SOME ASPECTS OF BCH DECODING

BY

SAMY EL-GUEBALY

Submitted to the
School of Graduate
Studies in partial
fulfillment of the
requirements for
the degree of
MASTER OF APPLIED SCIENCE

November 1978

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Professor S.G.S. Shiva for his help in the realization of this thesis.

-ABSTRACT-

The decoding of binary ($n=2^m-1,k,t$) Bose-Chaudhuri-Hocquenghem (BCH) codes involves two basic steps:

(i) Finding the error-locator polynomial $\psi(x)$.

(ii) Finding the roots of $\psi(x)$.

With regard to (i), we show that the Peterson matrix technique is faster than the Berlekamp iterative technique for small t.

Regarding (ii), the idea is essentially to compute $\psi(\alpha^i)$, and see for what values of i, $\psi(\alpha^i)=0$, where α is the primitive element of $GF(2^m)$. In this case, we draw attention to certain practically useful points in the case of quadratics, cubics, and quartics. Specifically showing that:

a) The roots of the quadratic x^2+x+k_2 over $GF(2^m)$ can be found by solving a set of simple simultaneous equations.

b) If we use a table-look-up approach, in the case of the cubic, x^3+x+k_3, then we have, at most, $\frac{n}{6}$ values of k_3 to consider; where n is the length of the code.

c) The quartic $x^4+k_2x^2+k_3x+k_4$, over $GF(2^m)$, does not have some or all of its roots in $GF(2^m)$ if $Tr\left(\frac{k_2^3}{k_3^2}\right) \neq Tr(1)$, where Tr indicates the trace.

TABLE OF CONTENTS

	Page
Chapter One Introduction	1
Chapter Two Review of concepts relevant to this thesis	8
Chapter Three Comparison between the Peterson matrix technique and the Berlekamp iterative technique	27
Chapter Four Practical results to determine the roots of a polynomial	50
Chapter Five Concluding remarks	99
List of Symbols	101

CHAPTER ONE

Introduction

Data communication consists of the transmission of symbols taken from some finite alphabet and sent through some communications channel. Data storage, also, consists of the storage of symbols taken from some finite alphabet in a storage medium. In both cases, imperfections in the communications channel, or storage medium (whether it be cables or wires, as in a telephone system; the vacuum of space, as in a satellite communications system; or magnetic tapes) will result in the finite probability of error that a given transmitted, or stored, symbol will be incorrectly received, or read.

Since it is very difficult to achieve a high degree of reliability by only improving the communications facilities or storage media, the field of Error-Correcting Codes has been introduced in order to combat the various types of channel noise disturbances.

Classically, coding problems have been approached by means of block codes and, in computer storage systems, nearly all types of codes used are block codes. In block codes, constraints between channel input symbols exist within blocks of length n, but adjacent blocks are independent. As a

result, block codes are defined by an encoder and a decoder. That is, the encoder will take each k-digit message and, according to certain rules, will encode it into an n-digit codeword. The decoder will perform the inverse operation.

The other form of coding, called tree coding, operates by not breaking up the information sequence into independent blocks. Convolutional codes are the most important subset of the class of tree codes. In these codes, the encoder breaks up the input sequences into k-symbol blocks, where k is usually a small integer. Then, on the basis of this k-tuple and the preceding information symbols, the encoder emits an n-symbol section of the code sequence.

A block diagram of a computer storage system is shown in Fig. 1.1. The coding problem is to design the encoder and decoder units so the symbols coming out of the decoder match those entered into the encoder, even if errors occur within the storage units.

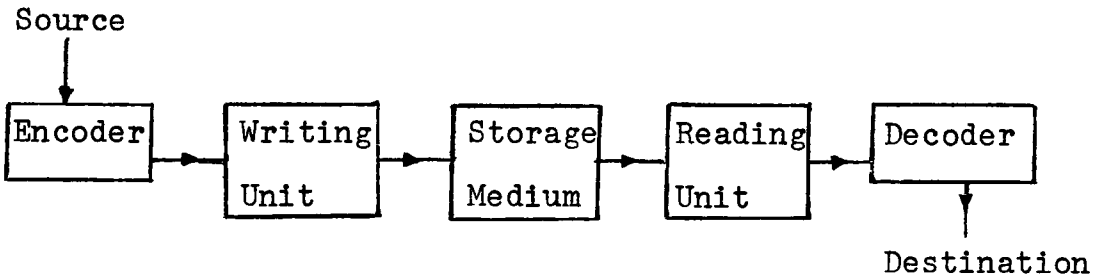


Fig. 1.1.

Practical error-correcting codes cannot compensate for every conceivable error pattern, but they can be designed to correct the most likely types of errors.

In early computer systems, parity checks were the main coding technique used to help improve data reliability. In their simplest form, such checks involve adding to each row or column of data bits one extra bit to indicate whether the sum of the data bits was even or odd. Unfortunately, such parity checks only detect errors. To correct a parity error, some additional time-consuming action must be taken to recapture the original data, using the principle of minimum distance.

The minimum distance of a code, using the binary system as an example, is the least number of bits that must change in a codeword so that another valid codeword of the code will result. Supposing that the minimum distance between any words in a code is three. If a single bit were to change, then the resulting incorrect codeword would be detected since it does not correspond to any of the codewords in the legitimate code.

Also, this incorrect codeword, when compared against all the legitimate codewords, is closer to one of them than to any other codeword. That is, it has a distance of one from one of the codewords, but a minimum distance of two to any other of the codewords. Thus, with the assumption of only a single error, it can be determined which of the codewords was actually transmitted, i.e. the codeword for which the distance was one. The improper word can then be corrected onto the actual codeword. Of course, if two bits should change, it would result in an incorrect codeword which could be detected, but not corrected.

In general, a code with a minimum distance d_{min} has an error-correcting capability of:

$$t = \left[(d_{min}-1)/2\right]$$

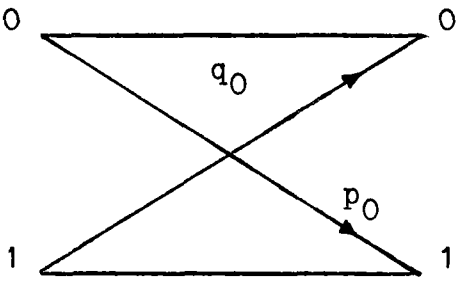
where $\left[(d_{min}-1)/2\right]$ denotes the highest integer no greater than $(d_{min}-1)/2$. A code of error-correcting capability t is generally called a t-error-correcting code.

In the case of error-detection, the decoder can detect all error patterns of $d_{min}-1$, or fewer, errors. This is so since no error pattern of $d_{min}-1$, or fewer, errors will alter the transmitted codeword into another codeword. To illustrate the technique used in error-correction, a decoding scheme is presented in the following discussion.

Supposing that: $c = (c_0, c_1, c_2, \ldots, c_{n-1})$ is the transmitted codeword, and $r = (r_0, r_1, r_2, \ldots, r_{n-1})$ is the sequence received at the channel output. The received sequence might differ from the transmitted codeword c because of channel noise disturbances.

If all codewords have an equal likelihood of being transmitted, then upon receiving the sequence r, the decoder computes the conditional probability $p(r/c_\ell)$ for all 2^k codewords. The codeword c_t is identified as the transmitted word if the conditional probability $p(r/c_t)$ is the largest. This decoding scheme is known as Maximum Likelihood Decoding. For a binary symmetric channel with crossover probability P_0, the conditional probability $p(r/c_\ell)$ can be expressed as follows:

$$p(r/c_\ell) = \prod_{i=0}^{n-1} p(r_i/c_{\ell i})$$



where $p(r_i/c_{\ell i})=q_0$ for $r_i=c_{\ell i}$

and $p(r_i/c_{\ell i})=p_0$ for $r_i \neq c_{\ell i}$

Let d_ℓ be the number of places where the codeword c_ℓ and the received sequence r differ:

$$p(r/c_\ell) = q_0^{n-d_\ell} p_0^{d_\ell}$$

Since $q_0 > p_0$, $p(r/c_\ell)$ decreases monotonically with increasing d_ℓ. Therefore, to find the codeword c_t, such that $p(r/c_t)$ is maximized, is equivalent to finding the codeword c_t that differs from the received sequence r in the fewest number of places.

Coding theory was initiated in 1948 by C.E. Shannon when he was able to prove that, given a channel with a capacity C, there exist codes of rate R, less than C, which--with Maximized Likelihood Decoding--have an arbitrarily small probability of erroneous decoding $p(\varepsilon)$. More specifically, for any given rate $R<C$ and length n, there exists a block code such that the probability of an erroneous decoding equals $p(\varepsilon) \leq e^{-nE(R)}$

where E(R) is a positive function of R for R<C and is specified by channel transition probabilities. Therefore, the probability of a decoding error can be made as small as we wish by increasing the code length n and keeping the rate R less than the channel capacity C. Shannon's theorem shows only the existence of codes which give an arbitrarily small probability of decoding error, but this does not indicate how these codes can be constructed.

The topic of this thesis is the study of some aspects of the Bose-Chaudhuri-Hocquenghem (BCH) codes. These codes were introduced approximately ten years after the Hamming single error-correcting codes were developed, and were originally double error-correcting codes. However, the generalization to t-error-correcting codes followed immediately, for all t-errors. BCH codes are not probabilistic. In other words, the crossover probability of the channel will not affect the error-correcting capability of the code.

For reasons of equipment flexibility, reliability, and low cost, most data communication, or storage systems, transmit or store sequences of signals in the form of binary symbols, and the binary system will be the only one discussed in this thesis.

The material to follow is organized into four main chapters:

Chapter Two: Review of concepts relevant to this thesis; and an introduction to BCH codes.

Chapter Three: Comparison between two decoding algorithms; the Peterson matrix and the Berlekamp iterative technique, from the point of view of speed.

Chapter Four: After a brief review of known techniques for determining the roots of a polynomial, practically useful results will be given in the case of quadratics, cubics, and quartics.

Chapter Five: Concluding remarks.

Some of the results of this thesis have been presented elsewhere (Ref. 19).

CHAPTER TWO

In Chapter Two, we will review the various algebraic structures, discuss the properties of finite fields, present the encoding process technique and introduce the subject of BCH codes.

2.1. Algebraic structures: [1,2,3,4,6]

Let a,b,c, represent the elements of a set S.

2.1.1. Group

Definition: A system with one operation and its inverse.

Properties: a) Additive group

a.1) $a+b \in S$ (Closure)

a.2) $(a+b)+c = a+(b+c)$ (Associative)

a.3) There is an element $\theta \in S$ such that $a+\theta=a$ (Additive Identity)

a.4) There is an $a^{-1} \in S$ such that $a^{-1}+a=\theta$. (Additive Inverse)

b) Multiplicative group

b.1) $a \times b \in S$ (Closure)

b.2) $(a \times b) \times c = a \times (b \times c)$ (Associative)

b.3) There is an element $\theta \in S$, such that $a \times \theta = a$ (Multiplicative Identity)

b.4) There is an $a^{-1} \in S$, such that for every non-zero element, $a \times a^{-1} = \theta$ (Multiplicative Inverse)

If, in addition to the above laws, a group satisfies the commutative law (that is, $a+b=b+a$, or $a \times b = b \times a$), then such a group is called Abelian, or commutative.

2.1.2. Ring

Definition: A ring R is a set of elements for which two operations are defined. One is addition, denoted by $a+b$, and the other is multiplication, denoted by $a \times b$.

Properties:

1. The set R is an Abelian group under addition.
2. The multiplicative closure law must be met.
3. The multiplicative associative law must be met.
4. $a(b+c) = ab+ac$ (Distributive Law)

A ring is called commutative if its multiplicative operation is commutative, i.e. $a \times b = b \times a$.

Example: The set of all real numbers is a ring under the operation of ordinary addition and multiplication.

2.1.3. Field

Definition: A field is a commutative ring in which the non-zero elements form a group under multiplication.

Example: A field must contain at least two elements. Therefore, the binary elements 0 and 1 will form a field.

2.2. Properties of finite fields [3,4,6]

As a result of the previous definition of a field, the following is derived:

2.2.1. The order of a field is the number of elements in the field.

2.2.2. The least positive integer n for which $\alpha^n=1$ is called the order of α.

2.2.3. α is a primitive n^{th} root of unity iff the order of α is n. In a field of order q, α is called a primitive field element iff the order of α is q-1.

2.2.4. Galois, the French mathematician, created the general theory of finite fields (called Galois fields.(Ref 2)

A Galois field will be defined by $GF(p^n)$ where p is a prime number, and n is any positive integer greater than zero. Only the binary system will be dealt with. Therefore, Galois field of the form $GF(2^m)$ will be used.

2.2.5. Definition:

A polynomial p(x) of degree m is said to be irreducible over the binary field GF(2) if p(x) is not divisible by any polynomial of degree less than m, and greater than zero.

2.2.6. Definition:

A polynomial g(x) of degree m over GF(2) is called primitive when $g(x) \mid (x^n-1)$ for $n=2^m-1$ and for no smaller n.

2.2.7. Example:

Selecting an irreducible polynomial of exponent 31 and of degree 5 over GF(2), and choosing α as the primitive element, the following Galois field of $2^5=32$ elements is derived:

Irreducible polynomial = $1+x^2+x^5$ $\qquad n=2^5-1=31$

1)	0	17)	$\alpha^{15}=1+\alpha+\alpha^2+\alpha^3+\alpha^4$
2)	1	18)	$\alpha^{16}=1+\alpha+\alpha^3+\alpha^4$
3)	α	19)	$\alpha^{17}=1+\alpha+\alpha^4$
4)	α^2	20)	$\alpha^{18}=1+\alpha$
5)	α^3	21)	$\alpha^{19}=\alpha+\alpha^2$
6)	α^4	22)	$\alpha^{20}=\alpha^2+\alpha^3$
7)	$\alpha^5=1+\alpha^2$	23)	$\alpha^{21}=\alpha^3+\alpha^4$
8)	$\alpha^6=\alpha+\alpha^3$	24)	$\alpha^{22}=1+\alpha^2+\alpha^4$
9)	$\alpha^7=\alpha^2+\alpha^4$	25)	$\alpha^{23}=1+\alpha+\alpha^2+\alpha^3$
10)	$\alpha^8=1+\alpha^2+\alpha^3$	26)	$\alpha^{24}=\alpha+\alpha^2+\alpha^3+\alpha^4$
11)	$\alpha^9=\alpha+\alpha^3+\alpha^4$	27)	$\alpha^{25}=1+\alpha^3+\alpha^4$
12)	$\alpha^{10}=1+\alpha^4$	28)	$\alpha^{26}=1+\alpha+\alpha^2+\alpha^4$
13)	$\alpha^{11}=1+\alpha+\alpha^2$	29)	$\alpha^{27}=1+\alpha+\alpha^3$
14)	$\alpha^{12}=\alpha+\alpha^2+\alpha^3$	30)	$\alpha^{28}=\alpha+\alpha^2+\alpha^4$
15)	$\alpha^{13}=\alpha^2+\alpha^3+\alpha^4$	31)	$\alpha^{29}=1+\alpha^3$
16)	$\alpha^{14}=1+\alpha^2+\alpha^3+\alpha^4$	32)	$\alpha^{30}=\alpha+\alpha^4$

TABLE 2.1.

Let f(x) be a polynomial of the form:

$$f(x)=f_k x^k+f_{k-1}x^{k-1}+\ \ldots\ +f_1 x+f_0$$

where f_i is either 0 pr 1.

Considering:

$$f^2(x)=(f_k x^k+f_{k-1}x^{k-1}+ \ldots +f_1 x+f_0)^2$$

$$= (f_k x^k)^2 + 2(f_k x^k)(f_{k-1}x^{k-1}+ \ldots +f_1 x+f_0)$$

$$+ (f_{k-1}x^{k-1}+ \ldots +f_1 x+f_0)^2$$

According to modulo-2 arithmetic, we have:

$$f^2(x)=f_k x^{2k}+(f_{k-1}x^{k-1}+ \ldots +f_1 x+f_0)^2$$

Expanding, we obtain:

$$f^2(x)=f(x^2)$$

It follows that, for any positive integer ℓ,

$$\left[f(x)\right]^{2^\ell}=f(x^{2^\ell})$$

Therefore, as α is a root of $1+x^2+x^5$; $\alpha^2,\alpha^4,\alpha^8$ and α^{16} are also roots of $1+x^2+x^5$.

Then: $1+x^2+x^5=(x+\alpha)(x+\alpha^2)(x+\alpha^4)(x+\alpha^8)(x+\alpha^{16})$

To determine the irreducible polynomial with α^3 as a root, it is sufficient to multiply the following:

$$(x+\alpha^3)(x+\alpha^6)(x+\alpha^{12})(x+\alpha^{24})(x+\alpha^{17})$$

2.2.8. A subset C of vectors of n binary digits is called a cyclic subspace if it contains the following two properties: (Ref. 7)

1. If v_1 and v_2 are in C, their sum mod 2 is also in C. That is, C is a subspace, or subgroup;
2. If $v = (a_0, a_1, \ldots a_{n-1})$ is in C, the vector $v^1 = (a_{n-1}, a_0, a_1, \ldots a_{n-2})$, obtained by shifting v cyclically one place, is also in C.

Let R_n denote the set of all polynomials

$$a_0+a_1x+a_2x^2+a_3x^3+ \ldots +a_{n-1}x^{n-1}$$

of degree less than n with coefficients 1 and 0. They form a group under modulo 2 addition. Multiplication can be defined modulo x^n-1. That is, these polynomials can be multiplied in the standard way, modulo 2, and reduced again to polynomials of degree less than n by the use of the equation $x^n=1$. Then R_n is a ring.

A subset I of R_n is called an ideal if it satisfies the following two properties:

1. I is a subset of R_n; and
2. if p(x) is in I, and a(x) is in R_n , then the product p(x)a(x) is in I.

Considering polynomials $p(x) = a_0+a_1x+ \ldots +a_{n-1}x^{n-1}$ to be vectors $(a_0, a_1, \ldots a_{n-1})$, a cyclic shift is the same as multiplication by x modulo x^n-1. Therefore, every ideal

is a cyclic subspace. Conversely, if p(x) is in a cyclic subspace C, so is x.p(x). It follows that $x^j p(x)$ must also be in C and, since C is a subspace:

$$\sum_j c_j x^j \; p(x) = p(x) \sum_j c_j x^j$$

must also be in C. Thus, if p(x) is in C, so is the product of p(x) and any polynomial. Therefore, every cyclic subspace is an ideal.

