and the all zero vector . The hamming distance between two vectors $\mathbf{v}_i$ and $\mathbf{v}_j$ is the hamming weight $\mathbf{v}_i - \mathbf{v}_j$.

2.3.4.4 : Relationship between the minimum distance of the linear block code *C* and the structure of the parity check matrix H :

H can be written in the form $\mathbf{H} = [\,\mathbf{h}_1 \;\; \mathbf{h}_2 \;\; \mathbf{h}_3 \;\; \ldots \;\; \mathbf{h}_n\,]$

Where $\mathbf{h}_i$ is the *i* th column of **H** . We know we can define *C* as the set of all *n*-tuples or vectors for which $\mathbf{cH}^T = \mathbf{0}$, where *C* is the code and **c** is a code vector from the code . We can restate this by saying that a codeword in *C* is a vector having ones in the position such that the corresponding rows of $\mathbf{H}^T$, i.e. corresponding columns of **H** sum to the zero vector **0** .

The minimum codeword weight(*d*)= the minimum hamming distance between any pair of different codewords
=the minimum number of columns of **H** summing to **0**
= the minimum distance of the code .

The parity check matrix for the (7,4) hamming code has columns that are the seven nonzero binary 3-tuples (n-k=7-4=3 , because **H** is an (n-k)×n matrix) , that is 7 columns and 3 rows .All columns are distinct - that is as no column is repeated , no two columns can sum to zero , and *d* must be at least 3 . Any two columns must sum to a third column in **H** because all the possible nonzero seven 3 tuples are used as columns of **H** (given with the description of Hamming Code), and thus adding the third column into the sum we produce **0** . Therefore it can be concluded that the minimum distance of the (7,4) hamming code is $d=3$.

2.3.4.5 : Relationship between the minimum distance and error correction ability of a code :

The minimum distance of a linear block code , d_{min} is an important parameter of the code, specifically it determines the error correcting capability of the code . Let us assume an (n,k) linear block code is required to detect and correct all error patterns , and whose hamming weight is less than or equal to t. That is, if a code vector $\mathbf{c}_i$ in the code is transmitted and the received vector is $\mathbf{r} = \mathbf{c}_i + \mathbf{e}$, we require that the decoder output $\hat{\mathbf{c}} = \mathbf{c}_i$, whenever the error pattern **e** has a hamming weight $w(\mathbf{e}) \leq t$. We assume that the 2^k code vectors are transmitted with equal probability .The best strategy for the decoder then is to pick the code vector closest to the received vector **r** , that is the one for which the hamming distance $d(\mathbf{c}_i ,\mathbf{r})$ is the smallest . With such a strategy the decoder will be able to detect and correct all error patterns of hamming weight $w(\mathbf{e})\leq t$, provided that the minimum distance of the code is equal to or greater than $2t+1$. That is lowest value of minimum hamming distance for maximum t error correction is $2t+1$.

We may demonstrate the validity of this requirement by adopting a geometric interpretation of the problem . In particular, the $1\times n$ code vectors and $1\times n$ received vectors are represented as in an n-dimensional space . Suppose we construct two sphere, each of radius t , around the points that represent code vectors $\mathbf{c}_i$ and $\mathbf{c}_j$.

Let these two spheres be disjoint , as shown in figure (2.1a) . For this condition to be satisfied , we require that $d(\mathbf{c}_i ,\mathbf{c}_j) \geq 2t+1$. If then the code vector $\mathbf{c}_i$ is transmitted and the hamming distance $d(\mathbf{c}_i , \mathbf{r}) \leq t$, it is clear that the decoder will pick $\mathbf{c}_i$ as it is the code vector closest to the received vector **r** . If on the other hand the hamming distance $d(\mathbf{c}_i , \mathbf{c}_j) \leq 2t$ the two spheres around $\mathbf{c}_i$ and $\mathbf{c}_j$ intersect , as depicted by figure (2.1b) . Here we see that if $\mathbf{c}_i$ is transmitted , there exists a received vector **r** such that the hamming distance $d(\mathbf{c}_i , \mathbf{r}) \leq t$, and yet is as close to $\mathbf{c}_j$ as it is to $\mathbf{c}_i$. Clearly there is now the possibility of the decoder picking the vector $\mathbf{c}_j$, which is wrong . We thus conclude that an (n,k) linear block code has the power to correct error patterns of weight t or less , if and only if ,

$$d(\mathbf{c}_i , \mathbf{c}_j) \geq 2t+1 \text{ for all } \mathbf{c}_i , \mathbf{c}_j$$

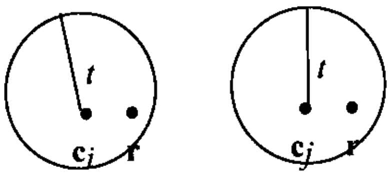


Figure 2.1(a) : Hamming distance $d(c_i, c_j) \geq 2t+1$

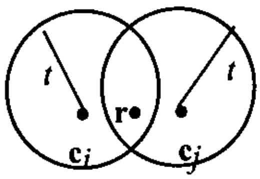


Figure 2.1(b) : Hamming distance $d(c_i, c_j) < 2t$

By definition, however the smallest distance between any pair of code vectors in a code is the minimum distance of the code d_{min} . We may therefore state that an (n,k) linear block code of minimum distance d_{min} can correct up to t errors if and only if

$$t \leq \left\lfloor \frac{1}{2}(d_{min} - 1) \right\rfloor \tag{2.26}$$

Where $\lfloor \; \rfloor$ denotes the largest integer less than or equal to the enclosed quantity . Equation (2.26) gives the error -correcting capability of a linear block code a quantitative meaning .

Theorem 2.3.4.1 : The code C contains a nonzero codeword of hamming weight w or less if and only if a linearly dependent set of w columns of **H** (that w rows of $\mathbf{H}^T$) exists :

proof :

For any codeword **c**, $\mathbf{cH}^T = \mathbf{0}$. Let **c** has weight w (nonzero elements) . Then $\mathbf{cH}^T = \mathbf{0}$ is a linear dependence relation or sum of w rows of $\mathbf{H}^T$ those sums to zero . Hence **H** has a linearly dependent set of w columns .

Conversely , if **H** has a linearly dependent set of w columns , then a linear combination of at most (with respect to code vector) w columns of **H** is equal to zero .(If there are more than w nonzero symbols in the codeword and **H** has some sets with more than w columns linearly independent then their sum will not produce the zero vector , but when at most w nonzero symbols in the codeword , then the w columns of **H** (from the set of more than w columns which are independent -clear from the **H** matrix of hamming code given later) may produce linearly dependent relation and other columns of the set being multiplied with zero , still suffice the definition of linear dependence - not all scalar multipliers are zero . If w columns of **H** are linearly dependent then **c** may have any number of nonzero symbols from 1 up to w. If w -1 columns of **H** are linearly dependent then **c** may have any number of nonzero symbols from 1 up to w-1 (number of nonzero symbols)) . These w

or less than w nonzero coefficients define a vector **c** of weight w or less , i.e. **c** has non zero symbols at those positions for which $\mathbf{cH}^T = \mathbf{0}$.

Theorem 2.3.4.2 : A code has minimum weight (at least) not smaller than w if and only if every set of (w-1) columns of **H** are linearly independent :

proof:

$\mathbf{cH}^T = \mathbf{0}$, and if (w-1) columns are linearly independent , then at least one column is needed to make it linearly dependent , that is $\mathbf{cH}^T = \mathbf{0}$. This means the codeword or codevector also have at least w-1+1 = w nonzero elements . That is the code has minimum weight at least w .

Hence , to find an (n,k) code that can correct t errors (the **H** matrix has every set of at least 2t+1 columns linearly dependent because the code has minimum distance 2t+1), it suffices to find an (n-k) by n matrix **H** with every set of 2t columns linearly independent . That is every set of 2t+1 column is linearly dependent .

Theorem 2.3.4.3 : The minimum distance d^* of any linear (n,k) (n is the codevector size , and k is the size of information vector) code satisfies $d^* \leq 1+n-k$ (1+number of parity symbols) (Singleton bound) :

proof:

The smallest weight nonzero codeword has weight d^* . In a systematic code some nonzero codewords may exist (among other codewords with more than 1 nonzero information symbol , also for clarity of thinking , example of systematic code is drawn , case of nonsystematic code is also same , because nonsystematic code from a systematic code is obtained by column interchange in the codeword which does not change the minimum weight) with only one nonzero information symbol and (n-k) parity symbols . Such a nonzero codeword cannot have weight larger than 1+n-k . That is minimum weight of nonzero codewords or number of nonzero symbols in such a code can be at most 1+n-k . If all n-k are non zero then $d^* = 1+n-k$ else $d^* < 1+n-k$.$\therefore$ $d^* \leq 1+n-k$.

example :

000,001,010,011, ... ,111 are each three bit information vector ,in codeword form they are 000 (n-k parity symbols), 001(n-k parity symbols), ... , 111 (n-k parity symbols) . The above discussion can be clear from this codeword structure .

2.3.5 : Syndrome Decoding And Standard Array :

Let , $\mathbf{c}_1$, $\mathbf{c}_2$, ..., $\mathbf{c}_{2^k}$ denote the 2^k code vectors of an (n,k) linear block code . Let **r** denotes the received vector , which may have one of 2^n possible values . The receiver has the task of partitioning the 2^n possible received vectors into 2^k disjoint subsets D_1 , D_2 , ... , D_{2^k} in such a way that the i th subset D_i corresponds to code vector $\mathbf{c}_i$ for $1 \leq i \leq 2^k$. The received vector is decoded into $\mathbf{c}_i$ if it is in the i th subset . For the decoding to be correct , **r** must be in the subset that belongs to the code vector $\mathbf{c}_i$ that was actually sent .The 2^k subsets described herein constitute standard array of linear block code . To constitute the standard array :

1. The 2^k code vectors are placed in a row with the all zero code vector $\mathbf{c}_1$ as the leftmost element .

2. An error pattern $\mathbf{e}_2$ is picked and placed under $\mathbf{c}_1$ and a second row is formed by adding $\mathbf{e}_2$ to each of the remaining code vectors in the first row ; it is important that error pattern chosen as the first element in a row does not have previously appeared in standard array .

3. Step 2 is repeated until all error patterns have been accounted for .

Now the code is a subgroup and the process generates cosets . Hence it halts with each word used exactly once .

$$\begin{array}{llllllll}
\mathbf{c}_1 = \mathbf{0} & \mathbf{c}_2 & \mathbf{c}_3 & \cdots & \mathbf{c}_i & \cdots & \mathbf{c}_{2^k} \\
\mathbf{e}_2 & \mathbf{c}_2 + \mathbf{e}_2 & \mathbf{c}_3 + \mathbf{e}_2 & \cdots & \mathbf{c}_i + \mathbf{e}_2 & \cdots & \mathbf{c}_{2^k} + \mathbf{e}_2 \\
\mathbf{e}_3 & \mathbf{c}_2 + \mathbf{e}_3 & \mathbf{c}_3 + \mathbf{e}_3 & \cdots & \mathbf{c}_i + \mathbf{e}_3 & \cdots & \mathbf{c}_{2^k} + \mathbf{e}_3 \\
\vdots & \vdots & \vdots & \cdots & \vdots & & \vdots \\
\mathbf{e}_j & \mathbf{c}_2 + \mathbf{e}_j & \mathbf{c}_3 + \mathbf{e}_j & \cdots & \mathbf{c}_i + \mathbf{e}_j & \cdots & \mathbf{c}_{2^k} + \mathbf{e}_j \\
\vdots & \vdots & \vdots & \cdots & \vdots & \cdots & \vdots \\
\mathbf{e}_{2^{n-k}} & \mathbf{c}_2 + \mathbf{e}_{2^{n-k}} & \mathbf{c}_3 + \mathbf{e}_{2^{n-k}} & \cdots & \mathbf{c}_i + \mathbf{e}_{2^{n-k}} & \cdots & \mathbf{c}_{2^k} + \mathbf{e}_{2^{n-k}}
\end{array}$$

The Standard Array For An (n,k) Block Code

Property 2.3.5.1 : There 2^{n-k} rows or cosets in the standard array.

Proof:

The standard array is formed in the way of coset decomposition . There are 2^n elements in the whole vector space or in the standard array . The code is a coset or row in the standard array , for k bit information there are 2^k valid codewords .Also , the standard array or coset is rectangular because it is constructed in that way . So , there are $2^n / 2^k = 2^{n-k}$ rows or cosets .For $GF(q)$ number of cosets would be q^{n-k} .

The figure of standard array illustrates the structure of the standard array so constructed. Each of the 2^{n-k} rows of this array represent the *coset* of the code , and their (rows) first elements $\mathbf{e}_2, \mathbf{e}_3, \dots, \mathbf{e}_2{}^{n-k}$ are called *coset leaders* .

For a given channel the probability of decoding error is minimized when the most likely error patterns i.e. , those with the largest probability of occurrence) are chosen as the coset leaders. In the case of binary symmetric channel , the smaller the hamming weight of the error pattern , the more likely it is to occur. Accordingly , the standard array should be constructed with each coset leader having the minimum weight in its coset. In other words the added error pattern will be the most probable one .

The table of standard array is of value only conceptually . For large n and k such a table would be impractical to list. The table can be simplified if we can store only the first column and compute the remaining columns as needed.This we do by introducing the concept of syndrome of the error pattern.

2.3.5.1 : Decoding procedure for a linear block code :

1. For a received vector **r**, the syndrome **s** is computed , ($\mathbf{s} = \mathbf{r}\mathbf{H}^T$ within the coset characterized by the syndrome **s**, the coset leader is identified (i.e. the error pattern with the largest probability of occurrence) ; let us call it **e** .

2. The code vector is computed.**r**=**c**+**e** , therefore $\mathbf{c} = \mathbf{r} - \mathbf{e}$. For $GF(2)$ '-' is equivalent to '+'. Therefore , $\mathbf{c} = \mathbf{r} + \mathbf{e}$. The code vector **c** is the decoded version of the received vector **r** .

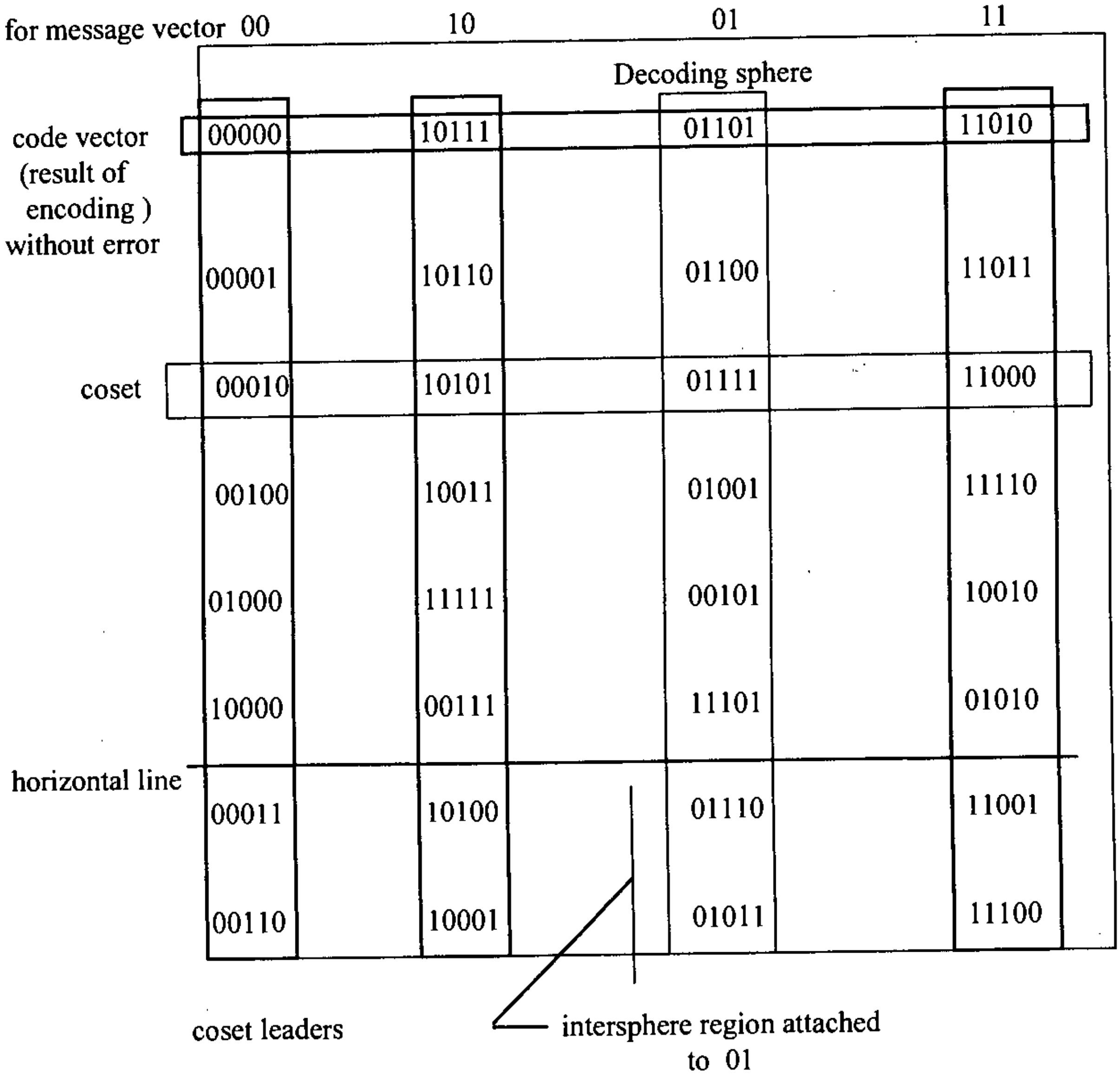
This procedure is called syndrome decoding .

Example :

Let us consider the (5,2) code with

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 \end{bmatrix}$$

As $2t+1 = n\text{-}k = 3$, this code can correct one error. The standard array is (by applying $\mathbf{c}_j = \mathbf{i}_j\mathbf{G}$, and then forming each row of the standard array by adding $\mathbf{e}_k$ with $\mathbf{c}_j$ for $j = 1,2, \ldots ,2^k$.


for message vector
Decoding sphere
code vector (result of encoding) without error
coset
horizontal line
coset leaders
intersphere region attached to 01

00	10	01	11
00000	10111	01101	11010
00001	10110	01100	11011
00010	10101	01111	11000
00100	10011	01001	11110
01000	11111	00101	10010
10000	00111	11101	01010
00011	10100	01110	11001
00110	10001	01011	11100

Syndrome for each coset is not listed here , but listed in the concise table given afterward .

The elements of the first column are coset leaders , the first column is for information vector 00, the second column is for the information vector 10 , the third is for 01 , and the fourth for 11 . For , this code by calculation it can be shown that 1 bit error is most probable . We may , (in row 2) instead of taking 00001 as the coset leader may choose 10110 (3 bit error , not most probable) as coset leader , in this case the same elements in the row would be produced but with different mappings between information vectors and standard array elements of this row . Each column is for each sphere.There are four spheres, six points per sphere, and eight points outside of any sphere .

Any two vectors in the same coset has same syndrome . Hence , we need only tabulate syndromes and coset leaders . We can then decode as follows. Given a received word (r) we computes the syndrome ($s=rH^T = cH^T + eH^T = eH^T$, It should be noted that e is the error vector , and in such way the concept that for a valid codeword c, $cH^T= 0$ is used in syndrome decoding , this concept is used in all decoding related to linear block code.)and look up its associated coset leader . This coset leader (most probable error pattern for the received word) is the difference between the received word and the center of the decoding sphere (codeword) - that is by subtracting coset leader from the received word we have the possibility of finding transmitted codeword - estimate of codeword (not information or message vector) .

For the example introduced above , the parity check matrix is

$\mathbf{H} = [\mathbf{P}^T \mid \mathbf{I}_r]$

(this r is not for received vector but indicates the number of parity symbols)

$$= \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Received words in the same coset (same row) of the standard array has same syndrome , this can be shown by considering row 2 of the standard array and its first two elements 00001 and 10110 . For received vector 00001

$$\mathbf{s} = \mathbf{r}\mathbf{H}^T = [00001] \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = [001]$$

For received vector 10110

$$\mathbf{s}=\mathbf{r}\mathbf{H}^T = [10110]\begin{bmatrix}1 & 1 & 1\\ 1 & 0 & 1\\ 1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} = [001]$$

The new table (from standard array) with coset leaders and syndromes is

coset leader	syndrome(S)
00000	000
00001	001
00010	010
00100	100
01000	101
10000	111
00011	011
00110	110

This is a simpler table than the standard array with reduced storage requirements.Suppose, **r** =10010 is received. Then **s** =**rH**^{T} = 101. The coset leader is 01000 . Therefore the transmitted word (code word or result of encoding) is 10010 - 01000 = 11010 , and the information word is 11. If syndrome is the zero vector then it is assumed that there is no error. But in some cases when the error pattern is such that it is uncorrectable by the code, a zero syndrome may be produced (though there is some error) . Such example is given later .

2.3.5.2 : Complete Decoder :

There are two basic classes of decoders that can be described in terms of the standard array : complete decoders and incomplete decoders . A complete decoder is one that assigns every received word to a nearby codeword . When a word is received , it is found in the standard array and decoded into the codeword at top of its column .

2.3.5.3 : Incomplete Decoder :

An incomplete decoder is one that assigns every received word to a codeword within distance t , if there is one , and otherwise refuses to decode . When a word is received , it is found in the standard array . If it lies above the horizontal line , it is decode into the word at the top of its column . If it is below the line , the received word is flagged as uncorrectable . It has more than t errors . The above given example is for incomplete decoder .

2.3.6 : Code Design And Sphere Packing :

The error-correction capability of an (n,k) linear block code is directly related to the defining property (**G** or **H**) of the code as a k dimensional subspace of the space of all n-tuples over a field. The way the subspace is constructed determines the minimum distance between the pairs of codewords , and this in terms determines how many errors can be corrected . In using a code , the " leftover" n-tuples , the n-tuples that are not in the code subspace (that is not actual codewords) , are identified (by the syndrome calculation) as codewords corrupted by errors . In bounded distance decoding , the problem is determining the radius t sphere to which the received n-tuples belongs .

Let us say we wish to have an (n,k) code capable of correcting t errors in each code block or code vector (code word) by the use of bounded distance decoding . Assume that we have chosen k and t and we want the block length n to be as small as possible in the interest of overall communication efficiency . Since a received word is decoded only if it lies within a radius t sphere centered on a codeword , we would like every received word to fall within one of the spheres . That is by minimizing the number of -tuples outside the spheres we are minimizing n (as information vector size k and radius of each sphere t are fixed) . This means that if we compute the volume of each sphere (number of possible received word within a sphere) and multiply by the number of codewords or number of spheres (2^k) , we would like the result to exactly equal to the total number of n-tuples (2^n) , as given by the following equation for a binary code

$$\left[1+\binom{n}{1}+\ldots+\binom{n}{t}\right]2^k = 2^n \tag{2.27}$$

Considering the case of all points are covered or all points are not covered this becomes

$$\left[1+\binom{n}{1}+\ldots+\binom{n}{t}\right]2^k \leq 2^n \tag{2.28}$$

Spheres cannot overlap , for there would then be an ambiguity as to the decoding of a received word lying simultaneously in more than one sphere .

Equation (2.28) is the generalization for an arbitrary t-error correcting binary code . A binary code for which equation (2.28) is satisfied is called a perfect code . It can be proved that all $(n,1)$ repetition codes of odd block length are perfect code (proof is given later). The binary Hamming codes are also perfect code , which can be shown as follows . The binary Hamming codes are defined to have block-length $n = 2^{n-k} -1$. So that all words within spheres of radius $t=1$ (single error correcting) are accounted for by $(1+n)2^k = 2^n$ which satisfies equation (2.27) or equation (2.28) with equality . For binary codes in general , the = sign in equation (2.27) must be replaced with ≤ sign , (as in equation (2.28)) , because the defining bound relating n,k,t that is satisfied with equality is true only for perfect codes . The bound in (2.28) can be generalized from the binary case to an arbitrary (n,k) t error correcting code over a q-ary alphabet , resulting in a sphere packing bound or Hamming bound , given by

$$\left[1+(q-1)\binom{n}{1}+(q-1)^2\binom{n}{2}+\ldots+(q-1)^t\binom{n}{t}\right]q^k \le q^n \tag{2.29}$$

The form of the bound is such because there is only one way of receiving a codeword correctly ($\binom{n}{0}=1$) here 0 means no symbol in the codeword of n symbols is in error , $\binom{n}{1}$ ways of having one wrong symbol in a codeword or block with $(q-1)$ possibilities for the wrong symbol value (over GF(q) there are q possible symbols in a position of a codeword but 1 of these symbol is for that position when no error have occurred in that position) , $\binom{n}{2}$ ways of having 2 wrong symbols in the block with $(q-1)(q-1) = (q-1)^2$ possibilities for two wrong symbol values , and so forth . The total volume of all the spheres cannot be greater than the total possibilities of received words q^n . A non binary code that satisfies the bound (2.29) with equality is a (nonbinary) *perfect code* .

It is customary to refer to codes as tightly packed or loosely packed in accordance with how well or how poorly they approach the sphere packing bound or Hamming-bound .

For a given k and t , block length n should be as small as possible in order to minimize the required redundancy . Also , given n and k , we want the error correction limit t (which is related to the minimum distance d^* or structure of the parity check matrix **H**) and hence the minimum distance d^* made as large as possible . The error detection and error correction power of a code is directly related to the minimum distance which should be made as large as possible .

2.3.7 : The Error Detection And Error Correction Properties Of A Code Are Related To The Structure Of Its Parity Check Matrix H And Thus The Minimum Weight Or Minimum Distance Of A Code :

The description of the parity check code can be given in terms of its parity-check matrix **H** and the syndrome can be interpreted as the sum of the columns of the parity check matrix (or rows of $\mathbf{H}^T$) corresponding to the locations of errors in the received word . Parity check matrices for two simple codes (single parity check code with $n = 7$ and a repetition code with $n = 3$) are listed .

$$[1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\,] \qquad \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

With the single-parity-check code , any odd number of errors produce a syndrome equal to 1 (odd number of errors , so odd number of columns all of which are 1 added modulo 2 and thus produces 1) , and since all columns of **H** are 1 (that is not distinct) the non-zero syndrome can yield no information on locations of errors (not correctable). However the parity check matrix for the repetition code has three distinct column vectors , and any single error will produce a syndrome that uniquely identifies one of the columns of **H** (or rows of $\mathbf{H}^T$) and in turn the position of single error in the received word .

Table For (3,1) Repetition Code

information	0	1	syndrome
code vector without error	000	111	00 (no error)
code vectors with error	001	110	01
	010	101	10
	100	011	11
	011	100	11
	101 (error in LSB and MSB)	010	10
	110	001	01
	111 (all three bits are in error)	000	00

Considering 1 bit error if we found 001 the correct code will be 000 and hence information vector would be 0 , from syndrome value of 01 and from **H** matrix we see the 3rd column (rightmost) column of **H** be 01 , so it is assumed that the rightmost bit of the received code word is in error . But we should also note that a two bit error can also cause a syndrome of 01 (sum of first and second columns of **H**) , but as 1 bit error is most probable (shown later) the erroneous code word is treated as the former case .

Every single error pattern will be *correctable* if every two columns of **H** are not the same, otherwise there would be ambiguity about the actual column of **H** from the calculated syndrome , in which position in the codeword the error has been occurred even in the case of single bit error .

2.3.8 : Singleton Bound and Maximum Distance Codes :

Property 2.3.8.1 : (Singleton Bound) the minimum distance (minimum weight) of any linear (n,k) code satisfies

$$d^* \leq 1+n-k$$

Proof :

The smallest-weight nonzero codeword has weight d*. Systematic codewords exist with only one nonzero information symbol and (n-k) parity symbols. Such a codeword cannot have weight larger than 1+(n-k). Hence the minimum weight of the code cannot be larger than 1+(n-k).

Any code whose minimum distance satisfies $d^* = n\text{-}k\text{+}1$ is called a maximum-distance code . Here maximum means maximum possible value of minimum distance $d^* = n\text{-}k\text{+}1$.

2.3.8.1 : Hamming code :

Parity check matrix **H** is a $(n\text{-}k)\times n$ matrix , $n\text{-}k = m$ is equal to the number of parity symbols . A code whose minimum weight (also minimum distance for linear code) is at least 3 must have a parity check matrix all of whose columns are distinct (as no pair of columns sums to zero) and nonzero by theorem 2.3.4.2 .

If a parity -check matrix for a binary code has m rows , then each column is an m bit binary number . There are 2^m - 1 possible columns (for excluding all zero column) . Hence if the **H** matrix of a binary code d^* (minimum distance) at least 3 has m rows , then it can have 2^m - 1 columns , but no more , to make each pair of columns linearly independent . This defines a $(2^m - 1, 2^m - 1\text{-}m)$= (n,k) code . [m is equal to number of parity symbols in a codeword , m=$n\text{-}k$, n=2^m - 1 , therefore k=n-m=2^m - 1-m]. The simplest nontrivial example is m=3 .Then in systematic form

$$\mathbf{H}=\begin{bmatrix}1&1&0&1&1&0&0\\1&0&1&1&0&1&0\\0&1&1&1&0&0&1\end{bmatrix} \qquad \mathbf{G}=\begin{bmatrix}1&0&0&0&1&1&0\\0&1&0&0&1&0&1\\0&0&1&0&0&1&1\\0&0&0&1&1&1&1\end{bmatrix}$$

These $(2^m - 1, 2^m - 1\text{-}m)$ codes are the hamming codes .Clearly , every pair of columns of **H** is independent (because no pair of distinct binary vectors can sum to zero) , and some set of three columns are dependent . ∴ By theorem 2.3.4.3 the minimum weight (w) is 3 , and the code can correct a single error , by applying equation 2.26 .

Hamming codes are codes for which the rows of **H** are distinct and includes all the nonzero sequences of length $n\text{-}k$ in the case of binary hamming code as columns of **H** . Hamming codes can be easily defined for larger alphabet sizes . The main idea is to define an **H** with every pair of columns linearly independent . Over GF(q) , $q \neq 2$, one cannot use all nonzero m-tuples as the columns of **H** , because some pairs will be linearly dependent The error-detection and error-correction properties of a code are related to the structure of its parity check matrix H , that is on minimum distance of the code . Every single error pattern in a received word will be detectable , as long as no columns of **H** is all zeros (if there is all zero column , there may be error in received word within correctable range but may produce all zero syndrome - which is according to assumption indication of no error), and every single error pattern in a received word will be correctable , if any two columns of **H** are not the same (because due distinctness of columns of **H** error positions can be located when error is in the correctable range). In other words , a single -error correcting code of block length n is one whose parity check matrix has n - unique nonzero columns . Therefore , we can construct a single-error correcting code by specifying an $(n\text{-}k)\times n$ parity check matrix with unique non zero columns . For , GF(2) , this can always be done as long as $n \leq 2^{n\text{-}k}$ -1 . The limitation reflects that there can be no more than $2^{n\text{-}k}$ -1 unique non-zero binary vectors of length $(n\text{-}k)$. Therefore , the longest single-error correcting (longest codeword) binary block code that can be constructed with r parity checks has block length $n = 2^r$ -1 . A code with this parameter is called binary hamming code .

For the (7,4) binary hamming code let us consider the parity -check matrix . Below two parity- check matrices for the (7,4) hamming code is given . The parity check matrix for the (7,4) hamming code has columns that are the seven nonzero binary 3-tuples ($n\text{-}k$=7-4=3 , because **H** is an $(n\text{-}k)\times n$ matrix) , that is 7 columns and 3 rows .All columns are distinct - that is as no column is repeated , no two columns can sum to zero , and d must be at least 3 . Any two column must sum to a third column in **H** , and thus adding the third column into the sum we produce **0** . therefore we conclude that the minimum distance of the (7,4) hamming code is d =3 .

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

On the left the columns of **H** are arranged as the binary coded numbers 1 to 7 . Any re-arrangement of the columns of **H** will preserve the single-error-correction capability (uniqueness of columns -so every pair of column is linearly independent , so , minimum weight of the code cannot be less than 3), and thus the ordering of the columns may be chosen in accordance with convenience of implementation for encoding or decoding . The second form of parity - check matrix is called reduced echelon form or systematic form . It is characterized by the partitioned structure **H**= [$\mathbf{P}^T$ | $\mathbf{I}_r$] , where $\mathbf{I}_r$ is the $r\times r$ identity matrix , and $\mathbf{P}^T$ is an $r\times k$ submatrix that defines the essential structure of the code . Where $r = n\text{-}k$ is the number of parity symbols in any code of the codeword .We already know that a block code constructed with a parity check matrix of this form is called a systematic code .

We also know that two codes are said to be equivalent if they differ only in the ordering of their symbols . Thus the codes formed by the two parity check matrices for (7,4) Hamming code are equivalent Hamming codes . We can generalize this by saying that all binary Hamming codes of a given length are equivalent .

