Linear Block Codes for Error Correction

3

Mathematics is an interesting intellectual sport but it should not be allowed to stand in the way of obtaining sensible information about physical processes.

Richard W Hamming

3.1 INTRODUCTION TO ERROR CORRECTING CODES

In this *age of information*, there is increasing importance not only of speed, but also of accuracy in the storage, retrieval, and transmission of data. The channels over which messages are transmitted are often imperfect. Machines do make errors, and their non-man-made mistakes can turn otherwise flawless programming into a worthless, or even dangerous, trash. Just as architects design buildings that will remain standing through even an earthquake, their computer counterparts have come up with sophisticated techniques capable of counteracting the digital manifestations of the Murphy's Law ("If anything can go wrong, it will go wrong"). **Error correcting codes** are a kind of safety net—the mathematical insurance against the vagaries of an imperfect material world.

Error correcting codes as the name suggests, are used for correcting errors when messages are transmitted over a noisy channel or when stored data is retrieved. The physical medium through which the messages are transmitted is called a channel (e.g. a telephone line, a satellite link, a wireless channel used for mobile communications etc.). Different kinds of channels are

Scanned by CamScanner

prone to different kinds of noise, which corrupt the data being transmitted. The noise could be caused by lightning, due to human errors, equipment malfunctioning, a voltage surge, the random motion of electrons in the receiver hardware etc. Because these error correcting codes try to overcome the detrimental effects of noise in the channel, the encoding procedure is also called *channel coding*. Error control codes are also used for accurate storage of information, for example storing data and reading it from a compact disc (CD). In this case the error could be due to a scratch on the surface of the CD. The error correcting coding scheme will try to recover the original data from the corrupted one. The motivation for the study of error control coding comes from the fact that nowadays a large volume of data is being communicated daily and there exists an ever increasing need to communicate, store and retrieve data with greater reliability.

The basic idea behind error correcting codes is to add a certain amount of redundancy to the message prior to its transmission through the noisy channel. This redundancy, which is basically some extra symbols, is added in a *known* manner. The encoded message when transmitted through the channel might get corrupted due to the noise in the channel. At the receiver, the original message can be recovered from the corrupted one if the number of errors is within the limit for which the coding strategy has been designed. By adding redundancy intelligently we can succeed in fooling the random noise to some extent and diluting its effect. The block diagram of a digital communication system is illustrated in Fig. 3.1. Note, that the most important block in the figure is that of noise, without which there will be no need for the channel encoder.

Example 3.1 Let us see how redundancy can help combat the effects of noise. The normal language that we use to communicate (say, English) has a lot of redundancy built in it already. Consider the following sentence (which may be thought of as being corrupted by noise – maybe the moths have eaten portions of a written text):

CODING THEORY IS AN INTRSTNG SUBJCT.

As we can see, there are a lot of errors in this sentence. However, due to the familiarity with the language we may probably guess that the original text would have looked something like:

CODING THEORY IS AN INTERESTING SUBJECT.

We have just now used an error correcting strategy that makes use of the inbuilt redundancy in the English language to construct the original message from the corrupted one.

The objectives of a good error control coding scheme are

(i) error correcting capability in terms of the number of errors that it can rectify,
(ii) fast encoding of the message, i.e., an efficient encoding strategy,
(iii) fast and efficient decoding of the received message,
(iv) maximum transfer of information per unit time (i.e., fewer overheads in terms of the redundancy).

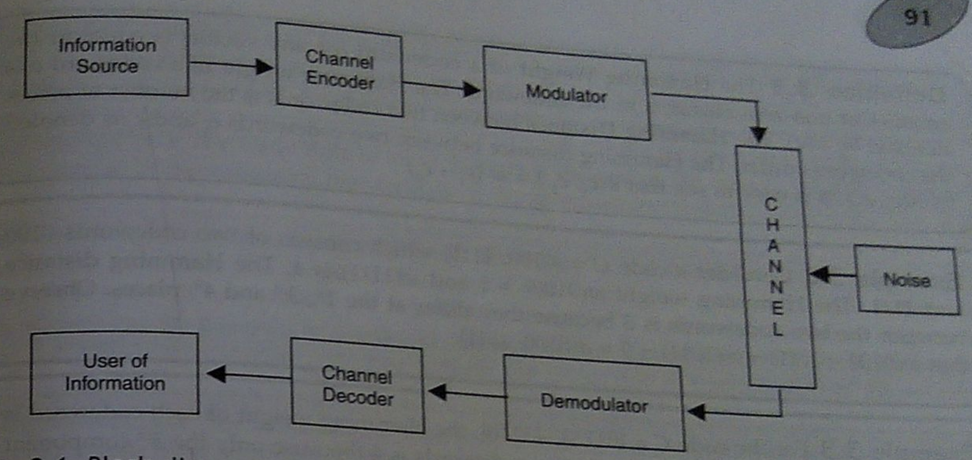


Fig. 3.1 Block diagram (and the principle) of a digital communication system. Here the source coder/decoder block has not been shown.

The first objective is the primary one. In order to increase the error correcting capability of a coding scheme one must introduce more redundancies. However, increased redundancy leads to a slower rate of transfer of the actual information. Thus the objectives (i) and (iv) are not necessarily compatible. Also, as the coding strategies become more complicated for correcting larger number of errors, the objectives (ii) and (iii) also become difficult to achieve.

In this chapter, we shall first learn about the basic definitions of error control coding. These definitions, as we shall see, would be used throughout this book. The concept of **Linear Block Codes** will then be introduced. Linear Block Codes form a very large class of useful codes. We will see that it is very easy to work with the matrix description of these codes. In the later part of this chapter, we will learn how to efficiently decode these Linear Block Codes. We will also study **Hamming Codes**, **Low Density Parity Check Codes** and **Space Time Block codes**. The notion of **perfect codes** and **optimal linear codes** will also be introduced.

3.2 BASIC DEFINITIONS

In this section we shall state some basic definitions, which will be frequently used here as well as in the later chapters.

Definition 3.1 A **Word** is a sequence of symbols.

Definition 3.2 A **Code** is a set of vectors called **codewords**.

Scanned by CamScanner

Definition 3.3 The **Hamming Weight** of a codeword (or any vector) is equal to the number of non-zero elements in the codeword. The Hamming weight of a codeword c is denoted by $w(c)$. The **Hamming Distance** between two codewords is the number of places the codewords differ. The Hamming distance between two codewords c_1 and c_2 is denoted by $d(c_1, c_2)$. It is easy to see that $d(c_1, c_2) = w(c_1 - c_2)$.

Example 3.2 Consider a code $C = \{0100, 1111\}$ which consists of two codewords 0100 and 1111. The Hamming weight $w(0100) = 1$ and $w(1111)$ is 4. The Hamming distance between the two codewords is 3 because they differ at the 1st, 3rd and 4th places. Observe that $w(0100 - 1111) = w(1011) = 3 = d(0100, 1111)$.

Example 3.3 For the code $C = \{01234, 43210\}$, the Hamming weight of each codeword is 4 and the Hamming distance between the codewords is 4 (because only the 3rd component of the two codewords are identical while they differ at 4 places).

Definition 3.4 A **Block Code** consists of a set of fixed length codewords. The fixed length of these codewords is called the **Block Length** and is typically denoted by n. Thus, a code of blocklength n consists of a set of codewords having n **components**.

A block code of size M defined over an alphabet with q symbols is a set of M q-ary sequences, each of length n. In the special case that $q = 2$, the symbols are called bits (short for binary digits) and the code is said to be a binary code. Usually, $M = q^k$ for some integer k, and we call such a code an (n, k) code.

Example 3.4 The code $C = \{00000, 10100, 11110, 11001\}$ is a block code of block length equal to 5. This code can be used to represent two bit binary numbers as follows:

Uncoded bits	*Codewords*
00	00000
01	10100
10	11110
11	11001

Here, $M = 4$, $k = 2$ and $n = 5$. Suppose we have to transmit a sequence of 1's and 0's using the above coding scheme. Lets say that the sequence to be encoded is 1 0 0 1 0 1 0 0 1 1 ...

The first step is to break the sequence in groups of two bits (because we want to encode two bits at a time). So we partition the sequence as follows:

10 01 01 00 11 ...

Next, replace each block by its corresponding codeword.

11110 10100 10100 00000 11001 ...

Thus 5 bits (coded) are sent for every 2 bits of uncoded message. It should be observed that for every 2 bits of information we are sending 3 extra bits (redundancy).

Definition 3.5 The **Code Rate** of an (n, k) code is defined as the ratio (k/n), and reflects the fraction of the codeword that consists of the information symbols.

Code rate is always less than unity (code rate equal to one implies no coding at all). The smaller the code rate, the greater is the redundancy within the code, i.e., more number of redundant symbols are present *per information symbol* in a codeword. A code with greater redundancy has the potential to detect and correct more number of symbols in error, as we shall soon see. However, a smaller code rate reduces the actual rate of transmission of information.

Definition 3.6 The **Minimum Distance** of a code is the minimum Hamming distance between any two codewords. If the code C consists of the set of codewords $\{c_i, i = 0, 1, ..., M\text{-}1\}$ then the minimum distance of the code is given by $d^* = \min d(c_i, c_j)$, $i \neq j$. An (n, k) code with minimum distance d^* is sometimes denoted by (n, k, d^*).

Definition 3.7 The **Minimum Weight** of a code is the smallest weight of any non-zero codeword, and is denoted by w^*.

Definition 3.8 A **Linear Code** has the following properties:

(i) The sum of two codewords belonging to the code is also a codeword belonging to the code.

(ii) The all-zero codeword is always a codeword.

(iii) The minimum Hamming distance between two codewords of a linear code is equal to the minimum weight of any non-zero codeword, i.e., $d^* = w^*$.

Note that if the sum of two codewords is another codeword, the difference of two codewords will also yield a valid codeword. For example, if c_1, c_2 and c_3 are valid codewords such that $c_1 + c_2 = c_3$ then $c_3 - c_1 = c_2$. Hence it is obvious that the all-zero codeword must always be a valid codeword for a linear block code (self-subtraction of a codeword).

Theorem 3.1 For a linear code the minimum distance is equal to the minimum weight of the code, i.e., $d^* = w^*$.

Intuitive proof: The distance d_{ij} between any two codewords c_i and c_j is simply the weight of the codeword formed by $c_i - c_j$. Since the code is linear, the difference of two codewords results in another valid codeword. Thus, the minimum weight of a non-zero codeword will reflect the minimum distance of the code.

Scanned by CamScanner

Example 3.5 The code $C = \{0000, 1010, 0101, 1111\}$ is a linear block code of block length $n = 4$. Observe that all the ten possible sums of the codewords:

$0000 + 0000 = 0000$, $0000 + 1010 = 1010$, $0000 + 0101 = 0101$,
$0000 + 1111 = 1111$, $1010 + 1010 = 0000$, $1010 + 0101 = 1111$,
$1010 + 1111 = 0101$, $0101 + 0101 = 0000$, $0101 + 1111 = 1010$ and
$1111 + 1111 = 0000$

are in C and the all-zero codeword is in C. The minimum distance of this code is $d^* = 2$. In order to verify the minimum distance of this linear w code we can determine the distance between all pairs of codewords (which is $\binom{4}{2}=6$ in number):

$d(0000, 1010) = 2$, $d(0000, 0101) = 2$, $d(0000, 1111) = 4$,
$d(1010, 0101) = 4$, $d(1010, 1111) = 2$, $d(0101, 1111) = 2$.

We observe that the minimum distance of this code is indeed 2.

Note that the code given in Example 3.4 is not linear because $1010 + 1111 = 0101$, which is not a valid codeword. Even though the all-zero word is a valid codeword, it does not guarantee linearity. *The presence of an all-zero codeword is thus a necessary but not a sufficient condition for linearity.*

In order to make the error correcting codes easier to use, understand and analyse, it is helpful to impose some basic algebraic structure on them. As we shall soon see, it will be useful to have an alphabet in which it is easy to carry out the basic mathematical operations such as add, subtract, multiply and divide.

Definition 3.9 A field F is a set of elements with two operations + (addition) and . (multiplication) satisfying the following properties:

(i) F is closed under + and . *i.e.*, $a + b$ and $a.b$ are in F if a and b are in F.

For all a, b and c in F, the following hold:

(ii) Commutative laws: $a + b = b + a$, $a.b = b.a$

(iii) Associative laws: $(a + b) + c = a + (b + c)$, $a.(b.c) = (a.b).c$

(iv) Distributive law: $a.(b+c) = a.b + a.c$

Further, identity elements 0 and 1 must exist in F satisfying:

(v) $a + 0 = a$

(vi) $a.1 = a$

(vii) For any a in F, there exists an additive inverse $(-a)$ such that $a + (-a) = 0$.

(viii) For any a in F other than 0, there exists an multiplicative inverse (a^{-1}) such that $a.(a^{-1})=1$.

The above properties are true for fields with both finite as well as infinite elements. A field with a finite number of elements (say, q) is called a **Galois Field** (pronounced Gal-oa Field) and is denoted by $GF(q)$. If only the first seven properties are satisfied, then it is called a ring.

Example 3.6 Consider $GF(4)$ with 4 elements $\{0, 1, 2, 3\}$. The addition and multiplication tables for $GF(4)$ are

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

.	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

It should be noted here that the addition in $GF(4)$ is not modulo 4 addition.

Let us define a vector space, $GF(q^n)$, which is a set of n-tuples of elements from $GF(q)$. Linear block codes can be looked upon as a set of n-tuples (vectors of length n) over $GF(q)$ such that the sum of two codewords is also a codeword, and the product of any codeword by a field element is a codeword. Thus, a linear block code is a subspace of $GF(q^n)$.

Let S be a set of vectors of length n whose components are defined over $GF(q)$. The set of all linear combinations of the vectors of S is called the linear span of S and is denoted by $<S>$. The linear span is thus a subspace of $GF(q^n)$, generated by S. Given any subset S of $GF(q^n)$, it is possible to obtain a linear code $C = <S>$ generated by S, consisting of precisely the following codewords:

(i) all-zero word,
(ii) all words in S,
(iii) all linear combinations of two or more words in S.

Example 3.7 Let $S = \{1100, 0100, 0011\}$. All possible linear combinations of S are:

$1100 + 0100 = 1000$, $1100 + 0011 = 1111$, $0100 + 0011 = 0111$, $1100 + 0100 + 0011 = 1011$.

Therefore, $C = <S> = \{0000, 1100, 0100, 0011, 1000, 1111, 0111, 1011\}$. The minimum distance of this code is $w(0100) = 1$.

Example 3.8 Let $S = \{12, 21\}$ defined over $GF(3)$. The addition and multiplication tables of field $GF(3) = \{0, 1, 2\}$ are given by:

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

.	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

All possible linear combinations of 12 and 21 are:

$12 + 21 = 00$, $12 + 2(21) = 21$, $2(12) + 21 = 12$.

Therefore, $C = <S> = \{00, 12, 21, 00, 21, 12\} = \{00, 12, 21\}$.

Scanned by CamScanner

3.3 MATRIX DESCRIPTION OF LINEAR BLOCK CODES

As we have observed earlier, any code C is a subspace of $GF(q^n)$. Any set of basis vectors can be used to generate the code space. We can, therefore, define a generator matrix, G, the rows of which form the basis vectors of the subspace. The rows of G will be linearly independent. Thus a linear combination of the rows can be used to generate the codewords of C. The generator matrix will be a $k \times n$ matrix with rank k. Since the choice of the basis vectors is not unique, the *generator matrix is not unique for a given linear code.*

The generator matrix converts (encodes) a vector of length k to a vector of length n. Let the input vector (uncoded symbols) be represented by i. The coded symbols will be given by

$$c = iG, \tag{3.1}$$

where c is called the codeword and i is called the information word.

The generator matrix provides a concise and efficient way of representing a linear block code. The $n \times k$ matrix can generate q^k codewords. Thus, instead of having a large look-up table of q^k codewords, one can simply have a generator matrix. This provides an enormous saving in storage space for large codes. For example, for the binary (46, 24) code the total number of codewords are $2^{24} = 1{,}777{,}216$ and the size of the lookup table of codewords will be $n \times 2^k =$ 771,751,936 bits. On the other hand if we use a generator matrix, the total storage requirement would be $n \times k = 46 \times 24 = 1104$ bits.

Example 3.9 Consider a generator matrix

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

$$c_1 = [00]\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = [000], \qquad c_2 = [01]\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = [010]$$

$$c_3 = [10]\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = [101], \qquad c_4 = [11]\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} = [111]$$

Therefore, this generator matrix generates the code $C = \{000, 010, 101, 111\}$. We observe that this is a (3, 2) code.

3.4 EQUIVALENT CODES

Definition 3.10 A permutation of a set $S = \{x_1, x_2, \ldots, x_n\}$ is a one-to-one mapping from S to itself. A permutation can be denoted as follows:

$$\begin{array}{cccc} x_1 & x_2 & \ldots\ldots & x_n \\ \downarrow & \downarrow & \ldots\ldots & \downarrow \\ f(x_1) & f(x_2) & \ldots\ldots & f(x_n) \end{array} \tag{3.2}$$

Definition 3.11 Two q-ary codes are called **equivalent** if one can be obtained from the other by one or both operations listed below:

(i) permutation of the components

(ii) permutation of the position of the codeword.

Suppose a code containing M codewords are displayed in the form of an $M \times n$ matrix, where the rows represent the codewords. The operation (i) corresponds to the re-labeling of the symbols appearing in a given column, and the operation (ii) represents the rearrangements of the colums of the matrix.

Example 3.10 Consider the ternary code (a code whose components $\in \{0, 1, 2\}$) of blocklength 3

$$C = \begin{Bmatrix} 2 & 0 & 1 \\ 1 & 2 & 0 \\ 0 & 1 & 2 \end{Bmatrix}$$

If we apply the permutation $0 \to 2$, $2 \to 1$, $1 \to 0$ to column 2 and $1 \to 2$, $0 \to 1$, $2 \to 0$ to column 3 we obtain

$$C_1 = \begin{Bmatrix} 2 & 2 & 2 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{Bmatrix}$$

The code C_1 is equivalent to a repetition of code length 3. *Note that the original code is not linear, but is equivalent to a linear code.*

Definition 3.12 Two linear q-ary codes are called **equivalent** if one can be obtained from the other by one or both of the operations listed below:

(i) multiplication of the components by a non-zero scalar

(ii) permutation of the position of the codeword.

Note that in definition 3.11 we have defined equivalent codes that are not necessarily linear.

Theorem 3.2 Two $k \times n$ matrices generate equivalent linear (n, k) codes over $GF(q)$ if one matrix can be obtained from the other by a sequence of the following operations:

(i) Permutation of rows,

(ii) Multiplication of a row by a non-scalar,

(iii) Addition of a scalar multiple of one row to another,

(iv) Permutation of columns,

(v) Multiplication of any column by a non-zero scalar.

Scanned by CamScanner

Proof The first three operations (which are just row operations) preserve the linear independence of the rows of the generator matrix. The operations merely modify the basis. The last two operations (which are column operations) convert the matrix to one which will produce an equivalent code.

Theorem 3.3 A generator matrix can be reduced to its **systematic form** (also called the standard form of the generator matrix) of the type $G = [\, I \mid P]$ where I is a $k \times k$ identity matrix and P is a $k \times (n-k)$ matrix.

Proof The k rows of any generator matrix (of size $k \times n$) are linearly independent. Hence, by performing elementary row operations and column permutations it is possible to obtain an equivalent generator matrix in a row echelon form. This matrix will be of the form $[I \mid P]$.

Example 3.11 Consider the generator matrix of a (4, 3) code over $GF(3)$:

$$G = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 2 & 2 & 1 \end{bmatrix}.$$

Let us represent the ith row by r_i and the jth column by c_j. Upon replacing r_3 by $r_3 - r_1 - r_2$ we get (note that in $GF(3)$, $-1 = 2$ and $-2 = 1$ because $1 + 2 = 0$.

$$G = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 2 & 0 \end{bmatrix}.$$

Next we replace r_1 by $r_1 - r_3$ to obtain

$$G = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 2 & 0 \end{bmatrix}.$$

Finally, shifting $c_4 \to c_1$, $c_1 \to c_2$, $c_2 \to c_3$ and $c_3 \to c_4$ we obtain the standard form of the generator matrix

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 \end{bmatrix} = [I \mid P].$$

3.5 PARITY CHECK MATRIX

One of the objectives of a good code design is to have fast and efficient encoding and decoding methodologies. So far we have dealt with the efficient generation of linear block codes using a generator matrix. Codewords are obtained simply by multiplying the input vector (uncoded word) by the generator matrix. Is it possible to detect a valid codeword using a similar concept? The answer is yes, and such a matrix is called the **Parity Check Matrix**, H, for the given code. For a parity check matrix,

$$cH^T = 0, \tag{3.3}$$

where c is a valid codeword. Since $c = iG$, therefore, $iGH^T = 0$. For this to hold true for all valid codewords, we must have

$$GH^T = 0. \tag{3.4}$$

The size of the parity check matrix is $(n-k) \times n$. A parity check matrix provides a simple method of *detecting* whether an error has occurred or not. If the multiplication of the received word (at the receiver) with the transpose of H yields a non-zero vector, it implies that an error has occurred, This methodology, however will fail if the errors in the transmitted codeword exceeds the number of error for which the coding scheme is designed. We shall soon find out that the non-zero product of cH^T might help us not only to detect but also to correct the errors under some conditions.

Suppose the generator matrix is represented in its systematic form $G = [\, I \mid P]$. The matrix P is called the **Coefficient Matrix**. Then the parity check matrix will be defined as

$$H = [\, -P^T \mid I \,], \tag{3.5}$$

where P^T represents the transpose of matrix P. This is because

$$GH^T = [I \mid P] \begin{bmatrix} -P \\ I \end{bmatrix} = 0. \tag{3.6}$$

Since the choice of a generator matrix is not unique for a code, *the parity check matrix will not be unique*. Given a generator matrix G, we can determine the corresponding parity check matrix and *vice-versa*. Thus the parity check matrix H can be used to specify the code completely.

From (3.3) we observe that the vector c must have 1's at such positions that the corresponding rows of H^T add up to the zero vector, **0**. Now, we know that the number of 1's in a codeword pertains to its Hamming weight. *Hence, the minimum distance d^* of a linear block code is given by the minimum number of rows of H^T (or, the columns of H) whose sum is equal to the zero vector.* This is true for binary linear block codes. For the non-binary case, the columns of H, weighted by the codewords have to be considered.

Scanned by CamScanner

Example 3.12 For a (7, 4) linear block code the generator matrix is given by

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

the matrix P is given by

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

and P^T is given by

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

Observing the fact that $-1 = 1$ for the case of binary, we can write the parity check matrix

$$H = [-P^T \mid I]$$

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

Note that the columns 1, 5 and 7 of the parity check matrix, H, add up to the zero vector. Hence, for this code, $d^* = 3$.

Theorem 3.4 The code C contains a non-zero codeword of Hamming weight w or less if and only if a linearly dependent set of w columns of H exist.

Proof Consider a codeword $c \in C$. Let the weight of c be w which implies that there are w non-zero components and $(n - w)$ zero components in c. If we throw away the w zero components then from the relation $CH^T = 0$ we can conclude that w columns of H are linearly dependent.

Conversely, if H has w linearly dependent columns, then a linear combination of at most w columns is zero. These w non-zero coefficients would define a codeword of weight w or less that satisfies $CH^T = 0$.

Definition 3.13 An (n, k) **Systematic Code** is one in which the first k symbols of the codeword of block length n are the information symbols themselves (i.e., the uncoded vector). The remainder of the $(n - k)$ symbols are called the **parity symbols**.

Example 3.13 The following is a (5, 2) systematic code over $GF(3)$.

S.N.	*Information symbols* ($k = 2$)	*Codewords* ($n = 5$)
1.	00	00 000
2.	01	01 121
3.	02	02 220
4.	10	10 012
5.	11	11 221
6.	12	12 210
7.	20	20 020
8.	21	21 100
9.	22	22 212

Note that the total number of codewords is $3^k = 3^2 = 9$. Each codeword begins with the information symbols, and has three parity symbols at the end. The parity symbols for the information word 01 are 121 in the above table. A generator matrix in the systematic form (standard form) will generate a systematic code.

Theorem 3.5 The minimum distance (minimum weight) of (n, k) linear code is bounded as follows:

$$d^* \le n - k + 1 \tag{3.7}$$

This is known as the **Singleton Bound**. This holds for both binary and non-binary linear block codes.

Proof We can reduce all linear block codes to their equivalent systematic forms. A systematic code can have one information symbol and $(n - k)$ parity symbols. At most all the parity symbols can be non-zero, resulting in the total weight of the codeword to be $(n - k) + 1$. Thus the weight of no codeword can exceed $n - k + 1$. Hence, we have the following definition of a maximum distance code.

Definition 3.14 A **Maximum Distance Code** satisfies $d^* = n - k + 1$.

Having familiarised ourselves with the concept of minimum distance of a linear code, we shall now explore how this minimum distance is related to the total number of errors the code can detect and possibly correct. So we move over to the receiver end and take a look at the methods of decoding a linear block code.

Scanned by CamScanner

3.6 DECODING OF A LINEAR BLOCK CODE

The basic objective of channel coding is to detect and correct errors when messages are transmitted over a noisy channel. The noise in the channel randomly transforms some of the symbols of the transmitted codeword into some other symbols. If the noise, for example, changes just one of the symbols in the transmitted codeword, the erroneous codeword will be at a Hamming distance of one from the original codeword. If the noise transforms t symbols (that is, t symbols in the codeword are in error), the Hamming distance of the received word will be at a Hamming distance of t from the originally transmitted codeword. Given a code how many errors can it detect and how many can it correct? Let us first look at the detection problem.

An error will be detected as long as, due to noise, one codeword does not transform into another valid codeword. If the minimum distance between the codewords is d^*, the weight of the error pattern must be d^* or more to cause a transformation of one codeword to another. Therefore, an (n, k, d^*) code will detect at least all non-zero error patterns of weight less than or equal to $(d^* - 1)$. Moreover, there is at least one error pattern of weight d^* which will not be detected. This corresponds to the two codewords that are the closest. It may be possible that some error patterns of weight d^* or more are detected, but *all* error patterns of weight d^* will not be detected.

Example 3.14 For the code $C_1 = \{000, 111\}$ the minimum distance is 3. Therefore error patterns of weight 2 or 1 can be detected. This means that any error pattern belonging to the set {011, 101, 110, 001, 010, 100} will be detected by this code.

Next consider the code $C_2 = \{001, 110, 101\}$ with $d^* = 1$. Nothing can be said regarding how many errors this code can detect because $d^* - 1 = 0$. However, the error pattern 010 of weight 1 can be detected by this code. But it cannot detect all error patterns with weight one, e.g., the error vector 100 cannot be detected.

Next let us look at the problem of error correction. The objective is to make the best possible guess regarding the originally transmitted codeword after observing the received word. What could be a smart decoding strategy? Since only one of the valid codewords must have been transmitted it is logical to conclude that a valid codeword nearest (in terms of Hamming distance) to the received word must have been actually transmitted. In other words, the codeword which resembles the received word most is assumed to be the one that was sent. This strategy is called the **Nearest Neighbour decoding**, as we are picking the codeword nearest to the received word in terms of the Hamming distance.

It may be possible that more than one codeword is at the same Hamming distance from the received word. In that case the receiver can do one of the following:

(i) It can pick one of the equally distant neighbours randomly, or

(ii) Request the transmitter to re-transmit.

To ensure that the received word (that has at most t errors) is closest to the original codeword, and farther from all other codewords, we must put the following condition on the minimum distance of the code

$$d^* \geq 2t + 1. \tag{3.8}$$

Graphically the condition for correcting t errors or less can be visualised from Fig. 3.2. Consider the space of all q-ary n-tuples. Every q-ary vector of length n can be represented as a point in this space. Every codeword can thus be depicted as a point in this space, and all words at a Hamming distance of t or less would lie within the sphere centred at the codeword and with a radius of t. If the minimum distance of the code is d^*, and the condition $d^* \geq 2t + 1$ holds good, then none of these spheres would intersect. Any received vector (which is just a point) within a specific sphere will be closest to its centre (which represents a codeword) than any other codeword. We will call the spheres associated with each codeword its **Decoding Sphere**. Hence it is possible to decode the received vector using the 'nearest neighbour' method without ambiguity.

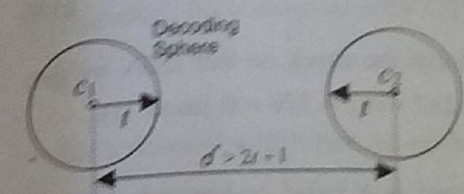


Fig. 3.2

Words within the sphere of radius t and centred at c_i will be decoded as c_i. For unambiguous decoding $d^* \geq 2t + 1$.

The condition $d^* \geq 2t + 1$ takes care of the worst case scenario. It may be possible, however, that the above condition is not met but still it is feasible to correct t errors as is illustrated in the following example.

Example 3.15 Consider the code $C = \{00000, 01010, 10101, 11111\}$. The minimum distance $d^* = 2$. Suppose the codeword 11111 was transmitted and the received word is 11110, i.e., $t = 1$ (one error has occured in the fifth component). Now,

$$d(11110, 00000) = 4,\ d(11110, 01010) = 2,$$
$$d(11110, 10101) = 3,\ d(11110, 11111) = 1.$$

Using the nearest neighbour decoding we can conclude that 11111 was transmitted. Even though a single error correction ($t = 1$) was done in this case, $d^* < 2t + 1 = 3$. So, for certain scenarios, it is possible to correct errors even when $d^* < 2t + 1$. However, in many cases a single error correction may not be possible with this code. For example, if 00000 was sent and 01000 was received,

$$d(01000, 00000) = 1,\ d(01000, 01010) = 1,$$
$$d(01000, 10101) = 4,\ d(01000, 11111) = 4.$$

In this case there cannot be a clear-cut decision, and a fair coin will have to be flipped.

Definition 3.15 An **Incomplete Decoder** decodes only those received codewords that are clearly closest to one of the codewords. In the case of ambiguity, the decoder declares that the received word to be unrecognisable. The receiver is then requested to re-transmit.

Scanned by CamScanner

A Complete Decoder decodes every received word, i.e., it tries to map every received word to some codeword, even if it has to make a guess.

Example 3.15 was that of a complete decoder. Such decoders may be used when it is better to have a good guess rather than to have no guess at all. Many of the real life decoders are incomplete decoders. Usually they send a message back to the transmitter requesting them to re-transmit.

Example 3.16 The **International Standard Book Number** (ISBN) is a linear block code of blocklength $n = 10$ over $GF(11)$. The symbols used are 0, 1, 2, ..., 9, X. Instead of using the symbol '10', 'X' is used in order to avoid confusion between '10' and the case when '1' is followed by a '0'. The ISBN satisfies the following constraint:

$$\sum_{i=0}^{9}(10-i)c_i = 0 \text{ calculated (mod 11).} \tag{3.9}$$

For example, consider the ISBN 0-07-048297-7. If we perform the check on this ISBN we get

$$10\times0 + 9\times0 + 8\times7 + 7\times0 + 6\times4 + 5\times8 + 4\times2 + 3\times9 + 2\times7 + 1\times7 = 176 = 0 \text{ (mod 11).}$$

Thus 0-07-048297-7 is a valid ISBN.

Since ISBN is a linear block code, the all-zero codeword is a valid codeword. Also, 1000000001 is a valid ISBN. Thus the minimum weight of the code is $d^* = 2$. Theoretically it cannot correct any errors and can only detect a single error. However, suppose one of the digits of the ISBN gets smudged and we have 0-07-048e97-7. We can recover the erroneous digit, e, by solving $(10\times0 + 9\times0 + 8\times7 + 7\times0 + 6\times4 + 5\times8 + 4\times e + 3\times9 + 2\times7 + 1\times7)$ mod 11 = 0. The solution of this equation with one unknown yields $e = 2$. Thus, we have been able to *correct* a single error simply because we know the location of the error digit.

Definition 3.16 A receiver declares that an **Erasure** has occurred (i.e., a received symbol has been erased) when the symbol is received ambiguously, or the presence of an interference is detected during reception.

Example 3.17 Consider a binary pulse amplitude modulation (PAM) scheme where 1 is represented by five volts and 0 is represented by zero volts. The noise margin is one volt, which implies that at the receiver:

if the received voltage is between 4 volts and 5 volts $\Rightarrow$ the bit sent is 1,
if the received voltage is between 0 volt and 1 volts $\Rightarrow$ the bit sent is 0,
if the received voltage is between 1 volt and 4 volts $\Rightarrow$ an erasure has occurred.

Thus if the receiver received 2.9 volts during a bit interval, it will declare that an erasure has occurred.

A channel can be prone both to errors and erasures. If in such a channel t errors and r erasures occur, the error correcting scheme should be able to compensate for the erasures and correct the errors as well. If r erasures occur, the minimum distance of the code will become $d^* - r$ in the worst case. This is because, the erased symbols have to be simply discarded, and if they were contributing to the minimum distance, this distance will reduce. A simple example will illustrate the point. Consider the **repetition code** in which

$$0 \rightarrow 00000$$
$$1 \rightarrow 11111$$

Here, $d^* = 5$. If $r = 2$, i.e., two bits get erased (let us say the first two), we will have

$$0 \rightarrow ??000$$
$$1 \rightarrow ??111$$

Now, the effective minimum distance $d_1^* = d^* - r = 3$.

Therefore, for a channel with t errors and r erasures, $d^* - r \geq 2t + 1$.

Or,

$$d^* \geq 2t + r + 1. \tag{3.10}$$

For a channel which has no errors ($t = 0$), only r erasures,

$$d^* \geq r + 1. \tag{3.11}$$

Next let us give a little more formal treatment to the decoding procedure. Can we construct some mathematical tools to simplify the nearest neighbour decoding? Suppose the codeword $c = c_1c_2 \ldots c_n$ is transmitted over a noisy channel. The noise in the channel changes some or all of the symbols of the codeword. Let the received vector be denoted by $v = v_1v_2 \ldots v_n$. Define the **error vector** as

$$e = v - c = v_1v_2 \ldots v_n - c_1c_2 \ldots c_n = e_1e_2 \ldots e_n. \tag{3.12}$$

The decoder has to decide from the received vector, v, which codeword was transmitted, or equivalently, it must determine the error vector, e.

