SIMULATION OF ERROR CORRECTION ALGORITHMS USING REED SOLOMON CODES

By

ARTHURINE RENEE DAVIS BRECKENRIDGE

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

1975

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fullfillments of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1984

SIMULATION OF ERROR CORRECTION ALGORITHMS USING REED SOLOMON CODES

Thesis Approved:

J. K. Phillips
Thesis Adviser

J. P. Chandler

J. R. Van Doren

Norman N Durham
Dean of the Graduate College

PREFACE

The Reed-Solomon codes for multiple-error-correction are examined in this study. The results of a comparison between the conventional Gorenstein-Zierler method and a transform method are discussed, and simple examples are given. Then decoding algorithms are compared in terms of the numerical complexity. Finally the conclusions of the simulation are stated.

I wish to acknowledge all the members of my family for their continual assistance throughout my studies at Oklahoma State University. My husband, Bruce, my children, Jennifer and Gage, deserve my deepest appreciation for their constant support, moral encouragement, and understanding. Also I wish to thank my committee members Dr. J. P. Chandler and Dr. J. R. Van Doren for their contributions and advise. Finally, I wish to express my sincerest appreciation to my major adviser, Dr. J. R. Phillips for his help on this paper.

TABLE OF CONTENTS

TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Algebraic coding theory has its origins in the work of R. W. Hamming and C. E. Shannon, who were colleagues at Bell Telephone Laboratories in the late forties. However Hamming was the first coding theorist whose work attracted widespread interest. Some of Hamming's early work appeared as an example in Shannon's classic 1948 and 1949 papers (53). Apparently delayed because of patent considerations, Hamming's own paper appeared in 1950. Even though both were concerned with the fundamental problem of communications over noisy channels, there was a clear difference between the combinatorial, constructive viewpoint of Hamming and the statistical, existential viewpoint of Shannon. The distinction between coding theory and Shannon theory has increased in subsequent years. This paper is limited to the topics of coding theory.

The major coding theory papers of the early 1950's introduced a number of important concepts which laid the basis for algebraic coding theory. Chapter II of this thesis introduces some examples of Hamming's work to help the reader build a foundation for coding theory. Hamming (23) was concerned both with code construction and the bound

on the distance of a code's capacity to detect or correct errors. D. E. Muller (40) and I. S. Reed (46) not only constructed an important class of codes, they also gave some preliminary indications about how a large body of knowledge about finite mathematical structures might be brought to bear on the coding problem. D. Slepian (52) exposed the mathematical foundations of the subject of linear codes. M. J. E. Golay (20) discovered a particular code that has been shown to contain as subsets some of the most efficient linear codes.

The big breakthrough in the construction of error-correcting codes came in 1959 and 1960. The codes that are now universally called BCH codes were discovered by the French mathematican, A. Hocquenghem (24) and independently by R. C. Bose and D. K. Ray-Chaudhuri (11). It is important to remember that only the code theory, not the decoding algorithms, were discovered by these early writers. The BCH codes of block length of the form (2**m)-1 has been, perhaps, the outstanding success of the search for codes based on algebraic structures. Chapter III broadly defines BCH codes and explains methods for construction. Chapter IV explains why encoding is necessary to help ensure reliability of information.

Chapters V, VI, and VII try to explain the decoding of BCH codes. One of the principal virtues of these codes turned out to be their capability of being decoded by relatively straightforward algorithms. W. W. Peterson (42)

was the first to outline an efficient and economical decoding procedure which was actually realized by T. C. Bartee and D. I. Schneider (5) in a small special-purpose computer. Peterson's algorithm involved the solution of simultaneous linear equations over certain finite fields.

More recent work focused attention on the multisymbol generalization of these codes. One year after Peterson's algorithm, D. C. Gorenstein and N. Zierler (22) generalized all the previous work on binary BCH codes to non-binary codes. It turns out that the polynomial codes discussed by I.S. Reed and G. Solomon (47) belong to this general code class and hence may be decoded by the Gorenstein-Zierler algorithm. The non-binary BCH codes contain the Reed-Solomon (RS) codes as a proper subset. There are two areas (at least) of application of codes in non-binary symbols. First, data to be transmitted may appear in such a form and second, although the binary BCH codes tend to be highly efficient for the correction of independent errors, still greater efficiency may be obtained with non-binary codes when the errors occur in bursts.

BCH or RS codes are some of the most important classes of random-error-correcting codes known. Considerable work has been done on decoding of these codes. Though the details of an algorithm were first presented by Peterson, many improvements soon followed. By using the fact that the BCH codes are cyclic, R. T. Chien (15) obtained a significantly better algorithm, which was then modified and

improved by G. D. Forney (18). All of these revolutionary decoding algorithms are based on Algebraic decoding. By associating the symbols of certain linear cyclic codes with corresponding elements in a finite field, it is possible to define an error locator polynomial, whose roots reveal the locations of the symbols which are in error. The decoding problem can then be reduced to the computational problem of setting up this algebraic equation and finding its roots.

The Chien algorithm finds the roots of the error locator polynomial by testing each candidate, but the tests are done sequentially as the about-to-be decoded digits leave the buffer. This method circumvents the computational problem of finding roots of the error locator polynomial. Therefore, the determination of the coefficients of the error locator polynomial became the bottleneck of the BCH decoding. This bottleneck was broken by an iterative algorithm presented by E. R. Berlekamp (8). J. L. Massey (34) pointed out that this same algorithm also solves the linear feedback shift register synthesis problem. By introducing the scalar multiples of the reciprocal monic polynomials upon which Berlekamp's algorithm iterates, one can decode BCH codes without doing Galois field divisions.

In 1970, V. D. Goppa (21) discovered the codes that bear his name and are a natural generalization of BCH codes. Goppa also gave a decoding procedure for his codes that was analogous to the old Peterson-Gorenstein-Zierler algorithm. Goppa did not, however, generalize Berlekamp's iterative

algorithm. In 1975, Sugiyama, Kasahara, Hirasawa, and Namekawa (55) discovered that one can use the Euclidean algorithm to decode Goppa or BCH codes. This algorithm is easier to understand and sheds considerable light on understanding Berlekamp's algorithm. In fact, with hindsight it is now possible to view Berlekamp's algorithm as an improved version of algorithms based on Euclid.

Since excellent algorithms now exist, current research is focused on reducing the numerical complexity of the conventional BCH or RS algorithms. Recently it was proposed that the use of a finite field transform may be possible for decoding. D. Mandelbaum (30) developed a decoding algorithm using a transform over GF(p**m). The disadvantage of this transform method is that the code length is such that the most efficient fast Fourier transform algorithms cannot be used to yield transform decoders. This problem was resolved soon by several solutions. One scheme investigates a modification of a method by S. Winograd (57) for computing transforms over GF(2**m) that is based on the Chinese remainder theorem. Another scheme was proposed by J. Justesen (25) that transforms over GF(Fermat prime). Many improvements along this scheme have been proposed (49). J. H. McClellan (35) recently constructed hardware to implement the Fermat prime theoretic transforms.

The major goal of this thesis is to provide a numerical comparison by software simulation to see if the new transform methods actually reduce numerical complexity.

Chapter VIII describes and gives the conclusions of this simulation. Improvements in decoding using transforms were also extended to encoding. D. Mandelbaum (31) showed how to construct error-correcting codes by interpolation by applying fast Fourier transform and Lagrange interpolation. Chapter IX discusses further research in the field of Algebraic coding theory. Another goal of this thesis is to interest the reader into further exploring coding theory. There are several textbooks available including: Peterson (43), Berlekamp (8), Lin (28), van Lint (29), Peterson-Weldon (44), and McWilliams-Sloane (38).

CHAPTER II

BACKGROUND THEORY

Algebraic coding theory history begins with the Shannon coding theorems which guarantees the existence of codes that permit the transmission of information at high rates with vanishingly small probability of error (53). M. J. E. Golay R. W. Hamming, D. E. Muller, I. S. Reed, and D. Slepian made the first essential steps in Algebraic coding theory with the effective encoding and decoding techniques of some particular linear codes (10). This paper is intended as an introduction to the encoding and decoding of the code developed by I. S. Reed and G. Solomon (47). Reed-Solomon codes (RS) are the most powerful of the known classes of block codes for correcting random errors and multiple burst errors (8,38). However before going into the details of RS codes, this background chapter was organized for those who need some acquaintance with coding theory.

Information is said to be placed into code form by encoding and extracted from code form by decoding. A basic class of error-control codes is linear block codes. The encoding procedure of linear block codes consists of two steps: 1) the initial information sequence is divided into message blocks of length k; and 2) every message block is

transformed into a codeword of length n by annexing r check symbols. Such encoding can not prevent transmission errors, but can reduce the error's undesirable effects. The n-k or r check symbols are redundant symbols which carry no new information but function to provide the code with the capacity of detecting and correcting errors. Also, the k message symbols are from an initial alphabet defined by a Galois finite field GF(q).

A block code is often denoted as an (n,k) code or as an (n,k,d) code on GF(q) where n, k, d are considered parameters. N is the block length of the code. There are q**k different codewords so k is the dimension of the code. The minimum distance d of a code is the minimum number of places in which any two codewords differ. "A linear code with minimum distance d can correct [1/2(d-1)] errors." (38,p.10) ([x] denotes the greatest integer less than or equal to x.) The distance, length, and number of information symbols in any RS code are related by d=n-k+1. An (n,k) code uses n symbols to send k message symbols, so it has a rate or efficiency, R=k/n (8).

Ideally, the decoding process for any code is generally easier if a message is encoded as a separable, systematic, and cyclic codeword. A separable code is one which divides a codeword into an information part and a redundant checking part. A systematic code is one which has distinguishable information symbols and check symbols. Over a finite field, all linear codes are systematic. A cyclic code is a linear

block code which any cyclic shift of a codeword is also a codeword (28).

The encoding problem is given any particular sequence of k message symbols, the transmitter must follow rules for selecting r check symbols so the receiver can decode and recover the message. A simple binary example in Figure 1 developed by R. W. Hamming illustrates basic concepts. Each check symbol must be some function of the message symbols. In the simple case of single-parity-check codes, the check symbol is chosen to be the binary sum of all the message symbols or parity. If there are several parity checks, one solution is to set each check symbol equal to the sum of subsets of the message digits. In the Hamming code example, a message symbol is a member of a subset if the binary representation of the number position has a one in that bit position. Each subset can correspond to a row of a matrix. A generator matrix G can be written which has one's in the place of each row where the corresponding check is applied. In general, the generator matrix of a separable code is an r by r identity submatrix and a k by r submatrix that describes the interdependence between information and check symbols. Notice the matrix in Figure 1 does not generate a separable code.

After the message sequence is encoded, the codeword is transmitted across the noisy channel. The channel adds the noise vector

$$E = \{ e_0, e_2, \ldots, e_{n-1} \} \text{ where} \tag{2.1}$$

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Message	*	*	1	*	0	1	0	*	0	1	0	1	0	1	1
Codeword	0	1	1	1	0	1	0	0	0	1	0	1	0	1	1
Error	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Recieved Vector	0	1	1	1	0	1	0	0	0	1	1	1	0	1	1
Correction	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
Message	*	*	1	*	0	1	0	*	0	1	0	1	0	1	1

* The check symbols to be determined

RESULTS OF PARITY ON CHECK SYMBOLS

Check Positions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ENCODING															
Check 1 = 0 =	*		+1		+0		+0		+0		+0		+0		+1
Check 2 = 1 =		*	+1			+1	+0			+1	+0			+1	+1
Check 3 = 1 =				*	+0	+1	+0					+1	+0	+1	+1
Check 4 = 0 =								*	+0	+1	+0	+1	+0	+1	+1
DECODING															
Check 1 = 1 =	+0		+1		+0		+0		+0		+1		+0		+1
Check 2 = 1 =		+1	+1			+1	+0			+1	+1			+1	+1
Check 3 = 0 =				+1	+0	+1	+0					+1	+0	+1	+1
Check 4 = 1 =								+0	+0	+1	+1	+1	+0	+1	+1

SYNDROME = 1011 OR THE ELEVENTH POSITION
Note the syndrome is written with the check 1 associated with the low order bit, so the order becomes Check 4, Check 3, Check 2, and Check 1

M =	0 0 0 0 0 0 0 1 1 1 1 1 1 1 1	Parity check 4
	0 0 0 1 1 1 1 0 0 0 0 1 1 1 1	Parity check 3
	0 1 1 0 0 1 1 0 0 1 1 0 0 1 1	Parity check 2
	1 0 1 0 1 0 1 0 1 0 1 0 1 0 1	Parity check 1

Figure 1. Single-Error-Correcting Hamming Code Over GF(2)

$$E_i = \begin{cases} 0 & \text{if the channel does not change the ith digit} \\ 1 & \text{if the channel does change the ith digit} \end{cases}$$

The received vector consists of the code vector plus the error vector where R = C + E or C = R - E. Therefore if the decoder is able to find the correct error vector E, then the code vector C can be found by subtracting the error vector from the received vector (In the binary case, of course +=-, or difference = sum).

In the decoding process, one can define parity check matrix M related as illustrated in Figure 2 to the transpose of the generator matrix where

$$(M C^T)^T = (C^T)^T M^T = C M^T \tag{2.2}$$

Using multiplication, the symbol subsets are checked for correctness. Any occurring abnormalities are called syndromes. The encoding process assures M*(C Transpose)=0; and the decoding process assures

$$M R^T = M (C^T + E^T) = M C^T + M E^T = M E^T \tag{2.3}$$

Therefore the syndrome depends only on the error and not on the codeword sent. Also the syndrome vector where

$$S = \{ s_1, s_2, \ldots, s_{d-1} \} \tag{2.4}$$

contains all the information regarding the error that has been added to the codeword during the transmission. If there is a single error, then the syndrome is exactly the corresponding column of the parity check matrix and the error position can be found by table lookup. Once the error location is known, in the binary case, the error is

A) The information $I = a_0, a_1, \ldots, a_{k-1}$

B) The (n-k) * n parity check matrix H

$$H = [\, A \mid \text{Idenity}_{n-k} \,]$$

C) The codeword $C = a_0, \ldots, a_{k-1}, a_k, \ldots, a_{n-1}$

D) The n * (n-k) generator matrix G

$$G = [\, \text{Idenity}_k \mid -A^T \,]$$

because $H C^T = 0$

$$[\, A \mid \text{Idenity}_{n-k} \,] \begin{vmatrix} a_0 \\ \vdots \\ a_{n-1} \end{vmatrix} = 0$$

$$\begin{vmatrix} a_k \\ \vdots \\ a_{n-1} \end{vmatrix} = -A \begin{vmatrix} a_0 \\ \vdots \\ a_{k-1} \end{vmatrix}$$

Note: In the binary case -A=A
in the non-binary case this is not true

Figure 2. Properties of Linear Codes

subtracted from the received vector and the codeword is corrected. In the non-binary case, the value of the error must be determined before subtracting. The message symbols can be separated from the check symbols and thus completing the decoding process.

A great deal of work in constructive coding theory followed the appearance of Hamming's pioneering paper (23). Improvements were pursued especially for a separable code with the capacity to correct more than a single error. In fact, a mathematical treatment of the encoding-decoding process was sought to build a structure so that the code may be decoded systematically without table lookup (which is clearly impractical for large code sets). Such a process was discovered known today as BCH codes. Reed-Solomon codes are an important subclass of BCH codes. Also the single-error-correcting Hamming code in Figure 1 can be defined as the simpliest type of BCH code.

The BCH codes viewed the code sequences as polynomials. The codeword vector

$$C = \{ a_0 , a_1 , \ldots , a_{n-1} \} \tag{2.5}$$

$$C(x) = \sum_{i=0}^{n-1} a_i x^i = a_0 + a_1 x^1 + \ldots + a_{n-1} x^{n-1}$$

$$\text{where } a_i \in GF(q).$$

is identified as the coefficents of the powers of a variable. Also the generator matrix can be constructed by a polynomial. Fundamentally, the vector representation is the

same as the polynomial representation. The code is the same; rearrangement of columns of the generator matrix only affects how and where one finds the column corresponding to the syndrome that emerges. The encoding-decoding process of the polynomial representation will be explained in the following chapters. The conceptual gap between the Hamming codes or the single-error-correcting codes and the t-error-correcting BCH codes is considerable and represents a decade of research.

CHAPTER III

ENCODING OF REED SOLOMON CODES

An important process in error correction is the encoding or placing information into codewords. The encoding of a Reed-Solomon (RS) code can be handled by any of the following three methods. The first or conventional method describes techniques which can be used for any cyclic code. RS codes are cyclic. The next method which is the original method of I.S. Reed and G. Solomon (47) is mentioned for theoretical interest though not generally used because the encoder is not systematic. Finally, a new scheme for reducing the numerical complexity of the standard RS encoding algorithm is developed. The new method is a combination of the Chinese remainder theorem, discrete Fourier transforms, and Lagrange interpolation..

Conventional Method

The conventional method of polynomial encoding still requires a codeword to have zero syndrome defined algebraically by M * (C Transpose) = 0. This method of encoding is based on the fact that the coded vector must be, considered as a polynomial, a multiple of the generator polynomial G(x). Since a linear code is generally defined

by a generator, a formal definition of RS codes are needed to insure proper encoding. Conventionally RS codes are generated based on the fact RS codes are a subset of BCH codes which are a subset of cyclic codes.

A cyclic code of length n is based on a generator polynomial G(x) with the following properties: 1) there is a unique monic polynomial G(x) of minimal degree in the code, 2) the code consists of all multiples of a fixed polynomial G(x), 3) G(x) is a factor of (x**n) -1, 4) the message I(x) becomes the codeword I(x)G(x), 5) code is generated by the rows of a generator matrix defined where multiplication by x corresponds to a cyclic shift where

(3.1)

$$G = \begin{vmatrix} G(x) \\ xG(x) \\ \cdots \\ x^{n-r-1}G(x) \end{vmatrix}$$

where $G(x) = g_0 + g_1 x + \ldots + g_r x^r$

The generator polynomial has degree equal to the distance of the code minus one or the number of check symbols. Since G(x) is a factor of the polynomial representation of the codeword, the generator has distinct roots w or zeros of the code. The number of zeros of the generator polynomial depend upon how many errors one wishes to detect (38).

Each element w**i of the field is a root of a unique irreducible polynomial M(i)(X) of minimal degree. Terms basic to theory of finite fields are reviewed in Appendix A.

Then G(X) must be divisible by each of the polynomials M(1)(X), M(2)(X), . . . , M(d-1)(X) and hence, by their least common multiple:

$$G(X) = \text{LCM} \left| \prod_{i=1}^{d-1} M(i)(X) \right| \tag{3.2}$$

Since each of the factors M(i)(X) is irreducible, the least common multiple of the M(i)(X) is simply the product of the minimal polynomials M(i)(X), with the duplicates omitted. Duplications are quite possible; and occur in fact for any w**i and w**j that are roots of the same polynomial M(i)(X) (42).

A cyclic code of length n over GF(q**m) is a BCH code of designed distance d defined by the distinct roots

$$w^{b}, w^{b+1}, \ldots, w^{b+d-2} \tag{3.3}$$

of the generator. RS codes are a subset of BCH and can be defined with the following restrictions: 1) the power of the field is one or GF(q**1), and 2) the zeros generally start with b=1. An RS code is a cyclic code of length n over GF(q) defined by the distinct roots

$$w^{1}, w^{2}, \ldots, w^{d-1} \tag{3.4}$$

of the generator. Thus an RS code is a block code with n=q-1 symbols, with k=n-d+1 message symbols where d is the minimum distance. Important special cases are b=1 (called narrow-sense BCH codes) or n=q-1 (called primitive BCH

codes). All BCH codes hereafter referred to are assumed to be narrow-sense and primitive.