Lemma 1: (Ref. 7)

If p(x) and q(x) are in an ideal I, the greatest common divisor (GCD), d(x), of p(x) and q(x) is in I. This follows directly from the fact that it is always possible to express the d(x) in the form: $d(x)=a(x)p(x)+b(x)q(x)$ (Ref 3) where a(x) and b(x) are polynomials

Lemma 2: (Ref. 7)

All polynomials in an ideal I are multiples of the unique polynomial of the least degree in I.

Proof:

Let p(x) be a polynomial of least degree in I. Then, if q(x) is any other polynomial in I, the greatest common divisor of p(x) and q(x) is in I. If p(x) does not divide q(x), then the greatest common divisor of p(x) and q(x) would have a lower degree than p(x), which is a contradiction. Therefore, every

polynomial in I is divisible by $p(x)$. If $p_1(x)$ and $p_2(x)$ both have minimum degree, each must be divisible by the other and, hence, be equal. The ideal consisting of all multiples of $p(x)$ will be denoted by $[p(x)]$. The polynomial of least degree in an ideal is called its generator.

Lemma 3: (Ref. 7)

The generator $p(x)$ of an ideal is a factor of x^n-1.

Proof:

The GCD $d(x)$ of $p(x)$ and x^n-1 can be expressed in the form:

$$d(x)=a(x)\ p(x)+b(x)(x^n-1)$$

$$\equiv a(x)\ p(x)\ \mathrm{mod}(x^n-1).$$

Hence, $d(x)$ is in the ideal. But $p(x)$ is divisible by $d(x)$, and since $d(x)$ is in the ideal, then $d(x)$ is divisible by $p(x)$. Hence, $p(x) = d(x)$. These results can be summarized as follows:

Theorem: (Ref. 7)

A set of polynomials is an ideal in the ring of polynomials modulo x^n-1 if and only if it consists of all multiples of degree less than n of a factor of x^n-1.

Corollary: (Ref. 7)

If $p(x)$ is a polynomial of degree k, which divides into x^n-1, $[p(x)]$ is a vector space of dimension n-k.

Proof:

The elements of $[p(x)]$ are in the form $c(x)\ p(x)$, where $c(x)$ is an arbitrary polynomial of degree less than n-k. Then the n-k coefficients of $c(x)$ are arbitrary.

2.3. Standard array: [4,6]

2.3.1. Hamming weight and distance:

The Hamming weight of an n-tuple A (w(A)) is defined as the number of non-zero components of A.
If A = (01101010), then w(A) =4.
Let A and B be two n-tuples. Then, the Hamming distance between A and B, that is d(A,B), is defined as the number of components in which they differ.
If A = (10010) and
if B = (00111), then d(A,B) =3.
If A and B are both codewords of a linear block code, then A+B (mod 2) must also be a codeword, since the set of all codewords is a vectorspace. Therefore, the distance between any two codewords equals the weight of some other codeword, and the minimum distance for a linear code equals the minimum weight of its non-zero vectors.

Polynomial representation:

If we have a codeword $A = (a_0, a_1, \ldots a_{n-1})$, then correspondingly

we have a polynomial of the form:

$$A(x)=a_0+a_1x+a_2x^2+a_3x^3+ \ldots +a_{n-1}x^{n-1}$$

of degree n-1. A(x) is the codepolynomial of A. Henceforth, we will use the terms codeword and codepolynomial interchangeably.

2.3.2. Standard array:

Considering an (n,k) linear code, let $v_1, v_2, \ldots v_2k$ be the 2^k codewords. For any codeword which is transmitted over the noisy channel, the received vector r may be any of the 2^n n-tuples. To represent the 2^n n-tuples, a standard array is constructed as follows:

	0	v_1	v_2	v_2k
	λ_1	$v_1+\lambda_1$	$v_2+\lambda_1$....$v_2k+\lambda_1$	
2^{n-k} rows				
	λ_2	$v_1+\lambda_2$	$v_2+\lambda_2$....$v_2k+\lambda_2$	
	.	.	.	.
	.	.	.	.
	.	.	.	.

STANDARD ARRAY

This process is carried out until all the 2^n n-tuples

are exhausted.

Each row is called a coset. Each λ is called a coset leader. The technique of choosing a coset leader is as follows: when choosing a coset leader, all possible coset leaders of weight i should have been considered before choosing coset leaders of weight i+1. This approach ensures that the decoding procedure is the best one possible because of the fact that it corrects the errors with a minimum weight, i.e. with a high probability of occurrence.

2.4. BCH codes introduction: [2,6]

Some of the most powerful multiple error-correcting codes for random independent errors which have been discovered to date are the Bose-Chaudhuri-Hocquenghem (BCH) polynomial codes. These codes were developed independently around 1960 by R.C. Bose, D.K. Ray-Chaudhuri, and A. Hocquenghem. The discussion of BCH codes will begin with an examination of the Hamming codes.

2.4.1. Hamming codes:

These codes are a special case of the BCH codes, and were originally discovered by R.W. Hamming in 1950. The Hamming codes are single error-correcting codes with minimum distance $d=2t+1=3$. They are perfect in that, for any positive integer m, there exists an ($n=2^m-1$, $k=2^m-m-1$) code which corrects each single error which might occur. No

($n=2^m-1$, $k=2^m-m-1$) code can possibly be constructed which will correct more than all single errors. This says, in fact, that if a standard array is constructed, as seen in the previous section, the coset leaders will consist of 0 and exactly the 2^m-1 error patterns with a single 1.

A t-error-correcting code, which has all error patterns with a weight up to t as coset leaders, and no others, is called 'perfect' because no better code for correcting random errors can be found.

2.5. General encoding procedures for BCH codes: [4,5,6,18]

In their unshortened form, binary BCH codes exist for lengths $n=2^m-1$ and with, at most, mt check bits, they can correct any set of t-independent errors within the block of n bits, where m and t are arbitrarily positive integers. Initially, the BCH codes were described by a matrix, as follows:

$$M=\begin{bmatrix} 1 & 1 & \cdots\cdots & 1 \\ \alpha & \alpha^3 & \cdots\cdots & \alpha^{2t-1} \\ \alpha^2 & (\alpha^3)^2 & \cdots\cdots & (\alpha^{2t-1})^2 \\ \vdots & \vdots & & \vdots \\ \alpha^{2^m-2} & (\alpha^3)^{2^m-2} & \cdots\cdots & (\alpha^{2t-1})^{2^m-2} \end{bmatrix}$$

α is a primitive element of the field. This is a 2^m-1×t matrix of GF(2^m) elements. But, as each field element is a vector of m binary digits, this is a 2^m-1×mt matrix of binary digits.

A vector of 2^m-1 binary digits is considered a codeword if it satisfies the parity check of each column, i.e. if the product of this vector with the matrix is zero. Therefore, the set of all codewords is the null space of this matrix.

Another approach to encoding is made possible by using a generator polynomial g(x) which has for its roots

$$\alpha, \alpha^3, \alpha^5 \ldots \alpha^{2t-1}$$

Each element α^j of the field is a root of a unique irreducible polynomial $m_j(x)$ of minimum degree. Then, g(x) must be divisible by $m_1(x)$, $m_3(x)$, $m_5(x)$ ... $m_{2t-1}(x)$ and, hence, by their least common multiple:

$$g(x)=\text{LCM } (m_1(x), m_3(x), m_5(x), \ldots m_{2t-1}(x))$$

Since each of the factors of $m_j(x)$ is irreducible, and has no degree greater than m, the least common multiple of g(x) is simply the product of the polynomials $m_j(x)$ with the duplicates omitted. Duplication is possible and will occur for any α^i and α^j which are the roots of the same minimum function $m_i(x)$. Then, for a given m and t, g(x) will generate a BCH binary code with a length $n=2^m-1$, that will correct t-random errors, or less, and has no more than mt parity check digits.

This particular property of the generator polynomial is due to a theorem stating: if α is a primitive element of the $GF(2^m)$, and if the codeword length is, at most, 2^m-1, then the BCH code with symbols in GF(2) and encoding polynomial

$$g(x)=\text{LCM}\ (m_1(x),\ m_2(x),\ m_3(x),\ \ldots\ m_{d-1}(x))$$

has minimum distance of at least d where d=2t+1 (Ref. 2).

This thesis will not be concerned with generalized BCH codes, i.e. we will deal only with primitive narrow-sense BCH codes. For a primitive BCH code, n is restricted to be 2^m-1, for a non-primitive BCH code, n may be any other odd number. In a narrow-sense BCH code, if α is a primitive element of $GF(2^m)$ and d=2t+1, then f(x) is a codeword iff $\alpha,\alpha^2,\alpha^3,\ldots\alpha^{2t}$ are roots of f(x). In a non-narrow-sense BCH code, the generator polynomial g(x) is the polynomial of lowest degree for which α^{m_0}, α^{m_0+1}, ... α^{m_0+d-2} are roots.

2.5.1. Systematic encoding: [5]

When the generator polynomial is directly multiplied by the information polynomial, the resulting codeword is in a non-systematic form. To obtain a systematic code, such that the first n-k digits are the parity check digits, and the last k digits are the information digits, the following procedure could be used:

Let the k digits to be encoded be:

$$m=(m_0,m_1,m_2, \ldots m_{k-1}) \qquad \text{or}$$

$$m(x)=m_0+m_1x+m_2x^2+m_3x^3+ \ldots +m_{k-1}x^{k-1}$$

Multiplying m(x) by x^{n-k}, we obtain:

$$x^{n-k}m(x)=m_0x^{n-k}+m_1x^{n-k+1}+ \ldots +m_{k-1}x^{n-1}$$

Dividing $x^{n-k}m(x)$ by g(x), we obtain:

$$x^{n-k}m(x)=q(x)g(x)+r(x) \qquad (1)$$

Since the degree of g(x) is n-k, the degree of r(x) must be n-k-1, or less.

$$r(x)=r_0+r_1x+r_2x^2+r_3x^3+ \ldots +r_{n-k-1}x^{n-k-1}$$

Rearranging equation number (1), we obtain:

$$r(x)+x^{n-k}m(x)=q(x)g(x)$$

Therefore, $r(x)+x^{n-k}m(x)$ is a multiple of g(x) and thus it is a codeword of the cyclic code generated by g(x), and has a degree n-1, or less.

Thus:

$$r(x)+x^{n-k}m(x)=r_0+r_1x+ \ldots +r_{n-k-1}x^{n-k-1}+$$

$$m_0x^{n-k}+m_1x^{n-k+1}+ \ldots +m_{k-1}x^{n-1}$$

corresponding to the codeword:

$$\underbrace{(r_0 r_1 r_2 \cdots r_{n-k-1})}_{\text{Parity Check Digits}} \qquad \underbrace{(m_0 m_1 \cdots m_{k-1})}_{\text{Information Digits}}$$

By altering the generator polynomial into a generator matrix, it is possible to obtain the same answer.

2.5.2. Example: BCH encoding:

Let us construct a binary BCH code with a codeword length n=31, and a minimum distance d=7, thus capable of correcting three errors as $d=2t+1=7=2\times3+1$. Initially, an irreducible polynomial of degree 5 is selected to form $GF(2^5)$. We choose $1+x^2+x^5$ as the irreducible polynomial and α as the root of $1+x^2+x^5$. We then form the successive powers of α as in Table 2.1. To determine the minimal polynomial with α^3 as a root, we simply multiply over $GF(2^5)$:

$$(x+\alpha^3)(x+\alpha^6)(x+\alpha^{12})(x+\alpha^{24})(x+\alpha^{17})$$

To obtain $M(x)=1+x^2+x^3+x^4+x^5$
we must use the same procedure for α^5, i.e.

$$(x+\alpha^5)(x+\alpha^{10})(x+\alpha^{20})(x+\alpha^9)(x+\alpha^{18})$$

over $GF(2^5)$ equal to: $M(x)=1+x+x^2+x^4+x^5$.

The generator polynomial will be:

$$g(x) = (1+x^2+x^5)(1+x^2+x^3+x^4+x^5)(1+x^2+x^4+x^5)$$

$$g(x) = 1+x+x^2+x^3+x^5+x^7+x^8+x^9+x^{10}+x^{11}+x^{15}$$

In binary: the generator polynomial

= 1111010111110001

The number of information digits is equal to:

$$n-k = \text{degree of } g(x) = 31-k = 15$$

$$k = 31-15 = 16 \quad \text{Information digits}$$

Thus, a typical message might be:

1000000000000001

<u>i.e.</u> $1+x^{15}$

The resulting codeword will be $C(x) = g(x)(1+x^{15})$

$$C(x) = (1+x+x^2+x^3+x^5+x^7+x^8+x^9+x^{10}+x^{11}+x^{15})(1+x^{15})$$

$$C(x) = 1+x+x^2+x^3+x^5+x^7+x^8+x^9+x^{10}+x^{11}+x^{16}+x^{17}+x^{18}+x^{20}+x^{22}$$
$$+x^{23}+x^{24}+x^{25}+x^{26}+x^{30}$$

In binary (non-systematic form):

1111010111110000111010111110001

The systematic form will be obtained by:

a) Multiplying $(1+x^{15})(x^{31-16}) = (1+x^{15})(x^{15}) = x^{15}+x^{30}$

b) Dividing $x^{15}+x^{30}/g(x)$ =

$$x^{15}+x^{30}=g(x)\ (x^{15}+x^{11}+x^{10}+x^{9}+x^{8}+x^{3})+(x^{3}+x^{4}+x^{5}+x^{6}+x^{11}+x^{14})$$

In the systematic form, the codeword will be:

$$x^{3}+x^{4}+x^{5}+x^{6}+x^{11}+x^{14}+x^{15}+x^{30}$$

000111100001001 1000000000000001

Check Digits Information Digits

CHAPTER THREE

Once the message to be transmitted has been encoded and sent through the channel, a decoding operation must be performed at the receiving end to obtain the original message.

Bounded-distance decoding consists basically of the following procedures:

a) determination of the Syndromes (Power Sums)

b) determination of the error-locator polynomial, using the Peterson or Berlekamp procedure.

c) determination of the roots of the error-locator polynomial.

d) the message, depending on whether the encoding used is to be systematic or non-systematic, will be contained in the last k-digits, or in the quotient resulting from the division of the corrected polynomial by the generator polynomial, respectively.

By using the term bounded-distance decoding, we mean if the actual number of errors is less than the error-correcting capability of the code t ($e \leq t$), correct decoding will occur. However, if the actual number of errors is greater than t ($e > t$), the

decoder will give an incorrect result, or will simply not provide a decision. In this chapter, we will deal only with steps a and b of the decoding procedure, studying the decoding delays associated with the Peterson and Berlekamp techniques. The main point of this thesis, step c, will be discussed in Chapter Four.

3.1. Determination of the Syndromes: [3,6]

If the encoder transmits the binary BCH codeword:

$$c(x)=\sum_{i=0}^{n-1} C_i x^i$$

and the channel causes additive errors given by the coefficients of the binary polynomial:

$$E(x)=\sum_{i=0}^{n-1} E_i x^i$$

then the received word will be given by:

$$R(x)=\sum_{i=0}^{n-1} R_i x^i=\sum_{i=0}^{n-1} C_i x^i+\sum_{i=0}^{n-1} E_i x^i$$

For $j = 1,2, \ldots 2t$, the codeword is a multiple of the irreducible polynomial of α^j and, therefore:

$$R(\alpha^j)=0+\sum_{i=0}^{n-1} E_i \alpha^{ji} = S_j$$

Thus, to obtain the syndromes, it is possible to simply substitute the received polynomial with the relevant various non-zero elements of the Galois field to obtain the values of $S_1, S_2, \ldots S_{2t}$.

Another method is available. It consists of dividing the received polynomial by the irreducible polynomial with α as the root to determine S_1.

We may write: $R(x) = M(x)Q(x)+r(x)$

where $M(x)$ is the irreducible polynomial with α as a root and $r(x)$ is the remainder with a degree less than $M(x)$.

Therefore: $R(\alpha) = M(\alpha)Q(\alpha)+r(\alpha)$

But $M(\alpha) = 0$ as α is a root and $R(\alpha) = r(\alpha)$.

Thus, after the binary polynomial $R(x)$ is divided by the binary polynomial $M(x)$, the syndrome is given by the remainder polynomial $r(x)$, evaluated at $x = \alpha$.

To compute S_3, we divide $R(x)$ by $M^3(x)$, which is the irreducible polynomial of α^3, to obtain the remainder $r^{(3)}(x)$. We then compute:

$$S_3 = r^{(3)}(\alpha^3)$$

To summarize: if (a,b,c, ...) are error locations, then, by definition:

$$S_1=\alpha^a+\alpha^b+\alpha^c+ \ldots$$

$$S_2=\alpha^{2a}+\alpha^{2b}+\alpha^{2c}+ \ldots$$

$$S_3=\alpha^{3a}+\alpha^{3b}+\alpha^{3c}+ \ldots$$

Since we are dealing with the binary case:

$$S_{2k} = S_k^2 \text{ ; meaning } S_1^2 = S_2, \quad S_2^2 = S_4, \ \ldots$$

3.1.1. Determination of the Syndromes for a BCH code with n=31 (code length), k=16 (number of information digits), and t=3 (maximum number of correctable errors).