We know for t error correcting binary code of block length n , the Hamming bound (sphere packing bound) is

$$\left[1+\binom{n}{1}+\binom{n}{2}+\ldots+\binom{n}{t}\right]2^k \le 2^n$$

And this bound can be generalized from the binary case to an arbitrary (n,k) t-error-correcting code over a q -ary alphabet , resulting in the sphere-packing bound or Hamming bound , given by

$$\left[1+(q-1)\binom{n}{1}+(q-1)^2\binom{n}{2}+\ldots+(q-1)^t\binom{n}{t}\right]q^k \le q^n$$

A nonbinary or binary code that satisfies this bound with equality is a *perfect code* . For a single error correcting nonbinary Hamming code this bound has the form

$$\left[1+(q-1)\binom{n}{1}\right]q^k = q^n \ .$$

The = sign is used instead of ≤ , because Hamming code is a perfect code and a perfect code satisfies the sphere packing bound or Hamming bound with equality .

$$\Rightarrow \left[1+(q-1)\binom{n}{1}\right] = q^{n-k}$$

$$\Rightarrow (q-1)\binom{n}{1} = q^{n-k} - 1$$

where n-k = r = m = number of parity symbols .

$$\Rightarrow n(q\text{-}1) = q^m - 1$$

$$\Rightarrow n = \frac{q^m - 1}{q-1}$$

$\therefore$ For a single error correcting non-binary Hamming code , there are $n = \dfrac{q^m - 1}{q-1}$ such distinct columns (n) . Hence the code is a (n,k) or $\left(\dfrac{q^m - 1}{q-1}, \dfrac{q^m - 1}{q-1} - m\right)$. A single error-correcting Hamming code with these parameters exists for each q (for which a field GF(q) exists) and for each m.

2.3.9 : The Decoder Performs Either The First Or All Three Of The Following Functions :

1. The encoding rules are reapplied to the received word to determine whether the parity check relationships are satisfied (If no errors have occurred in the transmitted codeword , after transmission , the received word **r** would produce $\mathbf{r}\mathbf{H}^T = \mathbf{0}$, though in some case codeword with error can produce the zero vector. Discrepancy in the parity check relationships ($\mathbf{r}\mathbf{H}^T \neq \mathbf{0}$) indicates the presence of one or more errors in the word . If only error detection is to be performed , the decoding function is completed with an announcement either that the received word is a codeword or that errors have been detected . If error corrections are to be performed ,the next two steps are required.

2. The parity check discrepancy is used to derive an estimate of the error pattern contained in the received word . (May be the syndrome) .

3. When an estimate of the error pattern has been determined , the errors are corrected (may be the example of syndrome decoding (for binary code) can be given . From computed syndrome , in the codeword where errors have been occurred can be assumed and the bits in the codewords at that positions can be inverted). Then the check digits are removed from the codeword to get the information vector or the message vector , and this decoded information is delivered to the user .

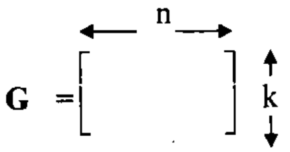
A central problem in coding theory is to find code designs for which the $r = n\text{-}k$ check digits can be used as effectively as possible in the detection and correction of errors . A given code may be utilized for error detection only or for error detection and correction . Principles governing the design of a code for error detection is essentially the same as that for effective error correction .

No code can be designed that can successfully correct all error patterns to which the transmitted codeword can be subjected . For example , an error pattern that changes one codeword into a different codeword is always undetectable (Also clear from the syndrome decoding of (3,1) repetition code when all three bits are in error ,errors cannot be detected) . Rather , codes are designed to correct the most likely error patterns .The effectiveness of a code on a given channel will depend in part upon the characteristics of the error patterns produced by the channel .

2.3.10 : Simple Modifications To A Linear Code :

Simple changes can be applied to a linear code to get a new code. If the new code is also linear, these changes corresponds to simple changes that can be made to the generator matrix **G**. Namely, one can add a column or a row, delete a column or a row, add both a column and a row, delete both a column and a row. These change the code in ways that are simply described, though it might not be a simple matter to find such modifications that are worthwhile.

The block length n can be increased by increasing k or by increasing number of parity symbols n-k. We call these lengthening and expanding and subsume both notions under the more general term extending that is by extending we mean increasing the block length either by expanding or by lengthening. The six basic changes are as follows :



Increasing the length by adding more parity check symbols is in effect *expanding a code* . This corresponds to increasing the larger dimension of the generator matrix.

Increasing the length by adding more information symbols (k) is called *lengthening a code* .. This corresponds to increasing both dimensions of the generator matrix by the same amount.

Reducing the length by dropping parity-check symbols is called *puncturing a code* . This corresponds to decreasing the larger dimension of the generator matrix.

Reducing the length by dropping information symbols is called *shortening a code* . This corresponds to decreasing both dimensions of the generator matrix by the same amount.

Increasing the number of information symbols without changing the length or block length n, that is increasing k, decreasing n-k) is called *augmenting a code* . This corresponds to increasing the smaller dimension of the generator matrix.

Decreasing number of information symbols without changing the length is called *expurgating a code*. This corresponds to decreasing the smaller dimension of the generator matrix.

2.3.11 : Reed-Muller Codes :

Reed-Muller codes are a class of linear code over GF(2) that are easy to describe and can be decoded by simple voting technique . For these reasons , the Reed Muller codes are important even though with some exceptions , their minimum distances are not noteworthy . For each integer m and for each integer r less than m there is a Reed-Muller code of block length 2^m called the r-th order Reed Muller code of block length 2^m.

A Reed Muller code is a linear code . This code will be defined by a procedure for constructing a generator matrix . we will construct a non systematic generator matrix that will prove convenient for decoding (easy grouping of received bits when forming check-sum equations to detect a information bit value by majority vote). First let us define the product of two vectors **a** and **b** by a componentwise multiplication (as already defined in case of inner product). That is , let

$$\mathbf{a} = (a_0, a_1, \dots, a_{n-1})$$
$$\mathbf{b} = (b_0, b_1, \dots, b_{n-1})$$

Then the product is the vector $\mathbf{ab} = (a_0b_0, a_1b_1, \dots, a_{n-1}b_{n-1})$

The generator matrix for the r-th order Reed-Muller code of block length 2^m is defined as an array of blocks

$$\mathbf{G} = \left.\overset{n}{\overline{\begin{bmatrix} \mathbf{G}_0 \\ \mathbf{G}_1 \\ \vdots \\ \mathbf{G}_r \end{bmatrix}}}\,\right|\, k$$

Where $\mathbf{G}_0$ is the vector of length $n = 2^m$ containing all ones , $\mathbf{G}_1$ an m by 2^m matrix , has each binary m tuple appearing once as a column ; and $\mathbf{G}_Z$ is constructed from $\mathbf{G}_1$ by taking its rows to be all products of rows of $\mathbf{G}_1$, z rows of $\mathbf{G}_1$ to a product . For definiteness (otherwise permutation of columns should be considered) , we take the leftmost column of $\mathbf{G}_1$ to be all zeros , the rightmost to be all one , and the others to be the binary m-tuples in increasing order , with the low order bit in the bottom row .

Because there are $\binom{m}{z}$ ways to choose the z rows to a product , $\mathbf{G}_Z$ is an $\binom{m}{z}$ by 2^m matrix . Clearly for any Reed-Muller code ,total number of rows in the generator matrix **G**

$$k=\binom{m}{0}+\binom{m}{1}+\binom{m}{2}+\ldots+\binom{m}{r}=1+\binom{m}{1}+\binom{m}{2}+\ldots+\binom{m}{r}$$

from definition of $\mathbf{G}_1$ block length $n = 2^m$ = all possible bit values combination in m positions (m rows) to produce 2^m columns = column with all zero symbol +columns with single 1 + columns with 2 ones +...+columns with m 1 =

$$\binom{m}{0}+\binom{m}{1}+\binom{m}{2}+\ldots+\binom{m}{r}+\binom{m}{r+1}+\binom{m}{r+2}+\ldots+\binom{m}{m}$$

$$=1+\binom{m}{1}+\binom{m}{2}+\ldots+\binom{m}{r}+\binom{m}{r+1}+\binom{m}{r+2}+\ldots+\binom{m}{m}$$

$$n\text{-}k=\left\{1+\binom{m}{1}+\binom{m}{2}+\ldots+\binom{m}{r}\right\}+\binom{m}{r+1}+\binom{m}{r+2}+\ldots+\binom{m}{m}-\left\{1+\binom{m}{1}+\binom{m}{2}+\ldots+\binom{m}{r}\right\}$$

$$=\binom{m}{r+1}+\binom{m}{r+2}+\ldots+\binom{m}{m-1}+\binom{m}{m}$$

$$=\binom{m}{m-r-1}+\binom{m}{m-r-2}+\ldots+\binom{m}{1}+\binom{m}{0}=1+\binom{m}{1}+\ldots+\binom{m}{m-r-2}+\binom{m}{m-r-1}$$

because $\binom{m}{r}=\binom{m}{m-r}$

Provided the rows of **G** are linearly independent (each combinations of bit appears only once , otherwise value of k will be of another magnitude, not as defined above).The 0-th order Reed Muller code is an $(n,1)$ code . It is a simple repetition code and is decoded trivially by a majority vote (For a (5,1) code by majority vote 00011 will be considered an erroneous form of the codeword 00000 , and the information is 0 .

Example :

For a (4,1) repetition code (As for Reed Muller code n is even , so this example of code with even block length is chosen)

The parity check matrix $\mathbf{H}=\begin{bmatrix}1&1&0&0\\1&0&1&0\\1&0&0&1\end{bmatrix}$

This form of **H** is formed from the parity check equations used for repetition code given below

i_1+p_1 = 0 mod 2

$$\begin{aligned} i_1 \quad +p_2 \qquad\qquad &= 0 \bmod 2 \\ i_1 \qquad +p_3 \qquad\qquad\qquad &= 0 \bmod 2 \end{aligned}$$

So, $\mathbf{G} = [1 : 111]$

For first column of **H** we get the last three columns of **G** , here **G** has form like $\mathbf{G}_0$ (all elements are 1) . As $\mathbf{G}_0$ is vector of length $n = 2^m$ containing all 1 , the 0-th order Reed Muller code has minimum distance 2^m . (For (4,1) repetition code the two possible correct codewords are 0000 , 1111 , that is minimum weight is the minimum distance between this two codewords).

An example of describing Reed Muller code in terms of generator matrix , let $m = 4$, $n = 16$, and $r = 3$. Then

$$\mathbf{G}_0 = [1111 \;\; 1111 \;\; 1111 \;\; 1111] = [\mathbf{a}_0]$$

$$\mathbf{G}_1 = \begin{bmatrix} 0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1 \\ 0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1 \\ 0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1 \\ 0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \\ \mathbf{a}_4 \end{bmatrix} \qquad \text{(say)}$$

Because $\mathbf{G}_1$ has 4 rows , $\mathbf{G}_2$ has $\binom{4}{2}$ rows .

$$\mathbf{G}_2 = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1 \\ 0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&1 \\ 0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&1 \\ 0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&1&0&1&0&0&0&0&0&1&0&1 \\ 0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1 \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1\mathbf{a}_2 \\ \mathbf{a}_1\mathbf{a}_3 \\ \mathbf{a}_1\mathbf{a}_4 \\ \mathbf{a}_2\mathbf{a}_3 \\ \mathbf{a}_2\mathbf{a}_4 \\ \mathbf{a}_3\mathbf{a}_4 \end{bmatrix}$$

And $\mathbf{G}_3$ has $\binom{4}{3}$ rows,

$$G_3 = \begin{bmatrix} 0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1 \end{bmatrix} = \begin{bmatrix} \mathbf{a}_1\mathbf{a}_2\mathbf{a}_3 \\ \mathbf{a}_1\mathbf{a}_2\mathbf{a}_4 \\ \mathbf{a}_1\mathbf{a}_3\mathbf{a}_4 \\ \mathbf{a}_2\mathbf{a}_3\mathbf{a}_4 \end{bmatrix}$$

Then the generator matrix for the third-order Reed Muller code of block length 16 is the $k{\times}n$ or $15{\times}16$ matrix. ($k = 1 + \binom{4}{1} + \binom{4}{2} + \binom{4}{3} = 1 + 4 + 6 + 4 = 15$).

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_0 \\ \mathbf{G}_1 \\ \mathbf{G}_2 \\ \mathbf{G}_3 \end{bmatrix}$$

This generator matrix gives a (n,k) or (16,15) code over GF(2) . In fact it is a single parity check code -can be shown by converting **G** to row echelon form and converting the row echelon form of **G** to systematic form . Then forming **H** from **G** , **H** will be as

$$\mathbf{H} = [\underbrace{1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1}_{\text{15 information bits}} \quad \underbrace{1}_{\substack{\text{1 parity bit} \\ (n\text{-}k = 16\text{-}15 = 1)}}]$$

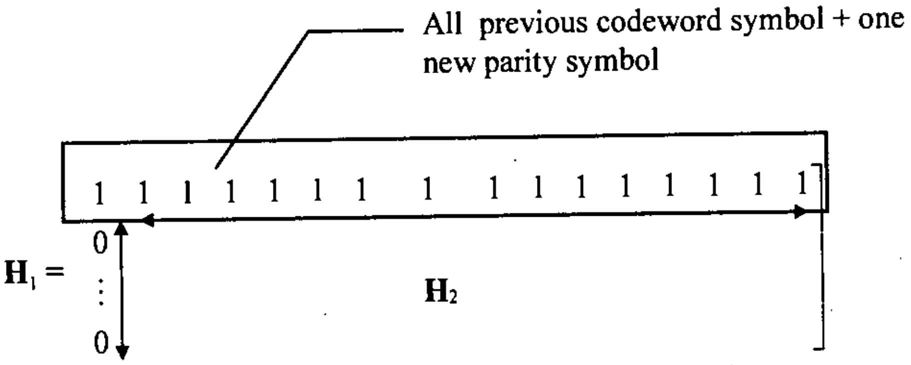
In equation form (assuming even parity)

$$i_1 + i_2 + i_3 + i_4 + i_5 + i_6 + i_7 + i_8 + i_9 + i_{10} + i_{11} + i_{12} + i_{13} + i_{14} + i_{15} + p = 0 \mod 2$$

Another Reed Muller code obtained from these same matrices is obtained by choosing r equal to 2 , then the generator matrix is

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_0 \\ \mathbf{G}_1 \\ \mathbf{G}_2 \end{bmatrix}$$

It gives a (16,11) Reed Muller code over GF(2) , in fact , it is the (15,11) Hamming code expanded by an extra parity check . (This can be shown by forming **H** matrix say $\mathbf{H}_1$ of (16,11) code , and extracting **H** matrix say $\mathbf{H}_2$ for (15,11) code from $\mathbf{H}_1$.



In the H_2 any two columns are distinct , and the code parameters (n,k) also satisfy the values (2^m-1 , 2^m -1- m) which is required for a Hamming code .

From the definition of their generator matrices , it is clear that an r-th order Reed Muller code can be obtained by augmenting an (r-1) th order Reed Muller code , and an (r-1) th order Reed Muller code can be obtained by expurgating an r-th order Reed Muller code . Clearly , because the r-th order Reed Muller code contains the (r-1) th order Reed Muller code , its minimum distance (depends on column structure . Let Number of dependent columns in a set , has minimum value x , then minimum weight is x+1) of **H** , which implies minimum distance depends on the structure of **G**)can not be larger (can be smaller) than that of the (r-1) th order Reed Muller code - this may be clear by observing the number of nonzero terms in each rows of the generator matrices . It will be proved later that $d^* = 2^{m-r}$ for an r-th order Reed Muller code .

Every row of $\mathbf{G}_z$ has weight 2^{m-z} . Hence , every row of **G** has even weight (as r is less than m and according to our initial assumption $m \neq z$, $2^{m-z} \neq 2^0$ or 1) . And the sum of two binary vectors of even weight must have an even weight . Hence all linear combination of rows of **G** (that is codewords) has even weight . The matrix $\mathbf{G}_r$ has rows of weight 2^{m-r} , and thus the minimum weight is not larger than 2^{m-r} .

We must show that the rows of **G** are linearly independent (otherwise more than one information vector (all zero vector) would produce all zero codeword) . We shall show that the code must have a minimum weight of 2^{m-r} and the rows of **G** must be linearly independent by developing a decoding algorithm - the Reed algorithm that corrects $t = \left.\frac{1}{2}\cdot 2^{m-r}-1\right)$ errors and recovers k information bits .This implies that the minimum distance is at least 2^{m-r} -1 (we know that , $d \geq 2t$+1, putting value of t , we get $d \geq 2\cdot \left.\frac{1}{2}\cdot 2^{m-r}-1\right)+1$), and because d is even , it is at least 2^{m-r} .

The Reed algorithm is designed specifically for Reed Muller codes . Of course the general syndrome techniques could be used but would not be as simple to implement . The Reed algorithm is unusual as compared to most algorithms for most codes in that it recovers the information directly from the received word and does not explicitly compute the error .Intermediate variables such as syndrome are not used .

Suppose that we have a decoder for an (r-1) th order Reed Muller code (that is the decoder is able to correct $\left.\frac{1}{2}\cdot 2^{m-(r-1)}-1\right)$ errors) . We will construct a decoder for an r-th order Reed Muller code in the presence of $\left.\frac{1}{2}\cdot 2^{m-r}-1\right)$ errors by reducing it into the earlier case . Because we already know that the 0-th order Reed Muller code can be decoded by majority vote (as is used for simple repetition code), we have a decoding algorithm by induction .

It is convenient to break the information vector into r + 1 segments written **i** = [$\mathbf{I}_0$, $\mathbf{I}_1$, $\mathbf{I}_2$, ... , $\mathbf{I}_r$] where segment $\mathbf{I}_Z$ contains $\left.\begin{matrix}m\\z\end{matrix}\right)$ information bits . Each segment multiplies one block of **G** . The encoding can be represented as a block vector matrix product .

$$\mathbf{c} = [\mathbf{I}_0 , \mathbf{I}_1 , \mathbf{I}_2 , \dots ,\mathbf{I}_r] = \left.\begin{matrix}\mathbf{G}_0\\ \mathbf{G}_1\\ \vdots\\ \mathbf{G}_r\end{matrix}\right] = \mathbf{iG}$$

Consider the information sequence broken into such sections : each section corresponds to one of the r+1 blocks of the generator matrix and is multiplied by the corresponding section of **G** during encoding . If we can recover the information bits in the r+1 th section (labeled as r), then we compute their contribution to the received word and subtract this contribution (because the **G** and **H** matrices are always known and fixed , so after recovering the information bits the contribution can be easily computed).This reduces the problem to that of decoding a smaller code . The decoding procedure is a succession of majority votes , starting with majority votes to determine th e information bits in the r+1 th section , then in the r th section and so on .

The received word is

$$\mathbf{v} = \underbrace{[\mathbf{I}_0 , \mathbf{I}_1 , \mathbf{I}_2 , \dots , \mathbf{I}_r]}_{k \text{ bits}} \overset{\xleftrightarrow{2^m \text{ bits}}}{\left.\begin{bmatrix}\mathbf{G}_0\\ \mathbf{G}_1\\ \vdots\\ \mathbf{G}_r\end{bmatrix}\right\updownarrow} k \text{ bits} \quad + \ \mathbf{e}$$

Where **e** is the introduced $1{\times}2^m$ error vector . The decoding algorithm will first recover $\mathbf{I}_r$ from $\mathbf{v}$. Then it computes $\mathbf{v}' = \mathbf{v} - \mathbf{I}_r\mathbf{G}_r$ (the first and second term on the right of '=' sign have same number of columns (2^m), otherwise this subtraction is not possible)

$$\therefore \mathbf{v}' = \underbrace{[\mathbf{I}_0 , \mathbf{I}_1 , \mathbf{I}_2 , \dots , \mathbf{I}_{r-1}]}_{k - \left.\begin{matrix}m\\ r\end{matrix}\right) \text{bits}} \overset{\xleftrightarrow{2^m \text{ bits}}}{\left.\begin{bmatrix}\mathbf{G}_0\\ \mathbf{G}_1\\ \vdots\\ \mathbf{G}_{r-1}\end{bmatrix}\right\updownarrow} k - \left.\begin{matrix}m\\ r\end{matrix}\right) \text{bits} \quad + \ \mathbf{e} \, .$$

First , let us consider decoding the information bit i_{k-1} , which multiplies the last row of $\mathbf{G}_r$. This is decoded by setting up 2^{m-r} linear check sums in the 2^m received bits . Each such check sum involves 2^r bits of the received word ($2^m/2^{m-r} = 2^r$), and each received bit (v_s , $s = 0,1,2, \dots 2^m-1$) is used in only one check sum . Information bits are i_y , $y = 0,1,2, \dots , k-1$, where k is the number of rows in the generator matrix). The check sums will be formed so that i_{k-1} (information bit) contributes to only one bit of each check sum , and every other information bit contributes to an even number of bits (so no contribution in the value of checksum , under mod 2 operation) in each check sum

. Hence , each check sum is equal to i_{k-1} in the absence of errors . But if there are at most $\left.\frac{1}{2}\cdot 2^{m-r}-1\right)$ errors , the majority of the check sums will still equal to i_{k-1}

Explanation :

Suppose the Reed Muller code of first order that is $r=1$. Let $m=4$. The generator matrix has $1+\left.\begin{matrix}4\\1\end{matrix}\right)=5$ rows , and also there are 5 bits in each information vector or message vector ($\mathbf{i}=[i_0, i_1, i_2, i_3, i_4]$) . There are $2^m=2^4=16$ bits in each codeword ,so 16 bits in each received vector $\mathbf{v}=[v_0, v_1, v_2, \ldots, v_{15}]$. There are $2^{m-r}=2^3=8$ check sums and $2^r=2^1=2$ received bits in each equation.

The generator matrix is

$$\mathbf{G}=\begin{bmatrix} 1&1&1&1&1&1&1&1&1&1&1&1&1&1&1&1\\ 0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1\\ 0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1\\ 0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1\\ 0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \end{bmatrix}=\begin{bmatrix}\mathbf{G}_0\\ \mathbf{G}_1\end{bmatrix}$$

Multiplying the information vector by **G** we get codeword vector without error $\mathbf{c}=[c_0, c_1, c_2, \ldots\ldots, c_{15}]$, the received vector $\mathbf{v}=\mathbf{c}+\mathbf{e}$ .If , no error occurs then $\mathbf{v}=\mathbf{c}$. After multiplication, the sequence of codewords thus formed may be expressed to have the following form:

$$\begin{aligned}
c_0 &= i_0\\
c_1 &= i_0+i_4\\
c_2 &= i_0+i_3\\
c_3 &= i_0+i_3+i_4\\
c_4 &= i_0+i_2\\
c_5 &= i_0+i_2+i_4\\
c_6 &= i_0+i_2+i_3\\
c_7 &= i_0+i_2+i_3+i_4\\
c_8 &= i_0+i_1\\
c_9 &= i_0+i_1+i_4\\
c_{10} &= i_0+i_1+i_3\\
c_{11} &= i_0+i_1+i_3+i_4\\
c_{12} &= i_0+i_1+i_2\\
c_{13} &= i_0+i_1+i_2+i_4\\
c_{14} &= i_0+i_1+i_2+i_3\\
c_{15} &= i_0+i_1+i_2+i_3+i_4
\end{aligned}$$

If the checksums are formed by

$c_0+c_1 \bmod 2 = i_0+i_0+i_4 \bmod 2 = i_4$
$c_2+c_3 \bmod 2 = 2i_0+2i_3+i_4 \bmod 2 = i_4$
$c_4+c_5 \bmod 2 = i_4$
$c_6+c_7 \bmod 2 = i_4$
$c_8+c_9 \bmod 2 = i_4$
$c_{10}+c_{11} \bmod 2 = i_4$
$c_{12}+c_{13} \bmod 2 = i_4$
$c_{14}+c_{15} \bmod 2 = i_4$

But in the presence of error these equations would be

$v_0+v_1 \bmod 2 = i_4^{(1)}$
$v_2+v_3 \bmod 2 = i_4^{(2)}$
$v_4+v_5 \bmod 2 = i_4^{(3)}$
$v_6+v_7 \bmod 2 = i_4^{(4)}$
$v_8+v_9 \bmod 2 = i_4^{(5)}$
$v_{10}+v_{11} \bmod 2 = i_4^{(6)}$
$v_{12}+v_{13} \bmod 2 = i_4^{(7)}$
$v_{14}+v_{15} \bmod 2 = i_4^{(8)}$

All the $i_4^{(j)}$ for j =1,2, ... , 8 are estimates of information bit i_4 . In the absence of error all these estimates are equal to i_4 , But if there are at most (8/2 -1= 3) errors , the majority of the check sums (8-3=5) will still equal to i_4 . Thus if the error is within correctable range we get the value of i_4 . If more 4 errors occur , then there is no majority ,so error is detected , but not correctable . If more than (t+1) errors , i.e. more than 4 errors occur then a wrong decision about the information bit under consideration is the result of decoding .

To get the value of i_3 , the check sums are formed by

$v_0+v_2 \bmod 2 = i_3^{(1)}$
$v_1+v_3 \bmod 2 = i_3^{(2)}$
$v_4+v_6 \bmod 2 = i_3^{(3)}$
$v_5+v_7 \bmod 2 = i_3^{(4)}$
$v_8+v_{10} \bmod 2 = i_3^{(5)}$
$v_9+v_{11} \bmod 2 = i_3^{(6)}$
$v_{12}+v_{14} \bmod 2 = i_3^{(7)}$
$v_{13}+v_{15} \bmod 2 = i_3^{(8)}$

These eight check sums are formed such that in each equations i_3 appears in one bit (that is odd number of times) and other information bits appear in each equation an even number of times , thus nullifying their contribution on the value of each check sum .

In the same way , each information bit multiplying a row in $\mathbf{G}_r$ can be decoded by setting up 2^{m-r} linear check sums in the 2^m received bits, followed by a majority vote. In this way all the elements of $\mathbf{I}_1$ that is i_1 , i_2 , i_3 , i_4 is computed .After these information bits are known , their contribution to the codeword is subtracted from the received word . This results in the equivalent of a codeword from an (r-1) th order Reed Muller code (all the information bits of $\mathbf{I}_r$ are excluded). It in turn can have the last section of its information bits recovered by the same procedure . The process is repeated until all information bits are recovered. The (1×16) vector $\mathbf{I}_1\mathbf{G}_1$ is subtracted from the (1×16) received vector **v** which is another (1×16) vector.

$$\mathbf{I}_1\mathbf{G}_1 = [i_1, i_2, i_3, i_4] \begin{bmatrix} 0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1 \\ 0&0&0&0&1&1&1&1&0&0&0&0&1&1&1&1 \\ 0&0&1&1&0&0&1&1&0&0&1&1&0&0&1&1 \\ 0&1&0&1&0&1&0&1&0&1&0&1&0&1&0&1 \end{bmatrix}$$

This implies $\mathbf{I}_1\mathbf{G}_1 = \left[c'_0, c'_1, c'_2, c'_3, c'_4, c'_5, c'_6, c'_7, c'_8, c'_9, c'_{10}, c'_{11}, c'_{12}, c'_{13}, c'_{14}, c'_{15}\right]$. .By subtracting $\mathbf{I}_1\mathbf{G}_1$ from **v** we get (r-1) th = 0-th order Reed Muller code .Each bit in the (r-1) th order Reed Muller code can be decoded by setting up $2^{m-(r-1)}$ checksums for each bit (starting from the last bit , then working back to the first bit of the vector section $\mathbf{I}_{r-1}$ in **i**) followed by a majority vote .

$c'_0 = 0$
$c'_1 = i_4$
$c'_2 = i_3$
$c'_3 = i_3 + i_4$
$c'_4 = i_2$
$c'_5 = i_2 + i_4$
$c'_6 = i_2 + i_3$
$c'_7 = i_2 + i_3 + i_4$
$c'_8 = i_1$
$c'_9 = i_1 + i_4$
$c'_{10} = i_1 + i_3$
$c'_{11} = i_1 + i_3 + i_4$
$c'_{12} = i_1 + i_2$
$c'_{13} = i_1 + i_2 + i_4$
$c'_{14} = i_1 + i_2 + i_3$
$c'_{15} = i_1 + i_2 + i_3 + i_4$

$$\begin{aligned}\mathbf{v}' &= \mathbf{v}\text{-}\ \mathbf{I}_1\mathbf{G}_1\\ &= [c_0 , c_1 , c_2 , c_3 , c_4 , c_5 , c_6 , c_7 , c_8 , c_9 , c_{10} , c_{11} , c_{12} , c_{13} , c_{14} , c_{15}]\\ &\quad - \left[c'_0,c'_1,c'_2,c'_3,c'_4,c'_5,c'_6,c'_7,c'_8,c'_9,c'_{10},c'_{11},c'_{12},c'_{13},c'_{14},c'_{15}\right] + \mathbf{e}\ .\\ &= [i_0 , i_0 , i_0 , i_0 , i_0 , i_0 , i_0 , i_0 , i_0 , i_0 , i_0 , i_0 , i_0 , i_0 , i_0 , i_0] + \mathbf{e}\ .\end{aligned}$$

From this , i_0 is decoded by majority vote .

2.3.12 : Single Parity Check Codes :

The very simple parity- check code is one that uses a single parity check bit or check bit , appended to a block of k information bits , the check bit being chosen to satisfy an overall parity rule for the codeword (even or odd parity) . Encoding of the single - parity- check code is described by the equation

$$i_1+ i_2+...+ i_k+ p_1= 0 \bmod 2$$

therefore , $\mathbf{H}$ = [1 1 ...11] , the all terms in between are also 1. Where $i_1, i_2, ..., i_k$ are arbitrary information bits and p_1 is the parity bit .The equation specifies that the parity bit is chosen so that the code word has an even parity , that is an even number of ones. We might just as easily have specified an odd parity rule(in the above equation 1 should be written in place of 0) for setting the check bit , and the properties of the code would have been exactly the same .

We consider the (4,3) binary code whose codewords are listed as follows :

The (4,3) = (n,k) single binary parity check code
(For even parity)
Single parity ,because (n-k)=1.

Information sequences	codewords	
	information	parity
000	000	0
001	001	1
010	010	1
011	011	0
100	100	1
101	101	0
110	110	0
111	111	1

2.3.12.1 : Error Detection Decoding For Single Parity Check (n,n-1) Code :

At the decoder , the received word , possibly containing transmission errors , is checked to determine whether or not the parity check relationship is satisfied . This is done by calculating from the received word ($\mathbf{r}$=[$r_1, r_2, ..., r_n$]) , the sum

$$s = r_1+ r_2+...+ r_n \bmod 2 \qquad (2.30)$$

And observing whether s is 0 or 1 . If s = 0 , the decoder assumes that the codeword was received correctly, the parity check bit is removed and the (n-1) information bits are released to the user. If s =1 , the decoder declares the received word to be in error .

2.3.12.2 : Performance of the decoding rule for the $(n,n\text{-}1)$ code on the binary symmetric channel :

let us consider the error patterns that preserve even parity (s = 0) in the received word , patterns having an even number of errors .On the binary symmetric channel , the probability of occurrence of any particular pattern of i errors is simply *Prob*(i error pattern) = $p^i(1\text{-}p)^{n\text{-}i}$, Where p is the bit error probability in the channel (p<0.5). Total probability of i error patterns for a particular i = $\binom{n}{i}p^i(1-p)^{n-i}$. For all i , i=1,2,...,n total probability of error = $\sum_{i=1}^{n}\binom{n}{i}p^i(1-p)^{n-i}$. If we compare the probabilities of zero error and a two error pattern , we see that for p<0.5 , Since $(1\text{-p})^2 > \text{p}^2$, $(1\text{-}p)^n > p^2(1\text{-}p)^{n\text{-}2}$. Thus , given the result of parity check s = 0 , the probability of no errors in the received word is greater than the probability of any particular two error pattern . It can be easily verified (by changing the above inequality for any even number of errors) , the probability of zero errors is greater than that of any error pattern containing an even number of errors .We can now state that the interpretation of s = 0 as the occurrence of "no error" and s =1 as errors detected is in fact a maximum likelihood *decoding* rule for the binary symmetric channel . *So* , though S = *O* may be *produced* for no error or *even* number of errors in a *received word* , in *decoding* S = *O* is treated as indication of n o error for supporting most probable *event* .

The discussion of the single parity check code has shown that the single check bit permits a decoder merely to classify received words in the two categories , "no error" and "errors detected". Let us consider the (7,6) single parity check code .

$$i_1 + i_2 + i_3 + i_4 + i_5 + i_6 + p_1 = 0 \bmod 2.$$

$\therefore$The parity check matrix **H** = [1 1 1 1 1 1 1]

With single parity check code (even parity), any odd number of errors produce a syndrome equal to 1 , and since all columns of **H** for this code are equal to 1 (that is they are same or not distinct), the nonzero syndrome can yield no information on the location of the errors .

2.3.13 : Binary Repetition Codes :

The simplest type of block code allowing a variable amount of redundancy is the repetition code. With this code a single information bit is encoded into a block of n identical bits , producing an $(n,1)$ code. The code contains only two codewords, The all-zero and the all-one codewords. The repetition code can be discussed without reference to error control coding concepts by observing that the same bit is transmitted by shifting n times . The received n bit word , detected 1 bit at a time , may be operated on by the majority voting rule to detect the actual transmitted bit in each codeword .We assume n is odd . Because for a repetition code with even block length , a received word containing an equal number of ones and zeros does not permit a maximum likelihood decision . In this case the receiver can do no better than announce that the n bit word was received in error . Incorrect decoding decision can be made , but equal number of errors and correct bits can not occur in a codeword .

