A REED-SOLOMON CODE SIMULATOR AND PERIODICITY ALGORITHM

ZHENPEN YOUNG

A Reed-Solomon Code Simulator and Periodicity Algorithm

By

Zhenpen Young

A thesis submitted to the School of Graduate Studies

in partial fulfillment of the requirements for the degree of

Master of Engineering

Faculty of Engineering and Applied Science

Memorial University of Newfoundland

St. John's Newfoundland Canada

ISBN 0-315-91581-1

To my dear parents

Acknowledgment

I would like to take this opportunity to express my sincere thanks to my thesis supervisor, Dr. Son Le-Ngoc. Without his assistance, I could not even have had a chance to come to Canada. Furthermore, his stimulating my interest in this exciting field and his constant encouragement and guidance during the course of my study led to the completion of this thesis.

The financial support from NSERC and the Faculty of Engineering and Applied Science is gratefully acknowledged.

Last but not the least, I would like to give my special thanks to my wife, Wenmao, for her love and quiet help throughout the composition of this thesis.

Abstract

Communication channels are usually affected by various kinds of noise. As a result, errors occur in data transmissions. Reed-Solomon (RS) codes, as other channel codings, are widely used to eliminate the errors due to its optimal characteristics in both Hamming distance and structure but most of all its capability of correcting both random and burst errors.

The selection of the best RS code for a specific communication channel is always a major issue in system design. Hence, this thesis introduces and implements an RS code simulator to study a wide range of RS codes. The simulator first encodes the user's message into a codeword. The user can choose the symbol length m from 3 bits up to 8 bits or the block length N from 7 symbols up to 255 symbols, and the error correcting capability T of up to 16 random errored symbols. Then the user enters an error pattern of arbitrary weight which the simulator adds to the generated codeword. The resulting received word is then decoded. Either the direct (Peterson's) or iterative (Berlekamp's) method is used to construct the error locator polynomial. Only the Chien search is used as a root search technique for the error locator polynomial. This simulator does not handle erasures.

Commonly, Chien search is used to find out all the possible roots of the error locator polynomial. It is found that for the double error correcting case

($T = 2$) these roots are not randomly distributed but they follow certain patterns. Based on these patterns, the periodicity algorithm is introduced and its validity is verified by exhaustive computer simulations.

With fewer than 8 additions, 4 decision operations and only N symbols of memory space required, the periodicity algorithm outperforms Chien search and the binary decision fast Chien search techniques in terms of decoding time. Most of all it also outperforms the Okano's analytical solutions by a decision operation. Of course, the look-up table is the fastest in terms of the decoding time but its memory space required is N^2. This will limit its use when N is large. Therefore, it is concluded that the periodicity algorithm is the optimal solution for both decoding time and memory space. This algorithm is found to be very suitable for use in microprocessor based decoders.

Contents

List of Figures

List of Tables

List of Symbols

α	primitive element in $GF(2^m)$
$\Sigma(x)$	error locator polynomial obtained by Peterson's method
$\sigma(x)$	error locator polynomial obtained by Berlekamp's method
Δ	determinant of the matrix
μ	index in Berlekamp's iterations
ν	actual error number
$B(x)$	remainder polynomial
$C(x)$	codeword polynomial
d_μ	discrepancy in Berlekamp's iterations
D	minimum distance
$D(x)$	decoded word polynomial
E_1	position of the first error
E_2	position of the second error
$E(x)$	error pattern polynomial
$\hat{E}(x)$	estimated codeword polynomial
$f(x)$	polynomial over Galois fields
$g(x)$	generator polynomial
GF	Galois field
$i(x)$	irreducible polynomial
I_{leader}	value of the leader

LT	leader table for periodicity algorithm
m	symbol length of a Galois field element (in bits)
$M(x)$	message polynomial
N	code length of an RS code
P_{leader}	position of the leader in the leader table
PA	periodicity algorithm
$p(x)$	primitive polynomial
ROM	read-only memory
RS	Reed-Solomon code
$R(x)$	remainder polynomial
S_i	syndrome
T	error correcting capability of an RS code
VLSI	very large scale integration
$V(x)$	received word polynomial

Chapter 1

Introduction

1.1 Statement of the problem

Most communication channels are affected by various kinds of noise. Error correcting codes were introduced to get rid of the errors caused by noise. In the early 1950's Hamming codes [1] were proposed, marking a new era in the error correcting codes. In 1960, Reed and Solomon [2] introduced their polynomial codes recently known as the Reed-Solomon (RS) codes. RS codes are a class of optimal error correcting codes in the sense that the codes have the maximum distance [3].

RS codes have many applications, such as land mobile communications [4][5], marine data communications [6] and even image transmission in digital television systems [7]. When designing an RS code for a certain application, the designer may want to examine his code to see whether it can meet the requirements of the application. He may need an RS code simulator to help

him evaluate the performance of various codes and select the best code for his particular application. Such a code simulator can also allow us to verify different properties of the RS codes, or even find some new decoding methods. For example, it can be used to test time characteristics of Peterson's and Berlekamp's methods. For different fixed error correcting capability T, when the actual error number ν is changing, the simulator can give us time estimates for the methods. Those results provide a sound basis for the selection among various choices for a certain application. In response to those needs, an RS code simulator is implemented in this thesis.

For syndrome based RS decoding, Chien search is normally performed to obtain the error location numbers. This method is the main obstacle to high speed RS decoding. Though it was believed that the roots of the error locator polynomials were randomly distributed, it is shown in this thesis that for the cases of two errors, the roots of the error locator polynomials are distributed according to certain patterns. This shows a possibility of developing a new algorithm to obtain the error location numbers. With this idea, a new algorithm called the periodicity algorithm is proposed for RS double error correction. The new algorithm requires an order of magnitude fewer operations than those required by Chien search.

1.2 Literature review

In this section, we first review general development and then syndrome based decoding. After that, the time domain decoding will be briefly mentioned. Several encoding and Chien search schemes will also be discussed.

In 1959-60, the BCH codes were independently proposed by Hocquenghem, Bose and Chaudhuri. In 1959, Hocquenghem [8] generalized Hamming codes and suggested his multi-error correcting codes. In 1960, Bose and Chaudhuri [9][10] published two papers to present their work on such codes. These codes are now known as the Bose-Chaudhuri-Hocquenghem (BCH) codes. Because of the properties of these codes, it is said [3] that the BCH codes are perhaps the most important class of codes known up to date.

At about the same time (manuscript received on January 21, 1959, published in 1960), and independently of the Bose, Chaudhuri, and Hocquenghem papers, Reed and Solomon [2] introduced a class of codes that were later established as a subclass of BCH codes. The RS codes are optimal in the sense that it is impossible for any linear codes with the same length to have a distance greater than that of the Reed-Solomon codes. Indeed they form an important and interesting subclass of BCH codes [11].

The encoding scheme introduced in Reed and Solomon's paper [2] is now known as systematic encoding. It divides the message by the generator

to get the remainder, then adds message and remainder together to get the codeword. This scheme is still widely used today .

The BCH and RS decoding methods can be divided into two categories: syndrome based decoding and time domain decoding. The syndrome based methods are now reviewed.

In 1960, Peterson [12] suggested his encoding and decoding procedures for the BCH codes. The encoder uses shift registers to construct the feedback circuits for encoding. The registers are wired according to the generator polynomial. The input of the feedback registers is the K-dimensional message, and the output is the N-dimensional codeword. As this encoding scheme uses the serial operations, its encoding speed is limited by the serial feedback operations. Peterson's decoding algorithm was the first efficient decoding algorithm for BCH codes. Since the RS codes are a subset of the BCH codes, Peterson's method can also be used for the RS codes. In his paper, Peterson not only introduced the encoding and decoding algorithm, but also showed that the BCH codes are cyclic. An evaluation of Peterson's contribution is given in Blake [3].

In 1965, Berlekamp [13] proposed his powerful iterative algorithm for decoding the BCH and RS codes. From a initial iteration table, the Berlekamp's algorithm calculates the coefficients of the error locator in $2T$ iterations. The algorithm makes use of the intermediate iteration information to save many

operations over Galois fields, and thus needs considerably less computation [3]. It was pointed out by Michelson and Levesque [14] that the Berlekamp's algorithm has a computational complexity that grows only linearly with the number of errors to be corrected while that of the Peterson's algorithm grows with approximately the square of the number of errors to be corrected.

Peterson's and Berlekamp's methods are known as algebraic methods. Besides the algebraic methods, the error locator polynomials can also be evaluated by transform methods. Since such methods are based on the transformation domain, they are also called transformation domain, spectral domain, or, more commonly, frequency domain[1] methods. Both algebraic and transform methods are considered as the syndrome based methods because their error locator evaluations are based on the syndromes.

For about ten years after Reed and Solomon suggested the RS codes, the techniques developed for decoding the RS codes primarily dealt with the raw data directly. In 1971, a frequency domain decoding method was first proposed by Mandelbaum [15]. He made use of the Chinese remainder theorem to perform the Galois field transform. After that, many researchers focused on the frequency domain RS decoding methods.

[1] More precisely, the term "frequency domain" is not very suitable because the transform used is not the Fourier transform, but the Galois field transform. These two kinds of transforms are not identical because the Galois field transform domain is not the same as the Fourier frequency domain. However, since the terms of transform- or spectral- or frequency-domains have been used for the same concept for many years, the terms are also used here interchangeably.

As pointed out by Blahut [16], the major difficulty for the frequency domain methods is that as the code length is not a power of 2, most of the fast Fourier transform (FFT) algorithms cannot be used directly to yield fast transforms.

In 1972, based on the theory of rings, Rader [17] suggested a more efficient method to calculate the Galois field transform, called Rader's Fermat number-theoretic transform. Rader solved the problem of the fast Galois field transform theoretically. It was because of Rader's work that the frequency domain RS decoding became feasible.

In 1973, based on Mandelbaum's work, Gore [18] proposed his frequency domain method to decode the Reed-Solomon codes. In his paper, he pointed out that the information can be encoded into the frequency domain and also that the error spectrum can be obtained by recursive extension.

From 1975 to 1976, Michelson [19][20] discussed and implemented a fast algorithm for the Galois field transform and used it to decode the RS codes. This was the first implemented frequency domain RS decoder.

Afterwards, in 1978, Reed [21] used the Fermat theoretic transform to implement a frequency domain RS decoder. This implementation was done entirely in software. In [21], Reed made the comparison between the transform method and the conventional method, and found that the trans-

formation method is faster.

In 1980, Miller and Truong [22] used Rader's result and developed a software implementation of an RS(255,223) decoder. In their paper, two tables were given to compare the executing time between the transform method and syndrome based method. It was shown that their decoder was three to seven time faster than a syndrome based decoder.

At the same time the RS encoding techniques were also being developed. In 1982, Berlekamp [23] proposed a bit-serial Reed-Solomon encoder which used an array of registers instead of the shift registers wired according to generator polynomial. As more registers are used in Berlekamp's scheme, it can use more intermediate information stored in the registers without doing the feedback calculations. In terms of computational complexity, this scheme requires less processing time at the expense of using more memory space.

In 1984, based on Berlekamp's scheme, Hsu [24] gave his VLSI implementation of the RS encoder. Hsu's implementation has the advantages of Berlekamp's scheme. Moreover, by slightly changing the circuit structure, Hsu's VLSI implementation can also fit different parameters.

In 1991, Seroussi [25] suggested a systolic RS encoder. He directly used the generator matrix for encoding, but he divided the hardware implementation into many identical cells to reduce the design complexity. His work had been

patented previously (US patent 4 835 775 issued in May 1989).

The syndrome calculation is the first step for the syndrome based decoding. Improving the speed of syndrome calculation can obviously increase the decoding speed.

In 1979, Truong [26] suggested a fast method for calculating the syndromes which utilizes the Chinese remainder theorem and Winograd's algorithm. It was reported that, for the 32 syndrome calculation of the RS(255,223,33), Truong's algorithm needs 90% fewer multiplications and 78% fewer additions than the conventional method of syndrome calculation.

In 1990-91, Cooper [27][28] proposed a method to calculate the error locator polynomial in one step. This method is based on modern algebra, especially on ring theory and ideal theory. There is no given comparison for this method.

Both algebraic and transformation methods need to locate the error position numbers based on the error locators.

In 1964, Chien [29] proposed his substituting method to obtain the roots of error locator polynomials. His method, which is called Chien search, substitutes all possible values of x (from α^0 to α^{N-1}) into the error locator and checks whether the result is equal to zero. Even after 30 years of its proposition, Chien search is still a practical method available in finding the

roots of the error locator polynomials.

Avoiding the heavy calculation load of Chien search has been the hot topic in RS decoding research community for a long time. One alternative to Chien search is the look-up table method. However, as the error locator polynomial for a T-error correcting RS code will have N^T possible vectors of the coefficient values, the memory space of such table will be N^T symbols. Therefore, even though the look-up table method can provide the fastest speed for RS error locating, it is not feasible for many practical applications, especially for large value of N.

In 1987, Shayan [30] suggested the binary decision approach to speed up Chien search. In his approach, a binary table was designed for double error correcting RS codes which needs one bit for each address instead of one symbol (as in a direct look-up table). However, the space factor N^2 bits is still large. This issue is a big concern for applications with large N.

In 1987, Okano and Imai [31] proposed a ROM method to decode the BCH and RS codes which uses ROM tables and the self-defined operating cells to speed up RS decoding. Besides their hardware implementation, they gave the derivation to get the solutions of fourth or lower order equations over GF(2^m). Based on this derivation, some of the operations can be avoided in hardware implementation.

