International Journal of Engineering Science Invention Research & Development; Vol. 12, Issue 8, February 2026
www.ijesird.com, E-ISSN: 2349-6185

MATHEMATICAL CONSTRUCTION OF ENCODING & DECODING THE CODES

Prabhat Kumar
Department of Applied Sciences & Humanities, Mangalmay Institute of Management & Technology Department of Mathematics, U.P. India

Abstract: **This paper presents a general method of constructing error correcting codes known as a BCH codes. A binary group code with n places, k of which are information places is called an (n, k) code. An explicit method of constructing t-error correcting (n, k) codes is given for $n = 2^m - 1$ and $k = 2^m - 1 - R(m, t) \geq 2^m - 1 - mt$ where R(m, t) is a function of m and t which cannot exceed mt. An example is worked out to illustrate the method of construction, and this paper also presents a encoding and error correction procedures for the BCH codes by W. W. Peterson and decoding the BCH codes by Elwyn R. Berlekamp.**

I. INTRODUCTION

Bose and Ray-Chaudhuri and Hocquenghem independently discovered what have become known as BCH codes around 1960. Many of the most important block codes for random error-correction fall into this family of codes. BCH codes have been the most extensively studied block codes and are very important in practice for many reasons. First of all, there is an ample selection of block lengths and code rates. Secondly, for short-to-moderate block lengths and small code alphabets, BCH codes are essentially the most powerful linear codes. Finally, there are elegant general-purpose decoding algorithms available for these codes. There are binary and multilevel BCH codes, although only the binary case will be investigated in this report. A BCH code, like other linear codes, can be defined in terms of its parity-check matrix H. And, because it is a cyclic code, it can also be defined by its generator polynomial g(t), and hence, its generator matrix G. Cyclic codes form the major class of error-correcting block codes. This is because of their well-understood mathematical structure (which is polynomial based) and because they can be conveniently implemented using shift registers. The term cyclic arises from the fact that a cyclic shift of a code word of a cyclic code generates another code word. Here, only the generator polynomial approach will be given as there are extensive tables of code generators available in many text books and it is easy to look them up. As a convention, in coding theory a block code is represented with two or three numbers, n, k, and optionally t. Here n is the length of the code word, k is the length of the data sequence and t is the number of errors that this particular code can correct. As an example a (7,4,1) binary BCH code has a code word length of 7 bits, data sequence length of 4 bits and it can correct one error in the code word. Likewise a (15,5,3) binary BCH code can encode a 5-bit long data sequence into a 15-bit long code word and can correct any three errors in the code word. The code rate or coding efficiency of a code is defined as the ratio of length of the data sequence to the length of the code word. For a block code R = k/n and for a fixed n, this will decrease as more check symbols are added to achieve greater error correction capability.

Here the choice of n, k and t is not arbitrary. The relationship among the three can be expressed as follows:

$n = 2^m - 1$

$n - k \leq mt$

where $m \geq 3$ and $t < 2^{m-1}$.

The parameter m has important algebraic significance since it denotes the particular Galois field associated with a particular code. A field with a finite number of elements, q, is called a finite field or Galois field and is denoted GF(q). In general, finite fields exist only when q is prime, or when $q = p^m$ where p is prime and m is an integer. The prime field GF(q), (q>l and q prime), will have elements 0,1,2,3,...,q–1, in other words, the set of all integers modulo q, and the operations will be addition and multiplication modulo q.

The field GF(2^m), m>l, is called an extension field of GF(2) and these fields are particularly relevant to cyclic codes and especially to Reed-Solomon codes. To generate a field GF(2^m) the elements 0, 1, which are the elements of GF(2), are extended using a primitive element α. If α is an element of GF(2^m)then the closure postulate under multiplication means that $\alpha.\alpha = \alpha^2$ and$\alpha.\alpha^2 = \alpha^3$ etc., are also elements of GF(2^m).
Therefore,

$$GF(2^m) = \{0, 1, \alpha, \alpha^2, \ldots\}.$$

By definition, GF(2^m) must be finite and this is achieved by associating it with a polynomial p(x) of degree m which is primitive over GF(2). Essentially, p(x) limits the field to a finite number of elements and defines the addition and multiplication rules which must be satisfied by these elements. This can be accomplished if for elementα,