That this is a maximum likelihood rule can be simply shown with the following example. Consider the transmission of a codeword from the (5,1) code . Suppose that the received word is 01001 . Obviously there are errors in the word , since only 00000 or 11111 could have been sent . Two interpretation are possible : either five zeros were transmitted and two errors occurred (a two error pattern), or five ones are transmitted and three errors occurred (a three error pattern). If we compare the conditional probabilities of these two events for the binary symmetric channel , we find that $p^2(1-p)^3 > p^3(1-p)^2$. For $p < 0.5$. Thus The two error pattern is the event of greater likelihood or maximum likelihood .The transmitted word was 00000 and also by majority vote it is 00000 ,as majority of bits are 0 in the received word). The corresponding analysis can be made of each possible received word to show that the majority -voting rule is a maximum likelihood decision rule for the binary symmetric channel . So,by the (5,1) repetition code all the single and double error patterns are correctable by the majority vote (because if more than 2 errors occur , then by majority voting wrong decision would be made about the transmitted codeword) .

2.3.13.1 : The generalization of the results for a Repetition Code of arbitrary length :

That is the majority voting rule decides between two explanations for the received word by favoring the one that assumes the smaller number of channel errors . It is easy to see that this is a maximum likelihood decoding rule by comparing the likelihood of an i error and a j error event, where $i<j$. For probability of one bit error $p < 0.5$, the likelihood obey the inequality $p^i(1-p)^{n-i} > p^j(1-p)^{n-j}$, $i<j$. Since $(1-p)^{j-i} > p^{j-i}$, $i<j$

2.3.13.2 : Discussion Of Repetition Code With Respect To Error Correcting Coding Concept :

The repetition code as a parity check code :

The $(n,1)$ repetition code can be described as a parity check code in which n-1 parity bits aregenerated from one information bit by the following set of equations :

$$
\begin{array}{llll}
(i_1 + p_1 & & &) \bmod 2 = 0 \\
(i_1 & + p_2 & &) \bmod 2 = 0 \\
(i_1 & & + p_3 &) \bmod 2 = 0 \\
 & & & \\
(i_1 & & & + p_{n\text{-}1} \;\;) \bmod 2 = 0
\end{array}
\qquad (2.31)
$$

These simultaneous equations are called *parity check equations* and can be generalized to describe encoding rule for any parity check block code (because from these equations **H** can be formed and from **H**, the matrix **G** can be derived which is used in encoding). It is readily seen that this set of equations produce a codeword that is a block of n replicas of the information bit .

From the above equations we can write the **H** matrix as :

$$
\mathbf{H} = \begin{bmatrix}
1 & 1 & 0 & 0 & \cdots & 0 \\
1 & 0 & 1 & 0 & \cdots & 0 \\
1 & 0 & 0 & 1 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\
1 & 0 & 0 & 0 & \cdots & 1
\end{bmatrix}
$$

Many of the commonly used algorithms for decoding parity -check codes begin by testing the parity -check relationships in the received word $r_1 , r_2 , \ldots , r_n$ as the first step in decoding . For the $(n,1)$ repetition code , this step can be written as follows :

$$
\begin{array}{lllll}
r_1 + r_2 & & & & = S_1 \\
r_1 & + r_3 & & & = S_2 \\
r_1 & & + r_4 & & = S_3 \\
 & & & & \\
r_1 & & & + \; r_n & = \; S_{n\text{-}1}
\end{array}
$$

(2.32)

It is obvious from examination of the parity check equations (2.31) , that if the received word $r_1 , r_2 , \ldots, r_n$ is error free, equation (2.32) yields n-1 parity check sums S_1,S_2 , ...,$S_{n\text{-}1}$ all equal to zero . Let , us now examine the set of check sums , that is , the **S** vector , resulting from various received error patterns for the example of the (5,1) code . (5,1) repetition code is capable of correcting any error pattern of up to two errors in a received word . In the table given below all such correctable errors (including the all zero pattern) together with the parity check result for each error is shown . The parity check result given in the table may be verified from equation (2.32) by noting that a single error in one of the two bits checked in any parity check equation will produce 1 ,while error in both bits will yield 0 .

Parity check tests for the (5,1) repetition code

Received	Error	Pattern			Parity Test Results			
1	2	3	4	5	S_1	S_2	S_3	S_4
0	0	0	0	0	0	0	0	0
x	0	0	0	0	1	1	1	1
0	x	0	0	0	1	0	0	0
0	0	x	0	0	0	1	0	0
0	0	0	x	0	0	0	1	0
0	0	0	0	x	0	0	0	1
x	x	0	0	0	0	1	1	1
x	0	x	0	0	1	0	1	1
x	0	0	x	0	1	1	0	1
x	0	0	0	x	1	1	1	0
0	x	x	0	0	1	1	0	0
0	x	0	x	0	1	0	1	0
0	x	0	0	x	1	0	0	1
0	0	x	x	0	0	1	1	0
0	0	x	0	x	0	1	0	1
0	0	0	x	x	0	0	1	1

x means bit value is changed at that location .The key point to be observed in the table is that each of the correctable error patterns results in a unique **S** vector in the parity check test . Thus the **S** vector which is called the syndrome vector or simply the syndrome , provides a unique indication for each error pattern that can be corrected by the code . We see , therefore , that as an alternative to a majority voting rule on the received bits , the syndrome may be used in a decoding procedure to determine which of the set of correctable error patterns has actually occurred .

In order to discuss parity check and syndromes more conveniently , we can rewrite the equations in matrix form .To rewrite the parity check equations given in equation (2.31) , we represent the codeword as a row vector $\mathbf{c} = [i_1, p_1 , p_2 , ..., p_{n-1}]$ and the matrix coefficients of the parity check equations as **H** , a matrix having n-1 (i.e. n-k) rows and n columns . The parity check matrix **H** for (5,1) repetition code can be written as

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad (2.33)$$

We can now write the parity check equations (2.31) in the form of a single compact matrix equation $\mathbf{cH}^{T} = \mathbf{0}$ .Where $\mathbf{H}^{T}$ denotes the transpose of **H** and **0** is the row vector of n-1 (number of parity symbols S_i) zeros .Similarly we can rewrite the set of syndrome equations , equation (2.32) , as the single matrix equation $\mathbf{S} = \mathbf{rH}^{T}$. Where **r** is the row vector representing the received word $r_1 , r_2 , ..., r_n$, and **S** is the resulting syndrome row vector .(Note , for typographical convenience , we write codewords , received words , and syndromes as row vectors .If we choose to use column vectors instead for **c**,**r**,**S**, the matrix equations would be written **Hc=0** and **S=Hr**) .

The key point to be observed in examining the matrix form of the syndrome equations is the following .The syndrome **S** resulting from multiplication of the received vector **r** by $\mathbf{H}^{T}$ is the sum of columns of **H** (rows of $\mathbf{H}^{T}$) corresponding to the locations of errors in **r** , by this operation the sum of the columns of **H** are transposed into a row vector . It can be easily verified that regardless of which codeword is transmitted , 00000 or 11111 , the error pattern 0000x will produce **S** = 0001 the fifth column of **H** transposed , and the error pattern 0xx00 will produce **S** = 1100 , which corresponds to the sum of the second and the third columns of **H** transposed . The task of the decoder then is to determine which column of **H** or which sum of columns of **H** produced the observed syndrome . This states the task that must be performed by the decoder for any parity-check block code . At these positions the bit of received word is changed .

A further point to be made with regard to the table of syndromes is that the number of possible four bit syndrome vectors $2^4 = 16$, is exactly equal to the number of error patterns that can be corrected (more can be detected - up to 3 or 4 bit error can be detected but can not be always corrected , because several different groups of columns of **H** may produce the sum equal to the syndrome , so no clear indication about error positions , different column group indicates different positions . 5 bit error can not even be detected with the (5,1) repetition code because in such case the syndrome is the all zero vector - which is an indication of no error . For this code $\binom{5}{0}+\binom{5}{1}+\binom{5}{2}=1+5+10=16$ Therefore , the set of possible syndrome vectors are fully utilized as indicators of the correctable error patterns , that is all error patterns having from 0 to (n-1)/2 errors . This exact sufficiency of parity checks for the required syndrome occurs only in special cases , and the code for which this relationship holds are referred to as *perfect codes* . So , a (5,1) repetition code is a perfect code .

2.3.13.3 : Performance Of Binary Repetition Codes :

Knowing that the (n,1) repetition code with n odd can be used . (if n is even , a received word containing an equal number of ones and zeros does not permit a maximum likelihood decision . In the case the receiver can do no better than announce that the n bit word was received in error) . To correct up to (n-1)/2 errors , we can calculate the probability of correctly decoding a received word on the binary symmetric channel as

$$prob(\text{correct decoding}) = \sum_{i=0}^{\frac{n-1}{2}} \binom{n}{i} p^i (1-p)^{n-i}$$

For n odd , all patterns of (n + 1)/2 or more errors will be decoded incorrectly , so we can also write $prob$(incorrect decoding) = 1- $prob$(correct decoding) . Repetition coding achieves consistently better error performance than uncoded transmission , the improvement increasing with n (more redundancy) . However , if the channel signaling rate is fixed , the use of an (n,1) repetition code causes a reduction of the overall information transmission rate for the system by the ratio 1/n .

Theorem 2.3.13. 7 : An (n,1) repetition code for odd n (n and r are integer for making ^nc_r meaningful) is a perfect code , that it satisfies the Hamming bound with equality :

proof : for t error correcting binary (n,k) code and which is perfect the Hamming bound is $\binom{n}{0}+\binom{n}{1}+\binom{n}{2}+\dots+\binom{n}{t}\Big]2^k = 2^n$ (2.34)

Where $2t+1 = n$, $\Rightarrow t = (n-1)/2$. We prove the above theorem by induction.

Basis : for (3 ,1) repetition code $\binom{3}{0}+\binom{3}{1}\Big]2^1 = (1+3)\times 2 = 8 = 2^3$ .That is the Hamming bound is satisfied with equality .

I

Induction : Let , the bound defined by equation (2.34) is true for (n,1) repetition code for n odd . Let us , prove it for code word size n+2. we , have to show

$$\binom{n+2}{0}+\binom{n+2}{1}+\dots+\binom{n+2}{\frac{n+1}{2}}\Bigg]2^k = 2^{n+2}$$

We know ,

$^{m+n}c_r = {}^mc_r + {}^mc_{r-1}.{}^nc_1 + {}^mc_{r-2}.{}^nc_2 + \dots + {}^mc_1.{}^nc_{r-1} + {}^nc_r$

Therefore

$$^{n+2}c_0 = {}^{n}c_0$$

$$^{n+2}c_1 = {}^{n}c_1 + {}^{n}c_0 \cdot {}^{2}c_1$$

$$^{n+2}c_2 = {}^{n}c_2 + {}^{n}c_1 \cdot {}^{2}c_1 + {}^{n}c_0 \cdot {}^{2}c_2$$

$$^{n+2}c_{\frac{n+1}{2}} = {}^{n}c_{\frac{n+1}{2}} + {}^{n}c_{\frac{n-1}{2}} \cdot {}^{2}c_1 + {}^{n}c_{\frac{n-3}{2}} \cdot {}^{2}c_2$$

Adding $^{n+2}c_0 + {}^{n+2}c_1 + {}^{n+2}c_2 + \ldots + {}^{n+2}c_{\frac{n+1}{2}} = \left. {}^{n}c_0 + {}^{n}c_1 + {}^{n}c_2 + \ldots + {}^{n}c_{\frac{n+1}{2}} \right) +$

$$\left. {}^{2}c \quad {}^{n}c_0 + {}^{n}c_1 + {}^{n}c_2 + \ldots + {}^{n}c_{\frac{n-1}{2}} \right) +$$

$$\left. + {}^{2}c_2 \quad {}^{n}c_0 + {}^{n}c_1 + {}^{n}c_2 + \ldots + {}^{n}c_{\frac{n-3}{2}} \right)$$

$$= {}^{n}c_0 + {}^{n}c_1 + {}^{n}c_2 + \ldots + {}^{n}c_{\frac{n-3}{2}} + {}^{n}c_{\frac{n-1}{2}} + {}^{n}c_{\frac{n+1}{2}} +$$

$$\left. 2 \quad {}^{n}c_0 + {}^{n}c_1 + {}^{n}c_2 + \ldots + {}^{n}c_{\frac{n-3}{2}} + {}^{n}c_{\frac{n-1}{2}} \right) +$$

$${}^{n}c_0 + {}^{n}c_1 + {}^{n}c_2 + \ldots + {}^{n}c_{\frac{n-3}{2}}$$

But , $^{n}c_r = {}^{n}c_{n-r}$

So , $^{n}c_{\frac{n+1}{2}} = {}^{n}c_{\frac{n-1}{2}}$

Therefore , $^{n+2}c_0 + {}^{n+2}c_1 + {}^{n+2}c_2 + \ldots + {}^{n+2}c_{\frac{n+1}{2}}$

$$= {}^{n}c_0 + {}^{n}c_1 + {}^{n}c_2 + \ldots + {}^{n}c_{\frac{n-3}{2}} + {}^{n}c_{\frac{n-1}{2}} + {}^{n}c_{\frac{n-1}{2}} +$$

$$\left. 2 \quad {}^{n}c_0 + {}^{n}c_1 + {}^{n}c_2 + \ldots + {}^{n}c_{\frac{n-3}{2}} + {}^{n}c_{\frac{n-1}{2}} \right) +$$

$$ {}^nc_0 + {}^nc_1 + {}^nc_2 +\ldots+ {}^nc_{\frac{n-3}{2}} $$

$$ = {}^nc_0 + {}^nc_1 + {}^nc_2 +\ldots+ {}^nc_{\frac{n-3}{2}} + {}^nc_{\frac{n-1}{2}} + $$

$$ 2\ {}^nc_0 + {}^nc_1 + {}^nc_2 +\ldots+ {}^nc_{\frac{n-3}{2}} + {}^nc_{\frac{n-1}{2}} \Big) + $$

$$ {}^nc_0 + {}^nc_1 + {}^nc_2 +\ldots+ {}^nc_{\frac{n-3}{2}} + {}^nc_{\frac{n-1}{2}} \ . $$

$$ = 4\ {}^nc_0 + {}^nc_1 + {}^nc_2 +\ldots+ {}^nc_{\frac{n-3}{2}} + {}^nc_{\frac{n-1}{2}} \Big) $$

$$ = 2^2\ {}^nc_0 + {}^nc_1 + {}^nc_2 +\ldots+ {}^nc_{\frac{n-3}{2}} + {}^nc_{\frac{n-1}{2}} \Big) \qquad (2.35) $$

$$ \therefore \ \binom{n+2}{0}+\binom{n+2}{1}+\ldots+\binom{n+2}{\frac{n+1}{2}}\Big]2^k \qquad \text{(using equation 2.35)} $$

$$ = 2^2\ {}^nc_0 + {}^nc_1 + {}^nc_2 +\ldots+ {}^nc_{\frac{n-3}{2}} + {}^nc_{\frac{n-1}{2}} \Big)2k $$

$= 2^{n+2}$,as the Hamming bound is satisfied with equality , so $(n,1)$ repetition code for n odd is a perfect code .

Necessity of n to be odd is explained before hand .

2.3.14 : Perfect And Quasi-Perfect Codes :

Let us visualize small sphere about each of the codewords of a code , each sphere with the same radius (an integer) . Let us allow these spheres to increase in radius by integer amounts until they cannot be made larger without causing some spheres to intersect .That value of the radius is equal to the number of errors that can be corrected by the code .It is called the *packing radius* of the code . Now let us allow the radii to continue to increase in by integer amounts , until every point in the space is contained in at least one sphere .That radius is called the *covering radius* of the code .

The packing radius and covering radius may be equal ; if so , then the construction of the standard array stops just when the sphere of radius t is completed . All points of the space are used in these spheres , none are left over .This is true for perfect codes .

A *perfect code* is one for which there are equal radius spheres about the codewords that are disjoint and completely fill the space .A perfect code satisfies the Hamming bound with equality. A Hamming code with blocklength $n=\frac{q^m-1}{q-1}$ is perfect . Because $1+(q-1)\binom{n}{1}\Big]q^k=q^n$ $\Rightarrow\left[1+(q-1)n\right]=q^{n-k}=q^m$. That is $[1+(q-1)n]q^k=q^n$ points for q^k codewords . ∴ $1+(q-1)n=q^{n-k}=q^m$ points for a codeword or sphere of radius one (this is clear from the term $\left.\begin{matrix}n\\1\end{matrix}\right)$, as this relation is for single error correcting) . The number of points in the space is q^n . The number of spheres or valid codewords is q^k . ∴The number of points in the space divided by the number of spheres is $q^n/q^k=q^{n-k}=q^m$ (as n-$k=m$ = number of parity symbols).

Perfect codes , when they exists , have nice properties and are aesthetically pleasant . But , they are not often important in practical applications , because they are so rare .

Quasi-perfect code is one in which spheres of radius t about each codeword are disjoint and all words not in such a sphere are at a distance t +1 from at least one codeword .

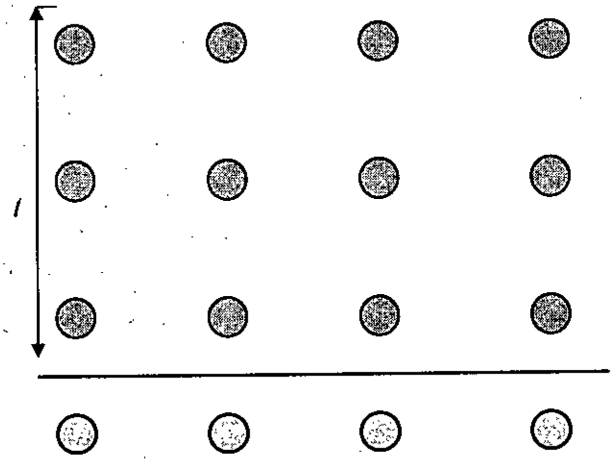


Figure 2.2 : Quasi-perfect code

Quasi-perfect codes are more common than perfect codes . When one exists for a given n and k (and no such perfect code) then for this n and k no other code can have larger d^* . But , these codes are rare and for practical applications have no special importance.

CHAPTER THREE : CYCLIC CODES

§ 3.1 : INTRODUCTION:

3.1.1 : Definition :

A linear code ζ over GF(q) is called a cyclic code if any cyclic shift of a code word is also a codeword.

So, cyclic codes have two properties:-

1. Linearity Property :

The sum of two code words is also a codeword.

2. Cyclic Property :

Any cyclic shift of a codeword is also a codeword.

Let n-tuple $(c_{n-1},..., c_1, c_0)$ denote a codeword in ζ. The code is a cyclic code if left cyclic shift of the code i.e. $(c_{n-2}, c_{n-3},...,c_0, c_{n-1})$ is also a codeword. Similarly $(c_{n-3}, c_{n-4},...,c_0, c_{n-1}, c_{n-2})$, $(c_{n-4}, c_{n-5},...,c_0, c_{n-1}, c_{n-2}, c_{n-3})$ are codewords. Similarly, right cyclic shift of a codeword is also a codeword. Here $c_{n-1},...,c_0$ are from GF(q). Its proof is obvious from the definition.

§ 3.2 : POLYNOMIAL REPRESENTATION OF CYCLIC CODES :

A polynomial over a field GF(q) is a mathematical expression

$$f(x) = f_{n-1}x^{n-1} + f_{n-2}x^{n-2} + ... + f_1x + f_0 \qquad (3.1)$$

where the symbol x is an indeterminate, the coefficients $f_{n-1},..., f_0$ are elements of GF(q) and the indices and exponents are integers.

So from the definition of cyclic property, we may treat the elements from GF(q) of a codeword of length n as the coefficients of the polynomial of degree (n-1) over GF(q). That is, the codeword $c = (c_{n-1}, c_{n-2},..., c_0)$ may be represented in the form of a codeword polynomial as follows :

$$c(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + ... + c_1x + c_0 \qquad (3.2)$$
$$= \sum_{i=0}^{n-1} c_i x^i$$

where x is an arbitrary variable(indeterminate), the coefficients or codeword elements c_{n-1}, c_{n-2},..., c_0 are elements of GF(q).

Each power of x in the polynomial $c(x)$ represents a one bit cyclic shift. Hence multiplication of $c(x)$ by x may be viewed as a cyclic shift or rotation to the left, subject to constraint $x^n = 1$. The constraint $x^n = 1$ achieves two objectives - (1) it keeps the polynomial $xc(x)$ to order (n-1) as the codeword polynomial definition. (2) the coefficients are rotated to the left and thus helps to satisfy cyclic property. These two objectives are cleared in the following.

Multiplication of $c(x)$ by x, subject to the constraint $x^n = 1$ is denoted by a special type of polynomial multiplication as multiplication modulo(x^n - 1). Since the codeword polynomial of equation (3.2) represents codeword, $c = (c_{n-1}, c_{n-2},..., c_0)$, for single cyclic shift, the codeword polynomial becomes -

$$x.c(x) \bmod (x^n - 1) = c_{n-2}x^{n-1} + c_{n-3}x^{n-2} + ... + c_0x + c_{n-1} \qquad (3.3)$$

here mod means 'modulo'.Equation (3.3) is a polynomial representation of a codeword, $c = (c_{n-2}, c_{n-3},..., c_0, c_{n-1})$ as predicted. Similarly for two cyclic shifts, the codeword polynomial becomes -

$$x^2.c(x) \bmod (x^n - 1) = c_{n-3}x^{n-1} + c_{n-4}x^{n-2} + ... + c_0x^2 + c_{n-1}x + c_{n-2} \qquad (3.4)$$

Equation (3.4) is a polynomial representation of the codeword, $c = (c_{n-3}, c_{n-4},..., c_0, c_{n-1}, c_{n-2})$.

So, generally we can describe the cyclic property of the cyclic code in polynomial notation by stating that if $c(x)$ is a codeword polynomial, then the polynomial $x^ic(x) \bmod (x^n-1)$ is also a codeword polynomial for any cyclic shift i.

3.2.1 : Ring Structure :

A ring R is a set with two operations defined: the first is called addition (denoted by +) and the second is called multiplication (denoted by juxtaposition); and the following axioms are satisfied:

1. R is an abelian group under addition(+).
2. Closure For any *a,b* in R, the product *ab* is in R
3. Associative Law: $a(bc)=(ab)c$
4. Distributive Law : $a(b+c)=ab+ac$; $(b+c)a=ba+ca$

A commutative ring is one in which the multiplication is commutative.

The above polynomial notation of the cyclic code can also be described by ring structure. The set of all polynomials over GF(q) forms a ring if addition and multiplication are defined as the usual addition and multiplication of polynomials. Such a polynomial ring for GF(q) is denoted by GF(q)[x].

Now we have to define quotient ring in GF(q)[x]. For any monic polynomial $p(x)$ with nonzero degree over GF(q), the quotient ring is the ring of polynomials modulo $p(x)$ i.e. the set of all polynomials with degree smaller than that of $p(x)$, together with polynomial addition and multiplication modulo $p(x)$.The quotient ring is denoted by GF(q)[x]/$p(x)$.

In the above definition, $p(x)$ is taken as monic polynomial. A monic polynomial is a polynomial with leading coefficient, (e.g. f_{n-1} of equation (3.1)) equal to 1.

From equation (3.3) and (3.4), it is clear that cyclic codes in polynomial notation can be described from the ring GF(q)[x]/(x^n-1). The following theorem is helpful in defining cyclic code in this ring.

Theorem 3.2.1.1:

In the ring GF(q)[x]/(x^n-1) a subset ζ is a cyclic code iff it satisfies the following two properties :

1. ζ is a subgroup of GF(q)[x]/(x^n-1) under addition.
2. If $c(x) \in \zeta$ and $a(x) \in$ GF(q)[x]/(x^n-1), then $R_{x^n-1}[a(x)c(x)] \in \zeta$

Proof :

At first let the subset satisfies the two properties. From property (1), it is closed under addition, that proves the linearity property of cyclic codes. From property (2) it is closed under multiplication by any ring element, particularly multiplication by x. This proves the cyclic property of cyclic codes. Hence it is a cyclic code.

Now let , the subset is a cyclic code. Then from linearity property, it is closed under addition. This proves property (1). From cyclic property, it is closed under multiplication by powers of x. From linearity and cyclic property, it can be said that the subset is closed under multiplication by linear combinations of powers by x. That is, it is closed under multiplication by an arbitrary polynomial, whose coefficients are in GF(q) $[x]/(x^n\text{-}1)$. Hence it proves property (2) . So the assumption is proved.

3.2.2 : Generator Polynomial :

Let, block length of a codeword be n, of which k ($n>k$) be message bits and(n-k) be parity bits. This codeword is represented by (n, k). A (n, k) codeword polynomial has minimum degree (n-k).

Proof :

A (n, k) codeword has (n-k) parity bits. If the codeword polynomial has degree (n-k), then it has (n-k+1) bits : 1 one message bit and (n-k) parity bits. If the codeword polynomial has degree (n-k-1), then it has only (n-k) parity bits, but no message bit. so, this cannot be a codeword. again if the codeword polynomial has the degree less than (n-k-1), then it has not (n-k) parity bits. This cannot also be a codeword. So, a (n, k) codeword polynomial has minimum degree (n-k).

Generator polynomial is a nonzero monic polynomial of smallest degree i.e. (n-k) and is denoted by $g(x)$.

Generator polynomial, $g(x)$ of (n, k) cyclic code has following properties :

Property 3.2.2.1:

Any multiple of the generator polynomial $g(x)$ with degree k-1 or less is a codeword polynomial, $c(x)$, i.e. $c(x) = a(x)g(x)$ for any $a(x)$.

Proof :

Let $c(x) = a(x)g(x) + S(x)$. This means when $c(x)$ is divided by $g(x)$ then quotient = $a(x)$ and remainder = $S(x)$. Here $a(x)g(x)$ must be in the code by theorem (3.2.1.1), because $g(x)$ is in the code.

So, deg $S(x)$ < deg $g(x)$ and $S(x) = c(x) - a(x)g(x)$ is a codeword polynomial, because $c(x)$ is a codeword polynomial and $a(x)g(x)$ is also a codeword polynomial. But deg $S(x)$ < deg $g(x)$ i.e. deg $S(x) < (n\text{-}k)$, which is the smallest degree of any nonzero codeword polynomial. Hence $S(x) = 0$. So, $c(x) = a(x)g(x)$.

Property 3.2.2.2 :

The generator polynomial, $g(x)$ is unique.

Proof :

Let $g(x)$ itself represents a codeword and let there be another monic codeword polynomial of degree (n-k). So, we can get another codeword polynomial by adding these two codeword polynomials. Since highest degree coefficient of these polynomials are 1, then the new codeword polynomial has degree less than $(n\text{-}k)$. This is impossible, since $(n\text{-}k)$ is the minimum degree. So, $g(x)$ is unique.

A polynomial $s(x)$ is said to be divisible by the polynomial $r(x)$or that $r(x)$ is a factor of $s(x)$, if there exists a polynomial $a(x)$, such that, $s(x)=a(x)\ r(x)$. And a polynomial $p(x)$ that is divisible only by $\alpha\, p(x)$ orα , where α is an arbitrary field element in GF(q) is called an irreducible polynomial.

Property 3.2.2.3 :

$g(x)$ divides x^n-1.

Proof :

Let $x^n\text{-}1 = a(x)g(x) + S(x)$. Here deg $S(x) <$ deg $g(x) = (n\text{-}k)$. So,

$$R_{x^n-1}(x^n\text{-}1) = R_{x^n-1}[a(x)g(x)] + R_{x^n-1}[S(x)]$$

Here $R_a(b)$ means residue or remainder of (b÷a).

$$\Rightarrow 0 = R_{x^n-1}[a(x)g(x)] + S(x) \qquad [\because \deg S(x) < n\text{-}k]$$

But the first term of the above equation is a codeword polynomial with degree (n-k) or higher. So, $S(x)$ is a codeword polynomial whose degree is less than $(n\text{-}k)$. The only such codeword polynomial $S(x) = 0$.

$$\therefore\ x^n\text{-}1 = a(x)g(x)$$

Thus g(x) divides x^n-1.

For large values of n, the polynomial x^n-1 may have many factors of degree $(n\text{-}k)$. Some of these polynomial factors generate good cyclic codes, which some generates bad.

From property (3.2.2.1), $c(x)= m(x)g(x)$. (3.5)

Here $m(x)$ = message polynomial. Since $c(x)$ is an $(n,\ k)$ cyclic codeword polynomial, deg $c(x) = n$-1, [equation (2)] and deg $g(x) = n\text{-}k$ [$g(x)$ definition]. So,

$$\begin{aligned}\deg m(x) &= \deg c(x) \text{ -}\deg g(x)\\ &= (n\text{-}1) - (n\text{-}k)\\ &= k\text{-}1.\end{aligned}$$

In equation (3.5), $c(x)$ is a nonsystematic codeword polynomial because message polynomial $m(x)$ is not immediately visible in $c(x)$. How to get a systematic code is described below :

Let $m(x) = m_{k\text{-}1}x^{k\text{-}1} + m_{k\text{-}2}x^{k\text{-}2} +\ldots+ m_0.$ (3.6)

So, message sequence is $(m_{k\text{-}1}, m_{k\text{-}2},\ldots, m_0)$. Again let, parity polynomial,

$p(x) = p_{n\text{-}k\text{-}1}x^{n\text{-}k\text{-}1} + p_{n\text{-}k\text{-}2}x^{n\text{-}k\text{-}2} +\ldots+ p_0.$ (3.7)

[In $(n,\ k)$ code, there are $(n\text{-}k)$ parity bits. So, degree of parity polynomial = $n\text{-}k\text{-}1$]

So, parity sequence is $(p_{n-k-1}, p_{n-k-2},..., p_0)$.

In (n, k) systematic cyclic codes, message bits are transmitted in unaltered, as shown by the following structure for a codeword :

$$\underbrace{m_{k-1}, m_{k-2}, \cdots, m_0}_{k \text{ message bits}}, \underbrace{p_{n-k-1}, p_{n-k-2}, \cdots, p_0}_{(n-k) \text{ parity bits}}$$

According to the structure specified for a codeword, the leading message bit m_{k-1} is the coefficient of x^{n-1} in the corresponding codeword polynomial representation.. To meet this requirement, we multiply the message polynomial $m(x)$ by x^{n-k}. Then the codeword polynomial is of the following form :

$$c(x) = x^{n-k}m(x) + p(x) \tag{3.8}$$

$$\Rightarrow \mathrm{R}_{g(x)}[\mathrm{c(x)}] = \mathrm{R}_{g(x)}[x^{n-k}m(x)] + \mathrm{R}_{g(x)}[p(x)]$$

$$\Rightarrow 0 = \mathrm{R}_{g(x)}[x^{n-k}m(x)] + p(x)$$

$$[\because \deg p(x) = n\text{-}k\text{-}1 < \deg g(x) = n\text{-}k]$$

$$\Rightarrow p(x) = -\mathrm{R}_{g(x)}[x^{n-k}m(x)] \tag{3.9}$$

Thus parity polynomial $p(x)$ of equation (3.8) can easily be chosen from equation (3.9).

In fact, systematic code from equation (3.8) and nonsystematic code from equation (3.5) is same, but the association between the $m(x)$ and $c(x)$ is different. Now we want to find the possible generator polynomials for a (n, k) cyclic code.

From property (3.2.2.3), we have seen that generator polynomial $g(x)$ over GF(q) of (n, k) cyclic code divides $(x^n\text{-}1)$. Now factoring $(x^n\text{-}1)$, we get

$$x^n\text{-}1 = f_1(x)f_2(x)...f_s(x) \tag{3.10}$$

where s is the number of prime polynomial, $f(x)$ over GF(q). A monic irreducible polynomial of degree of at least 1 is called a *prime polynomial*.

So, any subset of the prime polynomials of equation (3.10) can be multiplied together to produce a generator polynomial, $g(x)$. If the prime polynomials are distinct then there are 2^s different combinations of prime polynomials to produce $g(x)$. Among these 2^s combinations, we have to exclude trivial cases $g(x) = 1$ and $g(x) = (x^n\text{-}1)$ to get nontrivial cyclic codes. So, there may be $(2^s\text{-}2)$ different nontrivial (n, k) cyclic codes. Now, we want to find prime polynomials of equation (3.10).

There are two types of cyclic codes - (i) Primitive and (ii) Nonprimitive cyclic codes.

A blocklength n of the form $n = q^m\text{-}1$ is called a *primitive blocklength* for a code over GF(q) for nonzero positive integer m. A cyclic code over GF(q) of primitive blocklength is called a *primitive cyclic code*. At first we want to find prime polynomials of equation (3.10) for primitive cyclic codes.For this, we have to know, what are extension field and minimal polynomial.

Let F be a field. A subfield of F is called a subfield if it is a field under the inherited addition and multiplication. The original field F is then called an *extension field* of the subfield.

Let, GF(q) be a field and GF(Q) be an *extension field* of GF(q). Also let β be in over GF(Q). The prime polynomial, $f(x)$ of smallest degree over GF(q) with $f(\beta) = 0$ is called the *minimal polynomial* of β over GF(q). Since minimal polynomials are prime polynomial in GF(q), they are irreducible in GF(q).