For time domain methods, there are no transformation, no syndrome calculation and no Chien search. The time domain decoder works on the received raw data directly.

In 1980, Blahut [32] proposed the time domain method. Subsequently, he used the method in RS decoding and suggested two structures called "universal decoder structures" [33]. Although the time domain method can avoid using syndrome calculation and Chien search, it must always work on the raw data with vectors of length N rather than length T (for the case of frequency domain methods). Therefore the time domain method must works in N-dimensional space. Such operations limit the decoding speed of the time domain method. Blahut [33] gave a time comparison for the two methods, in which he concluded that there seems to be no obvious advantages in decoding speed for time domain methods. However, he also pointed out that the regular algorithm structure of time domain methods is very suitable for VLSI implementation. With the development of VLSI techniques, the time domain method may show its values in the near future.

There is a new development of the time domain decoding method in 1993. Based on the Berlekamp's iterative method, Sorger [34] derived the new algorithm of time domain RS decoding. His main contribution was to modify the iterative method to merge several steps, then, the decoding time can be saved.

1.3 Scope of the work

In this thesis, the main effort is put on the implementation of the RS codes simulator. This simulator is implemented in C language under UNIX operating system. It can simulate an RS code with length of up to $N = 255$, and on the error correcting capability up to $T = 16$. Both Peterson's and Berlekamp's methods are used for the RS decoding.

In the implementation, Chien search is used to exhaustively calculate all possible error location numbers, and solutions are saved in tables. From these solution tables, for the cases of double error correction, it was discovered that the roots of the error locator polynomials are not randomly distributed. With certain patterns, the error location numbers are located on several chains, and within the solution chains, those roots repeat themselves in the period of N shifts.

Although Chien search is the widely used method for RS error locating, it still needs a great deal of time to locate errors, moreover, the exact amount of time required is uncertain. This introduces difficulties for hardware implementation, making Chien search the bottleneck of RS decoding. Based on the periodicity property, a new algorithm called the periodicity algorithm is proposed. With fewer than eight additions and four decision operations, the periodicity algorithm can obtain the error location numbers easily. At present, the periodicity algorithm is available for the case of double error

correction.

1.4 Organization of the thesis

This thesis consists of six chapters. **Chapter 2**, encoder and decoder of RS codes, discusses the basic knowledge of the RS codes. **Chapter 3**, implementation of RS code simulator, presents the implementation of the RS code simulator. **Chapter 4**, periodicity algorithm, proposes a new decoding algorithm for the double error correcting RS codes. **Chapter 5**, comparison with other methods, based on the comparison, concludes that the periodicity algorithm needs the fewest operations among the available decoding methods, thereby making it the best algorithm. **Chapter 6** presents the conclusions.

Chapter 2

RS Code Simulator

The elementary knowledge of RS codes is discussed in this chapter to provide the necessary background. Most of the theorems are stated without proof, the details of which can be found in the proper references. In addition, many examples are provided to make the discussion easily understandable.

2.1 Primitive polynomials

A polynomial $f(x)$ with single variable x and with coefficients from $GF(2)$ is of the following form:

$$f(x) = f_0 + f_1 x + f_2 x^2 + \cdots + f_N x^N \tag{2.1}$$

where $f_i = 0$ or 1 for $0 \leq i \leq N$. The degree of a polynomial is the largest power of x with a nonzero coefficient. For the polynomial above, if $f_N = 1$, $f(x)$ is a polynomial of degree N. The polynomial $f(x) = f_0$ is a zero degree

m	primitive polynomial
3	$1 + x + x^3$
4	$1 + x + x^4$
5	$1 + x^2 + x^5$
6	$1 + x + x^6$
7	$1 + x^3 + x^7$
8	$1 + x^2 + x^3 + x^4 + x^8$
9	$1 + x^4 + x^9$
10	$1 + x^3 + x^{10}$

Table 2.1: List of primitive polynomials

polynomial.

A polynomial $i(x)$ over $GF(2)$ of degree m is said to be *irreducible* over $GF(2)$ if $i(x)$ is not divisible by any polynomial over $GF(2)$ of degree less than m but greater than zero.

An irreducible polynomial $p(x)$ of degree m is said to be *primitive* if the smallest positive integer N for which $p(x)$ divides $x^N + 1$ is $N = 2^m - 1$. For a given m, there may be more than one primitive polynomials of degree m [35]. The primitive polynomials used in this thesis, which have the smallest number of terms for each m, are given in Table 2.1. More detailed discussion is given in [35].

2.2 Construction of Galois field $GF(2^m)$

The method for constructing the Galois field of 2^m elements ($m \geq 3$) from the binary field $GF(2)$ is presented in this section. The process begins with the two elements 0 and 1, from $GF(2)$ and the an element α. If the element α satisfies $\alpha^{2^m-1} = 1$, it is called primitive element. Then the multiplication "$\cdot$" is introduced to give a sequence of powers of α as follows:

$0 \cdot 0 = 0,$

$0 \cdot 1 = 1 \cdot 0 = 0,$

$1 \cdot 1 = 1,$

$0 \cdot \alpha = \alpha \cdot 0 = 0,$

$1 \cdot \alpha = \alpha \cdot 1 = \alpha,$

$\alpha^2 = \alpha \cdot \alpha,$

$\alpha^3 = \alpha \cdot \alpha \cdot \alpha,$

.....

$\alpha^j = \alpha \cdot \alpha \cdots \alpha$ (j times),

.....

After α^{2^m-2}, the elements will repeat due to the fact that $\alpha^{2^m-1} = 1$ and $\alpha^{2^m} = \alpha$.

Now we have the following set of elements on which a multiplication "$\cdot$"

is defined:

$$F = \{0, 1, \alpha, \alpha^2, \cdots, \alpha^j, \cdots, \alpha^{2^m-2}\}. \tag{2.2}$$

The element 1 is sometimes denoted as α^0.

Example 2.1

Let $m = 4$. The polynomial $p(x) = 1 + x + x^4$ is a primitive polynomial over $GF(2)$. Set $p(\alpha) = 1 + \alpha + \alpha^4 = 0$. Then $\alpha^4 = 1 + \alpha$. Using this, we can construct $GF(2^4)$. The elements of $GF(2^4)$ are given in Table 2.2. The term $\alpha^4 = 1 + \alpha$ is used repeatedly to form the polynomial representations for the elements for the elements of $GF(2^4)$. For example,

$\alpha^5 = \alpha \cdot \alpha^4 = \alpha(1 + \alpha) = \alpha + \alpha^2$,

$\alpha^6 = \alpha \cdot \alpha^5 = \alpha(\alpha + \alpha^2) = \alpha^2 + \alpha^3$,

$\alpha^7 = \alpha \cdot \alpha^6 = \alpha(\alpha^2 + \alpha^3) = \alpha^3 + \alpha^4 = \alpha^3 + 1 + \alpha = 1 + \alpha + \alpha^3$, $\cdots\cdots$

The complete set of the elements in $GF(2^4)$ are given in Table 2.2. The elements in Table 2.2 are represented in three forms, that is, power, polynomial and 4-tuple representations. All those representations will be used in the discussion of the RS codes.

The multiplications in Galois fields are actually performed by adding the exponential powers together, that is, $\alpha^i \cdot \alpha^j = \alpha^{i+j}$. From the given GF value α^i to get its corresponding power value i, such operation is sometimes called GF-log operation. Its inverse is called GF-anti-log operation denoted as GF-

Power representation	Polynomial representation	4-Tuple representation
0	0	(0 0 0 0)
1	1	(1 0 0 0)
α	α	(0 1 0 0)
α^2	α^2	(0 0 1 0)
α^3	α^3	(0 0 0 1)
α^4	$1 + \alpha$	(1 1 0 0)
α^5	$\alpha + \alpha^2$	(0 1 1 0)
α^6	$+ \alpha^2 + \alpha^3$	(0 0 1 1)
α^7	$1 + \alpha + \alpha^3$	(1 1 0 1)
α^8	$1 + \alpha^2$	(1 0 1 0)
α^9	$\alpha + \alpha^3$	(0 1 0 1)
α^{10}	$1 + \alpha + \alpha^2$	(1 1 1 0)
α^{11}	$\alpha + \alpha^2 + \alpha^3$	(0 1 1 1)
α^{12}	$1 + \alpha + \alpha^2 + \alpha^3$	(1 1 1 1)
α^{13}	$1 + \alpha^2 + \alpha^3$	(1 0 1 1)
α^{14}	$1 + \alpha^3$	(1 0 0 1)

Table 2.2: Three representations for the elements of $GF(2^4)$

$\log^{-1}$. Therefore, the GF multiplications and divisions can be considered as the combination of GF-log, GF-$\log^{-1}$ and several addition operations. It should be noticed that the Galois field addition is carried out by a bitwise XOR operation.

2.3 RS code definition

Reed-Solomon codes are an important subset of the BCH codes, which are a large class of powerful cyclic codes. It has been shown by Blahut [11] that the Reed-Solomon codes have maximum-distance and optimal structure [3].

The RS codes can be defined [35][36] as:

For any given positive integer $m \geq 3$ and the correcting capability T, there exists an RS code such that

(i) code length $N = 2^m - 1$

(ii) number of information symbols $K = N - 2T$

(iii) minimum distance $D = 2T + 1$.

RS codes can correct T random errors. The following types of burst errors can also be corrected [36]:

- 1 burst of total length: $b_1 = (T-1)m + 1$ bits,
- or, 2 bursts of total length: $b_2 = (T-3)m + 3$ bits,

$\vdots$

- or, p bursts of total length: $b_p = (T - 2p + 1)m + (2p - 1)$ bits.

where, p is an integer number, and $(T - 2p + 1)$ is positive.

2.4 RS encoding

There are many ways to construct the generator polynomial $G(x)$. It can be selected as in following form [11],

$$G(x) = (x - \alpha)(x - \alpha^2) \cdots (x - \alpha^{2T}). \tag{2.3}$$

More generally, one can choose any integer j_0 for a Reed-Solomon code [11]. The corresponding generator polynomial then has the form

$$G(x) = (x - \alpha^{j_0})(x - \alpha^{j_0+1}) \cdots (x - \alpha^{j_0+2T-1}). \tag{2.4}$$

It is clear that Eq. 2.3 is a special case of Eq. 2.4, where $j_0 = 1$.

When j_0 is properly selected, the generator polynomial $G(x)$ will have the symmetric coefficient format. This format of the $G(x)$ is called *self-reciprocal.* If $j_0 = 2^{m-1} - T$, the self-reciprocal is given as [36][37]

$$G(x) = \prod_{i=l}^{2T+l-1} (x + \alpha^i) = g_0 + g_1 x + \cdots + g_{2T-1} x^{2T-1} + g_{2T} x^{2T}, \tag{2.5}$$

where $l = 2^{m-1} - T$, and the coefficients are such that $g_0 = g_{2T}$, $g_1 = g_{2T-1}$, $\cdots$. In other words, the i-th and $(2T - i)$-th coefficients of the $G(x)$ are identical. Therefore, half of the memory space for storing the coefficients can be saved.

There are also many ways to encode. In a non-systematic codeword $C(x)$, the message $M(x)$ is not explicitly present. When the message $M(x)$ and the generator polynomial $G(x)$ are given, non-systematic encoding can be represented as [11]

$$C(x) = M(x)G(x). \tag{2.6}$$

In the systematic encoding [35], the codeword is obtained by

$$\frac{x^{2T}M(x)}{G(x)} = A(x) + \frac{B(x)}{G(x)}, \tag{2.7}$$

$$C(x) = x^{2T}M(x) + B(x). \tag{2.8}$$

where,

$$B(x) = b_0 + b_1 x + \cdots + b_{2T-1}x^{2T-1} \tag{2.9}$$

is the parity check polynomial. It is easy to see that the message $M(x)$ is explicitly present in codeword $C(x)$.

In the implementation of the RS codes simulator, the self-reciprocal generator polynomial and the systematic encoding method are used.

2.5 Noisy channel

Most channels are affected by various kinds of noise. Due to the noise, the received word usually contains errors. The error pattern can be expressed as

$$E(x) = e_0 + e_1 x + \cdots + e_{T-1}x^{T-1}. \tag{2.10}$$

Those non-zero e_i's are caused by the noise [35].

The received word $V(x)$ can be expressed as

$$V(x) = C(x) + E(x), \tag{2.11}$$

2.6 Decoding

A major concern of many coding theorists is the practical implementation of encoding and decoding schemes. For most schemes, the encoding operation is simple and inexpensive in terms of its software and digital circuitry. Unfortunately, the decoding operation is expensive and presents the biggest obstacle in applications of error correcting codes [3]. Therefore, most effort is focused towards decoding methods.

In this section, two of the most common algebraic methods, Peterson's and Berlekamp's, are discussed. Both methods are used in the RS decoding simulator which will be described later in this chapter.

2.6.1 Syndrome calculation

The syndrome calculation is the first step for RS decoding with algebraic methods.

A self-reciprocal polynomial shown in Eq. 2.5 is chosen as the generator polynomial $G(x)$. The input to the RS decoder is the received codeword which can be expressed as :

$$V(x) = C(x) + E(x) = \sum_{i=0}^{N-1} v_i x^i \tag{2.12}$$

where v_{N-1} is the first received symbol.