$$p(\alpha) = 0.$$

Lists of primitive polynomials over various Galois fields can be found in many text books. Here, since we shall present our results based on GF(2^4), addition and multiplication rules for this field, GF(16), are given in Tables 3.1.and 3.2., respectively.
The defining rules from a primitive polynomial are obtained as follows. Consider GF(2^4) and assume the primitive polynomial chosen is

$$p(x) = 1 + x + x^4.$$

since $p(\alpha) = 0$ the following identity holds

$$1 + \alpha + \alpha^4 = 0.$$

The above equation can be used to define the arithmetic rules for this particular field. It should be noted that for GF(2^m) each element is its own additive inverse, so it can be stated that $\alpha^i + \alpha^i = 0$. In other words addition and subtraction are identical over GF(2^m).

$\alpha^4 = \alpha + 1$

Now the higher order terms can be defined as follows :

$$\alpha^5 = \alpha.\ \alpha^4 = \alpha(\alpha+1) = \alpha^2 + \alpha$$
$$\alpha^6 = \alpha.\alpha^5 = \alpha(\alpha^2+\alpha) = \alpha^3 + \alpha^2$$
$$\alpha^7 = \alpha.\ \alpha^6 = \alpha(\alpha^3+\alpha^2) = \alpha^4 + \alpha^3 = \alpha^3 + \alpha + 1$$
$$\alpha^8 = \alpha.\alpha^7 = \alpha(\alpha^3+\alpha+1) = \alpha^4 + \alpha^2 + \alpha = \alpha^2 + 1$$
$$\alpha^9 = \alpha.\alpha^8 = \alpha(\alpha^2+1) = \alpha^3 + \alpha$$
$$\alpha^{10} = \alpha.\alpha^9 = \alpha(\alpha^3+\alpha) = \alpha^4 + \alpha^2 = \alpha^2 + \alpha + 1$$
$$\alpha^{11} = \alpha.\alpha^{10} = \alpha(\alpha^2+\alpha+1) = \alpha^3 + \alpha^2 + \alpha$$
$$\alpha^{12} = \alpha.\alpha^{11} = \alpha(\alpha^3+\alpha^2+1) = \alpha^4 + \alpha^3 + \alpha^2 = \alpha^3 + \alpha^2 + \alpha + 1$$
$$\alpha^{13} = \alpha.\alpha^{12} = \alpha(\alpha^3+\alpha^2+\alpha+1) = \alpha^4 + \alpha^3 + \alpha^2 + \alpha = \alpha^3 + \alpha^2 + 1$$
$$\alpha^{14} = \alpha.\alpha^{13} = \alpha(\alpha^3+\alpha^2+1) = \alpha^4 + \alpha^3 + \alpha = \alpha^3 + 1$$

I continued the same way, it can be seen that $\alpha^{15} = \alpha.\alpha^{14} = 1$
Which leads to the conclusion that the element of GF(2^4) are restricted to

Table 3.1 Addition rules for GF(2^4) generated by p(x) = 1 + x + x^4.