Now, return to the original description. Since, for (n, k) primitive cyclic code, $n = q^m$-1. Putting this value in equation (3.10), we get,

$$x^{q^m-1} - 1 = f_1(x)f_2(x)\cdots f_s(x) \qquad (3.11)$$

Here each f_l (x) (for l = 1,..., s) is a prime polynomial over GF(q). The GF(q^m) is an extension field of GF(q). We know from definition of galois field there are (q^m-1) nonzero elements in GF(q^m). Let the nonzero elements are β1, β2,..., β_{q^m-1} in GF(qm). Then

$$x^{q^m-1} - 1 = (x-\beta_1)(x-\beta_2)\cdots(x-\beta_{q^m-1}) \qquad (3.12)$$

Before proving this equation, we have to prove that - the order of any element of GF(q^m) divides the order of GF(q^m).

Proof :

Let , H is a subgroup of finite group G , in the process of construction of coset decomposition . From the rectangular structure of the coset decomposition

(Order or number of elements of H).(Number of cosets or rows of G) =
(Order or number of elements of G).

$\Rightarrow n \, . \, m = x$

If H is formed from h , then the order of h is n . Also h is an element of G . So the order of an element of G (here h) divides the order (x) of G . This can be shown for all elements of G , choosing them as coset leaders (may be for different coset decomposition) . So it is proved that the order of a finite group is divisible by the order of any of its elements.

Since, a GF is a finite group, The above proof can be applied to prove the statement.

Now we can prove the equation (3.12).

Proof :

GF(q^m) has (q^m-1) nonzero elements. So, order of GF(q^m) is equal to (q^m-1). But we know that orders of elements of GF(q^m) divides the order of GF(q^m), i.e. (q^m-1). And order of an element β, in GF(q^m) is defined as the smallest positive integer p for which, β^p = an identity element, e.
(3.13)

Here since the nonzero elements of the GF(q^m) form an Abelian group under multiplication operation, identity element, e = 1(see definition of Galois Field in Linear Block Codes). Since p divides (q^m-1),

let $q^m\text{-}1 = bp$ for some positive integer b. (3.14)

$\Rightarrow \beta^{bp} = 1^b = 1$ [from equation (3.13)]

$\Rightarrow \beta^{q^m-1} = 1$ [from equation (3.14)]

$\Rightarrow \beta^{q^m-1}\text{-}1 = 0$

$\Rightarrow \beta_i^{q^m-1} - 1 = 0$ [$\because \beta$ represents any nonzero element of GF(q^m)]

for $0 < \mathrm{i} \le q^m\text{-}1$ (3.15)

This equation implies that each β_i is a root of the polynomial $x^{q^m-1} - 1$. Therefore,

$$x^{q^m-1} - 1 = (x-\beta_1)(x-\beta_2)\cdots(x-\beta_{q^m-1}).$$

Comparing between equation (11) and equation (12) each $f_l(x)$ (for l = 1,..., s) can be factored in GF(q^m) into a product of some of the (x-β_j) (for j = 1,..., q^m-1) linear terms. This, $f_l(x)$ is the minimal polynomial of β_j. Thus, generator polynomial, $g(x)$ is the product of any subset of the minimal polynomials of zeros of GF(q^m). Since zeros of GF(q^m) can easily be found from equation (3.12), to find $g(x)$, we have to find zeros of GF(q^m) at first, then minimal polynomials of these zeros.

But $g(x)$ may not be composed of all the minimal polynomials. So, we have to find these minimal polynomials. Let, these polynomials have zeros β_1, β_2,..., β_r in GF(q^m). So, $g(x)$ has also zeros β_1, β_2, ..., β_r in GF(q^m). We have to distinguish these zeros from others. Before this, we have to proof the following theorem.

Theorem 3.2.2.2 :

A polynomial $c(x)$ over GF(q) is a codeword polynomial iff $c(\beta_1) = c(\beta_2) = \ldots = c(\beta_r) = 0$. When $c(\beta_i)$ is calculated in GF(q^m) for i = 1,..., r.

Proof :

At first let $c(x)$ be codeword polynomial. So,

$c(x) = m(x)g(x)$ for some $m(x)$.

$$\Rightarrow c(\beta_i) = m(\beta_i)g(\beta_i)$$
$$= m(\beta_i).0 \qquad [\beta_i \text{ are zeros of } g(x)]$$
$$= 0$$

Now, let $c(\beta_i) = 0$. We can write, $c(x) = Q(x)f_i(x) + R(x)$. Here deg $R(x) <$ deg $f_i(x)$ and $f_i(x)$ is the minimal polynomial of β_i.

Therefore, $c(\beta_i) = Q(\beta_i)f_i(\beta_i) + R(\beta_i)$.

$$\Rightarrow 0 \quad = Q(\beta_i).0 + R(\beta_i).$$
$$\Rightarrow R(\beta_i) = 0.$$

That is, $R(x) = 0$. Then $c(x)$ must be divisible by the minimal polynomials, $f_i(x)$. Hence $c(x) = Q(x)f_i(x)$. So, $c(x)$ is a codeword, as $f_i(x)$ forms $g(x)$.

From theorem (3.2.2.2), we can easily distinguish desirable zeros $\beta_1, \beta_2,..., \beta_r$ in GF(q^m) from others. Again minimal polynomials may have degree greater than 1. So, they must have more than one zeros in GF(q^m). For this reason, if $f_1(x),...,f_r(x)$ are minimal polynomials of $\beta_1, \beta_2,..., \beta_r$, then two β s may have same minimal polynomials, i.e. $f(x)$s may not be distinct. So,

$$g(x) = \text{LCM}[f_1(x),...,f_r(x)] \tag{3.16}$$

Let the minimal polynomial $f(x)$ of β have degree m', then it must have m' zeros in GF(q^m). Therefore, if we can find these zeros, we can then find $f(x)$. We can identify the additional zeros of $f(x)$ (other than β)with the following two theorems-

Theorem 3.2.2.3 :

The number of elements in the smallest subfield of GF(q) is called the characteristics of GF(q). Let p be characteristics of the field GF(q). Then for any polynomial $s(x)$ over GF(q) and any integer m.

$$[s(x)]^{p^m} = \left[\sum_{i=0}^{\deg s(x)} s_i x^i\right]^{p^m} = \sum_{i=0}^{\deg s(x)} s_i^{p^m} x^{i p^m}$$

and also if p is replaced by any power of p.

Proof :

Here characteristic of GF(q) is p. Each galois field contains a unique smallest subfield, which has a prime number of elements. Hence the characteristic of every galois field is a prime number. The significance of characteristic, p of Galois field is that all integer arithmetic is modulo p in that galois field. Now we start to prove the theorem. Let

$$s(x) = s'(x)\,x + s_0.$$
$$\Rightarrow [s(x)]^p = [\,s'(x)\,x + s_0]^p.$$

$$\Rightarrow [s(x)]^p = \sum_{i=0}^{P} \binom{p}{i}[s'(x)x]^i s_0^{p-i} \tag{3.17}$$

Here $\binom{p}{i} = \frac{p!}{i!(p-i)!} = \frac{p(p-1)!}{i!(p-i)!}$ and p is a prime and does not appear in the denominator unless $i = 0$ or $i = p$. Since all integer arithmetic is modulo p in GF(q), except for $i = 0$ or p, $\binom{p}{i}$ is a multiple of p and equals zero modulo p. So, from equation (3.17).

$$[s(x)]^p = \binom{p}{0}[s'(x)x]^0 s_0^{p-0} + \binom{p}{p}[s'(x)x]^p s_0^{p-p}$$

$$= s_0^p + [s'(x)]^p x^p$$

Now, applying the same reasoning to $s'(x)$ and containing in this way, we can write,

$$[s(x)]^p = \sum_{i=0}^{\deg s(x)} s_i^p x^{ip} \tag{3.18}$$

$$\Rightarrow \left[\sum_{i=0}^{\deg s(x)} s_i x^i\right]^p = \sum_{i=0}^{\deg s(x)} s_i^p x^{ip} \tag{3.19}$$

So,

$$[s(x)]^{p^2} = \left[[s(x)]^p\right]^p$$

$$= \left[\sum_{i=0}^{\deg s(x)} s_i^p x^{ip}\right]^p \qquad \text{[From (3.18)]}$$

$$= \sum_{i=0}^{\deg s(x)} s_i^{p^2} x^{ip^2} \qquad \text{[From (3.19)]}$$

This calculation can be repeated for any times. So,

$$[s(x)]^{p^m} = \left[\sum_{i=0}^{\deg s(x)} s_i x^i\right]^{p^m}$$

$$= \sum_{i=0}^{\deg s(x)} s_i^{p^m} x^{ip^m} \tag{3.20}$$

Thus the theorem is proved.

Theorem 3.2.2.4 :

If $f(x)$ is the minimal polynomial over GF(q) of β, an element of GF(q^m). The $f(x)$ is also the minimal polynomial of β^q.

Proof :

It is known that q is some power of p.

Therefore, from equation (3.18)

$$[f(x)]^q = \sum_{i=0}^{\deg f(x)} f_i^q x^{iq} . \tag{3.21}$$

But in GF(q), all elements f_i satisfy

$$f_i^{q-1} = 1$$
$$\Rightarrow f_i^q = f_i$$

From (3.21)

$$[f(x)]^q = \sum_{i=0}^{\deg f(x)} f_i (x^q)^i$$
$$= f(x^q)$$

But $f(\beta) = 0$, so $[f(\beta)]^q = f(\beta^q) = 0$.Thus β^q is a zero of $f(x)$, i.e. $f(x)$ is also the minimal polynomial of β^q.

From the above theorem, if $f(x)$ is the minimal polynomial of β, it is also that of β^q. Here β and β^q are called conjugates. So, two elements of GF(q^m) are said to be conjugates, if they share the same minimal polynomial over GF(q). Similarly if $f(x)$ is minimal polynomial of β^q, it is also that of $(\beta^q)^q = \beta^{q^2}$ and so forth. Thus $f(x)$ is the minimal polynomial of any element of the set $\left\{\beta, \beta^q, \beta^{q^2}, \cdots, \beta^{q^{r-1}}\right\}$, where r is the smallest integer such that $\beta^{q^r} = \beta$. But in GF(q^m) $\beta^{q^m} = \beta^{q^m-1}.\beta = 1.\beta = \beta$, so $r \leq m$. Obviously, $r \geq 1$. If $r < 1$, then the power of β, q^{r-1} is not an integer. Thus $1 \leq r \leq m$. This above set is called a set of conjugates, because all elements of the set are conjugates. So, a single element can have upto m conjugates. Thus

$$f(x) = (x-\beta)(x-\beta^q)\cdots(x-\beta^{q^{r-1}}) \tag{3.22}$$

So, if we can know a nonzero element β in GF(q^m), then we can easily calculate its minimal polynomial $f(x)$ in GF(q).

From the above discussion, we can write, in short, to find the generator polynomial, $g(x)$ of primitive cyclic code -

(i) Find the nonzero elements of GF(q^m) by equation (3.12).

(ii) Among these elements, find the zeros of $g(x)$ by theorem (3.2.2.2).

(iii) For each of these zeros, find the minimal polynomial.

(iv) $g(x)$ = LCM of the above minimal polynomials.

Now we want to find prime polynomials of equation (3.10) for not primitive cyclic codes.

For primitive cyclic codes, these prime polynomials can easily be found by finding nonzero elements of GF(q^m) from equation (3.12). In that case, $n = q^m-1$. As a result, $\left(x^{q^m-1}-1\right)$ is divisible by $g(x)$, since $g(x)$ divides (x^n-1). But here $n \neq q^m-1$. So, if we can take n, q, m such that (x^n-1) divides $\left(x^{q^m-1}-1\right)$ then, the process to find g(x) of primitive cyclic codes can be applied to not primitive cyclic codes. For this, the following theorem is helpful :

Theorem 3.2.2.5 :

If n and q are relatively prime, then (x^n-1) divides $\left(x^{q^m-1}-1\right)$, for some m.

Proof :

At first, we have to prove n divides q^m-1 for some m. For this reason, we have to write following (n+1) equations :

$$q = Q_1n+S_1$$
$$q^2 = Q_2n+S_2$$
$$q^3 = Q_3n+S_3$$
$$\cdot \quad \cdot \quad \cdot$$
$$\cdot \quad \cdot \quad \cdot$$
$$\cdot \quad \cdot \quad \cdot$$
$$q^n = Q_nn+S_n$$
$$q^{n+1} = Q_{n+1}n+S_{n+1}$$

Since in the above equations, q, q^2,..., $q^{n+1}$are divided by n so all remainders are between 0 and n-1. Because there are (n+1) remainders, two must be same. let, S_i and S_j be same and $i < j$. Then

$$q^j - q^i = nQ_j + S_j - nQ_i - S_i$$
$$\Rightarrow q^i(q^{j-i}-1) = n(Q_j - Q_i) \quad [\because S_j = S_i]$$

Because n is relatively prime to q, n must divide q^{j-i}-1. Setting $m = j - i$, we get n divides (q^m-1).

Now let $q^m-1 = nb$ for some nonzero positive integer b. So,

$$x^{q^m-1}-1 = x^{nb}-1 = (x^n)^b-1$$
$$= (x^n-1)(x^{n(b-1)} + x^{n(b-2)} + ... + x^n + 1)$$

Therefore, (x^n-1) divides $x^{q^m-1}-1$.

So, for a (n, k) cyclic code in GF(q), if n, q are relatively prime then (x^n-1) divides $\left(x^{q^m-1}-1\right)$. Since $g(x)$ divides (x^n-1), $g(x)$ also divides $x^{q^m-1}-1$.

Before further proceeding, we have to know what the primitive element is and its property.

A *primitive element* field element of GF(q) is an element α such that every field element except zero can be expressed as a power of α.

Property 3.2.2.4:

Order of primitive element, α in GF(q) = (q-1).

Proof :

The number c is called order of α if $\alpha^c = 1$ in GF(q). So, we have to prove

$$c = q\text{-}1.$$

In GF(q), There are q elements of which (q-1) nonzero elements. Since, these (q-1) elements can be represented as a power of α (α, α^2, α^3,...), highest power of α to represent these elements must be (q-1) and $\alpha^{q\text{-}1} = 1$, otherwise 1 is not in GF(q). So, $c = q$-1.

Now, let α be primitive in GF(q^m), q^m-1 = nb and $\beta = \alpha^b$. Then all zeros of $g(x)$ are restricted to powers of β for nonprimitive cyclic codes.

Proof :

Since α is a primitive element of GF(q^m), so, order of α is (q^m-1)

i. e. $\alpha^{q^m-1} = 1$.

$$\text{Now, } \beta^{(q^m-1)/b} = \left(\alpha^b\right)^{(q^m-1)/b} \qquad [\beta = \alpha^b]$$

$$= \alpha^{q^m-1}$$

$$= 1$$

So, β is a zero of $\left(x^{(q^m-1)/b} - 1\right)$ polynomial. Hence minimal polynomial of β divides $\left(x^{(q^m-1)/b} - 1\right)$ and serves as a generator polynomial for some cyclic blocklength, n = (q^m-1)/b. Since this generator polynomial $g(x)$ is a minimal polynomial of β, from equation (3.22), $g(x) = (x\text{-}\beta\ )(x\text{-}\beta\ q)\ldots\left(x - \beta^{q^r-1}\right)$ for smallest r, for which $\beta^{q^r} = \beta$ in GF(q^m). Similarly β^i is a zero of $\left(x^{(q^m-1)/b} - 1\right)$ $\left[\beta^{i(q^m-1)/b} - 1 = \alpha^{bi(q^m-1)/b} = \left(\alpha^{(q^m-1)}\right)^i = 1^i = 1\right]$ for any i. Thus all zeros are restricted to the power of β.

In summary, if we use $\beta = \alpha^b$ in place of α and restrict zeros to powers of β, then we obtain a cyclic code of blocklength n = (q^m-1)/b

From the above theorem, we can easily find $g(x)$ of nonprimitive cyclic codes.

3.2.3 : Parity Check Polynomial :

From property (3.2.2.3) of generator polynomial x^n-1 = $h(x)g(x)$, for some polynomial $h(x)$. This $h(x)$ polynomial is called parity-check polynomial. Since deg $g(x) = n$-k, then deg $h(x) = n$-$(n$-$k) = k$. This parity-check polynomial has following property :

Property 3.2.3.1:

Each codeword $c(x)$ satisfies

$$R_{x^n-1}[c(x)h(x)] = 0 \qquad (3.23)$$

Proof :

$$c(x)h(x) = m(x)g(x)h(x)$$

$[c(x) = m(x)g(x)$

$m(x)$ = message polynomial

$g(x)$ = generator polynomial]

$$= [g(x)h(x)]m(x)$$

$$= (x^n-1)m(x) \qquad [\because x^n-1 = g(x)h(x)]$$

$$R_{x^n-1}[c(x)h(x)] = R_{x^n-1}[(x^n-1)m(x)]$$

$$= 0.$$

3.2.4 : Syndrome Polynomial :

Syndrome polynomial is calculated to detect and correct errors in the received word.

Let $c(x)$ be transmitted codeword polynomial, $v(x)$ received codeword polynomial, $e(x)$ error polynomial. $e(x)$ has nonzero coefficient in those locations where channel errors occur. Thus,

$$v(x) = c(x) + e(x) \qquad (3.24)$$

$v(x) = c(x)$ when $e(x) = 0$, i.e. there is no error in transmitting messages.

Syndrome polynomial, $s(x)$ is defined as the remainder of $v(x)$ divided by the generator polynomial, $g(x)$, i.e.

$$s(x) = R_{g(x)}[v(x)] \qquad (3.25)$$

So, $s(x)$ has degree less than deg $g(x)$. That is

$$\deg s(x) = \deg g(x) - 1$$

$$= n\text{-}k\text{-}1$$

The syndrome polynomial, $s(x)$ has the following properties :

Property 3.2.4.1 :

$s(x)$ depends only on error polynomial, $e(x)$ not on codeword polynomial, $c(x)$ or on message polynomial, $m(x)$.

Proof :

From equation (3.25) $s(x) = R_{g(x)}[v(x)]$

$$= R_{g(x)}[c(x) + e(x)] \quad \text{[From equation(3.24)} \; v(x) = c(x) + e(x)]$$

$$= R_{g(x)}[c(x)] + R_{g(x)}[e(x)]$$

$$= 0 + R_{g(x)}[e(x)]$$

$$[\because c(x) = g(x)m(x)]$$

$$= R_{g(x)}[e(x)].$$

Property 3.2.4.2 :

Let d^* be the minimum distance of a cyclic code ζ. Every error polynomial of weight less than ½ d^* has a unique syndrome polynomial.

Proof :

Suppose $e_1(x)$ and $e_2(x)$ each have weight less than ½ d^* and the same syndrome polynomial. Then

$$e_1(x) = Q_1(x)g(x) + S(x)$$
$$e_2(x) = Q_2(x)g(x) + S(x)$$

and

$$e_1(x) - e_2(x) = [Q_1(x) - Q_2(x)]g(x)$$

But by assumption, $e_1(x)$ and $e_2(x)$ each have weight less than ½ d^*, and thus the difference has weight less than d^*, the minimum weight of the code. Hence the right side is zero, and $e_1(x)$ equals $e_2(x)$. This proves the property.

To detect and correct errors in the received polynomial, a table is constructed. If the number of entries is not too large, this table can be realized in a memory or in a combinational logic circuit. For each correctable error, there is a syndrome in the table as shown in the following figure :

$e(x)$	$S(x)$
1	$R_{g(x)}[1]$
x	$R_{g(x)}[x]$
x^2	$R_{g(x)}[x^2]$
$\vdots$	$\vdots$
$1+x$	$R_{g(x)}[1+x]$
$1+x^2$	$R_{g(x)}[1+x^2]$
$\vdots$	$\vdots$

This table is called the syndrome evaluator table. When a codeword, $v(x)$ is received, then syndrome polynomial, $s(x)$ is calculated by

$$s(x) = R_{g(x)}\,[v(x)]$$

If $s(x) = 0$, then there is no error. Otherwise, the error can easily be detected from syndrome evaluator table. Since each $s(x)$ has its associated entry of $e(x)$ in the syndrome evaluator table.

§ 3.3 : MATRIX REPRESENTATION OF CYCLIC CODES :

Here derivation of generator matrix and parity-check matrix from the respective polynomials have been discussed. These matrices are often helpful in describing some cyclic codes.

In a (n, k) cyclic code, generator matrix is a k by n matrix and parity-check matrix is a $(n-k)$ by n matrix. At first, we want to generate generator matrix from generator polynomial.

3.3.1 : Generator Matrix Generation :

Let, the generator polynomial,

$$g(x) = g_{n-k}x^{n-k} + g_{n-k-1}x^{n-k-1} + \ldots + g_0 \qquad [\because \deg g(x) = n-k]$$

Then the generator matrix

$$G' = \begin{bmatrix} x^{k-1}g(x) \\ \vdots \\ x^2 g(x) \\ xg(x) \\ g(x) \end{bmatrix}$$

$$\begin{array}{c} \qquad\qquad\qquad k\text{th column} \\ \qquad\qquad\qquad \Downarrow \end{array}$$

$$= \underbrace{\left.\begin{bmatrix} g_{n-k} & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & g_{n-k} & g_{n-k-1} & \cdots & \cdots & \cdots & g_0 & 0 & 0 \\ 0 & \cdots & 0 & g_{n-k} & g_{n-k-1} & \cdots & \cdots & g_1 & g_0 & 0 \\ 0 & \cdots & 0 & 0 & g_{n-k} & g_{n-k-1} & \cdots & g_2 & g_1 & g_0 \end{bmatrix}\right\}k}_{n} \qquad (3.26)$$

We know, codeword matrix = message matrix × generator matrix.

$$\Rightarrow C = MG'$$

But the codeword is not systematic when generator matrix of equation (3.26) is used to generate the codeword. Because in this codeword first k bits does not represent message bits and remaining $(n-k)$ bits does not represent parity bits. For this purpose, generator matrix would be following form :

$$G = [I_k \mid P]$$

Where I_k = k by k identity matrix.

P = k by $(n-k)$ parity matrix.

To derive this generator matrix, G, we have to follow these steps :

1. Use $g(x)$ as the kth row.

2. To generate the (k-1)th row, cyclically shift the kth row one column to the left. This corresponds of course to the operation $x.g(x)$. But the kth column from left entry must be zero to have the systematic form. If this entry is nonzero, make it zero by linear combination with kth row ($g(x)$).

3. To generate the (k-2)th row repeat the same process : shift the (k-1)st row entries one column to the left, make linear combination with $g(x)$ if the kth column entry is not zero. Repeat this for all other rows.

So, it is apparent that the matrix, G is obtained from the matrix G' by elementary row operations. As a result the codeword does not change, but it simply reorders the codeword elements. Now we want to generate parity check matrix.

3.3.2 : Parity Check Matrix Generation :

From linear block codes, we know that parity check matrix,

$$H = [P^T \mid I_{n-k}]$$

Where I_{n-k} = (n-k) by (n-k) identity matrix.

P^T = transpose matrix of matrix P.
= (n-k) by k matrix.

From this equation, H can easily be derived from G. But here we want to generate parity check matrix from parity check polynomial.

Let parity check polynomial,

$$h(x) = h_k x^k + h_{k-1}x^{k-1} + \ldots + h_0 \qquad [\because \deg h(x) = k]$$

Hence parity check matrix,

$$H' = \begin{bmatrix} x^{n-1}h(x^{-1}) \\ x^{n-2}h(x^{-1}) \\ x^{n-3}h(x^{-1}) \\ \vdots \\ x^k h(x^{-1}) \end{bmatrix}$$

$$= \underbrace{\begin{bmatrix} h_0 & h_1 & h_2 & \cdots & \overset{\substack{(k+1)\text{th column} \\ \Downarrow}}{h_k} & 0 & 0 & \cdots & 0 \\ 0 & h_0 & h_1 & \cdots & h_{k-1} & h_k & 0 & \cdots & 0 \\ 0 & 0 & h_0 & \cdots & h_{k-2} & h_{k-1} & h_k & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & h_k \end{bmatrix}}_{n} \left.\vphantom{\begin{matrix}a\\a\\a\\a\\a\end{matrix}}\right\}(n-k) \qquad (3.27)$$

Here is also H' is not systematic form. To generate systematic form of parity check matrix, H, we have to follow these steps :

1. Use $x^{n-1}h(x^{-1})$ as the 1st row.

2. To generate 2nd row, cyclically shift the first row one column to the right. This corresponds, of course, to the operation $x^{n-2}h(x^{-1})$. But the $(k+1)$th column from left entry must be zero, to have the systematic form. If the entry is not zero, make a linear combination of the first row to it.

3. To generate 3rd row, repeat the same process : Shift the 2nd row entries one column to the right. Make a linear combination with $h(x)$ if $(k+1)$th column entry is not zero. Repeat this process for all other rows.

Now an example is given to generate G and H from $g(x)$ and $h(x)$ respectively to clarify the above process. Let, we want to generate G and H of an (7, 4) cyclic code in GF(2). We know that

$$x^7\text{-}1 = g(x)h(x) \text{ for } (7, 4) \text{ cyclic code.}$$

Now, factoring $(x^7\text{-}1)$ into irreducible polynomials :

$$x^7\text{-}1 = (I+x)(1+x^2+x^3)(1+x+x^3)$$

Since deg $g(x) = n\text{-}k = 7\text{-}4 = 3$

Let $g(x) = 1+x+x^3$

So, $h(x) = (1+x)(1+x^2+x^3)$

$= 1+x+x^2+x^4$

Here deg $h(x) = 4 = k$ as it would be.

At first, we want to generate G,

$$G = \begin{bmatrix} x[x^2g(x)+g(x)]+g(x) \\ x^2g(x)+g(x) \\ xg(x) \\ g(x) \end{bmatrix}$$

$$= \begin{bmatrix} x^6 & - & - & - & x^2 & - & 1 \\ - & x^5 & - & - & x^2 & x & 1 \\ - & - & x^4 & - & x^2 & x & - \\ - & - & - & x^3 & - & x & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

[$\because$replacing x by 1 and blank by 0]

Here $G = [I_k \mid P]$, where

$$P = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

So, H would be, $H = [P^T \mid I_{n-k}]$

$$= \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \quad (3.28)$$

Now we want to calculate H from $h(x)$.

$$H = \begin{bmatrix} x^6 h(x^{-1}) \\ x^5 h(x^{-1}) \\ x^4 h(x^{-1}) + x^6 h(x^{-1}) \end{bmatrix}$$

$$= \begin{bmatrix} x^6 & x^5 & x^4 & - & x^2 & - & - \\ - & x^5 & x^4 & x^3 & - & x & - \\ x^6 & x^5 & - & x^3 & - & - & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{bmatrix} \quad (3.29)$$

[replacing x by 1 and blank by 0]

Here $H = [P^T \mid I_{n-k}]$ as would be desired, i.e. it is same as equation (3.28).

§ 3.4 : CYCLIC CODES FOR CORRECTING SINGLE ERRORS :

We know that minimum distance ≥ 2×error +1. So, to correct one error, the minimum distance must be at least = 2×1+1 = 3.

The following theorem is helpful to find single error correcting (n, k) cyclic code over GF(q).

Theorem 3.4.6 :

The (n, k) cyclic code over GF(q) with parity-check matrix $H = [\beta^0, \beta^1, ..., \beta^{n-1}]$ where $\beta = \alpha^{q-1}$, α is primitive in GF(q^m) and blocklength, $n = (q^m-1)/(q-1)$, has minimum distance at least 3 iff m and q-1 are relatively prime.

Proof :

Suppose two columns of h are linearly dependent, then $\beta^i = \gamma\beta^j$ where γ is an element of GF(q). Thus β^{i-j} $(=\gamma)$ is an element of GF(q). From equation (3.12), β^{i-j} is a zero of $x^{q-1}-1$.

Every nonzero element of GF(q) and GF(q^m) can be expressed as powers of α, the primitive element of GF(q^m). Again elements of GF(q^m) are derived from m combinations of q elements of GF(q). As α is the primitive element of GF(q^m), $\alpha^{q^m-1} = 1$ and if α' is the primitive element of GF(q) then $(\alpha')^{q-1} = 1$. Thus elements of GF(q) can be expressed as the first (q-1) powers of $\alpha^{(q^m-1)/(q-1)}$.

$$\text{So, } \beta^{i-j} = \left(\alpha^{(q^m-1)/(q-1)}\right)^k = \alpha^{nk} \qquad \text{for some } k. \tag{3.30}$$

$$[\because n = (q^m-1)/(q-1)]$$

In our assumption $i \neq j$, so $\beta^{i-j} \neq \beta^0 = 1$. Therefore, $k <$ q-1, otherwise if $k = q$-1, then

$$\beta^{i-j} = \left(\alpha^{(q^m-1)/(q-1)}\right)^k$$

$$= \left(\alpha^{(q^m-1)/(q-1)}\right)^{(q-1)}$$

$$= \alpha^{q^m-1}$$

$$=1.$$

But $\beta^{i-j} \neq 1$ in our assumption. So $k < (q\text{-}1)$. From (3.30),

$$\beta^{i-j} = \alpha^{nk}$$

$$\Rightarrow \alpha^{(q-1)(i-j)} = \alpha^{nk}$$

$$\Rightarrow (q-1)(i-j) = nk$$

Because $k < (q\text{-}1)$, $n > (i\text{-}j)$. Now

$$(q-1)(i-j) = nk$$

$$\Rightarrow (i-j) = nk/(q-1)$$

So, $(i\text{-}j)$ has a solution iff n and $(q\text{-}1)$ are not relatively prime. That is, H cannot have two linearly dependent columns iff n and $(q\text{-}1)$ are relatively prime.

We have seen in linear block codes, that a code has minimum distance not smaller than w iff every set of$(w\text{-}1)$ columns of H is linearly independent. Thus the given cyclic code has minimum distance at least 3 iff n and $(q\text{-}1)$ are relatively prime.

$$\begin{aligned} \text{Now, } n &= (q^{m}\text{-}1)/(q\text{-}1) \\ &= (q\text{-}1)(q^{m\text{-}1}+q^{m\text{-}2}+\ldots+1)/(q\text{-}1) \\ \therefore\ n &= q^{m\text{-}1}+q^{m\text{-}2}+\ldots+1 \end{aligned} \tag{3.31}$$

Again

$$q^{m\text{-}j}\ \text{-}1 = (q\text{-}1)(q^{(m\text{-}j)\text{-}1}+q^{(m\text{-}j)\text{-}2}+\ldots+1) \quad \text{for integer } j.$$

$$\Rightarrow q^{m\text{-}j}\ \text{-}1 = (q\text{-}1)S_j \qquad [\text{Letting } S_j = q^{(m\text{-}j)\text{-}1}+q^{(m\text{-}j)\text{-}2}+\ldots+1]$$

$$\Rightarrow q^{m\text{-}j} \quad = (q\text{-}1)S_j+1$$

$$\Rightarrow \sum_{j=1}^{m} q^{(m-j)} = \sum_{j=1}^{m}\left[(q-1)S_j+1\right] \qquad [\text{Taking sum over } j \text{ from 1 to } m]$$

$$\Rightarrow q^{(m\text{-}1)}+q^{(m\text{-}2)}+\ldots+1 = (q\text{-}1)\sum_{j=1}^{m} S_j + \ \sum_{j=1}^{m} 1$$

$$\Rightarrow n = (q\text{-}1)\ \sum_{j=1}^{m} S_j + m \quad [\because \text{from equation (3.31)}]$$

$$\Rightarrow n = (q\text{-}1)S + m \qquad \left[\text{Letting } S = \sum_{j=1}^{m} S_j\right] \tag{3.32}$$

Now let m and $(q\text{-}1)$ are not relatively prime but n and $(q\text{-}1)$ are relatively prime and let a is a common factor of m and $(q\text{-}1)$. Therefore, from (3.32),

$$n = a[(q-1)^{'}S + m'\,] \quad [m = am' \quad \text{and} \quad q\text{-}1 = a(q-1)^{'}\,] \tag{3.33}$$

Thus n must be divided by a. Since n and $(q\text{-}1)$ are relatively prime, equation (3.33) is not possible. So, m and $(q\text{-}1)$ are also relatively prime.

Similarly it can be proved that if n and $(q\text{-}1)$ are relatively prime then m and $(q\text{-}1)$ are also relatively prime. Thus n and $(q\text{-}1)$ are relatively prime iff m and $(q\text{-}1)$ are relatively prime.

Now we want to give an example to build a single error correcting code (Hamming code) over GF(2) to clarify the above discussion. Let α be primitive element in GF(2^3) or GF(8). Then $m = 3$, $n = (2^3\text{-}1)/(2\text{-}1) = 7$ and $k = n - m = 7 - 3 = 4$. Here $m = 3$ and $(q\text{-}1) = 1$ are relatively prime. Thus (7, 4) hamming code is a single error correcting cyclic code over GF(2).