The syndromes can be calculated by

$$S_i = V(\alpha^i) = C(\alpha^i) + E(\alpha^i) = E(\alpha^i) \tag{2.13}$$

where $N-T-1 \leq i \leq N+T$. There are two ways to obtain the syndromes. One is to calculate them directly by

$$S_i = V(\alpha^i) \tag{2.14}$$

The other is to get the remainder $r(x)$ from the received word, via Eq. 2.15 and then, calculate the syndromes [36].

$$\begin{array}{c} \frac{V(x)}{G(x)} = A(x) + \frac{r(x)}{G(x)} \\ S_i = r(\alpha^i). \end{array} \tag{2.15}$$

In Eq 2.10, if only the non-zero e_i's are counted and $E(x)$ has T errors, $E(x)$ can be written as [36]

$$E(x) = Y_1X_1 + Y_2X_2 + \cdots + Y_TX_T, \tag{2.16}$$

where X_i are the error location numbers and Y_i are values of the errors. According to Eq. 2.13 and Eq. 2.16, the partial syndromes S_i are given by

$$\left\{ \begin{array}{l} S_l = Y_1X_1^l + Y_2X_2^l + \cdots + Y_TX_T^l \\ S_{l+1} = Y_1X_1^{l+1} + Y_2X_2^{l+1} + \cdots + Y_TX_T^{l+1} \\ S_{l+2} = Y_1X_1^{l+2} + Y_2X_2^{l+2} + \cdots + Y_TX_T^{l+2} \\ \quad \vdots \\ S_{l+2T-1} = Y_1X_1^{l+2T-1} + Y_2X_2^{l+2T-1} + \cdots + Y_TX_T^{l+2T-1}, \end{array} \right. \tag{2.17}$$

where X_i and Y_i are unknown.

2.6.2 Error locator polynomial

Peterson's method

In 1960, Peterson [12] proposed his decoding method for BCH codes. Since his method works directly on the received data, it is called Peterson's direct method. It can also be used for RS decoding. The detailed proof of Peterson's method is given in [12].

The error locator polynomial is

$$\begin{aligned}\Sigma(x) &= (x + X_1)(x + X_2)\cdots(x + X_T) \\ &= x^T + \sigma_1 x^{T-1} + \cdots + \sigma_{T-1}x + \sigma_T.\end{aligned} \tag{2.18}$$

where

$$\begin{aligned}\sigma_1 &= X_1 + X_2 + \cdots + X_T \\ \sigma_2 &= X_1X_2 + X_2X_3 + \cdots + X_{T-1}X_T \\ &\vdots \\ \sigma_T &= X_1X_2\cdots X_T.\end{aligned}$$

Those σ_i can be calculated from the known partial syndromes shown in Eq 2.17. By [36] this gives

$$\sigma_T S_j + \sigma_{T-1}S_{j+1} + \cdots + \sigma_1 S_{j+T-1} = S_{j+T}, \tag{2.19}$$

where $l \leq j \leq T$. Eq. 2.19 can then be re-written in linear equation form as

$$\begin{bmatrix} S_l & S_{l+1} & S_{l+2} & \cdots & S_{l+T-1} \\ S_{l+1} & S_{l+2} & S_{l+3} & \cdots & S_{l+T} \\ S_{l+2} & S_{l+3} & S_{l+4} & \cdots & S_{l+T+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ S_{l+T-1} & S_{l+T} & S_{l+T+1} & \cdots & S_{l+2T-2} \end{bmatrix} \begin{bmatrix} \sigma_T \\ \sigma_{T-1} \\ \sigma_{T-2} \\ \vdots \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} S_{l+T} \\ S_{l+T+1} \\ S_{l+T+2} \\ \vdots \\ S_{l+2T-1} \end{bmatrix} \tag{2.20}$$

Now the σ_i's can be solved. The solutions to Eq. 2.20, $\sigma_1, \cdots, \sigma_T$, are the coefficients of the error locator polynomial shown in Eq. 2.18.

To solve Eq. 2.20, Gaussian elimination can be used. With Gaussian elimination, an upper-right triangular matrix is obtained and the backward substitutions are performed to get all the σ_i's.

Berlekamp's method

Peterson's method is easy to understand. However, when implementing a decoder, one has to use a method which is computationally efficient. Peterson's method requires a T by T matrix inversion. When T is large, this involves large numbers of arithmetic operations. By using Berlekamp's method, we can get around this problem. Berlekamp's method uses iterations to get the error locator. The detailed proof is given in [11]. Michelson and Levesque [14] pointed out that Berlekamp's algorithm has a computational complexity that grows linearly with the number of errors to be corrected while that of the Peterson's method grows approximately proportional to the square of the number of errors to be corrected.

Table 2.3 is the initial table for the iterations of the Berlekamp's method. Lin and Costello [35] give the Berlekamp's iterative procedure as follows

1. According to the given received word $V(x)$, calculate each syndrome S_i. This step is the same as in Peterson's method as shown in Eq. 2.14

μ	$\sigma^{(\mu)}(X)$	d_μ	l_μ	$\mu - l_\mu$
-1	1	1	0	-1
0	1	S_l	0	0

Table 2.3: Initial table for Berlekamp's iterations

or Eq. 2.15. When all the S_i's are calculated, the iteration begins.

2. In the μth iteration, $\sigma^{(\mu+1)}(x)$ is determined, where $\sigma^{(\mu)}(x)$ is the μ-th minimum degree polynomial. It has following form:

$$\sigma^{(\mu)}(x) = 1 + \sigma_1^{(\mu)}x + \sigma_2^{(\mu)}x^2 + \cdots + \sigma_{l_\mu}^{(\mu)}x^{l_\mu},$$

where l_μ is the degree of $\sigma^{(\mu)}(x)$.

3. In each iteration:

If $d_\mu = 0$, $\sigma^{(\mu+1)}(x) = \sigma^{(\mu)}(x)$, and $l_{\mu+1} = l_\mu$, then, enter step 4.

If $d_\mu \neq 0$, find another row ρ in the table such that $d_\rho \neq 0$ and $\rho - l_\rho$ has the maximum value. After selecting the ρ, the $\sigma^{(\mu+1)}(x)$ can be found by

$$\sigma^{(\mu+1)}(x) = \sigma^{(\mu)}(x) - d_\mu \cdot d_\rho^{-1} \cdot x^{(\mu-\rho)} \cdot \sigma^{(\rho)}(x).$$

Then, $l_{\mu+1} = \max(l_\mu, l_\rho + \mu - \rho)$.

4. To prepare next loop of the iteration, calculate $d_{\mu+1}$:

$$d_{\mu+1} = S_{\mu+2} + \sigma_1^{(\mu+1)} S_{\mu+1} + \cdots + \sigma_{l_{\mu+1}}^{(\mu+1)} S_{\mu+2-l_{\mu+1}},$$

where, $\sigma_i^{(\mu+1)}$ are the coefficients in $\sigma^{(\mu+1)}(x)$.

5. Enter next iteration loop, *i.e.* go to step 2.

Let ν be the actual number of errors which happen in the noisy channel. If $T \geq \nu$, Berlekamp's method needs $2T$ iterations. If $T > \nu$, it still takes $2T$ iterations. But, some of the d_μ may be zero [35]. Once the d_μ is zero, the corresponding iteration will be skipped, and the operations will be saved. According to this, when the d_μ is zero twice, that means the $T > \nu$, the iteration loop can stop. For $T < \nu$, the iterations stop at $\mu = 2T$ so that the degree of the $\sigma^{(\mu)}$ is at most T. As T degree polynomial has no more than T roots, Berlekamp's method can do nothing when $T < \nu$. All these conclusions are derived in [35]. Fig. 3.8 presents the flow chart of the Berlekamp's method. It should be noted that the error locator polynomial obtained by Berlekamp's method is slightly different from Eq. 2.18. When X is a root of Eq. 2.18, the inverse $X' = 1/X$ is the root of the error locator obtained by Berlekamp's method. Therefore, $\sigma(x)$ is also called inverse error locator polynomial of $\Sigma(x)$.

2.6.3 Chien search

After the error locator polynomial is evaluated, the error location numbers can be found by using Chien search.

The error locator polynomial as in Eq. 2.18 is re-written below. The error location number can be found by setting the polynomial to zero and solving it.

$$\Sigma(x) = x^T + \sigma_1 x^{T-1} + \cdots + \sigma_{T-1} x + \sigma_T = 0$$

For this high order equation, there is no simple way to get its roots. Chien search substitutes 1, α, α^2, $\cdots$, *etc.*, into above equation to see whether the left hand of it is zero.

2.6.4 Error value calculation and error correction

When the error location numbers X_i's are found, Eq 2.17 becomes a linear system of equations. It can be re-written as the following equation [36]

$$\begin{bmatrix} X_1^l & X_2^l & X_3^l & \cdots & X_T^l \\ X_1^{l+1} & X_2^{l+1} & X_3^{l+1} & \cdots & X_T^{l+1} \\ X_1^{l+2} & X_2^{l+2} & X_3^{l+2} & \cdots & X_T^{l+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ X_1^{l+T} & X_2^{l+T} & X_3^{l+T} & \cdots & X_T^{l+T} \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ \vdots \\ Y_T \end{bmatrix} = \begin{bmatrix} S_l \\ S_{l+1} \\ S_{l+2} \\ \vdots \\ S_{l+T-1} \end{bmatrix} \tag{2.21}$$

The error values Y_i's can then be determined by using Gaussian elimination.

The decoded error pattern $\hat{E}(x)$ can be expressed as

$$\hat{E}(x) = Y_1 X_1 + Y_2 X_2 + \cdots + Y_T X_T \tag{2.22}$$

If $T \geq \nu$, $\hat{E}(x)$ should be equal to $E(x)$ as in Eq. 2.10.

Essentially error correction adds the decoded error pattern to the received word to get the decoded word $D(x)$, *i.e.*

$$D(x) = V(x) + \hat{E}(x). \tag{2.23}$$

Chapter 3

Implementation of RS Code Simulator

3.1 Overview of the Simulator

An RS code simulator has been implemented in C under UNIX operating system. Using this simulator, the user can simulate RS encoding and decoding procedures with different code lengths and error correcting capability, allowing the user to select the best code among the various choices available to fit a given application. The implementation of the simulator is described in the remainder of the chapter. Its software structure, flow charts and the time characteristic are also discussed in detail.

An overview of the simulator is given in Fig. 3.1. The details of those blocks are shown in the following figures. In Fig. 3.1, block 3 is for forming $GF(2^m)$ and $G(x)$. The details of block 3 is shown in Fig. 3.2 and Fig. 3.4. The block 4 is for forming systematic codeword. Fig. 3.3 presents more

details. Block 5 is for noisy channel simulation which is shown in Fig. 3.5. Block 6 is for syndrome calculation. It is also described in Fig. 3.6. Block 7 and 8 are for error locator evaluation. Block 9 is for Chien search which is shown in details in Fig. 3.9. The input data is read from the input files. The output of the simulator is simultaneously presented on the screen as well as written in the output files. By calling other executable modules in different blocks from the main program, it is possible to combine Peterson's and Berlekamp's program blocks together to form an easy-to-use interface. The flow charts for both blocks are shown in Fig. 3.1.

As space is limited, it is not possible to present all the details in one flow chart. Therefore, the flow charts of the simulator are arranged in several levels such that zooming into a block in a higher level details the corresponding flow chart at a lower level.

3.2 Forming Galois field

The flow chart of forming the Galois field is given in Fig. 3.2. The coefficients of the primitive polynomials can be expressed in binary format. For example, the eighth order primitive polynomial $1 + x^2 + x^3 + x^4 + x^8$ has coefficients 1,0,0,0,1,1,1,0,1, which can be expressed in binary format as 100011101. This format is convenient because it can make use of the shift register to perform the Galois field operations. When forming the Galois