Example: Assuming that the codeword sent is the same as in Example 2.5.2., i.e. equal to:

$$1111010111110000111010111110001$$

We will consider the case of three errors equal to:

$$x^2+x^8+x^{23}$$

the received polynomial will be:

$$1101010101111000011101010111000 01$$

or

$$R(x)=1+x+x^3+x^5+x^7+x^9+x^{10}+x^{11}+x^{16}+x^{17}+x^{18}+x^{20}+x^{22}$$
$$+x^{24}+x^{25}+x^{26}+x^{30}$$

Determination of the Syndromes:

a. Using the substitution method:

$$S_1=\alpha+\alpha^3+\alpha^5+\alpha^7+\alpha^9+\alpha^{10}+\alpha^{11}+\alpha^{16}+\alpha^{17}+\alpha^{18}+\alpha^{20}+\alpha^{22}+\alpha^{24}+\alpha^{25}+\alpha^{26}$$
$$+\alpha^{30}+1$$

$$S_1 = \alpha^{19}$$

$$S_3=1+\alpha^3+\alpha^9+\alpha^{15}+\alpha^{21}+\alpha^{27}+\alpha^{30}+\alpha^2+\alpha^{17}+\alpha^{20}+\alpha^{23}+\alpha^{29}+\alpha^4+\alpha^{10}$$
$$+\alpha^{13}+\alpha^{16}+\alpha^{28}$$

$S_3 = 0$

$S_5 = 1+\alpha^5+\alpha^{15}+\alpha^{25}+\alpha^4+\alpha^{14}+\alpha^{19}+\alpha^{24}+\alpha^{18}+\alpha^{23}+\alpha^{28}+\alpha^7+\alpha^{17}$

$+\alpha^{27}+\alpha^6+\alpha^{26}+\alpha$

$S_5 = \alpha^{24}$

b. Using the division method:

S_1=Remainder of division $\left(\frac{R(x)}{\text{Irreducible polynomial with } \alpha \text{ as a root}}\right)$

S_1=Remainder $\left(1+x+x^3+\frac{x^5+x^7+x^9+x^{10}+x^{11}+x^{16}+x^{17}}{(1+x^2+x^5)}+x^{18}+x^{20}+x^{22}\right.$

$\left.+x^{24}+x^{25}+x^{26}+x^{30}\right)$

$S_1 = \alpha^{19}$

S_3=Remainder $\left(\frac{R(x)}{\text{Irreducible polynomial with } \alpha^3 \text{ as a root}}\right)$ at $x = \alpha^3$

S_3=Remainder $\left(\frac{R(x)}{1+x^2+x^3+x^4+x^5}\right)$

$S_3 = 0$

S_5=Remainder $\left(\frac{R(x)}{\text{Irreducible polynomial with } \alpha^5 \text{ as a root}}\right)$ at $x = \alpha^5$

S_5=Remainder $\left(\frac{R(x)}{1+x+x^2+x^4+x^5}\right)$ at $x = \alpha^5$ $\quad S_5 = \alpha^{24}$

3.2. Once we have obtained the Syndromes to determine the position of the errors, it is necessary to solve a set of t-equations in t-unknowns. To solve this problem, two procedures are presently available: the Peterson and the Berlekamp techniques.

3.2.1. Peterson technique: [7,6]

The elementary symmetric functions σ_i are related to the power sum symmetric functions S_j by Newton's identities, as follows:

$$S_1-\sigma_1=0$$

$$S_2-S_1\sigma_1+2\sigma_2=0$$

$$S_3-S_2\sigma_1+S_1\sigma_2-3\sigma_3=0$$

$$S_4-S_3\sigma_1+S_2\sigma_2-S_1\sigma_3+4\sigma_4=0$$

$$S_5-S_4\sigma_1+S_3\sigma_2-S_2\sigma_3+S_1\sigma_4-5\sigma_5=0$$

$$\cdots\cdots\cdots$$

Considering the binary case only, we will obtain:

$$S_1=\sigma_1$$

$$S_3=S_2\sigma_1+S_1\sigma_2+\sigma_3$$

$$S_5=S_4\sigma_1+S_3\sigma_2+S_2\sigma_3+S_1\sigma_4+\sigma_5$$

$$\cdots\cdots\cdots$$

To summarize: If, for example, a,b,c,... are error locations, then, by definition , the elementary symmetric functions (σ's) are:

$$\sigma_1=\alpha^a+\alpha^b+\alpha^c+ \ldots$$

$$\sigma_2=\alpha^{a+b}+\alpha^{b+c}+\alpha^{a+c}+ \ldots$$

$$\sigma_3=\alpha^{a+b+c}+ \ldots$$

$$\ldots$$

Therefore, in a t random error-correcting BCH code, for i>t, $\sigma_i=0$.

Newton's identities in a matrix form are arranged as follows:

$$\begin{bmatrix} S_1 \\ S_3 \\ S_5 \\ \vdots \\ S_{2t-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0\ldots\ldots \\ S_2 & S_1 & 1 & 0 & 0 & 0\ldots\ldots \\ S_4 & S_3 & S_2 & S_1 & 1 & 0\ldots\ldots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ldots\ldots\ldots \\ S_{2t-2} & S_{2t-3} & . & . & . & \ldots S_{t-1} \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \vdots \\ \sigma_t \end{bmatrix}$$

It is possible to prove (Ref. 7) that the t×t (M) matrix is non-singular if the power sums symmetric functions S_j are the power sums of t or t-1 distinct field elements, and is singular if the S_j are power sums of fewer than t-1 distinct field elements.

If there are actually t-1 errors, then the solution for the σ's will result in $\sigma_t=0$. The corresponding polynomial equation will have zero as one root. Initially, the Peterson (matrix) technique operates basically by assuming first of all that no more than t errors have occurred in the codeword. We then check the value of the M determinant in the case of a t×t matrix; if $M\neq 0$, then we have t or t-1 errors and we can proceed with the evaluation of the σ's from the matrix. However, if $M=0$, then we have t-2, or less, errors and, in this case, we drop the last two equations and repeat the procedure, i.e. the evaluation of the determinant M until we reach an $M\neq 0$. If no errors have occurred, then it will not be possible to obtain an $M\neq 0$.

It would also be possible to obtain the same result by starting with the assumption that two errors have occurred, and then solving and checking the solution. If the solution does not check, four errors would be assumed, and so forth. When a set of answers that checks occurs, it must be the correct solution.

Once the σ's are determined; by substituting each of the 2^m-1 field elements into the equation (called the error locator polynomial)

$$x^t+\sigma_1 x^{t-1}+\sigma_2 x^{t-2}+ \ldots +\sigma_t$$

it is possible to obtain the error locations. For each digit in the received vector, the corresponding $GF(2^m)$ element is

replaced in the equation. If the equation is satisfied, this bit is wrong and must be changed. If the equation is not satisfied, the bit is correct.

A detailed study of the error locator polynomial will be performed in Chapter Four.

3.2.2. Berlekamp technique: [3,6]

The Berlekamp technique is an iterative algorithm for decoding BCH codes. Using this approach, the error-locator polynomial is defined by:

$$\sigma(z)=\prod_{i=1}^{e}(1+x_i z)=1+\sum_{j=1}^{e}\sigma_j z^j$$

where we let the Galois field error locations $x_1, x_2, \ldots x_e$ denote the positions where $E_i=1$. Once the decoder has determined the error locator polynomial $\sigma(z)$, the reciprocal roots of $\sigma(z)$ can be found and the errors located. To relate the σ's and the S's, Berlekamp introduced the generating function:

$$S(z)=\sum_{j=1}^{\infty}S_j z^j=\sum_{j=1}^{\infty}\sum_{i=1}^{e}x_i^j z^j=\sum_{i=1}^{e}\frac{x_i z}{1+x_i z}$$

Multiplying by $\sigma(z)$ gives:

$$S(z)\sigma(z)=\sum_{i=1}^{e}\frac{x_i z}{1+x_i z}\prod_{j=1}^{e}(1+x_j z)=\sum_{i=1}^{e}x_i z\prod_{j\neq 1}(1+x_j z)$$

Adding $\sigma(z)$ to both sides gives:

$$[1+S(z)]\ \sigma(z)=\sigma(z)+\sum_{i=1}^{e}x_i z\prod_{j\neq 1}(1+x_i z)$$

Defining the polynomial $w(z)=\sum_{k=0}^{e} w_k z^k$
by the equation

$$w(z)=\sigma(z)+\sum_{i=1}^{e} x_i z \prod_{j\neq 1} (1+x_j z)$$

we then obtain: $\left[1+S(z)\right]\ \sigma(z)=w(z)$

The decoder knows only the coefficients of the first 2t powers of z in S(z). It does not know $S_{2t+1}, S_{2t+2}, S_{2t+3}$, ... and so forth. Therefore, we obtain:

$$\left[1+S(z)\right]\ \sigma(z)\equiv w(z) \mod z^{2t+1}$$

which is Berlekamp's key equation.

Given S(z), and using an iterative procedure, it is possible to determine both $\sigma(z)$ and w(z) from the key equation. The algorithm will be stated only for the binary system; the complete proof could be found in Ref (3).

Abbreviated algorithm for binary BCH codes:

Initially define $\sigma^{(0)}=1$, $\tau^{(0)}=1$. Proceed recursively as follows:

If S_{2k+1} is unknown, stop; otherwise define $\Delta_1^{(2k)}$ as the coefficient of z^{2k+1} in the product $(1+S)\sigma^{(2k)}$.
Let $\sigma^{(2k+2)}=\sigma^{(2k)}+\Delta_1^{(2k)} z\ \tau^{(2k)}$

$$\tau^{(2k+2)} = \begin{cases} z^2\tau^{(2k)} & \text{if } \Delta_1^{(2k)}=0 \quad \text{or if } \deg \sigma^{(2k)}>k \\ \dfrac{z\sigma^{(2k)}}{\Delta_1^{(2k)}} & \text{if } \Delta_1^{(2k)}\neq 0 \text{ and } \deg \sigma^{(2k)}\leq k \end{cases}$$

Example 3.2.2.1.

Using the same syndromes as in example 3.1.1., we will obtain the error locator polynomial by the Berlekamp technique. The initial conditions are always the same and were determined in order to ensure that the error locator polynomial will have minimum degree.

$$S_1=\alpha^{19},\ S_3=0,\ S_5=\alpha^{24},\ \sigma^{(0)}=1,\ \tau^{(0)}=1$$

$\Delta_1^{(0)}$ is defined as the coefficient of z in the product $(1+S)\sigma^{(0)}$ <u>i.e.</u> $(1+S_1z+S_2z^2+\ \ldots)\sigma^{(0)}$

$$\Delta_1^{(0)}=S_1=\alpha^{19}$$

Setting k=0, the two iterative equations $\sigma^{(2)}$ and $\tau^{(2)}$ will have the form:

$$\sigma^{(2)} = \sigma^{(0)} + \Delta_1^{(0)}\ z\ \tau^{(0)} = 1+\alpha^{19}z$$

$$\tau^{(2)} = \frac{z\sigma^{(0)}}{\Delta_1^{(0)}} = z\alpha^{12}$$

$\Delta_1^{(2)}$ coefficient of z^3 in $(1+S)\ \sigma^{(2)}= \alpha^{26}$

Setting k=1, the two iterative equations $\sigma^{(4)}$

and $\tau^{(4)}$ will have the form:

$$\sigma^{(4)} = \sigma^{(2)} + \Delta_1^{(2)} \; z \; \tau^{(2)} = 1+\alpha^{19} \; z + \alpha^7 z^2$$

$$\tau^{(4)} = \frac{z\sigma^{(2)}}{\Delta_1^{(2)}} = z\alpha^5 + z^2\alpha^{24}$$

$\Delta_1^{(4)}$ coefficient of z^5 in $(1+S)\,\sigma^{(4)}$

$$\Delta_1^{(4)} = S_5 + \alpha^{19}S_4 + \alpha^7 S_3 = \alpha^9$$

For k=2 we have:

$$\sigma^{(6)} = \sigma^{(4)} + \Delta_1^{(4)} \; z \; \tau^{(4)}$$

$$\sigma^{(6)} = 1 + \alpha^{19}z + \alpha^{29}z^2 + \alpha^2 z^3$$

Error locator polynomial:

$$x^3 + \alpha^{19}x^2 + \alpha^{29}x + \alpha^2$$

3.3. Analysis of decoding delays: [10,11,12]

One of the purposes of this thesis is to emphasize that in case of a small number of errors, the Peterson technique is preferable to the Berlekamp approach if it is possible to store efficiently the expressions for the various sigmas obtained from the matrix procedure. This problem was tackled by simulating both methods in the case of a code of length 31 bits.

The two programs, using an all-zero codeword at the sending end, were written by taking into consideration

only the decoding delay, i.e. every step in the program was carefully studied to achieve the highest possible speed of execution since memory was always assumed to be cheap and unlimited.

Always assuming that the all-zeros codeword is sent, both methods were simulated as follows:

a) n=31 t=4 (maximum number of correctable errors)

Then, by independently feeding all the various combinations of one- and two-errors, and 465 distinct error patterns of three- and four-errors, it was possible to determine the time necessary to independently process the 1,2,3, and 4 error patterns.

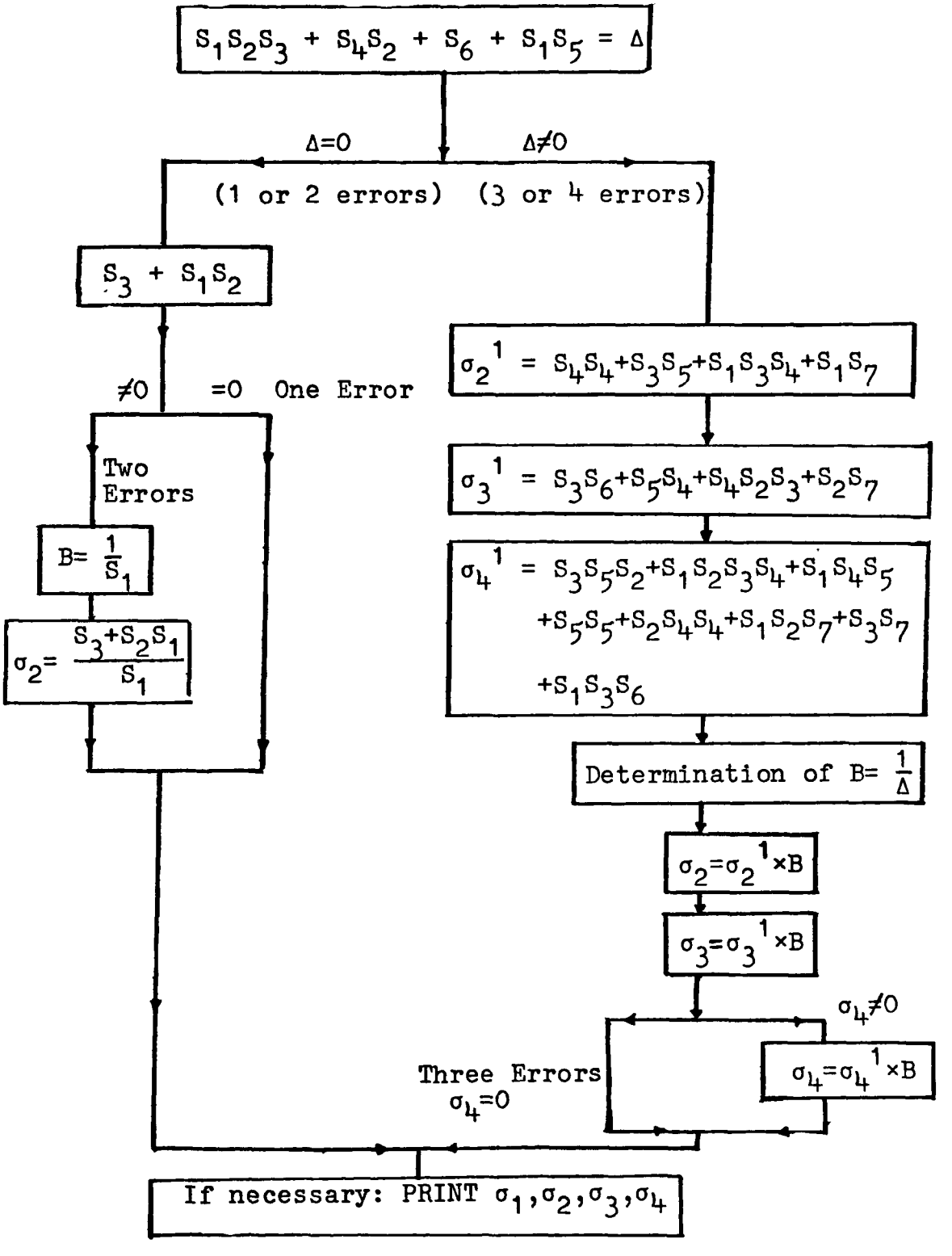
b) n=31 t=3 (maximum number of correctable errors)

The same procedure as in (a) was used, but with a maximum number of errors equal to three.

The following are flow diagrams for the Berlekamp and Peterson procedures in the case of a four error-correcting code:

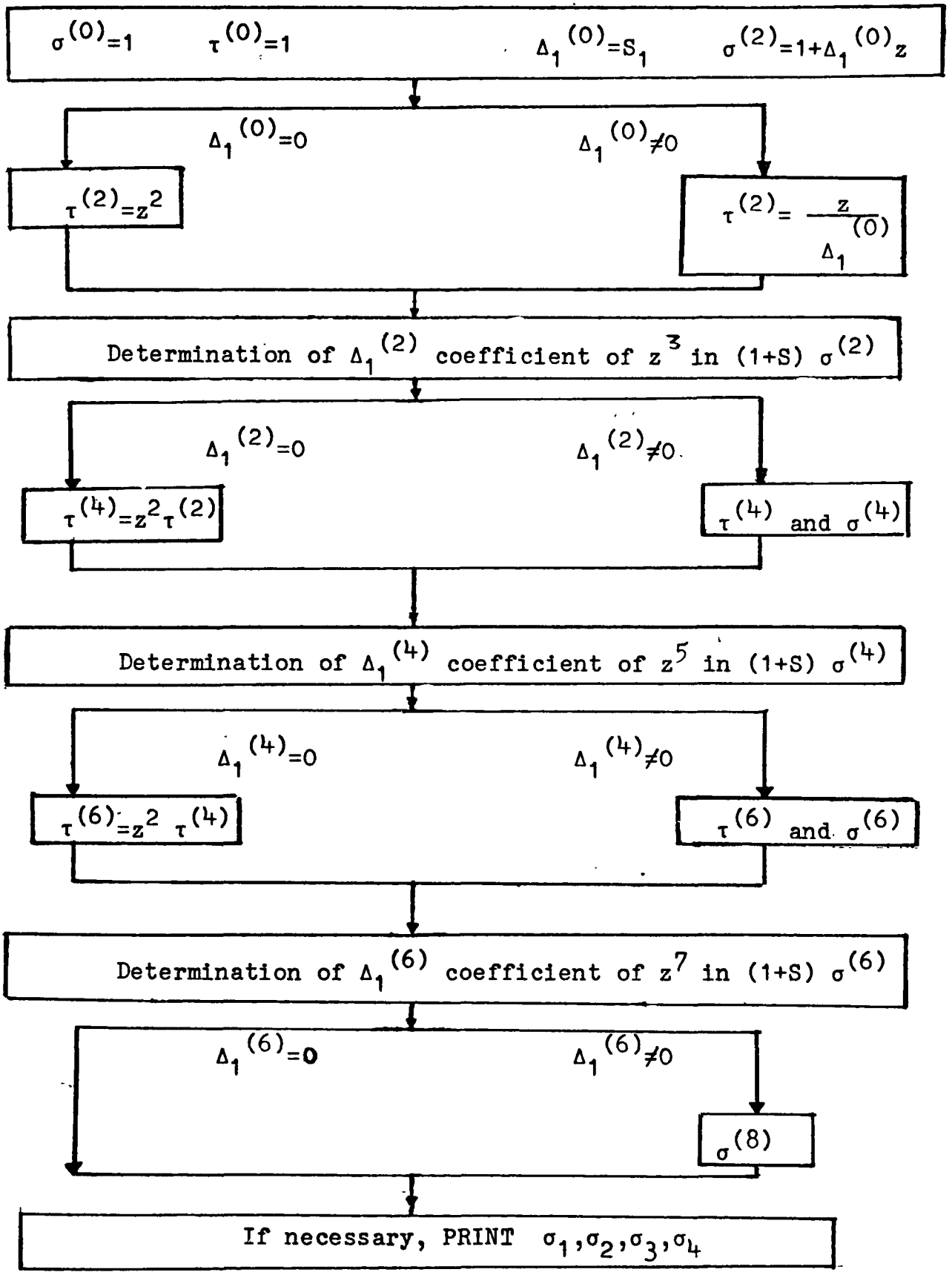
PETERSON TECHNIQUE

Flow Diagram (4 Errors)



BERLEKAMP TECHNIQUE

Flow Diagram (four errors)



3.3.1. Decoding time comparison between the Peterson and the Berlekamp Algorithms:

To compare the Peterson and Berlekamp algorithms, it was necessary to find a technique that would provide an accurate and precise measure of the time required to perform each algorithm.