§ 3.5 : CYCLIC CODES FOR CORRECTING DOUBLE ERRORS :

For double error correcting cyclic codes, over GF(q), let a generator polynomial having α and α^3 as zeros, where α is primitive element in GF(q^m). We exhibit here a decoding procedure that corrects all single and double error. We have considered double-error-correcting code over GF(2) for simplicity.

let $v(x)$ be received codeword polynomial, $m(x)$ be message polynomial, $g(x)$ be generator polynomial, $e(x)$ be error polynomial. Then

$$v(x) = m(x)g(x) + e(x) \tag{3.34}$$

here $g(x)$ is the smallest degree polynomial in GF(2) with α and α^3 as zeros in GF(2^m). So

$$g(\alpha) = g(\alpha^3) = 0 \tag{3.35}$$

Since we are considering double error-correcting codes,

$e(x) = 0$ or x^i or $x^i + x^{i'}$ for no, one or two errors respectively. The integers i and i' index the location in which errors occur. Let the syndromes, $S_1 = v(\alpha)$ and $S_3 = v(\alpha^3)$.

$$\begin{aligned} \text{Then } S_1 &= v(\alpha) = m(\alpha)g(\alpha)+e(\alpha) \\ &= m(\alpha)\times 0 + e(\alpha) \qquad \text{[from (3.35)]} \\ &= e(\alpha) \\ &= \alpha^i + \alpha^{i'} \end{aligned} \tag{3.36}$$

$$\begin{aligned} \text{and } S_3 &= v(\alpha^3) = m(\alpha^3)g(\alpha^3) + e(\alpha^3) \\ &= v(\alpha^3)\times 0 + e(\alpha^3) \\ &= e(\alpha^3) \\ &= \alpha^{3i} + \alpha^{3i'} \end{aligned} \tag{3.37}$$

In equation (3.36) and (3.37) α^i, $\alpha^{i'}$, α^{3i} and $\alpha^{3i'}$ all elements in GF(2^m). These are called error location numbers. Let $X_1 = \alpha^i$ and $X_2 = \alpha^{i'}$. Here we have not considered burst errors but only two consecutive errors. Also we have not considered cyclic errors, so $i \neq i'$ and $i, i' < n$. Thus X_1 and X_2 are unique. Here $X_1 = X_2 = 0$ for no error, $X_2 = 0$ for one error and $X_1 \neq X_2$ for two errors. From equation (3.36) and (3.37)

$$S_1 = X_1 + X_2$$
$$S_3 = X_1^3 + X_2^3$$

S_1 equals zero iff no error occurs. So, the decoder needs to proceed only if S_1 is not zero. If the above pair of nonlinear equations can be solved uniquely for X_1 and X_2, the two errors can be corrected.

To solve these equations, a new polynomial in GF(2) is defined to have the error-location numbers as zeros as :

$$(x + X_1)(x + X_2) = x^2 + (X_1 + X_2)x + X_1X_2 \tag{3.38}$$

But $\quad S_1{}^3 + S_3 = (X_1 + X_2)^3 + (X_1{}^3+X_2{}^3)$

$$= X_1{}^3 + 3X_1{}^2X_2 + 3X_1X_2{}^2 + X_2{}^3 + X_1{}^3 + X_2{}^3$$

$$= (X_1{}^3 + X_1{}^3) + (X_1{}^2X_2 + X_1{}^2X_2) + X_1{}^2X_2 + (X_1X_2{}^2 + X_1X_2{}^2) + X_1X_2{}^2 + (X_2{}^3 + X_2{}^3)$$

$$= X_1{}^2X_2 + X_1X_2{}^2 \qquad \text{[In GF(2), } a + a = 0\text{]}$$

$$= X_1X_2(X_1 + X_2)$$

$$= X_1X_2S_1 \qquad [S_1 = X_1 + X_2]$$

$$\therefore X_1X_2 = (S_1{}^3 + S_3)/S_1 \tag{3.39}$$

Thus equation (3.38) becomes

$$(x + X_1)(x + X_2) = x^2 + S_1x + (S_1{}^3 + S_3)/S_I \tag{3.40}$$

We know this equation because we know S_1, S_3 directly from equations (3.36) and (3.37). Thus roots of equation (3.40) i.e. X_1 and X_2 are easily found. Therefore the code is a double error correcting code.

§ 3.6 : CYCLIC CODES FOR CORRECTING BURST ERRORS :

(n, k) cyclic codes over GF(q) are designed to correct any random pattern of burst errors. A burst of length t is defined to be a set of t GF(q) elements , the first and the last of which are nonzero elements of GF(q). That is, burst error of length t has first and last (t-th) positions are in error with the (t-2) positions between either in errors or correct.

A t-error-correcting (n, k) cyclic code has the following bounds :

Bound 3.6.1 :

The t-error correcting cyclic code cannot have a burst of length $2t$ or less as a codeword.

Proof :

From linear block code, minimum distance $\geq 2t+1$. Since codelength must be at least equal to its minimum distance, the code cannot have a burst length $2t$ or less as a codeword.

Bound 3.6.2:

The cyclic code in GF(q) must have at least $2t$ parity symbols.(Rieger bound).

Proof :

Suppose the code corrects all burst of length t or less.Then there is no codeword that is a burst of length $2t$(from bound 3.6.1). Now if two vectors are in the same coset, then their difference is a codeword. Choose any two vectors that are zero except in their first $2t$ components. If these are in the same coset of the standard array, then their difference is a codeword; it is burst of length $2t$, and we have seen there are no such codewords. Therefore two such vectors must be in different cosets, and the number of cosets is at least as large as the number of such vectors. There are q^{2t} vectors that are zero except in their first $2t$ components; hence there are at least q^{2t} cosets. But we know there are $q^{\text{parity symbols}}$ cosets in the standard array(from linear block code). Hence at least $2t$ parity symbols.

Cyclic codes can also correct cyclic burst errors. A *cyclic burst of length t* is a vector whose nonzero components are among t (cyclically) successive components, the first and the last of which are nonzero. In this case, the following bound can be obtained :

Bound 3.6.3:

The parity symbols of the cyclic codes in GF(q) must be at least $(t\text{-}1+\log_q n)$.

Proof :

The t error correcting (n, k) cyclic code has to correct q^{t-1} (t-1(starting position))different bursts starting on a given position. These burst can start at any position of the blocklength n. Thus, the code has to correct nq^{t-1} burst errors.

But there are $q^{n\text{-k}}$ cosets of the cyclic code(from linear block code). So, the code can correct at most $q^{n\text{-k}}$ errors. Then

$$q^{n\text{-k}} \geq nq^{t\text{-}1}$$
$$\Rightarrow (n - k) \geq \log_q n + t - 1$$

Since $(n - k)$ is the number of parity symbols of the (n, k) cyclic code, this bound is proved.

From bound (3.6.3), we can easily determine the capability of the burst correcting cyclic code. for example, let a (7, 3) cyclic code over GF(2). Then $n = 7$, $k = 3$ and $q = 2$. So,

$$7 - 3 \geq \log_2 7 + t - 1$$
$$\Rightarrow 4 \quad \geq 2.8 + t - 1$$
$$\Rightarrow 4 \quad \geq 1.8 + t$$
$$\Rightarrow t \quad \leq 2.2$$

Thus $t = 2$ i.e. the (7, 3) cyclic code can correct burst of length 2 or less.

Now let we have already a t error correcting (n, k) cyclic code of generator polynomial $g(x)$. Then we can get jt burst correcting (jn, jk) cyclic code of generator polynomial $g(x^j)$ by the technique of interleaving.

Proof :

Since $g(x)$ divides $(x^n\text{-}1)$ [from property (3.2.2.3)], then $g(x^j)$ divides $(x^{nj}\text{-}1)$ and thereby generates a codeword of length jn. Again deg $g(x^j)$ is j times larger than deg $g(x)$, so the code has jk information symbols.[deg $g(x^j) = j \times$ deg $g(x)$. $\Rightarrow jn - k' = j(n\text{-}k)$. $\Rightarrow k' = jk$] Thus $g(x^j)$ generates (jn, jk) cyclic codes.

Now consider a j by n array in which each row is an n element cyclic codeword generated by $g(x)$. The array can be shifted so that the ith element in the kth row replace the ith element in the $(k+1)$th row if $k < j$ and in the first row if $k = j$. Thus the whole array can be considered as a (jn, jk) cyclic codeword generated by $g(x^j)$. Since each row is a (n, k) cyclic code with the t burst correcting ability, the whole array is a jt burst correcting (jn, jk) cyclic code with the generator polynomial $g(x^j)$.

Thus we can easily construct longer burst correcting cyclic code from shorter burst correcting cyclic code by interleaving.

Now let us to clarify the interleaving technique. To form a (jn, jk) codeword from (n, k) codeword, we have to take j (n, k) codewords. Let these are

$$C_1(x) = m_1(x)g(x)$$
$$C_2(x) = m_2(x)g(x)$$
$$\cdot$$
$$\cdot$$

$$C_j(x) = m_j(x)g(x)$$

To form interleaved codeword, the symbols of each of these codewords are spread out with $(j\text{-}1)$ zeros inserted after every symbol. These codewords are added to form interleaved codeword, i.e.

$$\begin{aligned} C(x) &= C_1(x^j) + xC_2(x^j) + ... + x^{j-1}C_j(x^j) \\ &= m_1(x^j)g(x^j) + xm_2(x^j)g(x^j) + ... + x^{j-1}m_j(x^j)g(x^j) \\ &= [m_1(x^j) + xm_2(x^j) + ... + x^{j-1}m_j(x^j)]g(x^j) \end{aligned} \tag{3.41}$$

This bracketed term is the interleaved message and can be replaced by $m(x)$. So from equation (3.41)

$$C(x) = m(x)g(x^j)$$

Burst correcting cyclic codes can be constructed analytically. The fire codes are a class of such codes. Fire codes are defined as follows :

3.6.1 : Fire Code :

A *fire code* is a cyclic burst correcting code over GF(q) with generator polynomial $g(x) = (x^{2t-1}-1)p(x)$, where $p(x)$ is a prime polynomial over GF(q) whose degree m is not smaller than t and $p(x)$ does not divide $x^{2t-1}-1$.The blocklength of the Fire code is the smallest integer n such that $g(x)$ divides x^n-1. Here $n = e(2t\text{-}1)$, where e is the smallest integer such that $p(x)$ divides x^e-1.

§ 3.7 : SOME IMPORTANT CYCLIC CODES :

3.7.1 : The Binary Golay Code :

The binary Golay code is a (23, 12) binary cyclic code over GF(2). This Golay code is generated either by the polynomial :

$$g(x) = x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1$$

or by the polynomial :

$$\tilde{g}(x) = x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1$$

These above two polynomials are reciprocal to each other. That is $\tilde{g}(x) = x^{11}g(x^{-1})$ and $g(x) = x^{11}\,\tilde{g}(x^{-1})$ From property (3.2.2.3) of generator polynomial, we can write

$$x^{23}\text{-}1 = a(x)g(x)$$

and $x^{23}\text{-}1 = b(x)\,\tilde{g}(x)$

for some $a(x)$ and $b(x)$. If can be proved that $a(x) = (x\text{-}1)\,\tilde{g}(x)$ and $b(x) = (x\text{-}1)g(x)$ i.e.

$$x^{23}\text{-}1 = (x\text{-}1)g(x)\,\tilde{g}(x) \text{ over GF(2)} \tag{3.42}$$

Proof :

R.H.S.

$$\begin{aligned}(x\text{-}1)g(x)\,\tilde{g}(x) &= (x\text{-}1)(x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1)\\
&\quad (x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1)\\
&= (x\text{-}1)(x^{22} + x^{20} + x^{18} + x^{17} + x^{16} + x^{12} + x^{11} +\\
&\quad x^{21} + x^{19} + x^{17} + x^{16} + x^{15} + x^{11} + x^{10} +\\
&\quad x^{17} + x^{15} + x^{13} + x^{12} + x^{11} + x^7 + x^6 +\\
&\quad x^{16} + x^{14} + x^{12} + x^{11} + x^{10} + x^6 + x^5 +\\
&\quad x^{15} + x^{13} + x^{11} + x^{10} + x^9 + x^5 + x^4 +\\
&\quad x^{13} + x^{11} + x^9 + x^8 + x^7 + x^3 + x^2 +\\
&\quad x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1).\\
&= (x\text{-}1)(x^{22} + x^{21} + x^{20} + x^{19} + x^{18} + x^{17} +\\
&\quad x^{16} + x^{15} + x^{14} + x^{13} + x^{12} + x^{11} +\\
&\quad x^{10} + x^9 + x^8 + x^7 + x^6 + x^5 +\\
&\quad x^4 + x^3 + x^2 + x + 1)\\
&= (x^{23} + x^{22} + x^{21} + x^{20} + x^{19} + x^{18} + x^{17} +\\
&\quad x^{16} + x^{15} + x^{14} + x^{13} + x^{12} + x^{11} +\\
&\quad x^{10} + x^9 + x^8 + x^7 + x^6 + x^5 +\\
&\quad x^4 + x^3 + x^2 + x)\text{-}\\
&\quad (x^{22} + x^{21} + x^{20} + x^{19} + x^{18} + x^{17} +\\
&\quad x^{16} + x^{15} + x^{14} + x^{13} + x^{12} + x^{11} +\end{aligned}$$

$$
\begin{aligned}
&x^{10} + x^{9} + x^{8} + x^{7} + x^{6} + x^{5} + \\
&x^{4} + x^{3} + x^{2} + x + 1) \\
&= x^{23} - 1 = \text{L. H. S.}
\end{aligned}
$$

This binary (23, 12) Golay code has following property :

Property 3.7.1.1 :

Minimum weight of the binary Golay code is at least 7.

Proof :

To prove this, we have to find the zeros of $g(x)$ and $\tilde{g}(x)$ in a suitable extension field of GF(2). We have already prove that if GF(2^m) is an extension field of GF(2) and if α be primitive in GF(2^m) and if 2^m-1 = nb and if $\beta = \alpha^b$, then all the zeros of the generator polynomial are restricted to powers of β.

Let the GF(2^{11}) be the extension field of GF(2). Let α be primitive in GF(2^{11}) and 2^{11}-1 = 2047 = 23 × 89, where n = 23 and b = 89. Thus if $\beta = \alpha^{89}$, then all the zeros of the generator polynomial are restricted to powers of β and β has order 23, because $\beta^{23} = \alpha^{89\times 23} = \alpha^{2^{11}-1} = 1$. Similarly β^{-1} has also order 23. Let $g(x)$ and $\tilde{g}(x)$ have zeros of power β and β^{-1} respectively. That is, $g(x)$ and $\tilde{g}(x)$ are minimal polynomial of β and β^{-1} respectively. So, from equation (3.22) we can write,

$$g(x) = (x- \beta)(x- \beta^{2})\ldots\left(x - \beta^{2^{r-1}}\right)$$

and

$$\tilde{g}(x) = (x- \beta^{-1})(x- \beta^{-2})\ldots\left(x - \beta^{-2^{r-1}}\right)$$

Since β and β^{1} has degree 23, the set of conjugates are

$$B = \left\{\beta^{2^m \bmod 23}\right\}$$

and

$$\tilde{B} = \left\{\beta^{-2^m \bmod 23}\right\} \text{ for some } m.$$

That is, $B = \{\beta, \beta^{2}, \beta^{4}, \beta^{8}, \beta^{16}, \beta^{9}, \beta^{18}, \beta^{13}, \beta^{3}, \beta^{6}, \beta^{12}\}$

and $\tilde{B} = \{\beta^{-1}, \beta^{-2}, \beta^{-4}, \beta^{-8}, \beta^{-16}, \beta^{-9}, \beta^{-18}, \beta^{-13}, \beta^{-3}, \beta^{-6}, \beta^{-12}\}$

Thus B and $\tilde{B}$ are the sets of zeros of $g(x)$ and $\tilde{g}(x)$. Since B contains consecutive four roots (), from property of cyclic codes, its minimum distance is greater than 4.

Now let $c(x) = a(x)\ g(x)$ be a binary Golay code, whose weight w is even. So, $c(x) = \sum_{i=0}^{22} c_i x^i$ where even number of c_i are equal to 1. Then $c(1) = 1+1+\ldots$ even number 1s = 0. Thus 1 is a zero of $c(x)$ i.e. $(x-1)$ divides $c(x)$. We have seen that $g(x)$ has weight 7, an odd number, thus $g(1) \neq 0$, i.e. $(x-1)$ does not divide $g(x)$. So, $c(x) = b(x)(x-1)\ g(x)$ for some $b(x)$, that is not divided by $(x-1)$. Now the reciprocal codeword polynomial is,

$$\begin{aligned}\tilde{c}(x) &= \tilde{a}(x)\ \tilde{g}(x)\\ &= \sum_{i=0}^{22} c_{22-i}x^i\\ &= x^{22}\sum_{i=0}^{22} c_i x^{-i}\end{aligned}$$

Therefore, $$\begin{aligned}c(x)\ \tilde{c}(x) &= b(x)(x-1)g(x)\ \tilde{a}(x)\ \tilde{g}(x)\\ &= b(x)\ \tilde{a}(x)\,(x-1)g(x)\ \tilde{g}(x)\\ &= b(x)\ \tilde{a}(x)\,(x^{23}-1) \qquad \text{[from equation (3.46)]}\end{aligned} \tag{3.43}$$

Again $$\begin{aligned}c(x)\ \tilde{c}(x) &= \left(\sum_{i=0}^{22} c_i x^i\right)\left(\sum_{j=0}^{22} c_{22-j}x^j\right)\\ &= \sum_{j=0}^{22}\sum_{i=0}^{22} c_i\, c_{22-j}\, x^{i+j}\\ &= \sum_{k=0}^{44}\sum_{i=0}^{22} c_i\, c_{22-(k-i)}\, x^k \qquad \text{[Putting } k = i+j\text{]}\\ &= \sum_{j=0}^{44}\sum_{i=0}^{22} c_i c_{22+i-j}x^j \qquad \text{[Putting } j = k\text{]}\\ &= \sum_{j=0}^{21}\sum_{i=0}^{22} c_i c_{22+i-j}x^j + \sum_{j=22}^{22}\sum_{i=0}^{22} c_i c_{22+i-j}x^j + \sum_{j=23}^{44}\sum_{i=0}^{22} c_i c_{22+i-j}x^j\\ &= \sum_{j=0}^{21}\sum_{i=0}^{22} c_i c_{22+i-j}x^j + \sum_{i=0}^{22} c_i c_{22+i-22}x^{22} + \sum_{j=23}^{44}\sum_{i=0}^{22} c_i c_{22+i-j}x^j\\ &= \sum_{j=0}^{21}\sum_{i=0}^{22} c_i c_{22+i-j}x^j + \sum_{i=0}^{22} c_i^2 x^{22} + \sum_{j=23}^{44}\sum_{i=0}^{22} c_i c_{22+i-j}x^j\end{aligned} \tag{3.44}$$

But over GF(2) $c_i^2 = c_i$.. Because if $c_i = 0$ then $c_i^2 = 0 = c_i$. and if $c_i = 1$ then $c_i^2 = 1 = c_i$.. So, middle term of equation (3.44)

$$\begin{aligned}\sum_{i=0}^{22} c_i^2 x^{22} &= x^{22}\sum_{i=0}^{22} c_i\\ &= 0 \qquad [\because \text{Even number of } c_i\text{.'s are equal to one}]\end{aligned}$$

For c_i., i is between 0 and 22. In third term of equation (3.44), if we take the limit of i from $-\alpha$ to α for simplicity, where c_i. = 0 for $i < 0$ and $i > 22$. Thus equation (3.44) can be written as

$$c(x)\ \widetilde{c}(x) = \sum_{j=0}^{21}\sum_{i=0}^{22} c_i c_{22+i-j} x^j + \sum_{j=23}^{44}\ \sum_{i=-\alpha}^{\alpha} c_i c_{22+i-j} x^j$$

Substituting $i' = i+22-j$ in second term and $j' = j$ in both terms.

$$c(x)\ \widetilde{c}(x) = \sum_{j'=0}^{21}\sum_{i=0}^{22} c_i c_{22+i-j'} x^{j'} + \sum_{j'=23}^{44}\ \sum_{i'=-\alpha}^{\alpha} c_{i'+j'-22} c_{i'} x^{j'}$$

Substituting $j = j' + 1$ in first term and $i = i'$ and $j = j' - 22$ in second term

$$c(x)\ \widetilde{c}(x) = \sum_{j=1}^{22}\sum_{i=0}^{22} c_i c_{23+i-j} x^{j-1} + \sum_{j=1}^{22}\ \sum_{i=-\alpha}^{\alpha} c_{i+j} c_i x^{j+22}$$

$$= \sum_{j=1}^{22}\sum_{i=0}^{22}\left[c_i c_{23+i-j} + c_i c_{i+j}\right] x^{j-1} + \left[\sum_{j=1}^{22}\ \sum_{i=-\alpha}^{\alpha} c_i c_{i+j} x^{j-1}\right]\left(x^{23} - 1\right) \tag{3.45}$$

From equation (3.43) $c(x)\ \widetilde{c}(x)$ is a multiple of $(x^{23}-1)$. So,

$$\sum_{i=0}^{22}\left[c_i c_{23+i-j} + c_i c_{i+j}\right] = 0 \qquad \text{for } j = 1, 2, \ldots, 22$$

$$\Rightarrow \sum_{i=0}^{22} c_i\left[c_{23+i-j} + c_{i+j}\right] = 0 \qquad \text{for } j = 1, 2, \ldots, 22$$

This equation holds under modulo-2 addition. Hence evaluating the left hand side under normal integer addition, must give an even number. So,

$$\sum_{i=0}^{22} c_i\left[c_{23+i-j} + c_{i+j}\right] = 2a_j \qquad \text{for } j = 1, 2, \ldots, 22 \text{ and for some } a_j. \tag{3.46}$$

Replacing j by $23-j$,

$$\sum_{i=0}^{22} c_i\left[c_{23+i-j} + c_{i+j}\right] = 2a_{23\text{-}j} \tag{3.47}$$

Equating equation (3.46) and equation (3.47), $a_j = a_{23\text{-}j}$. Therefore

$$\sum_{j=1}^{22}\sum_{i=0}^{22} c_i\left(c_{23+i-j} + c_{i+j}\right) = 2\sum_{j=1}^{22} a_j$$

$$= 2\sum_{j=1}^{11} a_j + 2\sum_{j=12}^{22} a_j$$

$$= 2\sum_{j=1}^{11} a_j + 2\sum_{j=12}^{22} a_{23-j} \quad [\because a_j = a_{23\text{-}j}]$$

$$= 2\sum_{j=1}^{11} a_j + 2\sum_{j=1}^{11} a_j$$

$$= 4\sum_{j=1}^{11} a_j$$

$$= 4a \quad \text{for some } a$$

Now let $j' = 23\text{-}j$ in the first term and $i = i' \text{-} j$ in the second term. Then

$$\sum_{j'=1}^{22}\sum_{i=0}^{22} c_i c_{i+j'} + \sum_{j=1}^{22}\sum_{i'=0}^{22} c_{i'} c_{i'-j} = 4a$$

$$\Rightarrow \sum_{j=1}^{22}\sum_{i=0}^{22} c_i\left[c_{i+j} + c_{i-j}\right] = 4a$$

$$\Rightarrow \sum_{i=0}^{22}\sum_{j=1}^{22} c_i\left[c_{i+j} + c_{i-j}\right] = 4a$$

$$\Rightarrow \sum_{i=0}^{22}\sum_{j\neq i} c_i c_j = 4a \qquad \text{for } j = 1, 2, \ldots, 44$$

$$\Rightarrow \left(\sum_{i=0}^{22} c_i\right)\left(\sum_{j\neq i} c_j\right) = 4a \tag{3.48}$$

Let binary nonzero component, c_i is nonzero in w places. So weight of the code, $c(x)$, w. Then in normal integer addition $\sum_{i=0}^{22} c_i = w$ and $\sum_{j\neq i} c_j = 1$ less than the number of nonzero component, $w = (w\text{-}1)$. Therefore from equation (3.48), $w(w\text{-}1) = 4a$. So, w is a multiple of 4. Thus every Golay codeword of even weight must be divisible by 4. Therefore, weight of the nonzero Golay codeword cannot be 1, 2, 3, 4, 6, 10, 14, 18 or 22.

There exists a Golay codeword with all ones, because from equation (3.42),

$$(x\text{-}1)g(x)\tilde{g}(x) = x^{23}\text{-}1$$

$$= (x\text{-}1)\sum_{i=0}^{22} x^i$$

$$\Rightarrow g(x)\tilde{g}(x) = \sum_{i=0}^{22} x^i$$

Thus $g(x)\tilde{g}(x)$ is a codeword with all ones. It's weight is 23. If all ones codeword is added to another codeword of weight w then another codeword results with weight $(23\text{-}w)$. Therefore the weight of the Golay codeword cannot be (23-2) = 21, (23-3) = 20, (23-4) = 19, (23-6) = 17, (23-10) = 13, (23-14) = 9 or (23-18) = 5.

Thus the weight of the nonzero Golay codeword may be 7, 8, 11, 12, 15 or 16. Therefore, minimum weight of the nonzero Golay code is 7.

From equation $d \geq 2t + 1$, where d = minimum distance of a code and t = number of errors that can be corrected, the Golay code corrects 3($7 \geq 2t{+}1 \Rightarrow t \leq 3$) or less error. So, the (23, 12) Golay code is a 3 error correcting code.

Proof :

A perfect code satisfies the Hamming bound

$q^{n-k} \geq 1+\binom{n}{1}(q\text{-}1)+\binom{n}{2}(q\text{-}1)^2+...+\binom{n}{t}(q\text{-}1)^t$ with equality. Now for 3 error correcting binary Golay code, $q = 2$, $n = 23$, $k = 12$, $t = 3$. Then $2^{23\text{-}12} = 2^{11} = 2048$ and

$$1+\binom{23}{1}+\binom{23}{2}+\binom{23}{3} = 2048$$

Thus the Golay code satisfies the Hamming bound with equality. Therefore the Golay code is a perfect code.

3.7.2 : Bose- Chaudhuri-Hocquenghem (BCH) Codes

BCH codes are another important group of codes. They belong to the family of cyclic codes with a wide variety of parameters. The most common BCH codes are characterized as follows. Specifically, for any positive integers m (equal to or greater than 3) and t [less than $(2^m\text{-}1)/2$] there exists a binary BCH code with the following parameters:

Block length:	$n= 2^m\text{-}1$
Number of message bits:	$k\geq n\text{-}mt$
Minimum distance :	$d_{min}\geq 2t+1$

Each BCH code is a *t error correcting code* in that it can detect and correct up to t random errors per codeword.

BCH codes offer flexibility in the choice of code parameters, namely, block length and code rate. Furthermore, at block lengths of a few hundred or less, the BCH codes are among the best known codes of the same block length and code rate. The following table shows some BCH code parameters.

n	*k*	*t*	*Generator Polynomial*
7	4	1	1 011
15	11	1	10 011
15	7	2	111 010 001
31	26	1	100 101

Therefore, a (15,7) BCH code will have generator polynomial $g(x) = D^8+ D^7+ D^6+ D^4+1$.

CHAPTER FOUR : CONVOLUTIONAL CODES

Proof :

A perfect code satisfies the Hamming bound

$q^{n-k} \geq 1+\binom{n}{1}(q\text{-}1)+\binom{n}{2}(q\text{-}1)^2+...+\binom{n}{t}(q\text{-}1)^t$ with equality. Now for 3 error correcting binary Golay code, $q = 2$, $n = 23$, $k = 12$, $t = 3$. Then $2^{23\text{-}12} = 2^{11} = 2048$ and

$$1+\binom{23}{1}+\binom{23}{2}+\binom{23}{3}=2048$$

Thus the Golay code satisfies the Hamming bound with equality. Therefore the Golay code is a perfect code.

3.7.2 : Bose- Chaudhuri-Hocquenghem (BCH) Codes

BCH codes are another important group of codes. They belong to the family of cyclic codes with a wide variety of parameters. The most common BCH codes are characterized as follows. Specifically, for any positive integers m (equal to or greater than 3) and t [less than $(2^m\text{-}1)/2$] there exists a binary BCH code with the following parameters:

Block length:	$n= 2^m\text{-}1$
Number of message bits:	$k \geq n\text{-}mt$
Minimum distance :	$d_{min} \geq 2t+1$

Each BCH code is a t *error correcting code* in that it can detect and correct up to t random errors per codeword.

BCH codes offer flexibility in the choice of code parameters, namely, block length and code rate. Furthermore, at block lengths of a few hundred or less, the BCH codes are among the best known codes of the same block length and code rate. The following table shows some BCH code parameters.

n	*k*	*t*	*Generator Polynomial*
7	4	1	1 011
15	11	1	10 011
15	7	2	111 010 001
31	26	1	100 101

Therefore, a (15,7) BCH code will have generator polynomial $g(x) = D^8+ D^7+ D^6+ D^4+1$.

§ 4.1 : INTRODUCTION :

The convolutional coder consists of a k - stage shift register . Input data bits are shifted along the register one bit at a time . The *constraint length of a convolutional coder* , expressed in terms of message bits , is defined as the number of shifts over which a single message bit can influence the encoder output . In an encoder with M-stage shift register , the memory of the encoder equals M message bits , and $k = M$+I shifts are required for a message bit to enter the shift register and finally come out , so over k shifts the output is influenced by that message bit . Hence the constraint length of the encoder is k . Modulo - 2 sums of the contents of the shift register stages are shifted out at a rate v times as fast . There are thus v bit out for every bit in . (In a symbol encoder , v symbols are shifted out for every symbol in) . An R bit/s input data stream thus gives rise to an Rv bit/s coded output stream . A coder outputting v bits for every bit in is called a rate $1/v$ coder .

An example of a k =3 ,v =2 convolutional coder appears in the figure given below .

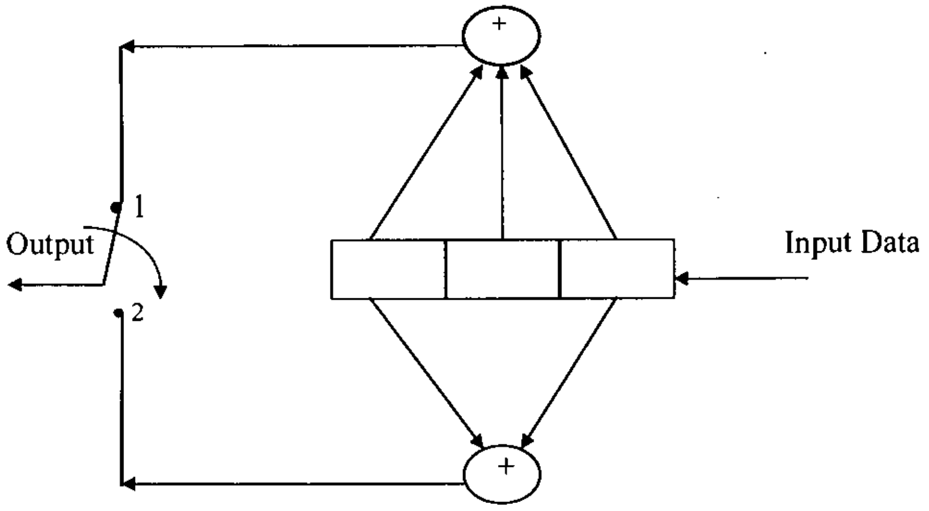


Figure 4.1 : Figure of an k =3 ,v =2 convolutional encoder

Calling the data bits in each of the three stages d_1 , d_2 , d_3 with the output coded bits labeled c_1 and c_2 corresponds to the two output bits , we have

$$c_1 = d_1 \oplus d_2 \oplus d_3$$
$$c_2 = d_1 \qquad \oplus d_3$$

The coding is carried out continuously , on all data bits as they are shifted through the register . An equivalent encoder (produces the same output sequence when the input sequence is the same) of the above encoder with smaller number of states is given in figure 4.2 :

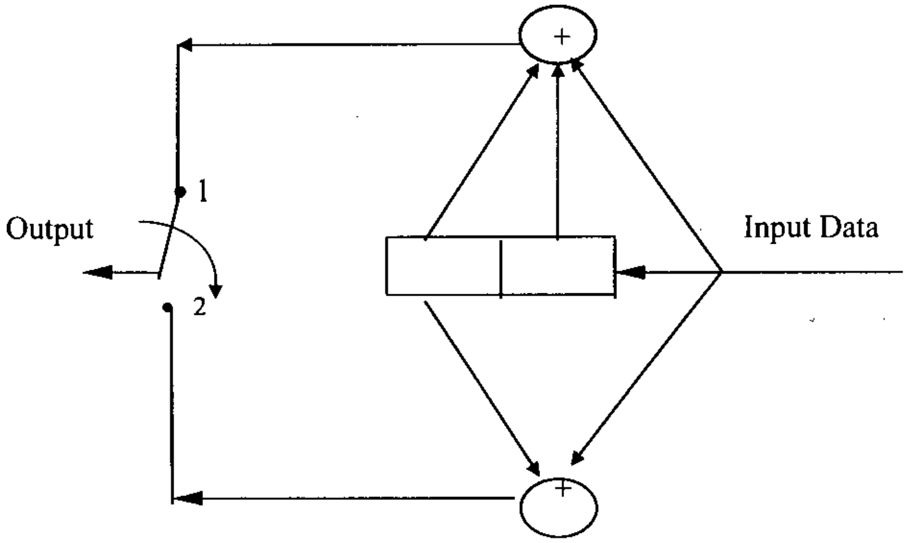


Figure 4 . 2 : An k =3 ,v =2 convolutional coder in minimized form

Because the above encoder is equivalent to a four state machine , the trellis diagram has only four horizontal line sections and the tree diagram has just four distinguishable nodes from the third level on (detail explanation is given later) .