Definition 3.17 Let C be an (n, k) code over $GF(q)$ and a be any vector of length n. Then the set

$$a + C = \{a + x \mid x \in C\} \tag{3.13}$$

is called a **Coset** (or translate) of C. a and b are said to be in the same coset if $(a - b) \in C$.

Theorem 3.6 Suppose C is an (n, k) code over $GF(q)$. Then

(i) every vector b of length n is in some coset of C,
(ii) each coset contains exactly q^k vectors,
(iii) two cosets are either disjoint or coincide (partial overlap is not possible).
(iv) if $a + C$ is a coset of C and $b \in a + C$, we have $b + C = a + C$.

Scanned by CamScanner

Proofs

(i) $b = b + 0 \in b + C$.

(ii) Observe that the mapping $C \to a + C$ defined by $x \to a + x$, for all $x \in C$ is a one-to-one mapping. Thus the cardinality of $a + C$ is the same as that of C, which is equal to q^k.

(iii) Suppose the cosets $a + C$ and $a + C$ overlap, i.e., they have atleast one vector in common. Let $v \in (a + C) \cap (b + C)$. Thus, for some $x, y \in C$,

$v = a + x = b + y$.

Or, $b = a + x - y = a + z$, where $z \in C$ (because, the difference of two codewords is also a codeword).

Thus, $b + C = a + C + z$ or $(b + C) \subset (a + C)$.

Similarly it can be shown that $(a + C) \subset (b + C)$. From these two we can conclude that $(b + C) = (a + C)$.

(iv) Since $b \in a + C$, it implies that $b = a + x$, for some $x \in C$. Next, if $b + y \in b + C$, then $b + y = (a + x) + y = a + (x + y) \in a + C$.

Hence, $b + C \subseteq a + C$. On the other hand, if $a + z \in a + C$, then,

$a + z = (b - x) + z = b + (z - x) \in b + C$.

Hence, $a + C \subseteq b + C$, and so $b + C = a + C$.

Definition 3.18 The vector having the minimum weight in a coset is called the **Coset Leader**. If there are more than one vector with the minimum weight, one of them is chosen at random and is declared the coset leader.

Example 3.18 Let C be the binary (3, 2) code with the generator matrix given by

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

i.e.

$C = \{000, 010, 101, 111\}$.

The cosets of C are

$000 + C = 000, 010, 101, 111,$

$001 + C = 001, 011, 100, 110.$

Note that all the eight vectors have been covered by these two cosets. As we have already seen (in Theorem 3.6), if $a + C$ is a coset of C and $b \in a + C$, we have $b + C = a + C$. Hence all cosets have been listed. For the sake of illustration we write down the following

$010 + C = 010, 000, 111, 101,$

$011 + C = 011, 001, 110, 101,$

$100 + C = 100, 110, 001, 011,$

$101 + C = 101, 111, 000, 101,$

$110 + C = 110, 100, 011, 001,$

$111 + C = 111, 101, 010, 000.$

It can be seen that all these sets are already covered.

Since two cosets are either disjoint or coincide (from Theorem 3.6), the set of all vectors, $GF(q^n)$ can be written as

$$GF(q^n) = C \cup (a_1 + C) \cup (a_2 + C) \cup \ldots \cup (a_t + C)$$

where $t = q^{n-k} - 1$.

Definition 3.19 A **Standard Array** for an (n, k) code C is a $q^{n-k} \times q^k$ array of *all* vectors in $GF(q^n)$ in which the first row consists of the code C (with **0** on the extreme left), and the other rows are the cosets $a_i + C$, each arranged in corresponding order, with the coset leader on the left.

Steps for constructing a standard array:

(i) In the first row write down all the valid codewords, starting with the all-zero codeword.

(ii) Choose a vector a_1 which is not in the first row. Write down the coset $a_1 + C$ as the second row such that $a_1 + x$ is written under $x \in C$.

(iii) Next choose another vector a_2 (not present in the first two rows) of minimum weight and write down the coset $a_2 + C$ as the third row such that $a_2 + x$ is written under $x \in C$.

(iv) Continue the process until all the cosets are listed and every vector in $GF(q^n)$ appears exactly once.

Example 3.19 Consider the code $C = \{0000, 1011, 0101, 1110\}$. The corresponding standard array is

codewords →	0000	1011	0101	1110
	1000	0011	1101	0110
	0100	1111	0001	1010
	0010	1001	0111	1100
	↑			
	coset leader			

Note that each entry is the sum of the codeword and its coset leader.

Let us now look at the concept of decoding (obtaining the information symbols from the received codewords) using the standard array. Since the standard array comprises of all possible words belonging to $GF(q^n)$, the received word can always be identified with one of the elements of the standard array. If the received word is a valid codeword, it is concluded that no errors have occurred (this conclusion may be wrong with a very low probability of error, when one valid codeword gets modified to another valid codeword due to noise). In the case that the received word, v, does not belong to the set of valid codewords, we surmise that an error has occurred. The decoder then declares that the coset leader is the error vector, e, and decodes the codeword as $v - e$. This is the codeword at the top of the column containing v. Thus, mechanically, we decode the codeword as the one on the top of the column containing the received word.

Scanned by CamScanner

Example 3.20 Suppose the code in the previous example $C = \{0000, 1011, 0101, 1110\}$ is used and the received word is $v = 1101$. Since it is not one of the valid codewords, we deduce that an error has occurred. Next we try to estimate which one of the four possible codewords was actually transmitted. If we make use of the standard array of the earlier example, we find that 1101 lies in the 3rd column. The top-most entry of this column is 0101. Hence the estimated codeword is 0101. Observe that:

$$d(1101, 0000) = 3,\ d(1101, 1011) = 2$$
$$d(1101, 0101) = 1,\ d(1101, 1110) = 2$$

and the error vector $e = 1000$, the coset leader.

Codes with larger blocklengths are desirable (though not always; see the concluding remarks on this chapter) because the code rates of larger codes perform closer to the Shannon limit. As we go to larger codes (with larger values of k and n) the method of standard array will become less practical because the size of the standard array ($q^{n-k} \times q^k$) will become unmanageably large. One of the basic objectives of coding theory is to develop efficient decoding strategies. If we are to build decoders that will work in real-time, the decoding scheme should be realisable both in terms of memory required as well as the computational load. Is it possible to reduce the standard array? The answer lies in the concept of syndrome decoding, which we are going to discuss next.

3.7 SYNDROME DECODING

The standard array can be simplified if we store only the first column, and compute the remaining columns, if needed. To do so, we introduce the concept of the *syndrome* of the error pattern.

Definition 3.20 Suppose H is a parity check matrix of an (n, k) code. Then for any vector $v \in GF(q^n)$, the vector

$$s = vH^T \tag{3.14}$$

is called the **Syndrome** of v. The syndrome of v is sometimes explicitly written as $s(v)$. It is called a syndrome because it gives us the symptoms of the error, thereby helping us to diagnose the error.

Theorem 3.7 Two vectors x and y are in the same coset of C if and only if they have the same syndrome.

Proof The vectors x and y belong to the same coset $\Leftrightarrow x + C = y + C$

$$\Leftrightarrow x - y \in C$$
$$\Leftrightarrow (x - y)H^T = 0$$

$$\Leftrightarrow xH^T = yH^T$$
$$\Leftrightarrow s(x) = s(y)$$

Thus there is a one-to-one correspondence between cosets and syndromes.

We can reduce the size of the standard array by simply listing the syndromes and the corresponding coset leaders.

Example 3.21 We now extend the standard array listed in Example 3.19 by adding the syndrome column.

The code is $C = \{0000, 1011, 0101, 1110\}$. The corresponding standard array is

					Syndrome
Codewords →	0000	1011	0101	1111	00
	1000	0011	1101	0110	11
	0100	1111	0001	1010	01
	0010	1001	0111	1100	10
	↑ coset leader				

The steps for syndrome decoding are as follows:

(i) Determine the syndrome ($s = vH^T$) of the received word, v.

(ii) Locate the syndrome in the 'syndrome column'.

(iii) Determine the corresponding coset leader. This is the error vector, e.

(iv) Subtract this error vector from the received word to get the codeword $y = v - e$.

Having developed an efficient decoding methodology by means of syndrome decoding, let us now find out how much advantage does coding actually provide.

3.8 ERROR PROBABILITY AFTER CODING (PROBABILITY OF ERROR CORRECTION)

Definition 3.21 The **probability of error** (or, the word error rate) P_{err} for any decoding scheme is the probability that the decoder output is a wrong codeword. It is also called the **Residual Error Rate**.

Suppose there are M codewords (of length n) which are used with equal probability. Let the decoding be done using a standard array. Let the number of coset leaders with weight i be denoted by α_i. We assume that the channel is a binary symmetric channel (BSC) with symbol error probability p. A decoding error occurs if the error vector e is *not* a coset leader. Therefore, the probability of correct decoding will be

Scanned by CamScanner

$$P_{cor} = \sum_{i=0}^{n} \alpha_i p^i (1-p)^{n-i}. \tag{3.14}$$

Hence, the probability of error will be

$$P_{err} = 1 - \sum_{i=0}^{n} \alpha_i p^i (1-p)^{n-i}. \tag{3.15}$$

Let us now consider m errors within the block length of n bits. We have seen that for a linear block code with minimum distance d^*, any number of errors up to $\left\lfloor \frac{1}{2}(d^*-1) \right\rfloor$ are always correctable. The probability of m errors in a block of n bits is

$$P_{m,n} = \binom{n}{m} p^m (1-p)^{n-m}, \tag{3.16}$$

where p is the probability of one bit in error. The codeword will be in error if more than t errors occur. Thus, the probability of a codeword error is upper bounded by

$$P_M \le \sum_{m=t+1}^{n} P_{m,n}. \tag{3.17}$$

The equality holds for the case of **Perfect Codes**, which we shall study shortly. We also note that P_M cannot be less than the probability of erroneously decoding a transmitted codeword as another codeword which is at a distance d^*. That is

$$P_M \ge \sum_{m=\lfloor d^*/2 \rfloor+1}^{d^*} \binom{d^*}{m} p^m (1-p)^{d^*-m}. \tag{3.18}$$

We can obtain an upper-bound if we consider all the $M-1$ other codewords. All these codewords are at a distance of at least d^*. Therefore, the union bound can be expressed as

$$P_M \le (M-1) \sum_{m=\lfloor d^*/2 \rfloor+1}^{d^*} \binom{d^*}{m} p^m (1-p)^{d^*-m}. \tag{3.19}$$

Example 3.22 Consider the standard array in Example 3.19. The coset leaders are 0000, 1000, 0100 and 0010. Therefore $\alpha_0 = 1$ (only one coset leader with weight equal to zero), $\alpha_1 = 3$ (the remaining three are of weight one) and all other $\alpha_i = 0$.

Therefore,

$$P_{err} = 1 - [(1-p)^4 + 3p(1-p)^3].$$

Recall that this code has four codewords, and can be used to send 2 bits at a time. If we did not perform coding, the probability of error of the 2-bit message being received incorrectly would be

$$P_{err} = 1 - P_{cor} = 1 - (1-p)^2.$$

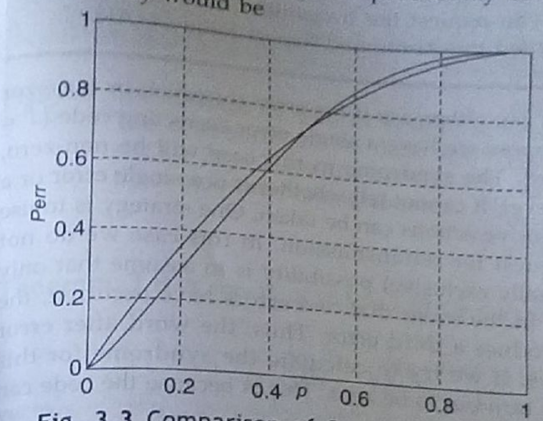


Fig. 3.3 Comparison of P_{err} for coded and uncoded 2-bit messages.

Note that for $p = 0.01$, the word error rate (upon coding) is $P_{err} = 0.0103$, while for the uncoded case $P_{err} = 0.0199$. So coding has almost halved the word error rate. The comparison of P_{err} for messages with and without coding is plotted in Fig. 3.3. It can be seen that coding outperforms the uncoded case only for $p < 0.5$ (guess why?). Note that the improvement due to coding comes at the cost of information transfer rate. In this example, the rate of information transfer has been cut down by half as we are sending two parity bits for every two information bits.

Example 3.23 This example will help us visualise the power of coding. Consider a binary symmetric channel with the probability of symbol error $p = 10^{-7}$. Suppose that, 10 bit long words are being transmitted *without coding*. Let the bit rate of the transmitter be 10^7 b/s, which implies that 10^6 word/s are being sent. The probability that a word is received incorrectly is

$$\binom{10}{1}(1-p)^9 p + \binom{10}{2}(1-p)^8 p^2 + \binom{10}{3}(1-p)^7 p^3 + \ldots \approx \binom{10}{1}(1-p)^9 p = 10^{-6} \text{ word/s}.$$

Therefore, in one second, $10^{-6} \times 10^6 = 1$ word will be in error. The implication is that every second a word will be in error and *it will not be detected*.

Next, lets add a parity bit to the uncoded words so as to make them 11 bits long. The parity makes all the codewords of even parity and thus ensures that a single bit in error will be detected. The only way that the coded word will be in error if two or more bits get flipped, i.e., at least two bits are in error. This can be computed as 1 – probability that less than two bits are in error. Therefore, the probability of word error will be

$$1-(1-p)^{11} - \binom{11}{1}(1-p)^{10} p \approx 1-(1-11p)-11(1-10p)p = 110p^2 = 11\times 10^{-13}.$$

The new word rate will be $10^7/11$ word/s because now 11 bits constitute one word and the bit rate is the same as before. Thus in one second, $(10^7/11) \times (11 \times 10^{-13}) = 10^{-6}$ words will be in error. This implies that, after coding, one word will be received incorrectly *without detection* every 10^6 seconds = 11.5 days!

Scanned by CamScanner

So just by increasing the word length from 10 bits (coded) to 11 bits (with coding), we have been able to obtain a dramatic decrease in the word error rate. For the second case, each time a word is detected to be in error, we can request the transmitter to re-transmit the word. This strategy for retransmission is called the **Automatic Repeat Request (ARQ)**.

If we have a linear block (n, k, d^*) code we can either use it for error correction or for error detection (typically combined with ARQ). Suppose we have a single error correcting code ($d^* = 3$) and two errors occur in the received word. The syndrome in this case will be non-zero, indicating the presence of 1 or 2 errors. However, it cannot tell whether it is a single error or a double error. At this stage, two mutually exclusive actions can be taken. One strategy is to use the code as an error detecting tool and request for retransmission. In this case we do not attempt to correct the error. The other (mutually exclusive) possibility is to assume that only one error has occurred and try to rectify it. In the event that two errors have occurred, the attempt to correct a single error might introduce a third error. Thus, the word after error correction may actually contain three errors. If we try to calculate the syndrome for this wrongly 'corrected' word, the syndrome may turn out to be zero. This is because the code can only detect up to 2 errors.

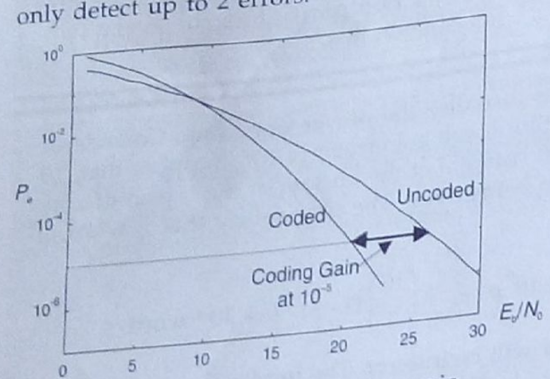


Fig. 3.4 Illustration of coding gain.

It is important to ask the amount of energy saved because of channel coding. The probability of error (P_e) curve versus the E_b/N_0 of a typical communication system is plotted in Fig 3.4. It is seen from the figure that with coding, the same value of P_e can be achieved at a lower E_b/N_0 than without coding. Thus the coding scheme is able to provide a **Coding Gain**. Coding gain is measured in dB. It should be observed that the coding gain can be negative if, the P_e is not low enough. This implies that channel coding provides improvement if the channel is not too bad to begin with. Also, coding gain typically increases as P_e decreases. The limiting value, as $P_e \to 0$, is called the **Asymptotic Coding Gain**.

3.9 PERFECT CODES

Definition 3.22 For any vector u in $GF(q^n)$ and any integer $r \geq 0$, the sphere of radius r and centre u, denoted by $S(u, r)$, is the set $\{v \in GF(q^n) \mid d(u, v) \leq r\}$.

This definition can be interpreted graphically, as shown in Fig. 3.5. Consider a code C with minimum distance $d^*(C) \geq 2t + 1$. The spheres of radius t centred at the codewords $\{c_1, c_2, \ldots, c_M\}$

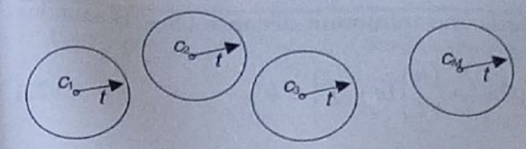


Fig. 3.5 The concept of spheres in $GF(q^n)$. The codewords of the code with $d^*(C) \geq 2t + 1$ are the centres of these non-overlapping spheres.

of C will then be in disjoint. Now consider the decoding problem. Any received vector can be represented as a point in this space. If this point lies within a sphere, then by nearest neighbour decoding it will be decoded as the centre of the sphere. If t or fewer errors occur, the received word will definitely lie within the sphere of the codeword that was transmitted, and will be correctly decoded. If, however, larger than t errors occur, it will escape the sphere, thus resulting in an incorrect decoding.

Theorem 3.8 A sphere of radius r $(0 \leq r \leq n)$ contains exactly

$$\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \ldots + \binom{n}{r}(q-1)^r \tag{3.20}$$

vectors.

Proof Consider a vector u in $GF(q^n)$ and another vector v which is at a distance m from u. This implies that the vectors u and v differ at exactly m places. The total number of ways in which m position can be chosen from n positions is $\binom{n}{m}$. Now, each of these m places can be replaced by $(q - 1)$ possible symbols. This is because the total size of the alphabet is q, out of which one is currently being used in that particular position in u. Hence, the number of vectors at a distance exactly m from u is

$$\binom{n}{0} + \binom{n}{1}(q-1) + \binom{n}{2}(q-1)^2 + \ldots + \binom{n}{r}(q-1)^r. \tag{3.21}$$

Example 3.24 Consider a binary code (i.e., $q = 2$) and blocklength $n = 4$. The number of vectors at a distance 2 or less from any codeword will be

$$\binom{4}{0} + \binom{4}{1}(1) + \binom{4}{2}(1)^2 = 1 + 4 + 6 = 11.$$

Without the loss of generality we can choose the fixed vector $u = 0000$. The vectors of distance 2 or less are:

vectors at a distance 2: 0011, 1001, 1010, 1100, 0110, 0101,
vectors at a distance 1: 0001, 0010, 0100, 1000,
vectors at a distance 0: 0000.
Thus there is a total of 11 such vectors.

Scanned by CamScanner

Theorem 3.9 A q-ary (n, k) code with M codewords and minimum distance $(2t + 1)$ satisfies

$$M\left\{\binom{n}{0}+\binom{n}{1}(q-1)+\binom{n}{2}(q-1)^2+\ldots+\binom{n}{t}(q-1)^t\right\}\le q^n. \tag{3.22}$$

Proof Suppose C is a q-ary (n, k) code. Consider spheres of radius t centred on the M codewords. Each sphere of radius t has

$$\binom{n}{0}+\binom{n}{1}(q-1)+\binom{n}{2}(q-1)^2+\ldots+\binom{n}{t}(q-1)^t,$$

vectors (theorem 3.8). Since none of the spheres intersect, the total number of vectors for the M disjoint spheres is

$$M\left\{\binom{n}{0}+\binom{n}{1}(q-1)+\binom{n}{2}(q-1)^2+\ldots+\binom{n}{t}(q-1)^t\right\},$$

which is upper bounded by q^n, the total number of vectors of length n in $GF(q^n)$.

This bound is called the **Hamming Bound** or the **Sphere Packing Bound** and it holds good for nonlinear codes as well. For binary codes, the Hamming bound will become

$$M\left\{\binom{n}{0}+\binom{n}{1}+\binom{n}{2}+\ldots+\binom{n}{t}\right\}\le 2^n. \tag{3.23}$$

It should be noted here that just because a set of integers n, M and t satisfies the Hamming bound, the code of these specifications may not exist. For example the set $n = 5$, $M = 5$ and $t = 1$ satisfies the Hamming bound, however, no binary code exists for these specifications.

Observe that for the case when $M = q^k$, the Hamming bound may be alternately written as

$$\log_q\left\{\binom{n}{0}+\binom{n}{1}(q-1)+\binom{n}{2}(q-1)^2+\ldots+\binom{n}{t}(q-1)^t\right\}\le n-k. \tag{3.24}$$

Definition 3.23 A **Perfect Code** is one which achieves the Hamming bound, *i.e.*,

$$M\left\{\binom{n}{0}+\binom{n}{1}(q-1)+\binom{n}{2}(q-1)^2+\ldots+\binom{n}{t}(q-1)^t\right\}=q^n, \tag{3.25}$$

for a perfect code.

For a perfect code there are equal radii disjoint spheres centred at the codewords which completely fill the space. Thus, a t error correcting perfect code utilises the entire space in the most efficient manner.

Example 3.25 Consider the binary repetition code

$$C=\begin{cases}00\ldots0\\11\ldots1\end{cases}$$

of block length n, where n is odd. In this case $M = 2$ and $t = (n - 1)/2$. Upon substituting these values in the left-hand side of the inequality for Hamming bound we get

$$\text{LHS} = 2\left\{\binom{n}{0}+\binom{n}{1}+\binom{n}{2}+\ldots+\binom{n}{(n-1)/2}\right\} = 2.2^{n-1} = 2^n = \text{RHS}.$$

Thus the repetition code is a perfect code. It is actually called a *trivial* perfect code. In the next chapter we shall see some examples of non-trivial perfect codes.

One of the ways to search for perfect codes is to obtain the integer solutions for the parameters n, q, M and t in the equation for Hamming bound. Some of the solutions found by exhaustive computer search are listed in the table below:

S.N.	n	q	M	t
1	23	2	2^{12}	3
2	90	2	2^{78}	2
3	11	3	3^6	2

3.10 HAMMING CODES

There are both binary and non-binary Hamming codes. Here, we shall limit our discussion to binary Hamming codes. The binary Hamming codes have the property that

$$(n, k) = (2^m - 1, 2^m - 1 - m), \tag{3.26}$$

where m is any positive integer. For example, for $m = 3$ we have a $(7, 4)$ Hamming code. The parity check matrix, H, of a Hamming code is a very interesting matrix. Recall that the parity check matrix of an (n, k) code has $n - k$ rows and n columns. For the binary (n, k) Hamming code, the $n = 2^m - 1$ columns consist of all possible binary vectors with $n - k = m$ elements, except the all-zero vector.

Example 3.26 The generator matrix for the binary $(7, 4)$ Hamming code is given by

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

Scanned by CamScanner

The corresponding parity check matrix is

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

Observe that the columns of the parity check matrix consist of (100), (010), (101), (110), (111), (011) and (001). These seven are all the possible non-zero binary vectors of length three. It is quite easy to generate a systematic Hamming code. The parity check matrix H can be arranged in a systematic form as follows

$$H=\begin{bmatrix}1&1&1&0&1&0&0\\0&1&1&1&0&1&0\\1&1&0&1&0&0&1\end{bmatrix}=[-P^T \mid I]$$

Thus, the generator matrix in the systematic form for the binary Hamming code is

$$G=[I \mid P]=\begin{bmatrix}1&0&0&0&1&0&1\\0&1&0&0&1&1&1\\0&0&1&0&1&1&0\\0&0&0&1&0&1&1\end{bmatrix}$$

From the above example we observe that no two columns of H are linearly dependent (otherwise they would be identical). However, for $m > 1$, it is possible to identify three columns of H that would add up to zero. Thus, the minimum distance, d^*, of an (n, k) Hamming code is equal to 3, which implies that it is a single-error correcting code. Hamming codes are perfect codes.

By adding an overall parity bit, an (n, k) Hamming code can be modified to yield an $(n+1, k)$ code with $d^* = 4$. On the other hand, an (n, k) Hamming code can be shortened to an $(n-l, k-l)$ code by removing l rows of its generator matrix G or, equivalently, by removing l columns of its parity check matrix, H. We can now give a more formal definition of Hamming Codes.

Definition 3.24 Let $n = (q^k - 1)/(q - 1)$. The (n, k) Hamming code over $GF(q)$ is a code for which the parity check matrix has columns that are pairwise linearly independent (over $GF(q)$), i.e., the columns are a maximal set of pairwise linearly independent vectors.

3.11 LOW DENSITY PARITY CHECK (LDPC) CODES

Definition 3.25 An (r, s) **Gallager code** is a linear code with a check matrix H, satisfying the condition that every column has r ones and every row has s ones.

Definition 3.26 A Gallager code with small values of r and s is a **Low Density Parity Check (LDPC)** code. Thus, an LDPC code has a sparse parity check matrix, with very few ones in each row and column. Typically, $r \le \log_2 n$, where n is the blocklength. The code is succinctly written as a (n, r, s) LDPC code.

Example 3.27 Consider the following parity check matrix

$$H=\begin{bmatrix}1&0&0&0&1&1&1&0&0&0\\0&1&0&0&1&0&0&1&1&0\\0&0&1&0&0&1&0&1&0&1\\0&0&0&1&0&0&1&0&1&1\\1&1&1&1&0&0&0&0&0&0\end{bmatrix} \tag{3.27}$$

The matrix has $r = 2$ ones in each column and $s = 4$ two ones in each row. We cannot have any more independent columns in this 5×10 matrix because all the possible combinations of choosing two ones out of the five possible locations in each column i.e. $\binom{5}{2}$ have been exhausted. Since the dimension of the parity check matrix is $(n-k) \times n$, we have $n = 10$ and $k = 5$. We also observe that the last three columns, i.e., 8 through 10 add up to the zero vector. Therefore, the minimum distance of this code is 3. This is a (10, 2, 4) LDPC code.

Definition 3.27 An LDPC code with a fixed r and s is called a **regular** LDPC code. If the number of ones in the columns and the number of ones in the rows are approximately r and s, it is called an **irregular** LDPC code.

LDPC codes can be constructed by using random constructions, algebraic constructions or a combination of the two. A simple algorithm for random construction of LDPC codes is given below:

Step 1: Set $i = 1$.

Step 2: Generate a random binary vector of length $\frac{nr}{s}$ and Hamming weight r. This is the ith column of H.

Step 3: If the weight of each row of H at this point is $\le s$, and the scalar product of each pair of columns is ≤ 1, set $i = i + 1$. Else, go to step 2.

Step 4: If $i = n$, stop. Else, go to step 2.

The random algorithm does not guarantee s ones in each row. Hence, we may generate

Scanned by CamScanner

irregular LDPC codes by this method. To make the code regular, suitable constraints have to be put in the algorithm.

An algorithm for algebraic construction of LDPC codes is given below:

Step 1: Choose $p > \frac{r-1}{s-1}$

Step 2: Construct a $p \times p$ matrix from the identity matrix I_p by cyclically shifting its rows by one position to the right:

$$J = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \tag{3.28}$$

Step 3: The parity check matrix is obtained as

$$H = \begin{bmatrix} J^0 & J^0 & J^0 & \cdots & J^0 \\ J^0 & J^1 & J^2 & \cdots & J^{(s-1)} \\ J^0 & J^2 & J^3 & \cdots & J^{(2(s-1))} \\ & & & \cdots & \\ J^0 & J^{(r-1)} & J^{(2(r-1))} & \cdots & J^{((r-1)(s-1))} \end{bmatrix} \tag{3.29}$$

where $J^0 = I_p$ and the lth power of J is obtained from I_p by cyclically shifting its rows by l mod p positions to the right. This matrix has exactly r ones in each column and s ones in each row. Thus it is a regular LDPC code, by design.

In a regular LDPC code, each received codeword is checked by precisely r equations and every check equation adds up to precisely s code symbols. All linear codes, including LDPC codes, can be represented using a **Tanner Graph**.

Definition 3.28 A **Tanner Graph** is a graphical representation of the linear code based on the set of check equations. It is a bipartite graph, which means it has two kinds of nodes: symbol nodes and check nodes. Each symbol node is connected only to check nodes and each check node is connected only to symbol nodes.

Each component of the codeword is represented in the Tanner graph by a symbol node. Hence, there are n symbol nodes. Each check equation of the code is represented in the Tanner graph by a check node. Hence, there are $n - k$ check nodes. Each check node is connected by an edge to those symbol nodes that it checks.

An LDPC code can be decoded using a simple iterative **bit flipping algorithm**. This sub optimum algorithm can be summarised in the following steps:

1. Perform hard decision decoding of the received symbols from the channel to form a binary vector, v. If we have the received word to start with, this step need not be performed. The vector v is n-bit long.
2. Compute the syndrome $s = vH^T$. Note that each component of v affects only s components of the syndrome, s. If only a single bit is in error, only s syndrome components will be equal to 1.
3. Compute all check sums and the number of unsatisfied parity checks involving each of the n bits of v.
4. Flip those bits of v which are involved in the largest number of unsatisfied parity checks.
5. Go back to step number 3. Keep iterating until either:
   a. All checks are satisfied, or
   b. A predetermined number of iterations are reached.

The flipping algorithm does not guarantee that errors up to half the minimum distance are corrected. However, for large block lengths, this sub optimum algorithm works remarkably well.

Example 3.28 Let us consider the linear rate 1/3 code given by the following parity check matrix

$$H = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix} \tag{3.30}$$

Here $n = 6$, $r = 3$ and $s = 2$. This is not exactly an LDPC code since n is not large enough to yield a sparse H matrix. The Tanner graph for this code is shown in Fig 3.6.

Suppose the received word is $v = [0\ 0\ 1\ 0\ 0\ 0]$. By performing a syndrome calculation we obtain $s = vH^T = [1\ 0\ 0\ 1]$, which is non-zero. Hence this is not a valid codeword. From the syndrome vector we note that the parity checks that have failed are 1 and 4. This implies that there is an error among the symbols connected to the check nodes 1 and 4 of the Tanner graph. We make the following observations:

(a) Bit 4 of the received vector corresponds to no failed checks because it is connected to check nodes 2 and 3, both of which are zeros in the syndrome vector.

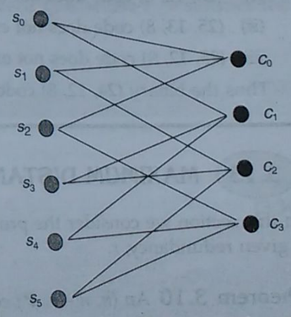


Fig. 3.6 The Tanner graph corresponding to the rate 1/3 linear code discussed in Example 3.28.

Scanned by CamScanner

(b) Bits 1 and 2 of the received vector correspond to one failed check because they are connected to check node 1.

(c) Bits 5 and 6 of the received vector correspond to one failed check because they are connected to check node 4.

(d) Bit 3 of the received vector corresponds to two failed checks because it is connected to check nodes 1 and 4, both of which are ones in the syndrome vector.

According to the bit flipping algorithm, we flip the 3rd bit of the received vector to obtain [0 0 0 0 0 0]. We again perform the parity check and find that this meets all the checks. Hence, the corrected vector is the all-zero vector in this case.

3.12 OPTIMAL LINEAR CODES

Definition 3.29 For an (n, k, d^*) **optimal code**, no $(n-1, k, d^*)$, $(n+1, k+1, d^*)$ or $(n+1, k, d^*+1)$ code exists.

Optimal linear codes give the best distance property under the constraint of the block length. Most of the optimal codes have been found by long computer searches. It may be possible to have more than one optimal code for a given set of parameters n, k and d^*. For instance there exist two different binary (25, 5, 10) optimal codes.

Example 3.29 Consider the (24, 12, 8) binary code. It can be verified that a

(i) (23, 12, 8) code does not exist (only the (23, 12, 7) code exists),

(ii) (25, 13, 8) code does not exist,

(iii) (25, 12, 9) code does not exist.

Thus the binary (24, 12, 8) code is an optimal code.

3.13 MAXIMUM DISTANCE SEPARABLE (MDS) CODES

In this section we consider the problem of finding as large a minimum distance d^* possible for a given redundancy, r.

Theorem 3.10 An $(n, n-r, d^*)$ code satisfies $d^* \le r+1$.

Proof From the Singleton bound we have

$$d^* \le n-k+1.$$

Substitute $k = n-r$ to get

$$d^* \le r+1.$$

Definition 3.30 A $(n, n-r, r+1)$ code is called a **Maximum Distance Separable** (MDS) code. An MDS code is a linear code of redundancy r, whose minimum distance is equal to $r+1$.

Except the trivial repetition code, there are no binary MDS codes. However, we do have non-binary MDS codes, for example, the Reed-Solomon codes, which we will study in Chapter 5.