Initially, a high-level language, FORTRAN H (with an optimized compiler), was used to simulate both algorithms. This approach was chosen because, in the Fortran printout from the IBM 360, the execution time is readily available. However, due to the very poor repeatability of the execution time, it was not possible to obtain adequate results. Although many precautions were taken to ensure that each job submitted was only processed when the IBM 360 was free from any other job, the results were only meaningful when the difference between the two algorithms was substantial.

The execution time, in seconds, was very high. This was due mostly to the inefficiency of the high-level language compiler.

From the obtained results, in the case of $n=31$ and $t=4$, it was not possible to draw any valid conclusion in the case of three or four errors. However, in the case of one or two errors, the Peterson technique was found to be definitely faster.

3.3.2. Results of the Comparison using an Intel 8080 Microprocessor:

Since it was not possible to obtain sufficiently repeatable and accurate results using the IBM 360, all programs were simulated on an MDS 800 system (MDS = Microcomputer Development System) using an 8080 microprocessor.

To measure real-time, using the MDS 800 system, we used the interrupt signal generated by the clock every 0.977 milliseconds, corresponding to a frequency of 1.024 Khz. The procedure was, basically, as follows (Fig. 3.3.2.1.):

1. To accommodate the facts that in an MDS 800 system:

a) both the operating system and the real-time clock are using the same priority level 1 (the 8080 has an eight-level priority interrupt structure); and

b) it is a hardware property that whenever an Interrupt occurs at level 1, the program execution will jump at a predetermined memory location (Intel 8080 memory location = 8H). Therefore, at the beginning of the source program, we moved the address of the Interrupt program to memory locations 9H and 10H.

Thus, whenever an Interrupt occurs, the program execution jumps automatically to location 8H, where there is a jump instruction to the beginning of the Interrupt program.

2. In the source program, various memory locations are cleared, and these locations will be used as a counter, storing the number of interrupts that have actually occurred during the execution of the problem program, (i.e. during the execution of the Peterson or the Berlekamp algorithm).
3. Considered as a subroutine of the source program, the problem program is called.
4. If an interrupt occurs during the execution of the problem program, the problem program is stopped and the CPU jumps to the interrupt program.
5. The first step in the interrupt program (subroutine) is to disable the interrupt hardware to prevent any further interrupts from occurring while the present interrupt is being serviced.
6. The memory locations, used as a counter and cleared in step 2, are incremented by one. Thus, we have a counter which records the total number of interrupts occurring during the execution of the problem program.
7. Before returning to the problem program, the interrupt system is re-activated so that other interrupts can occur.

8. The CPU jumps back to the problem program.

Once the execution of the problem program is completed, it is a simple matter to display the content of the memory locations where the total number of interrupts, that have occurred, is stored.

Knowing the total number of interrupts, and the time necessary to perform an interrupt, it is possible to obtain the time taken to execute the problem program.

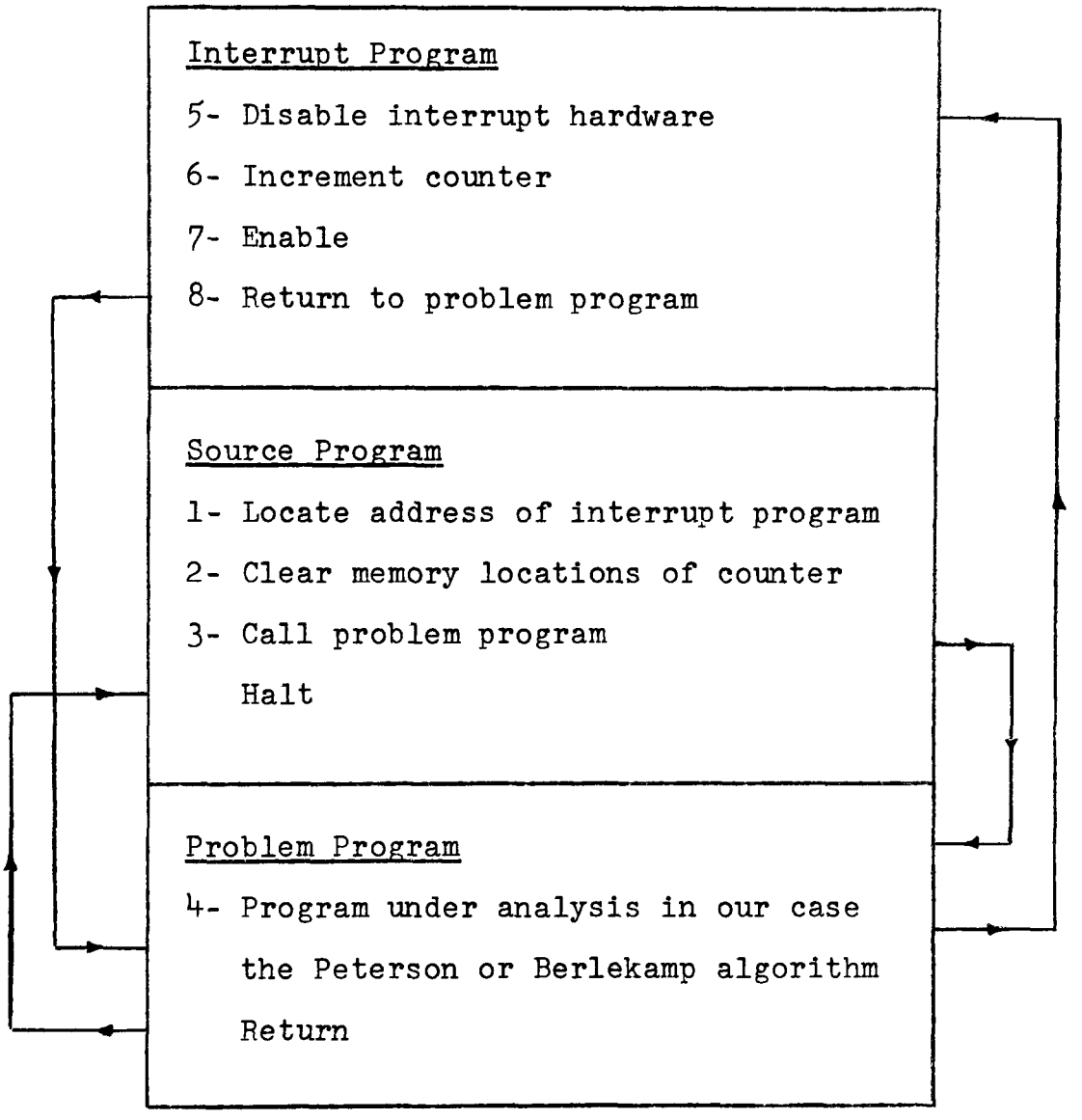

Interrupt Program
5- Disable interrupt hardware
6- Increment counter
7- Enable
8- Return to problem program
Source Program
1- Locate address of interrupt program
2- Clear memory locations of counter
3- Call problem program
Halt
Problem Program
4- Program under analysis in our case
the Peterson or Berlekamp algorithm
Return

Fig 3.3.2.1.

For a three Error-correcting code ($n=31=2^5-1$) the following results were obtained:

Number of errors in a word of n=31	Number of words of n=31	Execution Time
i.e. the simulation was performed by feeding the all-zeros codeword with a number of errors	i.e. the all-zero codeword, with a fixed number of errors but at different locations, was fed into the algorithm a number of times equal to	
1	31 (All combinations of single errors)	Berlekamp = 5.2 × Peterson algorithm execution time algorithm execution time
2	465 (All combinations of double errors)	Berlekamp = 1.2 × Peterson
3	465	Berlekamp = 1.3 × Peterson

Table 3.3.2.2.

For a four Error-correcting code ($n=31=2^5-1$) the following results were obtained:

Number of errors in a word of n=31	Number of words of n=31	Execution time
1	31 (All combinations of single errors)	Berlekamp = 2.6 × Peterson algorithm execution time algorithm execution time
2	465 (All combinations of double errors)	Berlekamp = 2.08 × Peterson
3	465	Berlekamp = .94 × Peterson
4	465	Berlekamp = 1.02 × Peterson

Table 3.3.2.3.

From table 3.3.2.2. and 3.3.2.3., it is evident that the Peterson technique is faster than the Berlekamp procedure if the maximum number of correctable error patterns of a code does not exceed three.

In the case of a four error-correcting code, the Peterson technique is not always faster than the Berlekamp procedure. Thus, it is possible to conclude that, for a small number of errors, i.e. with t not exceeding three, the Peterson (matrix) approach is preferable to the Berlekamp (iterative) procedure. The Peterson technique has, also, the advantage, of being able to determine the various sigmas in parallel and, therefore, of reducing the decoding time considerably. This property is not readily available in the iterative Berlekamp procedure, as it is necessary to compute sequentially each step.

The time necessary to execute the various algorithms was in the range of milliseconds. Berlekamp suggested a computer with which it is possible, according to him, to obtain a rate of a million bits per second.

In our case, the rates were low because of the structure of the MDS 800 system, where it is most time-consuming to fetch or store data from or into the memory.

CHAPTER FOUR

As previously mentioned, the decoding of binary $(n=2^m-1,k,t)$ BCH codes involves two basic steps:

(i) Find the error-locator polynomial $\psi(x)$.

(ii) Find the roots of $\psi(x)$.

We have already discussed (i) in chapter three. Regarding (ii), the idea is essentially to compute $\psi(\alpha^i)$ and see for what values of i, $\psi(\alpha^i)=0$, where α is the primitive element of the $GF(2^m)$.

In this chapter, we will briefly survey common techniques for the roots determination and, specifically, show the following:

a) The roots of the quadratic x^2+x+k_2 over $GF(2^m)$ can be found by solving a set of simple simultaneous equations.

b) It is known that the cubic x^3+x+k_3 over $GF(2^m)$ has an odd number of irreducible factors iff $Tr(k_3^{-1}) = Tr(1)$, where Tr indicates the trace. Since there are 2^{m-1} values of K_3^{-1} for which $Tr(k_3^{-1}) = Tr(1)$ and, since we are interested only in those values, of k_3, for which the cubic has three roots in $GF(2^m)$, the number of values, of k_3, in which we are interested, is $<2^{m-1}$. In this chapter, we will show that the number of such values of k_3 is actually $\leq\frac{2^{m-1}}{3}$. This means that

if we store the three roots of the cubic for each relevant value of k_3, and use a table-look-up approach for finding the roots of the cubic for a given k_3, we then have, at most, n/6 values of k_3 to consider. This result was published by F. Polkinghorn (Ref. 16) but here we give a simpler proof, as we are using the trace concept.

c) The reason we are interested only in the case, when $\psi^{(x)}$ has all of its roots in the $GF(2^m)$ under consideration, is the fact that the condition of $\psi(x)$, not having all of its roots in $GF(2^m)$, indicates the situation of detection-only. Thus, it would be advantageous to know whether or not $\psi(x)$ has all of its roots in $GF(2^m)$ without having to compute $\psi(\alpha^i)$. In this connection, we show that the quartic $(x^4+k_2x^2+k_3x+k_4)$ over $GF(2^m)$ does not have some or all of its roots in $GF(2^m)$ if $Tr(\frac{k_2^3}{k_3^2}) \neq Tr(1)$.

We also give other results arising out of this.

4.1. Chien search technique for finding roots of the error locator polynomial: [8]

The Chien procedure is based on the cyclic property of the BCH codes. Using this procedure, it is possible to locate the errors without explicitly solving the error locator polynomial.

The technique is described by examining the relationship between the sigma's (Elementary symmetric functions) and the roots β_j of the error locator polynomial:

$$x^t+\sigma_1 x^{t-1}+\sigma_2 x^{t-2}+ \ldots +\sigma_t = (x+\beta_1)(x+\beta_2) \ldots (x+\beta_t)$$

By definition:

$$\left.\begin{aligned} \sigma_1 &= \sum_{j=1}^{t} \beta_j \\ \sigma_2 &= \sum_{\substack{j,k=1 \\ j<k}}^{t} \beta_j\beta_k \quad \text{(Sum of the roots taken two by two)} \\ \sigma_3 &= \sum_{\substack{i,j,k=1 \\ i<j<k}}^{t} \beta_i\beta_j\beta_k \end{aligned}\right\} \quad (1)$$

Transforming each β_j (j=1,2, ... t) to $\bar{\beta}_j = \alpha\beta_j$ where α is the primitive element of $GF(2^m)$, we may define the new elementary symmetric functions $\bar{\sigma}_k$'s as functions of $\bar{\beta}_j$'s in the same manner as in (1).

The relationship between $\bar{\sigma}_k$'s and σ_k's is:

$$\bar{\sigma}_k = \alpha^k\sigma_k \rightarrow k = 1,2, \ldots t$$

Therefore, the $\bar{\beta}_j$'s are roots of the polynomial:

$$x^t+\bar{\sigma}_1 x^{t-1}+\bar{\sigma}_2 x^{t-2}+ \ldots +\bar{\sigma}_t$$

after applying such transformation τ times:

$$\bar{\beta}_j = \alpha^\tau\beta_j \rightarrow j = 1,2, \ldots t$$

and

$$\bar{\sigma}_k = \alpha^{\tau k}\sigma_k \rightarrow k = 0,1,2, \ldots t$$

Using this principle, and the fact that $\alpha^n = \alpha^{2^m-1} = 1$ in $GF(2^m)$, it is possible to obtain all the roots of the error locator polynomial by counting successive transformations, if we can detect whether a specific element of $GF(2^m)$ is a root of the error locator polynomial. To simplify the circuitry, the element to be detected is chosen to be the unit element of $GF(2^m)$.

When $\alpha^0 = 1$ is a root of the error locator polynomial, we find that:

$$x^t+\sigma_1 x^{t-1}+\sigma_2 x^{t-2}+ \ldots +\sigma_t \qquad \text{is}$$

$$1+\sigma_1+\sigma_2+ \ldots +\sigma_t = 0$$

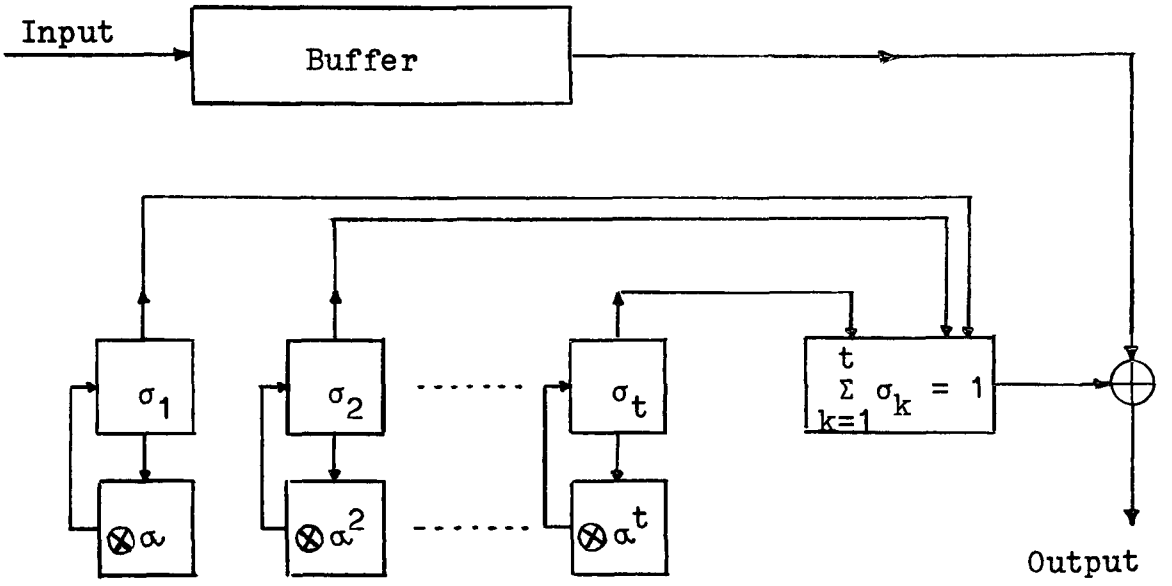
$$\sum_{k=1}^{t} \sigma_k = \sigma_1+\sigma_2+ \ldots +\sigma_t = 1$$

If $\Sigma\bar{\sigma}_k = 1$ after $\tau_1, \tau_2, \ldots \tau_t$ shifts, respectively, the roots of the error locator polynomial are:

$$\alpha^{n-\tau_1}, \alpha^{n-\tau_2}, \ldots \alpha^{n-\tau_t}$$

The implementation of such a procedure is simple and follows from theory in a straight forward manner.

The entire circuit is shown in Fig. 4.1.1.