In summary, to encode the k information symbols into an n=q-1 symbol RS code, one must first define the generator polynomial

$$G(X) = \prod_{i=1}^{d-1} (x - w^{i}) \tag{3.5}$$

where w is a primitive nth root of unity as defined in Appendix A. The code consists of all multiples of G(X) subject to the constraint

$$x^{n} - 1 = 0 \tag{3.6}$$

The message polynomial must be exactly divisible by the generator polynomial. Let I(X) be a temporary polynomial where the message corresponds to the positions

$$a_{n-1}, a_{n-2}, \ldots, a_{k}, 0, \ldots, 0 \tag{3.7}$$

and the coefficients of the remaining n-k lower order positions are momentarily zero. The division of this message polynomial I(X), by the generator polynomial G(X) will produce a remainder R(X). The remainder R(X) has degree less than (n-k) which is the degree of the generator polynomial.

$$\begin{aligned} I(X) &= Q(X)\, G(X) + R(X) \\ C(X) &= I(X) - R(X) \end{aligned} \tag{3.8}$$

If the remainder is subtracted from the message polynomial,

then the result is exactly divisible by the generator polynomial or a codeword. The calculation of the remainder can be accomplished in the general case by polynomial division illustrated in Figure 3.

However in the binary case, the apparent complexity of much of the division process for the modular reduction of polynomials over GF(2) can be handled simply by means of shift registers with feedback paths. In Figure 4, the digits of the message come from the right, the highest power first with trailing zeros automatically supplied for the check bits to be determined. The paths below each register show where the feedback occurs according to the generator polynomial. A practical encoder is where the message digits are shifted out and when the remainder is computed, the remainder is then shifted out to form the entire coded message. The first digit of the remainder is always omitted, of course, since it is always zero. In hardware, encoding is a shift register with the additional logic for addition by exclusive or in each position. In software, encoding is a test for a one in the leftmost position. If one is found, then logically add the pattern of ones required by the feedback paths or logically add the vector representation of the primitive polynomial. In either case, the codeword polynomial is congruent to zero modulo the generator polynomial. Also in either case, this encoding method only applies with codes with binary symbols.

```
Powers of x are represented by coefficients with positions:
          15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
          x  x  x  x  x  x  x  x  x  x  x  x  x  x  x  x

         ------------------------------------------------
1-7+1+2-2 |1 +2 +3 +2
          1 -7 +1 +2 +2
          -------------
            +9 +2 +0 +2
            +9-63 +9+18+18
            --------------
               -3 -9 +1+18
               -3+21 -3 -6 +6
               --------------
                  +4 +4 +7 -6
                  +4-28 +4 +8 -8
                  --------------
                     -2 +3 +3 +8
                     -2+14 -2 -4 +4
                     --------------
                        +6 +5 -5 -4
                        +6-42 +6+12-12
                        --------------
                           -4 +6 +1+12
                           -4+28 -4 -8 +8
                           --------------
                              -5 +5 +3 -8
                              -5+35 -5-10+10
                              --------------
                                 +4 +8 +2-10
                                 +4-28 +4 +8 -8
                                 --------------
                                  +2 -2 -1 +8
                                  +2-14 +2 +4 -4
                                    --------------
                                       -5 -3 +4 +4
                                       -5+35 -5-10+10
                                       --------------
                                          -4 +9 -3-10
                                          -4-28 -4 -8 +8
                                          --------------
                                              -2 +1 -2 -8
```

Generator Polynomial

$$= (x-6)(x-6^2)(x-6^3)(x-6^4)$$
$$= (x-6)(x-2)(x-12)(x-4)$$
$$= x^4 - 7x^3 + x^2 + 2x - 2$$

Primitive root of unity for GF(17) and N=16 is 6

$$C(X) = I(x) - R(x)$$
$$= x^{15} + 2x^{14} + 3x^{13} + 2x^{12} - (-2x^3 + 1x^2 - 2x - 8)$$
$$= x^{15} + 2x^{14} + 3x^{13} + 2x^{12} + 2x^3 + 16x^2 + 2x + 8$$

Figure 3. Encoding Reed-Solomon Code (16,12,5) Over GF(17) By Conventional Division Method

State	0 1 1 0 0	1 1 0 0 0 0 0 0 0 0
Feedback	0 0 0 0 0	
Result	0 1 1 0 0	
	1 1 0 0 1	1 0 0 0 0 0 0 0 0 0
	1 0 0 1 1	
Shift	0 1 0 1 0	
	1 0 1 0 1	0 0 0 0 0 0 0 0 0
	1 0 0 1 1	
Shift	0 0 1 1 0	
	0 1 1 0 0	0 0 0 0 0 0 0 0
	0 0 0 0 0	
Shift	0 1 1 0 0	
	1 1 0 0 0	0 0 0 0 0 0
	1 0 0 1 1	
Shift	0 1 0 1 1	
	1 0 1 1 0	0 0 0 0 0
	1 0 0 1 1	
Shift	0 0 1 0 1	
	0 1 0 1 0	0 0 0 0
	0 0 0 0 0	
Shift	0 1 0 1 0	
	1 0 1 0 0	0 0 0
	1 0 0 1 1	
Shift	0 0 1 1 1	
	0 1 1 1 0	0 0
	0 0 0 0 0	
Shift	0 1 1 1 0	
	1 1 1 0 0	0
	1 0 0 1 1	
Shift	0 1 1 1 1	
	1 1 1 1 0	
	1 0 0 1 1	
	0 1 1 0 1	Remainder

Message (0 1 1 0 0 1 1 0 0 0 0 * * * *)
Codeword (0 1 1 0 0 1 1 0 0 0 0 1 1 0 1)

Figure 4. Encoding Reed-Solomon Code (15,11,5) Over GF(16) By Linear Feedback Shift Register Method

Original Method

The next method of encoding is based on the original work of I.S. Reed and G. Solomon (47). Reed's work on coding theory developed because of the decoding failures that occurred in Hamming's coding if the number of errors was not equal to one (46). Reed and Solomon proposed a code which maps k-tuples over GF(q) into 2**n-tuples over GF(q). Let I(x), be the message symbols to be encoded into a codeword C(x), using the polynomial P(x) defined in Figure 5 where w is the primitive root of unity of a suitable irreducible polynomial over GF(q). Therefore the non-zero elements form a multiplicative cyclic group. The formulas and an example of this Reed-Solomon encoding method is given in Figure 5.

However the codeword is not systematic unless regarded as a mapping of binary sequences of (mn) bits into binary sequences of n(2**n) bits. However this theoretical approach suggested a certain viewpoint where RS codes can be said to result from a generalized interpolation. Observe the P(w**i) is the remainder when the message I(X), of degree less than k is divided by a minimal polynomial.

Let w be a primitive element of GF(q**m). Let
M(i)(x) = x - w**i for i = 0, 1 ,...,q**(m-2)
Therefore

$$R(x) = r_i \quad \text{modulo } (x - w^{**}i)$$

$$R(x) = Q(x) (x - w^{**}i) + r_i$$

$$R(w^{**}i) = r_i \qquad (4,p.298)$$

So one can say I(x) is encoded into

$$I(x) = (a_0, a_1, \ldots, a_{k-1}) \text{ where } a_i \in GF(q)$$

$$I(x) = (0,0,0,0,0,0,0,0,2,3,2,1)$$

$$P(x) = a_0 + a_1 x^1 + \ldots + a_{k-1} x^{k-1} = \sum_{i=0}^{k-1} a_i x^i$$

$$P(x) = 2x^8 + 3x^9 + 2x^{10} + x^{11}$$

$$C(x) = (P(0), P(w), P(w^2), \ldots, P(1))$$

$$C(x) = (0,15,7,3,15,2,3,16,14,14,13,12,9,9,2,12,8)$$

Figure 5. Encoding Reed-Solomon Code (16,12,5) Over GF(17) By Reed's Original Method

$$I(x) = (0,0,0,0,0,0,0,0,2,3,2,1)$$

$$I(w^i) = (0,0,0,0,0,0,0,0,0,0,0,0,2,3,2,1)$$

$$R(w^i) = (8,9,10,5,8,14,14,4,0,11,4,3,9,2,15,3)$$

$$R(x) = 9 (6x^3 + 11x^2 + 4x + 2) + 10 (7x^3 + 16x^2 + 5x + 7)$$

$$5 (13x^3 + 14x^2 + 11x + 5) + 8 (8x^3 + 10x^2 + 14x + 4)$$

$$R(x) = (-8,-2,+1,-2,0,0,0,0,0,0,0,0,0,0,0,0)$$

$$C(x) = (+8,+2,16,+2,0,0,0,0,0,0,0,0,2,3,2,1)$$

Figure 6. Encoding Reed-Solomon Code(16,12,5) Over GF(17) By New Interpolation Method

(3.9)

$$(r_0, \ldots, r_{k-1}, r_k, \ldots, r_{n-1})$$

where r is the residue of I(x) modulo M(i)(X). This generalization of the Chinese remainder theorem, which deals with polynomials over the GF(p**m) instead of integers, is stated in Appendix B. This theorem guarantees the first k residues are enough to reconstruct the message I(X) in the absence of errors. If additional residues were sent, the message might be communicated despite some disruption of the transmission. Thus the n-k residues are redundant residues which are included in a codeword for protection against errors. Reed-Solomon or any code encoded by this method are called redundant residue codes. Restated

> Reed-Solomon codes were the first codes constructed in terms of interpolation. It is well-known that a polynomial f(x) of degree k-1 over any field is determined uniquely by its values at any k distinct points x_i, $0<=i<k$. If these k values $f(x_i)$ are transmitted, the receiver can reconstruct the function f(x): this is called Lagrange interpolation. If one or more of the values $f(x_i)$ are changed by noise, then the wrong function will be constructed. However, if r extra (redundant) values of f(x) at r additional points are transmitted, then by taking all combinations the most often, the correct function f(x) will be selected even if noise has affected up to [r/2] values $f(x_i)$ (31,p.27).

From a practical standpoint, this encoding process makes complete decoding difficult.

Interpolation Method

However, Reed's original encoding process was the foundation for current research which states the Chinese

remainder theorem another way. After calculation of the residues, the vector can be viewed as the polynomial interpolated through the points

$$\left(w^{i}, r_{i} \right) \quad \text{for} \quad 0 <= i < n-k \tag{3.10}$$

Instead of transmitting the residues, the residues are used to interpolate a polynomial that is a valid codeword before transmission.

The new encoding procedure of an RS code is composed of the following two steps:

1) Compute I(w**i) for 1<=i<=d-1 by the technique used to compute syndromes in the decoder. Note that by

$$C(x) = I(x) - R(x)$$

$$I(w^{i}) = R(w^{i}) \quad \text{for} \quad 1 <= i <= d-1.$$

2) Compute R(x) from R(w**i) using Lagrange interpolation:

$$R(x) = \sum_{i=1}^{d-1} R(w^{i}) E_{i}(x)$$

where E(x) is defined by

$$E_{i}(x) = \frac{\prod_{j \neq i} (x - w^{i})}{\prod_{j \neq i} (w^{i} - w^{j})} \quad \text{for } 1<=i<=d-1$$

(50,p.223)

This method encodes a systematic codeword which results in the identical symbols transmitted as the conventional method. An example in given in Figure 6. The immediate advantage of Lagrange interpolations is that matrix $E_i(x)$

has to be calculated only once when initializing the encoder. This preprocessing of coefficients reduces the overall complexity of the encoder.

Summary

Encoding is very important process and the basis for any error correcting scheme. The methods discussed in this chapter have one immediate advantage. The encoding is very similar to the first step of decoding. Since the encoding can be viewed as a syndrome-like calculation, it can be implemented using the algorithm used in the decoder. Since the decoding process will be discussed in detail in the following chapters, program design logic and numerical complexity are not included at this time. The examples used in this chapter will be continued throughout the paper.

CHAPTER IV

TRANSMISSION

Recall in a linear block code, a particular sequence of n digits can be encoded as a codeword. Although there are q**n different sequences of length n, only q**k of these sequences are codewords, because the r check symbols within any codeword are completely determined by the k message symbols. No matter which codeword is transmitted, any of the q**n possible sequences of length n may be received if errors have been induced. The decoder must attempt to recover the correct codeword by implementing a coding scheme. When an error detection and correction coding scheme is considered in a transmission or a storage system, one should ask 1) what are the sources of errors anticipated, 2) what is needed to achieve the capacity of the code and 3) what error correction strategy is appropriate for the application. Each of these considerations play an important role in the design of the coding scheme.

The selection of an error management scheme is based on the type and distribution of the errors which occur. To better understand the type of errors that can be encountered, a specific example of an optical disc is given

to illustrate typical sources of errors. The sources can be divided into the following categories: 1) random-noise errors, 2) media-defect errors, 3) media-damage errors, 4) media-blockage errors, and 5) equipment-induced errors. Random-noise errors occur as a result of noise in the transmission causing a misinterpretation of a symbol. Media-defect errors occur as a result of imperfect fabrication of the optical disc storage media. As state-of-art technology of the fabrication processing improves, these errors can also be handled efficiently with encoding. Media-damage errors are more difficult to deal with since they occur at any point during the useful life of the storage media and can be very large in extent. Media-blockage errors occur as a result of dust or other pollutants settling on the recording media and causing optical blockage that prevents proper recording or playback. Equipment-induced errors are primarily related to disturbances. In the record mode, such a disturbance may result in a short loss of recorded data. In the playback mode, these disturbances may cause a decision error in the demodulator. Figure 7 illustrates an error management technique which can be implemented to minimize these error sources (6).

The application or sources of errors determine what type of errors one needs to detect. Already mentioned are random errors in a message or namely an equal probability of an error in each symbol position. However in practice there

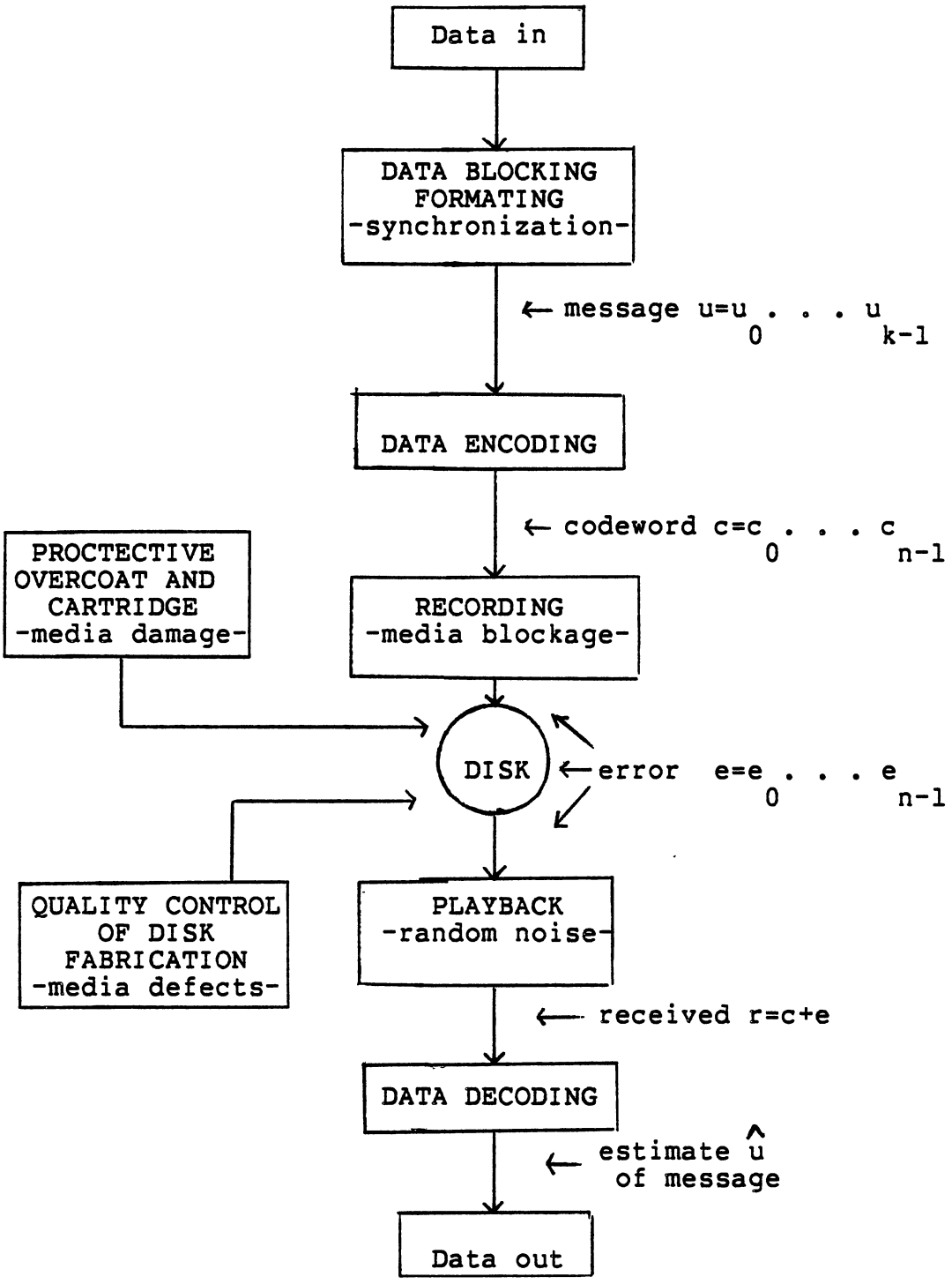


Figure 7. Flow Diagram of Error Management Shceme

are often reasons for errors to be more common in some positions in the message than in others, and it is often true that errors tend to occur in bursts and not be independent. One of the accepted definitions of a burst in coding theory is the following: "A burst of length b is a sequence of b digits, the first digit of which is non-zero (16,p.292)." The total number of bursts with a specific length can be readily determined in a given error sequence. For instance, a typical error sequence in a binary system may have the following appearance:

(4.1)

. . . 0 0 0 1 0 1 1 0 0 1 1 0 0 0 0 1 0 1 1 1 1 0 0 . . .

This sequence contains 10 bursts of length one, or 6 bursts of length two, or 5 bursts of length three, or four bursts of lengths four and five. In general, codes for correcting random errors are not efficient for correcting burst errors.

Coding is also required in communication systems to combat the errors that occur in the guesses of the demodulator. It has been recognized (18) that there are advantages in allowing the demodulator not to guess at all on certain transmissions when the evidence does not clearly indicate one signal as the most probable. Such events are called erasures. It is convenient to imagine that in the event of an erasure the demodulator does make some guess, perhaps arbitrary, but in addition passes on the side information to the decoder that this guess is absolutely unreliable and is to be disregarded. The best strategy is

to demodulate sufficiently weak or ambiguous received signals not as any of the q symbols in the input alphabet but as an additional symbol, such as ?. An erasure implies an unknown symbol at a known location and an error implies that the location and value are both unknown.

In this simulation, Reed-Solomon (RS) codes are constructed by algorithms that are very effective in correcting random and burst errors. RS codes are a maximum distance code capable of correcting d=2t+1 random errors. However due to present technology or applications only binary codes derived from q-ary RS codes are of interest. For example, GF(q) will be represented as GF(2**m) where m is a positive integer. Since each code symbol is an m-tuple over GF(2), a t-error-correcting RS code is capable of correcting any error pattern that affects t or fewer m-bit symbols. In general, the RS code with error correcting capacity t can be used to correct any of the following errors:

1) All single bursts of length b no matter where they start, if b<=m(t-1)+1.

2) Two bursts of length no longer than b each, no matter where each burst starts, if b<=m[(t/2)-1]+1, or any p bursts of length no longer than b each, no matter where each burst starts, if b<=m[(t/p)-1]+1.
(27,p.206)

For example, when each binary bit is considered as a symbol only random errors can be detected. A burst of errors in t adjacent positions is corrected identical to t random positions. However when a m-tuple of binary bits is

considered a symbol, a t-error-correcting code can correct more than t binary symbols. Let m=5 and t=4 then a burst of length 16 cannot affect more than four m-bit symbols, and can be corrected by a four-symbol-correcting code. However, considered as random one binary bit errors, one would need a t>=16 correcting code.