3.14 BOUNDS ON MINIMUM DISTANCE

A simple upper bound on the minimum distance of an (n, k) binary or non binary linear block code was given as the Singleton bound in (3.7) as $d^* \le n-k+1$. The normalised form of this bound can be written as

$$\frac{d^*}{n} \le (1-r) + \frac{1}{n}, \tag{3.31}$$

where r is the code rate. For large values of block length, the factor $1/n$ can be neglected. Another upper bound can be obtained directly from the Hamming bound expressed in (3.25). By dividing both sides of the inequality by n and setting $q = 2$ for the binary case, we obtain

$$\frac{1}{n}\log_2\left(\sum_{i=0}^{t}\binom{n}{i}\right) \le 1-r. \tag{3.32}$$

Since the minimum distance, d^*, and the error correcting capability, t, are related, the above relation is an upper bound on d^*. For any n, let t_0 be the largest integer t for which (3.32) holds. Then, it can be shown that as $n \to \infty$, the ratio t/n cannot exceed t_0/n, where t_0/n satisfies the equation

$$1-r = H\left(\frac{t_0}{n}\right). \tag{3.33}$$

The generalized Hamming bound can be written as

$$\frac{1}{n}\log_q\left(\sum_{i=0}^{t}\binom{n}{i}(q-1)^i\right) \le 1-r. \tag{3.34}$$

Another bound on d^* given by Plotkin is

$$\frac{1}{n}\left(\frac{qd^*-1}{q-1} - 1 - \log_q d^*\right) \le 1-r. \tag{3.35}$$

A tighter upper bound on d^* is due to Elias, and is given in its asymptotic form, in (3.36):

Scanned by CamScanner

$$\frac{d^*}{n} \le 2A(1-A), \tag{3.36}$$

where

$$r = 1 + A\log_2 A + (1-A)\log_2(1-A), \quad 0 \le A \le 1/2. \tag{3.37}$$

Lower bounds also exist for d^*. A lower bound on d^*, based on a theorem proposed by Gilbert and Varshamov, is given below

$$\frac{d^*}{n} \ge \alpha, \tag{3.38}$$

where

$$r = 1 + \alpha\log_2 \alpha + (1-\alpha)\log_2(1-\alpha) = 1 - H(\alpha), \quad 0 \le \alpha \le 1/2. \tag{3.39}$$

3.15 SPACE–TIME BLOCK CODES

Space–time codes are coding techniques designed for multiple antenna transmissions. In particular, coding is performed by adding properly designed redundancy in both spatial and temporal domains. Space–time codes were first introduced by Tarokh, Seshadri, and Calderbank. In a space–time coded system, the symbol error rate (SER) for the system can be approximated by

$$\overline{P}_e \approx \frac{c}{(G_c S)^{G_d}} \tag{3.40}$$

where S is the SNR and c is a scaling constant specific to the modulation employed and the nature of the channel. G_c ($G_c \ge 1$) denotes the **Coding Gain** and G_d is the **Diversity Order** of the system. The diversity gain/order determines the slope of an error rate curve plotted as a function of SNR, while the coding gain determines the horizontal shift of the uncoded system error rate curve to the space-time coded error rate curve obtained for the same diversity order. There are two main kinds of space-time coding schemes, namely, space–time trellis codes (STTC) and space–time block codes (STBC). While STTC are designed to provide both maximum coding and diversity gain, STBC provide only full diversity gain and zero or minimal coding gain. Here we will study the STBC and return to STTC in Chapter 7.

The **Alamouti scheme** presented in 1998 is, historically, the first space–time block code to provide full transmit diversity for systems with two transmit antennas. Tarokh later generalised Alamouti's transmit diversity scheme to an arbitrary number of transmit antennas and presented more complex STBCs based on orthogonal matrices. STBC achieve **Maximum Likelihood** (ML) decoding through **linear processing** at the receiver. Fig. 3.7 shows the block diagram of the Alamouti space-time encoder.

In each encoding operation, the encoder takes two symbols x_1 and x_2 and transmits according to the following orthogonal code matrix.

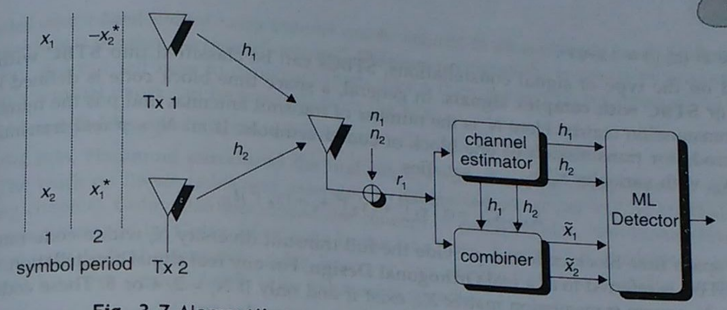


Fig. 3.7 Alamouti's two antenna transmit diversity scheme.

$$X = \begin{bmatrix} x_1 & -x_2^* \\ x_2 & x_1^* \end{bmatrix}, \tag{3.41}$$

where x_1^* is the complex conjugate of x_1. During symbol period 1, antenna 1 and antenna 2 transmit x_1 and x_2 respectively. During symbol period 2, antenna 1 and antenna 2 transmit $-x_2^*$ and x_1^* respectively. Note that

$$XX^H = (|x_1|^2 + |x_2|^2)I_2 \tag{3.42}$$

where I_2 is a 2 x 2 identity matrix. It is assumed that the fading channel coefficients h_1 and h_2 are constant across two consecutive symbol periods.

The received signals over two consecutive symbol periods, denoted by r_1 and r_2, can be expressed as

$$r_1 = h_1 x_1 + h_2 x_2 + n_1 \tag{3.43}$$

$$r_2 = -h_2 x_2^* + h_2 x_1^* + n_2 \tag{3.44}$$

where n_1 and n_2 are AWGN samples for the two symbol periods respectively.

The combiner combines the received signal as follows:

$$\tilde{x}_1 = h_1^* r_1 + h_2 r_2^* \tag{3.45}$$

$$\tilde{x}_2 = h_2^* r_1 - h_1 r_2^* \tag{3.46}$$

For equi energy M-ary (e.g. M–PSK) signal constellations, the ML decision rule can finally be simplified to

$$\hat{x}_1 = \arg\min_{\hat{x}_1 \in S} d^2(\tilde{x}_1, \hat{x}_1) \tag{3.47}$$

$$\hat{x}_2 = \arg\min_{\hat{x}_2 \in S} d^2(\tilde{x}_2, \hat{x}_2) \tag{3.48}$$

Scanned by CamScanner

where $d^2(x, y) = |x-y|^2$.

Based on the type of signal constellations, STBCs can be classified into **STBC with real signals** or **STBC with complex signals.** In general, a space time block code is defined by an $N_t \times p$ transmission matrix. Here N_t is the number of transmit antennas and p is the number of time periods for transmission of one block of coded symbols. If an $N_t \times p$ real transmission matrix X_{N_t}, with variables $x_1, x_2 \ldots x_k$, satisfies

$$X_{N_t}.X_{N_t}^T = c(|x_1|^2 + |x_2|^2 + \ldots + |x_k|^2)I_{N_t} \tag{3.49}$$

then the space time block code can provide the full transmit diversity N_t with a code rate k/p. Such an STBC is referred to as a real **Orthogonal Design**. For any real signal constellation, STBC with $N_t \times N_t$ square transmission matrix X_{N_t} exist if and only if $N_t = 2$, 4 or 8. These codes are full rate ($R = 1$) and offer full transmit diversity. Some examples are

$$X_2 = \begin{bmatrix} x_1 & -x_2 \\ x_2 & x_1 \end{bmatrix}, \tag{3.50}$$

for $N_t = 2$ transmit antennas and

$$X_4 = \begin{bmatrix} x_1 & -x_2 & -x_3 & -x_4 \\ x_2 & x_1 & x_4 & -x_3 \\ x_3 & -x_4 & x_1 & x_2 \\ x_4 & x_3 & -x_2 & x_1 \end{bmatrix}, \tag{3.51}$$

for $N_t = 4$ transmit antennas.

In general, for N_t transmit antennas, the minimum value of transmission periods p required to achieve full rate is given as follows:

$$\begin{array}{ll} N_t = 2, p = 2 & N_t = 6, p = 8 \\ N_t = 3, p = 4 & N_t = 7, p = 8 \\ N_t = 4, p = 4 & N_t = 8, p = 8\,. \\ N_t = 5, p = 8 & \end{array} \tag{3.52}$$

Like in Alamouti scheme, ML decoding is accomplished in orthogonal STBCs through linear processing at the receiver because of the orthogonality of the transmission matrices.

3.16 CONCLUDING REMARKS

The classic paper by Claude Elwood Shannon in the *Bell System Technical Journal* in 1948 gave birth to two important areas (i) information theory and (ii) coding theory. At that time Shannon was only 32 years old. According to Shannon's channel coding theorem, *"the error rate of data*

transmitted over a band limited noisy channel can be reduced to an arbitrarily small amount if the information rate is less than the channel capacity". Shannon predicted the existence of good channel codes, but did not tell how to construct them. Since then the search for good codes is on. Shannon's seminal paper can be accessed from the site:

http://cm.bell-labs.com/cm/ms/what/shannonday/paper.html.

In 1950, R.W. Hamming introduced the first single error-correcting code, which is still used today. The work on linear codes was extended by Golay (whose codes will be studied in the following chapter). Golay also introduced the concept of perfect codes. Non binary Hamming codes were developed by Golay and Cocke in the late 1950s. Lately a lot of computer searches have been used to find interesting codes, however some of the best known codes are the ones discovered by sheer genius, rather than exhaustive searches.

According to Shannon's theorem, if $C(p)$ represents the capacity (see chapter 1 for further details) of a binary symmetric channel with probability of bit error equal to p, then for arbitrarily low probability of symbol error we must have the code rate $R < C(p)$. Even though the channel capacity provides an upper bound on the achievable code rate ($R = k/n$), evaluating a code purely against channel capacity may be misleading. The block length of the code, which translates directly into the delay, is also an important parameter. Even if a code performs far from capacity, it may be possible that it is the best code possible of the same rate and length. It has been observed that as we increase the block length of the codes, the bounds on the code rate are closer to the channel capacity as opposed to the codes with smaller block lengths. However, longer block lengths imply longer delays for decoding. This is because we cannot begin the decoding of a codeword until we have receive the entire codeword. The maximum delay allowable is limited by practical constraints. For example, in mobile radio communications, packets of data are restricted to fewer than 200 bits. In these cases codewords with very large block lengths cannot be used.

Low Density Parity Check (LDPC) codes were introduced by Gallager in his doctoral thesis in 1963. They were rediscovered after three decades in 1990s and have become part of several wireless standards today. The use of multiple antennas is also rapidly becoming popular in wireless communications. Transmit diversity techniques are being currently integrated with 3G wireless standards. Space-time coding was introduced by Tarokh, Seshadri, and Calderbank in 1988. The Alamouti scheme presented in 1998 was the first space-time block code to provide full transmit diversity for systems with two transmit antennas.

SUMMARY

- A word is a sequence of symbols. A code is a set of vectors called codewords.
- The Hamming weight of a codeword (or any vector) is equal to the number of non-zero elements in the codeword. The Hamming weight of a codeword c is denoted by $w(c)$.
- A block code consists of a set of fixed length codewords. The fixed length of these codewords is called the block length and is typically denoted by n. A block coding scheme converts a

Scanned by CamScanner

block of k information symbols to n coded symbols. Such a code is denoted by (n, k).

- The code rate of a (n, k) code is defined as the ratio (k/n), and reflects the fraction of the codeword that consists of the information symbols.
- The minimum distance of a code is the minimum Hamming distance between any two codewords. A (n, k) code with minimum distance d^* is sometimes denoted by (n, k, d^*). The minimum weight of a code is the smallest weight of any non-zero codeword, and is denoted by w^*. For a linear code the minimum distance is equal to the minimum weight of the code, i.e., $d^* = w^*$.
- A linear code has the following properties:
  (i) The sum of two codewords belonging to the code is also a codeword belonging to the code.
  (ii) The all-zero codeword is always a codeword.
  (iii) The minimum Hamming distance between two codewords of a linear code is equal to the minimum weight of any non-zero codeword, i.e., $d^* = w^*$.
- The generator matrix converts (encodes) a vector of length k to a vector of length n. Let the input vector (uncoded symbols) be represented by i. The coded symbols will be given by

$$c = iG.$$

- Two q-ary codes are called equivalent if one can be obtained from the other by one or both the operations listed below:
  (i) permutation of the components, (ii) permutation of the position of the code word.
- A (n, k) systematic code is one in which the first k symbols of the codeword of block length n are the information symbols themselves. A generator matrix of the form $G = [\, I \mid P \,]$ is called the systematic form or the standard form of the generator matrix, where I is a $k \times k$ identity matrix and P is a $k \times (n\text{-}k)$ matrix.
- The parity check matrix, H, for the given code satisfies $cH^T = 0$, where c is a valid codeword. Since $c = iG$, therefore, $iGH^T = 0$. The parity check matrix is not unique for a given code.
- A maximum distance code satisfies $d^* = n - k + 1$.
- For a code to be able to correct up to t errors, we must have $d^* \geq 2t + 1$, where d^* is minimum distance of the code.
- Let C be an (n, k) code over $GF(q)$ and a be any vector of length n. Then the set $a + C = \{a + x \mid x \in C\}$ is called a coset (or translate) of C. a and b are said to be in the same coset if $(a - b) \in C$.
- Suppose H is a parity check matrix of an (n, k) code. Then for any vector $v \in GF(q^n)$, the vector, $s = vH^T$ is called the syndrome of v. It is called a syndrome because it gives us the symptoms of the error, thereby helping us to diagnose the error.
- A perfect code achieves the Hamming bound, *i.e.*,

$$M\left\{\binom{n}{0}+\binom{n}{1}(q-1)+\binom{n}{2}(q-1)^2+\ldots+\binom{n}{t}(q-1)^t\right\} = q^n.$$

- The binary Hamming codes have the property that $(n, k) = (2^m - 1, 2^m - 1 - m)$, where m is any positive integer. Hamming codes are perfect codes.
- An (r, s) Gallager code is a linear code with a check matrix H, satisfying the condition that every column has r ones and every row has s ones. Gallager code with small values of r and s is a Low Density Parity Check (LDPC) code.
- A Tanner graph is a graphical representation of the linear code based on the set of check equations. It is a bipartite graph, which means it has two kinds of nodes: symbol nodes and check nodes. Each of the symbol nodes are connected only to check nodes and each check node is connected only to symbol nodes.
- For a (n, k, d^*) optimal code, no $(n - 1, k, d^*)$, $(n + 1, k + 1, d^*)$ or $(n + 1, k, d^* + 1)$ code exists.
- A $(n, n - r, r + 1)$ code is called a maximum distance separable (MDS) code. An MDS code is a linear code of redundancy r, whose minimum distance is equal to $r + 1$.
- Space-time codes are coding techniques designed for multiple antenna transmissions. In particular, coding is performed by adding properly designed redundancy in both spatial and temporal domains.
- The orthogonal code matrix of the Alamouti scheme is given by $X = \begin{bmatrix} x_1 & -x_2^* \\ x_2 & x_1^* \end{bmatrix}$.

> *The greatest unsolved theorem in mathematics is why some people are better at it than others.*
>
> *Mathesis, Adrian*

PROBLEMS

3.1 Show that $C = \{0000, 1100, 0011, 1111\}$ is a linear code. What is its minimum distance?

3.2 Construct, if possible, binary (n, k, d^*) codes with the following parameters:
(i) (6, 1, 6), (ii) (3, 3, 1), (iii) (4, 3, 2).

3.3 Consider the following generator matrix over $GF(2)$

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

(i) Generate all possible codewords using this matrix.
(ii) Find the parity check matrix, H.
(iii) Find the generator matrix of an equivalent systematic code.

Scanned by CamScanner

(iv) Construct the standard array for this code.
(v) What is the minimum distance of this code?
(vi) How many errors can this code detect?
(vii) Write down the set of error patterns this code can detect.
(viii) How many errors can this code correct?
(ix) What is the probability of symbol error if we use this encoding scheme? Compare it with the uncoded probability of error.
(x) Is this a linear code?

3.4 For the code $C = \{00000, 10101, 01010, 11111\}$ construct the generator matrix. Since this G is not unique, suggest another generator matrix that can also generate this set of codewords.

3.5 Show that if there is a binary (n, k, d) code with d even, then there exists a binary (n, k, d) code in which all codewords have even weight.

3.6 Show that if C is a binary linear code, then the code obtained by adding an overall parity check bit to C is also linear.

3.7 Let us denote vectors $\boldsymbol{u} = u_1\, u_2\, u_3 \ldots u_n$ and $\boldsymbol{v} = v_1\, v_2\, v_3 \ldots v_n$. Denote $(\boldsymbol{u} \mid \boldsymbol{v}) = u_1\, u_2\, u_3 \ldots u_n\, v_1\, v_2\, v_3 \ldots v_n$ as a vector of length $2n$. Next, suppose C_1 is a binary (n, k_1, d_1) code and C_2 is a binary (n, k_2, d_2) code with codewords $\boldsymbol{u} \in C_1$ and $\boldsymbol{v} \in C_2$, where d_1 and d_2 represent the minimum distances of the codes respectively. Form a new codeword c_3 as $(\boldsymbol{u} \mid \boldsymbol{u}+\boldsymbol{v})$.
(i) Show that C_3 is a $(2n, k_1+k_2)$ code.
(ii) What is the minimum distance of C_3?

3.8 For each of the following sets S, list the code $\langle S \rangle$.
(i) $S = \{0101, 1010, 1100\}$, (ii) $S = \{1000, 0100, 0010, 0001\}$, (iii) $S = \{11000, 01111, 11110, 01010\}$.

3.9 Consider the (23, 12, 7) binary code. Show that if its used over a binary symmetric channel (BSC) with probability of bit error $p = 0.01$, the word error will be approximately 0.00008.

3.10 Suppose C is a binary code with parity check matrix, H. Show that the extended code C_e obtained from C by adding an overall parity bit, has the parity check matrix

$$H_1 = \left[\begin{array}{ccccc|c} & & & & & 0 \\ & & & & & 0 \\ & H & & & & \\ & & & & & 0 \\ \hline 1 & 1 & \ldots & 1 & & 1 \end{array}\right].$$

3.11 For a (5, 3) code over $GF(4)$, the generator matrix is given by

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 2 \\ 0 & 0 & 1 & 1 & 3 \end{bmatrix}.$$

(i) Find the parity check matrix for this code.
(ii) How many errors can this code detect?
(iii) How many errors can this code correct?
(iv) How many erasures can this code correct?
(v) Is this a perfect code?

3.12 Consider a linear block code over $GF(11)$ with block length $n = 10$, satisfying the following two constraints: $\sum_{i=0}^{9} i c_i = 0$ calculated (mod 11) and $\sum_{i=0}^{9} (10-i) c_i = 0$ calculated (mod 11). Find the minimum distance of this code.

3.13 Let C be a binary perfect code of length n with minimum distance 7. Show that $n = 7$ or $n = 23$.

3.14 Let r_H denote the code rate for the binary Hamming code. Determine $\lim_{k \to \infty} r_H$.

3.15 Show that a (15, 8, 5) code does not exist.

3.16 Check whether the following STBC code is orthogonal.

$$X = \begin{bmatrix} x_1 & -x_2^* & -x_3^* & 0 \\ x_2 & x_1^* & 0 & -x_3^* \\ x_3 & 0 & x_1^* & x_2^* \end{bmatrix}.$$

COMPUTER PROBLEMS

3.17 Write a computer program to find the minimum distance of a linear block code over $GF(2)$, given the generator matrix for the code.

3.18 Generalise the above program to find the minimum distance of any linear block code over $GF(q)$.

3.19 Write a computer program to exhaustively search for all the perfect code parameters n, q, M and t in the equation for the Hamming bound. Search for $1 \geq n \geq 200$, $2 \geq q \geq 11$.

3.20 Write a computer program for a universal binary Hamming encoder with rate $\dfrac{2^m - 1}{2^m - 1 - m}$.

The program should take as input the value of m and a bit stream to be encoded. It should then generate an encoded bit stream. Develop a program for the decoder also. Now, perform the following tasks:

(i) Write an error generator module that takes in a bit stream and outputs another bit stream after inverting every bit with probability p, i.e., the probability of a bit error is p.

Scanned by CamScanner

(ii) For $m = 3$, pass the Hamming encoded bit stream through the above mentioned module and then decode the received words using the decoder block.

(iii) Plot the residual error probability (the probability of error after decoding) as a function of p. Note that if you are working in the range of BER = 10^{-r}, you must transmit on the order of 10^{r+2} bits (why?).

(iv) Repeat your simulations for $m = 5$, 8 and 15. What happens as $m \to \infty$.

3.21 Write a computer program to generate a (20, 3, 4) LDPC code using the: (i) random construction method, (ii) algebraic construction method. Compare the two.

3.22 Write a computer program to search for orthogonal STBCs with $N_t > 2$.

Cyclic Codes

4

We arrive at truth, not by reason only, but also by the heart.

Pascal, Blaise (1623–1662)

4.1 INTRODUCTION TO CYCLIC CODES

In the previous chapter, while dealing with Linear Block codes, certain linearity constraints were imposed on the structure of the block codes. These structural properties help us to search for good linear block codes that are fast and easy to encode and decode. In this chapter, we shall explore a subclass of linear block codes which has another constraint on the structure of the codes. The additional constraint is that any cyclic shift of a codeword results in another valid codeword. This condition allows for a very simple implementation of these cyclic codes by using shift registers. Efficient circuit implementation is a selling feature of any error control code. We shall also see that the theory of Galois Field can be used effectively to study, analyse and discover new cyclic codes. The Galois Field representation of cyclic codes leads to low complexity encoding and decoding algorithms.

This chapter is organised as follows: In the first three sections we shall take a mathematical detour to **Polynomials**. We will review some of our old concepts and learn about a few new ones. Then we will use these mathematical tools to construct and analyse **Cyclic Codes**. The **Matrix Description** of cyclic will be introduced next, followed by a brief discussion on **Quasi Cyclic Codes** and **Shortened Cyclic Codes.** We will then study some of the popular codes that are cyclic. The chapter will conclude with a discussion of circuit implementation of cyclic codes.

Scanned by CamScanner

Definition 4.4 Let $f(x)$ be a fixed polynomial in $F[x]$. Two polynomials, $g(x)$ and $h(x)$ in $F[x]$ are said to be **Congruent Modulo** $f(x)$, depicted by $g(x) \equiv h(x) \pmod{f(x)}$, if $g(x) - h(x)$ is divisible by $f(x)$.

Example 4.6 Let the polynomials $g(x) = x^9 + x^2 + 1$, $h(x) = x^5 + x^2 + 1$ and $f(x) = x^4 + 1$ be defined over $GF(2)$. Since $g(x) - h(x) = x^5 f(x)$, we can write $g(x) \equiv h(x) \pmod{f(x)}$.

Next, let us denote $F[x]/f(x)$ as the set of polynomials in $F[x]$ of degree less than deg $f(x)$, with addition and multiplication carried out modulo $f(x)$ as follows:

(i) If $a(x)$ and $b(x)$ belong to $F[x]/f(x)$, then the sum $a(x) + b(x)$ in $F[x]/f(x)$ is the same as in $F[x]$. This is because deg $a(x) <$ deg $f(x)$, deg $b(x) <$ deg $f(x)$ and therefore deg $(a(x) + b(x)) <$ deg $f(x)$.

(ii) The product $a(x)b(x)$ is the unique polynomial of degree less than deg $f(x)$ to which $a(x)b(x)$ (multiplication being carried out in $F[x]$) is congruent modulo $f(x)$.

$F[x]/f(x)$ is called the **ring of polynomials** (over $F[x]$) modulo $f(x)$. As mentioned earlier in Definition 3.9, a ring satisfies the first seven of the eight axioms that define a field. A ring in which every element also has a multiplicative inverse forms a field.

Example 4.7 Consider the product $(x + 1)^2$ in $F[x]/(x^2 + x + 1)$ defined over $GF(2)$.

$(x + 1)^2 = x^2 + x + x + 1 = x^2 + 1 = x \pmod{x^2 + x + 1}$.

The product $(x + 1)^2$ in $F[x]/(x^2 + 1)$ defined over $GF(2)$ can be expressed as

$(x + 1)^2 = x^2 + x + x + 1 = x^2 + 1 = 0 \pmod{x^2 + x + 1}$.

The product $(x + 1)^2$ in $F[x]/(x^2 + x + 1)$ defined over $GF(3)$ can be expressed as

$(x + 1)^2 = x^2 + x + x + 1 = x^2 + 2x + 1 = x \pmod{x^2 + x + 1}$.

If $f(x)$ has degree n, then the ring $F[x]/f(x)$ over $GF(q)$ consists of polynomials of degree $\leq n - 1$. The size of ring will be q^n because each of the n coefficients of the polynomials can be one of the q elements in $GF(q)$.

Example 4.8 Consider the ring $F[x]/(x^2 + x + 1)$ defined over $GF(2)$. This ring will have polynomials with highest degree = 1. This ring contains $q^n = 2^2 = 4$ elements (each element is a polynomial). The elements of the ring will be 0, 1, x and $x + 1$. The addition and multiplication tables can be written as follows.

Table 4.1 Addition and multiplication tables for the field $F[x]/(x^2 + x + 1)$.

+	0	1	x	$x+1$
0	0	1	x	$x+1$
1	1	0	$x+1$	x
x	x	$x+1$	0	1
$x+1$	$x+1$	x	1	0

·	0	1	x	$x+1$
0	0	0	0	0
1	0	1	x	$x+1$
x	0	x	$x+1$	1
$x+1$	0	$x+1$	1	x

Next, consider $F[x]/(x^2 + 1)$ defined over $GF(2)$. The elements of the ring will be 0, 1, x and $x + 1$. The addition and multiplication tables can be written as follows.

Table 4.2 Addition and multiplication tables for the ring $F[x]/(x^2 + 1)$.

+	0	1	x	$x+1$
0	0	1	x	$x+1$
1	1	0	$x+1$	x
x	x	$x+1$	0	1
$x+1$	$x+1$	x	1	0

·	0	1	x	$x+1$
0	0	0	0	0
1	0	1	x	$x+1$
x	0	x	1	$x+1$
$x+1$	0	$x+1$	$x+1$	0

It is interesting to note that $F[x]/(x^2 + x + 1)$ is actually a field as the multiplicative inverse for all the non-zero elements also exists. On the other hand, $F[x]/(x^2 + 1)$ is not a field because the multiplicative inverse of element $x + 1$ does not exist.

It is worthwhile exploring the properties of $f(x)$ which makes $F[x]/f(x)$ a field. As we shall shortly find out, the polynomial $f(x)$ must be *irreducible* (*nonfactorisable*).

Definition 4.5 A polynomial $f(x)$ in $F[x]$ is said to be **Reducible** if $f(x) = a(x)\, b(x)$, where $a(x)$, $b(x)$ are elements of $f(x)$ and deg $a(x)$ and deg $b(x)$ are both smaller than deg $f(x)$. If $f(x)$ is not reducible, it is called **Irreducible**. A monic irreducible polynomial of degree at least one is called a **Prime Polynomial**.

It is helpful to compare a reducible polynomial with a positive integer that can be factorised into a product of prime numbers. Any monic polynomial in $f(x)$ can be factorised uniquely into a product of irreducible monic polynomials (prime polynomials). One way to verify a prime polynomial is by trail and error, testing for all possible factorisations. This would require a computer search. Prime polynomials of every degree exist over every Galois field.

Theorem 4.1

(i) A polynomial $f(x)$ has a linear factor $(x - a)$ if and only if $f(a) = 0$ where a is a field element.

(ii) A polynomial $f(x)$ in $F[x]$ of degree 2 or 3 over $GF(q)$ is irreducible if and only if $f(a) \neq 0$ for all a in $GF(q)$.

(iii) Over any field, $x^n - 1 = (x - 1)(x^{n-1} + x^{n-2} + \ldots + x + 1)$. The second factor may be further reducible.

Proof

(i) If $f(x) = (x - a)g(x)$, then obviously $f(a) = 0$. On the other hand if $f(a) = 0$, by division algorithm, $f(x) = q(x)(x - a) + r(x)$, where deg $r(x) <$ deg $(x - a) = 1$. This implies that $r(x)$ is a constant. But, since $f(a) = 0$, $r(x)$ must be zero, and therefore, $f(x) = q(x)(x - a)$.

(ii) A polynomial of degree 2 or 3 over $GF(q)$ will be reducible if and only if it has at least one linear factor. The result (ii) then directly follows from (i). This result does not necessarily

Scanned by CamScanner

hold for polynomials of degree more than 3. This is because it might be possible to factorise a polynomial of degree 4 or higher into a product of polynomials, none of which are linear, i.e., of the type $(x - a)$.

(iii) From (i), $(x - 1)$ is a factor of $(x^n - 1)$. By carrying out long division of $(x^n - 1)$ by $(x - 1)$ we obtain $(x^{n-1} + x^{n-2} + \ldots + x + 1)$.

Example 4.9 Consider $f(x) = x^3 - 1$ over $GF(2)$. Using (iii) of theorem 4.1 we can write $x^3 - 1 = (x - 1)(x^2 + x + 1)$. This factorisation is true over any field. Now, lets try to factorise the second term, $p(x) = (x^2 + x + 1)$.

$p(0) = 0 + 0 + 1 = 1$, over $GF(2)$,

$p(1) = 1 + 1 + 1 = 1$, over $GF(2)$.

Therefore, $p(x)$ cannot be factorised further (from (ii)).

Thus, over $GF(2)$, $x^3 - 1 = (x - 1)(x^2 + x + 1)$.

Next, consider $f(x) = x^3 - 1$ over $GF(3)$.

$x^3 - 1 = (x - 1)(x^2 + x + 1)$. Again, let $p(x) = (x^2 + x + 1)$.

$p(0) = 0 + 0 + 1 = 1$, over $GF(3)$,

$p(1) = 1 + 1 + 1 = 0$, over $GF(3)$.

$p(2) = 2.2 + 2 + 1 = 1 + 2 + 1 = 1$, over $GF(3)$,

Since, $p(1) = 0$, from (i) we have $(x - 1)$ as a factor of $p(x)$.

Thus, over $GF(3)$, $x^3 - 1 = (x - 1)(x - 1)(x - 1)$.

Theorem 4.2 The ring $F[x]/f(x)$ is a field if and only if $f(x)$ is a prime polynomial in $F[x]$.

Proof To prove that a ring is a field we must show that every non-zero element of the ring has a multiplicative inverse. Let $s(x)$ be a non-zero element of the ring.

We have, $\deg s(x) < \deg f(x)$, because $s(x)$ is contained in the ring $F[x]/f(x)$.

It can be shown that the greatest common divisor (GCD) of two polynomials $f(x)$ and $s(x)$ can be expressed as:

$$GCD(f(x), s(x)) = a(x)\, f(x) + b(x)\, s(x),$$

where $a(x)$ and $b(x)$ are polynomials over $GF(q)$. Since $f(x)$ is irreducible in $F[x]$, we have $GCD(f(x), s(x)) = 1 = a(x)\, f(x) + b(x)\, s(x)$.

$$\begin{aligned}
\text{Now, } 1 = R_{f(x)}[1] &= R_{f(x)}[\, a(x)\, f(x) + b(x)\, s(x)] \\
&= R_{f(x)}[\, a(x)\, f(x)] + R_{f(x)}[b(x)\, s(x)] \quad \text{(property (i) of residues)} \\
&= 0 + R_{f(x)}[b(x)\, s(x)] \\
&= R_{f(x)}\{R_{f(x)}[b(x)].R_{f(x)}[s(x)]\} \quad \text{(property (ii) of residues)} \\
&= R_{f(x)}\{R_{f(x)}[b(x)].s(x)\}
\end{aligned}$$

Hence, $R_{f(x)}[b(x)]$ is the multiplicative inverse of $s(x)$.

Next, let us prove the *only if* part of the theorem. Let us suppose $f(x)$ has a degree of at least 2, and is not a prime polynomial (a polynomial of degree one is always irreducible). Therefore we can write $f(x) = r(x)\, s(x)$ for some polynomials $r(x)$ and $s(x)$ with degrees at least one. If the ring $F[x]/f(x)$ is indeed a field, then a multiplicative index of $r(x)$, $r^{-1}(x)$ exists, since all polynomials in the field must have their corresponding multiplicative inverses. Hence, $s(x) = R_{f(x)}\{s(x)\} = R_{f(x)}\{r(x)r^{-1}(x)s(x)\} = R_{f(x)}\{r^{-1}(x)r(x)s(x)\} = R_{f(x)}\{r^{-1}(x)f(x)\} = 0$. However, we had assumed $s(x) \neq 0$. Thus, there is a contradiction, implying that the ring is not a field.

Note that a prime polynomial is both monic and irreducible. In the above theorem, it is sufficient to have $f(x)$ irreducible in order to obtain a field. The theorem could as well been stated as: "The ring $F[x]/f(x)$ is a field if and only if $f(x)$ is irreducible in $F[x]$".

So, now we have an elegant mechanism of generating Galois Fields. If we can identify a prime polynomial of degree n over $GF(q)$, we can construct a Galois Field with q^n elements. Such a field will have polynomials as the elements of the field. These polynomials will be defined over $GF(q)$ and consist of all polynomials of degree less than n. It can be seen that there will be q^n such polynomials, which form the elements of the field.

Example 4.10 Consider the polynomial $p(x) = x^3 + x + 1$ over $GF(2)$. Since, $p(0) \neq 0$ and $p(1) \neq 0$, the polynomial is irreducible in $GF(2)$. Since it is also monic, $p(x)$ is a prime polynomial. Here we have $n = 3$, so we can use $p(x)$ to construct a field with $2^3 = 8$ elements, i.e., $GF(8)$. The elements of this field will be $0, 1, x, x + 1, x^2, x^2 + 1, x^2 + x, x^2 + x + 1$, which are all possible polynomials of degree less than $n = 3$. It is easy to construct the addition and multiplication tables for this field (exercise).

Having developed the necessary mathematical tools, we now resume our study of cyclic codes. We now fix $f(x) = x^n - 1$ for the remainder of the chapter. We also denote $F[x]/f(x)$ by R_n. Before we proceed, we make the following observations:

(i) $x^n = 1 \pmod{x^n - 1}$. Hence, any polynomial, modulo $x^n - 1$, can be reduced simply by replacing x^n by 1, x^{n+1} by x and so on.