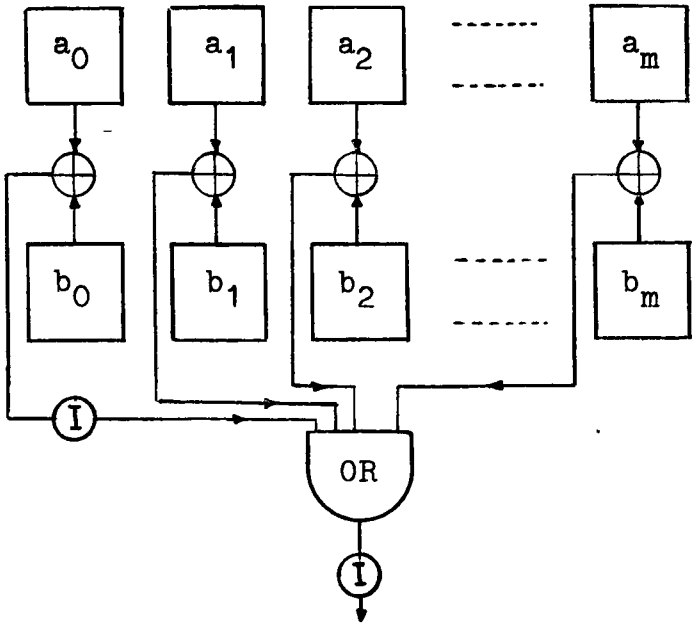


Cyclic Error-location Unit Fig. 4.1.1.

The initial values of σ_k's are stored in t, σ-registers, and the received sequence is stored in the buffer with high order bits first.

The circuits indicated by $\otimes\alpha^k$ are α^k-multipliers, their purpose being to multiply the current contents of the σ_k-register by α^k and subsequently store the product in the σ_k-register.

The circuit for detecting the condition $\Sigma\sigma_k = 1$ is a simple adder with an OR gate, followed by an inverter at the output. The sum of the zero-order is inverted, because the sum must be equal to one, not zero. (Fig. 4.1.2.)



σ_1 register

σ_2 register

Circuit for detecting $\Sigma\sigma_k=1$

Fig. 4.1.2.

The inverter output is a "one" if all inputs to the OR gate are zeros.

The error-location system of Fig. 4.1.1. operates as follows:

The received sequences are stored in the buffer with high-order bits first, and the initial values of σ_k's are stored in the σ_k-registers. Since the initial value of σ_k's will only indicate to the detection circuit the presence of errors at the α^{0th} position, the α^k multipliers are first pulsed once. After this shift transformation, the detection circuit indicates the presence of any error at the leading bit position of the buffer since that position corresponds to α^{n-1}. The bits in the buffer are then shifted out in sequence. Whenever a bit is in error, the detection circuit will produce a 'one' bit and, therefore, complement the data bit leaving the buffer at that time. All errors will be corrected with n-shifts, provided no more than t-errors

are present in the received sequence. This process will be illustrated by the following example:

Example 4.1.1.

We will use the following error locator polynomial:

$$x^3+\alpha^{19}x^2+\alpha^{29}x+\alpha^2$$

This polynomial was previously determined in Example 3.2.2.1. where $n=2^5-1=31$, $t=3$, and $\sigma_1=\alpha^{19}$, $\sigma_2=\alpha^{29}$, and $\sigma_3=\alpha^2$.

To obtain the roots by the Chien search technique, we will first multiply σ_1, σ_2, σ_3 by α, α^2, and α^3 respectively. Then, we will add the three resulting polynomials. If the sum is equal to one, then the bit under consideration is in error. On the other hand, if the sum is not equal to one, then the bit is correct.

This process is repeated for all bits as follows:

Initial values →	$\sigma_1=\alpha^{19}=\alpha+\alpha^2$	$\sigma_2=\alpha^{29}=1+\alpha^3$	$\sigma_3=\alpha^2$	
Bit # 31→	$\alpha^{20}=\alpha^2+\alpha^3$	$\alpha^{31}=1$	$\alpha^5=1+\alpha^2$	The sum is not equal to one
Bit # 30→	$\alpha^3+\alpha^4$	α^2	$1+\alpha^2+\alpha^3$	The sum is not equal to one
Bit # 24→	$1+\alpha+\alpha^3$	$1+\alpha^2+\alpha^3+\alpha^4$	$1+\alpha+\alpha^2+\alpha^4$	The sum is equal to one
Bit # 9 →	$1+\alpha+\alpha^2$	$\alpha^2+\alpha^3+\alpha^4$	$\alpha+\alpha^3+\alpha^4$	The sum is equal to one
Bit # 3 →	$1+\alpha+\alpha^4$	$1+\alpha^3+\alpha^4$	$1+\alpha+\alpha^3$	The sum is equal to one

The roots of the error locator polynomial were, therefore, determined as being equal to: x^2, x^8, and x^{23}.

4.2. Chien direct method of implementing the cyclic decoding procedure: [8]

Generally, it is necessary to compute the σ_k functions from the S_i power sums before the final step of error correction can be employed. However, R.T. Chien was able to develop a method to carry out the error-correction process automatically.

His second technique is based mostly on the one described in Section 4.1. and is fully explained in Ref. 8.

4.3. Hybrid methods for finding the roots of a polynomial: [15,16,17]

First, we will discuss a technique to lower the degree of a polynomial, then describe solutions for single degree polynomials, quadratics, cubics, quartics, and quintics, introducing new concepts.

4.3.1. The following theorem, relating roots of a pair of polynomials, is useful in that it enables us to apply fast root determination techniques which will be subsequently described once the degree (≥ 6) of the original polynomial is lowered.

Let $\sigma(x)$ and $\sigma^1(x)$ be polynomials over $GF(2^m)$:

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

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

Let β represent an element of $GF(2^m)$. Then β is a root of $\sigma(x)$ iff β^2 is a root of $\sigma^1(x)$.

Proof: $\sigma^1(\beta^2) = \beta^{2t} + \sigma_1^2 \beta^{2(t-1)} + \sigma_2^2 \beta^{2(t-2)} + \ldots + \sigma_t^2$

$$\sigma^1(\beta^2) = (\beta^t + \sigma_1 \beta^{t-1} + \sigma_2 \beta^{t-2} + \ldots + \sigma_t)^2 = \left[\sigma(\beta)\right]^2$$

Hence, $\sigma^1(\beta^2) = 0$ if $\sigma(\beta) = 0$. On the other hand, if r is a root of $\sigma^1(x)$, then $\sigma^1(r) = 0$.

Let α be a primitive root of $GF(2^m)$ if $r=\alpha^{2k}$, then $r^{1/2}=\alpha^k$.

If $r=\alpha^{2k+1}$, then $r^{1/2}=(\alpha^{2^m-1+2k+1})^{1/2}=\alpha^{2^{m-1}+k}$.

In either case, $r^{1/2}$ exists, and is unique.

Let $(r^{1/2})^t+\sigma_1(r^{1/2})^{t-1}+\sigma_2(r^{1/2})^{t-2}+ \ldots +\sigma_t = C$

Then by squaring:

$$r^t+\sigma_1^2 r^{t-1}+\sigma_2^2 r^{t-2}+ \ldots +\sigma_t^2 = C^2$$

and $C^2 = 0$ by assumption. As $C \neq 0$ would imply $C^2 \neq 0$, we conclude that $C = 0$, and $r^{1/2}$ is a root of $\sigma(x)$.

Defining $\beta = r^{1/2}$, we then have $\beta^2 = r$, and the theorem is proved. Therefore, by determining the GCD between (1) and (2), and using the above-mentioned theorem, it is possible to lower the degree of the original polynomial.

Example: Consider the polynomial over $GF(2^4)$:

$$\sigma(x) = x^7+\alpha\ x^6+\alpha\ x^5+\alpha^9\ x^4+\alpha^{10}\ x^3+\alpha^9\ x^2+\alpha^6\ x+\alpha^{14}$$

We initially compute:

$$\sigma^{(1)}(x) = x^7+\alpha^2\ x^6+\alpha^2\ x^5+\alpha^3\ x^4+\alpha^5\ x^3+\alpha^3\ x^2+\alpha^{12}\ x+\alpha^{13}$$

Then we obtain GCD $\left[\sigma(x),\sigma^{(1)}(x)\right] = x+\alpha^3$.

This implies that α^3 is a root of $\sigma(x)$ and $\sigma^{(1)}(x)$. By the previous theorem, β is a root of $\sigma(x)$ iff β^2 is a root of $\sigma^{(1)}(x)$. We may then deduce that α^3 and α^9 are roots of $\sigma(x)$.

Writing $\sigma(x) = (x+\alpha^3)(x+\alpha^9)\ \sigma^1(x)$

$$\sigma^1(x) = x^5+\alpha^{13}\ x^3+\alpha^4\ x^2+\alpha^5\ x+\alpha^2$$

The other roots of $\sigma^1(x)$ could be detected using other methods discussed in this thesis.

4.3.2. Solution for polynomials of first degree $f(x)=x+k_1$.

In this case, the location of the error is immediately determined as being equal to $k_1 = \alpha^j$.

4.3.3. Solutions for quadratics: [15 , 17]

4.3.3.1. We are interested in quadratics which have neither repeated roots nor zero as a root. These quadratics could be easily transformed from:

$$f(y) = y^2+\sigma_1\ y+\sigma_2$$

Using $y = \sigma_1\ x$

into: $\sigma_1^2 x^2+\sigma_1^2 x+\sigma_2$

or

$$f(x) = x^2+x+k_2,\ k_2= \frac{\sigma_2}{\sigma_1^2} \neq 0 \qquad (1)$$

which is over $GF(2^m)$.

The values of k_2 that allow solutions are obtained from the equation Trace $(k_2) = Tr(k_2) = k_2+k_2^2+k_2^{2^2}+ \ldots +k_2^{2^{m-1}} = 0$ (2)

where the Trace, abbreviated Tr, is defined by:

$$Tr(x) = \sum_{i=0}^{m-1} x^{2^i} \qquad Tr(x) \in GF(2)$$

This property (2) was proven by Berlekamp, Rumsey, and Solomon [17] in a theorem stating:

If $k_2 \in GF(2^m)$, the quadratic equation $x^2+x=k_2$ has solutions in $GF(2^m)$ iff $Tr(k_2) = 0$.

They were able to prove [17] that exactly half of the elements in $GF(2^m)$ have a $Tr(x) = 0$, and exactly half have a $Tr(x) = 1$.

The problem is to find the roots α_1, α_2, of (1) for a given k_2 when $Tr(k_2) = 0$. One way would be to apply the Chien search. Another way would be to use a table-look-up approach. In regards to the second method, we note that there are $\frac{n-1}{2} = \frac{2^m-2}{2} = 2^{m-1}-1$ values of k_2, satisfying the condition that the $Tr(k_2) = 0$.

Example: Let $n=31=2^5-1$

and α a root of the irreducible polynomial $1+x^2+x^5$.

Assuming that the all-zeros polynomial was sent, and two errors occurred at locations $x+x^3$,

then:

$$S_1 = \alpha+\alpha^3 = \alpha^6$$

$$S_3 = \alpha^3+\alpha^9 = \alpha^{30}$$

$$\sigma_1 = \alpha^6$$

$$\sigma_2 = \alpha^4$$

$$k_2 = \frac{\sigma_2}{\sigma_1^2} = \alpha^{23}$$

From the look-up Table 4.3.3.1.1. corresponding to $k_2 = \alpha^{23}$, the two roots are α^{26} and α^{28}. It is not actually necessary to list all the double values because, if we have one solution α_1, then the other one must be $\alpha_2 = \alpha_1+1$.

Proof: $y^2+y+k_2 = (y+\alpha_1)(y+\alpha_2) = y^2+(\alpha_1+\alpha_2)\ y+\alpha_1\alpha_2$

Therefore: $\alpha_1+\alpha_2 = 1$ or $\alpha_2 = \alpha_1+1$

Transforming back, using $x = \sigma_1 y$, we obtain:

$$x_1 = \alpha^6\alpha^{26} = \alpha$$

$$x_2 = \alpha^6\alpha^{28} = \alpha^3$$

which are the positions of the two errors.

#	k_2	1st root	2nd root
1	α	α^3	α^{29}
2	α^2	α^6	α^{27}
3	α^4	α^{12}	α^{23}
4	α^8	α^{15}	α^{24}
5	α^{16}	α^{17}	α^{30}
6	α^7	α^2	α^5
7	α^{14}	α^4	α^{10}
8	α^{28}	α^8	α^{20}
9	α^{25}	α^9	α^{16}
10	α^{19}	α	α^{18}
11	α^{15}	α^{21}	α^{25}
12	α^{30}	α^{11}	α^{19}
13	α^{29}	α^7	α^{22}
14	α^{27}	α^{13}	α^{14}
15	α^{23}	α^{26}	α^{28}

LOOK-UP TABLE

4.3.3.1.1.

The number of values of k_2 satisfying $Tr(k_2)=0$ is equal to:

$$\frac{n-1}{2} = \frac{31-1}{2} = 15$$

Memory size:

In our case, each element requires five bits. (Example α = 00010). Therefore, the look-up table would occupy $15 \times 5 \times 3 = 225$ bits.

If the initial transformation $y = \sigma_1 x$ is ignored, and we still wish to obtain the roots by the look-up table principle, then the number of values of σ_1 and σ_2 that must be stored is equal to $n(\frac{n-1}{2})$. This is illustrated in Table 4.3.3.1.2. where we used the same previously determined values for σ_1 and σ_2, <u>i.e.</u> $\sigma_1=\alpha^6$, $\sigma_2=\alpha^4$.

LOOK-UP TABLE 4.3.3.1.2.

For every σ_1 in the equation $x^2+\sigma_1 x+\sigma_2$, we have corresponding a table of $\frac{n-1}{2}$ elements for σ_2.

1	$\sigma_1 = 1$
2	α
3	α^2
4	α^3
5	α^4
6	α^5
7	α^6
8	α^7
9	α^8
10	α^9
11	α^{10}
12	α^{11}
⋮	⋮
20	α^{19}
21	α^{20}
22	α^{21}
23	α^{22}
24	α^{23}
25	α^{24}
26	α^{25}
27	α^{26}
28	α^{27}
29	α^{28}
30	α^{29}
31	α^{30}

Example

	σ_2	1st root	2nd root
1	α^{27}	1	α^{27}
2	α^4	α	α^3
3	α^{14}	α^2	α^{12}
4	α^{13}	α^4	α^9
5	α^{28}	α^5	α^{23}
6	1	α^7	α^{24}
7	α^{19}	α^8	α^{11}
8	α^{26}	α^{10}	α^{16}
9	α^{10}	α^{13}	α^{28}
10	α^7	α^{21}	α^{30}
11	α^9	α^{14}	α^{26}
12	α^6	α^{15}	α^{22}
13	α^{11}	α^{17}	α^{25}
14	α^{16}	α^{18}	α^{29}
15	α^8	α^{19}	α^{20}

Number of values of σ_1 and $\sigma_2 = \frac{n(n-1)}{2} = 31.15 = 465$

Memory size:

$31\times5+31\times15\times5\times3 = 7130$ bits

Therefore, by simply changing the form of the initial equation, it is possible to reduce the size of the look-up table by 1/n.

4.3.3.2. Now we will show that α_1, α_2 can be determined for a given k_2 by solving a set of simple simultaneous equations. Using the logic according to which $GF(2^m)$ is constructed, we have the given:

$$k_2 = e_0+e_1\ \alpha+e_2\ \alpha^2+ \ldots +e_{m-1}\alpha^{m-1} \qquad (1)$$

where e_i is in GF(2) and α is the primitive element of $GF(2^m)$. Thus, the e_i's are known. Expressing α_1 and α_2, respectively, as:

$$\alpha_1 = a_0+a_1\ \alpha+a_2\ \alpha^2+ \ldots +a_{m-1}\alpha^{m-1} \qquad (2)$$

$$\alpha_2 = b_0+b_1\ \alpha+b_2\ \alpha^2+ \ldots +b_{m-1}\alpha^{m-1} \qquad (3)$$

The problem is to find a_i's and b_i's for the given e_i's of (1).

With reference to $f(x) = x^2+x+k_2$, we have

$$\alpha_1+\alpha_2 = 1 \qquad (4)$$

$$\alpha_1\alpha_2 = k_2 \qquad (5)$$

Using (4) in (2) and (3), we get

$$a_0+b_0 = 1$$

$$a_i+b_i = 0 \quad i=1,2, \ldots m-1 \qquad (6)$$

Multiplying (2) and (3), and using (6), we get

$$\alpha_1\alpha_2 = a_1\ \alpha+(a_2+a_1)\ \alpha^2+a_3\ \alpha^3+(a_4+a_2)\ \alpha^4$$
$$+a_5\ \alpha^5+(a_6+a_3)\ \alpha^6+a_7\ \alpha^7+\ \ldots+$$
$$(a_{m-1}+a_{\frac{m-1}{2}})\ \alpha^{m-1} + a_{\frac{m+1}{2}}\ \alpha^{m+1}$$
$$+ a_{\frac{m+3}{2}}\ \alpha^{m+3} + a_{\frac{m+5}{2}}\ \alpha^{m+5}+\ \ldots$$
$$+a_{m-1}\ \alpha^{2m-2},\ \text{odd } m, \qquad (7a)$$

and

$$\alpha_1\alpha_2 = a_1\ \alpha+(a_2+a_1)\ \alpha^2+a_3\ \alpha^3+(a_4+a_2)\ \alpha^4$$
$$+a_5\ \alpha^5+(a_6+a_3)\ \alpha^6+a_7\ \alpha^7+\ \ldots$$
$$+ a_{m-1}\ \alpha^{m-1} + a_{\frac{m}{2}}\ \alpha^m + a_{\frac{m+2}{2}}\ \alpha^{m+2}$$
$$+a_{\frac{m+4}{2}}\ \alpha^{m+4}+\ \ldots$$
$$+a_{m-1}\ \alpha^{2m-2},\ \text{even } m \qquad (7b)$$

Using the logic according to which $GF(2^m)$ is constructed, (7a) or (7b), as the case may be, can be reduced to the form:

$$\alpha_1\alpha_2 = C_0+C_1\ \alpha+C_2\ \alpha^2+\ \ldots\ +C_{m-1}\ \alpha^{m-1} \qquad (8)$$

where C_i is a linear combination of a_i's.

Comparing (1) and (8), we obtain a set of simple simultaneous equations which can be easily solved for a_i's. Using these a_i's in (2), we get α_1, then $\alpha_2 = 1+\alpha_1$ from (4). This completes the process of finding α_1 and α_2.