Reed-Solomon codes are also implemented by algorithms that handle both errors and erasures. An erasure pattern is correctable if (and only if), by substituting all possible combinations of symbols at these erased symbols, only one results in a codeword. With a t-error-correcting code, any pattern of 2t erasures is correctable. This follows immediately from the fact that, with s=2t erasures, any two n-tuple resulting from different substitutions can differ at most at 2t digits. However the minimum distance is d=2t+1 which means these two n-tuples can not both be codewords. Erasures are often compounded with non-erasure errors. Therefore there is a trade-off between the number of correctable errors and erasures. However a multiple-error-correcting code is capable of correcting any combination of t errors and s erasures as long as the minimum distance of the code is at least 2t+s+1<=d. Since the erasure positions are known, improvements are made in the capacity of the code.

Now that sources and type of errors have been defined, one needs to design a coding strategy to achieve the capacity of the code. Generally speaking the longer the

block length n, the more storage the decoder requires, and the greater the minimum decoding delay. It is also generally true that the longer the code block the larger the class of errors to be corrected, hence the more complicated the decoding circuits or decoding procedures. However, the distribution of errors in longer code blocks becomes much more predictable, thereby permitting the use of codes with smaller redundancy while maintaining the same reliability. Therefore the actual length of the block will depend on the application. For example, for intramachine transmission, such as going in and out of an internal random-access storage, the primary coding requirements are very high reliability and speed. For intermachine data transmission, the primary requirements are still high reliability but also high information rate. Since a decoding delay does not reduce throughput, one would tend to use longer codes with lower redundancy even though they require more decoding complexity.

An optimal coding strategy can be achieved and the best code obtained, only after the designer evaluates several alternatives. The designer has control over the parameters, distance, length, and number of information symbols, which are all related by d=n-k+1. One main consideration is if the number of errors greater than or equal to the distance then the algorithm will either misdecode or fail to decode. A decoding failure is when the decoder will not decode the received word into any of the possible transmitted message

words. A decoding error is when the decoder decodes the received word into the wrong codeword. This simulation considers a decoding failure to be preferable to a decoding error.

If a decoding algorithm decodes every possible received word into one of the possible transmitted codewords then it is a complete decoding algorithm. Since different applications have different requirements there are many courses of action besides full-power correction with block codes.

One approach is error detection. The main advantage is the simplicity of its implementation. An error is detected if the received message yields a non-zero syndrome. For cyclic codes, a division circuit plus a test for zero constitutes a complete decoder. Error detection is an attractive means of error control provided it is possible for retransmission. On the other hand, an error due to permanent damage in the storage medium will not be successfully avoided by retransmission.

Another approach is partial correction in the error-control scheme. One major reason is to minimize the decoding complexity. In the case of multiple-error-correction, decoding complexity grows exponentially with the number of errors corrected. Thus, even if a given code can correct $t>2$ errors, one may still want to go through a double- error-correction procedure and test the syndromes for possible erroneous correction. If single or double

errors account for a large portion of the overall error rate, considerable reduction in average decoding delay can thereby be achieved. If more than two errors occur the correction algorithm can output a decoding failure message or try a more powerful correction procedure.

Another approach is the use of erasures which tends to reduce the uncorrectable-error rate. "The amount of improvement is a function of the detailed statistics of the detected signals and of the thresholds that define the erasures (56,p.64)." The price of improvement is an increase in decoding complexity. When correcting combinations of errors and erasures with a multiple-error-correcting code, one must perform the additional step of transforming the error syndromes in order to separate the erasures from the non-erasures before the ordinary decoding procedures can be applied. Another price of improvement is a decreased information rate. Since q symbols in a field are represented by m-tuples, in order to represent an unique erasure symbol an (m+1) bit is necessary. Therefore decreasing the efficiency of the code defined as R=k/n. Erasures are not considered in the simulation of this study. The main reason for exclusion is current literature still uses conventional algorithms enhanced to correct erasures developed by G.D. Forney, Jr. (19).

The approach of this simulator is to use a combination of detection, partial correction, and full-power correction. The codeword is encoded then transmission is simulated.

During this phase, an error vector is added to the codeword. Continuing the example of Chapter II, then

$$
\begin{aligned}
C(x) &= (+8,+2,16,+2,0,0,0,0,0,0,0,0,2,3,2,1) \\
E(x) &= (\ 0,\ 0,\ 0,\ 1,0,2,0,0,0,0,0,0,0,0,0,0) \\
R(x) &= (+8,+2,16,+3,0,2,0,0,0,0,0,0,2,3,2,1)
\end{aligned}
\tag{4.2}
$$

the received word would be tested to see if it was a valid codeword. If the syndromes indicate an error, then the decoder will attempt to locate the errors. Details of the decoder are discussed in the following chapters.

CHAPTER V

STAGE I: CALCULATION OF THE SYNDROMES

Extensive research has been done on decoding BCH codes, and efficient algorithms exist. Most of the current research has been focused upon reducing the numerical complexity of the conventional BCH or Reed-Solomon (RS) encoding/decoding algorithm. Since the encoding of RS code is performed block by block (k message symbols encoded to n code symbols), the received sequence is therefore decoded a block of n digits at a time. The basic function of a decoder is to test whether or not the received word is a codeword (or whether it is divisible by the generator polynomial G(x) of the code used at the encoder). Detection of an error and possible correction can be accomplished. The decoding is generally divided into three stages: 1) calculation of the syndromes, 2) calculation of error locator and error evaluator polynomials, and 3) evaluation of the calculated polynomials for the locations and values of the errors. This chapter deals only with the initial stage of calculating the syndromes. Major emphasis has been placed on evaluating the syndromes from the received vector. Since the syndromes contain all the information about the errors, efficient calculations of the syndromes is very

important in the decoding process. Various methods will be introduced, however, if more details are desired, references will be cited.

Matrix Multiplication

One method of syndrome calculation is matrix multiplication. The codeword represented as a vector defined the syndrome equal to the parity check matrix times the transpose of the codeword. The parity check matrix M is an 2t*n matrix and the received vector R is an n-vector. The syndrome vector S is the product of M and R. "By definition, the i-th component of S is the dot product of the i-th row of M with R. Computing vector S as indicated requires 2tn multiplications and 2t(n-1) additions." (4,p.195)

This straightforward method was used to introduce the concept of syndrome as illustrated by Hamming's single-error-correcting code in Figure 1, each of the n columns of the parity check matrix must contain a different non-zero binary m-tuple, which is the location position of that symbol. As long as the n different symbols of the code are assigned different non-zero location numbers, the order of the error locations does not matter. For a single-error-correcting parity matrix, an efficient method is to rearrange the columns so each of the n error location positions is considered as a non-zero element in GF(2**m). Each element can be represented as a binary polynomial of

degree < m. Appendix A explains how to construct such a field. For example if m=4 and if the primitive polynomial is to be (x**4)+x+1, the parity check matrix for a single-error-correction code of block length 15 can be given as

(5.1)

$$M = \begin{vmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{vmatrix}$$

$$M = \begin{vmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 \\ w & w & w & w & w & w & w & w & w & w & w & w & w & w & w \end{vmatrix}$$

Since w is a root of a primitive polynomial, every non-zero element in GF(16) is a power of w. Therefore one can assign the successive digits of the error location positions to successive powers of w.

Such labeling also proves advantageous for multiple-error-correcting BCH codes. In order to correct additional errors, one needs additional information obtained by adding more rows to the parity check matrix. In the non-binary case, the parity check matrix is given by

(5.2)

$$M = \begin{vmatrix} 1 & w & w^2 & \cdots & w^{n-1} \\ 1 & w & (w^2) & \cdots & (w^{n-1})^2 \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & w & (w^2) & \cdots & (w^{n-1})^{2t} \end{vmatrix}$$

The first m digits of the parity check matrix give the first syndrome, the sum of the error locations, the second m digits give the second syndrome, the sum of the squares of

the error locations, etc. Note in the binary case, all even powers can be computed and need not be columns of the parity check matrix although there is no harm in including them.

$$(i + j)^2 = i^2 + 2ij + j^2 = i^2 + j^2 \mod 2 \tag{5.3}$$

$$\text{or } (S_1)^2 = S_2$$

Polynomial Division

The codeword viewed as a polynomial also assigns the successive digits of the error positions to successive powers of w. This assignment of location numbers has the great advantage that the syndrome of the received vector becomes a polynomial in w:

$$H R^T = \sum_{i=0}^{n-1} R_i (w^j)^i = R(w^j) = S_j \tag{5.4}$$

$$\text{for } 1 \leq j \leq d-1$$

Also viewing the calculation of syndromes as evaluation of polynomials enables one to apply many existing improved algorithms for polynomial operations.

Generally the fastest method of calculating the syndromes from the received vector R(x) is by hardware implementation. By the division algorithm,

$$R(x) = Q(x)M(i)(x) + r(x) \; ; \text{ degree } r(x) < M(i)(x) \tag{5.5}$$

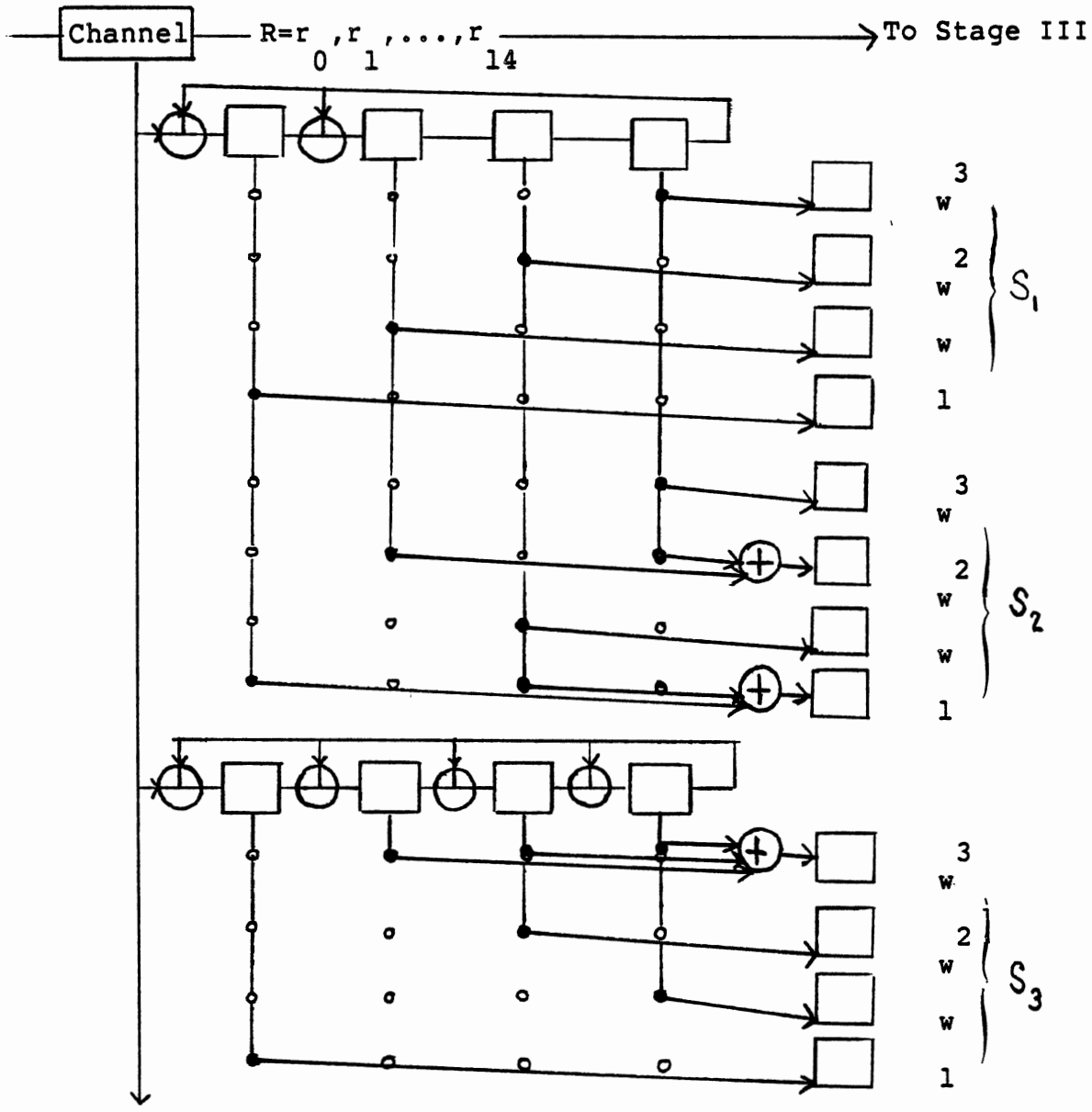
In a single-error-correcting codes G(x)=M(i)(x) or the

irreducible polynomial of degree m which is the minimal polynomial of w, the M(i)(w)=0 and R(x)=r(x). Thus after the division, the syndrome is given by the remainder polynomial r(x), evaluated at x=w. In multiple-error-correcting codes, these syndromes or power sums can be computed from the received word separately. To compute the first syndrome, one divides R(x) by M(1)(x), the minimal polynomial of w, to obtain the remainder of $r_1(x)$. To compute the i-th syndrome, one divides R(x) by M(i)(x), the minimal polynomial of w**i. Figure 8 is an example of circuitry needed for the first stage of the decoder for multiple-error-correcting BCH code defined on GF(16) defined by (x**4)+x+1 (see Appendix A).

One advantage of the polynomial representation over the matrix representation is the parity check matrix need not be stored therefore avoiding table-lookup. Another advantage aside from speed, is that the syndrome calculation can be implemented by the division algorithm used in the encoder thus a possible reduction in total hardware cost of the encoder/decoder system.

Direct Polynomial Evaluation

Other very economical methods include the direct calculation of the syndromes by evaluating the polynomial representation of the received word. Algorithms for evaluation of polynomials differ in the amount of computation required, the amount of storage required, and



$$S_3 = r(w^3) = r_0 + r_1(w^3) + r_2(w^3)^2 + r_3(w^3)^3$$

$$= r_0 + r_1 w^3 + r_2 w^6 + r_3 w^9$$

$$= r_0 + r_1 w^3 + r_2(w^2+w^3) + r_3(w+w^3)$$

$$= r_0 + r_3 w + r_2 w^2 + (r_1 + r_2 + r_3)w^3$$

Figure 8. Calculating the Syndromes by Division Circuitry

the effects of arithmetic roundoff. It is somewhat difficult to compare various algorithms because of the tradeoffs between these various factors that depend on the hardware of software that is available. However perhaps the most straightforward way to solve for the syndrome is to compute each term and add it to the sum of the others already computed. The program design logic and Figure 9 and example in Figure 10 evaluates polynomials using 2n-1 multiplications and n additions. However a more efficient algorithm exists. Figure 11 illustrates the program design logic for Horner's method of evaluating R(w**i) by using a simple factorization of R. This reduces the computations to n multiplications and n additions. Finally the polynomial evaluation must be repeated for successive powers of w depending on the error correcting capacity of the code.

Fast Fourier Transform

This repeated evaluation of roots of unity suggests the use of a transform method. These methods have proven to be useful when an application such as decoding allows sequences to be processed in blocks. The most versatile transform is the discrete Fourier transform (DFT) which has been defined in finite Galois fields (45) and much more familiarly in the complex number fields (17). A basic introduction and program design logic is given in Appendix C. Investigation into transforms defined in the arithmetic of finite fields developed so truncation and rounding effects when performing

```
Input:  The coefficients of P(x) in the array R
        X and N>=1
Output: S, the value of P(x)

     S <--  R(0) + R(1) * X
     XPOWER <-- X
     For I <-- 2 to N do
           XPOWER <-- XPOWER * X
     End
```

Figure 9. Polynomial Evaluation --Term by Term

Received word = (+8,+2,-1,+3,0,2,0,0,0,0,0,0,2,3,2,1)

$$S(x) = 8x^0 + 2x^1 - 1x^2 + 3x^3 + 2x^5 + 2x^{12} + 3x^{13} + 2x^{14} + 1x^{15}$$

$$S(w) = 8(1)+2(6)-(2)+3(12)+2(7)+2(13)+3(10)+2(9)+(3) \bmod 17 = 9$$

$$S(w^2) = 8(1)+2(2)-(4)+3(8)+2(15)+2(14)+3(11)+2(5)+(10) \bmod 17 = 4$$

$$S(w^3) = 8(1)+2(w^3)-(w^6)+3(w^9) \ldots = 0$$

$$S(w^4) = 4$$

Figure 10. Syndrome Calculation by Polynomial Evaluation

$$P(x) = [\ \ldots\ (\ (a_n x + a_{n-1})x + a_{n-2})x + \ldots + a_1]x + a_0$$

```
Input: The coefficients of P(x) in the array R
       X and N>=1
Output: S, the value of P(x)

     S <-- R(N)
     For I <-- N-1 to 0 by -1
          S <-- S * X + R(I)
```

Figure 11. Polynomial Evaluation--Horner's Method

polynomial operations by transform methods in the complex number field may be avoided. In any practical application, such as coding theory, the message symbols are from a finite field, and therefore without loss of generality, the data can be considered to be integers with some upper bound. By special choices of three requirements: 1) the length N, 2) the modulus F, and 3) the primitive root w, it is possible to develop improved transform algorithms known as number theoretic transforms (NTT).

The first improvement is the choice of n or the length of the polynomial evaluated. The best choice is for n to be a power of two. If n can not be a power of two, the disadvantage of such a transform method over GF(2**m) is that the transform length must be an odd number so that the most efficient fast Fourier transform (FFT) cannot be used. The next choice is values of n that are highly composite. Winograd suggested a method for computing transforms over GF(2**m) for larger values of n.

> Let $n = n_1 \cdot n_2$ where $(n_1, n_2) = 1$. Using the Chinese remainder theorem, one can represent every integer $i \in \{0,1,\ldots,n-1\}$ by a pair of integers (i_1, i_2)
>
> where $i_1 = i \bmod n_1$, $i_2 = i \bmod n_2$
>
> Consequently:
>
> $$w^{ij} = w^{(i_1,i_2)(j_1,j_2)} = w^{(i_1 j_1, i_2 j_2)} = w^{(i_1 j_1, 0)\,,\,(0, i_2 j_2)}$$
>
> This means that the computation of DFT of $n = n_1 * n_2$ points can be decomposed into computing the DFT for n points in which each multiplication is replaced by computing the DFT of n points.
> (57,p.1005)

To avoid direct computation of S , a n-point transform over GF(2**m) defined in equation (5.1) can be decomposed into a multidimensional transform over GF(2**m) as illustrated in Figure 12. However observe in Stage r, one needs only compute the first d-1 points of the transform. An example is a block length of n=255 over GF(2**8) is $n=n_1 n_2 n_3$ =17*5*3. Then by suitably applying the above technique for each factor of n, the original syndrome form can be reconstituted by using the Chinese remainder theorem as stated in Appendix B.