(ii) A codeword can uniquely be represented by a polynomial. A codeword consists of a sequence of elements. We can use a polynomial to represent the locations and the values of all the element in the codeword. For example, the codeword $c_1c_2 \ldots c_n$ can be represented by the polynomial $c(x) = c_0 + c_1x + c_2x^2 + \ldots\ldots c_nx^n$. As another example, the codeword over $GF(8)$, $c = 207735$ can be represented by the polynomial $c(x) = 2 + 7x^2 + 7x^3 + 3x^4 + 5x^5$.

(iii) Multiplying any polynomial by x corresponds to a single cyclic rightshift of the codeword elements. More explicitly, in R_n, by multiplying $c(x)$ by x we get

$$x.\, c(x) = c_0x + c_1\, x^2 + c_2x^3 + \ldots\ldots c_nx^{n+1} = c_n + c_0x + c_1x^2 + c_2x^3 + \ldots\ldots c_{n-1}x^n.$$

Theorem 4.3 A code C in R_n is a cyclic code if and only if C satisfies the following conditions:

(i) $a(x), b(x) \in C \Rightarrow a(x) + b(x) \in C$, (4.4)

(ii) $a(x) \in C$ and $r(x) \in R_n \Rightarrow a(x)r(x) \in C$. (4.5)

Scanned by CamScanner

Proof

(i) Suppose C is a cyclic code in R_n. Since cyclic codes are a subset of linear block codes, the first condition holds.

(ii) Let $r(x) = r_0 + r_1x + r_2x^2 + \ldots r_nx^n$. Multiplication by x corresponds to a cyclic right-shift. But, by definition, the cyclic shift of a cyclic codeword is also a valid codeword. That is, $x.a(x) \in C$, $x.(xa(x)) \in C$, and so on. Hence

$$r(x)a(x) = r_0a(x) + r_1xa(x) + r_2x^2a(x) + \ldots r_nx^na(x)$$ is also in C since each sum is also in C.

Next, we prove the only if part of the theorem. Suppose (i) and (ii) hold. Take $r(x)$ to be a scalar. Then (i) implies that C is linear. Take $r(x) = x$ in (ii), which shows that any cyclic shift also leads to a codeword. Hence (i) and (ii) imply that C is a cyclic code.

In the next section, we shall use the mathematical tools developed so far to construct cyclic codes.

4.4 A METHOD FOR GENERATING CYCLIC CODES

The following steps can be used to generate a cyclic code:

(i) Take a polynomial $f(x)$ in R_n.

(ii) Obtain a set of polynomials by multiplying $f(x)$ by all possible polynomials in R_n.

(iii) The set of polynomials obtained above corresponds to the set of codewords belonging to a cyclic code. The block length of the code is n.

Example 4.11 Consider the polynomial $f(x) = 1 + x^2$ in R_3 defined over $GF(2)$. In general a polynomial in R_3 ($= F[x]/(x^3 - 1)$) can be represented as $r(x) = r_0 + r_1x + r_2x^2$, where the coefficients can take the values 0 or 1 (since defined over $GF(2)$). Thus, there can be a total of $2 \times 2 \times 2 = 8$ polynomials in R_3 defined over $GF(2)$, which are $0, 1, x, x^2, 1 + x, 1 + x^2, x + x^2, 1 + x + x^2$. To generate the cyclic code, we multiply $f(x)$ with these 8 possible elements of R_3 and then reduce the results modulo $(x^3 - 1)$:

$(1 + x^2).0 = 0$, $(1 + x^2).1 = (1 + x^2)$, $(1 + x^2).x = 1 + x$, $(1 + x^2).\ x^2 = x + x^2$,

$(1 + x^2).\ (1 + x) = x + x^2$, $(1 + x^2).\ (1 + x^2) = 1 + x$, $(1 + x^2).(x + x^2) = (1 + x^2)$,

$(1 + x^2).\ (1 + x + x^2) = 0$.

Thus there are only four distinct codewords: $\{0, 1 + x, 1 + x^2, x + x^2\}$ which correspond to $\{000, 110, 101, 011\}$.

From the above example it appears that we can have some sort of a **generator polynomial** which can be used to construct the cyclic code.

Theorem 4.4 Let C be a (n, k) non-zero cyclic code in R_n. Then

(i) there exists a unique monic polynomial $g(x)$ of the smallest degree in C,

(ii) the cyclic code C consists of all multiples of the generator polynomial $g(x)$ by polynomials of degree $k - 1$ or less.

(iii) $g(x)$ is a factor of $x^n - 1$.

Proof

(i) Suppose both $g(x)$ and $h(x)$ are monic polynomials in C of smallest degree. Then $g(x) - h(x)$ is also in C, and has a smaller degree. If $g(x) \neq h(x)$, then a suitable scalar multiplier of $g(x) - h(x)$ is monic, and is in C, and is of smaller degree than $g(x)$. This gives a contradiction.

(ii) Let $a(x) \in C$. Then, by division algorithm, $a(x) = q(x)g(x) + r(x)$, where deg $r(x) <$ deg $g(x)$. But $r(x) = a(x) - q(x)g(x) \in C$ because both word on the right-hand side of the equation are codewords. However, the degree of $g(x)$ must be the minimum among all codewords. This can only be possible if $r(x) = 0$ and $a(x) = q(x)g(x)$. Thus, a codeword is obtained by multiplying the generator polynomial $g(x)$ with the polynomial $q(x)$. For a code defined over $GF(q)$, here are q^k distinct codewords possible. These codewords correspond to multiplying $g(x)$ with the q^k distinct polynomials, $q(x)$, where deg $q(x) \leq (k - 1)$.

(iii) By the division algorithm, $x^n - 1 = q(x)g(x) + r(x)$, where deg $r(x) <$ deg $g(x)$. Or, $r(x) = \{(x^n - 1) - q(x)g(x)\}$ modulo $(x^n - 1) = -q(x)g(x)$. But $-q(x)g(x) \in C$ because we are multiplying the generator polynomial by another polynomial $-q(x)$. Thus, we have a codeword $r(x)$ whose degree is less than that of $g(x)$. This violates the minimality of the degree of $g(x)$, unless $r(x) = 0$. Which implies $x^n - 1 = q(x)g(x)$, i.e., $g(x)$ is a factor of $x^n - 1$.

The last part of the theorem gives us the recipe to obtain the generator polynomial for a cyclic code. All we have to do is to factorise $x^n - 1$ into irreducible, monic polynomials. We can find all the possible cyclic codes of blocklength n simply by combining the factors of $x^n - 1$.

Note 1: A cyclic code C may contain polynomials other than the generator polynomial which also generate C. But the polynomial with the minimum degree is called the generator polynomial.

Note 2: The degree of $g(x)$ is $n - k$ (this will be shown later in section 4.5).

Example 4.12 To find all the binary codes of block length 3, we first factorise $x^3 - 1$. Note that for $GF(2)$, $1 = -1$, since $1 + 1 = 0$. Hence, $x^3 - 1 = x^3 + 1 = (x + 1)(x^2 + x + 1)$. Thus, we can construct Table 4.3.

Table 4.3 All possible binary cyclic codes of block length 3.

Generator Polynomial	*Code (polynomial)*	*Code (binary)*
1	$\{R_3\}$	{000, 001, 010, 011, 100, 101, 110, 111}
$(x + 1)$	$\{0, x + 1, x^2 + x, x^2 + 1\}$	{000, 011, 110, 101}
$(x^2 + x + 1)$	$\{0, x^2 + x + 1\}$	{000, 111}
$(x^3 + 1) = 0$	$\{0\}$	{000}

A simple encoding rule to generate the codewords from the generator polynomial is

$$c(x) = i(x)g(x), \tag{4.6}$$

Scanned by CamScanner

where $i(x)$ is the information polynomial, $c(x)$ is the codeword polynomial and $g(x)$ is the generator polynomial. We have seen already that there is a one-to-one correspondence between a word (vector) and a polynomial. The error vector can be also represented as the error polynomial, $e(x)$. Thus, the received word at the receiver, after passing through a noisy channel can be expressed as

$$v(x) = c(x) + e(x). \tag{4.7}$$

We define the **Syndrome Polynomial**, $s(x)$ as the remainder of $v(x)$ under division by $g(x)$, i.e.,

$$s(x) = R_{g(x)}[v(x)] = R_{g(x)}[\,c(x) + e(x)] = R_{g(x)}[c(x)] + R_{g(x)}[e(x)] = R_{g(x)}[e(x)] \tag{4.8}$$

because $R_{g(x)}[c(x)] = 0$ from (4.6).

Example 4.13 Consider the generator polynomial $g(x) = x^2 + 1$ for ternary cyclic codes (i.e., over $GF(3)$) of block length $n = 4$. Here we are dealing with cyclic codes, therefore, the highest power of $g(x)$ is $n - k$. Since $n = 4$, k must be 2. So, we are going to construct a (4, 2) cyclic ternary code. There will be a total of $q^k = 3^2 = 9$ codewords. The information polynomials and the corresponding codeword polynomials are listed in Table 4.4.

Table 4.4 Ternary cyclic code constructed using $g(x) = x^2 + 1$.

i	$i(x)$	$c(x) = i(x)g(x)$	c
00	0	0	0000
01	1	$x^2 + 1$	0101
02	2	$2x^2 + 2$	0202
10	x	$x^3 + x$	1010
11	$x + 1$	$x^3 + x^2 + x + 1$	1111
12	$x + 2$	$x^3 + 2x^2 + x + 2$	1212
20	$2x$	$2x^3 + 2x$	2020
21	$2x + 1$	$2x^3 + x^2 + 2x + 1$	2121
22	$2x + 2$	$2x^3 + 2x^2 + 2x + 2$	2222

It can be seen that the cyclic shift of any codeword results in another valid codeword. By observing the codewords we find that the minimum distance of this code is 2 (there are four non zero codewords with the minimum Hamming weight = 2). Therefore, this code is capable of detecting one error and correcting zero errors.

Observing the fact that the codeword polynomial is divisible by the generator polynomial, we can *detect* more number of errors than suggested by the minimum distance of the code. Since we are dealing with cyclic codes that are a subset of linear block codes, we can use the all-zero codewords to illustrate this point without loss of generality.

Assume that $g(x) = x^2 + 1$ and the transmitted codeword is the all-zero codeword. Therefore, the received word is the error polynomial, i.e.,

$$v(x) = c(x) + e(x) = e(x). \tag{4.9}$$

At the receiver end, an error will be detected if $g(x)$ fails to divide the received word $v(x) = e(x)$. Now, $g(x)$ has only two terms. So if the $e(x)$ has odd number of terms, i.e., of the number of errors are odd, it will be caught by the decoder. For example, if we try to divide $e(x) = x^3 + x + 1$ by $g(x)$, we will always get a remainder. In the case of the (4, 2) cyclic code with $g(x) = x^2 + 1$, the $d^* = 2$, suggesting that it can detect $d^* - 1 = 1$ error. However, by this simple observation we find that it can detect any odd number of errors $\le n$. In this case it can detect 1 error or 3 errors, but not 2 errors.

4.5 MATRIX DESCRIPTION OF CYCLIC CODES

Theorem 4.5 Suppose C is a cyclic code with generator polynomial $g(x) = g_0 + g_1x + \ldots + g_rx^r$ of degree r. Then the generator matrix of C is given by

$$G = \underbrace{\begin{bmatrix} g_0 & g_1 & \cdots & g_r & 0 & 0 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & g_r & 0 & 0 & \cdots & 0 \\ 0 & 0 & g_0 & g_1 & \cdots & g_r & 0 & \cdots & 0 \\ \vdots & \vdots & & & & & & & \vdots \\ 0 & 0 & 0 & 0 & 0 & g_0 & g_1 & \cdots & g_r \end{bmatrix}}_{n \text{ columns}} \Bigg\updownarrow k = (n-r) \text{ rows} \tag{4.10}$$

Proof The $(n - r)$ rows of the matrix are obviously linearly independent because of the echelon form of the matrix. These $(n - r)$ rows represent the codewords $g(x)$, $xg(x)$, $x^2g(x)$, ..., $x^{n-r-1}g(x)$. Thus, the matrix can generate these codewords. Now, to prove that the matrix can generate all the possible codewords, we must show that every possible codeword can be represented as linear combinations of the codewords $g(x)$, $xg(x)$, $x^2g(x)$, ..., $x^{n-r-1}g(x)$.

We know that, if $c(x)$ is a codeword, it can be represented as

$$c(x) = q(x).g(x)$$

for some polynomial $q(x)$.

Since the degree of $c(x) < n$ (because the length of the codeword is n), it follows that the degree of $q(x) < n - r$.

Hence,

$$q(x).g(x) = (q_0 + q_1x + \ldots + q_{n-r-1}x^{n-r-1})g(x) = q_0g(x) + q_1xg(x) + \ldots + q_{n-r-1}x^{n-r-1}g(x).$$

Thus, any codeword can be represented as a linear combination of

$g(x)$, $xg(x)$, $x^2g(x)$, ..., $x^{n-r-1}g(x)$. This proves that the matrix G is indeed the generator matrix.

We also know that the dimensions of the generator matrix is $k \times n$. Therefore, $r = n - k$, i.e., the degree of $g(x)$ is $n - k$.

Scanned by CamScanner

Example 4.14 To find the generator matrices of all ternary codes (i.e., codes over GF(3)) of block length $n = 4$, we first factorise $x^4 - 1$.

$$x^4 - 1 = (x - 1)(x^3 + x^2 + x + 1) = (x - 1)(x + 1)(x^2 + 1).$$

Thus all the divisors of $x^4 - 1$ are: $1, (x - 1), (x + 1), (x^2 + 1), (x - 1).(x + 1) = (x^2 - 1), (x - 1).(x^2 + 1) = (x^3 - x^2 + x - 1), (x + 1).(x^2 + 1) = (x^3 + x^2 + x + 1)$ and $(x^4 - 1)$. We know that all these factors of $x^4 - 1$ are capable of generating a cyclic code.

The corresponding generator matrices are listed in Table 4.5. Note that -1 = 2 for GF(3).

Table 4.5 Cyclic codes of block length $n = 4$ over $GF(3)$.

$g(x)$	(n, k)	d^*	G
1	$(4, 4)$	1	$[I_4]$
$(x - 1)$	$(4, 3)$	2	$\begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}$
$(x + 1)$	$(4, 3)$	2	$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$
$(x^2 + 1)$	$(4, 2)$	2	$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}$
$(x^2 - 1)$	$(4, 2)$	2	$\begin{bmatrix} -1 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix}$
$(x^3 - x^2 + x - 1)$	$(4, 1)$	4	$[-1\ 1\ -1\ 1]$
$(x^3 + x^2 + x + 1)$	$(4, 1)$	4	$[1\ 1\ 1\ 1]$
$(x^4 - 1)$	$(4, 1)$	0	$[0\ 0\ 0\ 0]$

It can be seen from Table 4.5 that none of the (4, 2) non-trivial ternary cyclic codes are single error correcting codes (since their minimum distance is less than 3). An interesting observation is that we do not have any ternary (4, 2) Hamming code that is cyclic. Remember, Hamming codes are single error correcting codes with $n = (q^r - 1)/(q - 1)$ and $k = (q^r - 1)/(q - 1) - r$, where r is an integer ≥ 2. Therefore, a (4, 2) ternary Hamming code exists, but it is not a cyclic code.

The next step is to explore if we can find a parity check polynomial corresponding to our generator polynomial, $g(x)$. We already know that $g(x)$ is a factor of $x^n - 1$. Hence we can write

$$x^n - 1 = h(x)g(x), \tag{4.11}$$

where $h(x)$ is some polynomial. The following can be concluded by simply observing the above equation:

(i) Since $g(x)$ is monic, $h(x)$ has to be monic because the left-hand side of the equation is also monic (the leading coefficient is unity).

(ii) Since degree of $g(x)$ is $n - k$, the degree of $g(x)$ must be k.

Suppose C is a cyclic code in R_n with the generator polynomial $g(x)$. Recall that we are denoting $F[x]/f(x)$ by R_n, where $f(x) = x^n - 1$. In R_n, $h(x)g(x) = x^n - 1 = 0$. Then, any codeword belonging to C can be written as $c(x) = a(x)g(x)$, where the polynomial $a(x) \in R_n$. Therefore in R_n,

$$c(x)h(x) = a(x)g(x)h(x) = a(x).0 = 0.$$

Thus, $h(x)$ behaves like a **Parity Check Polynomial**. Any valid codeword when multiplied by the parity check polynomial yields the zero polynomial. This concept is parallel to that of the parity check matrix introduced in the previous chapter. Since we are still in the domain of linear block codes, we go ahead and define the parity check matrix in relation to the parity check polynomial.

Suppose C is a cyclic code with the parity check polynomial $h(x) = h_0 + h_1x + \ldots + h_kx^k$, then the parity check matrix of C is given by

$$H = \begin{bmatrix} h_k & h_{k-1} & \cdots & h_0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & h_k & h_{k-1} & \cdots & h_0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & h_k & h_{k-1} & \cdots & h_0 & 0 & \cdots & 0 \\ \vdots & \vdots & & & & & & & \vdots \\ 0 & 0 & 0 & 0 & 0 & h_k & h_{k-1} & \cdots & h_0 \end{bmatrix} \quad (n - k) \text{ rows} \tag{4.12}$$

$$n \text{ columns}$$

Recall that $cH^T = 0$. Therefore, $iGH^T = 0$ for any information vector, i. Hence, $GH^T = 0$. We further have $s = vH^T$ where s is the syndrome vector and v is the received word.

Example 4.15 For binary codes of block length, $n = 7$, we have

$$x^7 - 1 = (x - 1)(x^3 + x + 1)(x^3 + x^2 + 1)$$

Consider $g(x) = (x^3 + x + 1)$. Since $g(x)$ is a factor of $x^7 - 1$, there is a cyclic code that can be generated by it. The generator matrix corresponding to $g(x)$ is

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

Scanned by CamScanner

The parity check polynomial $h(x)$ is $(x-1)(x^3+x^2+1) = (x^4+x^2+x+1)$, and the corresponding parity check matrix is

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

The minimum distance of this code is 3, and this also happens to be the (7, 4) Hamming code. Thus, the binary (7, 4) Hamming code is also a cyclic code with a generator polynomial $g(x) = (x^3 + x + 1)$.

4.6 QUASI CYCLIC CODES AND SHORTENED CYCLIC CODES

In this section we will study codes that are closely related to cyclic codes.

Definition 4.6 An (n, k) **Quasi Cyclic Code** is a linear block code such that, for some m coprime with n, the polynomial $x^m c(x)$ calculated mod $x^n - 1$ is a codeword polynomial if $c(x)$ is a valid codeword polynomial.

Example 4.16 Consider the (12, 4) quasi-cyclic code given by the generator matrix

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

Here $m = 3$ because the cyclic shift of any row of G by three places produces another row of the matrix.

Any cyclic code can be **punctured** to form a quasi cyclic code by dropping every m^{th} symbol where m is a factor of n. This quasi code will be a shortened code if none of the dropped symbols are check symbols. This means that all the dropped symbols are data symbols, which can be first set to zero.

Definition 4.7 An (n, k) linear code is called a proper **Shortened Cyclic Code** if it is obtained by deleting m consecutive places from an $(n+m, k+m)$ cyclic code.

The unused symbols in a shortened code are always set to zero and are not transmitted. The receiver reinserts the zeros prior to decoding. Thus, even though the code is no longer cyclic, the same encoder and decoder can be used after the zeros have been taken care of.

Example 4.17 Consider the (7, 4) Hamming code, which is also a cyclic code. We form a shortened (5, 3) code as follows. The two leading bits in the codeword are set to zero, followed by two information bits and the same three parity bits. The minimum distance of the shortened code is also 3, making it a single error correcting code.

4.7 BURST ERROR CORRECTION

In many real-life channels, errors are not random, but occur in bursts. For example, in a mobile communications channel, fading results in burst errors. When errors occur at a stretch, as opposed to random errors, we term them as **Burst Errors.**

Example 4.18 Let the transmitted sequence of bits, transmitted at 10 kb/s over a wireless channel, be

$c = 0\,1\,0\,0\,0\,1\,1\,1\,0\,1\,0\,1\,0\,0\,0\,0\,1\,0\,1\,1\,0\,1.$

Suppose, after 0.5 ms of the start of transmission, the channel experiences a fade of duration 1 ms. During this time interval, the channel corrupts the transmitted bits. The error sequence can be written as

$b = 0\,0\,0\,0\,0\,1\,1\,0\,1\,1\,0\,1\,1\,1\,1\,0\,0\,0\,0\,0\,0.$

This is an example of a burst error, where a portion of the transmitted sequence gets garbled due to the channel. Here the *length* of the burst is 10 bits, however, not all the ten locations are in error.

Definition 4.8 A cyclic burst of length t is a vector whose non-zero components are among t successive components, the first and last of which are non-zero.

If we are constructing codes for channels more prone to burst errors of length t (as opposed to an arbitrary pattern of t random errors) it might be possible to design more efficient codes. We can describe a burst error as

$$e(x) = x^i . b(x), \tag{4.13}$$

where $e(x)$ is a polynomial of degree $\leq t - 1$ and $b(x)$ is the burst pattern. x^i marks the starting location of the burst pattern within the codeword being transmitted.

A code designed for correcting a burst of length t must have unique syndromes for every error pattern, *i.e.*,

$$s(x) = R_{g(x)}[e(x)] \tag{4.14}$$

is different for each polynomial representing a burst of length t.

Scanned by CamScanner

Example 4.19 For a binary code of block length $n = 15$, consider the generator polynomial

$$g(x) = x^6 + x^3 + x^2 + x + 1.$$

This code is capable of correcting bursts of length 3 or less. To prove this, we must show that all the syndromes corresponding to the different burst errors are distinct. The different burst errors are:

(i) Bursts of length 1 $\quad e(x) = x^i$ for $i = 0, 1, ..., 14$

(ii) Bursts of length 2 $\quad e(x) = x^i.(1 + x)$ for $i = 0, 1, ..., 13$

(iii) Bursts of length 3 $\quad e(x) = x^i.(1 + x + x^2)$ for $i = 0, 1, ..., 12$ and $e(x) = x^i.(1 + x^2)$ for $i = 0, 1, ...12$

It can be shown that the syndrome of all these 55 (15 + 14 + 13 + 13) error patterns are distinct. A table can be made for each pattern and the corresponding syndrome which can be used for correcting a burst error of length 3 or less. It should be emphasised that the codes designed specifically for correcting burst errors are more efficient in terms of the code rate. The code being discussed here is a (15, 9) cyclic code with code rate $= k/n = 0.6$ and minimum distance $d^* = 3$. This code can correct only 1 random error (but up to three burst errors). Note that correction of one random error amounts to correcting a burst error of length 1.

Similar to the Singleton Bound studied in the previous chapter, there is a bound for the minimum number of parity bits required for a burst error correcting linear block code: 'A linear block code that corrects all bursts of length t or less must have at least $2t$ parity symbols'. This is known as the **Rieger Bound**.

In the next three sections, we will study three different sub-classes of cyclic codes. Each sub-class has a specific objective.

4.8 FIRE CODES

Definition 4.9 A **Fire code** is a cyclic burst error correcting code over $GF(q)$ with the generator polynomial

$$g(x) = (x^{2t-1}-1)p(x), \tag{4.15}$$

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 block length of the Fire code is the smallest integer n such that $g(x)$ divides x^n-1. A Fire code can correct all burst errors of length t or less.

Example 4.20 Consider the Fire code with $t = m = 3$. A prime polynomial over $GF(2)$ of degree 3 is $p(x) = x^3 + x + 1$, which does not divide $(x^5 - 1)$. The generator polynomial of the Fire code will be

$$\begin{aligned} g(x) &= (x^5 - 1)p(x) = (x^5 - 1)(x^3 + x + 1) \\ &= x^8 + x^6 + x^5 - x^3 - x - 1 \\ &= x^8 + x^6 + x^5 + x^3 + x + 1 \end{aligned}$$

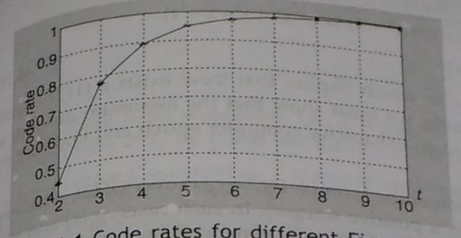


Fig. 4.1 Code rates for different Fire codes.

The degree of $g(x) = n - k = 8$. The block length is the smallest integer n such that $g(x)$ divides $x^n - 1$. After trial and error we get $n = 35$. Thus, the parameters of the Fire code are (35, 27) with $g(x) = x^8 + x^6 + x^5 + x^3 + x + 1$. This code can correct up to bursts of length 3. The code rate of this code is 0.77, and is more efficient than the code generated by $g(x) = x^6 + x^3 + x^2 + x + 1$ which has a code rate of only 0.6.

Fire codes become more efficient as we increase t. The code rates for binary Fire codes (with $m = t$) for different values of t are plotted in Fig. 4.1.

4.9 GOLAY CODES

Binary Golay Code In the previous chapter, section 3.9, we saw that a (23, 12) perfect code exists with $d^* = 7$. Recall that, for a perfect code,

$$M\left\{\binom{n}{0}+\binom{n}{1}(q-1)+\binom{n}{2}(q-1)^2+\ldots+\binom{n}{t}(q-1)^t\right\} = q^n, \tag{4.16}$$

which is satisfied for the values: $n = 23$, $k = 12$, $M = 2^k = 2^{12}$, $q = 2$ and $t = (d^* - 1)/2 = 3$. Thus (23, 12) perfect code is the binary Golay code. We shall now explore this perfect code as a cyclic code. We start with the factorisation of $(x^{23} - 1)$.

$$\begin{aligned} (x^{23} - 1) &= (x - 1)(x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1)(x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1) \\ &= (x - 1)g_1(x)g_2(x). \end{aligned} \tag{4.17}$$

The degree of $g_1(x) = n - k = 11$, hence $k = 12$, which implies that there exists a (23, 12) cyclic code. In order to prove that it is a perfect code, we must show that the minimum distance of this (23, 12) cyclic code is 7. One way, is to write out the parity check matrix, **H**, and show that no six columns are linearly dependent. Another way is to prove it analytically, which is a long

Scanned by CamScanner

and drawn-out proof. The easiest way is to write a computer program to list out all the 20 codewords and find the minimum weight (on a fast computer it takes about 20 seconds). The code rate is 0.52 and it is a triple error correcting code. However, the relatively small block length of this perfect code makes it impractical for most real life applications.

Ternary Golay Code We next examine the ternary (11, 6) cyclic code, which is also the ternary Golay code. This code has a minimum distance = 5, and can be verified to be a perfect code. We begin by factorising $(x^{11} - 1)$ over $GF(3)$.

$$(x^{11}-1) = (x-1)(x^5 + x^4 - x^3 + x^2 - 1)\,(x^5 - x^3 - x^2 - x - 1)$$
$$= (x-1)g_1(x)g_2(x). \tag{4.18}$$

The degree of $g_1(x) = n - k = 5$, hence $k = 6$, which implies that there exists a (11, 6) cyclic code. In order to prove that it is a perfect code, we must show that the minimum distance of this (11, 6) cyclic code is 5. Again, we resort to an exhaustive computer search and find that the minimum distance is indeed 5.

It can be shown that (x^p-1) has a factorisation of the form $(x-1)g_1(x)g_2(x)$ over $GF(2)$, whenever p is a prime number of the form $8m \pm 1$ (m is a positive integer). In such cases, $g_1(x)$ and $g_2(x)$ generate equivalent codes. If the minimum distance of the code generated by $g_1(x)$ is odd, then it satisfies the **square root bound**

$$d^* \geq \sqrt{p}\,. \tag{4.19}$$

Note that p denotes the block length.

4.10 CYCLIC REDUNDANCY CHECK (CRC) CODES

One of the common error detecting codes is the Cyclic Redundancy Check (CRC) codes. For a k-bit block of bits the (n, k) CRC encoder generates $(n - k)$ bit long frame check sequence (FCS). Let us define the following;

T = n-bit frame to be transmitted.

D = k-bit message block (information bits).

F = $(n - k)$ bit FCS, the last $(n - k)$ bits of T.

P = the predetermined divisor, a pattern of $(n - k + 1)$ bits.

Since the predetermined divisor, P, should be able to divide the codeword T. Thus, T/P has no remainder. Now, D is the k-bit message block. Therefore, $2^{n-k}D$ amounts to shifting the k bits to the left by $(n - k)$ bits, and padding the result with zeros (recall that left shift by 1 bit of a binary sequence is equivalent to multiplying the number represented by the binary sequence by two). The codeword, T, can then be represented as

$$T = 2^{n-k}D + F. \tag{4.20}$$

Adding F in the above equation yields the concatenation of D and F. If we divide $2^{n-k}D$ by P, we obtain

$$\frac{2^{n-k}D}{P} = Q + \frac{R}{P}, \tag{4.21}$$

where, Q is the quotient and R/P is the remainder. Suppose we use R as the FCS, then,

$$T = 2^{n-k}D + R. \tag{4.22}$$

In this case, upon dividing T by P, we obtain

$$\frac{T}{P} = \frac{2^{n-k}D + R}{P} = \frac{2^{n-k}D}{P} + \frac{R}{P}$$
$$= Q + \frac{R}{P} + \frac{R}{P} = Q + \frac{R+R}{P} = Q\,. \tag{4.23}$$

Thus there is no remainder, i.e., T is exactly divisible by P. To generate such an FCS we simply divide $2^{n-k}D$ by P and use the $(n - k)$-bit remainder as the FCS.

Let an error E occur when T is transmitted over a noisy channel. The received word is given by

$$V = T + E. \tag{4.24}$$

The CRC scheme will fail to detect the error only if V is completely divisible by P. This translates to the case when E is completely divisible by P (because T is divisible by P).

Example 4.21 Let the message D = 1010001101, i.e., k = 10 and the pattern, P = 110101. The number of FCS bits = 5. Therefore, n = 15. We wish to determine the FCS.

First, the message is multiplied by 2^5 (left shift by 5 and pad with 5 zeros). This yields

$$2^5D = 101000110100000$$

Next divide the resulting number by P = 110101. By long division, we obtain Q = 1101010110 and R = 01110. The remainder is added to 2^5D to obtain

$$T = 101000110101110$$

T is the transmitted codeword. If no errors occur in the channel, the received word when divided by P will yield 0 as the remainder.

CRC codes can also be defined using the polynomial representation. Let the message polynomial be $D(x)$ and the predetermined divisor be $P(x)$. Therefore,

$$\frac{x^{n-k}D(x)}{P(x)} = Q(x) + \frac{R(x)}{P(x)}$$

$$T(x) = x^{n-k}D(x) + R(x)\,. \tag{4.25}$$

Scanned by CamScanner

At the receiver end, the received word is divided by $P(x)$. Suppose the received word is

$$V(x) = T(x) + E(x), \tag{4.26}$$

where $E(x)$is the error polynomial.Then $[T(x) + E(x)]/P(x) = E(x)/P(x)$ because $T(x)/P(x) = 0$. Those errors that happen to correspond to polynomials containing $P(x)$ as a factor will slip by, and the others will be caught in the net of the CRC decoder. The polynomial $P(x)$ is also called the generator polynomial for the CRC code. CRC codes are also known as **polynomial codes**.

Example 4.22 Suppose the transmitted codeword undergoes a single-bit error. The error polynomial $E(x)$ can be represented by $E(x) = x^i$, where i determines the location of the single error bit. If $P(x)$ contains two or more terms, $E(x)/P(x)$ can never be zero. Thus all the single errors will be caught by such a CRC code.

Example 4.23 Suppose two isolated errors occur, i.e., $E(x) = x^i + x^j$, $i > j$. Alternately, $E(x) = x^j(x^{i-j} + 1)$. If we assume that $P(x)$ is not divisible by x, then a sufficient condition for detecting all double errors is that $P(x)$ does not divide $x^k + 1$ for any k up to the maximum value of $i - j$ (i.e., the frame length). For example $x^{15} + x^{14} + 1$ will not divide $x^k + 1$ for any value of k below 32,768.

Example 4.24 Suppose the error polynomial has an odd number of terms (corresponding to an odd number of errors). An interesting fact is that there is no polynomial with an odd number of terms that has $x + 1$ as a factor if we are performing a binary arithmetic (modulo 2 operations). By making $(x + 1)$ as a factor of $P(x)$, we can catch all errors consisting of odd number of bits (i.e., we can catch at least half of all possible errors).

Another interesting feature of CRC codes is its ability to detect burst errors. A burst error of length k can be represented by $x^i(x^{k-1} + x^{k-2} + \ldots + 1)$, where i determines how far from the right-end of the received frame the burst is located. If $P(x)$ has a x^0 term, it will not have a x^i as a factor. So, if the degree of $(x^{k-1} + x^{k-2} + \ldots + 1)$ is less than the degree of $P(x)$, the remainder can never be zero. Therefore, a polynomial code with r check bits can detect all burst errors of length $\le r$. If the burst length is $r + 1$, the remainder of the division by $P(x)$ will be zero if and only if the burst is identical to $P(x)$. Now, the 1st and last bits of a burst must be 1 (by definition). The intermediate bits can be 1 or 0. Therefore, the exact matching of the burst error with the polynomial $P(x)$ depends on the $r - 1$ intermediate bits. Assuming all combinations are equally likely, the probability of a miss is $\frac{1}{2^{r-1}}$. One can show that when error bursts of length greater than $r + 1$ occurs, or several shorter bursts occur, the probability of a bad frame slipping through is $\frac{1}{2^r}$.