More generally , a constraint-length k , rate -1/v coder would have its output (coded) bits appearing in the form

$$c_1 = h_{11} d_1 \oplus h_{12} d_2 \oplus \ldots \oplus h_{1k} d_k$$
$$c_2 = h_{21} d_1 \oplus h_{22} d_2 \oplus \ldots \oplus h_{2k} d_k$$

$$c_v = h_{v1} d_1 \oplus h_{v2} d_2 \oplus \ldots \oplus h_{vk} d_k$$

The h_{ij} 's are 1 or 0 , depending on whether a connection is made or not . The convolution of F *G of the functions $F(t)$ and $G(t)$ is defined as the function $F(t)$ * $G(t) = \int_0^t F(\tau)G(t-\tau)d\tau$, also convolution in time domain is equivalent to multiplication in frequency domain. The v encoded bits (or symbols in a more general encoder) are outputted sequentially after each input shift. This form represents a modulo 2 convolution of the input data bits with the h_{ij} 's , so the name convolutional encoder or convolutional coder , and the code is called *convolution code.*

If $g^{j}(D)$ $\Rightarrow$ which stages are connected to adder j then for convolutional coder of 4.1(a) generator polynomial $g^1(D) = 1+D+D^2$, $g^2(D) = 1+D^2$. For message sequence (10011) , we have polynomial representation $m(D) = 1+D^3 +D^4$. D means delay due to memory stage .

With Fourier transformation , convolution in time domain is equivalent to multiplication in D domain . The output polynomial of path 1 is given by

$$\begin{aligned} c^1(D) &= \int g(\tau)m(t-\tau)d\tau \quad \text{(convolution in time domain)} \\ &= g^1(D).m(D)\,, \quad \text{(multiplication in } D \text{ domain)} \\ &= (1+D+D^2)(\ 1+D^3+D^4) \\ &= 1+D+D^2+D^3+D^6 \end{aligned}$$

From this , the output sequence for path 1 is immediately deduced as (1111001).

Similarly the output polynomial of path 2 is

$$\begin{aligned} c^2(D) &= g^2(D).m(D)\,, \\ &= (1+D^2)(\ 1+D^3+D^4) \\ &= 1+D^2+D^3+D^4+D^5+D^6\,. \end{aligned}$$

The output sequence of path 2 is therefore (1011111) , finally multiplexing the two output sequences of path 1 and 2 , we get the encoded sequence **c** = (11,10,11,11,01,01,11).

To produce the same encoded sequence for same code using generator matrix representation we have to collect connection vector for each stage to all adder.

$\mathbf{g}_1$ = vectors representing adders connected to stage 0 (current information bit).
$\mathbf{g}_2$ = vectors representing adders connected to stage 1.
And so on

$\therefore\ \mathbf{g}_1 = [\ 1 \quad 1\ ]$
= [coefficient of D^0 for adder 1 coefficient of D^0 for adder 2]

$\mathbf{g}_2 = [\ 1 \quad 0\ ]$
= [coefficient of D for adder 1 coefficient of D for adder 2]

$\mathbf{g}_3 = [\ 1 \quad 1\ ]$
= [coefficient of D^2 for adder 1 coefficient of D^2 for adder 2]

Considering shifting over time the second row of the generator matrix would be the shifted form of the first row of the generator matrix by one time unit , the third row would be the shifted form of the first row by two time units .

$$\text{generator matrix } \mathbf{G}= \left. \begin{matrix} \mathbf{g}_1 & \mathbf{g}_2 & \mathbf{g}_3 & & & & \\ & \mathbf{g}_1 & \mathbf{g}_2 & \mathbf{g}_3 & & & \\ & & \mathbf{g}_1 & \mathbf{g}_2 & \mathbf{g}_3 & & \\ & & & \mathbf{g}_1 & \mathbf{g}_2 & \mathbf{g}_3 & \\ & & & & \mathbf{g}_1 & \mathbf{g}_2 & \mathbf{g}_3 \end{matrix} \right]$$

The unspecified elements are all zero vector of size 1×2

$$= \left. \begin{matrix} 11 & 10 & 11 & 00 & 00 & 00 & 00 \\ 00 & 11 & 10 & 11 & 00 & 00 & 00 \\ 00 & 00 & 11 & 10 & 11 & 00 & 00 \\ 00 & 00 & 00 & 11 & 10 & 11 & 00 \\ 00 & 00 & 00 & 00 & 11 & 10 & 11 \end{matrix} \right]$$

$\therefore\ m(D).\mathbf{G}=(11,10,11,11,01,01,11)$

§ 4.2 : CONCEPTS ABOUT CONVOLUTIONAL CODE :

4.2.1 : Truncation :

Though the main information sequence may be arbitrarily long, actually , even during convolutional coding very long information sequence is segmented into blocks. This process is called truncation. And each block size is called the dividing length (L).

4.2.2 : Tail Bits :

After getting the last information bit of a message block and encoding in the presence of it, we get v channel or encoded bits. If we stop at this point only v output bits (bits on one code branch on the tree representation . Tree is discussed later) are affected . But each of all other information bits of a information block or message block affects kv output bits. So, to treat the last bit of an information vector like other bits of the information vector, (constraint length -1) that is k-1 additional known bits (usually 0's) should be appended to the information sequence and then to be encoded . This bits are called tail bits.

4.2.3 :Why All Zero Path Is Considered As Transmitted Or Reference codeword During Any Calculation :

If all paths are equally likely, any path may be chosen as the correct one and the probability of deviating from it is the probability of error. For simplicity and without loss of generality , let this path is the all zero path .

§ 4.3 : PARITY CHECK CONVOLUTIONAL ENCODER :

Inside the parity check convolutional encoder there is

1. one x register having number of stages equal to constraint length k (when is the encoder representation is not minimized , otherwise in the equivalent minimized form there would be k-1 stages in the shift register).

2. some modulo 2 adders . There is no register which is fed by the output of the adders inside the encoder like linear block encoder.

For communicating an L bit message vector $\boldsymbol{x} = (x_1 \ , x_2 \ , ..., x_L \ )$ in which L may be greater than the constraint length k . First , the content of all k stages of the shift register are set equal to zero . Next , the first bit x_1 is shifted into the stage 1 of the shift register . The v modulo 2 adders are sampled one after another when each bit of **x** (input vector or information vector or message vector) is inserted into the shift register . So, adders are sampled after each bit loading , not only after with all the k bits of a message available in the shift register . At each sampling interval (the time span during which the adders are sampled) when the v th or last adder output has been sampled and transmitted , the second bit (x_2) of the message is shifted into stage 1 of the shift register, which causes x_1 to shift into stage 2 . Each of the v modulo 2 adder outputs are again sampled and transmitted . This procedure continues until the last bit x_L of **x** has been shifted into stage 1 of the x register . After this , with each adder output being sampled and transmitted , k 0's (tail bits) are fed in turn (in each shift) into the x register , thereby returning the shift register to its initial condition . During each shift the symbol (in our case bit) forced out of the k-th stage of the shift register is discarded .

It will suffice to insert (k-1) 0's (tail bits) instead of k 0's into the x register after the last bit of a message block, because the L th bit (last) bit will shifted out of the x register when the first digit of a new message is shifted in .But if there is no other message block , in that case k 0's should be inserted .

Let , the information block is $\mathbf{x} = (x_1 \ , x_2 \ , ... \ , x_L \ )$ and for this whole information block total $(L+k)v$ bits output is denoted as **y**

Since the x register is initially set to zero , the first v bits of **y** , obtained by shifting the first component of **x** into stage 1 of the shift register and sampling v adders , depend only on x_1 . Similarly the second v bits depends only upon x_1 and x_2 . In general , the v bits of output **y** obtained immediately after shifting component x_h into the x register depends only on x_h and (h-1) components of x preceding x_h ,when $h < k$. But depends only on x_h and (k-1) components of **x** preceding x_h , when $h \geq k$. If two information vectors agree in their first (h-1) coordinates , the corresponding output vectors agree in their first $(h\text{-}1)v$ coordinates .

4.3.1 : Properties Of Convolutional Code :

Property 4.3.1.1 : Linearity :

A convolutional code is a linear code .That is if , c_i and c_j are two code words , then $c_i + c_j$ is also a codeword .

Let when the L bit input vector **x** has $x_h = 1$ and all other components each equal to zero , the output vector **y** is f_h . Let $\mathbf{x} = (1, 0,0,0, ..., 0)$ .And $\mathbf{g}_J$ denotes which of the modulo 2 adders are connected to the J th stage of the shift register .

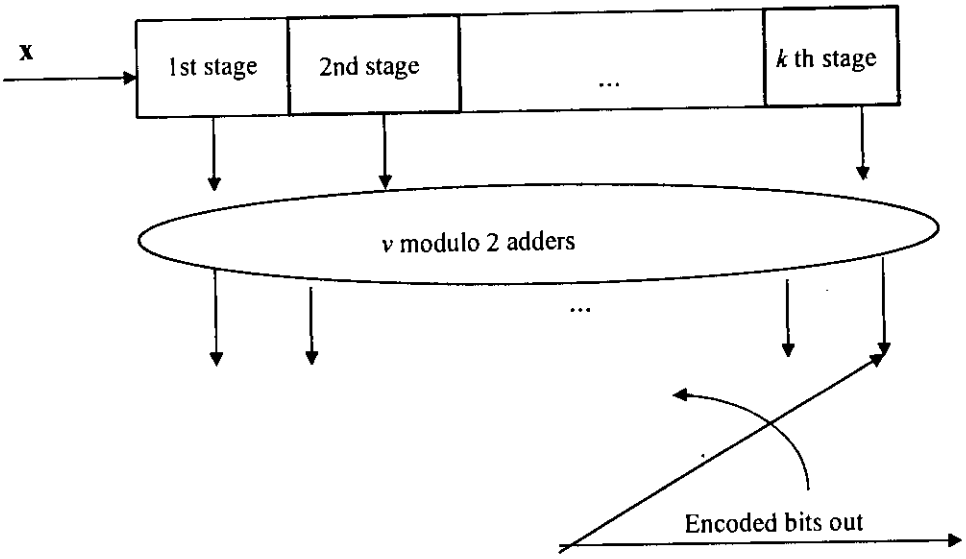


Figure 4.3 : A general form of a Convolutional Encoder

Example:

For the $k = 4$, $v = 3$ convolution coder of figure 4.4

If $\mathbf{g}_J = (g_{J1}, g_{J2}, ..., g_{Jv})$ $J = 1, 2, ..., k$ means stage number in the shift register . Then

$\mathbf{g}_1 = (1,1,1)$
$\mathbf{g}_2 = (0,1,0)$
$\mathbf{g}_3 = (0,1,1)$
$\mathbf{g}_4 = (0,1,1)$

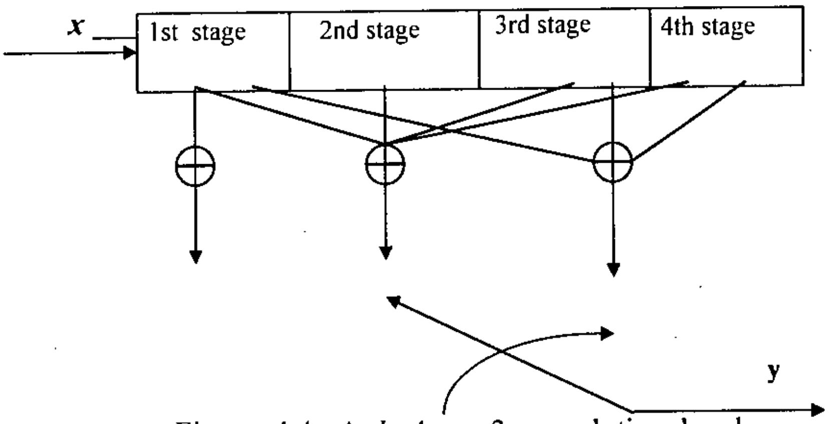


Figure 4.4 : A k=4 , v= 3 convolutional coder

we identify $f_1 = (\underbrace{g_1 , g_2, ..., g_k}_{\text{k vectors of } v \text{ bits. each}} , \underbrace{0 ,0,...,0}_{L \text{ zero vectors of } v \text{ bits each}})$

The commas in the definition of f_h indicates when the shift register shifts right .Where L is the dividing length of the message that is it is the length of a information block or information vector .

Explanation : for L zero vectors instead of T (number of tail bits for last information bit):

Let , the information vector has L bits . There are

1. (L-1) information bits in the information vector , before the last bit . Therefore correspondingly (L-1)v bits in the output .

2. For the last bit in the information vector , v bits are out . If there are no other message block to consider , then tail bits $T = k$ = constraint length of the encoder . So , correspondingly kv output bits .

$$\begin{aligned} \text{Total number of output bits} &= (L\text{-}1)v + v + kv \\ &= (k+L)v \text{ bits} \\ &= (k+L) \text{ vectors of } v \text{ bits each} \end{aligned}$$

If x = (0,1,0, ... ,0)

$$\text{Then output } \mathbf{y} = f_2 = (\underbrace{\mathbf{0}, \mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_k}_{k+1 \text{ vectors of } v \text{ bits each}}, \underbrace{\mathbf{0}, \mathbf{0}, ..., \mathbf{0}}_{L-1 \text{ zero vectors of } v \text{ bits each}})$$

Because , when only the second bit of **x** , i.e. x_2 is 1 . The output **y** is the delayed replica of the output when only $x_1 = 1$ in **x** . If we delete the explicit mention of the vectors $\{\mathbf{0}\}$, the $\{f_h\}$ may be described pictorially as shown in figure 4.5 for $k = 4$ and $L = 6$.

	colum n 1	Colum n 2	column 3	column 4	column 5	colum n 6	column 7	column 8	column 9	column 10
f_1:	$\mathbf{g}_1$	$\mathbf{g}_2$	$\mathbf{g}_3$	$\mathbf{g}_4$						
f_2:		$\mathbf{g}_1$	$\mathbf{g}_2$	$\mathbf{g}_3$	$\mathbf{g}_4$					
f_3:			$\mathbf{g}_1$	$\mathbf{g}_2$	$\mathbf{g}_3$	$\mathbf{g}_4$				
f_4:				$\mathbf{g}_1$	$\mathbf{g}_2$	$\mathbf{g}_3$	$\mathbf{g}_4$			
f_5:					$\mathbf{g}_1$	$\mathbf{g}_2$	$\mathbf{g}_3$	$\mathbf{g}_4$		
f_6:						$\mathbf{g}_1$	$\mathbf{g}_2$	$\mathbf{g}_3$	$\mathbf{g}_4$	

$\longleftrightarrow$ v bits

Figure 4.5 : Linearity property and formation of Convolutional Code

When considering f_6 the current information bit is x_6 . From the h th column of the figure -it is clear that the v digits of **y** is produced when x_h is first shifted in .The v output digits are computed from $x_h \cdot \mathbf{g}_1 \oplus x_{h-1} \cdot \mathbf{g}_2 \oplus ... \oplus x_{h-k-1} \cdot \mathbf{g}_k$. And taking into consideration , more than 1 bit of **x** to be 1 , output $\mathbf{y} = x_1 . f_1 \oplus x_2 . f_2 \oplus ... \oplus x_L . f_L$ .The effect of the tail bits are included in the construction of f_h , $h = 1,2, ..., L$. f_h's are vector and x_i's are scalar over the same finite field GF(2) and operations are modulo 2 , therefore according to the definition of vector space , and vector (section 2.2.4 , 2.2.5, 2.2.6) the convolutional code is a linear code . When $x_1 . f_1$ is a convolution codeword , $x_2 . f_2$ is a convolution codeword , and so on , their sum **y** is also a codeword of that convolution code with information vector is the sum of the information vectors of the two codewords . So, it is proved that *a convolutional code is a linear code* .

Normally block length (L) of a message is greater than the constraint length k . But let us consider for example a case of $\mathbf{x} = (101)$ $(k > L)$

L zero vectors of v bits each

$$\begin{array}{llllllll}
x_1 . f_1 = x_1 . \mathbf{g}_1 & x_1 . \mathbf{g}_2 & x_1 . \mathbf{g}_3 & x_1 . \mathbf{0} & x_1 . \mathbf{0} & x_1 . \mathbf{0} & x_1 . \mathbf{0} \\
\oplus x_2 . f_2 = x_2 . \mathbf{0} & x_2 . \mathbf{g}_1 & x_2 . \mathbf{g}_2 & x_2 . \mathbf{g}_3 & x_2 . \mathbf{0} & x_2 . \mathbf{0} & x_2 . \mathbf{0} \\
\oplus x_3 . f_3 = x_3 . \mathbf{0} & x_3 . \mathbf{0} & x_3 . \mathbf{g}_1 & x_3 . \mathbf{g}_2 & x_3 . \mathbf{g}_3 & x_3 . \mathbf{0} & x_3 . \mathbf{0}
\end{array}$$

output bits for entering tail bits

Adding (mod 2) column by column

$$\mathbf{y} = x_1 . f_1 \oplus x_2 . f_2 \oplus x_3 . f_3 \tag{4.1}$$

$$= [x_1 \;\; x_2 \;\; x_3] \begin{bmatrix} \mathbf{g}_1 & \mathbf{g}_2 & \mathbf{g}_3 & \mathbf{g}_4 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{g}_1 & \mathbf{g}_2 & \mathbf{g}_3 & \mathbf{g}_4 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{g}_1 & \mathbf{g}_2 & \mathbf{g}_3 & \mathbf{g}_4 & \mathbf{0} \end{bmatrix}$$

The matrix defined with $\mathbf{g}_i$'s is the generator matrix . But $\mathbf{g}_i$'s are not generator polynomial , they indicates to what adders , stage j is connected . It is clear (as first column has only $\mathbf{g}_1$ non zero , the rows of the generator matrix are linearly independent .The generator matrix has rows of size equal to the message size , i.e. is equal to the dividing length L .

So , from equation (4.1)

$x_1 . f_1$ = 111	010	0l1	0l1	000	000	000	000
$x_2 . f_2$ = 000	000	000	000	000	000	000	000
$x_3 . f_3$ = 000	000	111	010	011	011	000	000
y = 111	010	100	001	011	011	000	000

The encoder is a systematic coder because , the first adder takes as input , only the current information bit , and no other stage of the shift register is fed to it . So , other shift register stages have no effect in the first bit of each group of v output bits of rate $1/v$ encoders . Also for this , except $\mathbf{g}_1$ all other $\mathbf{g}_j$'s have first component equal to zero .

For , infinitely large message , the generator matrix is semi-infinite , that it grows to down and to right . If message vector is $\mathbf{x} = [x_1, x_2, ...]$ and k is the constraint length , the generalized form of the generator matrix is (The bounding braces of the generator matrix is not closed downward , as it is infinite .) :

$$\left.\begin{matrix}
\mathbf{g}_1 & \mathbf{g}_2 & \mathbf{g}_3 & \cdots & \mathbf{g}_k & \mathbf{0} & \cdots \\
\mathbf{0} & \mathbf{g}_1 & \mathbf{g}_2 & \cdots & \mathbf{g}_{k-1} & \mathbf{g}_k & \cdots \\
\mathbf{0} & \mathbf{0} & \mathbf{g}_1 & \cdots & \mathbf{g}_{k-2} & \mathbf{g}_{k-1} & \cdots \\
\vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots
\end{matrix}\right]$$

The downward growth is for the growth in the size of the message vector and the right side growth is to provide infinite number of shifts for message vector of infinite size. Also , each row is the right shifted form of the previous row .

Property 4.3.1.2 : Time invariance :

If the , input to output mapping function fun_u changes with time u , i.e. changes of function occur , as a new bit of a message vector enters the shift register , then all input bits are not treated equally . In this case the coder and code is time variant . But if the function is fixed with time , i.e.
$fun_u = fun$ for all u , $0 \leq u < L+T$, then the coder and code are time invariant (fixed over time). Here , L is the message length and T is the length (number) of tail bits .

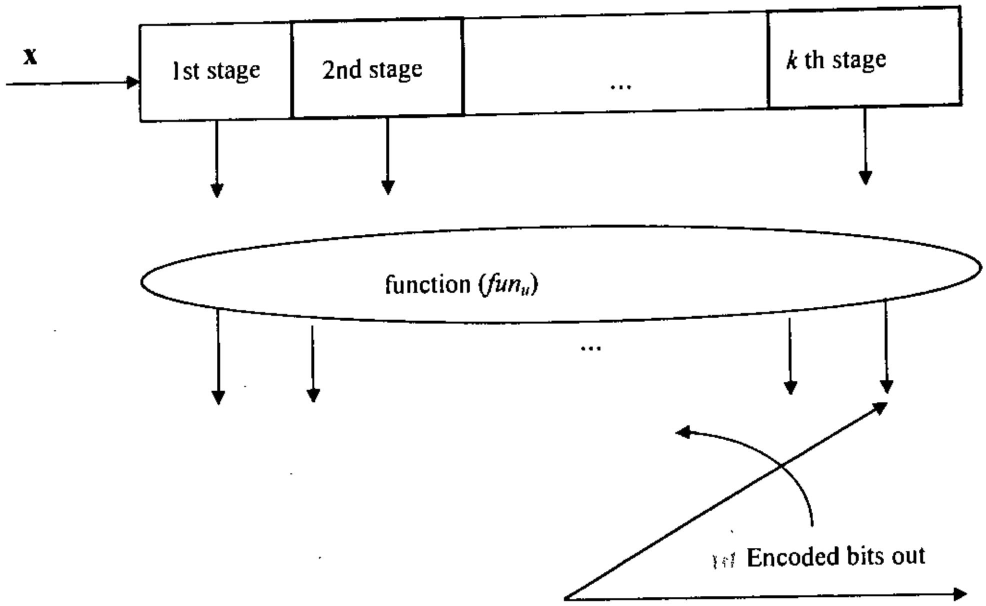


Figure 4.6 :A Convolutional Encoder for discussing time invariance

As a convolution code is linear, the mapping function is made with addition (mod 2 for binary case) . Also, the convolution code is time invariant, as the shift register stage to adders connection is fixed over time. The stage, to adder connection defines the generator matrix of the code, as is shown in section 4.1.

§ 4.4 : CODE TREE , TRELLIS, AND STATE DIAGRAM :

The structural properties of a convolutional encoder can be shown in graphical form by using any one of three equivalent diagrams :

- Code tree
- Trellis
- State diagram

All the graphical forms are discussed below in the context of encoder of figure 4.9(a) .

4.4.1 : Tree :

Each branch of the tree represents an input symbol , with the corresponding pair of output binary symbols indicated on the branch . The convention used to distinguish the input binary symbols 0 and 1 is as follows . An input zero specifies the upper branch of a bifurcation , where as input 1 specifies the lower branch . A specific path in the tree is traced from left to right in accordance with the input (message sequence) . The corresponding encoded sequence is found by concatenating labels on the followed branches .The diagram is called a *tree* because of its tree like structure . For example , let us consider , the message sequence 10011 applied to the input of the encoder of figure 4.9(a) . Following the procedure just described , we find the corresponding encoded sequence is (11,10,11,11,01) .

From the tree diagram , we observe that the tree becomes repetitive after the first 3 (equal to constraint length *k*) branches . Beyond the third branch , the two nodes labeled *a* are identical , the two nodes labeled *b* are identical and so on .

4.4.1.1 : Explanation Of The Repetitive Property :

The encoder has memory $M = k\text{-}1 = 2$ message bits . Hence when the third message bit enters the encoder , the first message bit (which is the reason of the first upper and lower branch or reason of variation) is shifted out of the encoder .When points *A2* or *B2* is reached the encoder state is same (00) . So , input of same message bits causes the same output at branch 4 . Count for same output after third level is 2 because of the variation of *A0* , *B0* message bits in figure 4.7.

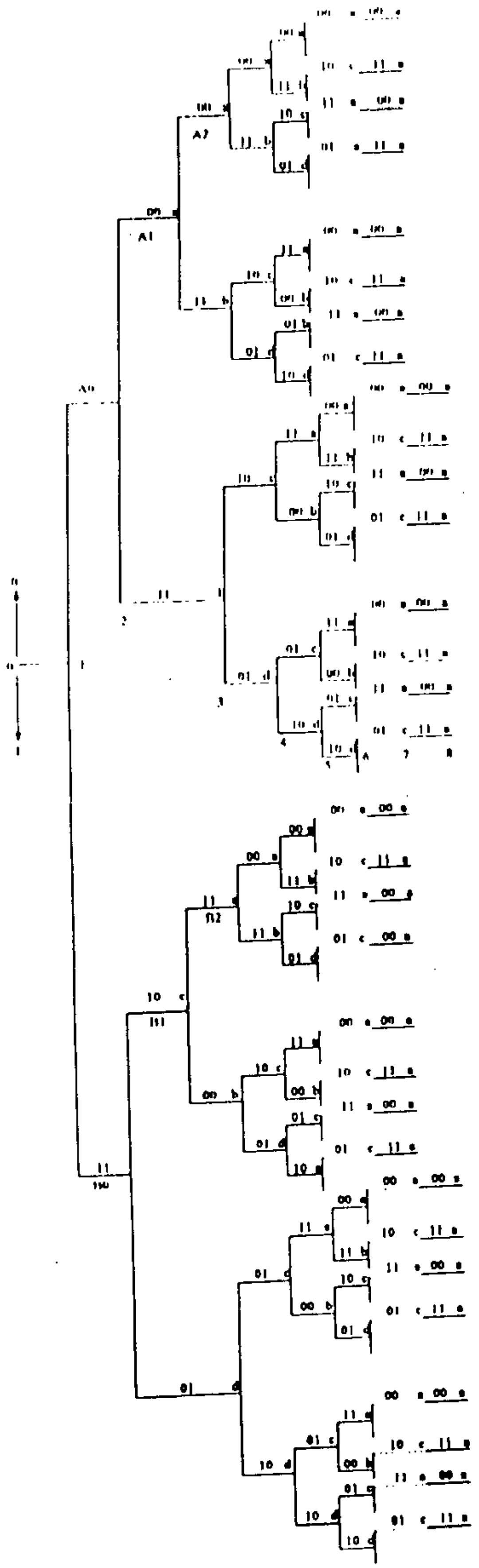


Figure 4.7 : Tree for a Convolutional Coder

4.4.2 : Trellis :

In another way , the explanation is - after the third branch , the message sequences (100 $m_3\,m_4$...) and (000 $m_3\,m_4$...) generate the same encoded symbols or code symbols . And the pair of nodes *a* may be joined together . The same reasoning may be applied to other nodes . Accordingly , we may collapse the code tree of figure 4.7 into a new form called trellis . It is so called since a trellis is a treelike structure with merged branches . The convention used to distinguish between input symbol 0 and 1 is as follows . (But in the trellis for different cases of Viterbi algorithm this convention is not followed , instead the upper branch diverging from a node is for 0 input and the lower branch diverging from a node is for input bit 1)

- A code branch produced by an input 0 is drawn as a solid line.
- Whereas a code branch produced by an input 1 is drawn as a dashed line .

Each input sequence corresponds to a specific path through the trellis . The message sequence (10011) produces the encoded output sequence (11,10,11,11,01)

State	Binary description
a	00
b	10
c	01
d	11

Table 4.1 : State table for the convolutional encoder of figure 4.9(a)

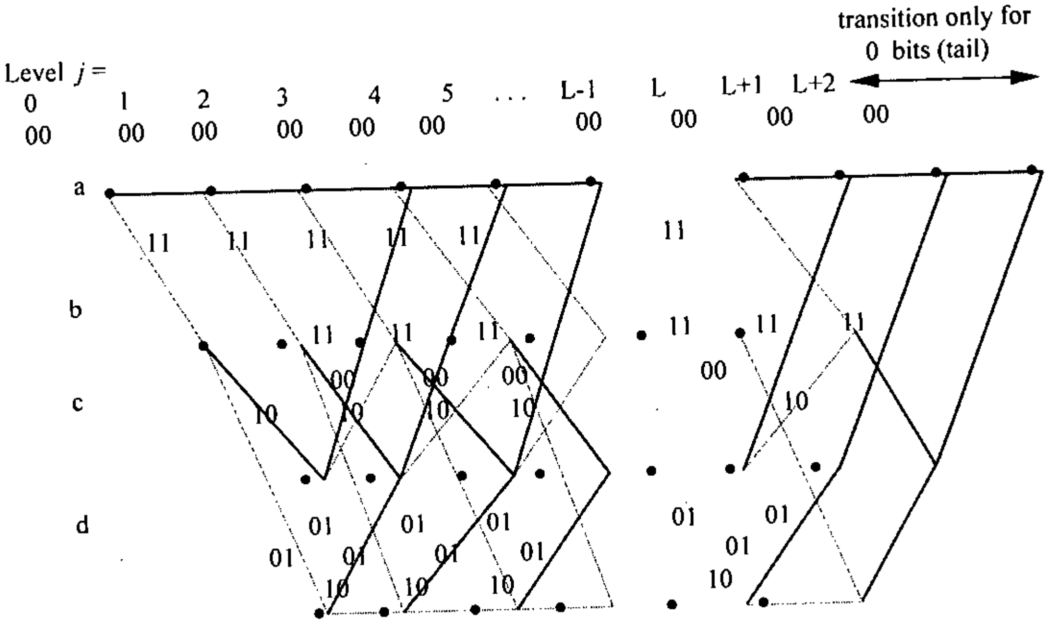


Figure 4.8 : A trellis with return to the all zero state

But in the trellis for different cases of Viterbi algorithm this convention of solid and dashed branch is not followed , instead the upper branch diverging from a node is for input bit 0 and the lower branch diverging from a node is for input bit 1.

A trellis is more instructive than a tree in that it brings out explicitly the fact that the associated convolutional encoder has finite number of states . The *state* of a rate $1/v$ encoder is defined as the (k-1) message bits stored in the encoder's shift register .

At time j , the portion of the message sequence containing the most recent k bits are written as $(m_{j-k+1}, \ldots, m_{j-1}, m_j)$- from oldest to newest , where m_j is the current bit .The (k -1) bit state of the encoder at time j is therefore written as $(m_{j-1}, \ldots, m_{j-k+2}, m_{j-k+1})$ .In our example (k-1) = 2 . Hence , the state of the encoder can assume any one of the four possible values ($q^{k-1} = 2^2 = 4$ states).

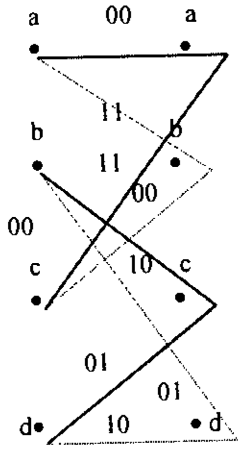
The trellis contains L+k levels (*level* is also termed as *depth*) , where L is the length of the incoming message sequence , and k is the constraint length of the code . The levels of the trellis are labeled as $j = 0,1,\ldots,L+k-1$.

The first (k-1) levels correspond to the encoder's departure from the initial state a and the last (k-1) levels correspond to the encoder's return to the state a , that is not all four states can be reachable in these two portions of the trellis .

However , the central portion of the trellis , for which the level j lies in the range k-1 $\le j \le$ L, all states of the encoder are reachable . The central portion of the trellis shows a fixed periodic structure .

4.4.3 : Signal Flow Graph Or State Diagram :

Next , let us consider a portion of the trellis corresponding to times j and j+1 . We assume $j \ge 2$, so that all four states a,b,c,d appears in the trellis . The left nodes represent the four possible current states of the encoder , whereas the right nodes represent the next states . We may coalesce the left and right nodes . By doing so , we obtain the *state diagram* of the encoder .



The nodes of the above figure represent the four possible states of the encoder, with each node having two incoming branches and two outgoing branches . The state diagram is given in figure 4.9(b).

1. A transition from one state to anothcr in response of input 0 is rcpresented by writing a 0 on the branch .

2. A transition from one state to another in response to input 1 is represented by writing a 1 on that branch .

3. A binary label (in our examples it is a two tuples)on each branch represents the encoder's output (encoded sequence) as it moves from one state to another .

For example , let us assume that , the current state of the encoder is *c* (01) . The application of input 1 to the encoder causes the state to be state *b* (10) and the encoded output is 00 .

4. From this state diagram , for any incoming message sequence , we can determine the output of the encoder . For this we simply start at state *a* (00) (because initially the encoder is at all zero state), the all zero initial state , and walk through the state diagram in accordance with the message sequence .We follow a branch labeled 0 if the input is a 0 and a branch labeled 1 if it is a 1 . As each branch is traversed , we output the corresponding binary label (in our examples two tuples) on the branch . For example , for the message sequence 10011 , we follow the path *abcabd* , and therefore the output sequence is (11,01,11,11,01) .

We shall find it convenient to write $\mathbf{i}_{[u,v]}$ and $\mathbf{i}_{[u,v)}$ for the sequences $\mathbf{i}_u$, $\mathbf{i}_{u+1}$, ... , $\mathbf{i}_v$ and $\mathbf{i}_u$, $\mathbf{i}_{u+1}$, ... , $\mathbf{i}_{v-1}$ respectively and similarly for $\mathbf{t}_{[u,v]}$ and $\mathbf{t}_{[u,v)}$.