Example: Let us consider $f(x) = x^2+x+\alpha^7$ over $GF(2^5)$ constructed according to the logic $\alpha^5 = 1+\alpha^2$. According to (7a), we have:

$$\alpha_1\alpha_2 = a_1\ \alpha+(a_2+a_1)\ \alpha^2+a_3\ \alpha^3+(a_4+a_2)\ \alpha^4+a_3\ \alpha^6+a_4\ \alpha^8$$

Using the logic $\alpha^5 = 1+\alpha^2$, we get, with reference to (8):

$$\alpha_1\alpha_2 = a_1\ \alpha+(a_2+a_1)\ \alpha^2+a_3\ \alpha^3+(a_4+a_2)\ \alpha^4$$

$$+a_3(\alpha+\alpha^3) + a_4(1+\alpha^2+\alpha^3)$$

$$= a_4+(a_1+a_3)\alpha + (a_2+a_1+a_4)\ \alpha^2$$

$$+(a_3+a_3+a_4)\alpha^3 + (a_4+a_2)\ \alpha^4$$

Comparing this with the given $\alpha_1.\alpha_2 = k_2 = \alpha^7 = \alpha^2+\alpha^4$, we obtain:

$a_4=0$ $a_1+a_3=0$ $a_2+a_1+a_4=1$ $a_4=0$ $a_4+a_2=1$

These equations yield: $a_4=0$ $a_2=1$ $a_1=0$ $a_3=0$

This means, with reference to (2) and (3), that:

$$\alpha_1 = 1+\alpha^2 = \alpha^5 \quad \alpha_2 = 1+\alpha_1 = \alpha^2$$

This completes the example.

From the discussion so far, it is clear that the method does not require search in the sense of Chien search, or storage in the sense of table-look-up approach.

4.3.4. Solutions for cubics: [15]

The cubic $f(y) = y^3+\sigma_1 y^2+\sigma_2 y+\sigma_3$ can be transformed using the translation $y = x+\sigma_1$ into:

$$f(x) = (x+\sigma_1)^3+\sigma_1(x+\sigma_1)^2+\sigma_2(x+\sigma_1)+\sigma_3$$

$$f(x) = x^3+(\sigma_1^2+\sigma_2)\, x+\sigma_1\sigma_2+\sigma_3$$

Assuming that $\sigma_1^2+\sigma_2 \neq 0$, we may use another scaling transformation $x = (\sigma_1^2+\sigma_2)^{1/2}z$ to get, over $GF(2^m)$:

$$f(z) = z^3+z+k_3,\ k_3 = \frac{\sigma_1\sigma_2+\sigma_3}{(\sigma_1^2+\sigma_2)^{3/2}} \neq 0 \qquad (1)$$

One way of finding the roots of (1) is to apply the Chien search. Another way would be to store, against each relevant value of k_3, the three roots of $f(z)$ and take a table-look-up approach.

By a relevant value of k_3, we mean one for which $f(z)$ has three roots, as we are interested in cubics which have neither repeated roots, nor zero as a root. Berlekamp [3] proved that the general cubic, over $GF(2^m)$, $f_0+f_1 x+f_2x^2+f_3 x^3$, has no roots or three roots in $GF(2^m)$ iff:

$$\mathrm{Tr}\left[\frac{f_0^2 f_3^2 + f_0 f_2^3 + f_1^3 f_3}{(f_1 f_2 + f_0 f_3)^2}\right] = 0 \qquad (2)$$

By using (2) on f(z), we find that f(z) has three roots or no roots in $GF(2^m)$ if $\mathrm{Tr}(k_3^{-1}) = \mathrm{Tr}(1)$.

Now we will discuss the table-look-up approach.

Suppose that α^i and α^j are two elements of $GF(2^m)$ such that, with reference to (1), $f(\alpha^i) = f(\alpha^j)$, α being the primitive element. This means:

$$\alpha^{3i} + \alpha^{3j} = \alpha^i + \alpha^j$$

or

$$\frac{\alpha^{2i+1}}{\alpha^j} = \frac{\alpha^{2j+1}}{\alpha^i} = A$$

$$\alpha^{2i+1} = \alpha^j \; A \qquad (3)$$

$$\alpha^{2j+1} = \alpha^i \; A \qquad (4)$$

Adding (3) and (4):

$$\alpha^{2i} + \alpha^{2j} = (\alpha^i + \alpha^j) \; A$$

$$(\alpha^i + \alpha^j)^2 = (\alpha^i + \alpha^j) \; A$$

Therefore:

$$\frac{\alpha^{2i+1}}{\alpha^j} = \frac{\alpha^{2j+1}}{\alpha^i} = \alpha^i + \alpha^j$$

or

$$\alpha^{2i}+\alpha^i \cdot \alpha^j+\alpha^{2j+1} = 0 \qquad (5)$$

Using the fact that, for quadratics $(x^2+k_1 x+k_2)$ to have a solution, we must have:

$$Tr\left(\frac{k_2}{k_1^2}\right) = 0$$

(5) implies that:

$$Tr\left(\frac{1+\alpha^{2j}}{\alpha^{2j}}\right) = 0$$

or

$$Tr\left(\alpha^{-j}\right) = Tr(1) \qquad (6)$$

On the other hand, if f(z) has three roots for both $k_3=\alpha^u$ and $k_3=\alpha^v$, then the set of three roots corresponding to α^u and the set of three roots corresponding to α^v are disjoint.

Proof: Assuming that α^i is a root of f(z) for both $k_3=\alpha^u$ and $k_3=\alpha^v$, then:

$$\alpha^{3i}+\alpha^i+\alpha^u = 0$$

and

$$\alpha^{3i}+\alpha^i+\alpha^v = 0$$

Adding the above equations, we obtain $\alpha^u = \alpha^v$.

From this condition, we conclude that the number of k_3's values must be $\leq \frac{n}{3}$.

This leads, in view of (6) to the following:

Theorem: the number N of values of $k_3 \neq 0$ for which $f(z)=z^3+z+k_3$ over $GF(2^m)$ has three roots, is upper-bounded by $\left[\frac{n}{6}\right]$ where n is the block-length and $\left[\frac{n}{6}\right]$ is the integer part of $\frac{n}{6}$.

The amount S of storage required for the table-look-up approach is, at most, $4m\left[\frac{n}{6}\right]$ bits. The number 4 is a result from the fact that we require four columns: three for the three roots, and one for k_3.

If $\frac{n}{6}$ is relatively small, this approach is quite attractive. For example, if m=6, and $n=2^m-1=63$, then N=10 and S=240.

By computing $f(\alpha^i)$, $i=\alpha,\alpha^2,\alpha^3, \ldots$, it is found that for m=3,4,5,6, the number N is exactly equal to $\left[\frac{n}{6}\right]$ and therefore $S=4m\left[\frac{n}{6}\right]$.

Example: Let $n=31=2^5-1$

and α be a root of the irreducible polynomial $1+x^2+x^5$. Assuming that the all-zeros polynomial was sent, and three errors occurred at locations $x^2+x^8+x^{23}$, (as in Example 3.2.2.1.).

$$S_1 = \alpha^{19} \qquad S_3 = 0 \qquad S_5 = \alpha^{24}$$

Error locator polynomials:

$$x^3+\alpha^{19}x^2+\alpha^{29}x+\alpha^2$$

$$\sigma_1=\alpha^{19} \qquad \sigma_2=\alpha^{29} \qquad \sigma_3=\alpha^2$$

$$B = \frac{\sigma_1\sigma_2 +\sigma_3}{(\sigma_1{}^2+\sigma_2)^{3/2}} = \alpha^5$$

	B	1st root	2nd root	3rd root
1	α^9	α^5	α^{14}	α^{21}
2	α^{20}	α^7	α^{18}	α^{26}
3	α^{10}	α^9	α^{13}	α^{19}
4	α^{18}	α^{10}	α^{11}	α^{28}
5	α^5	α^{20}	α^{22}	α^{25}

Memory size S

$4^m \left[\frac{n}{6}\right]$

$4\times5\times5=100$ bits

LOOK-UP TABLE 4.3.2.1.

From the above table, corresponding to α^5, we have α^{20}, α^{22} and α^{25}.

Transforming back:

First: $y = (\sigma_1{}^2+\sigma_2)^{1/2}z$

$y_1 = \alpha^7\alpha^{20} = \alpha^{27}$ $\quad y_2 = \alpha^7\alpha^{22} = \alpha^{29}$ $\quad y_3 = \alpha^7\alpha^{25} = \alpha$

Second: $x=y+\sigma_1$

$x_1 = \alpha^{27}+\alpha^{19} = \alpha^8 \qquad x_2 = \alpha^{29}+\alpha^{19} = \alpha^{23} \qquad x_3 = \alpha+\alpha^{19} = \alpha^2$

Therefore, the positions of the errors are determined.

Going back to the case where $\sigma_1^2 = \sigma_2$, we have, after the initial transformation:

$$f(y) = y^3+\sigma_1^3+\sigma_3$$

The three roots of this f(y) are in $GF(2^m)$ if and only if $\sigma_1^3+\sigma_3 = \alpha^{3j}$ for some j and $n=2^m-1$ is divisible by 3. The latter condition implies that m is even.
Thus, if m is even, $\sigma_1^2=\sigma_2$ and $\sigma_1^3+\sigma_3=\alpha^{3j}$, then the roots of f(y) are straightway:

$$\alpha^j, \ \alpha^{j+n/3}, \ \alpha^{j+2n/3}$$

Now we will prove that $\sigma_1^2=\sigma_2$ implies that $\sigma_1^3+\sigma_3=\alpha^{3j}$ for some j.
With the error polynomial $E(x) = x^i+x^j+x^k$, $\sigma_1^2=\sigma_2$ means that $\sigma_1^2=\alpha^{i+j}+\alpha^{i+k}+\alpha^{j+k}$
or

$$\sigma_1^2= \alpha^i(\sigma_1+\alpha^i)+\alpha^{j+k}$$

$$\alpha^i\sigma_1^2= \alpha^{2i}\sigma_1+\alpha^{3i}+\sigma_3$$

$$\alpha^{3i}+\alpha^{2i}\sigma_1+\alpha^i\sigma_1^2+\sigma_1^3=\sigma_3+\sigma_1^3$$

$$(\sigma_1+\alpha^i)^3=\sigma_3+\sigma_1^3$$

Implying that:

$$\sigma_1^3+\sigma_3 = \alpha^{3u} \quad \text{for some } u.$$

Thus, if m is even and $\sigma_1^2 = \sigma_2$, then the roots of

$$x^3+\sigma_1 x^2+\sigma_2 x+\sigma_3$$

are:

$$x_1 = (\sigma_1^3+\sigma_3)^{1/3}+\sigma_1$$

$$x_2 = (\sigma_1^3+\sigma_3)^{1/3}\alpha^{n/3}+\sigma_1$$

$$x_3 = (\sigma_1^3+\sigma_3)^{1/3}\alpha^{2n/3}+\sigma_1$$

Simulation results: Simulations were performed to determine the percentage of occurrences of the condition $\sigma_1^2=\sigma_2$ in some codes.

a) $n=15=2^4-1$ with α as a root of the irreducible polynomial $1+x+x^4$

From $\binom{15}{3} = 455$ error patterns, 65 were meeting the condition $\sigma_1^2=\sigma_2$. PERCENTAGE = $\frac{65}{455}$ = 14%

b) $n=63=2^6-1$ with α as a root of the irreducible polynomial $1+x+x^6$.

From $\binom{63}{3} = 39711$ error patterns, 1281 were meeting the condition $\sigma_1^2=\sigma_2$. PERCENTAGE = $\frac{1281}{39711}$ = 3.2%

From our results, the condition $\sigma_1^2=\sigma_2$ does not frequently occur.

Example: n=15, irreducible primitive polynomial $1+x+x^4$

Assuming that the all-zeros polynomial was sent, and $x^3+x^5+x^9$ the received polynomial.

Therefore: $S_1 = \alpha^2$ $S_3 = \alpha^2$ $S_5 = \alpha^{10}$

$\sigma_1 = \alpha^2$ $\sigma_2 = \alpha^4$ $\sigma_3 = \alpha^2$

Since $\sigma_1^2 = \sigma_2$ and m is even, the roots are equal to:

$$x_1 = (\sigma_1^3+\sigma_3)^{1/3}+\sigma_1 = \alpha^5$$

$$x_2 = (\sigma_1^3+\sigma_3)^{1/3}\alpha^{n/3}+\sigma_1 = \alpha^3$$

$$x_3 = (\sigma_1^3+\sigma_3)^{1/3}\alpha^{2n/3}+\sigma_1 = \alpha^9$$

4.3.5. Solutions for quartics: [15]

The decoding of binary BCH codes involves finding the roots of what are usually called error-locator polynomials $\sigma(x)$ over $GF(2^m)$. If $\sigma(x)$ of degree e has e irreducible factors over $GF(2^m)$, the roots of $\sigma(x)$ indicate the error-locations. Otherwise, it is a situation of detection only. Thus, it would be advantageous to know whether or not $\sigma(x)$ has e roots in $GF(2^m)$ without having to compute $\sigma(\alpha^i)$, i = 0,1,2, ... where α is the primitive element in $GF(2^m)$. In this context, we will draw attention to a few practically useful properties of quartics over $GF(2^m)$.

The fourth degree polynomial:

$$\sigma(x) = x^4+\sigma_1 x^3+\sigma_2 x^2+\sigma_3 x+\sigma_4$$

with $\sigma_1 \neq 0$ can be transformed by the relation

$$x = (\frac{1}{z} + (\frac{\sigma_3}{\sigma_1})^{1/2})$$

to obtain:

$$z^4 f(z) = \sigma_0{}^1 z^4+\sigma_2{}^1 z^2+\sigma_3{}^1 z+\sigma_4{}^1$$

where

$$\sigma_0{}^1 = \frac{\sigma_3{}^2}{\sigma_1{}^2} + \frac{\sigma_2\sigma_3}{\sigma_1} + \sigma_4$$

$$\sigma_1{}^1 = 0$$

$$\sigma_2{}^1 = \sigma_2 + (\sigma_1\sigma_3)^{1/2}$$

$$\sigma_3{}^1 = \sigma_1$$

$$\sigma_4{}^1 = 1$$

It is therefore possible to express a quartic into the form:

$$f(y) = y^4+k_2 y^2+k_3 y+k_4, \quad k_4 \neq 0 \qquad (1)$$

which is over $GF(2^m)$. Here we assume $k_4 \neq 0$, since $k_4 = 0$ reduces the problem to one of cubics.

Now, we will discuss (1). This quartic has four distinct roots in $GF(2^m)$, if it can be expressed over $GF(2^m)$ in the form:

$$f(y) = (y^2+ay+b_1)(y^2+ay+b_2) \qquad (2)$$

for 2 values of a.

Comparing (1) and (2), we obtain:

$$a^2+b_1+b_2 = k_2 \qquad (3a)$$

$$a(b_1+b_2) = k_3 \qquad (3b)$$

$$b_1 b_2 = k_4 \qquad (3c)$$

Combining (3a) and (3b), we obtain:

$$a^3+ak_2+k_3 = o \qquad (4)$$

and combining (3b) and (3c), we obtain:

$$b_1^2+\frac{k_3}{a}\, b_1+k_4 = 0 \qquad (5)$$

As mentioned previously, the general cubic, over $GF(2^m)$,

$$f_0+f_1x+f_2x^2+f_3x^3$$

has no roots or three roots in $GF(2^m)$ iff:

$$Tr\left[\frac{f_0^2 f_3^2+f_0 f_2^3+f_1^3 f_3}{(f_1 f_2+f_0 f_3)^2}\right] = 0$$

Hence, (4) has three roots or no roots in $GF(2^m)$ if

$$Tr\left(1+\frac{k_2^3}{k_3^2}\right) = 0 \qquad \text{or} \qquad Tr\left(\frac{k_2^3}{k_3^2}\right) = Tr(1)$$

where Tr indicates the trace.

Thus, we have the following:

Theorem 1: If $Tr\left(\frac{k_2^3}{k_3^2}\right) \neq Tr(1)$, then the quartic (1) over $GF(2^m)$ cannot be factored into four factors.

If m is odd, $Tr(1) = 1$. If $k_2 = 0$, then

$$Tr\left(\frac{k_2^3}{k_3^2}\right) = 0$$

Therefore, theorem 1 yields the following:

Corollary 1: If m is odd, the quartic $y^4+k_3y+k_4$, $k_4 \neq 0$, over $GF(2^m)$ cannot be factored into four factors.

The validity of Corollary 1 can also be seen from (4). If $k_2 = 0$, then $a^3=k_3$, indicating that, for odd m, (4) has only one solution in $GF(2^m)$, whereas we require two solutions. We note that if (4) has two solutions, then this implies its having three solutions.

If m is even, $k_2=0$ and k_3 cannot be expressed in the form of α^{3j}, then (4) has no solution at all.

This means the following:

Theorem 2: If m is even and k_3 cannot be expressed in the form α^{3j}, the quartic $x^4+k_3\,x+k_4$ cannot be factored into two quadratics.

With m being even and $k_2=0$, suppose k_3 can be expressed as α^{3j}.

Then the solutions of (4) are:

$$a = k_3^{1/3},\ (k_3\alpha^n)^{1/3},\ (k_3\alpha^{2n})^{1/3}$$

Now (5) has two solutions if:

$$Tr\left(\frac{k_4 a^2}{k_3^2}\right) = 0$$

Thus, we have the following:

Theorem 3: With m even and $k_3=\alpha^{3j}$, the quartic $y^4+k_3y+k_4$ over $GF(2^m)$ can be factored into four factors if:

$$Tr\left(\frac{k_4}{\alpha^{4j}}\right) = Tr\left(\frac{k_4}{\alpha^{4j}}\alpha^{2n/3}\right) = 0$$

Theorems 1,2, and 3 are practically useful because they allow us, in some cases, to determine if the quartic f(y) has four roots in $GF(2^m)$ without having to compute $f(\alpha^i)$, i=0,1,2, ...

4.3.5.1. Table-look-up approach:

If we wish to use the table-look-up approach for polynomials of the form:

$$f(x) = x^a+x^{a-2}+k_3x^{a-3}+k_4x^{a-4}+ \ldots +k_a, \quad a\geq 4$$

then the number of combinations of $k_3,k_4, \ldots k_a$ to

be stored is $\leq \frac{(2^m)^{a-2}}{a}$.