Another source of improvements to make these transforms computationally efficient is in the choice of modulo F. A systematic investigation of good choices of F, for which the maximum transform length of N is not too small reveals some interesting results. Of course, one would like F to have a minimal binary bit representation in order to facilitate arithmetic modulo F. The first possibility is 2**k; it has a prime factor two and therefore the maximum possible transform length is one. For (2**k)-1, let k be a composite PQ, where P is prime. Then (2**P)-1 divides (2**PQ)-1 and the maximum possible length of the transform will be governed by the length possible for (2**P)-1. Numbers of this form are known as Mersenne numbers. Mersenne number transforms are not highly composite, and therefore fast FFT-type computational algorithms do not exist to compute the transforms. For (2**k)+1, say k is odd, then three

Given integer $j = (j_1, j_2, \ldots, j_r)$ where $j_k = j \bmod n_k$ for $1 <= k <= r$

then $w_k = w^{(0,0\ldots0,1,0\ldots0)}$ where 1 is the kth position

w is the primitive n_k th root of unity

then

$$S_j = S_{(j_1, j_2, \ldots, j_r)} \quad \text{for } (1 <= j <= r)$$

$$= \sum_{i_1=0}^{n_1-1} \sum_{i_2=0}^{n_2-1} \cdots \sum_{i_r=0}^{n_r-1} a_{(i_1, i_2, \ldots, i_r)} w_1^{i_1 j_1} w_2^{i_2 j_2} \cdots w_r^{i_r j_r}$$

The r stages

Stage 1

$$S^1_{(i_1, i_2 \ldots i_{r-1} j_r)} = \sum_{i_r=0}^{n_r-1} a_{(i_1, i_2 \ldots i_r)} w_r^{i_r j_r} \quad \text{for } 0 <= j_r <= n_r - 1$$

Stage 2

$$S^2_{(i_1, i_2 \ldots j_{r-1} j_r)} = \sum_{i_{r-1}=0}^{n_{r-1}-1} S^1_{(i_1, i_2 \ldots j_r)} w_{r-1}^{i_{r-1} j_{r-1}} \quad \text{for } 0 <= j_{r-1} <= n_{r-1} - 1$$

.
.
.

Stage r

$$S^r_{(j_1 \ldots j_r)} = S_j = \sum_{i_1=0}^{n_1-1} S^{r-1}_{(i_1, j_2 \ldots j_r)} w_1^{i_1 j_1} \quad \text{for } 1 <= j <= d-1$$

Figure 12. Multidimensional Transform Over GF(2**m)

divides (2**k)+1 and the largest possible transform length is two. Thus k is even. Integers of the form (2**b)+1 where b is (2**t) and t is a positive integer are known as Fermat numbers. "Fermat numbers seem to be optimal in the sense of having transforms whose length is interesting while word size is moderate." (1,p.90) Ft is called the t-th Fermat number, and it seems only the first five are prime and all the others are composite. The first few values are

(5.6)

F0 =3
F1 = 5
F2 = 17
F3 = 257
F4 = 65 537

$$Ft = 2^{2^t} + 1 = 2^b + 1$$

where b= 2^t

F5 = 4 294 967 297 =641 * 6 700 417
F6 = 274 177 * 67 280 421 310 721

Number theoretic transforms with a Fermat number as a modulus are called Fermat number transforms (FNT).

In software or hardware realization of the FNT, one ordinarily defines a binary arithmetic modulo Ft. The utilization of such a modulus requires b+1 bits. The representation of the quantity 2**b=-1 modulo Ft requires the (b+1)th bit. In order to simplify modular arithmetic operations, one can limit the realization of the FNT to b-bit arithmetic. This is possible especially in an application such as coding theory. No input bits will be in error due to the message bits being defined on a b-bit representation. However there may be some error in representing the redundant check bits. Realistically, the probability that this number will appear after the

arithmetical operations during the transform is approximately 2**-b. If an occasional error is permissible, for these cases, probablily there is no need for any extra hardware to represent 2**b. If the need exists, an extra bit could be used to represent 2**b at the expense of a more complicated hardware.

The following discussion is based on the b-bit representation of integers by Agarwal and Burrus (1). Another binary arithmetic for the Fermat number transform is suggested by Leibowitz (26) which is only mentioned here as a reference to indicate that other possible implementations exist. Various basic arithmetic examples are illustrated in Figure 13. To negate a number, one has to complement each bit and add two to the result. When one adds two b-bit integers, one obtains a b-bit integer and possibly a carry bit. The carry bit represents 2**b=-1 modulo a Fermat prime. To implement arithmetic modulo Ft, one adds then subtracts the carry bit. Subtraction is implemented as an addition by first negating the subtrahend and then adding the two b-bit integers. When one multiplies two b-bit integers in general, one gets a 2b-bit product. Let C-low be the b-bit low-order part of the product and C-high be the b-bit high order part. Thus all one has to do is subtract the high order register from the low order register. When multipling by a power of 2, these computations are particularly simple to implement in arithmetic modulus the Fermat number. All one needs to do is left-shift the

A) Negation

(mod 17): 4 = 0100; -4 = 1011 = 1101 = 13

```
    1011
+     10
------
    1101
```

B) Addition

(mod 17): 10 + 9 = 17 = 2 mod 17

```
  1010
 +1001
------
 10011
 -   1
------
  0010  = 2
```

C) Subtraction

(mod 17): 10 - 4 = 6

```
  1010
 +1101
------
 10111
 -   1
------
  0110  = 6
```

D) Multiplication

(mod 17): 13 * 9 = 117 = 15 mod 17 = 0111 0101

High Low

```
    Low  0101
(-High)  1010
         ----
         1111 = 15
```

E) Multiplication By Power of Two

(mod 17): 11 * (2**3) = 88 = 3 mod 17

```
11 = 0000 1011              Low 1000
Shift left 3 positions  (-High) 1100
                                ----
     0101 1000                 10100
     High  Low                 -   1
                               ------
                                0011 = 3
```

Figure 13. Arithmetical Operations Performed Modulus Fermat Prime

contents of the register by k bits and subtract the k overflow bits that are in the high register. Division in a finite field is multiplication by its inverse.

Finally, since multiplications take most of the computational effort in calculating the FFT, it is important that the multiplication by powers of w be a simple operation. This is possible if the powers of w have very few bit binary representation; preferably also a power of two, where multiplication by a power of w reduces to a word shift. With this in mind, one considers transform lengths possible in arithmetic modulo various Fermat numbers with the corresponding values of the root of unity. Since Fermat numbers up to F4 are prime, one can have an FNT for any length N=2**m, m<=b. For these Fermat primes the integer 3 is an w of order N=2**b, allowing the largest possible transform length. However there are other integers which are of order 2**b. If w is taken as 2 or a power of 2, all the powers of w would be some power of 2 and for these cases, the FNT can be computed very efficiently. These transforms are called Rader transforms (49).

For a better understanding of these prime moduli consider an example for F2. If the modulus is M=17 the 3 and 6 are primitive roots that will generate the entire field as shown in Appendix A. The value 2 is of order 8 and 4 is of order 4. Also note that $6=\sqrt{2}$ in the same sense that (6**2)=2(mod17). Other excellent choices of the root of unity are listed in Figure 14.

F2 and N=2**4

$\sqrt{2} = 2^{3} - 2^{1} = 6$

F3 and N=2**5

$\sqrt{2} = 2^{6} - 2^{2} = 60$

F3 and N=2**6

$\sqrt[4]{2} = 35$

F3 and N=2**7

$\sqrt[8]{2} = 42$

F4 and N=2**6

$\sqrt{2} = 2^{12} - 2^{4}$

F4 and N=2**7

$\sqrt[4]{2} = 4938$

F4 and N=2**8

$\sqrt[8]{2} = 5574$

Figure 14. Roots of Unity

$$R(x) = (+8,+2,16,+3,0,2,0,0,0,0,0,0,2,3,2,1)$$

$$R(w^{i}) = (5,9,4,0,4,1,15,8,0,6,1,8,14,1,4,14)$$

Figure 15. Calculating the Syndromes by Transform Method

TABLE I

PARAMETERS FOR SEVERAL POSSIBLE IMPLEMENTATIONS FOR FNT

t	b	Ft	N;w=2	N;w=$\sqrt{2}$	N Max	w for N Max
3	8	2**8 +1	16	32	256	3
4	16	2**16+1	32	64	65536	3
5	32	2**32+1	64	128	128	$\sqrt{2}$
6	64	2**64+1	128	256	256	$\sqrt{2}$

In summary, Table I gives values of N for the two most important values of w, and also gives the maximum possible N for the most important values of b. However, the parameters chosen depends on the application, but hopefully this introduction indicates careful study can improve numerical complexity considerably. Continuing the example in the previous chapters, Figure 15 finds the syndromes by using a fast Fourier transform with its improvements. The program design logic and numerical complexity for a general fast Fourier transform is included in Appendix C since this algorithm is also used in Stage III.

Summary

The final results depend on the application. If small values of n and t are used, the implementation by linear feedback shift registers or direct polynomial evaluation still may be feasible. However as the complexity of the code increases the new transform method has been shown (51) to reduce numerical complexity substantially. This method applies to either a software or hardware implementation.

However by the end of Stage I regardless of method implemented, the decoder has found the syndromes

$$S_1, S_2, \ldots, S_{d-1} = \sum_{i=1}^{d-1} Y_i X_i^k \tag{5.7}$$

Thus the added error can be described by a vector of values and locations of its nonzero components. The location

will be given in terms of an error location value, which is simply w**(i-1) for the i-th symbol. Thus each nonzero component of the error vector is described by a pair of field elements, Y_i (the value of the component) and X_i (the error location number). Y is an element of GF(p) and X is an element of GF(p**m). The syndromes are calculated from the received vector, and in order to correct the errors, the pair (Y_i, X_i) must be found for each of the t or fewer errors. The syndromes are called the weighted power sum symmetric functions. The syndromes consists of a set of t equations in t unknowns. Any method of solving these equations is the basis an error-correction procedure (43). It appears impossible to solve the equations by any direct method, and trying all combinations of t of the q field would require too many computations. There is however, an interesting solution which is the next step in the decoding process.

CHAPTER VI

STAGE II: CALCULATION OF THE ERROR LOCATOR AND THE ERROR EVALUATOR POLYNOMIALS

A major stage in a typical decoding procedure for Bose-Chaudhuri-Hocquenghem (BCH) codes or Reed-Solomon (RS) codes is the calculation of the error locator and the error evaluator polynomials (44, 8). This chapter uses the syndromes calculated in Stage I to determine these two polynomials that facilitate in finding the location and the values of the errors in Stage III. The calculation of these polynomials is the most complex stage of the decoding procedure. Several methods are introduced, however, if more details are desired, references will be cited.

Simultaneous Equations

At the end of Stage I, the syndromes of the received vector were calculated. With the usual notation, one defines

(6.1)

$$\begin{array}{lll} \text{Received vector} & = \text{Code vector} & + \quad \text{Error vector} \\ R(x) = \sum_i r_i x^i & C(x) = \sum_i c_i x^i & E(x) = \sum_i e_i x^i \end{array}$$

If the error word consists of an error of value Y at

location X_1 and an error of value Y at location X , ..., then the syndrome is defined

$$E(w^j) = \sum_i Y_i X_i^j = S_j \tag{6.2}$$

In general, a solution to the system of equations in Equation 6.2 is the basis for the error-correction procedure. This stage is complicated by the fact that these non-linear equations will have many solutions. Using a vector representation and maximum likehood decoding (23, 8) seems impossible. Each solution corresponds to different error patterns in the same coset of the additive group of codewords (52, 8). The decoder must find a solution where the error vector has as small a weight v as possible. The weight of a vector is defined as the number of non-zero elements in that vector (23). There are only a finite number of possible solutions, and the correct solution could be found by simply trying all possible solutions. In practical application, however, there are simply too many possible solutions for this to be an effective method. There is, however, an effective compromise.

Suppose that v<=t errors actually occur. These are described by v pairs (Y_i, X_i), for which neither Y nor X is zero. In order to make a total of t pairs, one can add t-v pairs of zeros, that is

$$X_i = Y_i = 0 \quad \text{for } v < i <= t \tag{6.3}$$

Then let the equation

$$(X-X_1)(X-X_2)\ldots(X-X_t) = \sigma_t - \sigma_{t-1}(X) + \ldots + \sigma_1(X)^{t-1} + (X)^t \tag{6.4}$$

define the quantities $\sigma_1, \sigma_2, \ldots, \sigma_t$. These are the elementary symmetric functions defined in more detail in Appendix D. Then if x is substituted for X in Equation 6.4, both sides are zero. This is also true if both sides are multiplied by $Y_i X_i$. Thus relating the S's and the σ's gives

$$S_j \sigma_t - S_{j+1} \sigma_{t-1} + \ldots + (-1)^{t-1} S_{j+t-1} \sigma_1 + (-1)^t S_{j+t} = 0 \tag{6.5}$$

which must hold for all j. Since S is found from the parity check calculations for $1 \le j \le 2t-1$, a set of t equations in which all the S are known can be found: j=1 in the first and j=t-1 in the last (42).

Before presenting more complex theory, a basic example is necessary. For the binary case, the Y_i which can not be zero, must be one. Therefore

$$S_j = \sum_i X_i^j \tag{6.6}$$

Thus the parity checks give the first t odd power-sum symmetric functions. The proof that it is indeed possible to solve for the elementary symmetric functions from the power-sum symmetric functions is given by Theorem 1 in Figure 16. For example, a double-error-correcting code in

GF(2) has the following solutions illustrated in Figure 17. This figure gives a possible decoding scheme to use in order to keep decoding complexity to a minimum. This scheme is very practical especially if retransmission is possible when more than two errors are detected. In fact E. R. Berlekamp (8) has implemented a complete decoding scheme for double error correcting binary BCH codes.

In general, the following is an iterative algorithm for finding $\sigma(z)$, for a BCH code of designed distance d, assuming v errors occur where v<=t. The t-error correcting BCH code give, as the parity check on received sequences, the odd power-sum symmetric functions up to S_{2t-1} and the intermediate even functions can be calculated simply from these. If it is assumed that no more than t errors occur, then by Theorem 1 in Figure 16, with k=t, it is either possible to solve for the error position numbers, or there are t-2 or fewer errors. In the latter case $\sigma_{t-1}=\sigma_t=0$ and two equations can be dropped, giving a set of t-2 equations in t-2 unknowns to which Theorem 1 can be applied again. Eventually, if there were any errors at all, a set of equations that can be solved for the elementary symmetric functions of the error positions will be found (42).

This step involves a certain amount of trial and error because it is possible to solve the equations and obtain correct solutions only when the number of equations used equals or exceeds by one the number of errors that actually occur. This step might be carried out instead by starting

The k*k Matrix

$$M_k = \begin{vmatrix} 1 & 0 & 0 & 0 & 0 & \cdots & 0 \\ S_2 & S_1 & 1 & 0 & 0 & \cdots & 0 \\ S_4 & S_3 & S_2 & S_1 & 1 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ S_{2k-4} & S_{2k-5} & \cdots & & & & S_{k-3} \\ S_{2k-2} & S_{2k-3} & \cdots & & & & S_{k-1} \end{vmatrix}$$

Figure 16. Theorem 1

$$S_1 - \sigma_1 = 0$$

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

If $S_1 = S_3 = 0$; no errors have occurred

If $S_1 \neq 0$, $S_3 = S_1^3$; correct single error $X_1 = S_1$

If $S_1 \neq 0$ and $\mathrm{Tr}\left(\frac{S_1^3 + S_1 S_2}{S_1^3}\right) = 0$; Correct double error

$$\sigma(x) = (X - X_1)(X - X_2)$$

$$\sigma(z) = 1 + \sigma_1 z + \sigma_2 z \qquad \text{where } z = 1/x$$

$$= 1 + (S_1)z + \frac{S_3}{S_1} + S_1^2 z^2$$

If $S_1 = 0$ and $\mathrm{Tr}\left(\frac{S_1^3 + S_1 S_2}{S_1^3}\right) = 0$ or $S_3 + S_1 S_2 \neq 0$;

More than two errors have occurred

Figure 17. Decoding Scheme for Double-Error-Correcting BCH Code

with the assumption that two errors occurred, 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 occur, it must be the correct solution. This step amounts to solving, at most, a set of t simultaneous linear equations. But one may have to try t-1 times since the actual number of errors present may be anywhere from one to t. Therefore this method is not practical unless t is small, but introduced because of its simplicity. Also this error-correction procedure has advantages for binary codes, but apparently can not be generalized for q-ary codes.

Background Theory

However much research as been done on solving these simultaneous equations. It turns out to be advantageous to replace the ordering of the position locations and define the error locator polynomial $\sigma(z)$ based on the elementary symmetric functions in Equation 6.4 as follows

(6.7)

For $(w_0^{-1}, w_1^{-1}, \ldots, w_n^{-1})$ then

$$\sigma(z) = \prod_{i=1}^{v} (1 - X_i z)$$

$$= \sum_{i=0}^{v} \sigma_i z^i \; ; \; \sigma_0 = 1$$

$$= 1 + \sigma_1 z + \ldots + \sigma_v z^v$$

Therefore the roots of $\sigma(z)$ are the reciprocals of the locations or the X_i values.

Once the position of the error has been determined, then it is necessary to determine the value of the error in order for correction to take place. Therefore if $E_i=0$, then an error has occurred in position $X_i z$, and $V=\{w_i : e_i \neq 0\}$. The set of error locations and the number of elements in V will be denoted by v. Using the above notation, one can rewrite the syndrome vector S(x).

$$S(x) = - \sum_{i=1}^{v} \frac{Y_i X_i z}{1 - X_i z} \tag{6.8}$$

The syndrome polynomial can be expressed in this form due to the reordering of the error positions. Figure 18 is provided to aide in the theoretical background if necessary or Berlekamp (8) is an excellent reference.

Therefore the error evaluator polynomial is derived by multiplying the error locator polynomial and the syndrome polynomial as indicated in Equation 6.5. Thus relating the S's and the σ's gives the following equations defined in Figure 19. Now observe that if one could somehow find the polynomials $\sigma(z)$ and $\omega(z)$, one could recover the transmitted codeword C from the received word R. Of course there are efficient algorithms for computing $\sigma(z)$ and $\omega(z)$, which are based on the key equations in Figure 19. That is the solution can be found provided that one makes the additional assumption that v, the number of errors that actually

$$S_j = \sum_{i=0}^{n-1} R_i w_i^{-j} \qquad \text{for } j=1,2,\ldots,2t \text{ and}$$

$$\text{and } w_i = (w_0^{-1}, w_1^{-1}, \ldots, w_{n-1}^{-1})$$

$$S(x) = S_1 + S_2 x + \ldots + S_{2t} x^{2t-1} + \ldots$$

$$= \sum_{j=1}^{2t} S_j x^{j-1}$$

$$= \sum_{j=1}^{2t} x^{j-i} \sum_{i=0}^{n-1} R_i w_i^{-j}$$

$$= \sum_{i=0}^{n-1} R_i \sum_{j=1}^{2t} x^{j-1} w_i^{-j}$$

$$= \sum_{i=0}^{n-1} R_i \frac{x^{2t} w^{-2t} - 1}{(x - w^{-i})}$$

$$= \sum_{i=0}^{n-1} R_i \frac{-1}{x - w^{-i}} \mod x^{2t}$$

$$= -\sum_{i=0}^{n-1} \frac{E}{x - w^{-i}} \mod x^{2t}$$

Figure 18. Definition of Syndrome

Key Equation in General Theory

$$S(z)\sigma(z) = -\sum_{i=1}^{v} \frac{Y_i X_i z}{1 - X_i z} \prod_{j=1}^{v} (1 - X_j z)$$

$$= -\sum_{i=1}^{v} Y_i X_i z \prod_{j \neq i}^{v} (1 - X_j z)$$

$$= \omega(z)$$

Key Equation in Berlekamp Algorithm

-- For ease of computation add $\sigma(z)$ to both sides

-- Reduce modulus z^{2t+1} since the decoder only knows the first 2t powers of z

$$\sigma(z) + S(z)\sigma(z) = [\sigma(z) - \sum_{i=1}^{v} Y_i X_i z \prod_{j \neq i}^{v} (1 - X_j z)]$$

$$(1 + S(z))\sigma(z) = \omega(z) \bmod z^{2t+1}$$

Figure 19. Key Equations of Decoding BCH codes

occurred, satisfies v<=t or v<=[d-1/2]. Notice that the error evaluator polynomial $\omega(z)$ depends both on the locations and the values of the errors, although the error locator $\sigma(z)$ depends only on the locations of the errors.