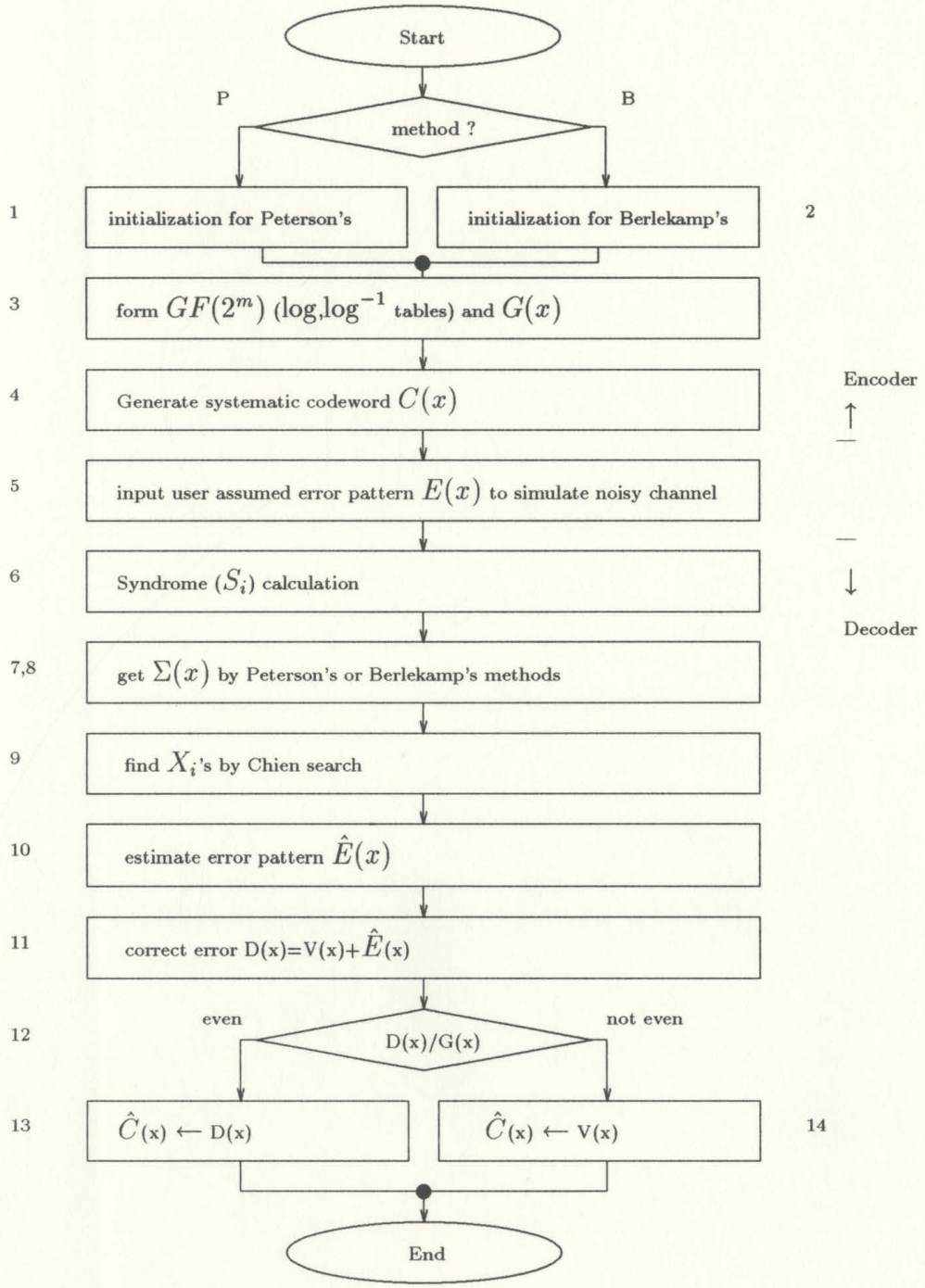

Figure 3.1: Overview of the simulator

field, let α be the root of the equation

$$p(\alpha) = 1 + x^2 + x^3 + x^4 + x^8 = 0, \tag{3.1}$$

that is,

$$1 + x^2 + x^3 + x^4 = x^8. \tag{3.2}$$

In binary, Eq. 3.2 has the format

$$100000000 = 000011101. \tag{3.3}$$

Eq. 3.3 will be used as "adjustment" if an overflow is detected when forming the Galois field.

It should be mentioned here that the GF-log and GF-log^{-1} operations are based on the GF table in the implementation. The table is stored according to the GF power so that it can be used as the pointer to the desired values, and *vice versa.*

3.3 Forming generator polynomial and encoding

As mentioned in Chapter 2, the self-reciprocal is used as the generator. To form the self-reciprocal, $2T$ number of the first order polynomials are multiplied together. Each of those polynomials is of the form $(x + \alpha^i)$. By changing the i, the self-reciprocal can be formed easily.

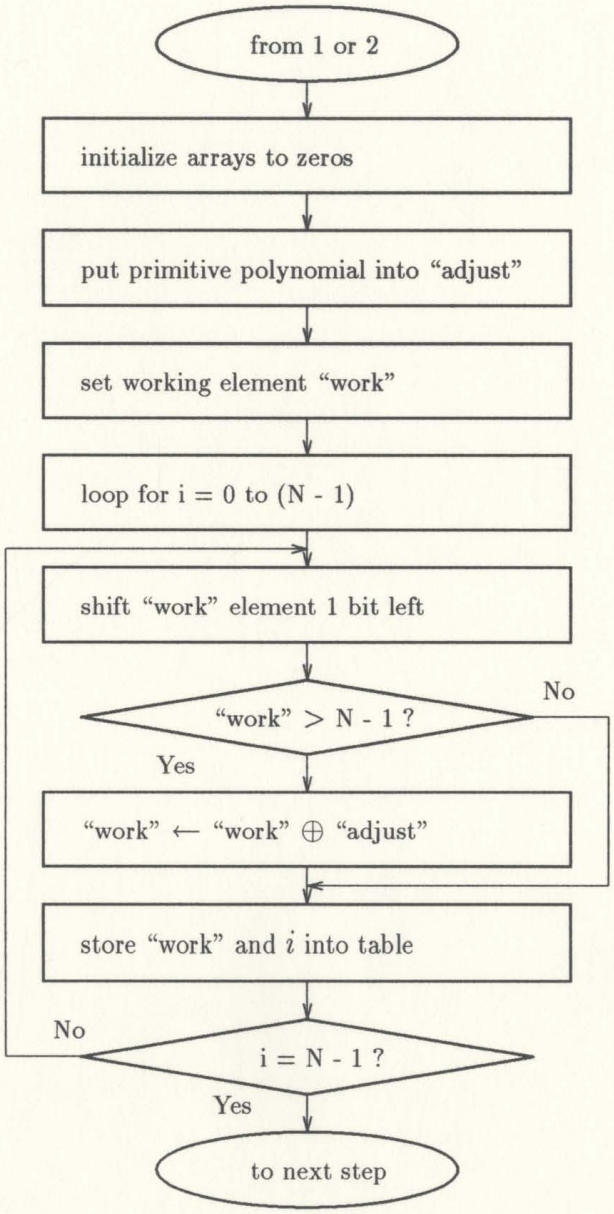

Figure 3.2: Construction of the Galois field

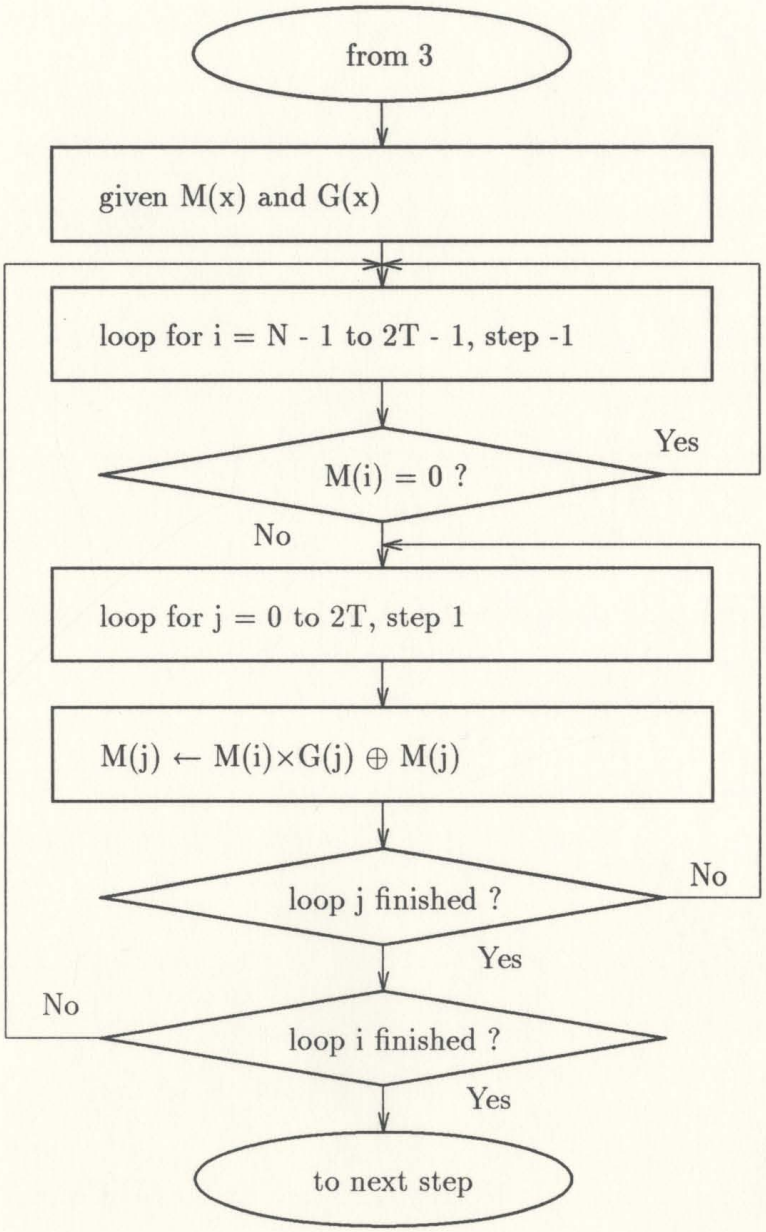

Figure 3.3: Calculation of remainder $B(x)$

Finding the remainder is an important step in encoding. The flow chart to get the remainder is given in Fig. 3.3. In fact, this is the flow chart to perform the polynomial division, which can be used for many other cases of polynomial division. The dividend here is the message polynomial. The divisor is the generator polynomial. The loop i is for $M(x)$, and the loop j is for $G(x)$. From the flow chart, it can be clearly seen that the operations are actually shift-addition operations. Thus, in the Galois field, the shift-addition is equivalent to shift-subtraction, which is the basic element of the division.

After obtaining the remainder, the rest of encoding is to add the remainder to the shifted message polynomial to get the codeword $C(x)$. This step has been shown in Fig. 3.4.

3.4 Simulating noisy channel

The fifth block in Fig. 3.1 is the noisy channel simulation. It is detailed in Fig. 3.5. The error pattern $E(x)$ is read from the input data file. The codeword $C(x)$ is from the encoding simulation.

3.5 Calculating syndromes

Syndrome calculation is shown in Fig. 3.6. Two loops are set for $2T$ syndromes and N GF multiplication-accumulations.