Example 4.25 Four versions of $P(x)$ have become international standards:

CRC-12: $P(x) = x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$

CRC-16: $P(x) = x^{16} + x^{15} + x^2 + 1$

CRC-CCITT: $P(x) = x^{16} + x^{12} + x^5 + 1$

CRC-32: $P(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x^1 + 1$ (4.27)

All the four contain $(x + 1)$ as a factor. CRC-12 is used for transmission of streams of 6-bit characters and generates a 12-bit FCS. Both CRC-16 and CRCCCITT are popular for 8-bit characters. They result in a 16 bit FCS and can catch all single and double errors, all errors with odd number of bits, all burst errors of length 16 or less, 99.997% of 17-bit bursts and 99.998% of 18-bit and longer bursts. CRC32 is specified as an option in some point-to-point **Synchronous Transmission Standards.**

4.11 CIRCUIT IMPLEMENTATION OF CYCLIC CODES

Shift registers can be used to encode and decode cyclic codes easily. Encoding and decoding of cyclic codes require multiplication and division by polynomials. The shift property of shift registers are ideally suited for such operations. Shift registers are banks of memory units which are capable of shifting the contents of one unit to the next at every clock pulse. Here we will focus on circuit implementation for codes over $GF(2^m)$. Beside the shift register, we will make use of the following circuit elements:

(i) A scaler, whose job is to multiply the input by a fixed field element.

(ii) An adder, which takes in two inputs and adds them together. A simple circuit realisation of an adder is the 'exclusive-or' or the 'xor' gate.

(iii) A multiplier, which is basically the 'and' gate.

These elements are depicted in Fig. 4.2.

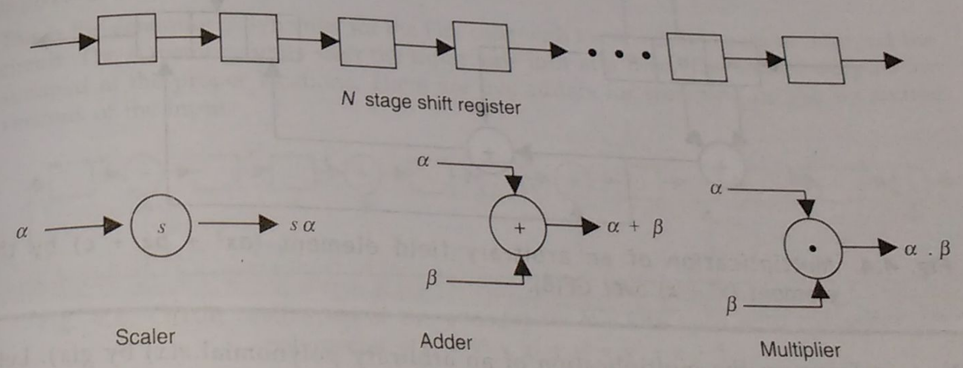


Fig. 4.2 Circuit elements used to construct encoders and decoders for cyclic codes.

Scanned by CamScanner

A field element of $GF(2)$ can simply be represented by a single bit. For $GF(2^m)$ we require m bits to represent one element. For example, elements of $GF(8)$ can be represented as the elements of the set {000, 001, 010, 011, 100, 101, 110, 111}. For such a representation we need three clock pulses to shift the element from one stage of the shift register to the next. The effective shift register for $GF(8)$ is shown in Fig. 4.3. Any arbitrary element of this field can be represented by $ax^2 + bx + c$, where a, b, c are binary, and the power of the indeterminate x is used to denote the position. For example, $101 = x^2 + 1$.

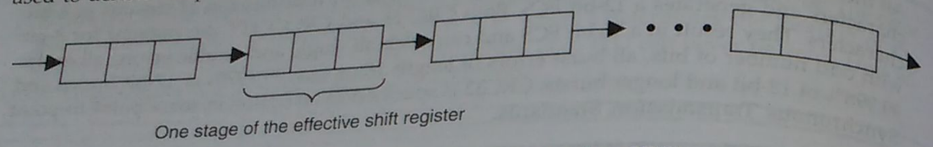


Fig. 4.3 The effective shift register for $GF(8)$.

Example 4.26 We now consider the multiplication of any arbitrary element by another field element over $GF(8)$. Recall the construction of $GF(8)$ from $GF(2)$ using the prime polynomial $p(x) = x^3 + x + 1$. The elements of the field will be $0, 1, x, x + 1, x^2, x^2 + 1, x^2 + x, x^2 + x + 1$. We want to obtain the circuit representation for the multiplication of any arbitrary field element $(ax^2 + bx + c)$ by another element, say, $x^2 + x$. We have

$$(ax^2 + bx + c)(x^2 + x) = ax^4 + (a+b)x^3 + (b+c)\,x^2 + cx \quad (\text{modulo } p(x))$$

$$= (a+b+c)x^2 + (b+c)x + (a+b)$$

One possible circuit realisation is shown in Fig. 4.4.

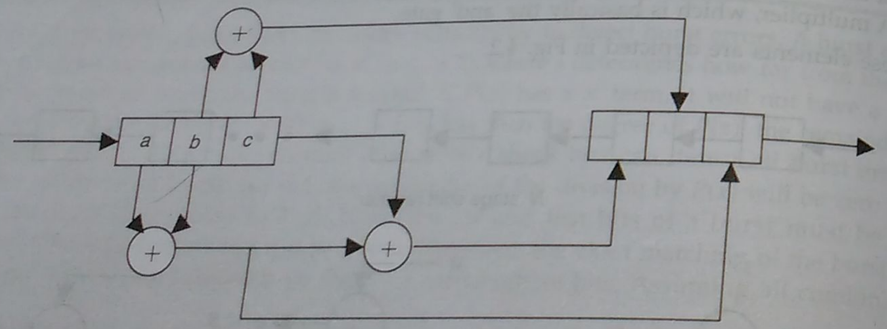


Fig. 4.4. Multiplication of an arbitrary field element $(ax^2 + bx + c)$ by the element $(x^2 + x)$ over $GF(8)$.

We next focus on the multiplication of an arbitrary polynomial $a(x)$ by $g(x)$. Let the polynomial $g(x)$ be represented as

$$g(x) = g_L x^L + \ldots + g_1 x + g_0, \tag{4.28}$$

the polynomial $a(x)$ be represented as

$$a(x) = a_k x^k + \ldots + a_1 x + a_0, \tag{4.29}$$

the resultant polynomial $b(x) = a(x)g(x)$ be represented as

$$b(x) = b_{k+L} x^{k+L} + \ldots + b_1 x + b_0. \tag{4.30}$$

The circuit realisation of $b(x)$ is given in Fig. 4.5. This is linear feedforward shift register. It is also called a **Finite Impulse Response (FIR)** filter.

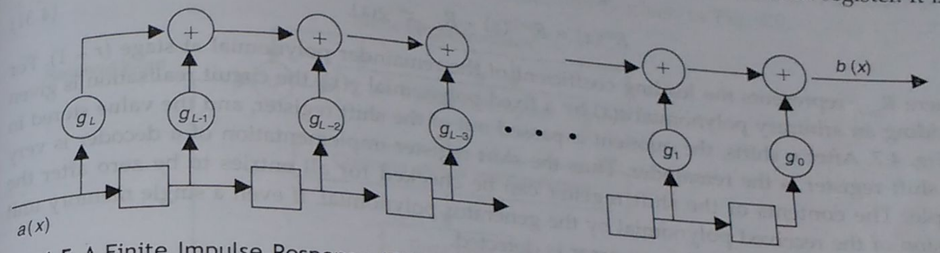


Fig. 4.5 A Finite Impulse Response (FIR) filter realization of the product of two polynomials $b(x) = a(x)g(x)$.

In Electrical Engineering jargon, the coefficients of $a(x)$ and $g(x)$ are convolved by the shift register. For our purpose, we have a circuit realisation for multiplying two polynomials. Thus, we have an efficient mechanism of encoding a cyclic code by multiplying the information polynomial by the generator polynomial.

Example 4.27 The encoder circuit for the generator polynomial

$$g(x) = x^8 + x^6 + x^5 + x^3 + x + 1$$

is given in Fig. 4.6.

This is the generator polynomial for the Fire code with $t = m = 3$. It is easy to interpret the circuit. The 8 memory units shift the input, one unit at a time. The shifted outputs are summed at the proper locations. There are five adders for summing up the six shifted versions of the input.

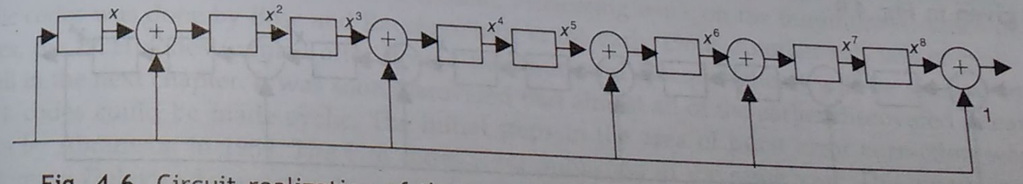


Fig. 4.6. Circuit realization of the encoder for the Fire code with the generator polynomial $g(x) = x^8 + x^6 + x^5 + x^3 + x + 1$.

Scanned by CamScanner

We can also use a shift register circuit for dividing an arbitrary polynomial, $a(x)$, by a fixed polynomial $g(x)$. We assume here that the divisor is a monic polynomial. We already know how to factor out a scalar in order to convert any polynomial to a monic polynomial. The division process can be expressed as a pair of recursive equations. Let $Q^{(r)}(x)$ and $R^{(r)}(x)$ be the quotient polynomial and the remainder polynomial at the rth recursion step, with the initial conditions $Q^{(0)}(x) = 0$ and $R^{(0)}(x) = a(x)$. Then, the recursive equations can be written as

$$Q^{(r)}(x) = Q^{(r-1)}(x) + R_{(n-r)}x^{k-r},$$

$$R^{(r)}(x) = R^{(r-1)}(x) - R_{(n-r)}x^{k-r}g(x), \tag{4.31}$$

where $R_{(n-r)}$ represents the leading coefficient of the remainder polynomial at stage $(r-1)$. For dividing an arbitrary polynomial $a(x)$ by a fixed polynomial $g(x)$, the circuit realisation is given in Fig. 4.7. After n shifts, the quotient is passed out of the shift register, and the value stored in the shift register is the remainder. Thus the shift register implementation of a decoder is very simple. The contents of the shift register can be checked for all entries to be zero after the division of the received polynomial by the generator polynomial. If even a single memory unit of the shift register is non zero, an error is detected.

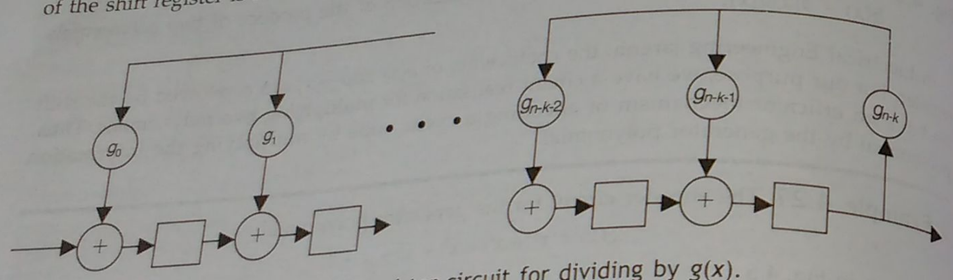


Fig. 4.7. A shift register circuit for dividing by $g(x)$.

Example 4.28 The shift register circuit for dividing by $g(x) = x^8 + x^6 + x^5 + x^3 + x + 1$ is given in Fig. 4.8.

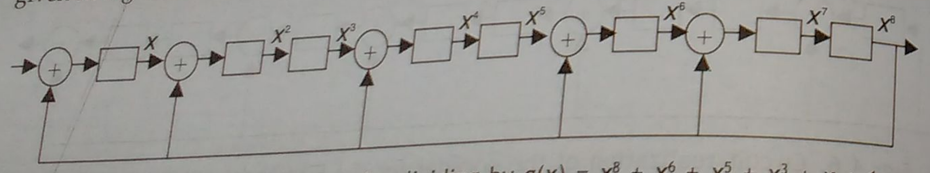


Fig. 4.8 A shift register circuit for dividing by $g(x) = x^8 + x^6 + x^5 + x^3 + x + 1$.

The procedure for error detection and error correction is as follows. The received word is first stored in a buffer. It is subjected to divide-by-$g(x)$ operation. As we have seen, the division can be carried out very efficiently by a shift register circuit.The remainder in the shift register is then compared with all the possible (precomputed) syndromes. These set of syndromes correspond to the set of correctable error patterns. If a syndrome match is found, the error is subtracted out form the received word. The corrected version of the received word is then passed on to the next stage of the receiver unit for further processing. This kind of a decoder is known as the **Meggitt Decoder.** The flowchart for a decoder is given in Fig. 4.9.

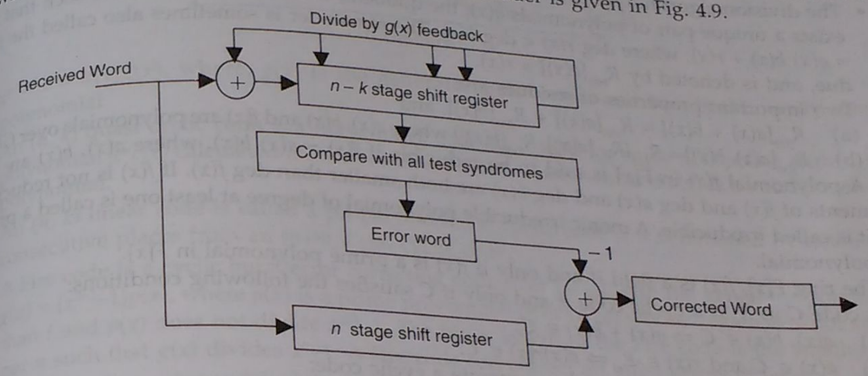


Fig. 4.9. The flowchart of a Meggitt Decoder.

4.12 CONCLUDING REMARKS

Cyclic codes are interesting because of two primary reasons:

(i) the structure of polynomial rings leads to structure in the code and

(ii) the existence of elegant mathematical tools based on polynomials that can be used to analyse these codes.

The notion of cyclic codes was first introduced by Prange in 1957. The work on cyclic codes was further developed by Peterson and Kasami. Pioneering work on the minimum distance of cyclic codes was done by Bose and Raychaudhuri in the early 1960s. Another subclass of cyclic codes, the BCH codes (named after Bose, Chaudhuri and Hocquenghem) will be studied in detail in the next chapter. It was soon discovered that almost all of the earlier discovered linear block codes could be made cyclic. The initial steps in the area of burst error correction was taken by Abramson in 1959. The Fire codes were published in the same year. The binary and the ternary Golay codes were published by Golay in, as early as, 1949.

Shift register circuits for cyclic codes were introduced in the works of Peterson, Chien and Meggitt in the early 1960s. Important contributions were also made by Kasami, MacWilliams, Mitchell and Rudolph.

Scanned by CamScanner

SUMMARY

- A polynomial is a mathematical expression $f(x) = f_0 + f_1x + ... + f_mx^m$, where the symbol x is called the indeterminate and the coefficients $f_0, f_1, ... f_m$ are the elements of $GF(q)$. The coefficient f_m is called the leading coefficient. If $f_m \neq 0$, then m is called the degree of the polynomial, and is denoted by deg $f(x)$. A polynomial is called monic if its leading coefficient is unity.
- The division algorithm states that, for every pair of polynomial $a(x)$ and $b(x) \neq 0$ in $F[x]$, there exists a unique pair of polynomials $q(x)$, the quotient, and $r(x)$, the remainder, such that $a(x) = q(x)\, b(x) + r(x)$, where deg $r(x) <$ deg $b(x)$. The remainder is sometimes also called the residue, and is denoted by $R_{b(x)}[a(x)] = r(x)$.
- Two important properties of residues are:
  (a) $R_{f(x)}[a(x) + b(x)] = R_{f(x)}[a(x)] + R_{f(x)}[b(x)]$, and
  (b) $R_{f(x)}[a(x).\, b(x)] = R_{f(x)}\{R_{f(x)}[a(x)].\, R_{f(x)}[b(x)]\}$ where $a(x)$, $b(x)$ and $f(x)$ are polynomials over $GF(q)$.
- A polynomial $f(x)$ in $F[x]$ is said to be reducible if $f(x) = a(x)\, b(x)$, where $a(x)$, $b(x)$ are elements of $f(x)$ and deg $a(x)$ and deg $b(x)$ are both smaller than deg $f(x)$. If $f(x)$ is not reducible, it is called irreducible. A monic irreducible polynomial of degree at least one is called a prime polynomial.
- The ring $F[x]/f(x)$ is a field if and only if $f(x)$ is a prime polynomial in $F[x]$.
- A code C in R_n is a cyclic code if and only if C satisfies the following conditions:
  (a) $a(x), b(x) \in C \Rightarrow a(x) + b(x) \in C$,
  (b) $a(x) \in C$ and $r(x) \in R_n \Rightarrow a(x)r(x) \in C$.
- The following steps can be used to generate a cyclic code:
  (a) Take a polynomial $f(x)$ in R_n.
  (b) Obtain a set of polynomials by multiplying $f(x)$ by all possible polynomials in R_n.
  (c) The set of polynomials obtained as above corresponds to the set of codewords belonging to a cyclic code. The blocklength of the code is n.
- Let C be a (n, k) non-zero cyclic code in R_n. Then,
  (a) there exists a unique monic polynomial $g(x)$ of the smallest degree in C,
  (b) the cyclic code C consists of all multiples of the generator polynomial $g(x)$ by polynomials of degree $k - 1$ or less.
  (c) $g(x)$ is a factor of $x^n - 1$.
  (d) The degree of $g(x)$ is $n - k$.
- For a cyclic code, C, with generator polynomial $g(x) = g_0 + g_1x + ...+ g_rx^r$ of degree r, the generator matrix is given by

$$G = \begin{bmatrix} g_0 & g_1 & \cdots & g_r & 0 & 0 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & g_r & 0 & 0 & \cdots & 0 \\ 0 & 0 & g_0 & g_1 & \cdots & g_r & 0 & \cdots & 0 \\ \vdots & \vdots & & & & & & & \vdots \\ 0 & 0 & 0 & 0 & 0 & g_0 & g_1 & \cdots & g_r \end{bmatrix} \quad k = (n-r) \text{ rows}$$

n columns

- For a cyclic code, C, with the parity check polynomial $h(x) = h_0 + h_1x + ...+ h_kx^k$, the parity check matrix is given by

$$H = \begin{bmatrix} h_k & h_{k-1} & \cdots & h_0 & 0 & 0 & 0 & \cdots & 0 \\ 0 & h_k & h_{k-1} & \cdots & h_0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & h_k & h_{k-1} & \cdots & h_0 & 0 & \cdots & 0 \\ \vdots & \vdots & & & & & & & \vdots \\ 0 & 0 & 0 & 0 & 0 & h_k & h_{k-1} & \cdots & h_0 \end{bmatrix} \quad (n-k) \text{ rows}$$

n columns

- $x^n - 1 = h(x)g(x)$, where, $g(x)$ is the generator polynomial and $h(x)$ is the parity check polynomial.
- An (n, k) quasi cyclic code is a linear block code such that, for some m coprime with n, the polynomial $x^mc(x)$ calculated mod $x^n - 1$ is a codeword polynomial if $c(x)$ is a valid codeword polynomial.
- An (n, k) linear code is called a proper shortened cyclic code if it is obtained by deleting m consecutive places from an $(n+m, k+m)$ cyclic code.
- A Fire code is a cyclic burst error correcting code over $GF(q)$ with the 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 block length of the Fire code is the smallest integer n such that $g(x)$ divides x^n-1. A Fire code can correct all burst errors of length t or less.
- The generator polynomial of the binary Golay Code:
  $g_1(x) = (x^{11} + x^{10} + x^6 + x^5 + x^4 + x^2 + 1)$, or
  $g_2(x) = (x^{11} + x^9 + x^7 + x^6 + x^5 + x + 1)$.
- The generator polynomial of the ternary Golay Code:
  $g_1(x) = (x^5 + x^4 - x^3 + x^2 - 1)$, or
  $g_2(x) = (x^5 - x^3 - x^2 - x - 1)$.
- One of the common error detecting codes is the Cyclic Redundancy Check (CRC) codes. For a k-bit block of bits the (n, k) CRC encoder generates $(n - k)$ bit long frame check sequence (FCS).
- Shift registers can be used to encode and decode cyclic codes easily. Encoding and decoding of cyclic codes require multiplication and division by polynomials. The shift property of shift registers are ideally suited for such operations.

> *Everything should be made as simple as possible, but not simpler.*
>
> Einstein, Albert (1879–1955)

Scanned by CamScanner

PROBLEMS

4.1 Which of the following codes are (a) cyclic, (b) equivalent to a cyclic code?
(i) {0000, 0110, 1100, 0011, 1001} over $GF(2)$
(ii) {00000, 10110, 01101, 11011} over $GF(2)$
(iii) {00000, 10110, 01101, 11011} over $GF(3)$
(iv) {0000, 1122, 2211} over $GF(3)$
(v) The q-ary repetition code of length n

4.2 Construct the addition and multiplication table for
(i) $F[x]/(x^2 + 1)$ defined over $GF(2)$, (ii) $F[x]/(x^2 + 1)$ defined over $GF(3)$
Which of the above is a field?

4.3 List out all the irreducible polynomials over
(i) $GF(2)$ of degrees 1 to 5, (ii) $GF(3)$ of degrees 1 to 3

4.4 Find all the cyclic binary codes of block length 5. Find is the minimum distance of each code.

4.5 Suppose $x^n - 1$ is a product of r distinct irreducible polynomials over $GF(q)$. How many cyclic codes of block length n over $GF(q)$ exist? What can you comment on the minimum distance of these codes?

4.6 (i) Factorise $x^8 - 1$ over $GF(3)$.
(ii) How many ternary cyclic codes of length 8 exist ?
(iii) How many quaternary cyclic codes of length 8 exist ?

4.7 Let the polynomial
$g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$
be the generator polynomial of a cyclic code over $GF(2)$ with block length 15.
(i) Find the generator polynomial G.
(ii) Find the parity check matrix H.
(iii) How many errors can this code detect?
(iv) How many errors can this code correct?
(v) Write the generator matrix in the systematic form.

4.8 Consider the polynomial
$g(x) = x^6 + 3x^5 + x^4 + x^3 + 2x^2 + 2x + 1$
(i) Is this a valid generator polynomial for a cyclic code over $GF(4)$ with block length 15?
(ii) Find the parity check matrix H.
(iii) What is the minimum distance of this code?
(iv) What is the code rate of this code?
(v) Is the received word $v(x) = x^8 + x^5 + 3x^4 + x^3 + 3x + 1$ a valid codeword?

4.9 An error vector of the form $x^i + x^{i+1}$ in R_n is called a double adjacent error. Show that the code generated by the generator polynomial $g_1(x) = (x - 1)g_H(x)$ is capable of correcting all double adjacent errors, where $g_H(x)$ is the generator polynomial of the binary Hamming code.

4.10 Determine the burst error correction length for the binary (1023, 1010) cyclic code represented by the following generator polynomial
$g(x) = x^{13} + x^{10} + x^7 + x^6 + x^5 + x^4 + x^2 + 1$.

4.11 Show that the block length of a Fire code is $n = \text{LCM}(2t - 1, q^m - 1)$.

4.12 Show that $g(x) = (x^{11} + 1)(x^6 + x + 1)$ is a valid generator polynomial of a Fire code. What is the burst error correction length for this code?

4.13 Show that a shortened binary cyclic code with generator polynomial of degree $n - k$ can detect a fraction $1 - 2^{-(n-k)+1}$ of all burst patterns of length $(n - k) + 1$.

4.14 Design the shift register encoder and the Meggitt decoder for the code generated in problem 4.8.

4.15 The code with the generator polynomial $g(x) = (x^{23} + 1)(x^{17} + x^3 + 1)$ is used for error detection and correction in the GSM standard.
(i) How many random errors can this code correct?
(ii) How many burst errors can this code correct?

COMPUTER PROBLEMS

4.16 Write a computer program to find the minimum distance of a cyclic code over $GF(q)$, given the generator polynomial (or the generator matrix) for the code.

4.17 Write a computer program to encode and decode a (35, 27) Fire code. It should be able to automatically correct bursts of length 3 or less. What happens when you try to decode a received word with a burst error of length 4? Write another computer program to encode and decode a (19437, 19408) Fire code. What is the burst error correction length for this code?

4.18 Write a computer program to search for cyclic Hamming codes over $GF(5)$. Is there a cyclic Hamming code of blocklength $n = 156$ over this Galois Field?

4.19 The IEEE 802.16 broadband wireless access group prescribes the following polynomial for CRC

$$g(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

(i) What error patterns can this code detect?
(ii) What is the burst error correction length for this code?

Scanned by CamScanner

Bose–Chaudhuri Hocquenghem (BCH) Codes

5

Although this may seem a paradox, all exact science is dominated by the idea of approximation.

Russell, Bertrand (1872–1970)

5.1 INTRODUCTION TO BCH CODES

The class of Bose–Chaudhuri Hocquenghem (BCH) codes is one of the most powerful known class of linear cyclic block codes. BCH codes are known for their multiple error correcting ability, and the ease of encoding and decoding. So far, our approach has been to construct a code, and then find out its minimum distance in order to estimate its error correcting capability. In this class of code, we will start from the other end. We begin by specifying the number of random errors we desire the code to correct. Then we go on to construct the generator polynomial for the code. As mentioned above, BCH codes are a subclass of cyclic codes, and therefore, the decoding methodology for any cyclic code also works for the BCH codes. However, more efficient decoding procedures are known for BCH codes, and will be discussed in this chapter.

We begin by building the necessary mathematical tools in the next couple of sections. We shall then look at the method for constructing the generator polynomial for BCH codes. Efficient decoding techniques for this class of codes will be discussed next. An important subset of BCH codes, the **Reed–Solomon codes**, will be introduced in the later part of this chapter.

5.2 PRIMITIVE ELEMENTS

Definition 5.1 A **Primitive Element** of $GF(q)$ is an element α such that every field element except zero can be expressed as a power of α.

Example 5.1 Consider $GF(5)$. Since $q = 5$ = a prime number, modulo arithmetic will work. Consider the element 2.

$2^0 = 1 \pmod 5 = 1,$

$2^1 = 2 \pmod 5 = 2,$

$2^2 = 4 \pmod 5 = 4,$

$2^3 = 8 \pmod 5 = 3.$

Hence, all the elements of $GF(5)$ i.e., $\{1, 2, 3, 4, 5\}$ can be represented as powers of 2. Therefore, 2 is a primitive element of $GF(5)$.

Next, consider the element 3.

$3^0 = 1 \pmod 5 = 1,$

$3^1 = 3 \pmod 5 = 3,$

$3^2 = 9 \pmod 5 = 4,$

$3^3 = 27 \pmod 5 = 2.$

Again, all the elements of $GF(5)$ i.e., $\{1, 2, 3, 4, 5\}$ can be represented as powers of 2. Therefore, 3 is also a primitive element of $GF(5)$.

However, it can be tested that the other non-zero elements $\{1, 4, 5\}$ are not primitive elements.

We saw in the example that there can be more than one primitive element in a field. But is there a guarantee of finding at least one primitive element? The answer is yes! The non-zero elements of every Galois Field form a cyclic group. Hence, a Galois Field will include an element of the order $q - 1$. This will be the primitive element. Primitive elements are very useful in constructing fields. Once we have a primitive element, we can easily find all the other elements by simply evaluating the powers of the primitive element.

Definition 5.2 A **Primitive Polynomial** $p(x)$ over $GF(q)$ is a prime polynomial over $GF(q)$ with the property that in the extension field constructed modulo $p(x)$, the field element represented by x is a primitive element.

Primitive polynomials of every degree exist over every Galois Field. A primitive polynomial can be used to construct an extension field.

Scanned by CamScanner

Example 5.2 We can construct $GF(8)$ using the primitive polynomial $p(x) = x^3 + x + 1$. Let the primitive element of $GF(8)$ be α. Then, we can represent all the elements of $GF(8)$ by the powers of a evaluated modulo $p(x)$. Thus, we can form the Table 5.1.

Table 5.1 The elements of GF(8).

Powers of α	Elements of GF(8)
α^1	z
α^2	z^2
α^3	$z + 1$
α^4	$z^2 + z$
α^5	$z^2 + z + 1$
α^6	$z^2 + 1$
α^7	1

Theorem 5.1 Let $\beta_1, \beta_2, \ldots, \beta_{q-1}$ denote the non-zero field elements of $GF(q)$. Then,

$$x^{q-1} - 1 = (x - \beta_1)(x - \beta_2) \ldots (x - \beta_{q-1}). \tag{5.1}$$

Proof The set of non-zero elements of $GF(q)$ is a finite group under the operation of multiplication. Let β be any non-zero element of the field. It can be represented as a power of the primitive element α. Let $\beta = (\alpha)^r$ for some integer r.

Therefore, $\beta^{q-1} = ((\alpha)^r)^{q-1} = ((\alpha)^{q-1})^r = (1)^r = 1$ because, $(\alpha)^{q-1} = 1$.

Hence, β is a zero of $x^{q-1} - 1$. This is true for any non-zero element β.

Hence, $x^{q-1} - 1 = (x - \beta_1)(x - \beta_2) \ldots (x - \beta_{q-1})$.

Example 5.3 Consider the field $GF(5)$. The non-zero elements of this field are {1, 2, 3, 4}. Therefore, we can write

$$x^4 - 1 = (x - 1)(x - 2)(x - 3)(x - 4)$$

5.3 MINIMAL POLYNOMIALS

In the previous chapter we saw that in order to find the generator polynomials for cyclic codes of blocklength n, we have to first factorise $x^n - 1$. Thus $x^n - 1$ can be written as the product of its p prime factors

$$x^n - 1 = f_1(x)\, f_2(x)\, f_3(x) \ldots f_p(x). \tag{5.2}$$

Any combination of these factors can be multiplied together to form a generator polynomial $g(x)$. If the prime factors of $x^n - 1$ are distinct, then there are $(2^p - 2)$ different non trivial cyclic codes of blocklength n. The two trivial cases that are being disregarded are $g(x) = 1$ and $g(x) = x^n - 1$. Not all of the $(2^p - 2)$ possible cyclic codes are good codes in terms of their minimum distance. We now evolve a strategy for finding good codes, i.e., of a desirable minimum distance.

In the previous chapter we learnt how to construct an extension field from the sub field. In this section we will study the prime polynomials (in a certain field) that have zeros in the

extension field. Our strategy for constructing $g(x)$ will be as follows. Using the desirable zeros in the extension field we will find the prime polynomials in the sub field, which will be multiplied together to yield a desirable $g(x)$.

Definition 5.3 A blocklength n of the form $n = q^m - 1$ is called a **Primitive Blocklength** for a code over $GF(q)$. A cyclic code over $GF(q)$ of primitive blocklength is called a **Primitive Cyclic Code**.

The field $GF(q^m)$ is an extension field of $GF(q)$. Let the primitive blocklength $n = q^m - 1$. Consider the factorisation

$$x^n - 1 = x^{q^m-1} - 1 = f_1(x) f_2(x) \ldots f_p(x) \tag{5.3}$$

over the field $GF(q)$. This factorisation will also be valid over the extension field $GF(q^m)$ because the addition and multiplication tables of the sub field form a part of the tables of the extension field. We also know that $g(x)$ divides $x^n - 1$, *i.e.*, $x^{q^m-1} - 1$, hence $g(x)$ must be the product of some of these polynomials $f_i(x)$. Also, every non-zero element of $GF(q^m)$ is a zero of $x^{q^m-1} - 1$. Hence, it is possible to factor $x^{q^m-1} - 1$ in the extension field $GF(q^m)$ to get

$$x^{q^m-1} - 1 = \prod_j (x - \beta_j) \tag{5.4}$$

where β_j ranges over all the non-zero elements of $GF(q^m)$. This implies that each of the polynomials $f_i(x)$ can be represented in $GF(q^m)$ as a product of some of the linear terms, and each β_j is a zero of *exactly one* of the $f_i(x)$. This $f_i(x)$ is called the minimal polynomial of β_j.

Definition 5.4 The smallest degree polynomial with coefficients in the base field $GF(q)$ that has a zero in the extension field $GF(q^m)$ is called the **Minimal Polynomial** of β_j.

Example 5.4 Consider the subfield $GF(2)$ and its extension field $GF(8)$. Here $q = 2$ and $m = 3$. The factorisation of $x^{q^m-1} - 1$ (in the sub field / extension field) yields

$$x^{q^m-1} - 1 = x^7 - 1 = (x - 1)(x^3 + x + 1)(x^3 + x^2 + 1)$$

Next consider the elements of the extension field $GF(8)$. The elements can be represented as $0, 1, z, z + 1, z^2, z^2 + 1, z^2 + z, z^2 + z + 1$ (from example 4.10 of Chapter 4).

Therefore, we can write

$$x^{q^m-1} - 1 = x^7 - 1 = (x - 1)(x - z)(x - z - 1)(x - z^2)(x - z^2 - 1)(x - z^2 - z)(x - z^2 - z - 1)$$
$$= (x - 1).[(x - z)(x - z^2)(x - z^2 - z)].[(x - z - 1)(x - z^2 - 1)(x - z^2 - z - 1)]$$

It can be seen that over $GF(8)$,

$(x^3 + x + 1) = (x - z)(x - z^2)(x - z^2 - z)$, and

$(x^3 + x^2 + 1) = (x - z - 1)(x - z^2 - 1)(x - z^2 - z - 1)$

Scanned by CamScanner

The multiplication and addition are carried out over $GF(8)$. Interestingly, after a little bit of algebra it is found that the coefficients of the minimal polynomial belong to $GF(2)$ only. We can now make Table 5.2.

Table 5.2 The elements of $GF(8)$ in terms of the powers of the primitive element, α.

Minimal polynomial $f_i(x)$	*Corresponding elements β_i in $GF(8)$*	*Elements in terms of powers of α*
$(x - 1)$	1	α^0
$(x^3 + x + 1)$	z, z^2 and $z^2 + z$	$\alpha^1, \alpha^2, \alpha^4$
$(x^3 + x^2 + 1)$	$z + 1$, $z^2 + 1$ and $z^2 + z + 1$	$\alpha^3, \alpha^6, \alpha^5\ (= \alpha^{12})$

It is interesting to note the elements (in terms of powers of the primitive element α) that correspond to the same minimal polynomial. If we make the observation that $\alpha^{12} = \alpha^7.\alpha^5 = 1.\alpha^5$, we see a pattern in the elements that correspond to a certain minimal polynomial. In fact, the elements that are roots of a minimal polynomial in the extension field are of the type $\beta^{q^{r-1}}$ where β is an element of the extension field. In the above example, the zeros of the minimal polynomial $f_2(x) = x^3 + x + 1$ are α^1, α^2 and α^4 and that of $f_3(x) = x^3 + x^2 + 1$ are α^3, α^6 and α^{12}.