Therefore by defining error locator and error evaluator polynomials, one defines the generator function of the sequence. Since $\sigma_0=1$, the generator function for the quotient $\omega(z)/\sigma(z)$ is well defined. Therefore there are basically two approaches to find $\sigma(z)$ and $\omega(z)$ based on the key equation derived by the elementary symmetric functions. The solution can be obtained by using continued fractions (32, 55) or by using Berlekamp's algorithm to solve the key equation (8, 34).

Euclidean Algorithm

From the theoretical standpoint, both approaches use the Euclidean algorithm given in Appendix E. The Euclidean algorithm is used to prove that the factorization of polynomials into irreducible polynomials is unique (except for scalar multiples) over any field and that a polynomial of degree d can not have more than d roots in any field. This fact is needed to prove that the error locator polynomial $\sigma(z)$ cannot have more roots than its degree. If it did, then the entire decoding procedure would be invalid, for several different pairs of error locations might conceivably be reciprocal roots of the same equation.

From the practical standpoint, the Euclidean algorithm is a simple and straightforward algorithm for finding the greatest common divisor (gcd) between two integers or two polynomials, or for finding the continued fractions expansion of a real number. Relative to decoding RS codes, the Euclidean algorithm is important because of one of its modifications. The method of convergents of continued fractions provides the basis for one of the most efficient methods for implementing division in finite fields. Thus in the decoding process $\sigma(z)$ and $\omega(z)$ can be found merely by applying Euclidean algorithm to x**2t and S(x) and stopping at the index i as soon as the degree of the remainder drops below t. Thus setting

(6.9)

$$\sigma(z) = t_i(z) \quad \text{and} \quad \omega(z) = r_i(z)$$

Many algorithms are known for computing greatest common denominator. A survey of classical techniques for gcd's was conducted by D. E. Knuth (55). An excellent algorithm is given in Aho, Hopcroft, and Ullman (4). A generally accepted method is E. R. Berlekamp's iterative algorithm using continued fractions in GF(p**m). This algorithm can be easily implemented on the computer (8). This algorithm is the one used to define the Euclidean algorithm in Appendix E. Table II illustrates the example of finding the polynomials by the Euclidean Algorithm.

TABLE II

CALCULATING THE POLYNOMIALS BY
THE EUCLIDEAN ALGORITHM

i	s_i	t_i	r_i	q_i
-1	1	0	x^4	. . .
0	0	1	$4x^3 + 4x + 9$	. . .
1	1	$4x$	$16x^2 + 2x$	$13x$
2	$4x + 8$	$16x^2 + 15x + 1$	$3x + 9$	$13x + 9$
3	$7x^2 + 13x + 16$	$11x^3 + 1x^2 + 2x + 4$	2	$11x + 13$
4	$4x^3 + 4x + 9$	x^4	0	$10x + 13$

TABLE III

CALCULATING THE POLYNOMIALS BY
THE BERLEKAMP ALGORITHM

$1 + S = 1 + 9z + 4z^2 + 0z^3 + 4z^4 + \ldots$

k	$D(k)$	$B(k)$	$\sigma(k)$	$\tau(k)$	$\omega(k)$	$\gamma(k)$	$\Delta(k)$
0	0	0	1	1	1	0	9
1	1	1	$1 + 8z$	2	1	2	8
2	1	0	$1 + 9z$	$15+z$	$1 + z$	15	2
3	2	1	$1+13z+15z^2$	$9+13z$	$1+5z$	$9+9z$	13
4	2	0	$1+15z+16z^2$	$4+1z+9z^2$	$1+7z+2z^2$	$4+3z$	Stop

Berlekamp Algorithm

The last method of locating and evaluating the errors is also based on using the generalized Newton's identities. This method is known as the Berlekamp algorithm for solving the key equation. For a more theoretical background and a heuristic solution of the key equation in more detail, one can refer to the original work by E.R. Berlekamp (8). The decoder can solve the key equation given in Figure 19 by using the program design logic given in Figure 20. This algorithm is the complete second stage of the decoder. Given S(z), one can find both $\sigma(z)$ and $\omega(z)$ from these equations. The unknown polynomials $\sigma(z)$ and $\omega(z)$ both have degrees <= v, the number of errors that actually occurred. The algorithm proceeds recursively and includes many conditions that ensures the smallest degree polynomials are found. Table III illustrates the example of finding the polynomials by the Berlekamp algorithm.

Summary

All the methods presented form the theoretical background for Berlekamp's algorithm. This algorithm is basically an improvement by Berlekamp on his own continued fractions algorithm. However any of the methods presented can be used to find the error locator and error evaluator polynomials. In comparison, Aho, Hopcroft, and Ullman (4) describe an algorithm which computes the greatest common divisor of two polynomials of degree n in Order ($n\log^2 n$)

Initially define: $\sigma(0)=1$ $\tau(0)=1$ $\omega(0)=1$
$\gamma(0)=0$ $D(0)=0$ $B(0)=0$

Proceed recursively as follows:

If S_{k+1} is unknown, stop; Otherwise

Define $\Delta_1(k)$ and the coefficient of z^{k+1} in the product $(1+S)$ and $\sigma(k)$

$$\sigma(k+1) = \sigma(k) - \Delta(k)*z*\tau(k)$$

$$\omega(k+1) = \omega(k) - \Delta(k)*z*\gamma(k)$$

If $[\Delta(k)=0] \mid [D(k)>(k+1)/2] \mid [\Delta(k)=0 \;\&\; D(k)=(k+1)/2 \;\&\; B(k)=0]$

$$D(k+1) = D(k)$$

$$B(k+1) = B(k)$$

$$\tau(k+1) = z*\tau(k)$$

$$\gamma(k+1) = z*\gamma(k)$$

But if $[\Delta(k)=0] \;\&\; [(D(k)<(k+1)/2) \mid (D(k)=(k+1)/2) \;\&\; B(k)=1)]$

$$D(k+1) = k + 1 - D(k)$$

$$B(k+1) = 1 - B(k)$$

$$\tau(k+1) = \sigma(k)/\Delta_1(k)$$

$$\gamma(k+1) = \omega(k)/\Delta_1(k)$$

Figure 20. Berlekamp Algorithm

steps. By using this modified version of the Euclidean algorithm, Justesen (25) shows that a t-error-correcting Reed-Solomon code of length n can be decoded in Order ($n\log^2 n$) arithmetic operations.

Similarly, a primitive binary BCH code of length n can be decoded up to its designed distance in Order(nlogn) arithmetic operations. These results are better than those obtained with the Euclidean algorithm, but unfortunately only for excessively large values of n. For practical purposes the original version of the Berlekamp algorithm is probably the fastest, although this depends on the machinery available for the decoding. Nevertheless, decoding using the Euclidean algorithm is by far the simplest to understand, and is certainly at least comparable in speed with the other methods (for n<10**6) (38). However, by the end of Stage II regardless of method implemented, the decoder has found the error locator and the error evaluator polynomials.

CHAPTER VII

STAGE III: EVALUATION OF ERROR LOCATOR AND ERROR EVALUATOR POLYNOMIALS

The final stage in a typical decoding procedure for Bose-Chaudhuri-Hocquenghem (BCH) codes or Reed-Solomon (RS) codes is the evaluation of the error locator polynomial and error evaluator polynomial. This stage can be divided essentially into two steps: 1) finding the error locations and 2) finding the error values if necessary. Once the roots of the error locator polynomial are found, the location of the errors are known. Generally, there are several methods that find the roots of the error locator polynomial. By Peterson's decoding procedure, each nonzero element of the field is generated and substituted in a trial and error search for the roots. This stage turned out to be the most time consuming. This chapter will discuss some of the solutions found to reduce the numerical complexity of Stage III. One solution was suggested by R.T. Chien (15) and another solution is based on transform methods. However even though the decoder knows the error locations, it is difficult to correct the errors immediately because the values of the errors still needs to be determined. It turns out to be simpler to wait until the erroneous symbols leave

the received word buffer, and then correct the errors as they leave. Thus the decoding procedure is complete.

Factoring Polynomials

The location of the errors depends on the roots of the error locator polynomial, $\sigma(z)$. If $\sigma(z)$ has degree one or two, the zeros can be found directly. Over GF(2**m) quadratic equations can be solved almost as easily as linear equations. The following references on factoring polynomials over finite fields are relevant: Berlekamp (6, 7, 8), Chien et al. (14), and McEliece (37).

Chien Search

In general the simplest technique is just to test each power of w in turn to see if it is a zero of the error locator polynomial shown in Equation (6.7). This part of the decoding is often called the Chien search. This approach avoids the explicit solution of the error locator polynomial $\sigma(z)$, whose roots are the reciprocal of the error locations. The Chien search may be used to test each of the locations to see if the symbol in the position X_i now leaving the buffer is a reciprocal root of the $\sigma(z)$.

By examining the relationship between the coefficients of $\sigma(z)$ and the roots of this polynomial, Chien observed that the "coefficients are homogeneous sums of square free products of the roots of order d-1." (15,p.361) He used this homogeneous property to develop a way to obtain all

the roots of (z) by counting and by successive transformations. To simplify circuitry, the element to be detected is chosen to be the unit element of GF(2**m), one sees that

$$\sigma(z) = 1 + \sigma_1 z^1 + \sigma_2 z^2 + \ldots \sigma_t z^t = 0 \tag{7.1}$$

or

$$\sum_{k=1}^{t} \sigma_k z^{i=} \sigma_1 z^1 + \sigma_2 z^2 + \ldots + \sigma_t z^t = 1$$

If the transformation equals one after $i_1, i_2, \ldots$ shifts, respectively, the roots of (z) are w**(n-i_1), w**(n-i_2), ... In Figure 21, the procedure is illustrated with a binary example defined on GF(2**4) as defined in Appendix A. Each successive transformation is listed.

The implementation of the error correction procedure for binary codes follows the above theory in a straightforward manner. Once the decoder has found the coefficients $\sigma_1, \sigma_2, \ldots, \sigma_t$, the Chien search proceeds as follows. First the decoder computes (w). Next, the decoder computes (w**2), then (w**3) .... In order to calculate these polynomials quickly, the decoder uses t+1 registers. At the k-th step these registers contain the quantities

(7.2)

Register	0	1	2	3	...	t
Contents	1	$\sigma_1 w^{1k}$	$\sigma_2 w^{2k}$	$\sigma_3 w^{3k}$		$\sigma_t w^{tk}$

In order to proceed to the (k+1) step, the decoder

C(x) = (0 1 1 0 0 1 1 0 0 0 0 1 1 0 1)

E(x) = (0 0 0 0 1 0 0 0 0 1 0 0 0 0 0)

R(x) = (0 1 1 0 1 1 1 0 0 1 0 1 1 0 1)

STAGE I: $S_1 = w^{14}$ $S_3 = 0$

STAGE II:

$$\sigma(z) = 1 + w^{14} z + w^{13} z^2$$

$$\sigma_1 = {}^{14} = 1 + w^3 \qquad \sigma_2 = w^{13} = 1 + w^2 + w^3$$

STAGE III:

	$\sigma_1 * w$	$\sigma_2 * w^2$		
Initial Value	$1 + w^3$	$1 + w^2 + w^3$	$\neq 1$	
After 1 shift	1	1	$\neq 1$	
2 shifts	w	w^2	$\neq 1$	
3 shifts	w^2	$1 + w^3$	$\neq 1$	
4 shifts	w^3	$w^2 + w^3$	$\neq 1$	
5 shifts	$1 + w^3$	$1 + w^2$	$\neq 1$	
6 shifts	$w + w^2$	$1 + w + w^2$	$= 1$	root
.....				
10 shifts	$w^3 + w^4$	w^3	$\neq 1$	
11 shifts	$1 + w + w^2$	$w^2 + w^4$	$= 1$	root
.....				

Roots $w^{15-6} = w^9$ and $w^{15-11} = w^4$

E(x) = (0 0 0 0 1 0 0 0 0 1 0 0 0 0 0)

Figure 21. Decoding Reed-Solomon Code (15,11,5) Over GF(16) By Chien Search

multiplies the register 0 by (w**0), register 1 by (w**1), register 2 by (w**2), certain relatively small values of t and m, it is feasible to build circuitry to multiply the registers by wired constants in a single clock cycle. For the binary example in Figure 21, the Chien search may be accomplished by the circuit of Figure 22. Initially, the top register of the circuit of Figure 22 is loaded with σ_2 and the bottom register with σ_1. At each clock cycle, the top register is multiplied by (w**2) and the bottom register by (w). After k clock cycles, the adders evaluate the polynomial (w**k). If this polynomial is zero, then (w**-k) is a reciprocal root of the error polynomial, and a one is added into the erroneous symbol at location (w**-k) which is now leaving the buffer. If (w**k) does not equal zero, then the symbol leaving the buffer remains unchanged because it is not in error.

For larger values of t and m, the cost of building wired circuitry to multiply by (w**t) in GF(2**m) in one clock cycle becomes substantial. For moderate values of t, one may multiply by (w**t) by executing t successive multiplications by w. This method requires only registers wired to multiply by w, but it requires too many shifts if t is large. In these cases, it is usually more economical to allow m clock cycles for each of the multiplications. Figure 23 illustrates how to evaluate the error locator polynomial calculated in Table II and III by Chien search.

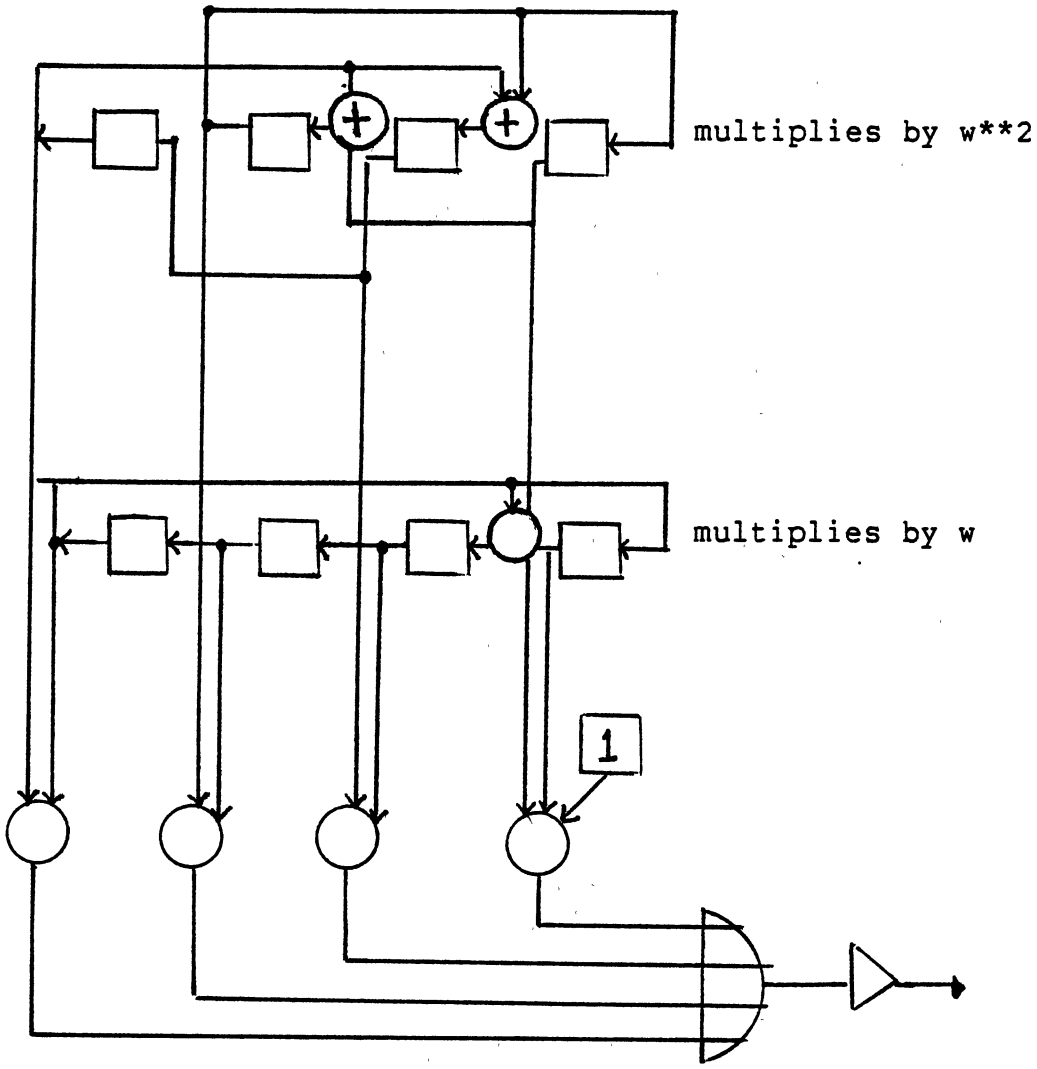


Figure 22. Chien Searcher for Double-Error-Correcting Binary BCH Code

$$\sigma(6^{13}) = \sigma(10) = 1+15z+16z^2 = 1+15(10)+16(10)^2 = 0 \bmod 17$$
$$\sigma(6^{11}) = \sigma(5) = 1 + 15(5) + 16(5)^2 = 0 \bmod 17$$

Therefore errors at $6^{16-13} = 6^3 = 12$ and $6^{16-11} = 6^5 = 7$

Figure 23. Evaluating the Error Locator Polynomial By Chien Search

Euclidean Algorithm	Berlekamp Algorithm
$w = 9 + 3x$	$w = 1 + 7x + 2z^2$
Reciprocal of $w = 9x + 3$	Reciprocal of $w = x^2 + 7z + 2$
$Y_1 = \frac{9(12) + 3}{12\ (12-7)} = 1$	$Y_1 = \frac{(12)^2 + 7(12) + 2}{12\ (12-7)} = 1$
$Y_2 = \frac{9(7) + 3}{7\ (7-12)} = 2$	$Y_2 = \frac{(7)^2 + 7(7) + 2}{7\ (7-12)} = 2$

Figure 24. Evaluating the Error Evaluator Polynomial

Given $S_{j+2} - \sigma_1 S_{j+1} + \sigma_2 S_j = 0$ or $1 + 15z + 16z^2$

Calculate the remainder of the Syndromes and Perform Inverse Fast Fourier Transform

S(x)=(3, 9, 4, 0, 4, 8, 3, 14, 14, 8, 13, 0, 13, 9, 14, 3)
E(x)=(0, 0, 0, 1, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0)

Figure 25. Evaluating the Error Locator Polynomial By Inverse Fast Fourier Transform

Location Values

The final step of the decoder is to determine the values of the errors once the error locations are known. This step is unnecessary in the binary case. Since Y_i is not zero it must be one. The location of the error is all one needs to know in order to correct it, and thus the error pattern in described completely by a list of error locations values X .

In the non-binary cases, when a zero of $\sigma(z)$ is found, indicating the presence of an error, the value of the error must be calculated as illustrated in Figure 24. The decoder can evaluate the polynomial $\omega(z)$ obtaining

(7.3)

$$\omega(z) = \omega(X_i^{-1}) = Y_i \prod_{j \neq i} (1 - X_j X_i^{-1})$$

Thus the decoder can evaluate the errors according to the formula

(7.4)

$$Y_i = \frac{\omega(X_i^{-1})}{\prod_{j \neq i} (1 - X_j X_i)} = \frac{X_i \omega(X_i^{-1})}{X_i \prod_{j \neq i} (X_i - X_j)} = \frac{\text{reciprocal of } (\omega(X^{-1}))}{X_i \prod_{j \neq i} (X_i - X_j)} = \frac{-X_i \omega(X_i^{-1})}{\sigma'(X_i)}$$

Inverse Fast Fourier Transform

Current research has shown improvements in Stage III that eliminates the need for the Chien search and the error evaluator polynomial. Since the S 's and σ's in Stage II

are related by a set of simultaneous linear equations, then for all j, the syndromes S satisfy the recurrence

(7.5)

$$S_{j+v} - \sigma_1 S_{j+v-1} + \ldots + (-1)^v \sigma_v S_j = 0$$

Thus upon completion of Stage II, then compute the remaining syndromes S for d<=k<=n from the known (z). Every RS code is a field generated by some polynomial G(x), i.e. a polynomial is a codeword if and only if it is divisible by G(x). This means that a vector is a codeword if and only if it satisfies the recursion relation corresponding to the polynomial (x**n-1)/G(x).