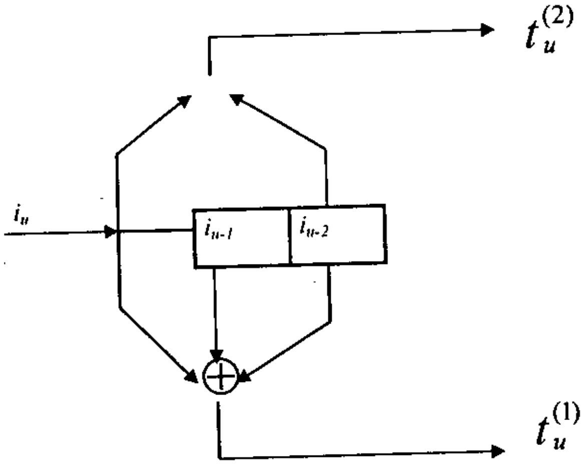


figure 4.9 (a) : A binary convolutional encoder (noncatastrophic and nonsystematic).

The above encoder has memory $M = 2$ and constraint length $k = M + 1 = 3$.

The signal flow chart or state diagram can be constructed as follows :

There is a node for every state $[i_{u-1}, i_{u-2}]$ .From each state there is a directed edge to each of the possible successor states , the directed edge is labeled with current input bit i_u . As for example , from state $[i_{u-1}, i_{u-2}]$ to one of the possible successor states $[0, i_{u-1}]$ or $[1, i_{u-1}]$ there is directed path . As $i_u = 0$ or $i_u = 1$ is shifted into the first stage of the shift-register , corresponding move of i_{u-1} to the second stage of the shift register and shifting out of i_{u-2} occurs).

The directed edge is labeled with Z^w or D^w , where w is the hamming weight of the encoded branch $[t_u^{(1)}, t_u^{(2)}]$ that results from the transition from a state to its successor state . For instance when $[i_{u-1}, i_{u-2}] = [1,1]$ the input $i_u = 0$ causes the encoded branch $[t_u^{(1)}, t_u^{(2)}] = [0,1]$ and causes the next state to be [0,1] . Since the hamming weight (i.e. number of nonzero digits) of the branch is 1 , the transition from state [1,1] to the state [0,1] is labeled with Z^1 or D^1 .

In the general case of GF(q) , and when the shift register has memory (k-1) and each input symbol i_j is an m-tuple over GF(q) then there are $q^{(k-1)m}$ states or nodes in the signal flow graph or state diagram . So , even in the binary case ($q = 2$) the construction of the flowchart is impractical unless (k-1)m is quite small . For our example in figure 4.9(b) k =3 and m= 1 bit .

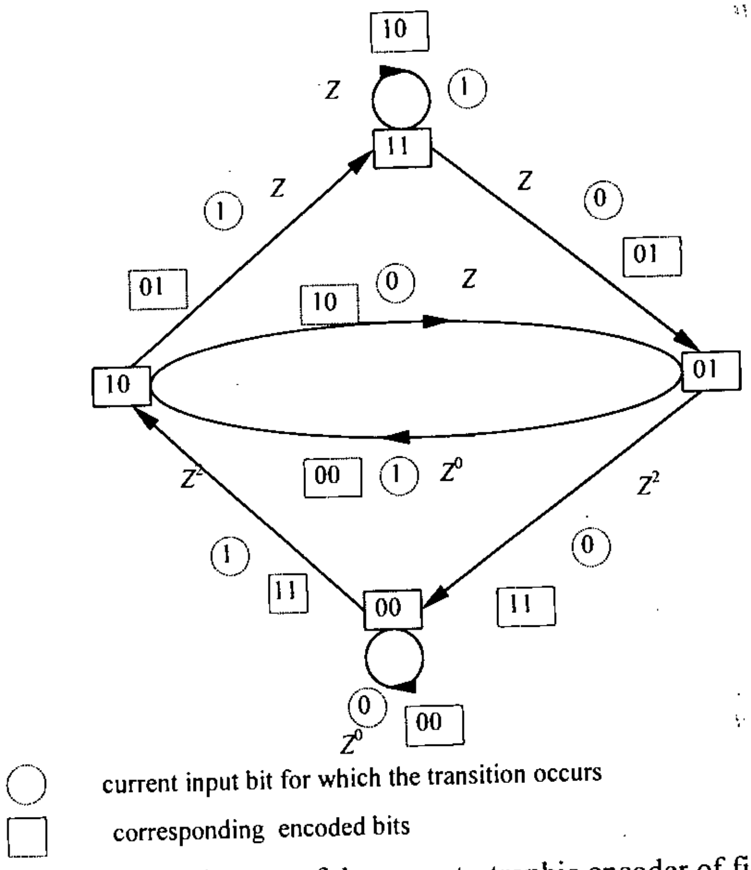

10
Z
1
11
1
Z
Z
0
01
01
10
0
Z
10
01
00
1
Z^0
Z^2
Z^2
1
0
11
00
11
0
00
Z^0
current input bit for which the transition occurs
corresponding encoded bits

figure 4.9(b) : State diagram of the noncatastrophic encoder of figure 4.9 (a)

4.4.3.1 : Why There Is A Self Loop In The All Zero State :

Since the input $i_u = 0$ applied to the state $[i_{u-1}, \dots, i_{u-M}] = [0,\dots,0]$ (where M is the encoder memory) , it always results in $\boldsymbol{t}_u = \boldsymbol{0}$, the zero vector ($\boldsymbol{t}_u$ is the vector of encoded bits , each distinct encoded bit of $\boldsymbol{t}_u$ is denoted as $[t_u^{(1)}, t_u^{(2)}, \dots, t_u^{(v)}]$ where $v = 1/\text{rate}$. The all zero state would return to the all zero state and the state diagram will always have a self loop labeled $Z^0 = 1$ at the zero state .

§ 4.5 : CATASTROPHIC CONVOLUTIONAL ENCODERS :

A convolutional code is *catastrophic* if and only if there is an information sequence of infinite weight which produces an encoded sequence of finite weight .The convolutional coder is catastrophic if there is any other self loop in the state diagram at a state other than the all zero state having label $Z^0 = 1$, i.e. whose loop gain is 1 (unity gain).

Mathematically , a convolutional encoder is catastrophic if and only if there is an information sequence $\mathbf{i}_{[0\,.\,\infty]}$ with hamming weight $W_H(\mathbf{i}_{[0,\infty)}) = \infty$ that produces an encoded sequence $\mathbf{t}_{[0,\infty)}$ for which $W_H(\mathbf{t}_{[0,\infty)}) < \infty$. To see this equivalence we note that if the signal flow graph or state diagram has a closed loop of weight zero besides the self-loop at the zero state , then the information sequence $\mathbf{i}_{[0,\infty)}$ which drives the encoder to a state on the former loop and moves around the loop forever for an infinite weight message sequence which produces a finite weight encoded sequence . Conversely , if there is no zero weight or unity gain loop besides the self - loop at the zero state , then an information sequence of infinite weight must drive the encoder through infinitely many loops besides the self-loop at the zero state and thus must produce an encoded sequence of infinite weight also .

More explanation of the above statement :

Let the transmitted sequence be the all zero codeword . But it is received with few bit error . If there is a self-loop (loop1) with unity gain other than that (loop2) at the all zero state , then a received sequence which drives the encoder to a state having the former loop (loop1) , loops forever as this loop has gain Z^0 and the loop can be traveled without increasing weight (number of non zero bits) of the estimate of the codeword - this path would be followed or preferred in making decoding decisions if the travel along other path to return to the all zero state would increase the weight or (as convolutional code is linear) the hamming distance (because less errors between received and transmitted sequence is most likely- we always prefer that path that is at minimum hamming distance from the received sequence).

When loop1 has gain Z^a and $a \neq 0$, to reduce hamming weight of the received word , the decoder would eventually return to the all zero state as a preferred path and loops there in order to keep the hamming weight low .

As the convolutional code is linear any example can be shown with respect to the all zero code word as the all zero vector is in the set of valid code words .

The encoder in figure 4.9(a) is non catastrophic . The encoder given in figure 4.10(a) is catastrophic because besides the self loop of label Z^0 at state [0,0] , in the state diagram there is another self loop with unity gain , namely the self loop at state [1,1] .

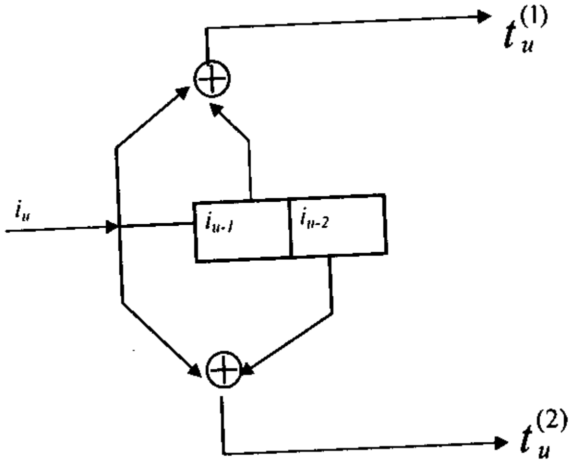
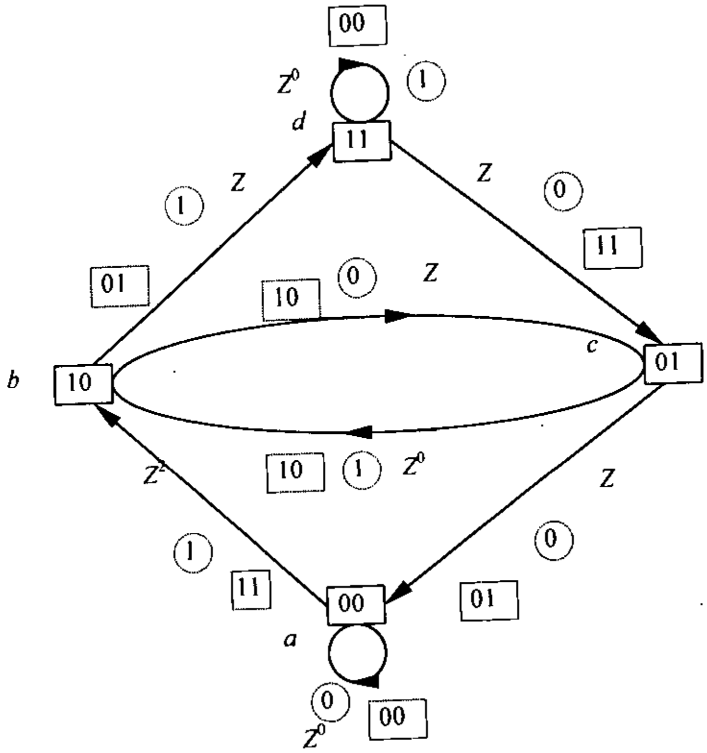


figure 4.10(a) : A binary convolutional encoder (catastrophic and nonsystematic) .



a ,*b*, *c*, *d* represents states .

figure 4.10 (b): State diagram of the catastrophic encoder of figure 4.10(a)

When an infinite number of decoding errors are caused by a finite number of transmission errors (that is the received vector contains error) the convolutional coder is then subject to catastrophic error propagation , and the code is called the catastrophic code . This case arises due to improper coder design .

- A systematic convolutional code can not be catastrophic .

- for a prescribed constraint length *k* , the free distance that can be attained with systematic convolutional codes is usually smaller than for the case of non systematic convolution codes - so , correspondingly poorer error performance (this is obvious from the fact that one of (may be the first) each *v* output bits in the case of systematic code is the current input bit itself - so no coding built into it).

- For catastrophic coder , calculation of T(*D*) leads to no solution .

Let us consider the case that for the encoder of 4.10(a) , the transmitted codeword is the all zero codeword (0000000...000) (so also the input is all zero). The correct path corresponds to staying in state *a*. For occurrence of error if the transmitted codeword is received as 11100000000...00 . That is , there is finite number of error (3 bit) . For this error , the preferred path (with respect to the lowest hamming weight between received and estimated bits) the path *a-b-d-d-...-d-d* instead of *a-b-c-a-a-...a-a* (does not cause infinite number of errors in making input bit decision) . The incorrect path *a-b-d-d-...dd* has a distance 6 independent of the number of times the self loop at the state *d* is traversed . A decoder selecting this path as correct one would incorrectly decode the data or input bit sequence as two 1's for the (*a-b* and *b-d* portions of the path) plus a 1 (cause of infinite number of errors in input bit decision) each time the self loop at *d* is traversed , because of the self loop of unity gain at state *d* there is increase in hamming weight due to infinite traversal along this self loop and so no chance of return to the correct path after few erroneous decision .

If the self loop at *d* has non unity gain , then due to transmission error if ever this path were selected as correct one by the decoder and the actual codeword was the all zero code word , before infinitely looping at this self loop or along other loop (but not self loop) the decoder would return to the all zero state . Because , traversal infinitely along any other loop except the self loop at the all zero state would increase the hamming weight of the estimated transmission , eventually after few looping at *d* the path *c-a-a-...a-a* would be traversed and as such no catastrophic error propagation is possible .

When the coder of a convolutional code and corresponding code are systematic , no self loop other than that at the all zero state can have gain $Z^0 = 1$, because in a systematic coder one of the *v* output bits (due to an input bit) is the input bit itself . Also there are only two possible self loop in the state diagram for a coder over GF(2) - at the all zero state and at the all one state .The all one state would be at the all one state when the current input is 1 so at least one output bit is also 1 , for this the self loop can not have label of $Z^0 = 1$, and if the decoder incorrectly reaches at this state it eventually (with a finite number of erroneous decision) would return to the all zero state .

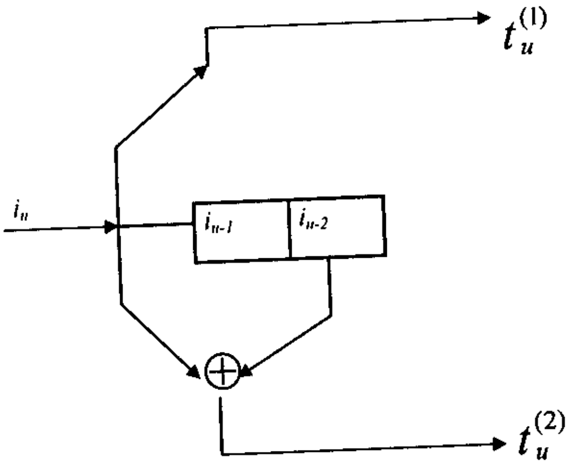
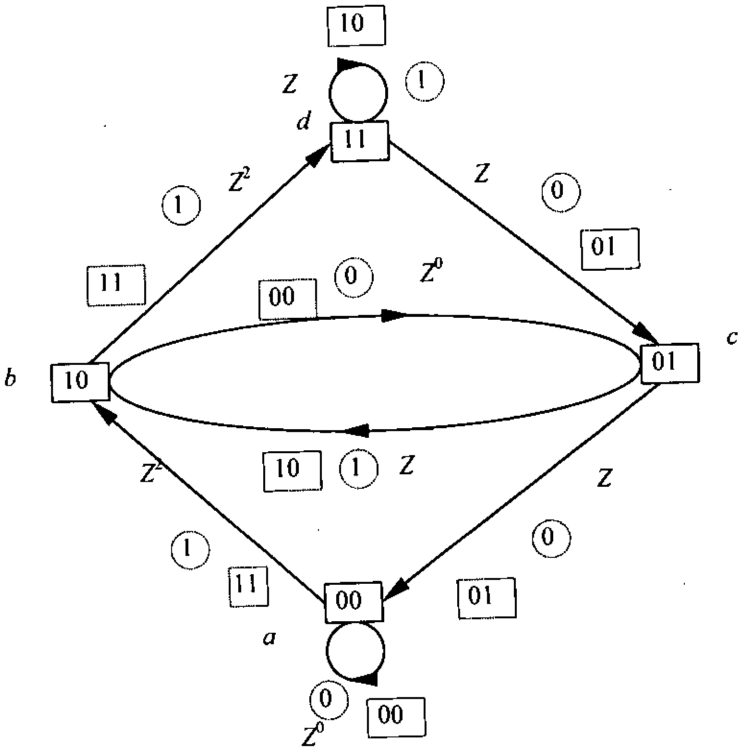


Figure 4.11(a) : A binary convolutional encoder or coder (noncatastrophic and systematic) .



a ,*b*, *c*, *d* represents states .

Figure 4.11(b) : State diagram of the encoder of figure 4.11(a)

In the discussion of linearity of convolutional code we have mentioned that g_1 means the vector representing which adders are connected to stage 1 (in non minimized form) of the shift register in the encoder .

g_2 means the vector representing which adders are connected to stage 2 (in non minimized form) of the shift register in the encoder . And so on .

Let $g_1 = (a_{11}, a_{12}, \ldots, a_{1v})$

where $a_{ij} = 1$ or 0 , means connection or no connection between state i and adder j . 1 means connection .

$g_2 = (a_{21}, a_{22}, \ldots, a_{2v})$. And so on .

Collecting stages over same adder $A_1 = (a_{11}, a_{21}, \ldots, a_{k1}) =$ connection vector from all stages to first adder . $A_2 = (a_{12}, a_{22}, \ldots, a_{k2}) =$ connection vector from all stages to 2nd adder . Similarly $A_v = (a_{1v}, a_{2v}, \ldots, a_{kv}) =$ connection vector from all stages to v th adder . (there are k stages in the shift register) .

Delay associated with 1st stage = 1 . Delay associated with 2nd stage = D^1. delay associated with 3rd stage = D^2. delay associated with k th stage = D^{k-1}

For , figure 4.4

$A_1 = (1\ 0\ 0\ 0)$ [vector representation]
= connection vector from all stages to first adder .
$= 1$ [polynomial representation]

$A_2 = (1\ 1\ 1\ 1)$
= connection vector from all stages to 2nd adder .
$= 1 + D^1 + D^2 + D^3$

$A_3 = (1\ 0\ 1\ 1)$
= connection vector from all stages to 3rd adder .
$= 1 + D^2 + D^3$

These A_j's are generator polynomials .

4.5.1 : Non Catastrophic Convolution Code :

A convolutional code whose generator polynomials $A_1, A_2, \ldots, A_v$ satisfy $GCD\,[A_1, A_2, \ldots, A_v] = D^a$, for some $a \geq 0$ is called a non catastrophic convolutional code . Otherwise it is called a catastrophic convolutional code .GCD means greatest common divisor .

Without loss of meaningful generality we can take $D^a = 1$, that is $a = 0$, because otherwise it corresponds to a simple delay in each stage of the shift register .

Theorem 4.5.1.1 : A Systematic Code is Always Non Catastrophic :
From the definition in section 4.5.1 we can prove that a systematic code is always non catastrophic.

proof:
Because for a non catastrophic code *GCD* of generator polynomials is D^a . There is one generator polynomial (connection of stages to an adder , and an adder output is one of the encoded bits for a new input) which has only one term D^a , this means the input is when multiplied by this generator polynomial of the generator matrix , the corresponding output (from the adder representing D^a) is only the delayed replica of the input bit . If the delay is $D^0 = 1$, we get the current input directly to the output without delay - which is true for systematic code . So a systematic code is always non catastrophic (from definition).

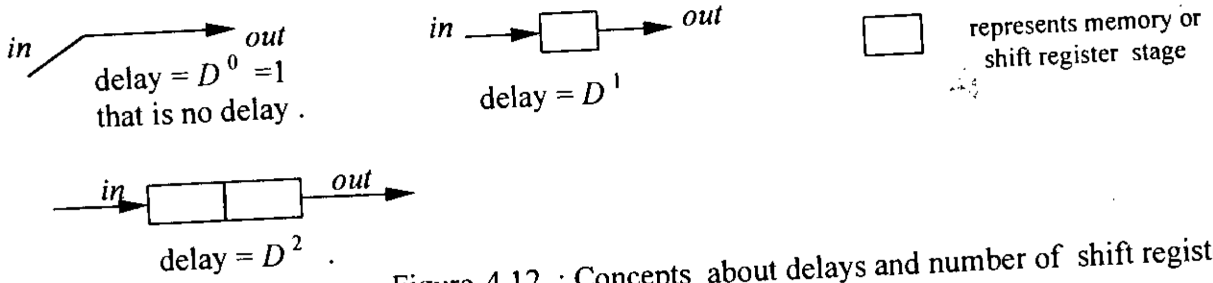


Figure 4.12 : Concepts about delays and number of shift register stages .

4.5.2 : Way Of Forming Non Catastrophic Coder From Catastrophic Coder :

For the catastrophic coder of figure 4.10(a) the transfer function matrix formed from generator polynomials A_1 , A_2 is

$$\begin{aligned} T(D) &= [A_1 \quad A_2\ ] \\ &= [1+D \quad 1+D^2] \\ &= [1+D \quad 1+2D+D^2] \qquad \text{(as over mod 2 operation } 2D=0\text{)} \\ &= [1+D \quad (1+D)^2\ ] \end{aligned}$$

$$\mathrm{GCD}[A_1 \quad A_2] = 1+D \neq D^a \ \text{ for } a \geq 0$$

$$\begin{aligned} T(D) &= (1+D)[1 \quad 1+D] \\ &= (1+D)\, T'(D) \qquad \text{Let } T'(D) = [1 \quad 1+D] \end{aligned}$$

Here $T'(D)$ is a new transfer function matrix . If one divides each entry of the transfer function matrix $T(D)$ (i.e. one row of the generator matrix) for a catastrophic convolutional coder by the *GCD* , transfer function matrix $T'(D)$ for a simpler convolutional coder with memory M =1 (highest power of D) is found , it has $d_0 = 2$ (0 means 1 branch), $d_i = 3$, for $i \geq 1$ (2 or more branch) just as was the catastrophic coder of figure 4.10(a) from which it is derived . The coder from $T^{/}(D)$ and its state diagram is given below .

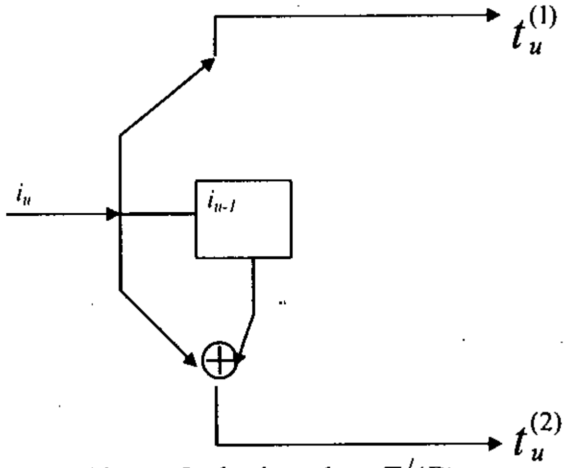


Figure 4.13(a) : Coder based on $T^{/}(D)$

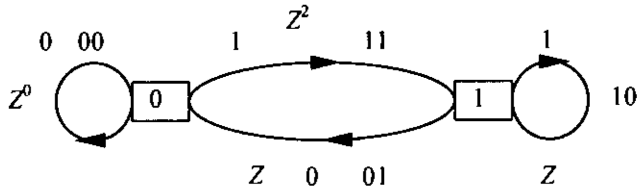


Figure 4.13(b) : State diagram for the noncatastrophic coder of figure 4.13(a)

So (as $T^{/}(D)$ is formed by removing GCD from T (D)) from any catastrophic convolutional coder one can obtain an equivalent non catastrophic convolutional coder with same d_i and perhaps less memory (for binary case memory is highest power of D, in non binary cases and when each input is an m tuple , memory is multiple of highest power of D).

4.5.3 : Why Catastrophic Coder Should Be Avoided :

1. Due to the fact that finite number of transmission errors results in infinite number of errors in decoding decisions .

2. For any catastrophic convolutional coder one can find an equivalent non catastrophic convolutional coder with the same minimum free distance (d_F) and perhaps less memory .And it is always desired to have less memory with same (d_F) , to reduce cost .

§ 4.6 : VITERBI DECODING :

The equivalence between maximum likelihood decoding and minimum distance decoding for a binary symmetric channel implies that , we may decode a convolutional code by choosing a path in the code tree whose coded sequence differs from the received sequence in the fewest number of places .

Since a code tree is equivalent to a trellis , we may equally limit our choice to the possible paths in the trellis representation of the code . The reason for preferring the trellis over the tree is that the number of node at any level of the trellis does not continue to grow as the number of incoming message bits increases , rather it remains constant at 2^{k-1} , where k is the constraint length of the code .

The viterbi decoder operates iteratively frame by frame , tracing through a trellis identical to that used by the encoder in an attempt to emulate the encoders behavior .The algorithm operates by computing a metric or discrepancy for every possible path in the trellis . The metric for a particular path is defined as the hamming distance between the coded sequence represented by the path and the received sequence . Thus , for each node (state) in the trellis , the algorithm compares the two paths entering the node . The path with the lower hamming distance metric is retained , and the other path is discarded The computation is repeated for every level j of the trellis in the range $M \leq j \leq L$, where $M = k\text{-}1$ is the encoders memory and L is the length of the incoming message sequence . The path that are retained by the algorithm are called the survivor or active paths . For a convolutional code of constraint length k=3 for example , no more than $2^{k-1} = 2^M = 2^2 = 4$ survivor paths and their metrics will ever be stored . The list of 2^{k-1} paths is always guaranteed to contain the maximum likelihood choice .

4.6.1 : Viterbi Algorithm :

Step1 : Initialization :

Let us label the left most state of the trellis (i.e. , the all-zero state at level 0) as 0 , since there is no discrepancy at this point in computation .

Step2 : Computation step j+1

let , j=0,1,2, ... , and suppose that at the previous step j we have done two things : identified all survivor paths to each state .Stored the survivor path and its metric for each state of the trellis .Then , at level (clock time) j+1 the metric for all the paths entering each state of the trellis is found .

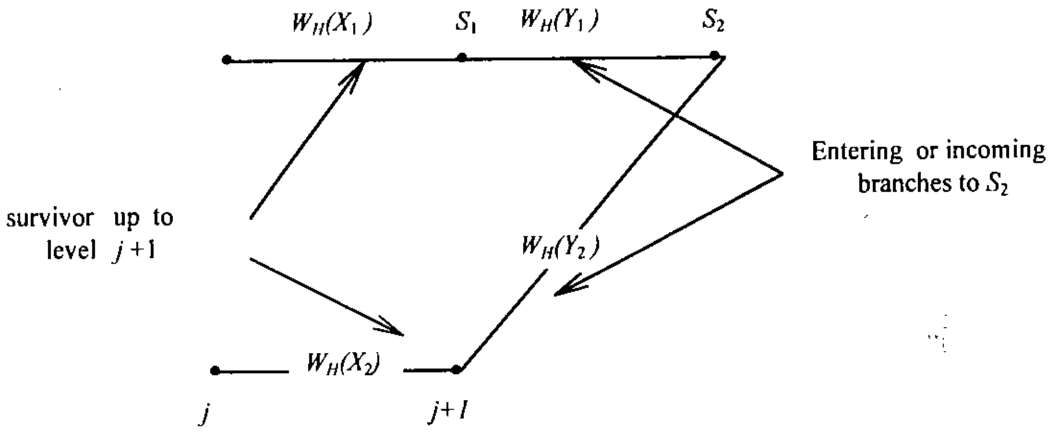
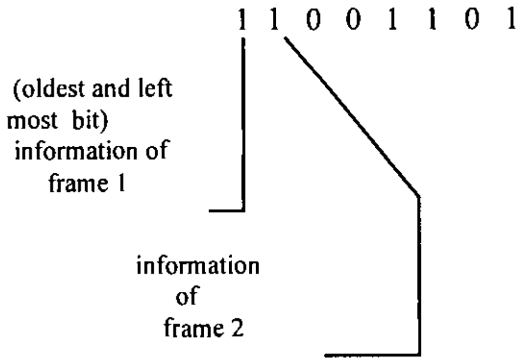

$W_H(X_1)$
S_1
$W_H(Y_1)$
S_2
Entering or incoming branches to S_2
survivor up to level $j+1$
$W_H(Y_2)$
$W_H(X_2)$
j
$j+1$

Figure 4.14 : Calculation of Survivor

By adding the metric of incoming branches $W_H(Y_m)$ for some m , with the metric of connecting survivor up to level $j+1$ for state S_2 . Let metric 1 = $W_H(X_1) + W_H(Y_1)$ .And metric 2 = $W_H(X_2) + W_H(Y_2)$. From this two metric , the metric which have lowest value defines the survivor path to state S_2 , at level $j+1$, and as such , metric for each state at each level is updated .

The iteration of computation is continued until the algorithm reaches the all zero state .(If there is error introduced in the tail , even then return to the all zero state would occur as the trellis is formed in that way for $L+T$, but the label of the traversed path may be different .) At which time it makes a decision on the maximum likelihood path . All zero path should be reached because of the added tail bits . At this time the decoder makes a decision on the maximum likelihood path (minimum probability of error) . The sequence of information which causes the transition along the path is the decoded information sequence .



When the received sequence is very long (near infinite), the storage requirement of the viterbi algorithm becomes too high (if decoding window width is 15 and no tie , then for rate $1/v$ coder of constraint length k , $1\times15\times2^{k-1}$ bits are required for estimated information bit storage) but when no truncation of received sequence is employed , if the whole information has length L , then $1\times L\times2^{k-1}$ (when no tie) bits are required and this approaches to infinity when $L \rightarrow \infty$. The active path through the trellis could be represented as a table . Let information sequence of active path is as given above .

To reduce the storage requirement some compromises must be made . The usual approach is to "truncate" the path memory of the decoder as described here . A decoding window of length L is specified , and the algorithm always stops after L steps . A decision is then made based on the "best" path and the information bit symbol associated with the first branch on the path is released to the user . Next , the decoding window is moved forward one time interval , and the decision for the next information bit is made , and so on. The decoding decision with truncation employed is no longer truly maximum likelihood . Because when the whole received sequence is considered without truncation and when error is within correctable range then only unambiguous (one) decision is made taking path with lowest metric (lowest probability of error) as the most likely path .

But , when truncation is employed , and decoding window width is so small that unambiguity may not be resolved , in such case decoding failure occur - choosing path with probability of lower error leads to no solution - in some cases no longer most likelihood - due to decoding error probability of error is high . This problem can be resolved by choosing the decoding window width long enough .

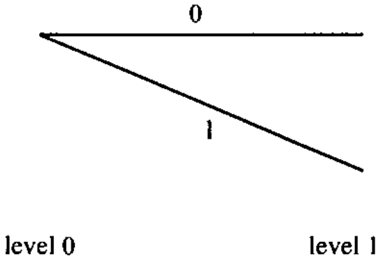


Figure 4.15 Ambiguity in Decision

If both I and 0 can be decided in the first frame due to be part of path of same discrepancy , no unambiguity in decision is possible .

Theorem 4.6.2 : A path that is not part by part optimum cannot be optimum as a whole :

proof:

Suppose that $y_{[0,L+T)}$ is the sequence received over the channel where $y_{[0,L+T)} = [y_0, y_1, ..., y_{L+T-1}]$ and where each y_u is a v (=1/rate ,for our example 2 bit) tuple of channel output letters .

Similarly let $x_{[0,L+T]}$ be the encoded sequence of the information sequence $i_{[0,L+T)}$. A maximum likelihood decoder chooses as its estimate $\hat{i}_{[0,L+T)}$ which maximizes

$$p\left(y_{[0,L+T)}\middle|x_{[0,L+T)}\right) = \prod_{u=0}^{L+T-1} p\left(y_u \middle| x_u\right)$$

y_u , x_u each are v bits

or equivalently which maximizes the statistics

$$\log p\left(y_{[0,L+T)}\middle|x_{[0,L+T)}\right) = \sum_{u=0}^{L+T-1} \log p\left(y_u \middle| x_u\right) \tag{4.2}$$

since y will be fixed throughout our discussion , for compactness we can write simply

$$L_0\left(x_{[u,v]}\right) = \log p\left(y_{[u,v]}\middle|x_{[u,v]}\right) \tag{4.3}$$

And from (4.2) and (4.3) $\quad L_0\left(x_{[u,v]}\right) = \sum_{i=u}^{v} L_0\left(x_i\right)$

In fact , it is not needed that L_0 always be the logarithmic function , but it is simply a statistics whose maximization yields an *MLD* (maximum likelihood decoding) and which is additive .As for example the statistics can be hamming distance (minimum hamming distance maximizes statistics). So we can take

$L_0(x_i) = -d(x_i, y_i)$ i.e., the negative hamming distance .