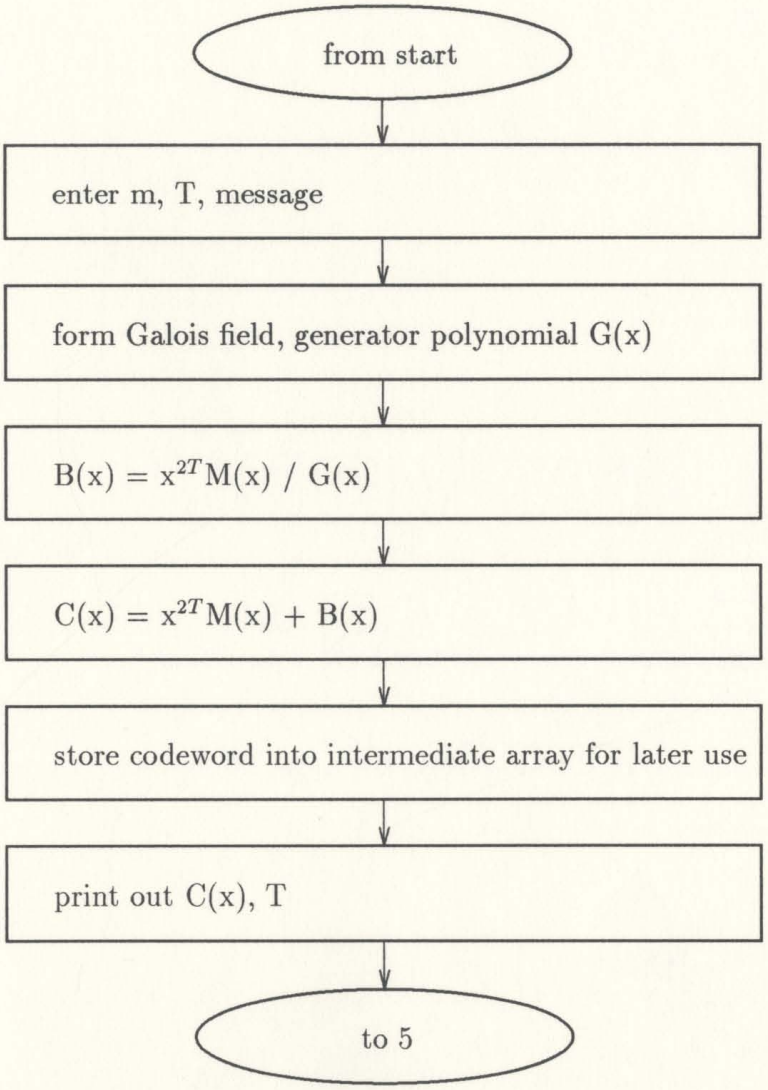

Figure 3.4: Simulation of the encoder

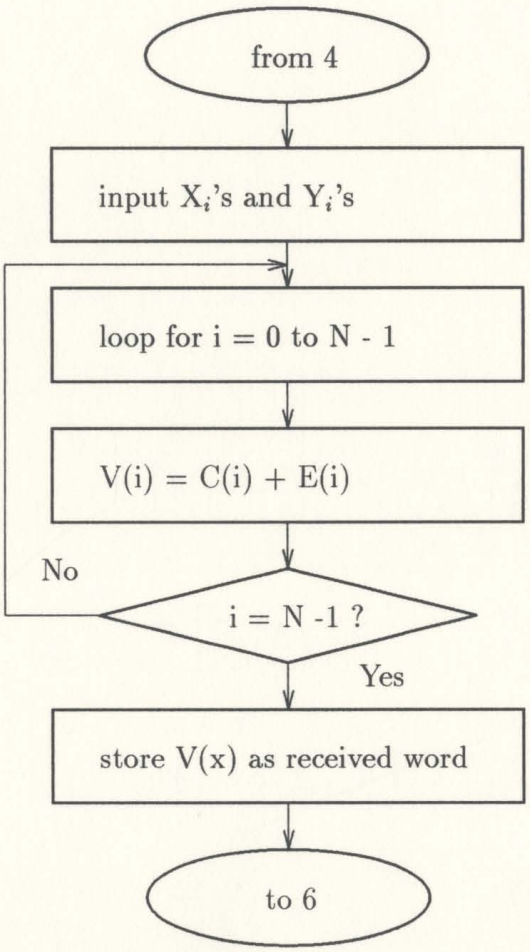

Figure 3.5: Noisy channel simulation

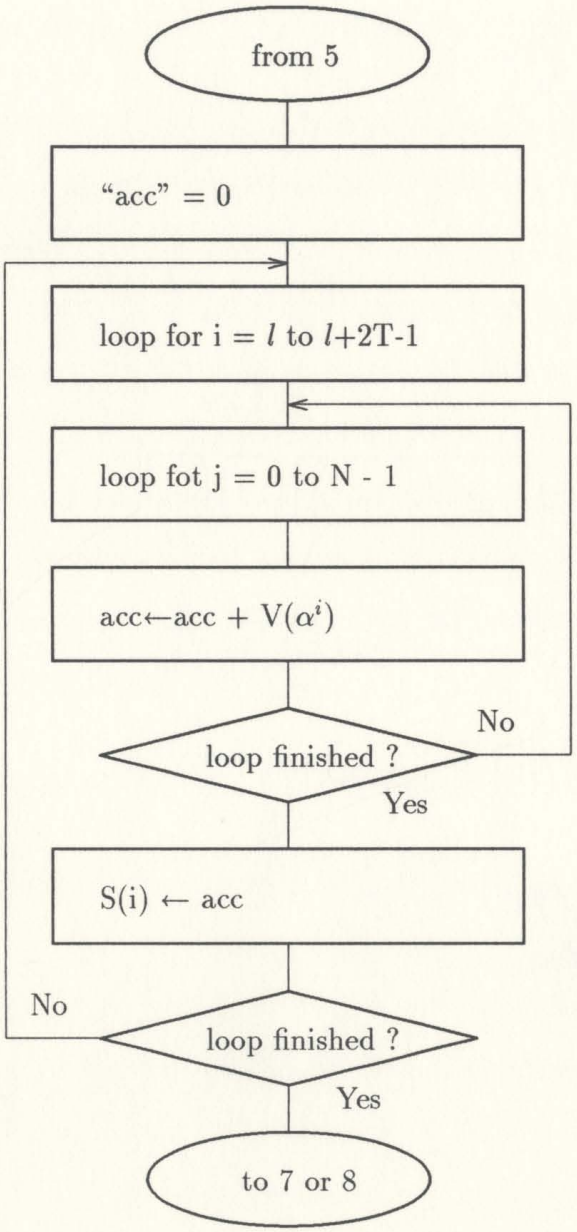

Figure 3.6: Syndrome calculation

3.6 Implementing Peterson's method

In the simulator, the error locator polynomial can be obtained by using Peterson's The Peterson's method flow chart is shown in Fig. 3.7. Note that Gaussian elimination is used to solve the linear equations via loops i and j. Before the backward substitutions, the determinant of the matrix, Δ, is checked to see whether it is zero or not. If $\Delta = 0$, it means that the equations are not independent. The equations can not give T σ_i's. For such case, the actual number of errors is assumed to be fewer than T. When this happens, the order of the matrix should be reduced. Afterwards, similar procedures described above have to be taken again until the determinant is non-zero then σ_i's can be determined.

3.7 Implementing Berlekamp's method

Berlekamp's method has been described in in Section 2.6.2. Its flow chart is shown in Fig. 3.8. The iterative operations are explained in Section 2.6.2. General speaking, when actual error number ν is the same as the error correcting capability T, that is $\nu = T$, Berlekamp's method needs $2T$ iterations. After those iterations, the so-called inverse error locator polynomial $\sigma(x)$ is obtained. When $\nu > T$, the decoder cannot know anything before obtaining $\hat{E}(x)$. Therefore, the Berlekamp's method still needs $2T$ iterations to get the error locator. However, when $\nu < T$, it may happen that the discrepancy d_μ may be zeros for two successive iterations. This could be used as the criteria

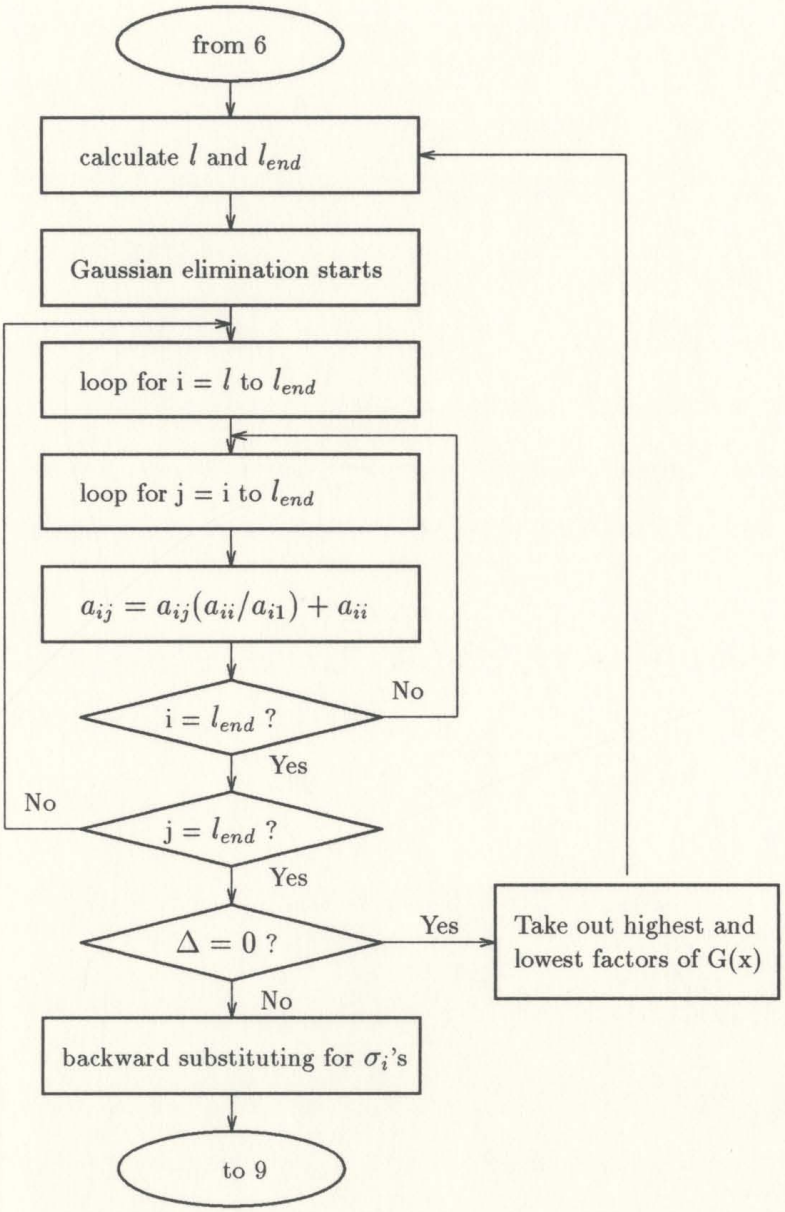

Figure 3.7: Peterson's method to get σ_i's

to stop the iterations. Either of such conditions will give an inverse error locator which will be used in Chien search.

3.8 Implementing Chien search

Chien search is the most time-consuming procedure in RS decoding. It is shown in detail in Fig. 3.9. Loop i is set to fit the worst case for Chien search. However, if T roots have been found in the search, the search should be broken right away. The decision "all roots found ?" is set for this purpose. After Chien search gives all the error location numbers, the error values should be calculated. As mentioned in last chapter, the roots of $\Sigma(x)$ and $\sigma(x)$ are inverse. In the implementation, such GF inverse operation is performed by following relationship

if $\beta = \alpha^i$ is a root of $\Sigma(x)$,

then, $\beta' = 1/\beta = \alpha^{N-i}$ will be a root of $\sigma(x)$.

3.9 Calculating error values

The calculation of error values employs Gaussian elimination to get the upper-right matrix and then to calculate the error values by backward substitution. This algorithm has been discussed in Fig. 3.7, and will not be repeated here. As the addition of the decoded error pattern to the received word is similar to the procedure shown in Fig. 3.5, it is also omitted.

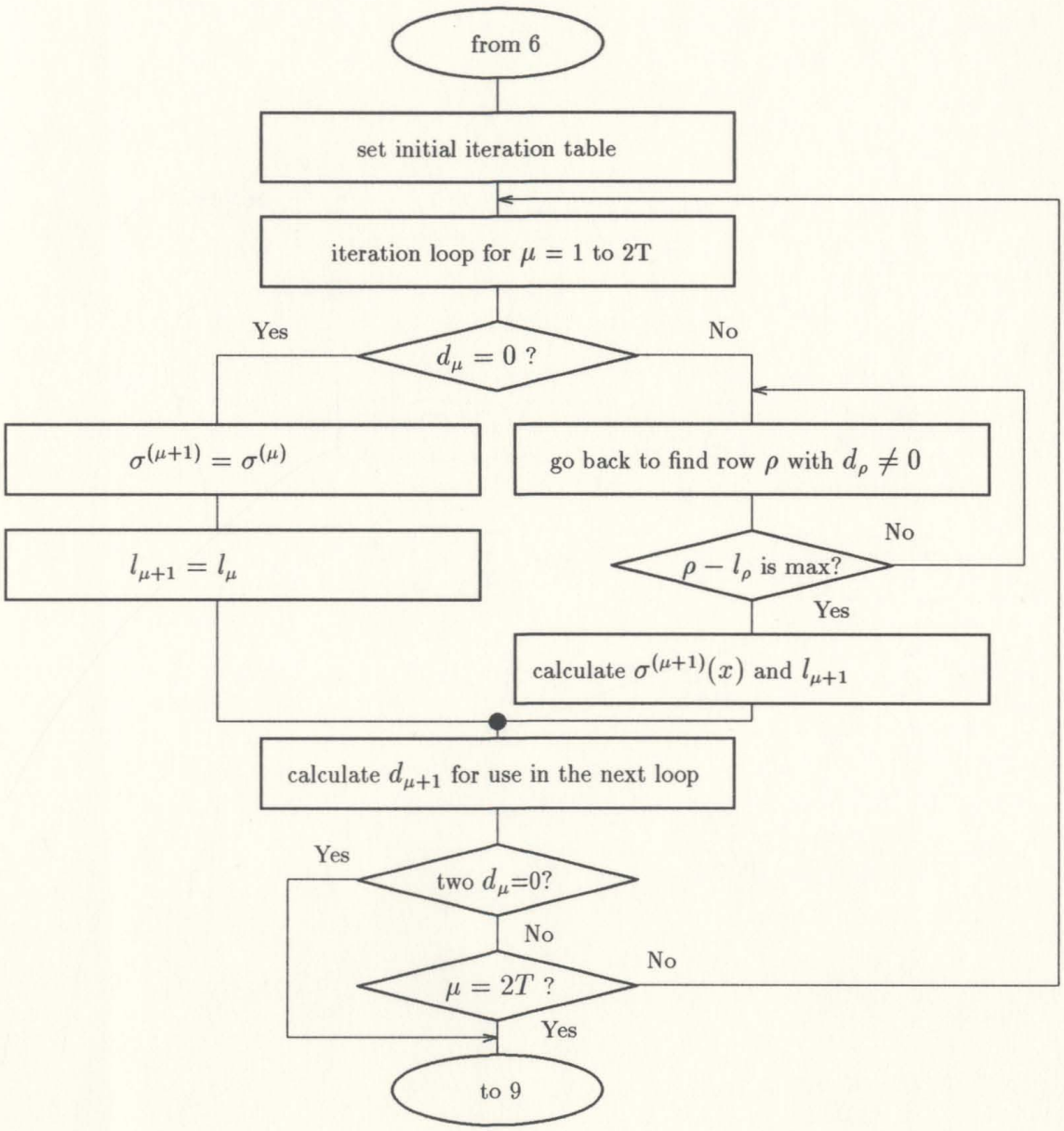

Figure 3.8: Berlekamp's algorithm to determine σ_i's

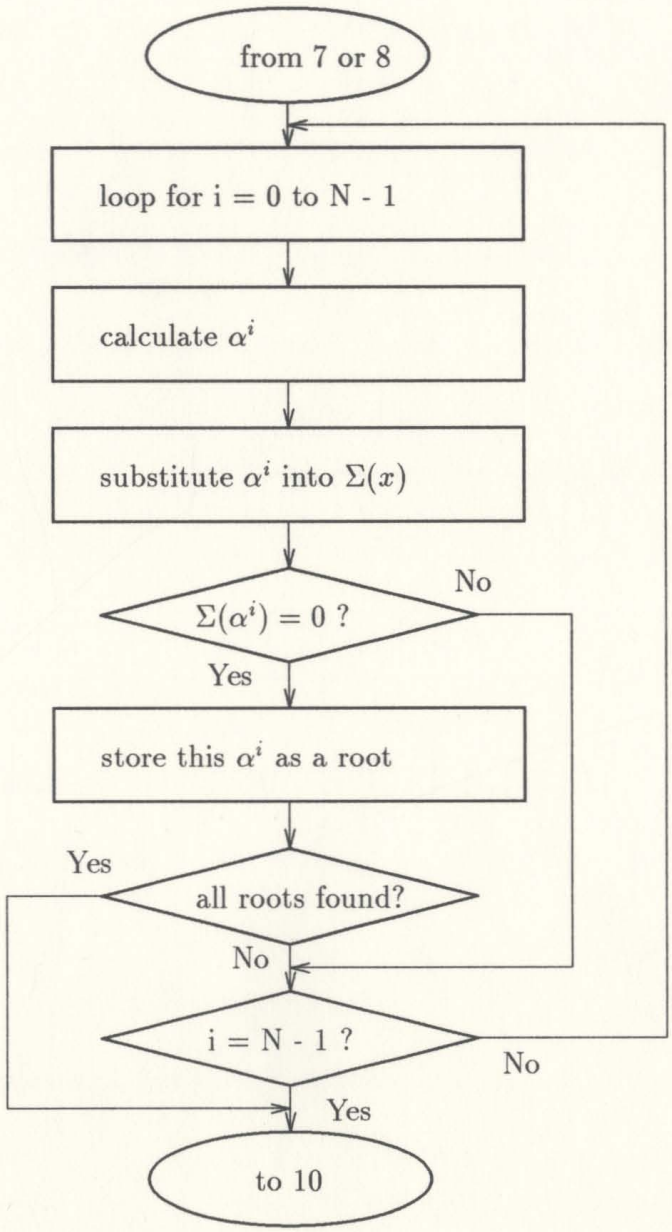

Figure 3.9: Chien search

All the final and intermediate outputs of the simulator are written into the output data file. The user can then use this file to compare different codes and select the optimal code for a given application.