$+$	0	1	α	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
0	0	1	α	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
1	1	0	α^4	α^8	α^{14}	α	α^{10}	α^{13}	α^9	α^2	α^7	α^5	α^{12}	α^{11}	α^6	α^3
α	α	α^4	0	α^5	α^9	1	α^2	α^{11}	α^{14}	α^{10}	α^3	α^8	α^6	α^{13}	α^{12}	α^7
α^2	α^2	α^8	α^5	0	α^6	α^{10}	α	α^3	α^{12}	1	α^{11}	α^4	α^9	α^7	α^{14}	α^{13}
α^3	α^3	α^{14}	α^9	α^6	0	α^7	α^{11}	α^2	α^4	α^{13}	α	α^{12}	α^5	α^{10}	α^8	1
α^4	α^4	α	1	α^{10}	α^7	0	α^8	α^{12}	α^3	α^5	α^{14}	α^2	α^{13}	α^6	α^{11}	α^9
α^5	α^5	α^{10}	α^2	α	α^{11}	α^8	0	α^9	α^{13}	α^4	α^6	1	α^3	α^{14}	α^7	α^{12}
α^6	α^6	α^{13}	α^{11}	α^3	α^2	α^{12}	α^9	0	α^{10}	α^{14}	α^5	α^7	α	α^4	1	α^8
α^7	α^7	α^9	α^{14}	α^{12}	α^4	α^3	α^{13}	α^{10}	0	α^{11}	1	α^6	α^8	α^2	α^5	α
α^8	α^8	α^2	α^{10}	1	α^{13}	α^5	α^4	α^{14}	α^{11}	0	α^{12}	α	α^7	α^9	α^3	α^6
α^9	α^9	α^7	α^3	α^{11}	α	α^{14}	α^6	α^5	1	α^{12}	0	α^{13}	α^2	α^8	α^{10}	α^4
α^{10}	α^{10}	α^5	α^8	α^4	α^{12}	α^2	1	α^7	α^6	α	α^{13}	0	α^{14}	α^3	α^9	α^{11}
α^{11}	α^{11}	α^{12}	α^6	α^9	α^5	α^{13}	α^3	α	α^8	α^7	α^2	α^{14}	0	1	α^4	α^{10}
α^{12}	α^{12}	α^{11}	α^{13}	α^7	α^{10}	α^6	α^{14}	α^4	α^2	α^9	α^8	α^3	1	0	α	α^5
α^{13}	α^{13}	α^6	α^{12}	α^{14}	α^8	α^{11}	α^7	1	α^5	α^3	α^{10}	α^9	α^4	α	0	α^2
α^{14}	α^{14}	α^3	α^7	α^{13}	1	α^9	α^{12}	α^8	α	α^6	α^4	α^{11}	α^{10}	α^5	α^2	0

Table 3.2 Multiplication rules for GF(2^4) generated by p(x) = 1 + x + x^4.

$\times$	0	1	α	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	α	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
α	0	α	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1
α^2	0	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α
α^3	0	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α	α^2
α^4	0	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α	α^2	α^3
α^5	0	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α	α^2	α^3	α^4
α^6	0	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α	α^2	α^3	α^4	α^5
α^7	0	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α	α^2	α^3	α^4	α^5	α^6
α^8	0	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α	α^2	α^3	α^4	α^5	α^6	α^7
α^9	0	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α	α^2	α^3	α^4	α^5	α^6	α^7	α^8
α^{10}	0	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9
α^{11}	0	α^{11}	α^{12}	α^{13}	α^{14}	1	α	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}
α^{12}	0	α^{12}	α^{13}	α^{14}	1	α	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}
α^{13}	0	α^{13}	α^{14}	1	α	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}
α^{14}	0	α^{14}	1	α	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}

$GF(2^m) = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^m-2}\}$

It is also possible that a polynomial r(α) can be reduced modulo p(α) by saving the remainder after division by p(α), i.e.

$$r(\alpha) \bmod p(\alpha) = rem\left[\frac{r(\alpha)}{p(\alpha)}\right].$$

II. ENCODING

One of the advantages of BCH codes over other codes is their simplicity in encoding and decoding. Encoding of an information vector can be done in two ways. One can use either the generator polynomial approach or the generator matrix approach. Here, due to their simplicity both of the algorithms for the binary case will be presented In the generator polynomial approach, without a loss of generality, a systematic coding procedure will be explained. A code is said to be systematic if it carries the original information vector elements in its code words. Otherwise it is called a non-systematic code. For a systematic (n, k) code, a code word vector of the form

$v=\lfloor p_0, p_1, \ldots + p_{n-k-2}, p_{n-k-1}, i_0, i_1, \ldots, i_{k-2}, i_{k-1} \rfloor$ is required. The encoder wi ll then generate k message symbols in unaltered form, followed by n - k symbols. The correspond ing code polynomial is then given by

$$\text{v(x)} = \text{p}_0 + p_1 x + \ldots + p_{n-k-2} x^{n-k-2} + p_{n-k-1} x^{n-k-1} + i_0 x^{n-k} + i_1 x^{n-k+1} + \ldots + i_{k-1} x^{n-1}$$
$$= p(x) + x^{n-k} i(x)$$

where p(x) is a check or parity polynomial . Since we are dealing with binary numabers, this can be rearranged over GF(2) as: $\text{x}^{\text{n-k}}\, i(x) = p(x) + v(x)$