Definition 5.5 *Two elements of $GF(q^m)$ that share the same minimal polynomial over $GF(q)$ are called* **conjugates** *with respect to $GF(q)$.*

Example 5.5 *The elements $\{\alpha^1, \alpha^2, \alpha^4\}$ are conjugates with respect to $GF(2)$. They share the same minimal polynomial $f_2(x) = x^3 + x + 1$.*

As we have seen, a single element in the extension field may have more than one conjugate. The conjugacy relationship between two elements depends on the base field. For example, the extension field $GF(16)$ can be constructed using either $GF(2)$ or $GF(4)$. Two elements that are conjugates with respect to $GF(2)$ may not be conjugates with respect to $GF(4)$.

If $f(x)$ is the minimal polynomial of β, then it is also the minimal polynomial of the elements in the set $\{\beta, \beta^q, \beta^{q^2}, \ldots, \beta^{q^{r-1}}\}$, where r is the smallest integer such that $\beta^{q^{r-1}} = \beta$. The set $\{\beta, \beta^q, \beta^{q^2}, \ldots, \beta^{q^{r-1}}\}$ is called the **Set of Conjugates**. The elements in the set of conjugates are all the zeros of $f(x)$. Hence, the minimal polynomial of β can be written as

$$f(x) = (x-\beta)(x-\beta^q)(x-\beta^{q^2})\ldots(x-\beta^{q^{r-1}}). \tag{5.5}$$

Example 5.6 *Consider $GF(256)$ as an extension field of $GF(2)$. Let α be the primitive element of $GF(256)$. Then a set of conjugates would be*

$$\{\alpha^1, \alpha^2, \alpha^4, \alpha^8, \alpha^{16}, \alpha^{32}, \alpha^{64}, \alpha^{128}\}.$$

Note that $\alpha^{256} = \alpha^{255}\alpha^1 = \alpha^1$, hence the set of conjugates terminates with α^{128}. The minimal polynomial of α is

$$f(x) = (x - \alpha^1)(x - \alpha^2)(x - \alpha^4)(x - \alpha^8)(x - \alpha^{16})(x - \alpha^{32})(x - \alpha^{64})(x - \alpha^{128}).$$

The right-hand side of the equation when multiplied out would only contain coefficients from $GF(2)$.

Similarly, the minimal polynomial of α^3 would be

$$f(x) = (x - \alpha^3)(x - \alpha^6)(x - \alpha^{12})(x - \alpha^{24})(x - \alpha^{48})(x - \alpha^{96})(x - \alpha^{192})(x - \alpha^{129}).$$

Definition 5.6 BCH codes defined over $GF(q)$ with blocklength $q^m - 1$ are called **Primitive BCH codes.**

Having developed the necessary mathematical tools, we shall now begin our study of the BCH codes. We will develop a method for constructing the generator polynomials of BCH codes that can correct the pre-specified t random errors.

5.4 GENERATOR POLYNOMIALS IN TERMS OF MINIMAL POLYNOMIALS

We know that $g(x)$ is a factor of $x^n - 1$. Therefore, the generator polynomial of a cyclic code can be written in the form

$$g(x) = \text{LCM}\ [f_1(x)\ f_2(x), \ldots, f_p(x)], \tag{5.6}$$

where, $f_1(x)\ f_2(x), \ldots, f_p(x)$ are the minimal polynomials of the zeros of $g(x)$. Each minimal polynomial corresponds to a zero of $g(x)$ in an extension field. We will design good codes (*i.e.*, determine the generator polynomials) with desirable zeros using this approach.

Let $c(x)$ be a codeword polynomial and $e(x)$ be an error polynomial. Then the received polynomial can be written as

$$v(x) = c(x) + e(x), \tag{5.7}$$

where the polynomial coefficients are in $GF(q)$. Now consider the extension field $GF(q^m)$. Let $\gamma_1, \gamma_2, \ldots, \gamma_p$ be those elements of $GF(q^m)$ which are the zeros of $g(x)$, i.e, $g(\gamma_i) = 0$ for $i = 1, .., p$. Since, $c(x) = a(x)g(x)$ for some polynomial $a(x)$, we also have $c(\gamma_i) = 0$ for $i = 1, .., p$. Thus,

$$\begin{aligned} v(\gamma_i) &= c(\gamma_i) + e(\gamma_i) \\ &= e(\gamma_i) \text{ for } i = 1, .., p. \end{aligned} \tag{5.8}$$

For a blocklength n, we have

$$v(\gamma_i) = \sum_{j=0}^{n-1} e_j \gamma_i^j \text{ for } i = 1, .., p. \tag{5.9}$$

Scanned by CamScanner

Thus we have a set of p equations that involve components of the error pattern only. If it is possible to solve this set of equations for e_i, the error pattern can be precisely determined. Whether this set of equations can be solved depends on the value of p, the number of zeros of $g(x)$. In order to solve for the error pattern, we must choose the set of p equations properly. If we have to design for a t error correcting cyclic code, our choice should be such that the set of equations can solve for at most, t non-zero e_i.

Let us define the syndromes $S_i = e(\gamma_i)$ for $i = 1, .., p$. We wish to choose $\gamma_1, \gamma_2, ... \gamma_p$ in such a manner that t errors can be computed from $S_1, S_2, ..., S_p$. If α is a primitive element, then the set of γ_i which allows the correction of t errors is $\{\alpha_1, \alpha_2, \alpha_3, ... , \alpha^{2t}\}$. Thus, we have a simple mechanism of determining the generator polynomial of a BCH code that can correct t errors.

Steps for determining the generator polynomial of a t–error correcting BCH code:

For a primitive blocklength $n = q^m - 1$:

(1) Choose a prime polynomial of degree m and construct $GF(q^m)$.

(2) Find $f_i(x)$, the minimal polynomial of α^i for $i = 1, .., 2t$.

(3) The generator polynomial for the t–error correcting code is simply

$$g(x) = LCM\ [f_1(x)\ f_2(x), ..., f_{2t}(x)]. \tag{5.10}$$

Codes designed in this manner can correct at least t errors. In many cases the codes will be able to correct more than t errors. For this reason,

$$d = 2t + 1, \tag{5.11}$$

is called the **designed distance** of the code, and the minimum distance $d^* \geq 2t + 1$. The generator polynomial has a degree equal to $n - k$ (see Theorem 4.4, previous chapter). It should be noted that once we fix n and t, we can determine the generator polynomial for the BCH code. We do not have a control over the information length k. After we determine $g(x)$, its degree will decide the value of k. Intuitively, for a fixed blocklength n, a larger value of t will force the information length k to be smaller (because a higher redundancy will be required to correct more number of errors). In the following section, we look at a few specific examples of BCH codes.

5.5 SOME EXAMPLES OF BCH CODES

The following example illustrates the construction of the extension field $GF(16)$ from $GF(2)$. The minimal polynomials obtained will be used in the subsequent examples.

Example 5.7 Consider the primitive polynomial $p(z) = z^4 + z + 1$ over $GF(2)$. We shall use this to construct the extension field $GF(16)$. Let $\alpha = z$ be the primitive element. The elements of $GF(16)$ as powers of α and the corresponding minimal polynomials are listed in Table 5.3.

Table 5.3 The elements of $GF(16)$ and the corresponding minimal polynomials.

Powers of α	Elements of $GF(16)$	Minimal Polynomials
α^1	z	$x^4 + x + 1$
α^2	z^2	$x^4 + x + 1$
α^3	z^3	$x^4 + x^3 + x^2 + x + 1$
α^4	$z + 1$	$x^4 + x + 1$
α^5	$z^2 + z$	$x^2 + x + 1$
α^6	$z^3 + z^2$	$x^4 + x^3 + x^2 + x + 1$
α^7	$z^3 + z + 1$	$x^4 + x^3 + 1$
α^8	$z^2 + 1$	$x^4 + x + 1$
α^9	$z^3 + z$	$x^4 + x^3 + x^2 + x + 1$
α^{10}	$z^2 + z + 1$	$x^2 + x + 1$
α^{11}	$z^3 + z^2 + z$	$x^4 + x^3 + 1$
α^{12}	$z^3 + z^2 + z + 1$	$x^4 + x^3 + x^2 + x + 1$
α^{13}	$z^3 + z^2 + 1$	$x^4 + x^3 + 1$
α^{14}	$z^3 + 1$	$x^4 + x^3 + 1$
α^{15}	1	$x + 1$

Example 5.8 We wish to determine the generator polynomial of a *single* error correcting binary BCH code, *i.e.*, $t = 1$ with a blocklength $n = 15$. From (5.10), the generator polynomial for a BCH code is given by LCM $[f_1(x)\ f_2(x), ..., f_{2t}(x)]$. We will make use of Table 5.3 to obtain the minimal polynomials $f_1(x)$ and $f_2(x)$. Thus, the generator polynomial of the single error correcting binary BCH code will be

$$\begin{aligned} g(x) &= LCM\ [f_1(x), f_2(x)] \\ &= LCM[(x^4 + x + 1)(x^4 + x + 1)] \\ &= x^4 + x + 1. \end{aligned}$$

Since, deg $(g(x)) = n - k$, we have, $n - k = 4$, which gives $k = 11$. Thus we have obtained the generator polynomial of the BCH(15, 11) single error correcting code. The designed distance of this code $d = 2t + 1 = 3$. It can be calculated that the minimum distance d^* of this code is also 3. Thus, in this case the designed distance is equal to the minimum distance.

Next, we wish to determine the generator polynomial of a *double* error correcting BCH code, *i.e.*, $t = 2$ with a blocklength $n = 15$. The generator polynomial of the BCH code will be

$$\begin{aligned} g(x) &= LCM\ [f_1(x), f_2(x), f_3(x), f_4(x)] \\ &= LCM[(x^4 + x + 1)(x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^4 + x + 1)] \\ &= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1) \\ &= x^8 + x^7 + x^6 + x^4 + 1. \end{aligned}$$

Since, deg$(g(x)) = n - k$, we have $n - k = 8$, which gives $k = 7$. Thus we have obtained the generator polynomial of the BCH(15, 7) double error correcting code. The designed distance

Scanned by CamScanner

of this code $d = 2t + 1 = 5$. It can be calculated that the minimum distance d^* of this code is also 5. Thus in this case the designed distance is equal to the minimum distance.

Next, we determine the generator polynomial for the *triple* error correcting binary BCH code. The generator polynomial of the BCH code will be

$$\begin{aligned} g(x) &= \text{LCM}\,[f_1(x)\, f_2(x), f_3(x), f_4(x), f_5(x), f_6(x)] \\ &= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1) \\ &= x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1. \end{aligned}$$

In this case, $\deg(g(x)) = n - k = 10$, which gives $k = 5$. Thus we have obtained the generator polynomial of the BCH(15, 5) triple error correcting code. The designed distance of this code $d = 2t + 1 = 7$. It can be calculated that the minimum distance d^* of this code is also 7. Thus in this case the designed distance is equal to the minimum distance.

Next, we determine the generator polynomial for a binary BCH code for the case $t = 4$. The generator polynomial of the BCH code will be

$$\begin{aligned} g(x) &= \text{LCM}\,[f_1(x)\, f_2(x), f_3(x), f_4(x), f_5(x), f_6(x), f_7(x), f_8(x)] \\ &= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1)(x^4 + x^3 + 1) \\ &= x^{14} + x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1. \end{aligned}$$

In this case, $\deg(g(x)) = n - k = 14$, which gives $k = 1$. It can be seen that this is the simple repetition code. The designed distance of this code $d = 2t + 1 = 9$. However, it can be seen that the minimum distance d^* of this code is 15. Thus in this case the designed distance is not equal to the minimum distance, and the code is overdesigned. This code can actually correct $(d^* - 1)/2 = 7$ random errors.

If we repeat the exercise for $t = 5$, 6 or 7, we get the same generator polynomial (repetition code). Note that there are only 15 non-zero field elements in $GF(16)$ and hence there are only 15 minimal polynomials corresponding to these field elements. Thus, we cannot go beyond $t = 7$ (because for $t = 8$ we need $f_{16}(x)$, which is undefined). Hence, to obtain BCH codes that can correct larger number of errors we must use an extension field with more elements.

Example 5.9 We can construct $GF(16)$ as an extension field of $GF(4)$ using the primitive polynomial $p(z) = z^2 + z + 2$ over $GF(4)$. Let the elements of $GF(4)$ consist of the quaternary symbols contained in the set {0, 1, 2, 3}. The addition and multiplication tables for $GF(4)$ are given in Table 5.4 for handy reference.

Table 5.4 Addition and multiplication tables for $GF(4)$.

+	0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

.	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	3	1
3	0	3	1	2

Table 5.5 lists the elements of $GF(16)$ as powers of α and the corresponding minimal polynomials.

Table 5.5 The elements of $GF(16)$ as powers of α.

Powers of α	Elements of $GF(16)$	Minimal Polynomials
α^1	z	$x^2 + x + 2$
α^2	$z + 2$	$x^2 + x + 3$
α^3	$3z + 2$	$x^2 + 3x + 1$
α^4	$z + 1$	$x^2 + x + 2$
α^5	2	$x + 2$
α^6	$2z$	$x^2 + 2x + 1$
α^7	$2z + 3$	$x^2 + 2x + 2$
α^8	$z + 3$	$x^2 + x + 3$
α^9	$2z + 2$	$x^2 + 2x + 1$
α^{10}	3	$x + 3$
α^{11}	$3z$	$x^2 + 3x + 3$
α^{12}	$3z + 1$	$x^2 + 3x + 1$
α^{13}	$2z + 1$	$x^2 + 2x + 2$
α^{14}	$3z + 3$	$x^2 + 3x + 3$
α^{15}	1	$x + 1$

For $t = 1$,

$$\begin{aligned} g(x) &= \text{LCM}\,[f_1(x), f_2(x)] \\ &= \text{LCM}[(x^2 + x + 2)(x^2 + x + 3)] \\ &= x^4 + x + 1. \end{aligned}$$

Since, $\deg(g(x)) = n - k$, we have $n - k = 4$, which gives $k = 11$. Thus we have obtained the generator polynomial of the *single* error correcting BCH(15, 11) code over $GF(4)$. It takes in 11 quaternary information symbols and encodes them into 15 quaternary symbols. Note that one quaternary symbol is equivalent to two bits. So, in effect, the BCH(15, 11) takes in 22 input bits and transforms them into 30 encoded bits (can this code be used to correct a burst of length 2 for a binary sequence of length 30?). The designed distance of this code $d = 2t + 1 = 3$. It can be calculated that the minimum distance d^* of this code is also 3. Thus in this case the designed distance is equal to the minimum distance.

For $t = 2$,

$$\begin{aligned} g(x) &= \text{LCM}\,[f_1(x), f_2(x), f_3(x), f_4(x)] \\ &= \text{LCM}[(x^2 + x + 2)(x^2 + x + 3)(x^2 + 3x + 1)(x^2 + x + 2)] \\ &= (x^2 + x + 2)(x^2 + x + 3)(x^2 + 3x + 1) \\ &= x^6 + 3x^5 + x^4 + x^3 + 2x^2 + 2x + 1. \end{aligned}$$

Scanned by CamScanner

This is the generator polynomial of a (15, 9) *double* error correcting BCH code over GF(4).

For $t = 3$,

$$g(x) = \text{LCM}\,[f_1(x), f_2(x), f_3(x), f_4(x), f_5(x), f_6(x)]$$
$$= \text{LCM}[(x^2 + x + 2)(x^2 + x + 3)(x^2 + 3x + 1)(x + 2)(x^2 + 2x + 1)$$
$$= (x^2 + x + 2)(x^2 + x + 3)(x^2 + 3x + 1)(x + 2)(x^2 + 2x + 1)$$
$$= x^9 + 3x^8 + 3x^7 + 2x^6 + x^5 + 2x^4 + x + 2.$$

This is the generator polynomial of a (15, 6) *triple* error correcting BCH code over GF(4).

Similarly, for $t = 4$,

$$g(x) = x^{11} + x^{10} + 2x^8 + 3x^7 + 3x^6 + x^5 + 3x^4 + x^3 + x + 3.$$

This is the generator polynomial of a (15, 4) *four* error correcting BCH code over GF(4).

Similarly, for $t = 5$,

$$g(x) = x^{12} + 2x^{11} + 3x^{10} + 2x^9 + 2x^8 + x^7 + 3x^6 + 3x^4 + 3x^3 + x^2 + 2.$$

This is the generator polynomial of a (15, 3) *five* error correcting BCH code over GF(4).

Similarly, for $t = 6$,

$$g(x) = x^{14} + x^{13} + x^{12} + x^{11} + x^{10} + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^3 + x^2 + x + 1.$$

This is the generator polynomial of a (15, 1) six error correcting BCH code over GF(4). As is obvious, this is the simple repetition code, and can correct up to 7 errors.

Table 5.6 lists the generator polynomials of binary BCH codes of length up to $2^5 - 1$. Suppose we wish to construct a generator polynomials of the BCH(15,7) code. From the table we have (111 010 001) for the coefficients of the generator polynomial. Therefore

$$g(x) = x^8 + x^7 + x^6 + x^4 + 1.$$

Table 5.6 The generator polynomials of binary BCH codes of length up to 2^5-1.

n	k	t	*Generator polynomial coefficients*
7	4	1	1 011
15	11	1	10 011
15	7	2	111 010 001
15	5	3	10 100 110 111
31	26	1	100 101
31	21	2	11 101 101 001
31	16	3	1 000 111 110 101 111
31	11	5	101 100 010 011 011 010 101
31	6	7	11 001 011 011 110 101 000 100 111

5.6 DECODING OF BCH CODES

So far we have learnt the method to obtain the generator polynomial for a BCH code given by the number of random errors to be corrected. With the knowledge of the generator polynomial, very fast encoders can be built in the hardware. We now shift our attention to the decoding of the BCH codes. Since the BCH codes are a subclass of the cyclic codes, any standard decoding procedure for cyclic codes is also applicable to BCH codes. However, better, more efficient algorithms have been designed specifically for BCH codes. We discuss the **Gorenstein–Zierler** decoding algorithm, which is the generalised form of the binary decoding algorithm first proposed by Petersen.

We develop here the decoding algorithm for a t error correcting BCH code. Suppose a BCH code is constructed based on the field element α. Consider the error polynomial

$$e(x) = e_{n-1}x^{n-1} + e_{n-2}x^{n-2} + \ldots + e_1x + e_0, \tag{5.12}$$

where at most t coefficients are non-zero. Suppose that v errors actually occur, where $0 \le v \le t$. Let these errors occur at locations $i_1, i_2, \ldots, i_v$. The error polynomial can then be written as

$$e(x) = e_{i_1}x^{i_1} + e_{i_2}x^{i_2} + \ldots + e_{i_v}x^{i_v}, \tag{5.13}$$

where e_{i_k} is the magnitude of the kth error. Note that we are considering the general case. For binary codes, $e_{i_k} = 1$. For error correction, we must know two things:

(i) *where* the errors have occurred, *i.e.* the error locations, and

(ii) what are the *magnitudes* of these errors.

Thus, the unknowns are $i_1, i_2, \ldots, i_v$ and $e_{i_1}, e_{i_2}, \ldots, e_{i_v}$, which signify the locations and the magnitudes of the errors respectively. The syndrome can be obtained by evaluating the received polynomial at α.

$$S_1 = v(\alpha) = c(\alpha) + e(\alpha) = e(\alpha)$$
$$= e_{i_1}x^{i_1} + e_{i_2}x^{i_2} + \ldots + e_{i_v}x^{i_v}. \tag{5.14}$$

Next, define the error magnitudes, $Y_k = e_{i_k}$ for $k = 1, 2, \ldots, v$ and the error locations for $X_k = \alpha^{i_k}$ for $X_k = 1, 2, \ldots, v$, where i_k is the location of the kth error and X_k is the field element associated with this location. Now, the syndrome can be written as

$$S_1 = Y_1X_1 + Y_2X_2 + \ldots + Y_vX_v. \tag{5.15}$$

We can evaluate the received polynomial at each of the powers of α that has been used the define $g(x)$. We define the syndromes for $j = 1, 2, \ldots, 2t$ by

$$S_j = v(\alpha^j) = c(\alpha^j) + e(\alpha^j) = e(\alpha^j). \tag{5.16}$$

Thus, we have the following set of $2t$ simultaneous equations, with v unknown error locations $X_1, X_2, \ldots, X_v$ and the v unknown error magnitudes $Y_1, Y_2, \ldots, Y_v$.

Scanned by CamScanner

$$
\begin{aligned}
S_1 &= Y_1X_1 + Y_2X_2 + \ldots + Y_vX_v \\
S_2 &= Y_1X_1^2 + Y_2X_2^2 + \ldots + Y_vX_v^2 \\
&\vdots \\
S_{2t} &= Y_1X_1^{2t} + Y_2X_2^{2t} + \ldots + Y_vX_v^{2t}.
\end{aligned} \tag{5.17}
$$

Next, define the **error locator polynomial**

$$\Lambda(x) = \Lambda_v x^v + \Lambda_{v-1}x^{v-1} + \ldots \Lambda_1 x + 1. \tag{5.18}$$

The zeros of this polynomial are the inverse error locations X_k^{-1} for $k = 1, 2, \ldots, v$. That is,

$$\Lambda(x) = (1 - xX_1)(1 - xX_2) \ldots (1 - xX_v). \tag{5.19}$$

So, if we know the coefficients of the error locator polynomial $\Lambda(x)$, we can obtain the error locations $X_1, X_2, \ldots, X_v$. After some algebraic manipulations we obtain

$$\Lambda_1 S_{j+v-1} + \Lambda_2 S_{j+v-2} + \ldots + \Lambda_v S_j = -S_{j+v} \text{ for } j = 1, 2, \ldots, v. \tag{5.20}$$

This is nothing but a set of linear equations that relate the syndromes to the coefficients of $\Lambda(x)$. This set of equations can be written in the matrix form as follows.

$$\begin{bmatrix} S_1 & S_2 & \ldots & S_{v-1} & S_v \\ S_2 & S_3 & \ldots & S_v & S_{v+1} \\ & & & & \\ S_v & S_{v+1} & \ldots & S_{2v-2} & S_{2v-1} \end{bmatrix} \begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{v+1} \\ -S_{v+2} \\ \\ -S_{2v} \end{bmatrix}. \tag{5.21}$$

The values of the coefficients of the error locator polynomial can be determined by inverting the syndrome matrix. This is possible only if the matrix is non-singular. It can be shown that this matrix is non-singular if there are v errors.

Steps for decoding BCH codes

(1) As a trial value, set $v = t$ and compute the determinant of the matrix of syndromes, M. If the determinant is zero, set $v = t - 1$. Again compute the determinant of M. Repeat this process until a value of v is found for which the determinant of the matrix of syndromes is non-zero. This value of v is the actual number of errors that occurred.

(2) Invert the matrix M and find the coefficients of the error locator polynomial $\Lambda(x)$.

(3) Solve $\Lambda(x) = 0$ to obtain the zeros and from them compute the error locations $X_1, X_2, \ldots, X_v$. If it is a binary code, stop (because the magnitudes of error are unity).

(4) If the code in not binary, go back to the system of equations:

$$
\begin{aligned}
S_1 &= Y_1X_1 + Y_2X_2 + \ldots + Y_vX_v \\
S_2 &= Y_1X_1^2 + Y_2X_2^2 + \ldots + Y_vX_v^2 \\
&\vdots \\
S_{2t} &= Y_1X_1^{2t} + Y_2X_2^{2t} + \ldots + Y_vX_v^{2t}
\end{aligned}
$$

Since the error locations are now known, these form a set of $2t$ *linear* equations. These can be solved to obtain the error magnitudes.

Solving for the Λ_i by inverting the $v \times v$ matrix can be computationally expensive. The number of computations required will be proportional to v^3. If we need to correct a large number of errors (i.e., a large v) we need a more efficient way to solve the matrix equation. Various refinements have been found which greatly reduce the computational complexity. It can be seen that the $v \times v$ matrix is not arbitrary in form. The entries in its diagonal, perpendicular to the main diagonal are all identical. This property is called *persymmetry*. This structure was exploited by Berleykamp (1968) and Massey (1969) to solve the system of equations in a much simpler manner. An algorithm proposed by Forney is used for finding the error magnitudes. The **Berleykamp–Massey–Forney** algorithm is commonly used for decoding BCH codes.

The simplest way to search for the zeros of $\Lambda(x)$ is to test out all the field elements one by one. This method of exhaustive search is known as the **Chien search**. The entire flow of the BCH decoding is shown in Fig. 5.1

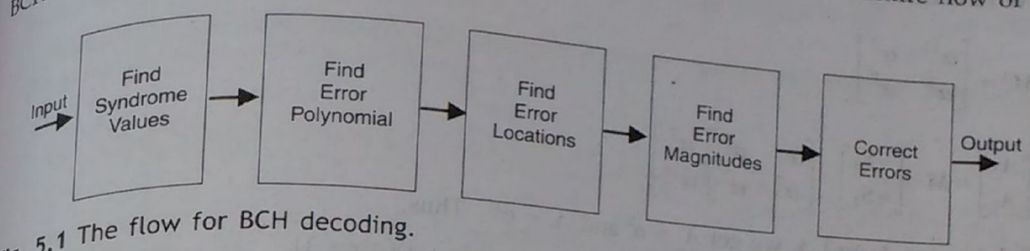


Fig. 5.1 The flow for BCH decoding.

Example 5.10 Consider the binary BCH (15, 5) triple error correcting code with the generator polynomial

$$g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1.$$

Let the all-zero codeword be transmitted and the received polynomial be $v(x) = x^5 + x^3$. Thus there are two errors at the locations 4 and 6. The error polynomial $e(x) = x^5 + x^3$. But the decoder does not know this. It does not even know how many errors have actually occurred. We use the Gorenstein–Zierler decoding algorithm. First we compute the syndromes using the arithmetic of $GF(16)$.

$$
\begin{aligned}
S_1 &= \alpha^5 + \alpha^3 = \alpha^{11}, \\
S_2 &= \alpha^{10} + \alpha^6 = \alpha^7, \\
S_3 &= \alpha^{15} + \alpha^9 = \alpha^7, \\
S_4 &= \alpha^{20} + \alpha^{12} = \alpha^{14}, \\
S_5 &= \alpha^{25} + \alpha^{15} = \alpha^5, \\
S_6 &= \alpha^{30} + \alpha^{18} = \alpha^{14}.
\end{aligned}
$$

First set $v = t = 3$, since this is a triple error correcting code.

Scanned by CamScanner

$$M = \begin{bmatrix} S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \\ S_3 & S_4 & S_5 \end{bmatrix} = \begin{bmatrix} \alpha^{11} & \alpha^7 & \alpha^7 \\ \alpha^7 & \alpha^7 & \alpha^{14} \\ \alpha^7 & \alpha^{14} & \alpha^5 \end{bmatrix}.$$

Det(M) = 0, which implies that fewer than 3 errors have occurred.

Next, set $v = t = 2$.

$$M = \begin{bmatrix} S_1 & S_2 \\ S_2 & S_3 \end{bmatrix} = \begin{bmatrix} \alpha^{11} & \alpha^7 \\ \alpha^7 & \alpha^7 \end{bmatrix}.$$

Det(M) = 1, which implies that 2 errors have actually occurred.

We next calculate M^{-1}. It so happens that in this case,

$$M^{-1} = \begin{bmatrix} \alpha^{11} & \alpha^7 \\ \alpha^7 & \alpha^7 \end{bmatrix},$$

$$\begin{bmatrix} \Lambda_2 \\ \Lambda_1 \end{bmatrix} = M^{-1} \begin{bmatrix} -S_3 \\ -S_4 \end{bmatrix} = \begin{bmatrix} \alpha^{11} & \alpha^7 \\ \alpha^7 & \alpha^7 \end{bmatrix} \begin{bmatrix} \alpha^7 \\ \alpha^{14} \end{bmatrix}.$$

Solving for Λ_1 and Λ_2 we get $\Lambda_2 = \alpha^8$ and $\Lambda_1 = \alpha^{11}$. Thus,

$$\Lambda(x) = \alpha^8 x^2 + \alpha^{11} x + 1 = (\alpha^5 x + 1)(\alpha^3 x + 1).$$

Thus, the recovered error locations are α^5 and α^3. Since the code is binary, the error magnitudes are 1. Thus, $e(x) = x^5 + x^3$.

In the next section, we will study the famous Reed–Solomon codes, an important sub-class of BCH codes.

5.7 REED–SOLOMON CODES

Reed–Solomon (RS) codes are an important subset of the non-binary BCH with a wide range of applications in digital communications and data storage. The typical application areas of the RS code are:

- Storage devices (including Compact Disk, Blu-ray Disk, DVD, barcodes, etc),
- Wireless or mobile communications (including cellular telephones, microwave links, etc),
- Satellite communications,
- Digital television / Digital Video Broadcast (DVB),
- High-speed modems such as those employing ADSL, xDSL, etc.

It all began with a five-page paper that appeared in 1960 in the Journal of the Society for Industrial and Applied Mathematics. The paper, "*Polynomial Codes over Certain Finite Fields*" by Irving S. Reed and Gustave Solomon, then staff members at MIT's Lincoln Laboratory, introduced the ideas that form a significant portion of current error correcting techniques for everything from computer hard disk drives to CD players. Reed–Solomon codes (plus a lot of engineering wizardry, of course), made possible the stunning pictures of the outer planets sent back by Voyager II. They make it possible to scratch a compact disc and still enjoy the music. And in the not-too-distant future, they will enable the profit mongers of cable television to squeeze more than 500 channels into their systems.

The payoff was a coding system based on groups of bits, such as bytes, rather than individual 0's and 1's. That feature makes Reed–Solomon codes particularly good at dealing with *bursts* of errors: six consecutive bit errors, for example, can affect at most two bytes. Thus, even a double error correction version of a Reed–Solomon code can provide a comfortable safety factor. Current implementations of Reed–Solomon codes in CD technology are able to cope with error bursts as long as 4000 consecutive bits.

In this subclass of BCH codes, the symbol field GF(q) and the error locator field GF(q^m) are the same, *i.e.*, $m = 1$. Thus, in this case

$$n = q^m - 1 = q - 1. \tag{5.22}$$

The minimal polynomial of any element β in the same field GF(q) is

$$f_\beta(x) = x - \beta. \tag{5.23}$$

Since the symbol field (sub field) and the error locator field (extension field) are the same, all the minimal polynomials are linear. The generator polynomial for a t error correcting code will be simply

$$\begin{aligned} g(x) &= \text{LCM}[f_1(x)\, f_2(x), \ldots, f_{2t}(x)] \\ &= (x - \alpha)(x - \alpha^2) \ldots (x - \alpha^{2t-1})(x - \alpha^{2t}). \end{aligned} \tag{5.24}$$

Hence, the degree of the generator polynomial will always be $2t$. Thus, the RS code satisfies

$$n - k = 2t. \tag{5.25}$$

In general, the generator polynomial of an RS code can be written as

$$g(x) = (x - \alpha^i)(x - \alpha^{i+1}) \ldots (x - \alpha^{2t+i-1})(x - \alpha^{2t+i}). \tag{5.26}$$

Example 5.11 Consider the double error correcting RS code of blocklength 15 over $GF(16)$. Here $t = 2$. We use here the elements of the extension field GF(16) constructed from $GF(2)$ using the primitive polynomial $p(z) = z^4 + z + 1$. The generator polynomial can be written as

$$\begin{aligned} g(x) &= (x - \alpha)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4) \\ &= x^4 + (z^3 + z^2 + 1)x^3 + (z^3 + z^2)x^2 + z^3 x + (z^2 + z + 1) \\ &= x^4 + (\alpha^{13})x^3 + (\alpha^6)x^2 + (\alpha^3)x + \alpha^{10}. \end{aligned}$$

Scanned by CamScanner

Here $n - k = 4$, which implies $k = 11$. Thus we have obtained the generator polynomial of a $RS(15, 11)$ code over $GF(16)$. Note that this coding procedure takes in 11 symbols (equivalent to $4 \times 11 = 44$ bits) and encodes them into 15 symbols (equivalent to 60 bits).

Theorem 5.2 A Reed–Solomon code is a maximum distance separable (MDS) code and its minimum distance is $n - k + 1$.

Proof Let the designed distance of the RS code be $d = 2t + 1$. The minimum distance d^* satisfies the condition $d^* \geq d = 2t + 1$.

But, for an RS code, $2t = n - k$. Hence,

$$d^* \geq d = 2t + 1 = n - k + 1.$$

But, by the Singleton bound for any linear code,

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

Thus, $d^* = n - k + 1$, and the minimum distance $d^* = d$, the designed distance of the code.

Since RS codes are maximum distance separable (MDS), all of the possible code words are as far away as possible algebraically in the code space. It implies a uniform code word distribution in the code space.

Table 5.7 lists the parameters of some RS codes. Note that for a given minimum distance, in order to have a high code rate, one must work with larger Galois Fields.

Table 5.7 Some RS code parameters.

m	$q = 2^m$	$n = q - 1$	t	k	d^*	$r = k/n$
2	4	3	1	1	3	0.3333
3	8	7	1	5	3	0.7143
			2	3	5	0.4286
			3	1	7	0.1429
4	16	15	1	13	3	0.8667
			2	11	5	0.7333
			3	9	7	0.6000
			4	7	9	0.4667
			5	5	11	0.3333
			6	3	13	0.2000
			7	1	15	0.0667
5	32	31	1	29	3	0.9355
			5	21	11	0.6774
			8	15	17	0.4839
8	256	255	5	245	11	0.9608
			15	225	31	0.8824
			50	155	101	0.6078

Example 5.12 A popular Reed–Solomon code is RS (255,223) with 8–bit symbols (bytes), *i.e.*, over $GF(255)$. Each codeword contains 255 code word bytes, of which 223 bytes are data and 32 bytes are parity. For this code, $n = 255$, $k = 223$ and $n - k = 32$. Hence, $2t = 32$, or $t = 16$. Thus, the decoder can correct any 16 symbol random error in the codeword, *i.e.* errors in up to 16 bytes anywhere in the codeword can be corrected.