Proof: For a given combination of $k_4, k_5, \ldots k_a$, the set of 'a' roots corresponding to k_3 and the set of 'a' roots corresponding to $k_3^1 \neq k_3$ are disjoint. Therefore, the number of values of k_3 is $\leq \left(\frac{2^m}{a}\right)$

On the other hand, the number of choices of $k_4, k_5, \ldots k_a$ is $\leq (2^m)^{a-3}$.

Thus, the number of combinations of $k_3, k_4, \ldots k_a$, is

$$\leq \frac{2^m(2^m)^{a-3}}{a} = \frac{(2^m)^{a-2}}{a}$$

4.3.5.2. Example: $n=2^m-1=2^5-1=31$; $GF(2^5)$ of Table 2.1.

Assuming that the all-zeros polynomial was sent, and errors occurred at locations $x+x^5+x^{17}+x^{23}$.

As usual, the syndromes are determined first. Next, the error locator polynomial is obtained.

a) $S_1 = \alpha^{16}$ $S_3 = \alpha^{23}$ $S_5 = \alpha^{19}$ $S_7 = \alpha^{25}$

b) $x^4+\alpha^{16}x^3+\alpha^{12}x^2+\alpha^6x+\alpha^{15}$

Using the first transformation $x = \frac{1}{z} + (\frac{\alpha^6}{\alpha^{16}})^{1/2} = \frac{1}{z} + \alpha^{26}$

we obtain: $f(y) = y^4+\alpha^{16}y^2+\alpha^3y+\alpha^{18}$

To check that, possibly, we have four roots, the condition

$$Tr\left(\frac{k_2^3}{k_3^2}\right) = Tr(1)$$

must be satisfied.

In our case, $Tr(\alpha^{11}) = 1$ and the condition $Tr\left(\frac{k_2^3}{k_3^2}\right) = Tr(1)$ is fulfilled.

Transforming the cubic $a^3+a\ \alpha^{16}+\alpha^3 = 0$

into $z^3+z+\alpha^{10} = 0$ using $a = \alpha^8 z$

we can obtain the roots of the cubic from Table 4.3.2.1., i.e. corresponding to α^{10}, we have:

$$\alpha^9 \qquad \alpha^{13} \qquad \alpha^{19}$$

Transforming back:

$$a_1 = \alpha^9\ \alpha^8 = \alpha^{17} \qquad a_2 = \alpha^{13}\ \alpha^8 = \alpha^{21} \qquad \text{and}$$

$$a_3 = \alpha^{19}\ \alpha^8 = \alpha^{27}$$

Now, any one of the obtained values of a (in our case, $a_1 = \alpha^{17}$) could be utilized in the equation:

$$b_1^2 + \frac{k_3}{a} b_1 + k_4 = 0$$

<u>i.e.</u>

$$b_1^2 + \alpha^{17} b_1 + \alpha^{18} = 0$$

Using Table 4.3.3.1.1., we have from $\frac{\alpha^{18}}{\alpha^3} = \alpha^{15}$, two roots α^{21} and α^{25}.

Transforming back:

$$b_1 = \alpha^{17}\alpha^{21} = \alpha^7 \qquad b_2 = \alpha^{17}\alpha^{25} = \alpha^{11}$$

Using the equation:

$$f(y) = (y^2 + ay + b_1)(y^2 + ay + b_2)$$

and the quadratic look-up Table 4.3.3.1.1., we obtain:

$y^2 + ay + b_1$	$y^2 + ay + b_2$
$y^2 + \alpha^{17} y + \alpha^7$	$y^2 + \alpha^{17} y + \alpha^{11}$
$y_1 = \alpha^{29}$	$y_3 = \alpha$
$y_2 = \alpha^9$	$y_4 = \alpha^{10}$

Transforming to obtain the error locations

$$x_1 = \frac{1}{y_1} + \alpha^{26} = \alpha^{17} \qquad x_2 = \frac{1}{y_2} + \alpha^{26} = \alpha$$

$$x_3 = \frac{1}{y_3} + \alpha^{26} = \alpha^5 \qquad x_4 = \frac{1}{y_4} + \alpha^{26} = \alpha^{23}$$

The choice of any value of a at the beginning would have provided the same results.

4.3.5.3. A table look-up approach could also be adapted to determine the roots. To illustrate this technique, we will deal with the same previous example.

After the first transformation, we obtained:

$$y^4+\alpha^{16}\ \ y^2+\alpha^3\ \ y+\alpha^{18}$$

Transforming again, using $y = z\alpha^8$, we get:

$$z^4+z^2+\alpha^{10}\ \ z+\alpha^{17}$$

From the table-look-up 4.3.5.2.1., the roots for $k_3=\alpha^{10}$ and $k_4=\alpha^{17}$ are:

$$\alpha,\ \alpha^2,\ \alpha^{21},\ \alpha^{24}$$

Transforming back:

$$y_1 = \alpha^8\ \alpha = \alpha^9 \qquad y_2 = \alpha^8\alpha^2 = \alpha^{10}$$

$$y_3 = \alpha^8\alpha^{21} = \alpha^{29} \qquad y_4 = \alpha^8\alpha^{24} = \alpha$$

Then, applying the transformation $x = \frac{1}{y} + \alpha^{26}$, we secure the locations of the errors:

$$\alpha,\ \alpha^5,\ \alpha^{17},\ \alpha^{23}$$

Making use of a table-look-up, because of its simplicity, is a technique that should be considered when determining the roots of a polynomial.

Table 4.3.5.2.2. and Table 4.3.5.2.3. were developed for $n=2^4-1=15$ and $n=2^6-1=63$, respectively.

Look-up Table 4.3.5.2.1.

$f(z) = z^4+z^2+k_3\ z+k_4$

$n=2^5-1=31$

Irreducible polynomial: $1+x^2+x^5$

k_3	k_4	Roots of f(z)			
α^5	α^5	1	α^7	α^8	α^{21}
α^5	α^{24}	α	α^{12}	α^{16}	α^{26}
α^5	α^{29}	α^2	α^3	α^{10}	α^{14}
α^5	α^{20}	α^4	α^5	α^{13}	α^{29}
α^5	α^{26}	α^6	α^{17}	α^{19}	α^{15}
α^5	α^{16}	α^9	α^{18}	α^{23}	α^{28}
α^5	α^{30}	α^{11}	α^{24}	α^{27}	α^{30}
α^9	α^9	1	α^2	α^{13}	α^{25}
α^9	α^5	α	α^9	α^{11}	α^{15}
α^9	α^6	α^3	α^4	α^8	α^{22}
α^9	α^{23}	α^6	α^{23}	α^{26}	α^{30}
α^9	α^4	α^7	α^{10}	α^{20}	α^{29}
α^9	α^{22}	α^{12}	α^{17}	α^{27}	α^{28}
α^9	α^{15}	α^{16}	α^{18}	α^{19}	α^{24}
α^{10}	α^{10}	1	α^{14}	α^{16}	α^{11}
α^{10}	α^{17}	α	α^2	α^{21}	α^{24}
α^{10}	α^{21}	α^3	α^7	α^{12}	α^{30}
α^{10}	α^{27}	α^4	α^6	α^{20}	α^{28}
α^{10}	α	α^5	α^{18}	α^{25}	α^{15}
α^{10}	α^{29}	α^{17}	α^{22}	α^{23}	α^{29}
α^{10}	α^9	α^8	α^{10}	α^{26}	α^{27}
α^{18}	α^{18}	1	α^4	α^{19}	α^{26}
α^{18}	α^{30}	α	α^5	α^7	α^{17}
α^{18}	α^{10}	α^2	α^{18}	α^{22}	α^{30}
α^{18}	α^{13}	α^3	α^{23}	α^{24}	α^{25}
α^{18}	α^{12}	α^6	α^8	α^{13}	α^{16}
α^{18}	α^8	α^9	α^{14}	α^{20}	α^{27}
α^{18}	α^{15}	α^{12}	α^{15}	α^{21}	α^{29}

$n=2^5=31$

k_3	k_4		Roots of f(z)			
α^{20}	α^{20}		1	α	α^{22}	α^{28}
α^{20}	α^{3}		α^{2}	α^{4}	α^{11}	α^{17}
α^{20}	α^{27}		α^{3}	α^{13}	α^{15}	α^{27}
α^{20}	α^{2}		α^{5}	α^{10}	α^{19}	α^{30}
α^{20}	α^{11}		α^{6}	α^{14}	α^{24}	α^{29}
α^{20}	α^{23}		α^{8}	α^{9}	α^{12}	α^{25}
α^{20}	α^{18}		α^{16}	α^{20}	α^{21}	α^{23}

Look-up Table 4.3.5.2.2.

$f(z) = z^4+z^2+k_3\, z+k_4$

$n=2^4-1=15$

Irreducible polynomial: $1+x+x^4$

k_3	k_4		Roots of f(z)			
α^{5}	α^{5}		1	α^{3}	α^{5}	α^{12}
α^{5}	α^{7}		α	α^{6}	α^{7}	α^{8}
α^{5}	α^{13}		α^{2}	α^{4}	α^{9}	α^{13}
α^{10}	α^{10}		1	α^{6}	α^{9}	α^{10}
α^{10}	α^{14}		α	α^{2}	α^{12}	α^{14}
α^{10}	α^{11}		α^{3}	α^{4}	α^{8}	α^{11}

Look-up Table 4.3.5.2.3.

$f(z) = z^4+z^2+k_3\ z+k_4$

$n=2^6-1=63$

Irreducible polynomial: $1+x+x^6$

	k_3	k_4		Roots of f(z)			
1	1	1		1	α^9	α^{18}	α^{36}
2	1	α^{36}		α	α^8	α^{38}	α^{52}
3	1	α^9		α^2	α^{13}	α^{16}	α^{41}
4	1	α^{27}		α^3	α^7	α^{24}	α^{56}
5	1	α^{18}		α^4	α^{19}	α^{26}	α^{32}
6	1	α^{37}		α^5	α^{43}	α^{55}	α^{60}
7	1	α^{54}		α^6	α^{14}	α^{48}	α^{49}
8	1	α^{11}		α^{10}	α^{23}	α^{47}	α^{57}
9	1	α^{21}		α^{11}	α^{42}	α^{44}	α^{50}
10	1	α^{45}		α^{12}	α^{28}	α^{33}	α^{35}
11	1	α^{25}		α^{15}	α^{17}	α^{58}	α^{61}
12	1	α^{22}		α^{20}	α^{31}	α^{46}	α^{51}
13	1	α^{42}		α^{21}	α^{22}	α^{25}	α^{37}
14	1	α^{44}		α^{29}	α^{39}	α^{40}	α^{62}
15	1	α^{50}		α^{30}	α^{34}	α^{53}	α^{59}
1	α^9	α^9		1	α^5	α^{27}	α^{40}
2	α^9	α^{34}		α	α^{11}	α^{37}	α^{48}
3	α^9	α^{58}		α^2	α^{31}	α^{35}	α^{53}
4	α^9	α^6		α^3	α^{17}	α^{23}	α^{26}
5	α^9	α^{52}		α^4	α^{56}	α^{57}	α^{61}
6	α^9	α^{20}		α^6	α^8	α^{25}	α^{44}
7	α^9	α^{38}		α^7	α^{15}	α^{47}	α^{32}
8	α^9	α^{36}		α^9	α^{39}	α^{54}	α^{60}
9	α^9	α^{48}		α^{10}	α^{19}	α^{24}	α^{58}
10	α^9	α^{47}		α^{12}	α^{13}	α^{34}	α^{51}

$n=2^6-1=63$

	k_3	k_4		Roots of f(z)			
11	α^9	α^{53}		α^{14}	α^{22}	α^{42}	α^{38}
12	α^9	α^{23}		α^{16}	α^{28}	α^{46}	α^{59}
13	α^9	α^{61}		α^{20}	α^{30}	α^{33}	α^{41}
14	α^9	α^{46}		α^{21}	α^{49}	α^{50}	α^{52}
15	α^9	α^{27}		α^{29}	α^{36}	α^{43}	α^{45}
1	α^{18}	α^{18}		1	α^{10}	α^{17}	α^{54}
2	α^{18}	α^{13}		α	α^{14}	α^{30}	α^{31}
3	α^{18}	α^{5}		α^{2}	α^{11}	α^{22}	α^{33}
4	α^{18}	α^{59}		α^{3}	α^{19}	α^{40}	α^{60}
5	α^{18}	α^{53}		α^{4}	α^{7}	α^{43}	α^{62}
6	α^{18}	α^{31}		α^{5}	α^{24}	α^{26}	α^{39}
7	α^{18}	α^{12}		α^{6}	α^{34}	α^{46}	α^{52}
8	α^{18}	α^{41}		α^{8}	α^{49}	α^{51}	α^{59}
9	α^{18}	α^{54}		α^{9}	α^{23}	α^{27}	α^{58}
10	α^{18}	α^{40}		α^{12}	α^{16}	α^{25}	α^{50}
11	α^{18}	α^{43}		α^{13}	α^{21}	α^{28}	α^{44}
12	α^{18}	α^{9}		α^{15}	α^{18}	α^{45}	α^{57}
13	α^{18}	α^{33}		α^{20}	α^{38}	α^{48}	α^{53}
14	α^{18}	α^{46}		α^{29}	α^{32}	α^{55}	α^{56}
15	α^{18}	α^{29}		α^{35}	α^{37}	α^{41}	α^{42}
1	α^{31}	α^{31}		1	α^{19}	α^{22}	α^{53}
2	α^{31}	α^{41}		α	α^{39}	α^{41}	α^{23}
3	α^{31}	α^{27}		α^{2}	α^{10}	α^{38}	α^{40}
4	α^{31}	α^{25}		α^{3}	α^{11}	α^{20}	α^{54}
5	α^{31}	α^{14}		α^{4}	α^{18}	α^{21}	α^{34}

$n=2^6-1=63$

	k_3	k_4		Roots of f(z)			
6	α^{31}	α^{49}		α^{5}	α^{14}	α^{35}	α^{58}
7	α^{31}	α^{16}		α^{7}	α^{44}	α^{45}	α^{46}
8	α^{31}	α^{37}		α^{8}	α^{16}	α^{29}	α^{47}
9	α^{31}	α^{39}		α^{9}	α^{26}	α^{30}	α^{37}
10	α^{31}	α^{58}		α^{13}	α^{52}	α^{57}	α^{62}
11	α^{31}	α^{29}		α^{15}	α^{28}	α^{43}	α^{6}
12	α^{31}	α^{32}		α^{17}	α^{33}	α^{48}	α^{60}
13	α^{31}	α^{61}		α^{24}	α^{27}	α^{31}	α^{42}
14	α^{31}	α^{26}		α^{25}	α^{36}	α^{59}	α^{32}
15	α^{31}	α^{51}		α^{49}	α^{55}	α^{61}	α^{12}
1	α^{36}	α^{36}		1	α^{20}	α^{34}	α^{45}
2	α^{36}	α^{29}		α	α^{47}	α^{49}	α^{58}
3	α^{36}	α^{26}		α^{2}	α^{28}	α^{60}	α^{62}
4	α^{36}	α^{10}		α^{3}	α^{4}	α^{22}	α^{44}
5	α^{36}	α^{55}		α^{6}	α^{17}	α^{38}	α^{57}
6	α^{36}	α^{58}		α^{7}	α^{11}	α^{19}	α^{21}
7	α^{36}	α^{43}		α^{8}	α^{14}	α^{23}	α^{61}
8	α^{36}	α^{62}		α^{10}	α^{15}	α^{48}	α^{52}
9	α^{36}	α^{24}		α^{12}	α^{29}	α^{41}	α^{5}
10	α^{36}	α^{3}		α^{13}	α^{33}	α^{40}	α^{43}
11	α^{36}	α^{19}		α^{16}	α^{35}	α^{39}	α^{55}
12	α^{36}	α^{45}		α^{18}	α^{46}	α^{53}	α^{54}
13	α^{36}	α^{23}		α^{25}	α^{26}	α^{42}	α^{56}
14	α^{36}	α^{18}		α^{27}	α^{30}	α^{36}	α^{51}
15	α^{36}	α^{17}		α^{32}	α^{37}	α^{50}	α^{24}

$n=2^6-1=63$

	k_3	k_4		Roots of f(z)			
1	α^{47}	α^{47}		1	α^{11}	α^{41}	α^{58}
2	α^{47}	α^{45}		α	α^{5}	α^{19}	α^{20}
3	α^{47}	α^{7}		α^{2}	α^{9}	α^{17}	α^{42}
4	α^{47}	α^{46}		α^{3}	α^{14}	α^{39}	α^{53}
5	α^{47}	α^{50}		α^{4}	α^{8}	α^{46}	α^{55}
6	α^{47}	α^{57}		α^{6}	α^{56}	α^{59}	α^{62}
7	α^{47}	α^{56}		α^{7}	α^{29}	α^{34}	α^{49}
8	α^{47}	α^{44}		α^{10}	α^{27}	α^{33}	α^{37}
9	α^{47}	α^{62}		α^{12}	α^{21}	α^{45}	α^{47}
10	α^{47}	α^{51}		α^{13}	α^{15}	α^{36}	α^{50}
11	α^{47}	α^{13}		α^{16}	α^{18}	α^{44}	α^{61}
12	α^{47}	α^{8}		α^{22}	α^{23}	α^{35}	α^{54}
13	α^{47}	α^{16}		α^{24}	α^{30}	α^{40}	α^{48}
14	α^{47}	α^{29}		α^{26}	α^{31}	α^{38}	α^{60}
15	α^{47}	α^{52}		α^{32}	α^{43}	α^{51}	α^{52}
1	α^{55}	α^{55}		1	α^{29}	α^{37}	α^{52}
2	α^{55}	α^{35}		α	α^{21}	α^{36}	α^{40}
3	α^{55}	α^{25}		α^{2}	α^{4}	α^{23}	α^{59}
4	α^{55}	α^{60}		α^{3}	α^{28}	α^{31}	α^{61}
5	α^{55}	α^{22}		α^{5}	α^{48}	α^{50}	α^{45}
6	α^{55}	α^{31}		α^{6}	α^{42}	α^{54}	α^{55}
7	α^{55}	α^{23}		α^{7}	α^{33}	α^{51}	α^{58}
8	α^{55}	α^{38}		α^{8}	α^{9}	α^{22}	α^{62}
9	α^{55}	α^{54}		α^{10}	α^{32}	α^{34}	α^{41}
10	α^{55}	α^{4}		α^{11}	α^{27}	α^{43}	α^{49}