It is known that a binary code word polynomial v(x) in an (n, k) cyclic code can be derived as $\text{v}(\text{x}) = i(x) g(x)$ where i(x) is the message polynomial and g(x) is the generator polynomial . Then it is obvious that a codeword is divisible by g(x). If this statement is applied to the equation $\text{x}^{\text{n-k}}\, i(x) = v(x) + p(x)$

It can be shown that $\dfrac{\text{x}^{\text{n-k}}\, i(x)}{g(x)} = \text{a}(\text{x}) + \dfrac{p(x)}{g(x)}$

where $\text{a}(\text{x})$ is the quotient and p(x) is the remainder. The above equation fully defines the following encoding algorithem for a systematic cyclic code.

1. Multiply i(x) by $\text{x}^{\text{n-k}}$
2. Divide $\text{x}^{\text{n-k}}\, i(x)\; by\; g(x)$. The remainder p(x) gives the required check symbols.
3. Combine informatio n and check symbols.

As an example, let us use the (15, 7) binary BCH code to encode the message vector $\text{i} = [0, 1, 0, 1, 1, 0, 1]$

A (15, 7) BCH code is defined by its generator polynomial as;

$$\text{g(x)} = 1 + \text{x}^4 + \text{x}^6 + x^7 + x^8$$

In this case the parameters are $\text{n} = 15,\ \text{k} = 7\ ,\ \text{n - k} \le \text{mt} \Rightarrow \text{t} = \dfrac{15 - 7}{4} = 2,\ m = 4$

Now the message polynomial is given by $\text{i}(\text{x}) = x + x^3 + x^4 + x^6$

Following the above steps;

1. $\text{x}^8\, i(x) = x^9 + x^{11} + x^{12} + x^{14}$
2. Dividing this by g(x) gives the remainder or the check polynomial $\text{x}^7 + x^5 + x^4 + x^3 + x^2$

3. Combining both informatio n and check polynomial we have

$$v(x) = x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^9 + x^{11} + x^{12} + x^{14}$$

and the codeword is

$v = [0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1]$.

As can be seen, the message vector appears in the last seven digits of the codes word as a result of the systematic coding $v = [0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1]$.

Likewise it can be shown that the message vector $i = [1, 0, 0, 1, 1, 1, 0]$

can be encoded as $v = [0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0]$

In the generator matrix approach, we have a matrix G, and by premultipl ying it with th e message vectors i, we can dirrectly obtain the codeword.

The generator matrix G is a $k \times n$ matrix and it is constructe d from the cofficient s of the generator polynomial g(x) . Having the generator polynomial as

$$g(x) = g_0 + g_1 x + g_2 x^2 + \ldots + g_{n-k-1} x^{n-k-1} + g_{n-k} x^{n-k}$$

the generator matrix $G_{k \times n}$ due to the nature of cyclic code is

$$G_{k\times n} = \begin{bmatrix} g_{n-k} & g_{n-k-1} & . & . & . & . & g_3 & g_2 & g_1 & g_0 & 0 & 0 & 0 & 0 & 0 \\ 0 & g_{n-k} & g_{n-k-1} & . & . & . & . & g_3 & g_2 & g_1 & g_0 & 0 & 0 & 0 & 0 \\ 0 & 0 & g_{n-k} & g_{n-k-1} & . & . & . & . & g_3 & g_2 & g_1 & g_0 & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . & . & . & . & . & 0 & 0 & 0 \\ . & . & . & . & . & . & . & . & . & . & . & . & . & 0 & 0 \\ . & . & . & . & . & . & . & . & . & . & . & . & . & . & 0 \\ 0 & 0 & 0 & 0 & 0 & g_{n-k} & g_{n-k-1} & . & . & . & . & g_3 & g_2 & g_1 & g_0 \end{bmatrix}$$

In the previous example, the generator polynomial was chosen as

$g(x) = 1 + x^4 + x^6 + x^7 + x^8$ so this time the generator matrix wil l be

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

and if the above matrix is pre - multiplied by the message vector in the first example