3.10 Examples for the simulator

A simple example ($m = 3$) is used at first to show how the encoder and decoder work. This example will also be used to test the simulator. Then several other examples ($m = 4$) are used to illustrate different cases when $T = \nu$, $T < \nu$, and $T > \nu$.

Example 3.1

Given $m = 3$, $T = 1$. $N = 2^m - 1 = 7$.

1. **Encoding**

   $l = 2^{m-1} - T = 2^{3-1} - 1 = 3$, $2T + l - 1 = 2 + 3 - 1 = 4$.

   Then,

   $G(x) = (x + \alpha^3)(x + \alpha^4) = 1 + \alpha^6 x + x^2$

   The message "0010" is used as the input.

   $M(x) = x$

   From $M(x)/G(x)$, $B(x)$ is obtained

   $B(x) = \alpha^4 x + \alpha^6$.

   Hence,

$C(x) = x^{2T}M(x) + B(x) = x^3 + \alpha^4 x + \alpha^6.$

2. **Noisy Channel**

   Error pattern is given as

   $E(x) = x.$

   Then,

   $V(x) = C(x) + E(x) = x^3 + \alpha^4 x + \alpha^6 + x = x^3 + \alpha^5 x + \alpha^6.$

3. **Syndrome Calculation**

   $V(x) = x^3 + \alpha^5 x + \alpha^6.$

   $S_i = V(\alpha^i) = C(\alpha^i) + E(\alpha^i) = E(\alpha^i)$

   where $N - T - 1 \leq i \leq N + T$, that is, $3 \leq i \leq 4$.

   $S_3 = \alpha^3$

   $S_4 = \alpha^4$

   Error Locator Evaluation

   $S_3\sigma_1 = S_4$, $\alpha^3\sigma_1 = \alpha^4$.

   Then, $\sigma_1 = \alpha$.

   $\Sigma(x) = x + \alpha.$

4. **Error Location Numbers**

   Obviously, $\Sigma(\alpha) = 0$, therefore,

   $X_1 = \alpha.$

5. **Error Value Calculation**

   $X_1^3 Y_1 = S_3$, $\alpha^3 Y_1 = \alpha^3$, hence, $Y_1 = 1$.

   $\hat{E}(x) = x$

   $D(x) = V(x) + \hat{E}(x) = x^3 + \alpha^4 x + \alpha^6$.

   The input and output for the Example 1 is shown in Table 3.1.

Example 3.2

- Given: $m = 4$, $T = 2$, $\nu = 2$.
- Peterson's method is selected.
- $T = \nu$ so that all the errors can be corrected.
- The input and output are shown in Table 3.2.

Example 3.3

- Given: $m = 4$, $T = 2$, $\nu = 2$.
- Berlekamp's method is selected.
- $T = \nu$ so that all the errors can be corrected.
- The input and output are shown in Table 3.3.

Example 3.4

```
Script started on Sep  4 11:25:25 1993
/nfs/pico/grad3/zyoung/simulator

Please select the decoding method.
Select  P for Peterson's or B for Berlekamp's --- P
Peterson's method has been selected.
Please select m : 3
Please select T : 1
G(0)=0  G(1)=6  G(2)=0
Input error locations and values (decimal):
1 1 0 0
The decoded word is a codeword.
The output of the Peterson's Method
The correcting capability, T = 1
Syndrome[3] = 3 (Hex)
Syndrome[4] = 4 (Hex)
The 1x1 determinant is not zero !
Sigma[1] = 1 (decimal)
error position 1 = 1  value  = 1 (Hex)
The decoded word is a codeword,
since it is divisible by the G(X).

Index  Code  Err   Rcvd  Decoded
       Word  Ptn   Word  Word
  6     0     0     0     0
  5     0     0     0     0
  4     0     0     0     0
  3     1     0     1     1
  2     0     0     0     0
  1     6     1     7     6
  0     5     0     5     5
/nfs/pico/grad3/zyoung
script done on Sep  4 11:26:03 1993
```

Table 3.1: Input and output for Example 3.1

```
Script started on Tue Oct  5 11:30:36 1993
Please select the decoding method.
Select  P for Peterson's or B for Berlekamp's --- P
Peterson's method has been selected.
Please select m : 4
Please select T : 2
G(0)=0 G(1)=3 G(2)=1 G(3)=3 G(4)=0
Input error locations and values (decimal):
2 3 10 7 0 0
The output of the Peterson's Method
The correcting capability, T = 2
S[6] = 8 (Hex) S[7] = B (Hex) S[8] = A (Hex) S[9] = 6 (Hex)
The 2x2 determinant is not zero !
Sigma[1] = 4   Sigma[2] = C
error position 1 = 2 (decimal),  value  = 3 (Hex)
error position 2 = 10(decimal),  value  = 7 (Hex)
The decoded word is a codeword, since it is divisible by the G(X).
```

Index	Code Word	Err Ptn	Rcvd Word	Decoded Word
14	0	0	0	0
13	0	0	0	0
12	0	0	0	0
11	0	0	0	0
10	0	7	7	0
9	0	0	0	0
8	0	0	0	0
7	0	0	0	0
6	0	0	0	0
5	0	0	0	0
4	0	0	0	0
3	0	0	0	0
2	0	3	3	0
1	0	0	0	0
0	0	0	0	0

```
script done on Tue Oct  5 11:31:25 1993
```

Table 3.2: Example 3.2

```
Script started on Tue Oct  5 11:12:09 1993
/nfs/pico/grad3/zyoung% ./simulator
Please select the decoding method.
Select  P for Peterson's or B for Berlekamp's --- B
Berlekamp's method has been selected.
Please select m : 4
Please select T : 2
G(0)=0 G(1)=3 G(2)=1 G(3)=3 G(4)=0
Input error pattern (decimal):
2 3 10 7 0 0
The output of the Berlekamp's method :
The correcting capability, T = 2
sigma[1]=8   sigma[2]=C
error position 1 = 10  value  = 7 (Hex)
error position 2 = 2   value  = 3 (Hex)
The decoded word IS a codeword, since it can be evenly divided by the G(X).
```

Index	Code Word	Err Ptn	Rcvd Word	Decoded Word
14	0	0	0	0
13	0	0	0	0
12	0	0	0	0
11	0	0	0	0
10	0	7	7	0
9	0	0	0	0
8	0	0	0	0
7	0	0	0	0
6	0	0	0	0
5	0	0	0	0
4	0	0	0	0
3	0	0	0	0
2	0	3	3	0
1	0	0	0	0
0	0	0	0	0

```
script done on Tue Oct  5 11:12:36 1993
```

Table 3.3: Example 3.3

- Given: $m = 4$, $T = 3$, $\nu = 1$.
- Peterson's method is selected.
- $T > \nu$ so that all the errors can be corrected.
- The input and output are shown in Table 3.4.

Example 3.5

- Given: $m = 4$, $T = 3$, $\nu = 1$.
- Berlekamp's method is selected.
- $T > \nu$ so that all the errors can be corrected.
- The input and output are shown in Table 3.5.

Example 3.6

- Given: $m = 4$, $T = 2$, $\nu = 4$.
- Peterson's method is selected.
- $T < \nu$ so that all the errors can not be corrected.
- The input and output are shown in Table 3.6. It should be noticed that the wrong decoded word can be evenly divided by $G(x)$ even though it is not the original codeword. The error pattern made the received word be another code word.

```
Script started on Tue Oct  5 11:05:15 1993
Select  P for Peterson's or B for Berlekamp's --- P
Peterson's method has been selected.
Please select m :4
Please select T : 3
G(0)=0 G(1)=14 G(2)=7 G(3)=1 G(4)=7 G(5)=14 G(6)=0
Input error locations and values (decimal):
5 5 0 0
The output of the Peterson's Method: correcting capability, T = 3
S[5] = 3 S[6] = 8 S[7] = D S[8] = 3 S[9] = 8 S[10] = D
The 3x3 determinant is zero !
The 2x2 determinant is zero !
The 1x1 determinant is not zero !
Sigma[1] = 5
error position 1 = 5  value  = 5 (Hex)
The decoded word is a codeword, since it is divisible by the G(X).
```

Index	Code Word	Err Ptn	Rcvd Word	Decoded Word
14	0	0	0	0
13	0	0	0	0
12	0	0	0	0
11	0	0	0	0
10	0	0	0	0
9	0	0	0	0
8	0	0	0	0
7	0	0	0	0
6	0	0	0	0
5	0	5	5	0
4	0	0	0	0
3	0	0	0	0
2	0	0	0	0
1	0	0	0	0
0	0	0	0	0

```
script done on Tue Oct  5 11:06:34 1993
```

Table 3.4: Example 3.4

```
Script started on Tue Oct  5 11:08:09 1993
/nfs/pico/grad3/zyoung % ./simulator
Please select the decoding method.
Select  P for Peterson's or B for Berlekamp's --- B
Berlekamp's method has been selected.
Please select m : 4
Please select T : 3
G(0)=0 G(1)=14 G(2)=7 G(3)=1 G(4)=7 G(5)=14 G(6)=0
Input error pattern (decimal):
5 5 0 0
The output of the Berlekamp's method :
The correcting capability, T = 3
sigma[1]=5
error position 1 = 5  value  = 5 (Hex)
The decoded word IS a codeword, since it can be evenly divided by the G(X).
Index  Code  Err   Rcvd  Decoded
       Word  Ptn   Word  Word
 14     0     0     0     0
 13     0     0     0     0
 12     0     0     0     0
 11     0     0     0     0
 10     0     0     0     0
  9     0     0     0     0
  8     0     0     0     0
  7     0     0     0     0
  6     0     0     0     0
  5     0     5     5     0
  4     0     0     0     0
  3     0     0     0     0
  2     0     0     0     0
  1     0     0     0     0
  0     0     0     0     0
script done on Tue Oct  5 11:08:56 1993
```

Table 3.5: Example 3.5

```
Script started on Tue Oct  5 11:13:12 1993
/nfs/pico/grad3/zyoung % ./simulator
Select  P for Peterson's or B for Berlekamp's --- P
Peterson's method has been selected.
Please select m :4
Please select T : 2
G(0)=0 G(1)=3 G(2)=1 G(3)=3 G(4)=0
Input error locations and values (decimal):
3 3 4 7 5 7 6 7 0 0
The output of the Peterson's Method
The correcting capability, T = 2
S[6] = D (Hex) S[7] = 3(Hex)  S[8] = E(Hex)  S[9] = 2 (Hex)
The 2x2 determinant is not zero !
Sigma[1] = 4 (Hex) Sigma[2] = 3 (Hex)
error position 1 = 6  value  = 4 (Hex)
error position 2 = 12  value  = C (Hex)
The decoded word is a codeword, since it is divisible by the G(X).
```

Index	Code Word	Err Ptn	Rcvd Word	Decoded Word
14	0	0	0	0
13	0	0	0	0
12	0	0	0	C
11	0	0	0	0
10	0	0	0	0
9	0	0	0	0
8	0	0	0	0
7	0	0	0	0
6	0	7	7	3
5	0	7	7	7
4	0	7	7	7
3	0	3	3	3
2	0	0	0	0
1	0	0	0	0
0	0	0	0	0

```
script done on Tue Oct  5 11:14:19 1993
```

Table 3.6: Example 3.6

Example 3.7

- Given: $m = 4$, $T = 2$, $\nu = 4$.
- Berlekamp's method is selected.
- $T < \nu$ so that all the errors can not be corrected.
- The input and output are shown in Table 3.7. It should be noticed that the wrong decoded word can be evenly divided by $G(x)$ even though it is not the original codeword. The error pattern made the received word be another code word.

To show the capability of the simulator, the examples of $T = 16$ are given. However, the output is quite long so that they are given in the plain format.