$n=2^6-1=63$

	k_3	k_4		Roots of f(z)			
11	α^{55}	α^{8}		α^{12}	α^{15}	α^{24}	α^{20}
12	α^{55}	α^{46}		α^{13}	α^{19}	α^{30}	α^{47}
13	α^{55}	α^{26}		α^{16}	α^{26}	α^{53}	α^{57}
14	α^{55}	α^{28}		α^{17}	α^{35}	α^{46}	α^{56}
15	α^{55}	α^{57}		α^{18}	α^{25}	α^{38}	α^{39}
1	α^{59}	α^{59}		1	α^{46}	α^{50}	α^{26}
2	α^{59}	α^{44}		α	α^{2}	α^{43}	α^{61}
3	α^{59}	α^{47}		α^{3}	α^{27}	α^{59}	α^{21}
4	α^{59}	α^{19}		α^{4}	α^{11}	α^{31}	α^{36}
5	α^{59}	α^{27}		α^{5}	α^{16}	α^{52}	α^{17}
6	α^{59}	α^{13}		α^{8}	α^{13}	α^{58}	α^{60}
7	α^{59}	α^{60}		α^{9}	α^{51}	α^{19}	α^{44}
8	α^{59}	α^{4}		α^{10}	α^{39}	α^{6}	α^{12}
9	α^{59}	α^{30}		α^{14}	α^{62}	α^{47}	α^{33}
10	α^{59}	α^{49}		α^{18}	α^{20}	α^{32}	α^{42}
11	α^{59}	α^{11}		α^{24}	α^{25}	α^{34}	α^{54}
12	α^{59}	α^{14}		α^{28}	α^{40}	α^{49}	α^{23}
13	α^{59}	α^{43}		α^{29}	α^{35}	α^{48}	α^{57}
14	α^{59}	α^{2}		α^{37}	α^{45}	α^{53}	α^{56}
15	α^{59}	α^{23}		α^{38}	α^{41}	α^{55}	α^{15}
1	α^{61}	α^{61}		1	α^{13}	α^{23}	α^{25}
2	α^{61}	α^{22}		α	α^{62}	α^{32}	α^{53}
3	α^{61}	α^{41}		α^{2}	α^{18}	α^{37}	α^{47}
4	α^{61}	α^{2}		α^{3}	α^{5}	α^{6}	α^{51}
5	α^{61}	α^{38}		α^{4}	α^{29}	α^{30}	α^{38}

$n=2^6-1=63$

	k_3	k_4		Roots of f(z)			
6	α^{61}	α^{15}		α^{7}	α^{31}	α^{48}	α^{55}
7	α^{61}	α^{45}		α^{8}	α^{26}	α^{34}	α^{40}
8	α^{61}	α^{56}		α^{9}	α^{10}	α^{16}	α^{21}
9	α^{61}	α^{37}		α^{12}	α^{17}	α^{27}	α^{44}
10	α^{61}	α^{7}		α^{14}	α^{20}	α^{43}	α^{56}
11	α^{61}	α^{43}		α^{19}	α^{39}	α^{52}	α^{59}
12	α^{61}	α^{53}		α^{24}	α^{46}	α^{49}	α^{60}
13	α^{61}	α		α^{28}	α^{50}	α^{54}	α^{58}
14	α^{61}	α^{55}		α^{33}	α^{42}	α^{45}	α^{61}
15	α^{61}	α^{30}		α^{36}	α^{57}	α^{22}	α^{41}
1	α^{62}	α^{62}		1	α^{38}	α^{44}	α^{43}
2	α^{62}	α^{52}		α	α^{9}	α^{50}	α^{55}
3	α^{62}	α^{19}		α^{2}	α^{15}	α^{19}	α^{46}
4	α^{62}	α		α^{3}	α^{33}	α^{34}	α^{57}
5	α^{62}	α^{54}		α^{4}	α^{13}	α^{17}	α^{20}
6	α^{62}	α^{28}		α^{5}	α^{8}	α^{36}	α^{42}
7	α^{62}	α^{50}		α^{6}	α^{22}	α^{40}	α^{45}
8	α^{62}	α^{35}		α^{7}	α^{10}	α^{28}	α^{53}
9	α^{62}	α^{15}		α^{11}	α^{18}	α^{52}	α^{60}
10	α^{62}	α^{58}		α^{12}	α^{23}	α^{30}	α^{56}
11	α^{62}	α^{32}		α^{14}	α^{25}	α^{27}	α^{29}
12	α^{62}	α^{11}		α^{16}	α^{31}	α^{32}	α^{58}
13	α^{62}	α^{59}		α^{21}	α^{48}	α^{54}	α^{62}
14	α^{62}	α^{39}		α^{24}	α^{35}	α^{47}	α^{59}
15	α^{62}	α^{53}		α^{26}	α^{41}	α^{51}	α^{61}

4.3.6. Solutions for quintics: [15]

This case was developed by R.T. Chien, B.D. Cunningham, and I.B. Oldham [15], and will be described through the following example; where:

$$n=2^5-1 = 31$$

and the Galois field $GF(2^5)$ is constructed according to the logic $\alpha^5=1+\alpha^2$.

Assuming that the all-zeros polynomial was sent and the following sequence was received:

$$x+x^5+x^7+x^{15}+x^{25}$$

which represents the error locations. The first step, as usual, consists of finding the syndromes:

$$S_1 = \alpha^{22} \qquad S_7 = \alpha^5$$

$$S_3 = \alpha^{12} \qquad S_9 = \alpha^{23}$$

$$S_5 = \alpha^{30}$$

Using the Berlekamp or Peterson method, the error locator polynomial is:

$$\sigma(x) = x^5+\alpha^{22}x^4+\alpha^{28}x^3+\alpha^{21}x^2+\alpha^{28}x+\alpha^{22}$$

a) Using the translation $x = y+\frac{\alpha^{21}}{\alpha^{28}} = y+\alpha^{24}$, we

eliminate the quadratic term:

$$y^5+\alpha^{27}y^4+\alpha^{28}y^3+\alpha^{18}y+\alpha^{15}$$

b) Then, inverting, we eliminate the cubic term:

$$u^5+\alpha^3u^4+\alpha^{13}u^2+\alpha^{12}u+\alpha^{16}$$

c) Using the translation $u = z+\alpha^3$, we eliminate the quartic term:

$$z^5+\alpha^{13}z^2+\alpha$$

It is always possible to reduce a quintic of the form:

$\sigma(x) = x^5+\sigma_1x^4+\sigma_2x^3+\sigma_3x^2+\sigma_4x^1+\sigma_5$ into

$F(y) = y^5+Gy^3+Hy^2+Iy+J$

with G or H equal to zero (Ref. 3, p. 255)

In our case, we will perform a final inversion to obtain:

$$f(z) = z^5+\alpha^{12}z^3+\alpha^{30}$$

Assuming f(z) to have five distinct roots, we may write:

$$f(z) = (z^3+az^2+bz+c)(z^2+az+d)$$

where

$$a^2+b+d = \alpha^{12} = G \qquad (1)$$

$$ad+ab+c = 0 \qquad (2)$$

$$bd+ac = 0 \qquad (3)$$

$$cd = \alpha^{30} = J \qquad (4)$$

If C_1 is an entry in the quadratic look-up table 4.3.3.1.1., we may deduce:

$$\frac{d}{a^2} = C_1 \quad \text{or} \quad c = \frac{J}{a^2 C_1}$$

From (1) $\quad b = G+a^2(1+C_1)$

From (3) $\quad a^5(1+C_1)+a^3G+\frac{J}{C_1^2} = 0 \qquad (5)$

From (2) $\quad a^5+a^3G+\frac{J}{C_1} = 0 \qquad (6)$

(5) and (6) imply that $a^3 = \frac{J\beta}{GC_1^3}$

where $\beta = 1+C_1+C_1^2$

In our case, choosing C_1 from Table 4.3.3.1.1. as being equal to α^{23}, we obtain:

$$a^3 = \frac{\alpha^{30}\alpha^{10}}{\alpha^{12}\alpha^{23\times 3}} = \alpha^{21} \quad \text{or} \quad a = \alpha^7$$

Then, $d = \alpha^6$, $c = \alpha^{24}$, $b = \alpha^{25}$

Therefore, we obtain:

$$(z^3+\alpha^7\ z^2+\alpha^{25}\ z+\alpha^{24})(z^2+\alpha^7\ z+\alpha^6)$$

Using both the quadratic look-up Table 4.3.3.1.1., and the cubic Table 4.3.2.1., five roots are obtained:

$$\alpha^2,\ \alpha^4,\ \alpha^5,\ \alpha^{21},\ \alpha^{29}$$

Transforming back, we secure the location of the errors as:

$$x,\ x^5,\ x^7,\ x^{15},\ x^{25}$$

A certain amount of searching in the quadratic look-up table is necessary to determine a 'C_1' that will provide an 'a', which is a common solution to both (5) and (6). Because of the transformations and the search, it might be worthwhile, in the quintic case, to use the theorem described in Section 4.3.1. lowering initially the degree of the polynomial under consideration.

By adapting a Table look-up (4.3.6.1.), it is possible to immediately determine the roots of the quintics

$$z^5 + \alpha^{13}\, z^2 + \alpha$$

as being equal to

$$\alpha^2,\ \alpha^{10},\ \alpha^{26},\ \alpha^{27},\ \alpha^{29}$$

Transforming back:

a) $u = z+\alpha^3$

$$\alpha^{20},\ \alpha^{25},\ \alpha^{15},\ \alpha^{18},\ 1$$

b) Inverting: $\alpha^{11},\ \alpha^6,\ \alpha^{16},\ \alpha^{13},\ 1$

c) $x = y+\alpha^{24}$; the locations of the errors are determined and equal to:

$$\alpha^{25},\ \alpha^7,\ \alpha^5,\ \alpha,\ \alpha^{15}$$

Look-up Table 4.3.6.1.

$$f(z) = z^5+k_3z^2+k_5$$

$$n=2^5-1=31$$

Irreducible polynomial: $1+x^2+x^5$

k_3	k_5	Roots of f(z)				
1	1	α	α^2	α^4	α^8	α^{16}
α	α^{12}	α^6	α^{22}	α^{23}	α^{25}	α^{29}
α^2	α^{24}	α^{12}	α^{13}	α^{15}	α^{19}	α^{27}
α^3	α^5	α^2	α^3	α^5	α^9	α^{17}
α^4	α^{17}	α^7	α^{23}	α^{24}	α^{26}	α^{30}
α^5	α^{29}	α^{28}	α^{13}	α^{14}	α^{16}	α^{20}
α^6	α^{10}	α^3	α^4	α^6	α^{10}	α^{18}
α^7	α^{22}	1	α^8	α^{24}	α^{25}	α^{27}
α^8	α^3	α^{14}	α^{15}	α^{17}	α^{21}	α^{29}
α^9	α^{15}	α^4	α^5	α^7	α^{11}	α^{19}
α^{10}	α^{27}	α	α^9	α^{25}	α^{26}	α^{28}
α^{11}	α^8	α^{15}	α^{16}	α^{18}	α^{22}	α^{30}
α^{12}	α^{20}	α^5	α^6	α^8	α^{12}	α^{20}
α^{13}	α	α^2	α^{10}	α^{26}	α^{27}	α^{29}
α^{14}	α^{13}	1	α^{16}	α^{17}	α^{19}	α^{23}
α^{15}	α^{25}	α^6	α^7	α^9	α^{13}	α^{21}
α^{16}	α^6	α^3	α^{11}	α^{27}	α^{28}	α^{30}
α^{17}	α^{18}	α	α^{17}	α^{18}	α^{20}	α^{24}
α^{18}	α^{30}	α^7	α^8	α^{10}	α^{14}	α^{22}
α^{19}	α^{11}	1	α^4	α^{12}	α^{28}	α^{29}
α^{20}	α^{23}	α^2	α^{18}	α^{19}	α^{21}	α^{25}

$$f(z) = z^5+k_3z^2+k_5$$

$$n=2^5-1=31$$

k_3	k_5	Roots of f(z)				
α^{21}	α^{4}	α^{8}	α^{9}	α^{11}	α^{15}	α^{23}
α^{22}	α^{16}	α	α^{5}	α^{13}	α^{29}	α^{30}
α^{23}	α^{28}	α^{3}	α^{19}	α^{20}	α^{22}	α^{26}
α^{24}	α^{9}	α^{9}	α^{10}	α^{12}	α^{16}	α^{24}
α^{25}	α^{21}	α^{6}	α^{14}	1	α^{30}	α^{2}
α^{26}	α^{2}	α^{4}	α^{20}	α^{21}	α^{23}	α^{27}
α^{27}	α^{14}	α^{10}	α^{11}	α^{13}	α^{17}	α^{25}
α^{28}	α^{26}	1	α	α^{7}	α^{3}	α^{15}
α^{29}	α^{7}	α^{5}	α^{21}	α^{22}	α^{24}	α^{28}
α^{30}	α^{19}	α^{11}	α^{12}	α^{14}	α^{18}	α^{26}

CHAPTER FIVE

Conclusion

This thesis leads to the following conclusions:

a) The decoding of binary BCH codes involves finding the sigmas of the error-locator polynomial $\sigma(x)$ over $GF(2^m)$. In this context, when choosing a technique to obtain the sigmas of the error-locator polynomial for t not exceeding three, it is preferable to use the Peterson approach instead of the Berlekamp, if it is possible to store the various expressions for the sigmas efficiently.

b) Once the error-locator polynomial $\sigma(x)$ is determined, to get the error locations, the roots must be found. In this case, practical results have been found.
We have proved that the roots of the quadratic x^2+x+k_2, over $GF(2^m)$, can be found by solving a set of simple simultaneous equations.
In the cubic case, x^3+x+k_3, we have shown that when using a table-look-up approach, we have, at most, $\frac{n}{6}$ values of k_3 to consider.
We have established that the quartic $x^4+k_2x^2+k_3x+k_4$ over

$GF(2^m)$ does not have some or all of its roots in $GF(2^m)$ if $Tr\left(\frac{k_2^3}{k_3^2}\right) \neq Tr(1)$.

We have demonstrated that for polynomials of the form:

$$f(x) = x^a + x^{a-2} + k_3 x^{a-3} + k_4 x^{a-4} + \ldots + k_a \qquad a \geq 4$$

If a table-look-up is used, then the number of combinations of $k_3, k_4, \ldots k_a$ to be stored is $\leq \frac{(2^m)^{a-2}}{a}$.

Finally, we have developed some look-up tables.

LIST OF SYMBOLS

BCH	Bose-Chaudhuri-Hocquenghem
d_{min}	Minimum distance
e	Number of errors
E(x)	Error polynomial
GCD	Greatest common divisor
GF	Galois field
g(x)	Generator polynomial
I	Ideal
k	Information digits
k_i	Coefficient of error locator polynomial
LCM	Least common multiple
MDS	Microcomputer Development System
$m_j(x)$	Irreducible polynomial of minimum degree
n	Code length
P_o	Transition probability
q_o	Probability that a transmitted symbol will be correctly received
R	Ring
R(x)	Received polynomial
S_j	Syndrome (Power sum symmetric function)
Tr	Trace
t	Number of correctable errors

$w(A)$	Hamming weight
σ_i	Elementary symmetric function
$\Delta^{(k)}$	Coefficient used in the Berlekamp procedure
τ	Auxiliary polynomial
λ	Coset leader
α	Field element
θ	Identity element
$\psi(x)$	Error locator polynomial

REFERENCES

1. M. Pearl, "Matrix Theory and Finite Mathematics", International series in pure and applied mathematics, McGraw-Hill, 1973.

2. Garrett Birkhoff and Thomas C. Bartee, "Modern Applied Algebra", McGraw-Hill, 1970.

3. E.R. Berlekamp, "Algebraic Coding Theory', New York, McGraw-Hill Book Company, 1968.

4. W.Wesley Peterson and E.J. Weldon, Jr., "Error-Correcting Codes", Second edition, The MIT Press, Cambridge, Massachusetts, 1972.

5. Shu Lin, "An Introduction to Error-Correcting Codes", PrenticeHall Inc., Englewood Cliffs, New Jersey, 1970.

6. ELG 5373, ELG 5376, Notes of Lectures, Graduate courses 1975, 1976.

7. W.W. Peterson, "Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes", IRE Transactions on Information Theory, September, 1959.

8. R.T. Chien, "Cyclic Decoding Procedures for Bose-Chaudhuri-

Hocquenghem Codes", IEEE Trans. Inform. Theory, Vol. IT-10, pp. 357-363, October 1964.

9. R.T. Chien, "Memory Error Control Beyond Parity", Spectrum, July, 1973.

10. N.J.A. Sloane, "A Short Course on Error-Correcting Codes", Udine, 1975.

11. Adam Osborne and Associates Incorporated, "An Introduction to Microcomputers", Vol. I, P.O. Box 2036, Berkely, California 94702, 1976.

12. "8080 Microcomputer Development System Manual", Published by Intel.

13. R.C. Bose and D.K. Ray-Chaudhuri, "On a Class of Error-Correcting Binary Group Code", Information and Control, 1960.

14. Thomas C. Bartee, David I. Schneider, "An Electronic Decoder for Bose-Chaudhuri-Hocquenghem Error-Correcting Codes", IRE Transactions on Information Theory, Vol. IT-8, pp. 517-24, September, 1962.

15. R.T. Chien, B.D. Cunningham, and I.B. Oldham, "Hybrid Methods for Finding Roots of a Polynomial--With Application to BCH Decoding", IEEE Trans. Infor. Theory, Vol. IT-15,

p. 329, October, 1969.

16. Frank Polkinghorn, Jr., "Decoding of Double and Triple Error-Correcting Bose-Chaudhuri Codes", IEEE Transactions on Information Theory, October, 1966.

17. E.R. Berlekamp, H. Rumsey, and G. Solomon, "On the Solution of Algebraic Equations over Finite Fields", Information and Control 10, pp. 553-564, 1967.

18. G. Promhouse and S.E. Tavares, "The Minimum Distance of All Binary Cyclic Codes of Odd Lengths from 69 to 99", IEEE Transactions on Information Theory, Vol. It 24, No. 4, July, 1978.

19. S.G.S. Shiva - S. El-Guebaly, "Quartics in Relation to BCH decoding", to appear in digital processes.