$i = [0, 1, 0, 1, 1, 0, 1]$

we get $v = i.G$

$$= [0\,1\,1\,0\,0\,1\,0\,0\,0\,1\,0\,1\,1\,0\,1]$$

III. DECODING

Decoding of BCH codes is done in a manner that computes directly the locations and values of the individual errors, in contrast to storage-consuming table look-up methods associated with coset leaders and to exhaustive calculations involved in finding the maximum likelihood estimate of the transmitted code word.
Here the decoding of binary BCH codes will be presented. The decoding algorithm can be decomposed into four main steps as follows:
1. Calculation of syndromes,
2. Formation of the error locator polynomial based on syndrome values,
3. Calculation of the error locations,
4. Correcting the errors.
In the binary case, the fourth step consists only of the complementation of the bits in the error positions. Among the remaining steps, step 2 requires by far the most explanation and analysis, although in terms of concept and implementation, it is no more difficult than any of the other steps. Berlekamp and Massey made this step much easier to understand and carry out by the algorithm they developed.
Now steps 1, 2, and 3 of the binary BCH decoding procedure will be discussed below in further detail.
Step1. Calculation of Syndromes
Assume the received code word in polynomial form is

$$r(x) = r_0 + r_1 x + r_2 x^2 + \ldots + r_{n-2} x^{n-2} + r_{n-1} x^{n-1}$$

$$= v(x) + e(x)$$

$$\textit{where}\;\; v(x) = v_0 + v_1 x + v_2 x^2 + \ldots + v_{n-2} x^{n-2} + v_{n-1} x^{n-1}$$

$$\textit{and}\quad e(x) = e_0 + e_1 x + e_2 x^2 + \ldots + e_{n-2} x^{n-2} + e_{n-1} x^{n-1}$$

Here, e(x) is the error polynomial such that $e_i = 1$ for a received error in symbol r, the syndrome polynomial s(x) is generated by dividing r(x) by g(x), and since all code words are divisible by g(x) it can be written $\dfrac{r(x)}{g(x)} = a(x) + \dfrac{e(x)}{g(x)}$
The required syndrome s(x) is then the remainder e(x) resulting from the division e(x)/g(x) and, clearly, s(x) will be zero if e(x) is zero. Stating the above process in another way; r(x) = v(x) + e(x) and we require the syndrome of r(x). It is known that for every code word v(x), $v(\alpha^i) = 0$.
Therefore $r(\alpha^i) = v(\alpha^i) + e(\alpha^i) = e(\alpha^i) = s_i$
Where s_i is a syndrome symbol. Restating this result, the syndrome symbols can be found from
$s_i = r(\alpha^i)$, $i = 1, 2, \ldots, 2t$
Thus, each component of s_i is an element of $GF(2^m)$.
Step2. Formation of Error Locator Polynomial
In this step syndrome symbols, s_i's will be used to determine the coefficients of the error locator polynomial, the roots of which define the error locations. For the sake of clarity this procedure will be explained starting with an example.
Suppose that for a particular received polynomial, $e(x) = x^8 + x^{11}$
so that $e(\alpha^i) = (\alpha^i)^{11} + (\alpha^i)^8 = (\alpha^{11})^i + (\alpha^8)^i = X_1^i + X_2^i$.
Here, $X_1 = \alpha^{11}$ is an error locator, denoting an error in received symbol r_{11} and $X_2 = \alpha^8$ is an error locator

denoting an error in symbols r_8. Following the same notation, the kth error locator will then be of the form $X_k = \alpha^j$, α^j belongs to GF(2^m)
and denotes an error m symbol r_j, j = 0,l,...,n– l. Since here only the binary case is considered r_j will then be complemented to perform error correction. Error locators, X_k's are obtained from syndrome symbols s_i in an indirect way, $s_i = e(\alpha^i) = \sum_{k=1}^{i} X_k^i$, i = 1, 2, . . . , 2t.