Example 3.8

- Given: $m = 7$, $T = 16$, $\nu = 15$.
- Peterson's method is selected.
- $T > \nu$ so that all the errors can be corrected.
- The input and output are shown as follows.

```
Script started on Tue Oct  5 11:27:41 1993
/nfs/pico/grad3/zyoung % ./simulator
Please select the decoding method.
Select  P for Peterson's or B for Berlekamp's --- B
Berlekamp's method has been selected.
Please select m : 4
Please select T : 2
G(0)=0 G(1)=3 G(2)=1 G(3)=3 G(4)=0
Input error pattern (decimal):
3 3 4 7 5 7 6 7 0 0
The output of the Berlekamp's method :
The correcting capability, T = 2
sigma[1]=14   sigma[2]=D
error position 1 = 12  value  = C (Hex)
error position 2 = 6   value  = 4 (Hex)
The decoded word IS a codeword, since it can be evenly divided by the G(X).
Index  Code  Err   Rcvd  Decoded
       Word  Ptn   Word  Word
 14     0     0     0     0
 13     0     0     0     0
 12     0     0     0     C
 11     0     0     0     0
 10     0     0     0     0
  9     0     0     0     0
  8     0     0     0     0
  7     0     0     0     0
  6     0     7     7     3
  5     0     7     7     7
  4     0     7     7     7
  3     0     3     3     3
  2     0     0     0     0
  1     0     0     0     0
  0     0     0     0     0
script done on Tue Oct  5 11:28:46 1993
```

Table 3.7: Example 3.7

```
Please select the decoding method.
Select  P for Peterson's or B for Berlekamp's --- P
Peterson's method has been selected.
Please select m : 7
Please select T : 16
Input error locations and values (decimal):
21 1 32 2 43 3 54 4 55 5 66 6 77 7 78 8 79 9
60 10 41 11 12 12 83 13 94 14 15 15 0 0
The output of the Peterson's Method
The correcting capability, T = 16
Syndrome[48] = 95
Syndrome[49] = 13
Syndrome[50] = 124
Syndrome[51] = 49
Syndrome[52] = 15
Syndrome[53] = 22
Syndrome[54] = 4
Syndrome[55] = 35
Syndrome[56] = 104
Syndrome[57] = 39
Syndrome[58] = 56
Syndrome[59] = 47
```

```
Syndrome[60] = 4

Syndrome[61] = 12

Syndrome[62] = 32

Syndrome[63] = 15

Syndrome[64] = 109

Syndrome[65] = 36

Syndrome[66] = 18

Syndrome[67] = 50

Syndrome[68] = 51

Syndrome[69] = 9

Syndrome[70] = 125

Syndrome[71] = 63

Syndrome[72] = 54

Syndrome[73] = 44

Syndrome[74] = 24

Syndrome[75] = 82

Syndrome[76] = 11

Syndrome[77] = 48

Syndrome[78] = 110

Syndrome[79] = 90


The 16x16 determinant is zero !
```

```
Syndrome[49] = 13
Syndrome[50] = 124
Syndrome[51] = 49
Syndrome[52] = 15
Syndrome[53] = 22
Syndrome[54] = 4
Syndrome[55] = 35
Syndrome[56] = 104
Syndrome[57] = 39
Syndrome[58] = 56
Syndrome[59] = 47
Syndrome[60] = 4
Syndrome[61] = 12
Syndrome[62] = 32
Syndrome[63] = 15
Syndrome[64] = 109
Syndrome[65] = 36
Syndrome[66] = 18
Syndrome[67] = 50
Syndrome[68] = 51
Syndrome[69] = 9
```

```
Syndrome[70] = 125

Syndrome[71] = 63

Syndrome[72] = 54

Syndrome[73] = 44

Syndrome[74] = 24

Syndrome[75] = 82

Syndrome[76] = 11

Syndrome[77] = 48

Syndrome[78] = 110


The 15x15 determinant is not zero !


Sigma[1].index = 105

Sigma[2].index = 112

Sigma[3].index = 94

Sigma[4].index = 124

Sigma[5].index = 55

Sigma[6].index = 39

Sigma[7].index = 51

Sigma[8].index = 70

Sigma[9].index = 74

Sigma[10].index = 32
```

```
Sigma[11].index = 6
Sigma[12].index = 119
Sigma[13].index = 26
Sigma[14].index = 54
Sigma[15].index = 48

error position 1 = 12  value  = C (Hex)
error position 2 = 15  value  = F (Hex)
error position 3 = 21  value  = 1 (Hex)
error position 4 = 32  value  = 2 (Hex)
error position 5 = 41  value  = B (Hex)
error position 6 = 43  value  = 3 (Hex)
error position 7 = 54  value  = 4 (Hex)
error position 8 = 55  value  = 5 (Hex)
error position 9 = 60  value  = A (Hex)
error position 10 = 66  value  = 6 (Hex)
error position 11 = 77  value  = 7 (Hex)
error position 12 = 78  value  = 8 (Hex)
error position 13 = 79  value  = 9 (Hex)
error position 14 = 83  value  = D (Hex)
error position 15 = 94  value  = E (Hex)
```

The decoded word is a codeword, since it is divisible by the G(X).

Index	Code Word	Err Ptn	Rcvd Word	Decoded Word	Index	Code Word	Err Ptn	Rcvd Word	Decoded Word
126	0	0	0	0	63	0	0	0	0
125	0	0	0	0	62	0	0	0	0
124	0	0	0	0	61	0	0	0	0
123	0	0	0	0	60	0	A	A	0
122	0	0	0	0	59	0	0	0	0
121	0	0	0	0	58	0	0	0	0
120	0	0	0	0	57	0	0	0	0
119	0	0	0	0	56	0	0	0	0
118	0	0	0	0	55	0	5	5	0
117	0	0	0	0	54	0	4	4	0
116	0	0	0	0	53	0	0	0	0
115	0	0	0	0	52	0	0	0	0
114	0	0	0	0	51	0	0	0	0
113	0	0	0	0	50	0	0	0	0
112	0	0	0	0	49	0	0	0	0
111	0	0	0	0	48	0	0	0	0
110	0	0	0	0	47	0	0	0	0

109	0	0	0	0	46	0	0	0	0
108	0	0	0	0	45	0	0	0	0
107	0	0	0	0	44	0	0	0	0
106	0	0	0	0	43	0	3	3	0
105	0	0	0	0	42	0	0	0	0
104	0	0	0	0	41	0	B	B	0
103	0	0	0	0	40	0	0	0	0
102	0	0	0	0	39	0	0	0	0
101	0	0	0	0	38	0	0	0	0
100	0	0	0	0	37	0	0	0	0
99	0	0	0	0	36	0	0	0	0
98	0	0	0	0	35	0	0	0	0
97	0	0	0	0	34	0	0	0	0
96	0	0	0	0	33	0	0	0	0
95	0	0	0	0	32	0	2	2	0
94	0	E	E	0	31	0	0	0	0
93	0	0	0	0	30	0	0	0	0
92	0	0	0	0	29	0	0	0	0
91	0	0	0	0	28	0	0	0	0
90	0	0	0	0	27	0	0	0	0
89	0	0	0	0	26	0	0	0	0
88	0	0	0	0	25	0	0	0	0

87	0	0	0	0	24	0	0	0	0
86	0	0	0	0	23	0	0	0	0
85	0	0	0	0	22	0	0	0	0
84	0	0	0	0	21	0	1	1	0
83	0	D	D	0	20	0	0	0	0
82	0	0	0	0	19	0	0	0	0
81	0	0	0	0	18	0	0	0	0
80	0	0	0	0	17	0	0	0	0
79	0	9	9	0	16	0	0	0	0
78	0	8	8	0	15	0	F	F	0
77	0	7	7	0	14	0	0	0	0
76	0	0	0	0	13	0	0	0	0
75	0	0	0	0	12	0	C	C	0
74	0	0	0	0	11	0	0	0	0
73	0	0	0	0	10	0	0	0	0
72	0	0	0	0	9	0	0	0	0
71	0	0	0	0	8	0	0	0	0
70	0	0	0	0	7	0	0	0	0
69	0	0	0	0	6	0	0	0	0
68	0	0	0	0	5	0	0	0	0
67	0	0	0	0	4	0	0	0	0
66	0	6	6	0	3	0	0	0	0

65	0	0	0	0		2	0	0	0	0
64	0	0	0	0		1	0	0	0	0
63	0	0	0	0		0	0	0	0	0

Example 3.9

- Given: $m = 7$, $T = 16$, $\nu = 16$.
- Peterson's method is selected.
- $T = \nu$ so that all the errors can be corrected.
- The input and output are shown as follows.

```
Please select the decoding method.
Select  P for Peterson's or B for Berlekamp's --- P
Peterson's method has been selected.
Please select m : 7
Please select T : 16
Input error locations and values (decimal):
21 1 32 2 43 3 54 4 55 5 66 6 77 7 78 8 79 9
60 10 41 11 12 12 83 13 94 14 15 15 16 16 0 0
The output of the Peterson's Method
The correcting capability, T = 16
Syndrome[48] = 48
Syndrome[49] = 104
Syndrome[50] = 47
Syndrome[51] = 93
```

```
Syndrome[52] = 96

Syndrome[53] = 44

Syndrome[54] = 65

Syndrome[55] = 16

Syndrome[56] = 115

Syndrome[57] = 55

Syndrome[58] = 7

Syndrome[59] = 75

Syndrome[60] = 99

Syndrome[61] = 76

Syndrome[62] = 90

Syndrome[63] = 102

Syndrome[64] = 108

Syndrome[65] = 22

Syndrome[66] = 73

Syndrome[67] = 87

Syndrome[68] = 10

Syndrome[69] = 101

Syndrome[70] = 50

Syndrome[71] = 38

Syndrome[72] = 79

Syndrome[73] = 92
```

```
Syndrome[74] = 64

Syndrome[75] = 101

Syndrome[76] = 52

Syndrome[77] = 98

Syndrome[78] = 13

Syndrome[79] = 81


The 16x16 determinant is not zero !


Sigma[1].index = 63

Sigma[2].index = 29

Sigma[3].index = 105

Sigma[4].index = 116

Sigma[5].index = 93

Sigma[6].index = 15

Sigma[7].index = 48

Sigma[8].index = 74

Sigma[9].index = 102

Sigma[10].index = 107

Sigma[11].index = 86

Sigma[12].index = 118

Sigma[13].index = 96
```

```
Sigma[14].index = 70

Sigma[15].index = 116

Sigma[16].index = 64


error position 1 = 12  value  = C (Hex)

error position 2 = 15  value  = F (Hex)

error position 3 = 16  value  = 10 (Hex)

error position 4 = 21  value  = 1 (Hex)

error position 5 = 32  value  = 2 (Hex)

error position 6 = 41  value  = B (Hex)

error position 7 = 43  value  = 3 (Hex)

error position 8 = 54  value  = 4 (Hex)

error position 9 = 55  value  = 5 (Hex)

error position 10 = 60  value  = A (Hex)

error position 11 = 66  value  = 6 (Hex)

error position 12 = 77  value  = 7 (Hex)

error position 13 = 78  value  = 8 (Hex)

error position 14 = 79  value  = 9 (Hex)

error position 15 = 83  value  = D (Hex)

error position 16 = 94  value  = E (Hex)


The decoded word is a codeword, since it is divisible by the G(X).
```

Index	Code Word	Err Ptn	Rcvd Word	Decoded Word	Index	Code Word	Err Ptn	Rcvd Word	Decoded Word
126	0	0	0	0	63	0	0	0	0
125	0	0	0	0	62	0	0	0	0
124	0	0	0	0	61	0	0	0	0
123	0	0	0	0	60	0	A	A	0
122	0	0	0	0	59	0	0	0	0
121	0	0	0	0	58	0	0	0	0
120	0	0	0	0	57	0	0	0	0
119	0	0	0	0	56	0	0	0	0
118	0	0	0	0	55	0	5	5	0
117	0	0	0	0	54	0	4	4	0
116	0	0	0	0	53	0	0	0	0
115	0	0	0	0	52	0	0	0	0
114	0	0	0	0	51	0	0	0	0
113	0	0	0	0	50	0	0	0	0
112	0	0	0	0	49	0	0	0	0
111	0	0	0	0	48	0	0	0	0
110	0	0	0	0	47	0	0	0	0
109	0	0	0	0	46	0	0	0	0

108	0	0	0	0	45	0	0	0	0
107	0	0	0	0	44	0	0	0	0
106	0	0	0	0	43	0	3	3	0
105	0	0	0	0	42	0	0	0	0
104	0	0	0	0	41	0	B	B	0
103	0	0	0	0	40	0	0	0	0
102	0	0	0	0	39	0	0	0	0
101	0	0	0	0	38	0	0	0	0
100	0	0	0	0	37	0	0	0	0
99	0	0	0	0	36	0	0	0	0
98	0	0	0	0	35	0	0	0	0
97	0	0	0	0	34	0	0	0	0
96	0	0	0	0	33	0	0	0	0
95	0	0	0	0	32	0	2	2	0
94	0	E	E	0	31	0	0	0	0
93	0	0	0	0	30	0	0	0	0
92	0	0	0	0	29	0	0	0	0
91	0	0	0	0	28	0	0	0	0
90	0	0	0	0	27	0	0	0	0
89	0	0	0	0	26	0	0	0	0
88	0	0	0	0	25	0	0	0	0
87	0	0	0	0	24	0	0	0	0

86	0	0	0	0	23	0	0	0	0
85	0	0	0	0	22	0	0	0	0
84	0	0	0	0	21	0	1	1	0
83	0	D	D	0	20	0	0	0	0
82	0	0	0	0	19	0	0	0	0
81	0	0	0	0	18	0	0	0	0
80	0	0	0	0	17	0	0	0	0
79	0	9	9	0	16	0	10	10	0
78	0	8	8	0	15	0	F	F	0
77	0	7	7	0	14	0	0	0	0
76	0	0	0	0	13	0	0	0	0
75	0	0	0	0	12	0	C	C	0
74	0	0	0	0	11	0	0	0	0
73	0	0	0	0	10	0	0	0	0
72	0	0	0	0	9	0	0	0	0
71	0	0	0	0	8	0	0	0	0
70	0	0	0	0	7	0	0	0	0
69	0	0	0	0	6	0	0	0	0
68	0	0	0	0	5	0	0	0	0
67	0	0	0	0	4	0	0	0	0
66	0	6	6	0	3	0	0	0	0
65	0	0	0	0	2	0	0	0	0