There is a close relationship between Fourier transforms and polynomial evaluation and interpolation. Given a (n-1) degree polynomial, this polynomial can be uniquely represented in two ways, either by a list of its coefficients $a_0, a_1, \ldots, a_{n-1}$ or by a list of its values at n distinct points $x_0, x_1, \ldots, x_{n-1}$. The process of finding representation of a polynomial given its values is interpolation. Computing the Fourier transform of a vector is equivalent to converting the coefficient representation at its roots of unity. Likewise the inverse Fourier transform is equivalent to interpolating a polynomial given its value of the n-the roots of unity. Therefore the final step involves performing an inverse fast Fourier transform to recover the error vector. Figure 25 completes the decoding example presented in the previous chapters. Observe that Stage I involved the computation of fast

Fourier transforms and all the improvements discussed applies to the inverse as well. Details of the inverse Fourier transform are given in Appendix C.

Summary

In the cyclic procedure of the Chien search, it can be shown that Stage III may be accomplished in n clock periods and therefore realizes a great savings in decoding delay. For decoders with serial readout the error correction is accomplished during readout, hence it requires no additional time at all. However if the Chien search is simulated, the evaluation of each coefficients would require $n(t-1)$ multiplications and nt additions. In comparison, the inverse fast Fourier transform has been shown how it can be used in recovering the error vector. Being able to apply these faster forms of evaluating polynomials to coding theory allows the process of Stage III to take advantage of all the FFT speed. Therefore depending on the application, the inverse FFT will generally give better results.

CHAPTER VIII

SIMULATION AND CONCLUSIONS

The encoding and decoding described in the previous chapters was implemented in a software simulation. This program is used to correct any combination of t errors occuring in an RS codeword. The overall design of the program was to compare the conventional decoding with the new transform decoding. This chapter will give an outline of a general decoder, restate a summary of the algorithms, discuss the numerical complexity of each stage and explain the simulation design and implementation.

Outline of a Decoder for BCH Codes

Any decoding scheme that is to be used in a real-time application will eventually need to be implemented in a hardware design. The program simulation can not achieve cosequential processing but one needs to consider possible hardware implementations when designing the simulator. The following is a general discussion of the overall decoding of the Gorenstein-Zierler decoder which is the conventional method.

A sketch of an overall design for a RS code is shown in Figure 26. The decoder consists of four principal parts:

1) a buffer of 2n symbols, 2) shift registers wired to divide the incoming word by each irredicible factor of the generator polynomial, 3) a central Galois field processor to form $\sigma(z)$, and 4) a Chien searcher. An optional part is logic circuitry to calculate the error values. At a typical instant of time, the buffer will hold parts of three successive blocks as shown in Figure 27. The first i symbols of the incoming word; the next n symbols of the buffer hold the entire buffered word; the last n-i symbols of the buffer hold the last n-i symbols of the outgoing word. The Chien searcher is in the process of computing (w**i) in order to determine whether or not the next symbol to leave the buffer should be corrected. The shift registers are busy calculating the syndromes. The central processor is engaged in trying to find the error-locator polynomial for the buffered word.

When all the n symbols of the incoming block have been received, then all the symbols of the outgoing block have left. The buffer then appears as in Figure 27. The buffered block then becomes the outgoing block, and the incoming block becomes the buffered block. The coefficients of $\sigma(z)$ are read out of the central GF processor and into the Chien searcher, as the remainders of the received word or the syndromes are read into the central GF processor. As the next n digits of the incoming word arrive from the channel, the central GF processor must compute the coefficients of the error locator polynomial for the buffer word.

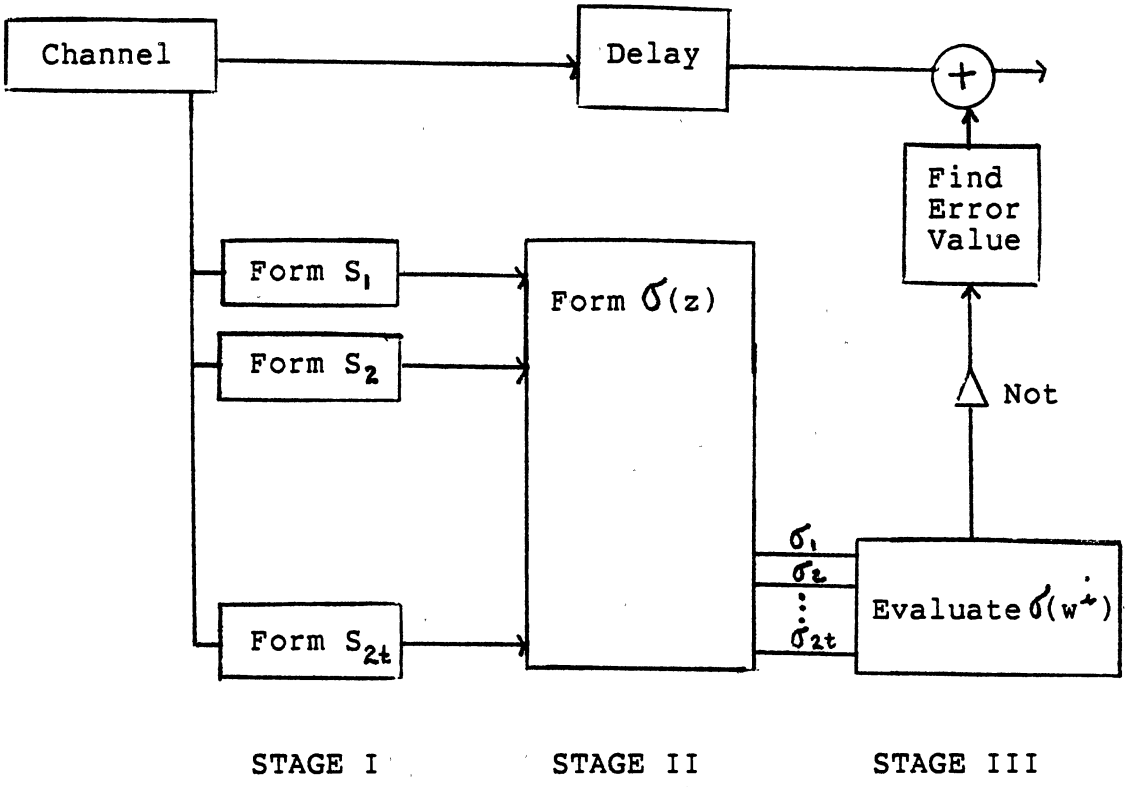


Figure 26. Overall Design of Hardware Decoder

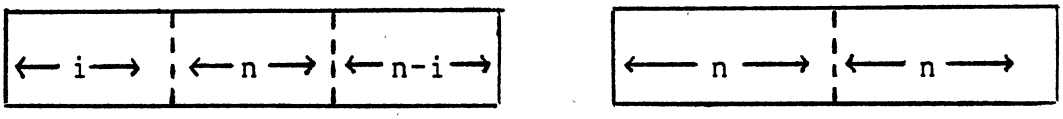


Figure 27. Typical Buffer During Decoding

The linear feedback shift registers that perform the division of the incoming polynomial by the irreducible factors of the generator polynomial and the Chien searcher must both operate in synchronization with the codewords in the buffer. The central GF computer, however, is totally disconnected from the rest of the decoder except between blocks when it outputs the error locator it has just computed and inputs the syndromes from the next block. It is not really essential that input and output of this central processor be executed simultaneously. If the central processor is so fast that it is able to compute the error locator before the new received word arrives then the buffer size may be reduced. In the extreme case of Hamming codes, the central processor may be eliminated altogether since S is the location of the only correctable error. The buffer may be reduced to n digits. In the other cases of q-ary symbols, the buffer may need to be extended to allow for finding the values of the errors after the Chien search.

Peterson Decoder

The Peterson algorithm has since been improved substantially and no longer is considered the standard decoding method. However for completeness of theory and hardware clock comparisons, it is briefly included. The Peterson binary procedure consists of three stages:

Stage I: Compute the power sums S_i from the received sequence through the relations $S_i = r(w^{**}i)$ for i=1,3,5,...,2t-1

Stage II: Compute the elementary symmetric

functions σ_k (k=1,2,...,t) from the power sums by using Newton's identities. The elementary symmetric functions are coefficients of the polynomial $\sigma(z)$.

Stage III: Find the roots of the polynomial $\sigma(z)$ by trial and error. These are the error locations.

(10,p.130)

Peterson (43) has given a rough estimate of the processing time required for each step of his procedure. It is assumed that addition or multiplication in GF(2**m) can be performed within one clock period, and division in a few clock periods. Following this computation, Stage I takes n clock periods where n is the length of the code block. Examples of circuits for accomplishing this are given in Figure 8. Stage II amounts to solving, at most, a set of t simultaneous linear equations. But one must try t-1 times, since the actual number of errors present may be anywhere from one to t. Roughly speaking, Stage II may take as many as (t**4)/2 clock periods. In accomplishing Stage III by the trial and error method, one generates each nonzero element in GF(2**m) in turn and substitutes it in $\sigma(z)$. One substitution may take 2t multiplications and t-1 additions, and this has to be done n times. Stage III, therefore, may take approximately 3tn clock periods. One might conclude that Stage III is the most time consuming of the three stages.

Gorenstein-Zierler Decoder

The Gorenstein-Zierler decoder has generally been accepted as the standard decoding method for q-ary RS codes.

In summary the multiple-error-correcting decoding procedure consists of three stages:

> Stage I: Find the weighted power sum symmetric functions $S_1, S_2, \ldots, S_{2t}$. For k=1,2,...,2t, S may be found from the formula $S_k = r_k(w^{**}k)$, where $r_k(x)$ is the remainder of the received polynomial R(x) divided by the minimal polynomial of w**k over GF(q), M(k)(x).
> Stage II: Knowing the generating function for S(z) mod z**(2t+1), use Berlekamp algorithm to solve the key equation for the polynomials $\sigma(z)$ and $\omega(z)$.
> Stage III: A) Using a Chien search over the n-th roots of unity over GF(q), find the reciprocal roots of the error locator polynomial (z). These are the error locations, and B) find the error values from Equation (7.6).
>
> (8,p.221)

Mandelbaum (30) has given the approximate processing time required for each step of the Gorenstein-Zierler algorithm. The comparisons are based on the number of operations in a software simulation. Stage I, the calculation of the syndromes require 2tn multiplications and the same number of additions assuming the polynomial is calculated by Horner's method. Stage II, the Berlekamp algorithm is used in both algorithms to be compared and is known to be Order(nlogn). The Chien search in Stage III can be simulated requiring n(t-1) multiplications and nt additions. In the non-binary case, the evaluation of the error values requires 2(t**2) multiplications and t(2t-1) additions.

Miller-Reed-Truong Decoder

In this thesis the methods developed by Miller, Reed, and Truong are applied to compute the syndromes in Stage I and establish the error vector in Stage III. In summary, the

procedure consists of three stages:

> Stage I: Compute the transform over GF(2**m) of the received n-tuple R(x) to obtain the syndromes S_k for 1<=k<=d-1.
> Stage II: Compute the error locator polynomial by Berlekamp algorithm.
> Stage III: A) Compute the remaining syndromes S for d<=k<=n from the known error locator polynomial and B) compute the inverse transform of S_k for 0<=k<=n-1 to recover the error vector. (51,p.136)

The approximate processing times for each step of this algorithm is easily calculated from the program design logic. Stage I, the calculation of the syndrome requires Order(nlogn) operations using the DFT. Stage II, the Berlekamp algorithm is used in both algorithms and is known to be Order(nlogn). The calculation of the rest of the syndromes in Stage III requires at most t(n-2t) multiplications and additions. The final step of Stage III, the inverse DFT is also Order(nlogn).

Program Design and Implementation

The decoding procedure described in the previous sections was implemented on the Vax 11/780 computer using PL/I language. The overall basic structure of the program is given in Figure 28. It is divided into a main program and six major subroutines. The main program is the driver of the rest of the program. It initializes the encoding and decoding processes and keeps track of the number of operations performed. The input subroutine obtains a code vector. The encoding subroutine encodes the code vector using the new interpolation method discussed in Chapter III.

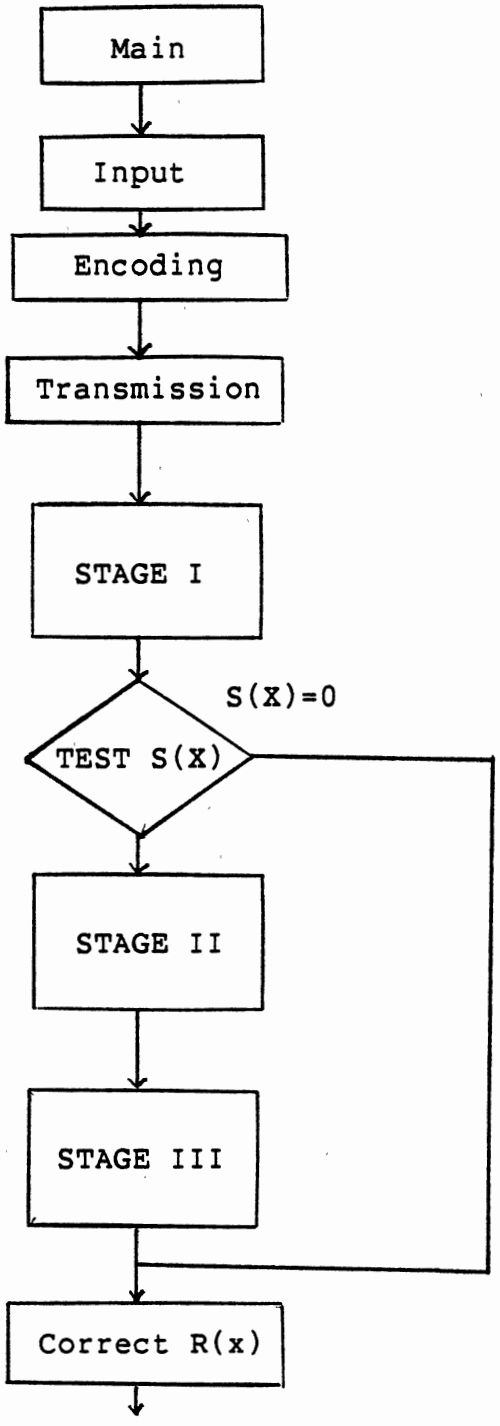


Figure 28. Overall Design of Simulator

The transmission subroutine adds the error vector to the codeword and simulates the buffer. Stage I, Stage II, Stage III subroutines simulate the stages of the Gorenstein-Zierler algorithm and the Miller-Reed-Truong algorithm. The stage subroutines of either methods can be included in the simulation in any combination.

Conclusions

In Stage II, both methods used the same algorithm so it was not included in the comparison. The basic differences between the two algorithms is Gorenstein-Zierler method computed the syndromes directly instead of using the FFT-like techniques of the Miller-Reed-Truong method. Also, the slower Chien search was used to find the roots of $\sigma(z)$ instead of another direct inverse transform of the syndrome vector. The overall conclusion is that the Miller-Reed-Truong algorithm reduces the numerical complexity. An important advantage of this new transform decoder is that the complexity of the syndrome calculation is substantially reduced. Furthermore, the Chien search is completely eliminated. The results is a simpler and faster decoder for finding the roots of the error locator polynomial than can be obtained by conventional means.

A specific example can be given how numerical complexity is reduced by this simulation's implementation. The Galois field operations of addition and multiplications are performed by PL/I statements. Addition is simple, and

basically an "exclusive OR" operation. However multiplication is more difficult. Multiplication is done by using two tables in memory: the logs and antilogs of a Galois field element. Two multiply two symbols, w_1 and w_2, one first finds from a table the exponents i and j such that $w_1 = B^{**}i$ and $w_2 = B^{**}j$. The $w_1 * w_2 = B^{**}(i+j)$ and the symbol corresponding to the exponent i+j must be found in the antilog table. It should be noted that in transform decoding the symbol w_i is multiplied with all the coefficients thus the log of w_i need only be found once. Not this savings does not apply when calculating the syndromes by Horner's method in which multiplication must be followed by an addition and then a multiplication, etc.

One disadvantage of the new q-ary codes is application. Often transmission is still in binary symbols and would need to be grouped to form q-ary symbols. Another disadvantage is solving the simultaneous equations based on the Newton's identities when t is small can not be generalized to q-ary symbols. In many applications the errors can be assumed to be independent either by nature or by the use of interleaved codes. Using this assumption, an error pattern containing a small number of erroneous symbols has a higher probability of occurrences than an error pattern containing a large number of erroneous symbols. In this case a good strategy in decoding BCH codes is to try the correction of a single error first. If the correction is unsuccessful, an attempt is made to correct a double error. It this correction is

still not successful, attempts are made to correct more errors. The advantage of this strategy is the increase in decoding speed which is crucial in many applications of error correcting codes. The aforementioned strategy was not implemented in this simulation because the simulation was a comparison of algorithms not a comparison of specific applications.

One final note on comparing the two decoding methods. Any decoding scheme has to be based on a specific application. Even though the Miller-Reed-Truong algorithm is numerically faster for a full-power correction scheme, there are so many variables involved in chosing a correction scheme it may not be the best for the application . These variables influence one's choice but hopefully this thesis and simulation has developed an overview of many of the choices.

CHAPTER IX

SUGGESTED FUTURE RESEARCH

The BCH and RS codes have become very important in coding theory. In one of Peterson's conclusions, he stated

> relatively simple coding and error-correcting methods have been described for Bose-Chaudhuri codes. The study of coding and error-correcting methods for these codes gives additional insight into the remarkable structure of the codes." (42,p.60)

Restated BCH or RS codes are codes with remarkable algebraic structures that allow decoding to occur in real-time applications. Methods of constructing efficient, very long codes have been devised; furthermore these codes have met the essential requirement that they can be implemented practically. "Recent work at RCA has concentrated on Reed-Solomon codes, which are felt to be more suitable to the optical disk." (6,p.37) RS are the most efficient among the known classes of codes. RS codes have the minimum redundancy for a given distance. RS theory has found significant applications in space communication systems, military communication systems, data communication systems, information retrieval systems, and in large secondary memories for computer systems (41).

While the problems of coding theory have originated from real engineering situations, coding theory in the early 1950's was distinctly academic. The fast development of practical applications came as a pleasant shock. Unlike the notion that the theory of finite fields might be applied to coding theory came as an unpleasant shock to certain pure mathematicians. This structure of the codes led to many improvements. Research in coding theory has developed over the past three decades and will continue to be important in the future. McWilliams and Sloane (38) suggest further research was needed to solve one major problem. "One version of the main problem of coding theory is to find codes with large R (for efficiency) and large d (to correct many errors). Of course these are conflicting goals." (38,p.23)

Coding theory has to develop with technology. For example, if a decoding procedure is implemented that has minimal decoding delay on current architectures, will this also be true on state-of-the-art designs? Also as transmission channels improve the emphasis may change where coding is used to overcome imperfections in the storage media rather than errors in transmission.