Example 5.13 Reed–Solomon error correction codes can have an extremely pronounced effect on the efficiency of a digital communication channel. For example an operation running at a data rate of 1 million bytes per second will carry approximately 4000 blocks of 255 bytes each second. If 1000 random short errors (less than 17 bits in length) per second are injected into the channel, about 600 to 800 blocks per second would be corrupted, which might require retransmission of nearly all of the blocks. By applying the Reed–Solomon (255,235) code (that corrects up to 10 errors per block of 235 information bytes and 20 parity bytes), the typical time between blocks that cannot be corrected and would require retransmission will be about 800 years. The mean time between incorrectly decoded blocks will be over 20 billion years.

5.8 IMPLEMENTATION OF REED–SOLOMON ENCODERS AND DECODERS

Hardware Implementation Let the generator polynomial of the RS encoder be represented by

$$g(x) = g_0 + g_0 x + g_0 x^2 + \ldots + g_{2t-1} x^{2t-1} + x^{2t}. \tag{5.27}$$

Note that the right-hand side of the equation is written such that the highest power of x is to the right. The hardware implementation of this RS encoder is shown in Fig. 5.2. The encoding steps are as follows:

1. Switch 1 is closed during the first k clock cycles to allow shifting the message symbols into the $(n - k)$ stage shift register. Note that in these k cycles, the contents of the shift register and the feedback loop is continually changing, based on the information symbols being shifted in, as well as, the additions prior to each element of the shift register.
2. Switch 2 is in the down position during the first k clock cycles in order to allow simultaneous transfer of the message symbols directly to an output register. Note, that this is a systematic design, and the first k symbols of the codeword are the original information symbols.
3. After transfer of the kth message symbol to the output register, switch 1 is opened and switch 2 is moved to the up position. What is residing in the shift register are elements of the parity symbols. These symbols are ready to be shifted out, and appended to the information symbols in order to create the entire codeword.

Scanned by CamScanner

4. During the remaining $(n - k)$ clock cycles, clear the parity symbols contained in the shift-register by moving them to the output register.
5. The total number of clock cycles is equal to n, and the contents of the output register are the final codeword polynomial, corresponding to the k information symbols.

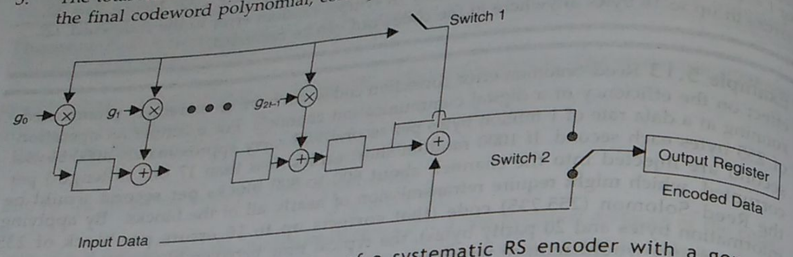


Fig. 5.2 The hardware implementation of a systematic RS encoder with a generator polynomial $g(x) = g^0 + g_0x + g_0x^2 + \ldots + g_{2t-1}x^{2t-1} + x^{2t}$.

Example 5.14 The IEEE 802.15.4a standard uses a RS (63, 55) encoder, implemented over $GF(2^6)$. Here, $n - k = 63 - 55 = 8 = 2t$. The generator polynomial of the encoder can be written as

$$g(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4)(x - \alpha^5)(x - \alpha^6)(x - \alpha^7)(x - \alpha^8)$$
$$= x^8 + \alpha^{43}x^7 + \alpha^{59}x^6 + \alpha^{31}x^5 + \alpha^{10}x^4 + \alpha^{40}x^3 + \alpha^{14}x^2 + \alpha^7x + \alpha^{36}. \quad (5.28)$$

The shift register portion of the encoder is shown in Fig. 5.3.

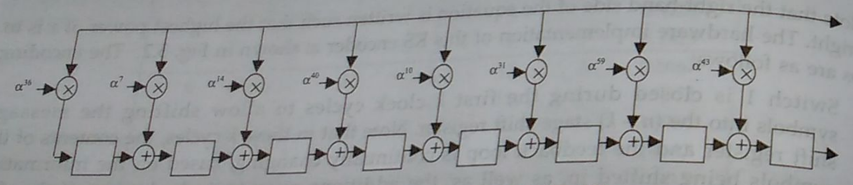


Fig. 5.3 The shift register portion of the encoder for RS(63, 55).

The decoding of RS codes follows the procedure depicted in Fig. 5.1. Each of the sub blocks, i.e., the syndrome calculator block, the block for error location and the block for error magnitudes can be implemented using an FPGA. Since all the arithmetic is done over $GF(q)$, lookup tables are used for addition, multiplication and inversion.

A number of commercial hardware implementations exist for the hardware implementation of RS codes. Many existing systems use *off-the-shelf* integrated circuits that encode and decode

Reed–Solomon codes. These IC's tend to support a certain amount of programmability, for example, RS(255, k) where t = 1 to 16 symbols. A recent trend is towards VHDL or Verilog designs (logic cores or intellectual property cores). These have a number of advantages over standard IC's. A logic core can be integrated with other VHDL or Verilog components and synthesised to an FPGA (Field Programmable Gate Array) or ASIC (Application Specific Integrated Circuit)–this enables so-called "System on Chip" designs where multiple modules can be combined in a single IC. Depending on production volumes, logic cores can often give significantly lower system costs than standard ICs. By using logic cores, a designer avoids the potential need to do a lifetime buy of a Reed–Solomon IC.

Software Implementation Until recently, software implementations in "real time" required too much computational power for all but the simplest of Reed–Solomon codes (i.e. codes with small values of t). The major difficulty in implementing Reed–Solomon codes in software is that general purpose processors do not support Galois field arithmetic operations. For example, to implement a Galois field multiplication in software requires a test for 0, two log table look ups, modulo add and anti-log table look-up. However, careful design together with increases in processor performance, mean that software implementations can operate at relatively high data rates. Table 5.8 gives sample benchmark figures on a 1.6 GHz Pentium PC. These data rates are for decoding only. Encoding is considerably faster since it requires less computation.

Table 5.8 Sample benchmark figures for software decoding of some RS codes.

Code	Data rate	t
RS(255,251)	~ 120 Mbps	2
RS(255,239)	~ 30 Mbps	8
RS(255,223)	~ 10 Mbps	16

5.9 PERFORMANCE OF RS CODES OVER REAL CHANNELS

One may be tempted to believe that as we decrease the code rate of an RS code, the BER performance would improve monotonically. However, in real world communications, the modulation scheme also plays a role with respect to the BER. Thus, both modulation and coding mechanisms have to be considered. One mechanism works to improve error performance while and the other works to degrade it. The improving mechanism is the coding. The greater the redundancy, the greater will be the error correcting capability of the code. The degrading mechanism is the energy reduction per channel symbol (compared to the data symbol) that results from the increased redundancy (and faster signaling in a real time communication system). The reduced symbol energy causes the demodulator to make more errors. Eventually, the second mechanism wins out, and thus at very low code rates the system experiences an error performance degradation.

Scanned by CamScanner

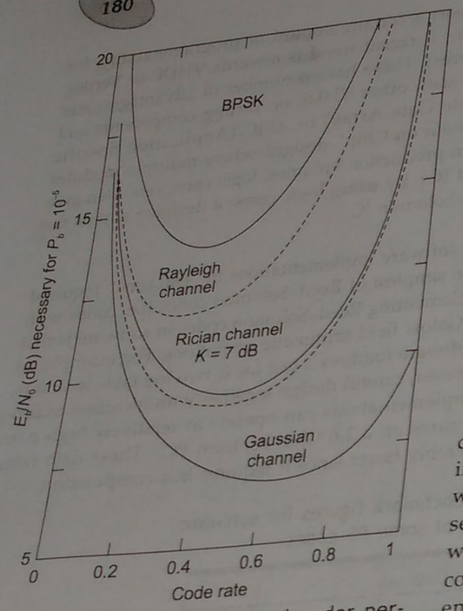


Fig. 5.4 BPSK plus RS (31, k) decoder performance as a function of code rate.

Fig. 5.4 shows the performance of a real time communication system considering both the error control coding and modulation. An RS (31, k) code together with a BPSK modulation scheme has been considered. Here the price paid for error correction coding is bandwidth expansion by a factor equal to the inverse of the code rate. The curves show clear optimum code rates that minimise the required E_b/N_0. We see that the optimum code rate is about 0.6 to 0.7 for a Gaussian channel, 0.5 for a Rician fading channel (with the ratio of direct to reflected received signal power, $K = 7$ dB), and 0.3 for a Rayleigh fading channel.

Energy consumption by the communication system is also becoming an important design constraint in many real-world scenarios, for example in the wireless sensor networks. If we consider the hardware implementation of the error control coding block and the modulator block, the energy expended can be categorized as follows:

(i) Computational energy for channel coding and decoding
(ii) Circuit energy for the modulator and demodulator
(iii) Signal energy (radio energy) for transmitting the redundant bits.

At short distances, the energy consumed in transceiver circuitry and computation is comparable to the signal energy. Thus, the design strategy is to minimise the overall energy consumption in the encoder decoder blocks and the modulator demodulator blocks, taking into consideration the transmitted signal energy as well. Error control codes used to achieve the desired bit error rate, reduce the signal energy due to their coding gain. However, more energy is spent for transmitting the redundant bits and for encoding decoding. Therefore for a particular coding modulation configuration, energy consumption is optimal. This optimal pair will also depend on the distance between the transmitter and receiver, because that will have an impact on the signal (radio) energy.

Example 5.15 Fig. 5.5 shows the energy required to obtain BER = 10^{-5} for different RS(31, k) coders and BPSK modulation (S Chouhan et al). The total energy per data bit is

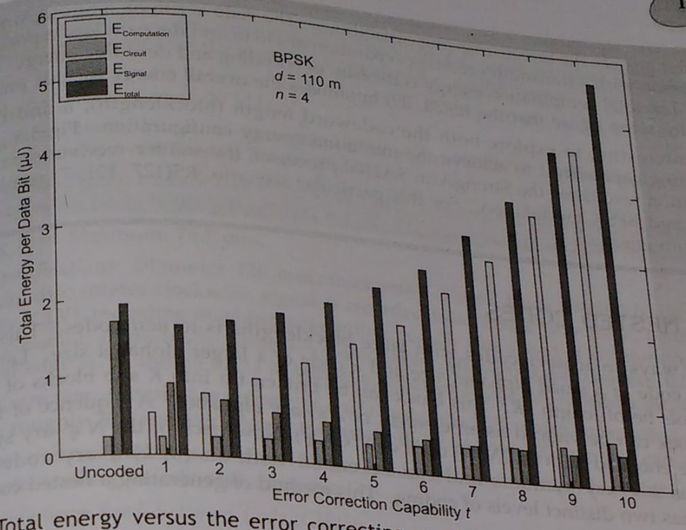


Fig. 5.5 Total energy versus the error correcting capability of RS(31, k).

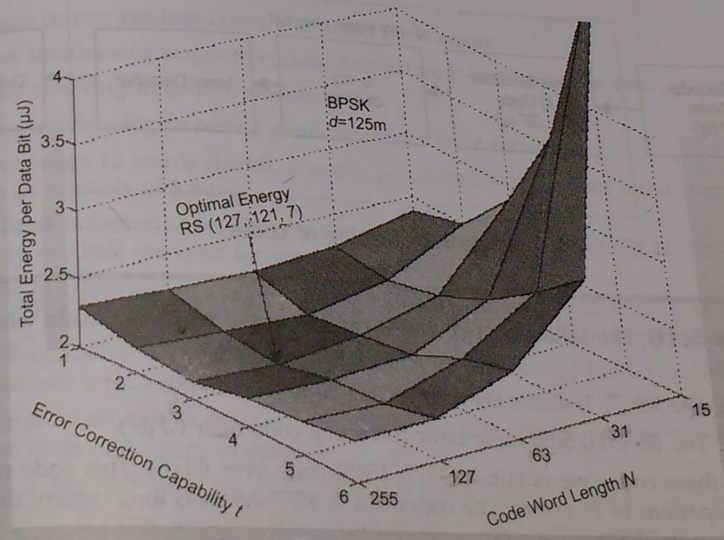


Fig. 5.6 Total energy versus the code word length and the error correcting capability.

plotted against the error correcting capability, t. The code is run on a StrongArm SA1100 processor, with a transmitter receiver separation of 110 m and the path loss exponent equal to 4. The total computation energy is the sum of encoding and decoding energy. It can be seen from the figure that the RS(31, 29) minimises the overall energy for this processor.

It is interesting to explore both the codeword length (blocklength), n and the error correcting capability t to achieve the minimum energy configuration. Fig 5.6 gives the simulation results for the StrongArm SA1100 processor, transmitter–receiver separation of 125 m and BPSK modulation. For this particular scenario, RS(127, 121, 7) results in the minimum energy.

5.10 NESTED CODES

One of the ways to achieve codes with large blocklengths is to *nest* codes. This technique combines a code of a small alphabet size and a code of a larger alphabet size. Let a block of q–ary symbols be of length kK. This block can be broken up into K sub blocks of k symbols. Each sub block can be viewed as an element of a q^k–ary alphabet. A sequence of K such sub blocks can be encoded with an (N, K) code over $GF(q^k)$. Now, each of the N q^k–ary symbols can be viewed as k q–ary symbols and can be coded with an (n, k) q–ary code. Thus, a nested code has two distinct levels of coding. This method of generating a nested code is given in Fig. 5.7.

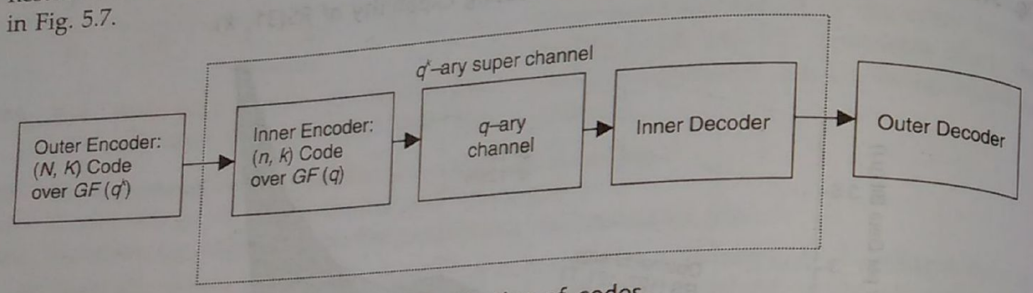


Fig. 5.7 Nesting of codes.

Example 5.16 The following two codes can be nested to form a code with a larger blocklength.

Inner code: The RS (7, 3) *double* error correcting code over $GF(8)$.

Outer code: The RS (511, 505) *triple* error correcting code over $GF(8^3)$.

On nesting these codes we obtain a (3577, 1515) code over $GF(8)$. This code can correct any random pattern of 11 errors. The codeword is 3577 symbols long, where the symbols are the elements of $GF(8)$.

Example 5.17 RS codes are extensively used in the compact discs (CD) for error correction. Below we give the standard Compact Disc digital format.

Sampling frequency: 44.1 kHz *i.e.*, 10 % margin with respect to the Nyquist frequency (audible frequencies below 20 kHz)

Quantisation: 16–bit linear => theoretical SNR about 98 dB (for sinusoidal signal with maximum allowed amplitude), 2's complement.

Signal format: Audio bit rate 1.41 Mbit/s (44.1 kHz × 16 bits × 2 channels), Cross Interleave Reed–Solomon Code (CIRC), total data rate (CIRC, sync, subcode) 2.034 Mbit/s.

Playing time: Maximum 74.7 min.

Disc specifications: Diameter 120 mm, thickness 1.2 mm, track pitch 1.6 μm, one side medium, disc rotates clockwise, signal is recorded from inside to outside, constant linear velocity (CLV), recording maximizes recording density (the speed of revolution of the disc is not constant; it gradually decreases from 500 to 200 r/min), pit is about 0.5 μm wide, each pit edge is '1' and all areas in between, whether inside or outside a pit, are '0's.

Error Correction: A typical error rate of a CD system is 10^{-5}, which means that a data error occurs roughly 20 times per second (bit rate × BER). About 200 errors/s can be corrected.

Sources of errors: Dust, scratches, fingerprints, pit asymmetry, bubbles or defects in substrate, coating defects and dropouts.

Cross Interleave Reed–Solomon Code (CIRC)

- C2 can effectively correct burst errors.
- C1 can correct random errors and detect burst errors.
- Three interleaving stages to encode data before it is placed on a disc.
- Parity checking to correct random errors.
- Cross interleaving to permit parity to correct burst errors.

1. Input stage: 12 words (16–bit, 6 words per channel) of data per input frame divided into 24 symbols of 8 bits.
2. C2 Reed–Solomon code: 24 symbols of data are encoded into a (28,24) RS code and 4 parity symbols are used for error correction.
3. Cross interleaving: to guard against the burst errors, separate error correction codes, one code can check the accuracy of another, error correction is enhanced.
4. C1 Reed–Solomon code: cross interleaved 28 symbols of the C2 code are encoded again into a (32,28) R S code (4 parity symbols are used for error correction).
5. Output stage: half of the code word is subject to a 1–symbol delay to avoid 2–symbol error at the boundary of symbols.

Performance of CIRC: Both RS coders (C1 and C2) have four parities, and their minimum distance is 5. If error location is not known, up to two symbols can be corrected. If the errors exceed the correction limit, they are concealed by interpolation. Since even

Scanned by CamScanner

numbered sampled data and odd numbered sampled data are interleaved as much as possible, CIRC can conceal long burst errors by simple linear interpolation.

- Maximum correctable burst length is about 4000 data bits (2.5 mm track length).
- Maximum correctable burst length by interpolation in the worst case is about 12320 data bits (7.7 mm track length).

Sample interpolation rate is one sample every 10 hours at BER (Bit Error Rate) = 10^{-4} and 1000 samples at BER = 10^{-3}. Undetectable error samples (clicks) are less than one every 750 hours at BER = 10^{-3} and negligible at BER = 10^{-4}.

5.11 CONCLUDING REMARKS

The class of BCH codes were discovered independently by Hocquenghem in 1959 and Bose and Ray Chaudhuri in 1960. The BCH codes constitute one of the most important and powerful classes of linear block codes, which are cyclic.

The Reed–Solomon codes were discovered by Irving S. Reed and Gustave Solomon who published a five-page paper in the Journal of the Society for Industrial and Applied Mathematics in 1960. They were staff members at MIT's Lincoln Laboratory at that time. The paper was titled *"Polynomial Codes over Certain Finite Fields"*. Despite their advantages, Reed–Solomon codes did not go into use immediately after their invention. They had to wait for the hardware technology to catch up. In 1960, there was no such thing as fast digital electronics, at least not by today's standards. The Reed–Solomon paper suggested some nice ways to process data, but nobody knew if it was practical or not, and in 1960 it probably wasn't practical.

Eventually technology did catch up, and numerous researchers began to work on implementing the codes. One of the key individuals was Elwyn Berlekamp, a professor of Electrical Engineering at the University of California at Berkeley, who invented an efficient algorithm for decoding the Reed–Solomon code. Berlekamp's algorithm was used by Voyager II and is the basis for decoding in CD players. Many other bells and whistles (some of fundamental theoretic significance) have also been added. Compact discs, for example, use a version called cross interleaved Reed–Solomon code, or CIRC. RS codes are also commonly used in wireless channels where burst errors are encountered.

SUMMARY

- *A primitive element of GF(q) is an element α such that every field element except zero can be expressed as a power of α. A field can have more than one primitive element.*
- *A primitive polynomial $p(x)$ over GF(q) is a prime polynomial over GF(q) with the property that in the extension field constructed modulo $p(x)$, the field element represented by x is a primitive element.*

- A blocklength n of the form $n = q^m - 1$ is called a primitive blocklength for a code over GF(q). A cyclic code over GF(q) of primitive blocklength is called a primitive cyclic code.
- It is possible to factor $x^{q^m-1} - 1$ in the extension field GF(q^m) to get $x^{q^m-1} - 1 = \prod_j (x - \beta_j)$, where β_j ranges over all the non-zero elements of GF(q^m). This implies that each of the polynomials $f_i(x)$ can be represented in GF(q^m) as a product of some of the linear terms, and each β_j is a zero of *exactly one* of the $f_i(x)$. This $f_i(x)$ is called the minimal polynomial of β_j.
- Two elements of GF(q^m) that share the same minimal polynomial over GF(q) are called conjugates with respect to GF(q).
- BCH codes defined over GF(q) with blocklength $q^m - 1$ are called primitive BCH codes.
- To determine the generator polynomial of a t–error correcting BCH code for a primitive blocklength $n = q^m - 1$, (i) Choose a prime polynomial of degree m and construct GF(q^m), (ii) find $f_i(x)$, the minimal polynomial of α^i for $i = 1, .., p$. (iii) obtain the generator polynomial $g(x) = \text{LCM}\ [f_1(x)\ f_2(x), ..., f_{2t}(x)]$. Codes designed in this manner can correct at least t errors. In many cases the codes will be able to correct more than t errors. For this reason, $d = 2t + 1$ is called the designed distance of the code, and the minimum distance $d^* \geq 2t + 1$.
- Steps for decoding BCH codes:
  (1) As a trial value, set $v = t$ and compute the determinant of the matrix of syndromes, **M**. If the determinant is zero, set $v = t - 1$. Again compute the determinant of **M**. Repeat this process until a value of v is found for which the determinant of the matrix of syndromes is non-zero. This value of v is the actual number of errors that occurred.
  (2) Invert the matrix ***M*** and find the coefficients of the error locator polynomial $\Lambda(x)$.
  (3) Solve $\Lambda(x) = 0$ to obtain the zeros and from them compute the error locations $X_1, X_2, \ldots, X_v$. If it is a binary code, stop (because the magnitudes of error are unity).
  (4) If the code in not binary, go back to the system of equations:

$$\begin{aligned} S_1 &= Y_1X_1 + Y_2X_2 + \ldots + Y_vX_v \\ S_2 &= Y_1X_1^2 + Y_2X_2^2 + \ldots + Y_vX_v^2 \\ &\vdots \qquad\qquad\qquad \vdots \\ S_{2t} &= Y_1X_1^{2t} + Y_2X_2^{2t} + \ldots + Y_vX_v^{2t} \end{aligned}$$

  Since the error locations are now known, these form a set of $2t$ *linear* equations. These can be solved to obtain the error magnitudes.
- The generator polynomial for a t error correcting RS code is

$$g(x) = \text{LCM}\ [f_1(x)\ f_2(x), ..., f_{2t}(x)] = (x - \alpha)(x - \alpha^2) \ldots (x - \alpha^{2t-1})(x - \alpha^{2t}).$$

  Hence, the degree of the generator polynomial will always be $2t$. Thus, the RS code satisfies

$$n - k = 2t.$$

- A Reed–Solomon code is a maximum distance separable (MDS) code and its minimum distance is $n - k + 1$.
- One of the ways to achieve codes with large blocklengths is to *nest* codes. This technique combines a code of a small alphabet size and a code of a larger alphabet size. Let a block of

Scanned by CamScanner

q–ary symbols be of length kK. This block can be broken up into K sub-blocks of k symbols. Each sub block can be viewed as an element of a q^k–ary alphabet

> "Once you eliminate the impossible, whatever remains, no matter how improbable, must be the truth."
>
> –Sherlock Holmes (by Sir Arthur Conan Doyle, 1859–1930)

PROBLEMS

5.1 Construct $GF(9)$ from $GF(3)$ using an appropriate primitive polynomial.

5.2 (i) Find the generator polynomial $g(x)$ for a double error correcting *ternary* BCH code of blocklength 8. What is the code rate of this code? Compare it with the (11, 6) ternary Golay code with respect to the code rate and the minimum distance.

(ii) Next, find the generator polynomial $g(x)$ for a triple error correcting *ternary* BCH code of blocklength 26.

5.3 Find the generator polynomial $g(x)$ for a single error correcting binary BCH code of blocklength 31. Use the primitive polynomial $p(x) = x^5 + x^2 + 1$ to construct $GF(32)$. What is the minimum distance of this code?

5.4 What are the possible generator polynomials for an RS code over $GF(2)$?

5.5 Find the generator polynomials and the minimum distance for the following codes:

(i) RS (15, 11) code

(ii) RS (15, 7) code

(iii) RS (31, 21) code.

5.6 Consider the triple error correcting BCH (15, 5) code over $GF(16)$. Given the received polynomial $v(x) = x^7 + x^2$, what is the transmitted polynomial?

5.7 Show that every BCH code is a sub–field subcode of a Reed–Solomon code of the same designed distance. Under what condition is the code rate of the BCH code equal to that of the RS code?

5.8 Show that the Vandermonde matrix, defined as

$$A = \begin{bmatrix} 1 & 1 & \dots & 1 \\ X_1 & X_2 & \dots & X_\mu \\ X_1^2 & X_2^2 & \dots & X_\mu^2 \\ \\ X_1^{\mu-1} & X_2^{\mu-1} & \dots & X_\mu^{\mu-1} \end{bmatrix} \tag{5.29}$$

has a non-zero determinant if and only if all of the X_i for $i = 1, 2, \dots \mu$ are distinct.

5.9 Consider the code over $GF(11)$ with a parity check matrix

$$H = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 2 & 3 & \dots & 10 \\ 1 & 2^2 & 3^2 & \dots & 10^2 \\ 1 & 2^3 & 3^3 & \dots & 10^3 \\ \vdots & & & & \vdots \\ 1 & 2^8 & 3^8 & \dots & 10^8 \end{bmatrix} \tag{5.30}$$

(i) Find the minimum distance of this code.

(ii) Show that this is an optimal code with respect to the Singleton bound.

5.10 Consider the code over $GF(11)$ with a parity check matrix

$$H = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & 2 & 3 & 4 & \dots & 10 \\ 1 & 2^2 & 3^2 & 4^2 & \dots & 10^2 \\ 1 & 2^3 & 3^3 & 4^3 & \dots & 10^3 \\ 1 & 2^4 & 3^4 & 4^4 & \dots & 10^4 \\ 1 & 2^5 & 3^5 & 4^5 & \dots & 10^5 \end{bmatrix} \tag{5.31}$$

(i) Show that the code is a triple error correcting code.

(ii) Find the generator polynomial for this code.

5.11 The ECMA standard for high speed Ultra Wideband (UWB) communication uses the RS(255, 249) code over $GF(2^8)$.

(i) Find the generator polynomial for this code.

(ii) Give the shift register based implementation of the systematic RS(255, 249) code.

COMPUTER PROBLEMS

5.12 Write a computer program which takes in the coefficients of a primitive polynomial, the values of q and m, and then constructs the extension field $GF(q^m)$.

5.13 Write a computer program that performs addition and multiplication over $GF(2^m)$, where m is an integer.

5.14 Find the generator polynomial $g(x)$ for a binary BCH code of blocklength 63. Use the primitive polynomial $p(x) = x^6 + x + 1$ to construct GF(64). What is the minimum distance of this code?

5.15 Write a program that performs BCH decoding given n, q, t and the received vector.

5.16 Write a program that outputs the generator polynomial of the Reed Solomon code with the codeword length n and the message length k. A valid n should be $2^M - 1$, where M is an integer not less than 3. The program should also list the minimum distance of the code.

Scanned by CamScanner

5.17 Write a computer program that performs the two level RS coding as done in a standard compact disc.

5.18 Write a computer program for the systematic encoder for $RS(63, 55)$ over $GF(2^6)$ using the shift register implementation.

Convolutional Codes

6

The shortest path between two truths in the real domain passes through the complex domain.

—Jacques Hadamard (1865–1963)

6.1 INTRODUCTION TO CONVOLUTIONAL CODES

So far we have studied block codes, where a block of k information symbols are encoded into a block of n coded symbols. There is always a one-to-one correspondence between the uncoded block of symbols (information word) and the coded block of symbols (codeword). This method is particularly useful for high data rate applications, where the incoming stream of uncoded data is first broken into blocks, encoded, and then transmitted (Fig. 6.1). A large blocklength is important because of the following reasons:

(i) Many of the good codes that have large distance properties are of large blocklengths (*e.g.*, the RS codes),

(ii) Larger blocklengths imply that the encoding overhead is small.

However, very large blocklengths have the disadvantage that unless the entire block of encoded data is received at the receiver, the decoding procedure cannot start, which may result in delays. In contrast, there is another coding scheme in which much smaller blocks of uncoded data of length k_0 are used. These are called **Information frames**. An information frame typically contains just a few symbols, and can have as few as just one symbol. These information frames are encoded into **Codeword Frames** of length n_0. However, just one information frame is not

used to obtain the codeword frame. Instead, the current information frame with previous m information frames are used to obtain a single codeword frame. This implies that such encoders have *memory*, which retain the previous m incoming information frames. The codes that are obtained in this fashion are called **Tree Codes**. An important subclass of tree codes, used frequently in practice, is called **Convolutional Codes**. Up to now, all the decoding techniques discussed are algebraic and are memoryless *i.e.*, decoding decisions are based only on the current codeword. Convolutional codes make decisions based on past information *i.e.* memory is required.

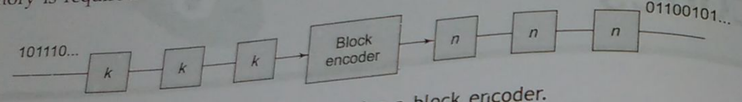


Fig. 6.1 Encoding using a block encoder.

In this chapter, we shall start with an introduction to **Tree** and **Trellis codes**. We will then develop the necessary mathematical tools to construct **Convolutional Codes**. We will see that convolutional codes can be easily represented by polynomials. Next, we will give a matrix description of convolutional codes. The chapter will next discuss the famous **Viterbi decoding technique**. We shall conclude this chapter by giving an introduction to **Turbo Coding and Decoding**. We will also briefly discuss **Interleaver** design for Turbo codes.

6.2 TREE CODES AND TRELLIS CODES

We assume that we have an infinitely long stream of incoming symbols (thanks to the volumes of information to be sent these days, it is not a bad assumption). This stream of symbols is first broken up into segments of k_0 symbols. Each segment is called an information frame, as mentioned earlier. The encoder consists of two parts (Fig. 6.2):

(i) memory, which basically is a shift register, and
(ii) a logic circuit.

The memory of the encoder can store m information frames. Each time a new information frame arrives, it is shifted into the shift register and the oldest information frame is discarded. At the end of any frame time the encoder has m most recent information frames in its memory, which corresponds to a total of mk_0 information symbols.

When a new frame arrives, the encoder computes the codeword frame using this new frame that has just arrived and the stored previous m frames. The computation of the codeword frame is done using the logic circuit. This codeword frame is then shifted out. The oldest information frame in the memory is then discarded and the most recent information frame is shifted into the memory. The encoder is now ready for the next incoming information frame. Thus for every information frame (k_0 symbols) that come in, the encoder generates a codeword frame (n_0 symbols). It should be observed that the same information frame may not generate the same

codeword frame because the codeword frame also depends on the m previous information frames.

Definition 6.1 The **Constraint Length** of a shift register encoder is defined as the number of symbols it can store in its memory. We shall give a more formal definition of constraint length later in this chapter.

If the shift register encoder stores m previous information frames of length k_0, the constraint length of this encoder $v = mk_0$.

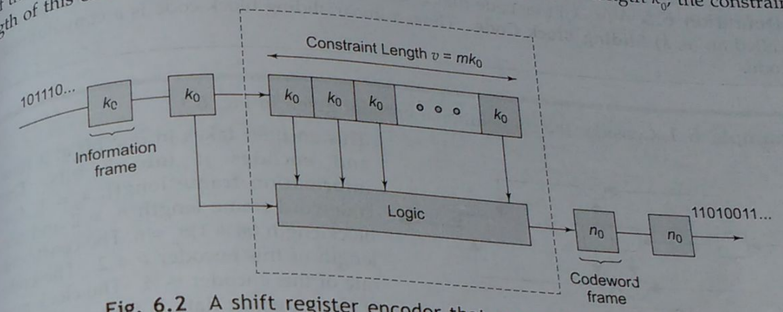


Fig. 6.2 A shift register encoder that generates a tree code.

Definition 6.2 The infinite set of all infinitely long codewords obtained by feeding every possible input sequence to a shift register encoder is called an (n_0, k_0) **Tree Code**. The rate of this tree code is defined as

$$R = \frac{k_0}{n_0} \tag{6.1}$$

A more formal definition is that a (n_0, k_0) **Tree Code** is a mapping, from the set of semi infinite sequences of elements of $GF(q)$ into itself such that, if for any M, two semi infinite sequences agree in the first Mk_0 components, then their images agree in the first Mn_0 components.

Definition 6.3 The **Wordlength** of a shift register encoder is defined as $k = (m + 1)k_0$. The **Blocklength** of a shift register encoder is defined as

$$n = (m + 1)n_0 = k\frac{n_0}{k_0}$$

Note that the code rate

$$R = \frac{k_0}{n_0} = \frac{k}{n}$$

Scanned by CamScanner

Normally, for practical shift register encoders, the information frame length k_0 is small (usually less than 5). Therefore, it is difficult to obtain the code rate R of tree codes close to unity, as is possible with block codes (e.g., RS codes).

Definition 6.4 A (n_0, k_0) tree code that is linear, timeinvariant, and has a finite wordlength $k = (m + 1)k_0$ is called an (n, k) **Convolutional Code**.

Definition 6.5 A (n_0, k_0) tree code that is time-invariant and has a finite wordlength k is called an (n, k) **Sliding Block Code**. Thus, a linear sliding block code is a convolutional code.

Example 6.1 Consider the convolutional encoder given in Fig. 6.3.