The usual approach to solve the non-linear equations above is to use an error locator polynomial, σ(x). The significance of σ(x) is that it transforms the problem into a set of linear equations, and that its solutions turn out to be the error locators, X_k. There are many different techniques for finding σ(x), and here only Peterson's direct method will be explained. Other than this method, Euclid's algorithm for polynomials or Berelekamp's iterative algorithm can also be used to find the error locator polynomial σ(x).
Proceeding with Peterson's direct method, one form for σ (x) is

$$\sigma(x) = x^t + \sigma_1 x^{t-1} + \sigma_2 x^{t-2} + \ldots + \sigma_{t-1} x + \sigma_t$$

and since the roots are the error locators $X_1, X_2, \ldots, X_{t-1}, X_t$ it can be written

$$X_k^t + \sigma_1 X_k^{t-1} + \sigma_2 X_k^{t-2} + \ldots + \sigma_{t-1} X_k^1 + \sigma_t = 0,\ k = 1,2,3,\ldots,t$$

multiplyin g both side by X_k^j,

$$X_k^{t+j} + \sigma_1 X_k^{t+j-1} + \sigma_2 X_k^{t+j-2} + \ldots + \sigma_{t-1} X_k^{j+1} + \sigma_t X_k^j = 0,$$

and then expanding the above expression

$$X_1^{t+j} + \sigma_1 X_1^{t+j-1} + \sigma_2 X_1^{t+j-2} + \ldots + \sigma_{t-1} X_1^{j+1} + \sigma_t X_1^j = 0$$
$$X_2^{t+j} + \sigma_1 X_2^{t+j-1} + \sigma_2 X_2^{t+j-2} + \ldots + \sigma_{t-1} X_2^{j+1} + \sigma_t X_2^j = 0$$
$$X_3^{t+j} + \sigma_1 X_3^{t+j-1} + \sigma_2 X_3^{t+j-2} + \ldots + \sigma_{t-1} X_3^{j+1} + \sigma_t X_3^j = 0$$
$$\vdots$$
$$X_t^{t+j} + \sigma_1 X_t^{t+j-1} + \sigma_2 X_t^{t+j-2} + \ldots + \sigma_{t-1} X_{tk}^{j+1} + \sigma_t X_t^j = 0$$

$$\sum_{k=1}^{t} X_k^{t+j} + \sigma_1 \sum_{k=1}^{t} X_k^{t+j-1} + \sigma_2 \sum_{k=1}^{t} X_k^{t+j-2} + \ldots + \sigma_{t-1} \sum_{k=1}^{t} X_k^{j+1} + \sigma_t \sum_{k=1}^{t} X_k^j = 0$$

$$\Rightarrow s_{t+j} + \sigma_1 s_{t+j-1} + \sigma_2 s_{t+j-2} + \ldots + \sigma_{t-1} s_{1+j} + \sigma_t s_j = 0,\ j = 1,2,\ldots,t$$

Equation represents a set of t linear equations that can be solved for the coefficients σ_i, i = 1, 2, 3... r, hence resulting m the polynomial σ(x).
Up to now all the statements hold for any general case. In addition to these, for binary codes the following identity holds, $s_{2k} = s_k^2$, k = 1, 2, 3, . . .
Step3. Calculation of Error Locations: Up to this point coefficients of the error locator polynomial have been found from syndrome symbols. In order to determine the error locators X_k, k = l, 2,...,t, it is necessary to calculate the roots of the error locator polynomial σ(X).

Since there are a finite number of elements, the obvious approach to find the roots of the polynomial is to substitute n values of X_k given by the equation $\{X_k = \alpha^j\}$
$X_k^{t+j} + \sigma_1 X_k^{t+j-1} + \sigma_2 X_k^{t+j-2} + \ldots + \sigma_{t-1} X_k^{j+1} + \sigma_t X_k^j = 0,$ and note those values which satisfy the equality.

Table 3.3 Coefficients of the error locator polynomial for 1,2, and 3 error-correcting binary BCH codes

t	σ_i
1	$\sigma_1 = s_1$
2	$\sigma_1 = s_1$ $\sigma_2 = \frac{s_1^3 + s_3}{s_1}$
3	$\sigma_1 = s_1$ $\sigma_2 = \frac{s_1^2 s_3 + s_5}{s_1^3 + s_3}$ $\sigma_3 = (s_1^3 + s_3) + s_1\sigma_2$