In comparison to current architecture, the Bartee and Schneider decoder which implemented Peterson's original decoding algorithm was at least on order of magnitude away both in hardware complexity and decoding delay. The need

existed for more efficient procedures and simpler circuits. The Chien search and the Berlekamp algorithm or the greatest common denominator algorithm improved the decoding procedure subtantially making the coding of RS codes practical. However the new transform algorithm has great potential in reducing the numberical complexity and therefore decoding delay. Similar to the finite theory application, the notion that the vast amount of research already done on fast Fourier transforms can be applied to coding theory is encouraging.

Research has shown that by the choices of three requirements: 1) the length N, 2) the modulus M, and the primitive root w, it is possible to develop improved transform algorithms. As transmission rates increase, the possible extensions of block length may be necessary. Research also has not been extensive on choices of modulus. A systematic investigation of those modulus which require more than two bit representation is difficult but may lead to interesting improvements. However achieved, improvements in transform methods would allow substantial improvements in the coding algorithm since the transform is performed in Stages I and III.

Stage II uses the efficient Berlekamp algorithm. Current research has made only minor improvements to this procedure. However it has not been proved and it would be interesting to see if the number of iterations the algorithm uses is minimal (8).

Another interesting problem is "find a complete decoding algorithm for all BCH codes." (38,p.277) The complete decoding of double-error- correcting BCH codes is given by Berlekamp (8). Complete decoding of some (perhaps all) triple-error-correcting BCH codes is given by Van de Horst and Berger (38). Their algorithm applies to all triple-error- correcting BCH codes if the following problem is settled: "show that the maximum weight of a coset leader of any coset of a triple-error- correcting BCH code is five." (38,p.293). The decoding algorithms discussed in this paper only correct t or fewer errors in a RS code or BCH code of designed distance 2t+1. If more than t errors are present a decoding error or decoding failure occurs. Further research may be able to eliminate both of these possibilities.

SELECTED BIBLIOGRAPHY

[1] Agarwal, R.C. and C.S. Burrus. "Fast Convolution Using Fermat Number Transforms With Application To Digital Fillering." IEEE Trans. on Acoustics, Speech, and Signal Processing. ASSP-22 (1974), 87-97.

[2] Alanen, J.D. and D.E. Knuth. "Tables of Finite Fields." Sankya. 26 (1964), 305-328.

[3] Albert, A.A. Fundamental Concepts of Higher Algebra. Chicago: University of Chicago Press, 1956.

[4] Aho, A.V., J.E. Hopcroft, and J.D. Ullman. The Design and Analysis of Computer Algorithms. Reading, MA: Addison-Wesley Publishing Company, 1978.

[5] Bartee, T.C., and D.I. Schneider. "An Electronic Decoder For Bose-Chaudhuri-Hocquenghem Codes." IEEE Trans. Info. Theory, 8 (1962), 17-24.

[6] Beiser, L., Ed. Advances in Laser Scanning Technology. San Diego, CA: SPIE, 1981.

[7] Berlekamp, E.R. "Factoring Polynomials Over Finite Fields." Bell Syst. Tech. J., 46 (1967) , 1853-1859.

[8] Berlekamp, E.R. Algebraic Coding Theory New York: MacGraw-Hill, 1968.

[9] Berlekamp, E.R. "Factoring Polynomials Over Large Finite Fields." Math. Comp, 24 (1970), 713-735.

[10] Berlekamp, E.R., Ed. Key Papers in Development of Coding Theory. New York: IEEE Press, 1974.

[11] Bose, R.C., and D.K. Ray-Chaudhuri. "On a Class of Error Correcting Binary Group Codes." Info. and Control, 3 (1960), 68-79.

[12] Bose, R.C., and D.K. Ray-Chaudhuri. "Further Results On Error-Correcting Binary Group Codes." Info. and Control, 3 (1960), 279-290.

[13] Burton, H.O. "Inversionless Decoding of Binary BCH Codes." IEEE Trans. Inform. Theory, 17 (1971), 464-466.

[14] Chien, R.T., B.D. Cunningham, and I.B. Oldham. "Hybrid Methods For Finding Roots of a PolynomialWith Application to BCH Decoding." IEEE Trans. Inform. Theory, 15 (1969), 329-335.

[15] Chien, R.T. "Cyclic Decoding Procedure For the Bose-Chaudhuri-Hocquenghem Codes." IEEE Trans. Info. Theory, 10 (1964), 357-363.

[16] Chien, R.T., and D.T. Tang. "On Definition of a Burst." IBM Journal, (July 1965), 292-293.

[17] Cooley, J.W., and J.W. Tukey. "An Algorithm For the Machine Computation of Complex Fourier Series." Math. Comput. 19 (1965), 297-301.

[18] Forney, G.D., Jr. "On Decoding BCH Codes." IEEE Trans. Info. Theory, 11 (1965), 549-557.

[19] Forney, G.D., Jr. Concatenated Codes. Cambridge, MA: MIT Press, 1966.

[20] Golay, M.J.E. "Notes on Digit Coding." Proc. IEEE, 37 (1949), 657.

[21] Goppa, V.D. "A New Class of Linear Error-Correcting Codes." Problems of Info. Trans., 6 (1970), 207-212.

[22] Gorenstein, D.C., and N. Zierler. "A Class of Error-Correcting Codes in P**M Symbols." J. Soc. Indus. Applied Math., 9 (1961), 207-214.

[23] Hamming, R.W. "Error Detecting and Error Correcting Codes." Bell Syst. Tech. J., 29 (1950), 147-150.

[24] Hocquenghem, A. "Codes Correcteurs D'Erreurs." Chiffres (Paris), 2 (1959), 147-156.

[25] Justesen, J. "On the Complexity of Decoding Reed-Solomon Codes." IEEE Trans. Info. Theory, 22 (1976), 237-238.

[26] Leibowitz, L.M. "A Simplified Binary Arithmetic For the Fermat Number Transform." IEEE Trans. on Acoustics, Speech, and Signal Processing, ASSP-24 (1976), 356-359.

[27] Lim, R.S. "A (31,15) Reed-Solomon Code For Large Memory Systems." AFIPS Conference Proceedings, 48 (1979), 205-208.

[28] Lin, S. An Introduction to Error-Correcting Codes. Englewood Cliffs, NJ: Prentice-Hall, 1970.

[29] van Lint, J.H. Coding Theory. New York: Springer, 1971.

[30] Mandelbaum, D. "On Decoding of Reed-Solomon Codes." IEEE Trans. Info. Theory, 17 (1971), 707-712.

[31] Mandelbaum, D. "Construction of Error Correcting Codes By Interpolation." IEEE Trans. Inform. Theory, IT-25 (1979), 27-35.

[32] Mandelbaum, D. "A Method for Decoding Generalized Goppa Codes." IEEE Trans. Inform. Theory, IT-23 (1977), 137-140.

[33] Marsh, R.W. Table of Irreducible Polynomials Over GG(2) Through Degree 19. Washington, DC: Dept. of Commerce, 1957.

[34] Massey, J.L. "Shift-register Synthesis and BCH Decoding." IEEE Trans. Info Theory, 15 (1969), 122-127.

[35] MacClellan, J.H. "Hardware Realization of a Fermat Number Transform." IEEE Trans. On Accoustics, Speech, and Signal Processing, ASSP-24 (1976), 216-225.

[36] MacCoy,N.H. The Theory of Numbers. New York: The MacMillian Company, 1965.

[37] MacEliece, R.J. The Theory of Information and Coding. Reading, MA: Addison-Wesley Publishing Company, 1977.

[38] MacWilliams, F.J., and N.J.A. Sloane. The Theory of Error-Correcting Codes Amsterdam: North-Holland Publishing Company, 1977.

[39] Mills, W.H. "Continued Fractions and Linear Recurrences." Math. Comp, 29 (1975), 173-180.

[40] Muller, D.E. "Application of Boolean Algebra to Switching Circuit and to Error Detection." IEEE Trans. Computer, 3 (1954), 6-12.

[41] Oldham, I.B., R.T. Chien, and D.T. Tang. "Error Detection and Correction in a Photo-Digital Storage System." IBM J. Res. Develop., 12 (1968), 422-430.

[42] Peterson, W.W. "Encoding and Error-Correction Procedure for the Bose-Chaudhuri Codes." IEEE Trans. Info. Theory, 6 (1960), 459-470.

[43] Peterson, W.W. Error-Correcting Codes. Cambridge, MA: MIT Press, 1961.

[44] Peterson, W.W. and E.J. Weldon, Jr. Error-Correcting Codes, 2nd ed. Cambridge, MA: MIT Press, 1972.

[45] Pollard, J.M. "The Fast Fourier Transform in a Finite Field." Math. Comp., 25 (1971), 365-374.

[46] Reed, I.S. "A Class of Multiple-Error-Correcting Codes and The Decoding Scheme." IEEE Trans. Info. Theory, 4 (1954), 38-49.

[47] Reed, I.S., and G. Solomon. "Polynomial Codes Over Certain Finite Fields." J. SIAM, 8 (1960), 300-304.

[48] Reed, I.S., and T.K. Truong. "Simple Proof of the Continued Fraction Algorithm for Decoding Reed-Solomon Codes." Proc. IEEE, 125 (1978), 1318-1320.

[49] Reed, I.S., R.A. Scholtz, T.K. Troung, and L.R. Welch. "The Fast Decoding of Reed-Solomon Codes Using Fermat Theoretic Transforms and Continued Fractions." IEEE Trans. Inform. Theory, IT-24 (1978), 100-106.

[50] Reed, I.S., T.K. Truong, and R.L. Miller. "Fast Algorithm for Encoding the (255,223) Reed-Solomon Code Over GF(2**8)." Electronic Letters, 16 (1980), 222-223.

[51] Reed, I.S., T.K. Truong, and R.L. Miller. "Efficient Program for Decoding the (255,223) Reed-Solomon Code Over GF(2**8) With Both Errors and Erasures Using Transform Decoding." IEE Proc., 127, Pt. E, No. 4 (1980), 136-142.

[52] Slepian, D. "A Note On Two Binary Signaling Alphabets." IEEE Trans. Info. Theory, 2 (1956), 84-86.

[53] Slepian, D.,Ed. Key Papers in the Development of Information Theory, New York: IEEE Press, 1974.

[54] Stone, J.J. "Multiple-Burst Error Correction with the Chinese Remainder Theorem." J. Soc. Indust. Applied Math, 11, No. 1 (1963), 74-81.

[55] Sugiyama, Y., M. Kasahara, S. Hirawaa, and T. Namekawa. "A Method for Solving Key Equation for Decoding Goppa Codes." Info. and Control, 27 (1975), 87-99.

[56] Tang, D.T., and R.T. Chien. "Coding for Error Control." IBM Sys. J., 8 (1969), 48-49.

[57] Winograd, S. "On Computing The Discrete Fourier Transform." Proc. Nat. Acad. Sci., 73, No. 4 (1976), 1005-1006.

APPENDIX A

BASIC FINITE FIELD THEORY

Basic to understanding algebraic coding theory is a general background in finite field theory. This appendix is designed as a basic tutorial to define the terms: 1) Galois feild, 2) irreducible polynomials, 3) minimal polynomials, and 4) primitive nth root of unity. For the reader who is interested in more detail coverage, references are cited.

The integers modulo p form a field of order p-1, denoted by GF(p), where p is a prime number. The elements of GF(p) are

$$\{ 1 , w^{1} , w^{2} , w^{3} , \ldots , w^{p-2} \} \text{ with } w^{p-1} = 1 \tag{A.1}$$

where w is the generator of the field. The set of p-1 non-zero elements is a cyclic multiplicative group with addition, subtraction, multiplication, and division carried out modulus p. Also there is essentially only one field of order (p**m)-1 called a Galois Field, denoted by GF(p**m). Any member of GF(p**m) can also be written as a m-tuple of elements from GF(p) (3). Extensive tables of binary irreducible polynomials can be found in Marsh (33) and Peterson (43). Or extensive tables of non-binary

irreducible polynomials can be located in Alanen and Knuth (2).

The knowledge of constructing a finite field is important in coding theory. The construction of GF(p**m) is possible if there exists an irreducible polynomial over GF(p) that has degree m. A polynomial f(x) is irreducible over a field if it is not the product of two polynomials of lower degree in the field. The set of all polynomials of degree <= m-1 and coefficients from GF(p) with calculations performed modulo f(x) form a field. An example of construction of a field GF(2**4) is given in Figure 29 (28).

One method of finding the irreducible polynomials is simply the enumeration shown in Figure 30. In general there are 2**degree polynomial combinations. If any futher enumeration is of interest, the number of irreducible polynomials of any degree can be obtained by an explicit formula which is an immediate consequence of the multiplicative Moebius inversion theorem (8, 38). Also relative and of interest is the reciprocal of an irreducible polynomial is also irreducible (3).

If w is a root in GF(p**m) of an irreducible polynomial of degree m over GF(p) then every element in GF(p**m) is a root of some minimal polynomial over GF(p). A minimal polynomial over GF(p) of every element Beta is the lowest degree polynomial M(x) with coefficients from GF(p) such that M(Beta) = 0. Several examples are given in Figure 31.

$GF(2^4)$ Defined by $x^4 + x + 1$ Represented as

Power of w	Negative power	Polynomial	Binary	Log w
w^0	w^{15}	1	0 0 0 1	0
w^1	w^{14}	w	0 0 1 0	1
w^2	w^{13}	w^2	0 1 0 0	2
w^3	w^{12}	w^3	1 0 0 0	3
w^4	w^{11}	$w + 1$	0 0 1 1	4
w^5	w^{10}	$w^2 + w$	0 1 1 0	5
w^6	w^9	$w^3 + w^2$	1 1 0 0	6
w^7	w^8	$w^3 + w + 1$	1 0 1 1	7
w^8	w^7	$w^2 + 1$	0 1 0 1	8
w^9	w^6	$w^3 + w$	1 0 1 0	9
w^{10}	w^5	$w^2 + w + 1$	0 1 1 1	10
w^{11}	w^4	$w^3 + w^2 + w$	1 1 1 0	11
w^{12}	w^3	$w^3 + w^2 + w + 1$	1 1 1 1	12
w^{13}	w^2	$w^3 + w^2 + 1$	1 1 0 1	13
w^{14}	w^1	$w^3 + 1$	1 0 0 1	14

$w^n = (w^{n-1})\, w$ Modulo Primitive Polynomial
where w is Primitive Root of Unity

Figure 29. Construction of Field GF(p**m)

BY ENUMERATION

Degree = 1	x	irreducible
	$x + 1$	irreducible
Degree = 2	x^2	x*x
	x^2	(x + 1) (x + 1)
	$x^2 + x$	x (x + 1)
	$x^2 + x + 1$	irreducible
Degree = 3	x^3	x*x*x
	x^3	$(x+1)(x^2+x+1)$
	$x^3 + x$	$x (x^2 + 1)$
	$x^3 + x + 1$	irreducible
	$x^3 + x^2$	$x^2 (x + 1)$
	$x^3 + x^2 + 1$	irreducible
	$x^3 + x^2 + x$	$x(x^2 + x + 1)$
	$x^3 + x^2 + x + 1$	$(x + 1)^3$

Degree = 4 Only three are irreducible out of 2**4

$$x^4 + x + 1 \; ; \; x^4 + x^3 + 1 \; ; \; x^4 + x^3 + x^2 + x + 1$$

Degree = 5 Only six are irreducible out of 2**5

$$x^5 + x^2 + 1; \; x^5 + x^4 + x^3 + x^2 + 1; \; x^5 + x^4 + x^3 + x + 1$$

and their inverses

Degree = 6 Only nine are irreducible out to 2**6

$$x^6 + x^4 + x^2 + x + 1; \; x^6 + x^5 + x^2 + x + 1; \; x^6 + x + 1$$

$$x^6 + x^5 + x^3 + x^2 + 1; \; x^6 + x^3 + 1; \text{ and inverses}$$

Figure 30. Irreducible Polynomials

ELEMENTS OF GF MINIMAL POLYNOMIAL OF ELEMENTS

P**M = 2**1 where $x - x = x\,(x+1)$

0	x	
1	$x + 1$	$= M(0)$

P**M = 2**2 where $x^4 - x = x\,(x+1)(x^2+x+1)$

0	x	
1	$x + 1$	$= M(0)$
w^1, w^2	$x^2 + x + 1$	$= M(1)=M(2)$

P**M = 2**4 since M is divisible by 1,2,4 then

$$x^{16} - x = x(x+1)(x^2+x+1)(x^4+x+1)(x^4+x^3+1)(x^4+x^3+x^2+x+1)$$

GF(2**4) DEFINED BY X**4 + X + 1

0	x	
1	$x + 1$	$= M(0)$
w^1, w^2, w^4, w^8	$x^4 + x + 1$	$= M(1)=M(2)=M(4)=M(8)$
w^3, w^6, w^{12}, w^9	$x^4 + x^3 + x^2 + x + 1$	$= M(3)=M(6)=M(12)=M(9)$
w^5, w^{10}	$x^2 + x + 1$	$= M(5)=M(10)$
$w^7, w^{14}, w^{13}, w^{11}$	$x^4 + x^3 + 1$	$= M(7)=M(14)=M(13)=M(11)$

EXAMPLE

$x^4 + x^3 + x^2 + x + 1 = 0$ for w modulo $x^4 + x + 1$

$$(w^3)^4 + (w^3)^3 + (w^3)^2 + w^3 + 1 = 0$$

$$w^{12} + w^9 + w^6 + w^3 + w^0 = 0$$

$$0 = 0$$

1	1	1	1
1	0	1	0
1	1	0	0
1	0	0	0
0	0	0	1
0	0	0	0

Figure 31. Minimal Polynomials

In coding theory minimal polynomials are sometime referred to as primitive polynomials. A primitive polynomial is the minimal polynomial of a primitive element of GF(p**m) and has degree m (38).

Basic to coding theory is Fermat's theorem that

Every element Beta of a field F of order p**m satisfies the identity

$$\text{Beta}^{p^m} = \text{Beta}$$

or equivalently is a root of the equation

$$x^{p^m} = x \quad \text{thus}$$

$$x^{p^m} - x = \prod_{\beta \in F} (x - \text{Beta})$$

or $x^{p^m} - x$ = Product of all irreducible polynomials over GF(p) whose degree divides m (38,p.96).

The factorization of this polynomial can be separated into the zero element and the non-zero element. Therefore there are n distinct zeros which are called the nth roots of unity defined in Figure 32.

In constructing GF(p**m) from a primitive irreducible polynomial f(x), the basis

$$\{ 1 , w^1 , w^2 , \ldots , w^{n-1} \}$$

where w is a zero of f(x). However there are other possibilites (8). One such possibility is a trace

$$\text{Trace} (\beta) = \beta + \beta^{\rho} + \beta^{\rho^2} + \ldots + \beta^{\rho^{m-1}} = \sum_{j=0}^{m-1} \beta^{\rho^j}$$

Trace (β) = element of GF(p) where Beta or $\beta \in$ GF (p^m)

GIVEN

$$x^q - x = x\,(x^{q-1} - 1) \qquad \text{where } q = 2 ** M$$

$$x\,(x^n - 1) \qquad \text{where } n = q - 1$$

THEN

$$x^n - 1 = \prod_{i=0}^{n-1} (x - w^i) = \prod_{i=1}^{n} (x - w^i)$$

where w is primitive root of unity if

a) $w \neq 1$

b) $w^n = 1$ modulo q

c) $w^{n/2} = -1$ modulo q if n is even

d) $\sum_{i=0}^{n-1} a^{ip} = 0$ for $1<=p<=n$

EXAMPLE q= 17 and n = 16

n =	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2^n =	1	2	4	8	16	15	13	9	1	2	4	8	16	15	13	9	1
3^n =	1	3	9	10	13	5	15	11	16	14	8	7	4	12	2	6	1
4^n =	1	4	16	13	1	4	16	13	1	4	16	13	1	4	16	13	1
6^n =	1	6	2	12	4	7	8	14	16	11	15	5	13	10	9	3	1

Only 3 and 6 generate the entire field GF(17)

$6 = \sqrt{2}$ in the sense that 6**2 = 2 modulo 17

Figure 32. Primitive root of unity

APPENDIX B

CHINESE REMAINDER THEOREM

The Chinese remainder theorem for polynomial is used to guarantee that a polynomial can be recovered from its residues.