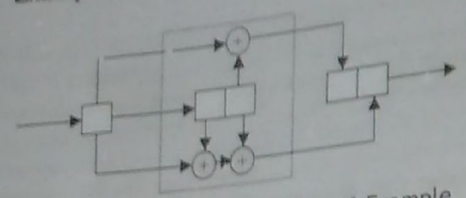
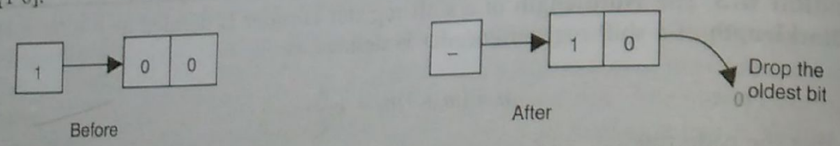


Fig. 6.3 Convolutional encoder of Example 6.1. Here $k_0 = 1$, $n_0 = 2$ and $m = 2$.

This encoder takes in one bit at a time and encodes it into 2 bits. The information frame length $k_0 = 1$, the codeword frame length $n_0 = 2$ and the blocklength $(m + 1)n_0 = 6$. The constraint length of this encoder $v = 2$. The code rate of this encoder is ½. The clock rate of the outgoing data is twice as fast as that of the incoming data. The adders are binary adders, and from the point of view of circuit implementation, are simply XOR gates.

Let us assume that the initial state of the shift register is [0 0]. Now, either '0' will come or '1' will come as the incoming bit. Suppose '0' comes. On performing the logic operations, we see that the computed value of the codeword frame is [0 0]. The 0 will be pushed into the memory (shift register) and the rightmost '0' will be dropped. The state of the shift register remains [0 0]. Next, let '1' arrive at the encoder. Again we perform the logic operations to compute the codeword frame. This time we obtain [1 1]. So, this will be pushed out as the encoded frame. The incoming '1' will be shifted into the memory, and the rightmost bit will be dropped. So the new state of the shift register will be [1 0].



Again, there are two possibilities regarding the incoming bit: '0' or '1'. It is possible to construct a table which lists all the possibilities (Table 6.1).

Table 6.1 The incoming and outgoing bits in the convolutional encoder given in Example 6.1.

Incoming Bit	Current State of the encoder		Outgoing Bits	
0	0	0	0	0
1	0	0	1	1
0	0	1	1	1
1	0	1	0	0
0	1	0	0	1
1	1	0	1	0
0	1	1	1	0
1	1	1	0	1

We observe that there are only $2^2 = 4$ possible states of the shift register. So, we can construct the state diagram of the encoder as shown in Fig. 6.4. The bits associated with each arrow represent the incoming bit. It can be seen that the same incoming bit gets encoded *differently*, depending on the current state of the encoder. This is different from the linear block codes studied in previous chapters where there is always a one-to-one correspondence between the incoming uncoded block of symbols (information word) and the coded block of symbols (codeword).

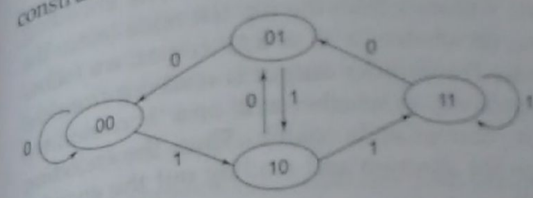


Fig. 6.4 The state diagram for the encoder in Example 6.1.

The same information contained in the state diagram can be conveyed usefully in terms of a graph called the **Trellis Diagram**. A trellis is a graph whose nodes are in a rectangular grid, which is semi infinite to the right. Hence, these codes are also called **Trellis Codes**. The number of nodes in each column is finite. The following example gives an illustration of a trellis diagram.

Example 6.2 The trellis diagram for the convolutional encoder discussed in Example 6.1 is given in Fig. 6.5.

Every node in the trellis diagram represents a state of the shift register. Since the rate of the encoder is ½, one bit at a time is processed by the encoder. The incoming bit can be either a '0' or a '1', therefore, there are two branches emanating form each node. The top branch represents the input as '0' and the lower branch corresponds to '1'. This is true for a binary trellis diagram. In general, one would also label each branch with the input symbol to which it corresponds. Normally, the nodes that cannot be reached by starting at the top left node and moving only to the right are not shown in the trellis diagram. Corresponding to a certain state and a particular incoming bit, the encoder will produce an output. The output of the encoder is written on top of that branch. Thus, a trellis

Scanned by CamScanner

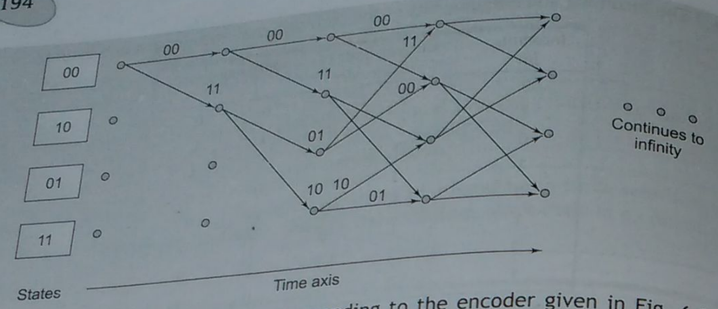


Fig. 6.5 The trellis diagram corresponding to the encoder given in Fig. 6.3.

diagram gives a very easy method to encode a stream of input data. The encoding procedure using a trellis diagram is as follows. We start from the top left node (since the initial state of the encoder is [0 0]). Depending on whether a '0' or a '1' comes, we follow the upper or the lower branch to the next node. The encoder output is read out from the top of the branch being traversed. Again, depending on whether a '0' or a '1' comes, we follow the upper or the lower branch from the current node (state). Thus, the encoding procedure is simply following the branches on the diagram and reading out the encoder outputs that are written on top of each branch.

Suppose we have to encode the bit stream 1 0 0 1 1 0 1 ... . We start from the top left node and follow the branches dictated by this input bit stream, *i.e.*, down, up, up, down, down, up, down ..., because '0' implies take the upper branch (up) and '1' implies take the lower branch (down). The encoded path in the trellis diagram is shown in Fig. 6.6. The encoded sequence can be read out from the diagram as 11 01 11 11 10 10 00 ... . *It can be seen that there is a one-to-one correspondence between the encoded sequence and a path in the trellis diagram.* Should the decoding procedure, then, just search for the most likely path in the trellis diagram? The answer is yes, as we shall see further along in this chapter.

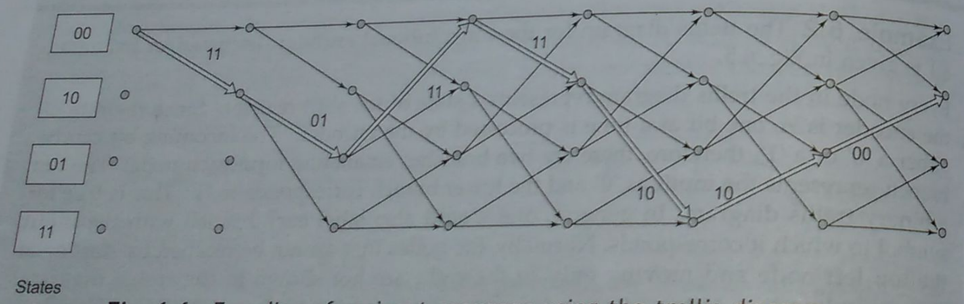


Fig. 6.6 Encoding of an input sequence using the trellis diagram.

6.3 POLYNOMIAL DESCRIPTION OF CONVOLUTIONAL CODES (ANALYTICAL REPRESENTATION)

In contrast to the two pictorial representations of the convolutional codes (the state diagram and the trellis diagram), there is a very useful analytical representation of convolutional codes. The representation makes use of the delay operator, D. We have earlier seen a one-to-one correspondence between a word (vector) and a polynomial. The delay operator is used in a similar manner. For example, consider the word 10100011 with the oldest digit at the left. The analytical representation (sometimes referred to as the *transform*) of this information word I(D) will be

$$I(D) = 1 + D^2 + D^6 + D^7. \tag{6.2}$$

The **Indeterminate** D is the number of time units of delay of that digit relative to the chosen time origin, which is usually taken to coincide with the first bit. In general, the sequence $i_0, i_1, i_2, i_3, \ldots$ has the representation $i_0 + i_1D + i_2D^2 + i_3D^3 + \ldots$

A convolutional code over $GF(q)$ with a wordlength $k = (m + 1)k_0$, a blocklength $n=(m + 1)n_0$ and a constraint length $v = mk_0$ can be encoded by sets of finite impulse response (FIR) filters. Each set of filters consist of k_0 FIR filters in $GF(q)$. The input to the encoder is a sequence of k_0 symbols, and the output is a sequence of n_0 symbols. Fig. 6.7 shows an encoder for a binary convolutional code with $k_0 = 1$ and $n_0 = 4$.

Fig. 6.8 shows a convolutional filter with $k_0 = 2$ and $n_0 = 4$.

Each of the FIR filters can be represented by a polynomial of degree $\leq m$. The input stream of symbols can also be represented by a polynomial. The operation of the filter can then simply be represented as the multiplication of the two polynomials. Thus the encoder (and hence the code) can be represented by a *set of polynomials*. These polynomials are called the **Generator Polynomials** of the code. This set contains k_0n_0 polynomials. The largest degree of a polynomial in this set of generator polynomials is m. We remind the reader that a block code was represented by a *single* generator polynomial. Thus we can define a **Generator Polynomial Matrix** of size $k_0 \times n_0$ for a convolutional code as follows:

$$G(D) = [g_{ij}(D)]. \tag{6.3}$$

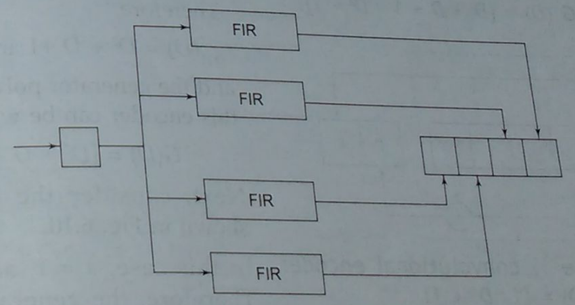


Fig. 6.7 A convolutional encoder in terms of FIR filters with $k_0 = 1$ and $n_0 = 4$.

Scanned by CamScanner

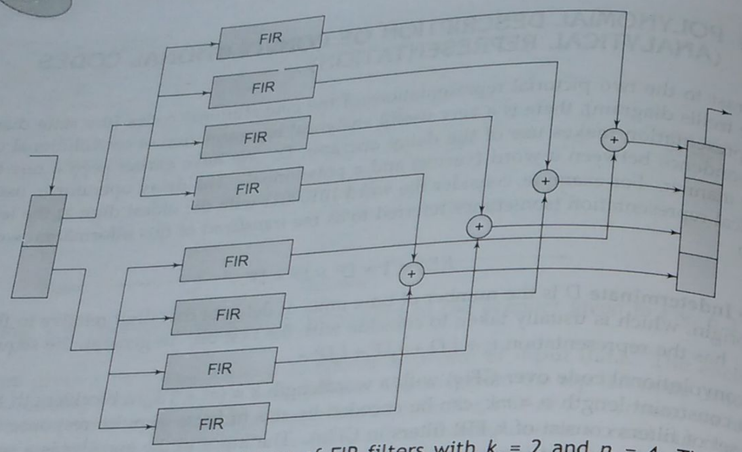


Fig. 6.8 A convolutional encoder in terms of FIR filters with $k_0 = 2$ and $n_0 = 4$. The rate of this encoder is $R = ½$.

Example 6.3 *Consider the convolutional encoder given in Fig. 6.9.*

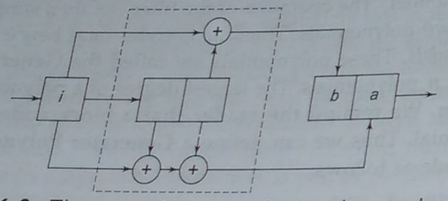


Fig. 6.9 The rate ½. convolutional encoder with $G(D) = [D^2 + D + 1 \quad D^2 + 1]$.

The first bit of the output $a = i_{n-2} + i_{n-1} + i_n$ and the second bit of the output $b = i_{n-2} + i_n$, where i_{n-l} represents the input that arrived l time units earlier. Let the input stream of symbols be represented by a polynomial. We know that multiplying any polynomial by D corresponds to a single cyclic right-shift of the elements. Therefore,

$$g_{11}(D) = D^2 + D + 1 \text{ and } g_{12}(D) = D^2 + 1$$

and the generator polynomial matrix of this encoder can be written as

$$G(D) = [D^2 + D + 1 \quad D^2 + 1]$$

Next, consider the encoder circuit shown in Fig. 6.10.

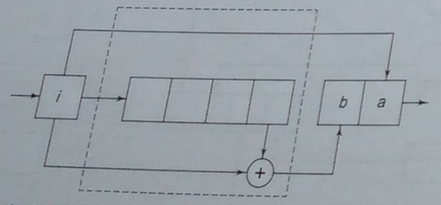


Fig. 6.10 The rate ½ convolutional encoder with $G(D) = [1 \quad D^4 + 1]$

In this case, $a = i_n$ and $b = i_{n-4} + i_n$. Therefore, the generator polynomial

matrix of this encoder can be written as

$$G(D) = [1 \quad D^4 + 1].$$

Note that, the first k_0 bits ($k_0 = 1$) of the codeword frame is identical to the information frame. Hence, this is a **Systematic Convolutional Encoder.**

Example 6.4 Consider the systematic convolutional encoder represented by the following circuit (Fig. 6.11).

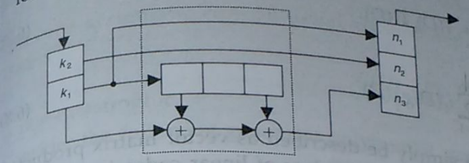


Fig. 6.11 The rate 1/3 convolutional encoder for Example 6.4.

The generator polynomial matrix of this encoder can be written as

$$G(D) = \begin{bmatrix} g_{11}(D) & g_{12}(D) & g_{13}(D) \\ g_{21}(D) & g_{22}(D) & g_{23}(D) \end{bmatrix} = \begin{bmatrix} 1 & 0 & D^3+D+1 \\ 0 & 1 & 0 \end{bmatrix}.$$

It is easy to write the generator polynomial matrix by visual inspection. The element in the ith row and j^{th} column of the matrix represents the relation between the ith input bit and the jth output bit. To write the generator polynomial for the (i^{th}, j^{th}) entry of the matrix, just trace the route from the ith input bit to the jth output bit. If no path exists, the generator polynomial is the zero polynomial, as in the case of $g_{12}(D)$, $g_{21}(D)$ and $g_{23}(D)$. If only a direct path exists without any delay elements, the value of the generator polynomial is unity, as in $g_{11}(D)$ and $g_{22}(D)$. If the route from the ith input bit to the jth output bit involves a series of memory elements (delay elements), represent each delay, by an additional power of D, as in $g_{13}(D)$. Note that three of the generator polynomials in the set of generator polynomials are zero. When k_0 is greater than 1, it is not unusual for some of the generator polynomials to be zero polynomials.

We can now give formal definitions of the wordlength, the blocklength and the constraint length of a convolutional encoder.

Definition 6.6 Given the generator polynomial matrix $[g_{ij}(D)]$ of a convolutional code:

(i) The **Wordlength** of the code is

$$k = k_0 \max_{i,j}[\deg g_{ij}(D) + 1]. \tag{6.4}$$

(ii) The **Blocklength** of the code is

$$n = n_0 \max_{i,j}[\deg g_{ij}(D) + 1]. \tag{6.5}$$

(iii) The **Constraint Length** of the code is

$$v = \sum_{i=1}^{k_0} \max_j [\deg g_{ij}(D)]. \tag{6.6}$$

Recall that the input message stream $i_0, i_1, i_2, i_3 \ldots$ has the polynomial representation $I(D) = i_0 + i_1 D + i_2 D^2 + i_3 D^3 + \ldots + i_{k_0} D^{k_0}$ and the codeword polynomial can be written as $C(D) = c_0 + c_1 D + c_2 D^2 + c_3 D^3 + \ldots + c_{n_0} D^{n_0}$. The encoding operation can simply be described as a vector matrix product,

$$C(D) = I(D)\, G(D). \tag{6.7}$$

or equivalently,

$$c_j(D) = \sum_{i=1}^{k_0} i_i(D) g_{ij}(D). \tag{6.8}$$

Observing that the encoding operation can simply be described as vector matrix product, it can be easily shown that convolutional codes belong to the class of linear codes (exercise).

Definition 6.7 A **Parity Check Matrix** $H(D)$ is an $(n_0 - k_0)$ by n_0 matrix of polynomials that satisfies

$$G(D)H(D)^T = 0. \tag{6.9}$$

and the **Syndrome Polynomial Vector** which is a $(n_0 - k_0)$–the component row vector is given by

$$s(D) = V(D)H(D)^T. \tag{6.10}$$

where $V(D)$ is the received word.

Definition 6.8 A **Systematic Encoder** for a convolutional code has the generator polynomial matrix of the form

$$G(D) = [I \mid P(D)]. \tag{6.11}$$

where I is a k_0 by k_0 identity matrix and $P(D)$ is a k_0 by $(n_0 - k_0)$ matrix of polynomials. The parity check polynomial matrix for a systematic convolutional encoder is

$$H(D) = [-P(D)^T \mid I]. \tag{6.12}$$

where I is a $(n_0 - k_0)$ by $(n_0 - k_0)$ identity matrix. It follows that

$$G(D)H(D)^T = 0. \tag{6.13}$$

Definition 6.9 A convolutional code whose generator polynomials $g_1(D), g_2(D), \ldots, g_{n_0}(D)$ satisfy

$$\mathrm{GCD}[g_1(D), g_2(D), \ldots, g_{n_0}(D)] = D^a. \tag{6.14}$$

for some a is called a **Non-catastrophic Convolutional Code**. Otherwise it is called a **Catastrophic Convolutional Code**.

Without loss of generality, one may take $a = 0$, *i.e.*, $D^a = 1$.

Thus the task of finding a non-catastrophic convolutional code is equivalent to finding a good set of relatively prime generator polynomials. Relatively prime polynomials can be easily found by computer searches. However, what is difficult to find is a set of relatively prime generator polynomials that have good error correcting capabilities.

Example 6.5 All systematic codes are non-catastrophic because for them $g_1(D) = 1$ and therefore,

$$\mathrm{GCD}[1, g_2(D), \ldots, g_{n_0}(D)] = 1.$$

Thus the systematic convolutional encoder represented by the generator polynomial matrix

$$G(D) = [1 \quad D^4 + 1]$$

is non-catastrophic.

Consider the following generator polynomial matrix of a binary convolutional encoder

$$G(D) = [D^2 + 1 \quad D^4 + 1].$$

We observe that

$$(D^2 + 1)^2 = D^4 + (D^2 + D^2) + 1 = D^4 + 1,$$

for binary encoder (modulo 2 arithmetic). Hence,

$$\mathrm{GCD}[g_1(D), g_2(D)] = D^2 + 1 \neq 1.$$

Therefore, this is a catastrophic encoder.

Next consider the following generator polynomial matrix of a non systematic binary convolutional encoder

$$G(D) = [D^2 + 1 \quad D^4 + D^2 + 1].$$

The two generator polynomials are relatively prime, *i.e.*,

$$\mathrm{GCD}[g_1(D), g_2(D)] = 1.$$

Hence this represents a non-catastrophic convolutional encoder.

In the next section, we shall see that the distance notion of convolutional codes is an important parameter, and determines the number of errors a convolutional code can correct.

6.4 DISTANCE NOTIONS FOR CONVOLUTIONAL CODES

Recall that for block codes, the concept of the (Hamming) distance between two codewords provided a way of quantitatively describing how different the two vectors were. It was shown that a good code must possess the maximum possible minimum distance. Convolutional codes also have a distance concept that determines how good the code is.

When a codeword from a convolutional encoder passes through a channel, errors occur from time to time. The job of the decoder is to correct these errors by processing the received vector.

Scanned by CamScanner

In principle, the convolutional codeword is infinite in length. However, the decoding decisions are made on codeword *segments* of a finite length. The number of symbols that the decoder can store is called the **Decoding Window Width**. Regardless of the size of these finite segments (decoding window width), there is always some effect of the previous frames on the current frame because of the memory of the encoder. In general, one gets a better performance by increasing the decoding window width, but one eventually reaches the point of diminishing returns.

Most of the decoding procedures for decoding convolutional codes work by focussing on the errors in the first frame. If this frame can be corrected and decoded, then the first frame of the information is known at the receiver end. The effect of these information symbols on the subsequent information frames can be computed and subtracted from subsequent codeword frames. Thus the problem of decoding the second codeword frame is the same as the problem of decoding the first frame. We extend this logic further. If the first f frames have been decoded successfully, the problem of decoding the $(f+1)^{th}$ frame is the same as the problem of decoding the first frame. But what happens if an in-between frame is not decoded correctly? If it is possible for a single decoding error event to induce an infinite number of *additional* errors, then the decoder is said to be subject to **Error Propagation**. In the case where the decoding algorithm is responsible for error propagation, it is termed as **Ordinary Error Propagation**. In the case where the poor choice of catastrophic generator polynomials cause error propagation, we call it **Catastrophic Error Propagation**.

Definition 6.10 The lth **minimum distance** d_l^* of a convolutional code is equal to the smallest Hamming distance between any two initial codeword segments l frames long that are not identical in the initial frame. If $l = m + 1$, then this $(m+1)^{th}$ minimum distance is called the **Minimum Distance** of the code and is denoted by d^*, where m is the number of information frames that can be stored in the memory of the encoder. In literature, the minimum distance is also denoted by d_{min}.

We note here that a convolutional code is a linear code. Therefore, one of the two codewords used to determine the minimum distance of the code can be chosen to be the all-zero codeword. The lth minimum distance is then equal to the weight of the smallest-weight codeword segment l frames long that is non-zero in the first frame (*i.e.*, different from the all-zero frame).

Suppose the lth minimum distance of a convolutional code is d_l^*. The code can correct t errors occurring in the first l frames provided

$$d_l^* \geq 2t+1 . \tag{6.15}$$

Next, put $l = m + 1$, in which case $d_l^* = d_{m+1}^* = d^*$. The code can correct t errors occurring in the first blocklength $n = (m+1)n_0$ provided

$$d^* \geq 2t+1 . \tag{6.16}$$

Example 6.6 Consider the convolutional encoder of Example 6.1 (Fig. 6.3). The trellis diagram for the encoder is given in Fig. 6.12.

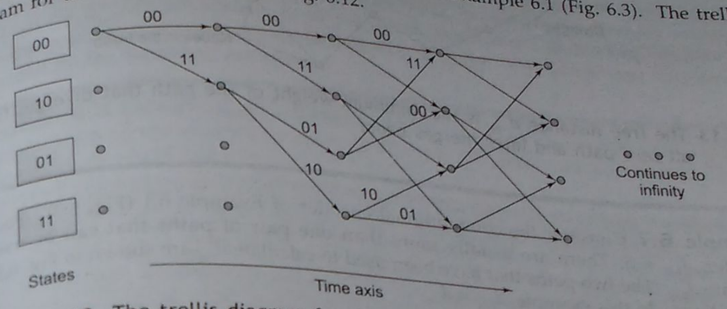


Fig. 6.12 The trellis diagram for the convolutional encoder of Example 6.1.

In this case $\overline{\wedge} = 2$, $d_2^* = 3$, $d_3^* = 5$, $d_4^* = 5$, ... We observe that $d_i^* = 5$ for $i \geq 3$. For this encoder, $m = 2$. Therefore, the minimum distance of the code is $d_3^* = d^* = 5$. This code can correct $(d^*-1)/2 = 2$ random errors that occur in one blocklength, $n = (m+1)n_0 = 6$.

Definition 6.11 The **Free Distance** of a convolutional code is given by

$$d_{free} = \max_l [d_l] . \tag{6.17}$$

It follows that $d_{m+1} \leq d_{m+2} \leq \ldots \leq d_{free}$.

The term d_{free} was first coined by Massey in 1969 to denote a type of distance that was found to be an important parameter for the decoding techniques of convolutional codes. Since, d_{free} represents the minimum distance between arbitrarily long (possibly infinite) encoded sequences, d_{free} is also denoted by d_∞ in literature. The parameter d_{free} can be directly calculated from the trellis diagram. The free distance d_{free} is the minimum weight of a path that deviates from the all-zero path and later merges back into the all-zero path at some point further down the trellis as depicted in Fig. 6.13. Searching for a code with a large minimum distance and a large free distance is a tedious process, and is often done using a computer. Clever techniques have been designed to reduce the effort by avoiding exhaustive searches. Most of the good convolutional codes known today have been discovered by computer searches.

Definition 6.12 The **Free Length** n_{free} of a convolutional code is the length of the non-zero segment of a smallest weight convolutional codeword of non-zero weight. Thus, $d_l = d_{free}$ if $l = n_{free}$, and $d_l < d_{free}$ if $l < n_{free}$. In literature, n_{free} is also denoted by n_∞.

Scanned by CamScanner

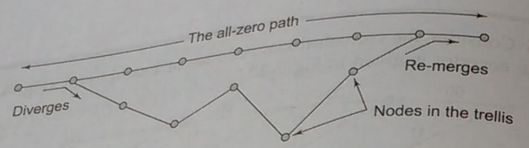


Fig. 6.13 The free distance d_{free} is the minimum weight of the path that diverges from the all-zero path and later merges back.

Example 6.7 Consider the convolutional encoder of Example 6.1 (Fig. 6.3). For this encoder, $d_{free} = 5$. There are usually more than one pair of paths that can be used to calculate d_{free}. The two paths that have been used to calculate d_{free} are shown in Fig. 6.14 by double lines. In this example, $d_{min} = d_{free}$.

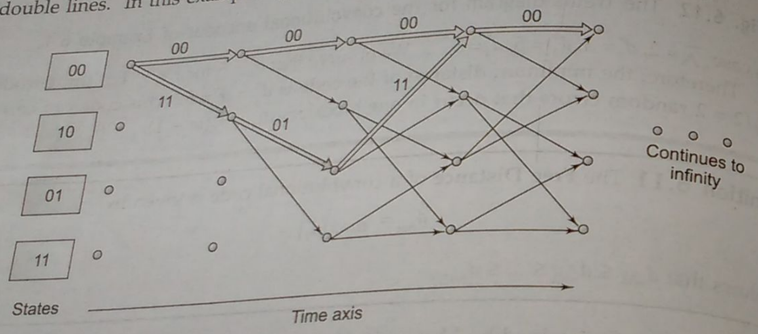


Fig. 6.14 The pair of paths that is used to calculate d_{free} in the trellis diagram.

The free length of the convolutional code is $n_{free} = 6$. In this example, the n_{free} is equal to the blocklength n of the code. In general it can be longer than the blocklength.

6.5 THE GENERATING FUNCTION

The performance of a convolutional code depends on its free distance, d_{free}. Since convolutional codes are a sub class of linear codes, the set of Hamming distances between coded sequences is the same as the set of distances of the coded sequences from the all-zero sequence. Therefore, we can consider the distance structure of the convolutional codes with respect to the all-zero sequence without loss in generality. In this section we shall study an elegant method of determining the free distance, d_{free}, of a convolutional code.

To find d_{free} we need the set of paths that diverge from the all-zero path and later merge back at some later time. The brute force (and time consuming, not to mention, exasperating) method would be to determine the instances of all possible paths from the trellis diagram. Another way to find out the d_{free} of a convolutional code is use the concept of a **generating function**, whose expansion provides all the distance information directly. The generating function can be understood by the following example.

Example 6.8 Consider again the convolutional encoder of Example 6.1 (Fig. 6.3). The state diagram of the encoder is given in Fig. 6.4. We now construct a modified state diagram as shown in Fig. 6.15.

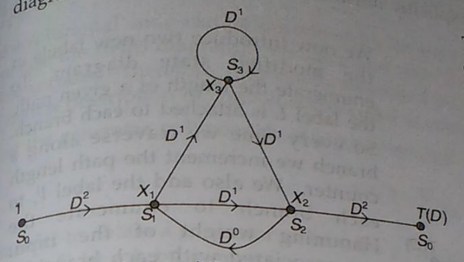


Fig. 6.15 The modified state diagram of the convolutional encoder shown in Fig. 6.3.

The branches of this modified state diagram are labeled by the branch gain D^i, $i = 0,1,2$, where the exponent of D denotes the Hamming weight of the branch. Note that the self-loop at S_0 has been neglected as it does not contribute to the distance property of the code. Circulating around this loop simply generates the all-zero sequence. Note that S_0 has been split into two states, initial and final. Any path that diverges from state S_0 and later merges back to S_0 can be thought of equivalently as traversing along the branches of this modified state diagram, starting from the initial S_0 and ending at the final S_0. Hence this modified state diagram encompasses all possible paths that diverge from and then later merge back to the all-zero path in the trellis diagram.

We can find the distance profile of the convolutional code using the state equations of this modified state diagram. These state equations are

$$\begin{aligned} X_1 &= D^2 + X_2 \\ X_2 &= DX_1 + DX_3 \\ X_3 &= DX_1 + DX_3 \\ T(D) &= D^2X_2 \end{aligned} \tag{6.18}$$

where X_i are dummy variables. Upon solving these equations simultaneously we obtain the generating function

$$\begin{aligned} T(D) &= \frac{D^5}{1-2D} \\ &= D^5 + 2D^6 + 4D^7 + \ldots + 2^k D^{k+5} + \ldots \\ &= \sum_{d=5}^{\infty} a_d D^d \end{aligned} \tag{6.19}$$

Scanned by CamScanner

Note that the expression for $T(D)$ can also be (easily) obtained by the Mason's gain formula, which is well-known to the students of digital signal processing.

Following conclusions can be drawn from the series expansion of the generating function:

(i) There are an infinite number of possible paths that diverge from the all-zero path and later merge back again (this is also intuitive).

(ii) There is only one path with Hamming distance 5, two paths with Hamming distance $k + 5$ from the all-zero path.

(iii) The free Hamming distance d_{free} for this code is 5. There is only one path corresponding to d_{free}. Example 6.7 explicitly illustrates the pair of paths that result in $d_{free} = 5$.

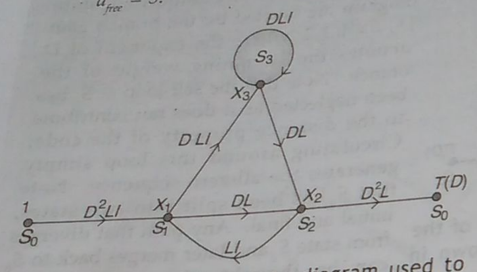


Fig. 6.16 The modified state diagram used to determine the augmented generating function.

We now introduce two new labels in the modified state diagram. To enumerate the length of a given path, the label L is attached to each branch. So every time we traverse along a branch we increment the path length counter. We also add the label I^i, to each branch to enumerate the Hamming weight of the input bits associated with each branch of the modified state diagram (see Fig. 6.16).

The state equations in this case would be

$$X_1 = D^2LI + LIX_2$$
$$X_2 = DLX_1 + DLX_3$$
$$X_3 = DLIX_1 + DLIX_3$$

and the **Augmented Generating Function** is

$$T(D) = D^2LX_2. \tag{6.20}$$

On solving these simultaneous equations we obtain

$$T(D,L,I) = \frac{D^5L^3I}{1-DL(L+1)I} \tag{6.21}$$
$$= D^5L^3I + D^6L^4(L+1)I^2 + \ldots + D^{k+5}L^{3+k}(L+1)^k I^{k+1} + \ldots$$

Further conclusions from the series expansion of the augmented generating function are:

(i) The path with the minimum Hamming distance of 5 has length equal to 3.

(ii) The input sequence corresponding to this path has weight equal to 1.

(iii) There are two paths with Hamming distance equal to 6 from the all-zero path. Of

these, one has a path length of 4 and the other a path length of 5 (observe the power of L in the second term in the summation). Both these paths have an input sequence weight of 2.

In the next section, we will study the matrix description of convolutional codes. The matrix representation of convolutional codes, as we shall shortly discover, is a bit more complicated than the matrix description of linear block codes.

6.6 MATRIX DESCRIPTION OF CONVOLUTIONAL CODES

A convolutional code can be described by an infinite number of infinitely long codewords (visualise the trellis diagram). We have also seen that convolutional codes belong to the class of linear codes. Therefore, they can be described by an infinite generator matrix. As expected, the matrix description of convolutional codes is messier than that of the linear block codes.

Let the generator polynomials of a convolutional code be represented by

$$g_{ij}(D) = \sum_l g_{ijl} D^l. \tag{6.22}$$

In order to obtain a generator matrix, the g_{ijl} coefficients are arranged in a matrix format. For each l, let G_l be a k_0 by n_0 matrix.

$$G_l = [g_{ijl}]. \tag{6.23}$$

Then the generator matrix for the convolutional code that has been truncated to a block code of blocklength n is

$$G^{(n)} = \begin{bmatrix} G_0 & G_1 & G_2 & \ldots & G_m \\ 0 & G_0 & G_0 & \ldots & G_{m-1} \\ 0 & 0 & G_0 & \ldots & G_{m-2} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \ldots & G_0 \end{bmatrix}, \tag{6.24}$$

where 0 is a k_0 by n_0 matrix of zeros and m is the length of the shift register used to generate the code. The generator matrix for the convolutional code is given by

$$G = \begin{bmatrix} G_0 & G_1 & G_2 & \ldots & G_m & 0 & 0 & 0 & 0 & \ldots \\ 0 & G_0 & G_1 & \ldots & G_{m-1} & G_m & 0 & 0 & 0 & \ldots \\ 0 & 0 & G_0 & \ldots & G_{m-2} & G_{m-1} & G_m & 0 & 0 & \ldots \\ \vdots & \vdots & & & & & & & & \vdots \end{bmatrix}. \tag{6.25}$$

The matrix extends infinitely far down and to the right. For a systematic convolutional code, the generator matrix can be written as

Scanned by CamScanner