Now $\frac{\sigma(\mathrm{x})}{\mathrm{x}^{\mathrm{i}}} = 1 + \sigma_1(\mathrm{x})^{-1} + \ldots + \sigma_i(\mathrm{x})^{-i}$ over GF(2^{m}), equating $\sigma(\mathrm{x})$ to zero is equivalent to finding the roots of $\sigma_1 x^{-1} + \sigma_2 x^{-2} + \ldots + \sigma_{\mathrm{i}} x^{-i} = 1$ *and* $\sum_{i=1}^{t} \sigma_{\mathrm{i}} x^{-i} = 1$

The roots of $\sigma(\mathrm{x})$ are the error locators and they are of the form α^{j}, $\mathrm{j} = 0, 1, 2, \ldots, \mathrm{n} - 1$. Ine other words an error in symbols $\mathrm{r}_{\mathrm{n-1}}$ will correspond to an arror locator or root of the form $\alpha^{\mathrm{n-1}}$.

$$\sum_{i=1}^{t} \sigma_{\mathrm{i}} \alpha^{i\,j} = 1,\ j = 1, 2, \ldots, n$$

If the above equation hold for the i = j, then the symbol $\mathrm{r}_{\mathrm{n-j}}$ is to be performed.

Now , for the sake of clarificat ion of the decoding algorithem , we shall carry out one of the previous examples.

Consider t he (15, 7) BCH code example , where the message vector $\mathrm{i} = [1, 0, 0, 1, 1, 1, 0]$ and the encoded codeword vector wa s $\mathrm{v} = [0,1, 0,0 ,0, 0, 0,1, 1, 0, 0, 1, 1, 1, 0]$.

The above codeword vector correspond to a codeword polynomial of the form below

$\mathrm{v}(\mathrm{x}) = x^{13} + x^{12} + x^{11} + x^{8} + x^{7} + x$

Now, assume that thre e occurred two errors in the processed codeword vector r in the position 0 and 12

That is the purpose $\mathrm{r} = [1,1,0,0,0,0,0,1,1,0,0,1,0,1,0]$

Then the processed codeword polynomial is

$\mathrm{r(x)} = \mathrm{x}^{13} + x^{11} + x^{8} + x^{7} + x = 1$

If we calculate the syndrome values for the odd values

$s_1 = r(\alpha) = \alpha^{13} + \alpha^{11} + \alpha^{8} + \alpha^{7} + \alpha + 1$

$s_3 = r(\alpha^3) = \alpha^{39} + \alpha^{33} + \alpha^{24} + \alpha^{21} + \alpha^{3} + 1$

simplifyin g the above equation

$s_1 = \alpha^3 + \alpha^2 + 1 + \alpha^3 + \alpha^2 + \alpha + \alpha^2 + 1 + \alpha^3 + \alpha + 1 + \alpha + 1$

$\quad = \alpha^3 + \alpha^2 + \alpha$

$\quad = \alpha^{11}$

$s_3 = \alpha^3 + \alpha^2 + 1 = \alpha^{13}$

Now we are ready to form the error locator polynomial using the above syndrome values. Using the results given in the tables 3.3. for the error locator polynomial coefficien ts we have,

$\sigma_1 = s_1 = \alpha^{11}$

$$\sigma_2 = \frac{s_1^3 + s_3}{s_1} = \frac{\alpha^{33} + \alpha^{13}}{\alpha^{11}} = \frac{\alpha^{3} + \alpha^{13}}{\alpha^{11}} = \frac{\alpha^3 + \alpha^3 + \alpha^2 + 1}{\alpha^{11}} = \frac{\alpha^2 + 1}{\alpha^{11}} = \frac{\alpha^8}{\alpha^{11}} = \alpha^{-3} = \alpha^{12}$$

Consequent ly the error locator polynomial ,

$\sigma(\mathrm{x}) = x^2 + \sigma_1 x + \sigma_2 = x^2 + \alpha^{11} x + \alpha^{12}$

Since we know the error locator polynomial now, we can find the location of the errors by finding the roots of this polynomial. This task can be accomplished by applying equation $\sum_{i=1}^{t} \sigma_i \alpha^{ij} = 1,\ j = 1, 2, \ldots, n$ to the above polynomial and noting the values satisfying that equation. That is we are looking for they values which satisfy the equation below;

$\sigma_1 \alpha^j + \sigma_2 \alpha^{2j} = 1,\ \ j = 1, 2, \ldots, 14, 15$

Tracing the above equation for all those j values, and simplifying the results using, Table 3.1, Table 3.2 and Table 3.3., we have:

$j = 1 \Rightarrow \sigma_1 \alpha + \sigma_2 \alpha^2 = \alpha^{12} + \alpha^{14} = \alpha^5$

$j = 2 \Rightarrow \sigma_1 \alpha^2 + \sigma_2 \alpha^4 = \alpha^{13} + \alpha^{16} = \alpha^{12}$

$\vdots$

$j = 14 \Rightarrow \sigma_1 \alpha^{14} + \sigma_2 \alpha^{28} = \alpha^{25} + \alpha^{40} = 0$

$j = 15 \Rightarrow \sigma_1 \alpha^{15} + \sigma_2 \alpha^{26} = \alpha^{26} + \alpha^{42} = 1$

Note that the equation $\sum_{i=1}^{t} \sigma_i \alpha^{ij} = 1,\ j = 1, 2, \ldots, n$ is satisfied for two values of j; for j = 3 and for j = 15.

Therefore the elements $r_{15-3} = r_{12}$ and $r_{15-15} = r_0$ are to be corrected. In fact if we compare the original codeword and the noisy codeword, we see that those are the bits which were altered during the process.

$$\begin{array}{l} v = [\,0\ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 1\ 0\,] \\ \qquad \updownarrow \qquad\qquad\qquad\qquad\qquad\quad \updownarrow \\ r = [\,1\ \ 1\ 0\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 1\ 0\,]. \end{array}$$

REFERENCES:

1) D. Slepian, “A class of binary signalling alphabets,” Bell System Technical Journal, pp. 122-127; Vol.5 January 1956.
2) R. C. Bose and D. K. Ray-Chaudhuri, “On A Class of Error-Correcting Binary Group Codes,” Information and Control, Vol.3, pp. 68-79; 1959.
3) W. W. Peterson, “Encoding and Error-Correction procedures for the Bose-Chaudhuri Codes,” Ire Trans on Information Theory, Vol.it-6, pp.459-470; September, 1960.
4) R. T. Chien, “Cyclic decoding procedures for Bose-Chaudhuri Hocquenghem codes,” IEEE Trans. on Information Theory, vol. IT-10, pp. 357-362, October 1964.
5) E. Berlekamp, “On decoding binary Bose-Chaudhuri Hocquenghem codes,” IEEE Trans. on Information Theory, vol. IT-11, pp. 577-579, October 1965.
6) Mandelbaum David and N. J Paramus, “A method of coding for multiple errors,” IEEE Trans. Inform. Theory, vol IT-14, pp. 518-521, May 1968.
7) C. R. P. Hartmann,” A note on the decoding of double error correcting BCH codes of primitive length,” IEEE Trans. Syst. Inform. Theory, November 1971, pp. 426.
8) H. Robert, Morelos-Zaragoza, “On Primitive BCH codes with unequal error protection capabilities, “IEEE Trans. Inform, Theory, vol 41, No. 3, May 1995.
9) J. Gross Alan, “Augmented Bose-Chaudhuri codes which correct single bursts of errors,” IEEE Transaction, May 28, 1962.
10) J. H. van Lint,“ On the non-existence of perfect 2- and 3- Hamming –error-correcting codes over GF (q),” Information and control, 16(1970), pp. 396-401.
11) J. H. van Lint, “On the non-existence of certain perfect codes,” Computers in numbers theory, proc. Science Research Council Atlas Symposium (Oxford, 1969), 1971.
12) Kanemasu Melissa,” Perfect Codes,” MIT Undergraduate Journal of Mathematics, “volume 11, issue 2, pp. 205-223, 1998.
13) B. Masnick and J. Wolf,” On linear unequal error protection codes,” IEEE Trans. Inform. Theory, vol. IT-13, no. 4, pp. 600-607, July 1967.
14) W. J van Gils, “Linear unequal error protection codes from shorter codes,” IEEE Trans. Inform. Theory IT-29 (6), November, 1983.
15) Lin Shu, Robert H. Morelos-Zaragoza,” On Primitive BCH codes with unequal error protection capabilities,” IEEE Transactions on Information Theory, vol. 41, No. 3, May 1995.