Chinese Remainder Theorem For Polynomial. Let $m_0(x),\ldots,m_{k-1}(x)$ be polynomials over GF(q) which are pairwise relatively prime, and set $M(x)=m_0(x)m_1(x)\ldots m_{k-1}(x)$. If $r_0(x),\ldots,r_{k-1}(x)$ are any polynomials over GF(q), there exists exactly one polynomial u(x) with deg u(x) < deg M(x) such that

(B.1)

$$u(x)=r_i(x) \pmod{m_i(x)},$$

for all i=0,...,k-1. In fact, let $a_i(x)$ be such that

(B.2)

$$\frac{M(x)}{m_i(x)}\, a_i(x) = 1 \pmod{m_i(x)}, \quad i= 0,\ldots, k-1.$$

(Such an $a_i(x)$ exists by Euclidean algorithm.) The the solution to Equation B.1 is

(B.3)

$$u(x) = \sum_{i=0}^{k-1} \frac{M(x)}{m_i(x)}\, r_i(x)\, a_i(x) \text{ reduced mod } M(x).$$

(38,p305)

APPENDIX C

DISCRETE FOURIER TRANSFORMS

In many applications it is convenient to transform a problem into another, easier problem. In this appendix, one will be introduced to the Fourier transform, its inverse, and its numerical complexity. An efficient algorithm called the fast Fourier transform (FFT) is developed. The algorithm which is based on techniques of polynomial evaluation by division, makes use of the fact that a polynomial is being evaluated at the roots of unity. Appendix A gives the necessary background material for the n-th root of unity.

The Fourier transform is usually defined over the complex numbers. For example,

(C.1)

$$e^{2\pi i/n} \quad \text{where} \quad i = \sqrt{-1}$$

is a principal nth root of unity in the ring of complex numbers. However for application in coding theory, one can define the Fourier transform over a arbitrary field. Computing the discrete Fourier transform of the codeword vector means evaluating the polynomial representation at each of the nth roots of unity.

(C.2)

$$(w^0, w^1, w^2, \ldots, w^{n-1}) \text{ where } n=2^k \text{ for } k>=0$$

The algorithm is developed by grouping the terms of P(X) with even powers and the terms of P(X) with odd powers as illustrated in Figure 33. As the algorithm design is presented the breakdown of the polynomial seems systematic enough that one should be able to carry out the scheme with a divide and conquer algorithm.

To help suggest the pattern of the computation, an example is presented in a tree diagram in Figure 34. Computation can be simplified by starting at the leaves. The leaves are components of the vector P permuted in the following way. Let t be an integer between 0 and n-1. Then t can be represented in binary as and the reverse of t be the number represented by the bits in reverse order.

(C.3)

$$t = [\, b_0\, b_1 \ldots b_{k-1} \,] \quad \text{where } n = 2^{**}k$$

$$rev(t) = [\, b_{k-1} \ldots b_1\, b_0 \,]$$

Therefore the algorithm in Figure 35 computes the values of P(X) at the nth roots of unity or it computes the discrete Fourier transform of the vector P.

The inverse discrete Fourier transform is

(C.4)

$$\frac{1}{n} \sum_{k=0}^{n-1} a_k\, w^{-ik} \qquad 0<=i<n$$

In the inverse transform, substitute w**-1 for w and

$$P(X) = p_0 + p_1 x + \ldots + p_{n-1} x^{n-1} = \sum_{i=0}^{n-1} p_i x^i$$

where n = 2**k for k >= 0

$$P(x) = \sum_{i=0}^{(n/2)-1} p_{2i} x^{2i} + x \sum_{i=0}^{(n/2)-1} p_{2i+1} x^{2i}$$

define $P_{even}(X) = \sum_{i=0}^{(n/2)-1} p_{2i} x^i$ and $P_{odd}(X) = \sum_{i=0}^{(n/2)-1} p_{2i+1} x^i$

then

$$P(X) = P_{even}(x^2) + x P_{odd}(x^2) \text{ and } P(-X) = P_{even}(x^2) - x P_{odd}(x^2)$$

Recall $w^{n/2} = -1$ so for $0 <= j <= (n/2)-1$

$w^{(n/2)+j} = -w^j$

or $\{ 1, w^1, w^2, \ldots, w^{n-1} \}$

Roots of unity is equivalent to

$$\{ 1, w^1, \ldots, w^{(n/2)-1}, -1, -w^1, \ldots, -w^{(n/2)-1} \}$$

or it suffices to evaluate P_{even} and P_{odd} at

$$\{ 1, w^2, \ldots, (w^{(n/2)-1})^2 \}$$

Figure 33. Development of Fast Fourier Algorithm

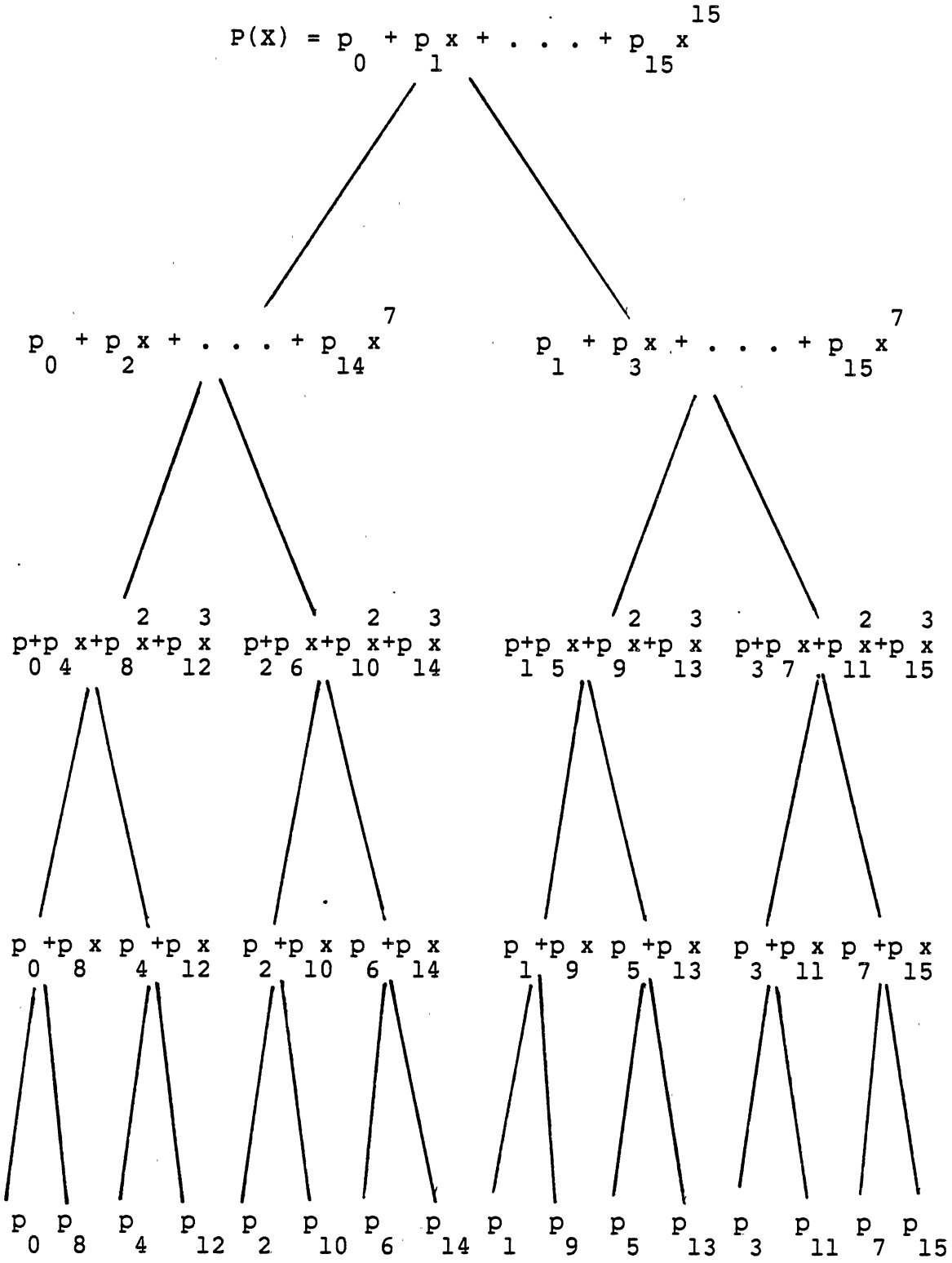


Figure 34. Polynomial Evaluation At Roots of Unity

```
Input: The n-vector P=(p ,p ,...,p    ) where n=2**k for k>0

Output:  VAL, the Discrete Fourier Transform of P

Comment:  Omega is an array containing the nth primitive
          root of unity
          VAL is initialized to contain the values for
          level k-1 in the tree of Figure 2C.

Comment:  Initialization of VAL

1.  for t <-- 0 to n-2 by 2 do

2.        VAL(t)   <-- p       (t) + p       (t+1)
                       rev(k)        rev(k)

3.        VAL(t+1) <-- p       (t) - p       (t+1)
                       rev(k)        rev(k)

4.  end


Comment: NVAL is number of points at which each polynomial
         at the current level is evaluated.
         The levels are indexed by 1.

5.  m <-- n/2;  NVAL <-- 2

6.  for 1 <-- k-2 to 0 by -1 do

7.        m <-- m/2; NVAL <-- 2 * NVAL

8.        for t <-- 0 to [(2**1)-1]NVAL by NVAL;

9.        for j <-- 0 to (NVAL/2)-1 do

10.            XPODD <-- OMAEGA(mj)* VAL[t + (NVAL/2) + j]

11.            VAL[t + (NVAL/2) + j] <-- VAL(t + j) - XPODD

12.            VAL (t + j) <-- VAL(t + J) + XPODD

13.       end

14.  end
```

Figure 35. Fast Fourier Transform Algorithm

multiply each result by the inverse of n. These changes are given in Figure 36.

An analysis of the operations done by the FFT gives the following results. Lines 10,11, and 12, which do one multiplication, one addition, and one subtraction, respectively, all in a triply nested loop. NVAL=2**(k-1) so the ranges of the loops indexes indicate that the number of each operation done in these lines is given in Figure 37. Line 2 and Line 3 do n/2 additions and n/2 subtractions. Therefore there are n*(logn/2)+n additions/subtractions and n/(2*(logn/2) multiplications. Even with the reversing of the bits of k, the running time of the FFT is Order(nlogn) (4).

Input: A,n, where A is an n-vector,;n is a power of 2.

Output: The vector $B=(b_0, b_1, \ldots, b_{n-1})$, the inverse FFT of A.

1. Compute FFT of A using Fast Fourier Transform Algorithm in Figure 3C and leaving the results in VAL.

2. b_0 <-- VAL(0)/n

   for i <--1 to n-1; b_i <-- VAL(n-i)/n

Figure 36. Inverse Discrete Fourier Transform

$$\sum_{l=0}^{k-2} 2^{l} \frac{NVAL}{2} = \sum_{l=0}^{k-2} 2^{l} 2^{k-l-1}$$

$$= \sum_{l=0}^{k-2} 2^{k-l}$$

$$= (k-1)\, 2^{k-1}, \quad (n/2) \log (n/2)$$

Figure 37. Number of Operations in Loops of FFT

APPENDIX D

NEWTON'S IDENTITIES

A polynomial $P(x_0,\ldots,x_{n-1})$ in n indeterminates x is called "symmetric" if it is invariant under the symmetric group of all permutations of its subcripts. For the n=3 example, particular symmetric polynomials which are the coefficients in the expansion are

(D.1)

$$\sigma_1 = x_1 + x_2 + x_3 ; \quad \sigma_2 = x_1 x_2 + x_1 x_3 + x_2 x_3 ; \quad \sigma_3 = x_1 x_2 x_3$$

$$(X - X_1)(X - X_2)(X - X_3) = X^3 - \sigma_1 X^2 + \sigma_2 X - \sigma_3$$

In general such polynomials are called elementary symmetric polynomials (in n variables)

(D.2)

$$\sigma_1 = \sum_i x_i , \sigma_2 = \sum_{i<j} x_i x_j , \sigma_3 = \sum_{i<j<k} x_i x_j x_k , \ldots, \sigma_n = x_1 \ldots x_n$$

Since $(-1)^{**}k\,\sigma_k$ is the coefficient of $t^{**}(n-k)$ in the expansion of

(D.3)

$$P(x) = \prod_k (X - X_k)$$

As a polynomial in X, the expression σ_i give the coefficients of P(x) as functions of its roots. In conclusion, any

symmetric polynomial can be expressed as a polynomial in the elementary symmetric polynomials. For example,

(D.4)

$$x^2 + y^2 = (x+y)^2 - 2xy = \sigma_1^2 - 2\sigma_2$$

The elementary symmetric functions σ_i are related to the power sum symmetric functions by Newton's identities which are

(D.5)

$$S_1 - \sigma_1 = 0$$

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

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

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

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

$$\vdots$$

etc.

APPENDIX E

EUCLIDEAN ALGORITHM

This information is included in an appendix because it does not deal directly with the problem of decoding BCH or RS codes. The reader should bear in mind, however, that our goal is to solve the key equation in Figure 19 for $\sigma(z)$ and $\omega(z)$, given S(x). Throughout this section a(x) and b(x) will be fixed polynomials over field F, with deg(a)>=deg(b)>=0. When applied to coding theory a(x) will be replaced by x**2t, and b(x) by the syndrome polynomial S(x).

The Euclidean algorithm is a simple and straightforward algorithm for finding the greatest common divisor (gcd) of two intergers or polynomials, or for finding the continued fraction expansion of a real number. This algorithm is discussed only as it applies to polynomials. if a(x) and b(x) are polynomials, by a gcd of a(x) and b(x), one means a polynomial of highest degree which divides both a(x) and b(x).

By the division algorithm, one may divide a(x) by b(x):

(E.1)

$$a(x) = b(x) * q_1(x) + r_1(x)$$

It follows that the gcd of a(x) and b(x) is the same as the

gcd of $b(x)$ and $r_1(x)$. This procedure can now be repeated on $b(x)$ and $r_1(x)$; divide $b(x)$ by $r_1(x)$:

(E.2)

$$b(x) = r_1(x) * q_2(x) + r_2(x)$$

Next

$$r_1(x) = r_2(x) * q_3(x) + r_3(x)$$

$$\vdots$$

Finally

$$r_{n-1}(x) = r_n(x) * q_{n+1}(x) + 0$$

In other words, one continues to divide each remainder by the suceeding remainder. Since the remainder continually decrease in degree, there must ultimately by a zero remainder.

But one sees that since $r_n(x)$ is a divisor of $r_{n-1}(x)$, it must be the gcd of $r_n(x)$ and $r_{n-1}(x)$. Thus

(E.3)

$$\gcd[a(x),b(x)] = \gcd[b(x),r_1(x)] = \ldots = \gcd[r_{n-1}(x),r_n(x)] = r_n(x)$$

As a by-product of the Euclidean algorithm, the

(E.4)

$$s_n(x) * a(x) + t_n(x) * b(x) = r_n(x)$$

is also produced which expresses $r_n(x)$ as a linear combination of $a(x)$ and $b(x)$. The algorithm involves four sequences of polynomials which initial conditions are

(E.5)

$$s_{-1}(x) = 1 \quad t_{-1}(x) = 0 \quad r_{-1}(x) = a(x) \quad q_{-1}(x) = \text{not defined}$$

$$s_{0}(x) = 0 \quad t_{0}(x) = 1 \quad r_{0}(x) = b(x) \quad q_{0}(x) = \text{not defined}$$

For $i>=1$, $q_i(x)$ and $r_i(x)$ are defined to be the quotient and remainder, respectively, when $r_{i-2}(x)$ is divided by $r_{i-1}(x)$ as shown in Equation E.2. The polynomials are then defined by

(E.6)

$$r_i(x) = q_{i-2}\, r_{i-1} + r_i$$

$$s_i(x) = s_{i-2}(x) - q_i(x) * s_{i-1}(x)$$

$$t_i(x) = t_{i-2}(x) - q_i(x) * t_{i-1}(x)$$

Since the degrees of the remainders r are strictly decreasing, there will be a last non-zero one: call it $r_n(x)$. It turns out that $r_n(x)$ is the gcd of a(x) and b(x), and furthermore the desired equation (E.1) is acheived.

When the algorithm terminates with $r_n = 0$, the desired multipliers s_{i-1} and t_{i-1} as well as the gcd (r_{n-1}) have all been computed. This method is called the continued-fractions version of Euclidean algorithm. The reason for this nomenclature is shown in Figure 39. It can be shown that the quotients s_k/t_k represent the successive convergents of this continued fraction. The reader interested in learning more about continued fractions will find an excellent introduction in McCoy (36), Mills (39), and Reed (48). Figure 38 is an example of finding the greatest common divisor.

TABLE IV

GREATEST COMMON DIVISOR OF POLYNOMIALS

i	s_i	t_i	r_i	q_i
-1	1	0	x^8	...
0	0	1	$x^6 + x^4 + x^2 + x + 1$	...
1	1	$x^2 + 1$	$x^3 + x + 1$	$x^2 + 1$
2	$x^3 + 1$	$x^5 + x^3 + x^2$	x^2	$x + 1$
3	$x^4 + x + 1$	$x^6 + x^4 + x^3 + x^2 + 1$	$x + 1$	x
4	$x^5 + x^4 + x^3 + x^2$	$x^7 + x^6 + x^3 + x + 1$	1	$x + 1$
5	$x^6 + x^4 + x^2 + x + 1$	x^8	0	$x + 1$

$$\frac{r_{-2}}{r_{-1}} = q_0 + \cfrac{1}{q_1 + \cfrac{1}{q_2 + \cfrac{1}{\ddots + \cfrac{1}{q_n}}}}$$

Figure 38. Continued-Fractions of Euclidean Algorithm

The main result of this algorithm applied to coding theory is that

(E.7)

$$s(x)a(x) + t(x)b(x) = r(x)$$

$$t(x)b(x) = r(x) \bmod a(x)$$

where

$$\deg(t) + \deg(r) < \deg(a)$$

Then there exists a unique index i and a polynomial $\lambda(x)$ such that

(E.8)

$$t(x) = \lambda(x)\, t_i(x) = t_i(x)$$
$$s(x) = \lambda(x)\, s_i(x) = s_i(x)$$
$$r(x) = \lambda(x)\, r_i(x) = r_i(x)$$

However, if t(x) and r(x) are relatively prime, the polynomial $\lambda(x)$ must be a constant. Finally if t(x) is defined as a monic polynomial, then the constant must be one.

VITA 2

Arthurine Renee Davis Breckenridge

Candidate for the Degree of

Master of Science

Thesis : SIMULATION OF ERROR CORRECTION ALGORITHMS USING REED SOLOMON CODES

Major Field: Computer Science

Biographical:

Personal Data: Born in Oklahoma City, Oklahoma, August 18, 1953, the daughter of Wendell B. and Ludie A. Davis.

Education: Graduated from Del City High School, Del City, Oklahoma, in May 1971; received Bachelor of Science degree in Social Studies Education from Oklahoma State University in December, 1981; completed requirements for the Master of Science degree at Oklahoma State University in May, 1984.

Professional Experience: High School Mathematics Teacher at Oilton High School, Oilton, Oklahoma, August 1978 to May, 1981. Programmer at AMOCO Production Company, Tulsa, Oklahoma, June, 1983 to August, 1983. Graduate Teaching Assistant, Department of Mathematics, Oklahoma State University, Stillwater, Oklahoma, August, 1981 to December, 1983.