If the paths $i'_{[0,u]}$ and $i''_{[0,u]}$ terminates at the same node of the trellis and

$$L_0\left(x'_{[0,u]}\right) > L_0\left(x''_{[0,u]}\right) \tag{4.4}$$

Then the nonoptimum path $i''_{[0,u]}$ cannot be the first u+1 branches of the path (optimum path or the path which maximizes statistics) $i_{[0,L+T)}$ which maximizes $L_0\left(x_{[0,L+T]}\right)$. The portion of the optimum path is also optimum . Suppose , that not $i'_{[0,L+T]}$, but

$$i''_{[0,L+T)} = i''_{[0,u]} * i''_{[u+1,L+T)} \tag{4.5}$$

(The * denotes concatenation of sequences) maximizes the statistics L_0 . Let us consider the path $i'_{[0,L+T)} = i'_{[0,u)} * i''_{[u+1,L+T)}$. The encoded sequence $x'_{[0,L+T)} = x'_{[0,u)} * x'_{[u+1,L+T)}$ must have $x'_{[u+1,L+T)} = x''_{[u+1,L+T)}$ since $i'_{[0,u]}$ and $i''_{[0,u]}$ terminates on the same node and we have used the same subsequent information sequence (branch) $i''_{[u+1,L+T)}$ in both cases .Also , $i''_{[0,L+T)}$ has encoded sequence $x''_{[0,L+T)} = x''_{[0,u)} * x''_{[u+1,L+T)}$. Now

$$
\begin{aligned}
L_0\left(x'_{[0,L+T)}\right) &= L_0\left(x'_{[0,u]} * x''_{[u+1,L+T)}\right) \\
&= L_0\left(x'_{[0,u]}\right) + L_0\left(x''_{[u+1,L+T)}\right) \\
\text{By 4.4} \quad &> L_0\left(x''_{[0,u]}\right) + L_0\left(x''_{[u+1,L+T)}\right) \\
&= L_0\left(x''_{[0,L+T]}\right)
\end{aligned}
$$

That is if $L_0\left(x'_{[0,u]}\right) > L_0\left(x''_{[0,u]}\right)$, $i''_{[0,L+T)}$ does not maximize the statistics (if its portion is nonoptimal by (4.4)) as is initially assumed - a contradiction .

The viterbi algorithm is based on this logic . When decision is based on the whole information sequence viterbi algorithm is optimum . But , when truncation is employed on received sequence (that is decision is made after seeing a portion) and there are multiple optimum paths and on guessing one of the several optimum path is selected for making decision , in later portions of the received words the selected path may not be optimum - causing suboptimum decision on the whole sequence - in such case the viterbi algorithm is no longer optimum - but becomes suboptimum .

The maximum likelihood solution is to always select the path with the largest likelihood metric , the metric or path retained for entering each state is called the survivor . Since the maximum likelihood solution is to always select the path with the largest metric , a path with a smaller metric that has been rejected could never have caught up with the survivor . So , always selecting the survivor of the two , entering each state thus retains the optimum (maximum likelihood character of the solution) .

The number of surviving path always remains at 2^{k-1} , never increasing exponentially with L (like 2^L).

For binary GF(2) encoder there are only two possible ways of leaving (2) or entering (2) a state for each bit input .

The metric for 2^{k-1} survivors are kept .

4.6.2 : Difficulties In The Application Of Viterbi Algorithm :

When paths entering a state are compared and found to have identical metric there is a problem in selecting survivor . Solution: the decoder may either break a tie by guessing or keep all such paths (in our case two paths to a state) until they are replaced by a more likely path.

As the decoder progress through many frames , the accumulating discrepancies continue to increase . Overflow may occur .To avoid overflow problems , they must be reduced occasionally . A simple procedure is to periodically subtracting the smallest discrepancy from all of them .This does not affect the logic of path choice .

When the received sequence is very long (near infinite) , the storage requirement of the Viterbi algorithm becomes too high (number of state * length of information sequence), and some compromise must be made . The approach usually taken is to truncate the path memory of the decoder . A decoding window of length l is specified , and the algorithm always stops after l steps . A decision is then made on the "best" path and the symbol associated with the first branch on that path is released to the user and this stored symbol is dropped . Next , the decoding window is moved forward one time interval , and a decision on the next information bit is made , this stored information bit is also dropped out .

Since at each level there are 2^{k-1} comparisons , k cannot be too large .

4.6.3 : Examples Of Different Cases With Viterbi Algorithm :

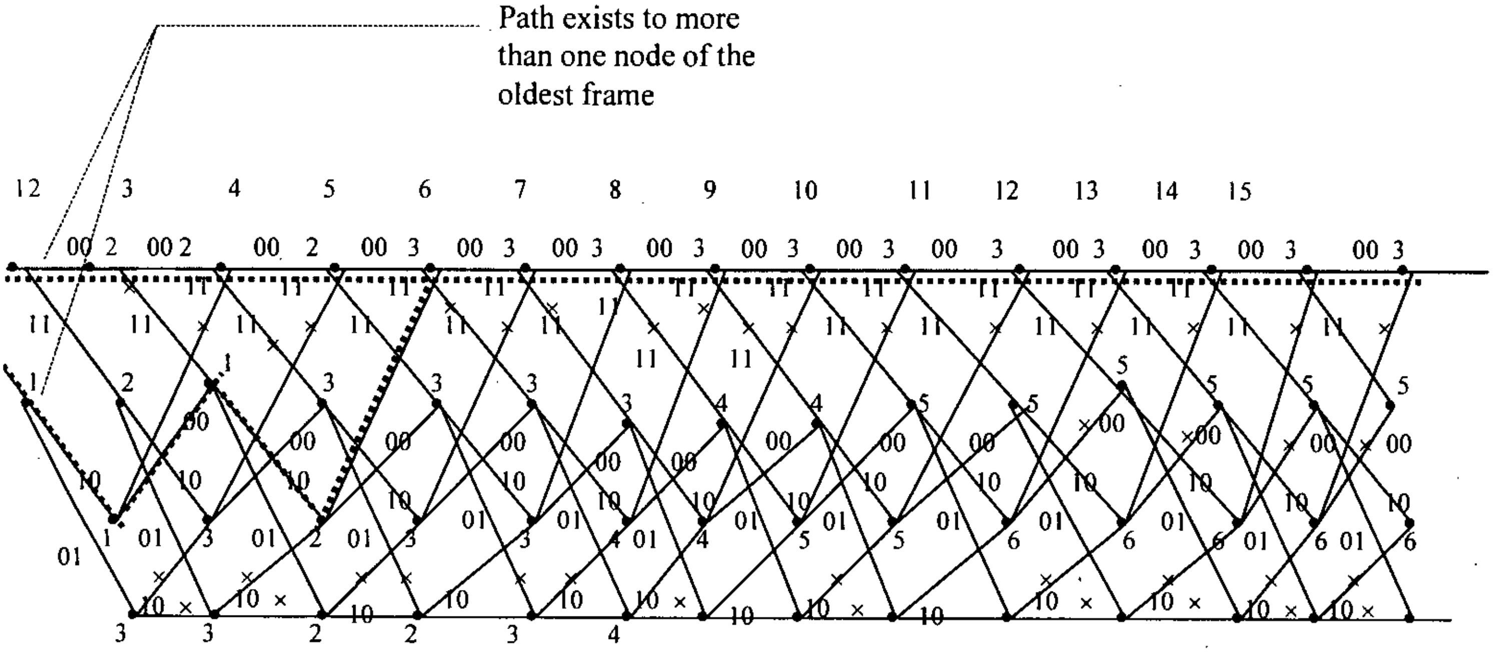

Path exists to more than one node of the oldest frame

Figure 4.16 : Trellis for an encoder with generator polynomials
$g_1(x) = x^2 + x + 1$ and $g_2(x) = x^2 + 1$

The code has two error correcting capability (clear from state diagram and minimum free distance value which is equal to 5). But , the received word is all zero codeword with three errors (1010000010000...) . So , even if we change the decoding window width , the ambiguity in making decision can never be removed or resolved . From the above figure up to decoding window width 15 , there are two survivors , correspondingly two decoded information sequence 0000000... or 1010000... . Increasing the window width , the problem can not be solved as the error has exceeded the error correction capability .

Crossed path means path out of consideration due to nonoptimality - not surviving path . It is previously stated that of the two paths diverging from a node , the upper one is for input 0 and the lower one is for input 1 .

4.6.3.1 : Within correctable range errors can be corrected with choosing appropriate window width :

The vertical and thin dotted line means span (Last boundary is shown, not first boundary)of decoding window width .Thick dotted line means survivor . This convention is used in all the next trellis .

When decoding window width is 5 :

The overlapping double line (dotted and solid) means survivor for the whole decoding window width .

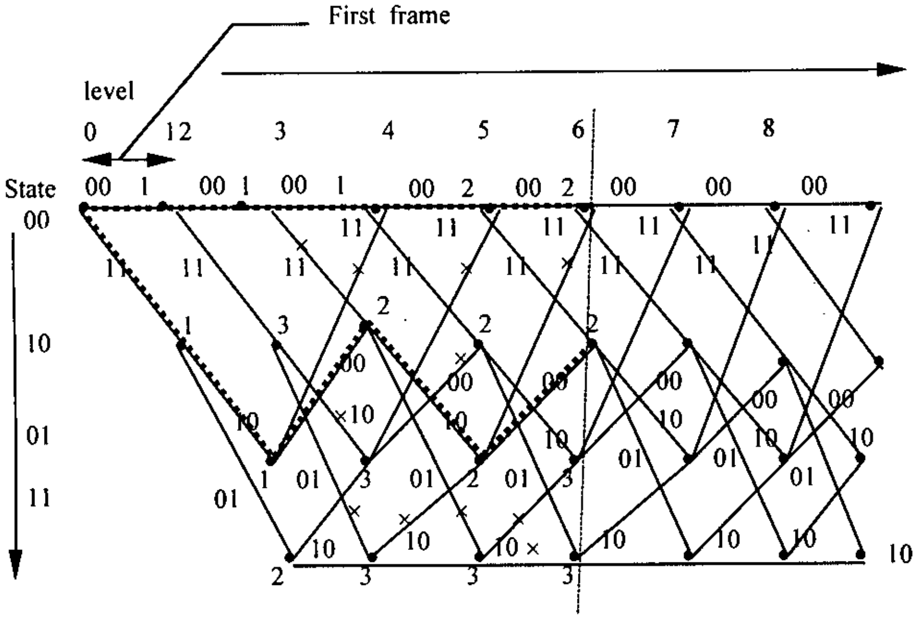
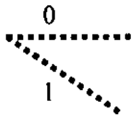


Figure 4.17 : Example with the same encoder having trellis of figure 4.16 (to show the effect of decoding window width on information bit decision).

Let All zero codeword is received as 10 00 00 10 00 00 00 decoding window width is 5 two available paths for the first information bit decision .

Two available paths



If decision is made at this moment , information frame decision in the first frame may be both I and 0 , because two paths have smallest and equal discrepancy 2 up to decoding window width 5 , both of them are survivors .

When decoding window width is 6 :

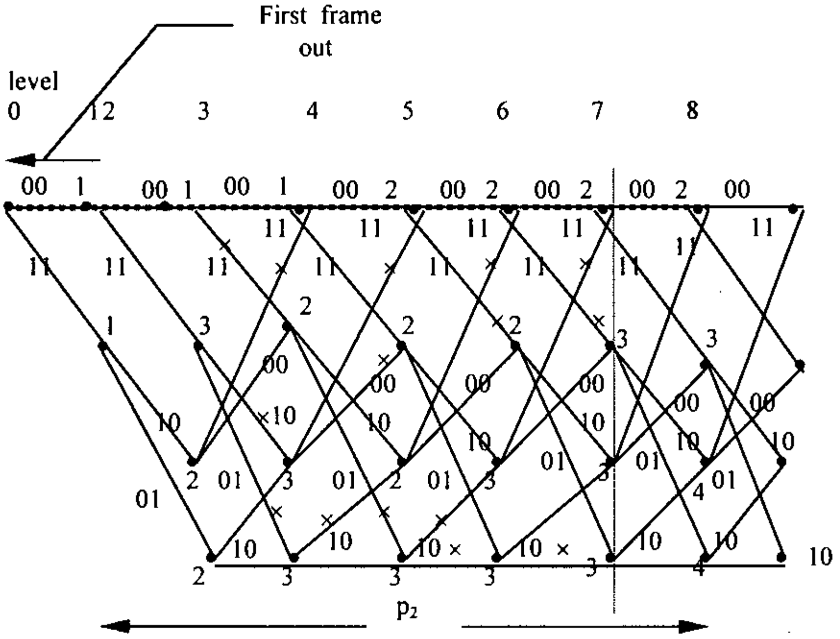


Figure 4.18 : The Trellis of figure 4.16 with decoding window width 6.

The all zero codeword is received as

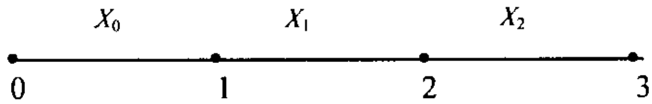
10 00 00 10 00 00 00 00

When decoding window width is 6 , one path has the smallest discrepancy (2) . Taking this path we decide the first information frame is 0 .

$\overset{0}{\overline{\qquad\qquad}}$ single path for first information bit decision .

§ 4.7 : MINIMUM FREE DISTANCE :

4.7.1 : Distance :



Subscript i=2 but number of branch =3

d_i is the minimum hamming distance between two paths i+1 (because X_0 , X_1, ..., X_i) branches in length , which diverge at the root node of the trellis .Because the convolution code is linear , d_i is the minimum hamming weight of encoded path $\mathbf{c}_{[0,i\,]}$ resulting from an information sequence with i_0 (first information bit) ≠ 0 . Let , W_H denotes the hamming weight of a sequence ,

$$\mathbf{c} = \mathbf{i}\,.\,\mathbf{G}$$

$$\therefore d_j = \min_{i_0 \neq 0} W_H(\mathrm{c})$$

$$= \min_{i_0 \neq 0} W_H\left([i_0, i_1, \ldots, i_j] \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & \mathbf{G}_2 & \cdots & \mathbf{G}_j \\ \mathbf{0} & \mathbf{G}_0 & \mathbf{G}_1 & \cdots & \mathbf{G}_{j-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{G}_0 \end{bmatrix} \right) \quad \updownarrow j{+}1 \text{ rows}$$

$$\longleftarrow j{+}1 \text{ columns} \longrightarrow$$

But , for infinite message sequence , this d_j is called d_{free} or d_f (minimum free distance) . And the generator matrix is semi infinite matrix - infinite to the right and below . So the free distance of a convolutional code is defined as the minimum hamming distance between any two words in the code . (codes start and ends at zero state).Where M is the number of encoders memory units.

$$d_{free} = \min_{i_0 \neq 0} W_H\left([i_0, i_1, \ldots] \begin{bmatrix} \mathbf{G}_0 & \mathbf{G}_1 & \cdots & \mathbf{G}_M & & & \\ & \mathbf{G}_0 & \mathbf{G}_1 & \cdots & \mathbf{G}_{M-1} & \mathbf{G}_M & \\ & & & & \mathbf{G}_0 & & \mathbf{G}_M \\ & & & & \cdots & & \\ & & & & \cdots & & \end{bmatrix} \right)$$

A convolutional code with free distance d_{free} can correct t errors if and only if d_{free} is greater than $2t$. Any nonzero code sequence corresponds to a complete path beginning and ending at the all zero state . In decoding we starts from all zero state , because initially there is no discrepancy .And this process ends at the all zero state , because of all zero tail bits . For this return to the all zero state , $d_\infty = d_{free} = \lim_{i \to \infty} d_i$ exists and finite for non catastrophic code . For catastrophic code it may be infinite , as the all zero state may never be reached .

Unlike block coding , the use of non systematic codes is ordinarily preferred over systematic codes in convolutional coding . Because , though from systematic code it is easy to retrieve information frames but it has lower minimum distance than corresponding nonsystematic code - correspondingly poorer error correcting ability .

From state diagram 4.9 (b) d_0=2 , d_1 = 3 , d_2 = 4 , d_3 =4 , d_4=4 , d_5 (a-b-c-a-a-a-a) = 5 ; d_6 (a-b-c-a-a-a-a-a)=5 . By gaining weight 5 by the path of d_5 and then again and again moving along the self loop at the all zero state we can keep the $d_i = 5$ for $i \geq 5$. $\therefore$ $d_{free} = 5$ (2 errors can be corrected , using formula 2.26) . If d_{free} is larger , more errors can be corrected - so better error performance .

We may want to determine the error correcting ability of a code . The larger the hamming distance , the more transmitted bits are in error and less likely that particular event is to be selected in *MLD* .The path with the smallest hamming distance away from the all zero path thus represents the most probable error event.

CHAPTER FIVE : EXPERIMENTAL RESULTS AND RECOMMENDATIONS

§ 5.1 : DESIGN OF EXPERIMENTS:

Studies have so far been carried out on Error Control Codes. A thorough study leads to verification of the concepts learned along with attempts to discover properties. The scope of the current work spans linear block codes, cyclic codes and convolutional codes. The underlying concepts behind their formation and construction, their representation, error-correction capabilities and encoding and decoding algorithms along with some other related topics were covered in previous parts of this thesis. A solid foundation for experimentation has been laid through these studies. Examination through experiments is therefore possible using computer programs. This chapter focuses on the experiments carried out during the current work and their outcomes.

Computer programs have been developed for the purpose of experimentation by taking representatives from different groups and families of codes. The idea is to develop programs which implement the representatives of families of codes and examine their performance parameters of interest in connection with the current work. During the study phase, families of block and cyclic coes were taken into account. Encoding and decoding of these families of codes have been a major consideration of the study. The encoding and decoding of both these codes can be done using generator and parity check matrices. The idea behind this sort of encoding and decoding is to form a codeword of the derived code by multiplying information vector with generator matrix. This information is transmitted through channels or store in database. Upon receiving the codeword or retrieval from database, its correctness is verified by multiplying with parity check matrix. If errors are encountered within correctable range, they are corrected through a process called syndrome decoding. Since both these families follow the same procedure, a code that belongs to both of them, i.e. Hamming Code has been taken as representative for experimentation purpose. Again, there are codes which follow different but specialized procedure in encoding and decoding although normal methods of encoding and decoding equally apply to them. A representative from this sort of code, namely Reed Muller code with Reed algorithm for decoding has been taken as representative.

The programs developed for these codes implement their encoding and decoding and measure the operations and time requirement in doing so. Therefore an analysis of impact of blocklength on encoding and decoding of these codes can be made.

§ 5.2 : EFFECT OF BLOCKLENGTH ON ENCODING :

5.2.1 : Hamming Codes :

Experiments with different values of m yielding different sizes of generator matrices were performed. Each value of m represented a different Hamming Code. The time requirements in encoding for different values of blocklength have been computed. The values are also normalized to represent the time requirement for carrying equal length information bits. All these figure have been shown in table 5.1. The values indicated in columns dealing with time are some multiple of unit time. They are quoted for the purpose of comparison. The reason for multiplication by some suitable factor is to make the data presentable.

From the entries of table 5.1, it is seen that time requirement increases with the increase in blocklength as expected. However, the normalized time column reveals some interesting information

Table 5.1 : Effect of blocklength on Encoding of Hamming Code

M	*Block length n*	*Information length k*	*Time*	*Normalized Time (k=120)*
3	7	4	0.11	3.30
4	15	11	0.60	6.55
5	31	26	2.26	10.43
6	63	57	9.06	19.07
7	127	120	36.74	36.74

that if smaller blocks of codeword with smaller information length or information content are transmitted several times to carry the same amount of information, even then the required time for decoding is less than larger blocklength. It is to be noted here that all values of m as shown in table 5.1 represent different Hamming Codes each with single error correcting capability. Therefore for example if a Hamming Code with m=3 is used instead of m=7, then to carry the same amount of information as in the case with m=7 the m=3 code has to be transmitted 30 times. As such, 30 single errors can be corrected against only one error correcting ability with m=7. Therefore, Hamming Codes with smaller blocklength seem more desirable than longer blocklength in terms of encoding complexity and error correcting capability. Since Hamming Codes have cyclic property, this observation also makes smaller blocklength Hamming Codes excellent candidates for being used in interleaved form. The above mentioned m=3 code can then correct a burst error of length up to 30.

However this overwhelming performance of smaller blocklength codes have some other not so attractive aspects too. Using smaller blocklength codes would cause the total transmitted codeword length to become higher than their larger counterparts if both are meant to carry the same amount of information.

The above statistics can also be presented graphically as in figure 5.1.

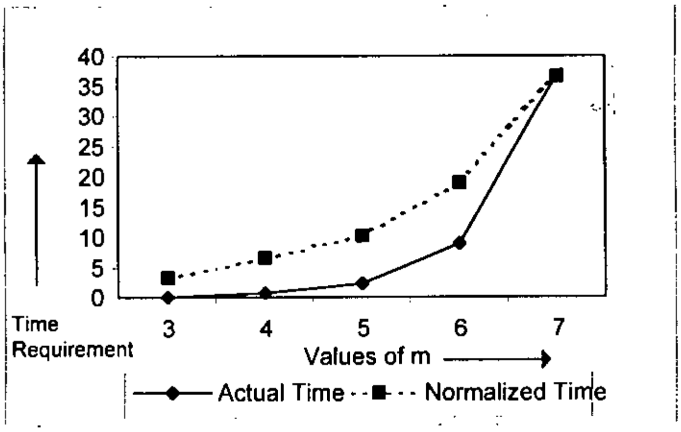


Figure 5.1 : Hamming Code : Effect of blocklength on Encoding

5.2.2 : Reed Muller Codes :

Experiments with different values of m and r($r<m$) yielding different sizes of generator matrices were performed. Each value of m and r represented a different Reed Muller Code. The time requirements in encoding for different values of blocklength have been computed. The values are also normalized to represent the time requirement for carrying equal length information bits. All these figure have been shown in table 5.2. The values indicated in columns dealing with time are some multiple of unit time. They are quoted for the purpose of comparison. The reason for multiplication by some suitable factor is to make the data presentable.

Table 5.2 : Effect of blocklength on Encoding of Reed Muller Code

m	*r*	*Block length, n*	*Information length, k*	*No. of errors correctable, t*	*Actual Time*	*Normalized Time (k=127)*
7	1	128	8	31	0.88	13.79
	2		29	15	3.90	17.08
	3		64	7	10.55	20.94
	4		99	3	19.00	24.37
	5		120	1	25.27	26.74
	6		127	-	27.85	27.85
6	1	64	7	15	0.39	7.08
	2		22	7	1.48	8.54
	3		42	3	3.46	10.46
	4		57	1	5.27	11.74
	5		63	-	6.26	12.62
5	1	32	6	7	0.16	3.39
	2		16	3	0.55	4.37
	3		26	1	1.04	5.08
	4		31	-	1.43	5.86
4	1	16	5	3	0.06	1.52
	2		11	1	0.22	2.54
	3		15	-	0.33	2.79

The figures in table 5.2 show that smaller information length codes take less time in encoding- a natural expectation. Here, however the information length remains unchanged for a particular value of m. Therefore, information content for a particular blocklength varies with the value of r. The number of errors that can be corrected increases with the decrease of r for a particular blocklength (i.e.value of m). In the normalized time column it is seen that if smaller information containing codes are used repeatedly even for a fixed blocklength, the time requirement for encoding to carry as much information as is carried by larger information length codes would be smaller. This justifies using smaller values of k (i.e. smaller values of r) from encoding point of view. Suppose one is to employ encoding for Reed Muller Code using m=7, n=128, k=120 and t=1. One can use a Reed Muller encoding with m=7, n=128, k=8, t=31 fifteen times instead to carry the same amount of information but taking less time to encode. The error correcting ability would then become 31*15 in 15 blocks of 128 bit codeword. This however reveals the fact that channel utilization would then become 15 times higher. Thus, though smaller

Thus, though smaller information length Reed Muller codes are desirable from encoding and error correcting ability point of views, they are less attractive when channel capacity is relatively small.

The statistics presented in table 5.2 have been shown graphically in figure 5.2 and figure 5.3.

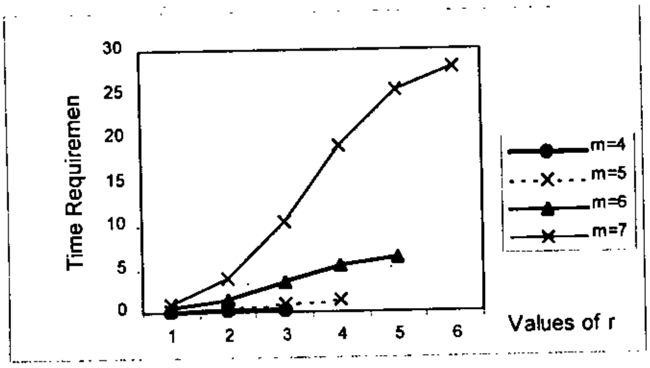


Figure 5.2 : Effect of blocklength on Encoding of Reed Muller Code

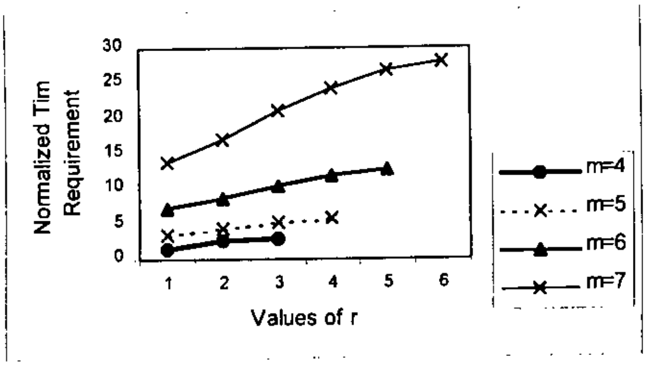


Figure 5.3: Effect of blocklength on encoding of Reed Muller Codes(Normalized Time)

§ 5.3 : EFFECT OF BLOCKLENGTH ON DECODING :

5.3.1 : Hamming Codes :

Experiments with different values of m yielding different sizes of parity check matrices were performed. Each value of m corresponds to a different Hamming Code. The time requirements in decoding for different values of blocklength have been computed. The values are also normalized to represent the time requirement for carrying equal length information bits. All these figures have been shown in table 5.3. The values indicated in columns dealing with time are some multiple of unit time. These values are significant for the purpose of comparison. The reason for multiplication by some suitable factor is to make the data presentable.

Table 5.3 : Effect of blocklength on Decoding of Hamming Code

M	*Block length n*	*Information length k*	*Time*	*Normalized Time (k=120)*
3	7	4	5	150.00
4	15	11	9	98.20
5	31	26	17	78.46
6	63	57	33	69.47
7	127	120	65	65.00

The decoding process involves calculation of the syndrome for the received information vector, then comparison of the syndrome with the syndrome table is made and upon finding the syndrome in the syndrome table, the information vector is found. Therefore the process does not involve equivalent amount of operations all the times. Rather, the syndrome finding process is probabilistic. Hence the average amount of operations or time requirement has taken into account for the calculations in table 5.3. Here it is observed that decoding in case of repeated sequence of smaller blocklength corresponding to information length equal to higher blocklength require more operations than decoding in case of higher blocklength. Figure 5.4 depicts the scenario.

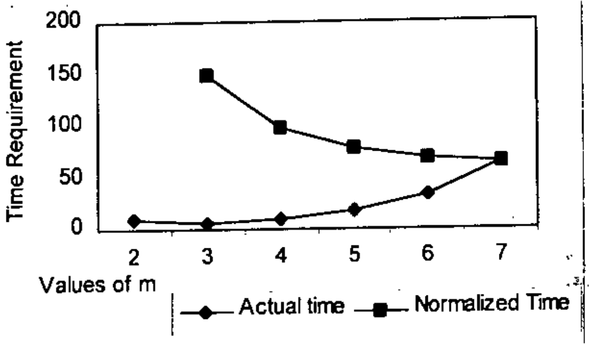


Figure 5.4 : Effect of blocklength on Decoding of Hamming Code

The decoding normalized curve is seen to decrease as blocklength increases. That is, the decoding time requirement decreases with higher blocklength. However the merits of smaller block length as observed in case of encoding should not be overcast by this apparent decoding demerit since smaller blocklength Hamming Codes provide higher numbers of error correcting capability and if interleaved, can correct burst errors. Moreover, if decoding involves only syndrome calculation without search into the syndrome table than the smaller blocklength codes outperform their longer blocklength counterparts in terms of statistics presented in table 5.3.

5.3.2 : Reed Muller Codes :

Experiments with different values of m and r yielding different sizes and numbers of checksums were performed. Each value of m corresponds to a different Reed Muller Code. The time requirements in decoding for different values of blocklength or information length have been computed. The values are also normalized to represent the time requirement for carrying equal length information bits. All these figures have been shown in table 5.4. The values indicated in columns dealing with time are some multiple of unit time. These values are significant for the purpose of comparison. The reason for multiplication by some suitable factor is to make the data presentable.

Table 5.4 : Effect of blocklength on Encoding of Reed Muller Code

M	*r*	*Block length, n*	*Information length, k*	*No. of errors correctable, t*	*Actual Time*	*Normalized Time (k=120)*
7	1	128	8	31	384	5760
	2		29	15	1024	4237
	3		64	7	1568	2940
	4		99	3	1840	2230
	5		120	1	1920	1920
	6		127	-	-	-
6	1	64	7	15	160	2743
	2		22	7	384	2095
	3		42	3	536	1531
	4		57	1	592	1248
	5		63	-	-	-
5	1	32	6	7	64	1280
	2		16	3	136	1020
	3		26	1	172	794
	4		31	-	-	-
4	1	16	5	3	24	576
	2		11	1	44	480
	3		15	-	-	-

The decoding operation has been performed using Reed algorithm. The Reed Muller code can be decoded using normal syndrome decoding but the specialized Reed algorithm works good here. The points of interest is that here also smaller blocklength codes have better performance in terms of operational or timing requirements to correspond to same amount of information. And as usual the no of errors that can be corrected would be higher in case of using repeated smaller blocklength codes instead of their larger blocklength counterparts. In fact, using Reed algorithm for decoding involves fixed time requirement for a particular value of m and r. Here the majority resolver has been assumed to require equal amount of time for each case and the decoding sequences have been assumed to be available previously. A point to note is that whenever r =m-1, no errors can be corrected. Therfore, the corresponding entries have been left blank in table 5.4. The above facts are also represented in figure 5.5 and figure 5.6.

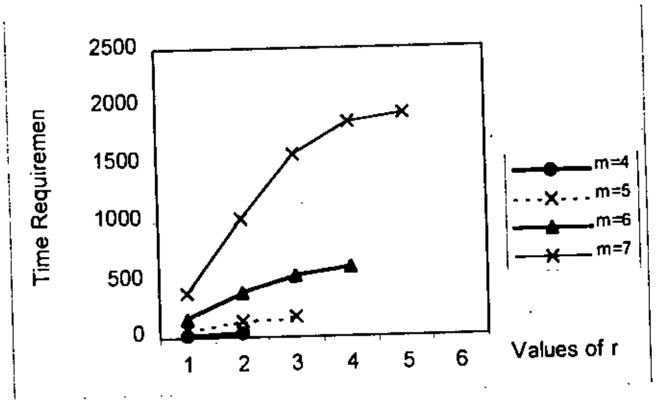


Figure 5.5 : Effect of blocklength on Decoding of Reed Muller Code

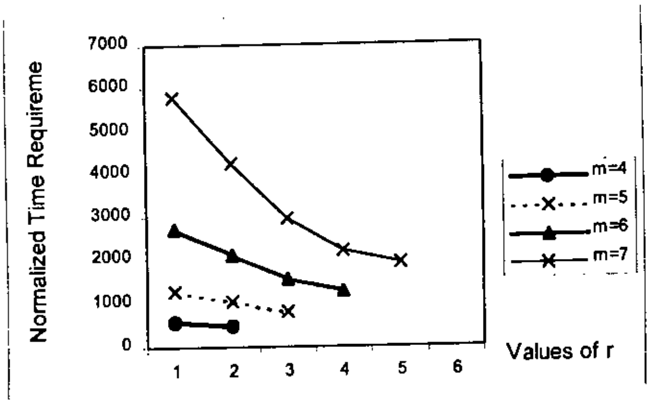


Figure 5.6: Effect of blocklength on Decoding of Reed Muller Codes(Normalized Time)

§ 5.4 : RECOMMENDATIONS FOR FUTURE WORK IN RELATED FIELDS:

The current work has dealt with three families of Error Control Codes and examined the effect of blocklength on encoding and decoding of these codes. Further works in related fields may examine the behavior of these codes in context of different transmission channels. In case of cyclic codes interleaving is an interesting property to work with. Therefore, effect of interleaving in different cyclic codes can be analyzed using computer programs. Performance analysis of Viterbi decoding in different scenario may also be an interesting area to deal with. Lastly, similar works in case of families of codes not covered by current work may also be carried out.

BIBLIOGRAPHY

1. Blahut, R. E., *Theory and Practice of Error Control Codes*, Addison Wesley Publishing Company, 1983 .
2. Haykin, S., *Communication Systems*, John Wiley & Sons, 1988.
3. Michelson A.M. and A. H. Levesque, *Error-control Technique for Digital Communication*. John Wiley & Sons, 1985.
4. Schwartz, M., *Information, Transmission, Modulation and Noise*, John Wiley & Sons(Fourth edition).
5. Mc Williams, F. J. and N. J. Sloane, The theory of Error Correcting Codes, North-Holland. New York,1977.
6. Lucky, R.W., J. Salz, and E. J. Weldon, *Principles of Data Communication*, McGraw Hill, NewYork, 1968.
7. Gallager, R.G., *Information Theory And Reliable Communication*, McGraw Hill, NewYork, 1968.
8. Lin, S., *Introduction to Error Correction Codes*, Prentice Hall, Englewood Cliffs, N.J., 1970.
9. Berlekamp, E. R., *Algebraic Coding Theory*, McGraw Hill, NewYork, 1968.
10. Fossorier, M. P., and S. Lin, *Soft Decision Decoding of Lonear Block Codes Based on Ordered Statistics*, IEEE Transactions on Information Theory, September, 1995.
11. Caire, G., and S. Lin, *Linear Block Codes over Cyclic Groups*, IEEE Transactions on Information Theory, September, 1995.