64	0	0	0	0		1	0	0	0	0
63	0	0	0	0		0	0	0	0	0

Example 3.10

- Given: $m = 7$, $T = 16$, $\nu = 17$.
- Peterson's method is selected.
- $T < \nu$ so that all the errors can not be corrected.
- The input and output are shown as follows.

```
Please select the decoding method.
Select  P for Peterson's or B for Berlekamp's --- P
Peterson's method has been selected.
Please select m : 7
Please select T : 16
Input error locations and values (decimal):
21 1 32 2 43 3 54 4 55 5 66 6 77 7 78 8 79 9
60 10 41 11 12 12 83 13 94 14 15 15 16 16 17 17 0 0
The output of the Peterson's Method
The correcting capability, T = 16
Syndrome[48] = 55
Syndrome[49] = 117
Syndrome[50] = 5
Syndrome[51] = 10
```

```
Syndrome[52] = 85
Syndrome[53] = 0
Syndrome[54] = 83
Syndrome[55] = 6
Syndrome[56] = 86
Syndrome[57] = 123
Syndrome[58] = 115
Syndrome[59] = 124
Syndrome[60] = 73
Syndrome[61] = 93
Syndrome[62] = 61
Syndrome[63] = 97
Syndrome[64] = 126
Syndrome[65] = 101
Syndrome[66] = 72
Syndrome[67] = 44
Syndrome[68] = -1
Syndrome[69] = 47
Syndrome[70] = 58
Syndrome[71] = 27
Syndrome[72] = 109
Syndrome[73] = 99
```

```
Syndrome[74] = 49

Syndrome[75] = 113

Syndrome[76] = 103

Syndrome[77] = 38

Syndrome[78] = 34

Syndrome[79] = 104


The 16x16 determinant is not zero !


Sigma[1].index = 61

Sigma[2].index = 0

Sigma[3].index = 110

Sigma[4].index = 119

Sigma[5].index = 20

Sigma[6].index = 1

Sigma[7].index = 64

Sigma[8].index = 14

Sigma[9].index = 87

Sigma[10].index = 79

Sigma[11].index = 107

Sigma[12].index = 107

Sigma[13].index = 24
```

```
Sigma[14].index = 65
Sigma[15].index = 100
Sigma[16].index = 43

error position 1 = 20  value  = 36 (Hex)
error position 2 = 0  value  = 1 (Hex)
error position 3 = 0  value  = 1 (Hex)
error position 4 = 0  value  = 1 (Hex)
error position 5 = 0  value  = 1 (Hex)
error position 6 = 0  value  = 1 (Hex)
error position 7 = 0  value  = 1 (Hex)
error position 8 = 0  value  = 1 (Hex)
error position 9 = 0  value  = 1 (Hex)
error position 10 = 0  value  = 1 (Hex)
error position 11 = 0  value  = 1 (Hex)
error position 12 = 0  value  = 1 (Hex)
error position 13 = 0  value  = 1 (Hex)
error position 14 = 0  value  = 1 (Hex)
error position 15 = 0  value  = 1 (Hex)
error position 16 = 0  value  = 1 (Hex)

The decoded word is NOT a codeword,
```

because it cannot be evenly divided by the G(X).

Index	Code Word	Err Ptn	Rcvd Word	Decoded Word	Index	Code Word	Err Ptn	Rcvd Word	Decoded Word
126	0	0	0	0	63	0	0	0	0
125	0	0	0	0	62	0	0	0	0
124	0	0	0	0	61	0	0	0	0
123	0	0	0	0	60	0	A	A	A
122	0	0	0	0	59	0	0	0	0
121	0	0	0	0	58	0	0	0	0
120	0	0	0	0	57	0	0	0	0
119	0	0	0	0	56	0	0	0	0
118	0	0	0	0	55	0	5	5	5
117	0	0	0	0	54	0	4	4	4
116	0	0	0	0	53	0	0	0	0
115	0	0	0	0	52	0	0	0	0
114	0	0	0	0	51	0	0	0	0
113	0	0	0	0	50	0	0	0	0
112	0	0	0	0	49	0	0	0	0
111	0	0	0	0	48	0	0	0	0
110	0	0	0	0	47	0	0	0	0

109	0	0	0	0	46	0	0	0	0
108	0	0	0	0	45	0	0	0	0
107	0	0	0	0	44	0	0	0	0
106	0	0	0	0	43	0	3	3	3
105	0	0	0	0	42	0	0	0	0
104	0	0	0	0	41	0	B	B	B
103	0	0	0	0	40	0	0	0	0
102	0	0	0	0	39	0	0	0	0
101	0	0	0	0	38	0	0	0	0
100	0	0	0	0	37	0	0	0	0
99	0	0	0	0	36	0	0	0	0
98	0	0	0	0	35	0	0	0	0
97	0	0	0	0	34	0	0	0	0
96	0	0	0	0	33	0	0	0	0
95	0	0	0	0	32	0	2	2	2
94	0	E	E	E	31	0	0	0	0
93	0	0	0	0	30	0	0	0	0
92	0	0	0	0	29	0	0	0	0
91	0	0	0	0	28	0	0	0	0
90	0	0	0	0	27	0	0	0	0
89	0	0	0	0	26	0	0	0	0
88	0	0	0	0	25	0	0	0	0

87	0	0	0	0	24	0	0	0	0
86	0	0	0	0	23	0	0	0	0
85	0	0	0	0	22	0	0	0	0
84	0	0	0	0	21	0	1	1	1
83	0	D	D	D	20	0	0	0	36
82	0	0	0	0	19	0	0	0	0
81	0	0	0	0	18	0	0	0	0
80	0	0	0	0	17	0	11	11	11
79	0	9	9	9	16	0	10	10	10
78	0	8	8	8	15	0	F	F	F
77	0	7	7	7	14	0	0	0	0
76	0	0	0	0	13	0	0	0	0
75	0	0	0	0	12	0	C	C	C
74	0	0	0	0	11	0	0	0	0
73	0	0	0	0	10	0	0	0	0
72	0	0	0	0	9	0	0	0	0
71	0	0	0	0	8	0	0	0	0
70	0	0	0	0	7	0	0	0	0
69	0	0	0	0	6	0	0	0	0
68	0	0	0	0	5	0	0	0	0
67	0	0	0	0	4	0	0	0	0
66	0	6	6	6	3	0	0	0	0

65	0	0	0	0		2	0	0	0	0
64	0	0	0	0		1	0	0	0	0
63	0	0	0	0		0	0	0	0	1

Example 3.11

- Given: $m = 7$, $T = 16$, $\nu = 16$.
- Berlekamp's method is selected.
- $T = \nu$ so that all the errors can be corrected.
- The input and output are shown as follows.

```
Please select the decoding method.
Select  P for Peterson's or B for Berlekamp's --- B
Berlekamp's method has been selected.
Please select m : 7
Please select T : 16
Input error locations and values (decimal):
1 2 5 6 8 9 11 13 15 24 18 36 23 1 25 4 35 9 39 2
41 4 45 8 50 7 57 9 60 7 66 5 0 0 0 0
The output of the Berlekamp's method :
The correcting capability, T = 16
sigma(1)=14 (hex)
sigma(2)=9 (hex)
sigma(3)=6A (hex)
sigma(4)=46 (hex)
```

```
sigma(5)=76 (hex)
sigma(6)=47 (hex)
sigma(7)=4F (hex)
sigma(8)=73 (hex)
sigma(9)=4B (hex)
sigma(10)=66 (hex)
sigma(11)=63 (hex)
sigma(12)=7B (hex)
sigma(13)=76 (hex)
sigma(14)=71 (hex)
sigma(15)=48 (hex)
sigma(16)=76 (hex)
error position 1 = 66  value  = 0x5
error position 2 = 60  value  = 0x7
error position 3 = 57  value  = 0x9
error position 4 = 50  value  = 0x7
error position 5 = 45  value  = 0x8
error position 6 = 41  value  = 0x4
error position 7 = 39  value  = 0x2
error position 8 = 35  value  = 0x9
error position 9 = 25  value  = 0x4
error position 10 = 23  value  = 0x1
```

```
error position 11 = 18  value  = 0x24
error position 12 = 15  value  = 0x18
error position 13 = 11  value  = 0xD
error position 14 = 8  value  = 0x9
error position 15 = 5  value  = 0x6
error position 16 = 1  value  = 0x2

The decoded word IS a codeword, since it can be evenly divided by the G(X).
```

Index	Code Word	Err Ptn	Rcvd Word	Decoded Word	Index	Code Word	Err Ptn	Rcvd Word	Decoded Word
126	0	0	0	0	63	0	0	0	0
125	0	0	0	0	62	0	0	0	0
124	0	0	0	0	61	0	0	0	0
123	0	0	0	0	60	0	7	7	0
122	0	0	0	0	59	0	0	0	0
121	0	0	0	0	58	0	0	0	0
120	0	0	0	0	57	0	9	9	0
119	0	0	0	0	56	0	0	0	0
118	0	0	0	0	55	0	0	0	0
117	0	0	0	0	54	0	0	0	0

116	0	0	0	0	53	0	0	0	0
115	0	0	0	0	52	0	0	0	0
114	0	0	0	0	51	0	0	0	0
113	0	0	0	0	50	0	7	7	0
112	0	0	0	0	49	0	0	0	0
111	0	0	0	0	48	0	0	0	0
110	0	0	0	0	47	0	0	0	0
109	0	0	0	0	46	0	0	0	0
108	0	0	0	0	45	0	8	8	0
107	0	0	0	0	44	0	0	0	0
106	0	0	0	0	43	0	0	0	0
105	0	0	0	0	42	0	0	0	0
104	0	0	0	0	41	0	4	4	0
103	0	0	0	0	40	0	0	0	0
102	0	0	0	0	39	0	2	2	0
101	0	0	0	0	38	0	0	0	0
100	0	0	0	0	37	0	0	0	0
99	0	0	0	0	36	0	0	0	0
98	0	0	0	0	35	0	9	9	0
97	0	0	0	0	34	0	0	0	0
96	0	0	0	0	33	0	0	0	0
95	0	0	0	0	32	0	0	0	0

94	0	0	0	0	31	0	0	0	0
93	0	0	0	0	30	0	0	0	0
92	0	0	0	0	29	0	0	0	0
91	0	0	0	0	28	0	0	0	0
90	0	0	0	0	27	0	0	0	0
89	0	0	0	0	26	0	0	0	0
88	0	0	0	0	25	0	4	4	0
87	0	0	0	0	24	0	0	0	0
86	0	0	0	0	23	0	1	1	0
85	0	0	0	0	22	0	0	0	0
84	0	0	0	0	21	0	0	0	0
83	0	0	0	0	20	0	0	0	0
82	0	0	0	0	19	0	0	0	0
81	0	0	0	0	18	0	24	24	0
80	0	0	0	0	17	0	0	0	0
79	0	0	0	0	16	0	0	0	0
78	0	0	0	0	15	0	18	18	0
77	0	0	0	0	14	0	0	0	0
76	0	0	0	0	13	0	0	0	0
75	0	0	0	0	12	0	0	0	0
74	0	0	0	0	11	0	D	D	0
73	0	0	0	0	10	0	0	0	0

72	0	0	0	0
71	0	0	0	0
70	0	0	0	0
69	0	0	0	0
68	0	0	0	0
67	0	0	0	0
66	0	5	5	0
65	0	0	0	0
64	0	0	0	0
63	0	0	0	0

9	0	0	0	0
8	0	9	9	0
7	0	0	0	0
6	0	0	0	0
5	0	6	6	0
4	0	0	0	0
3	0	0	0	0
2	0	0	0	0
1	0	2	2	0
0	0	0	0	0

Chapter 4

The Periodicity Algorithm

Chien search is the main obstacle to high speed RS decoding. Based on the exhaustive Chien search, the error position numbers of the double error correcting RS codes are carefully examined. It is found that for the case of double error correcting, the roots of the error locators are not randomly distributed. This gives us the possibility to develop a new algorithm to locate the error position numbers without Chien search. In this chapter, a new algorithm for double error correcting called the **Periodicity Algorithm** (PA) is proposed.

4.1 Basic properties

On examining the exhaustive Chien search, it can be shown that the error positions of a double error correcting RS code are not distributed randomly, but rather according to certain patterns. To show this important property clearly, the case of $(m = 3)$ 2-error correcting RS code is used as an example.

$\downarrow \sigma_1^{'}$ $\sigma_2^{'} \rightarrow$	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^0	x,x	1,5	2,3	x,x	4,6	x,x	x,x
α^1	x,x	x,x	x,x	0,4	1,2	x,x	3,5
α^2	x,x	2,4	x,x	x,x	x,x	3,6	0,1
α^3	2,5	0,6	x,x	1,3	x,x	x,x	x,x
α^4	x,x	x,x	1,4	5,6	x,x	0,2	x,x
α^5	1,6	x,x	x,x	x,x	0,3	4,5	x,x
α^6	3,4	x,x	0,5	x,x	x,x	x,x	2,6

Table 4.1: Periodicity of the error positions ($m = 3$, $T = 2$), (see text for the description of elements)

From the equation

$$\Sigma(x) = x^2 + \sigma_1 x + \sigma_2 x = 0 \tag{4.1}$$

by dividing all the coefficients by σ_2, Eq. 4.1 becomes:

$$1 + \sigma_1^{'} x + \sigma_2^{'} x^2 = 0. \tag{4.2}$$

After performing the exhaustive Chien search to Eq. 4.2, all of the possible solutions are obtained. The results for $m = 3$ are given in Table 4.1.

The meaning of the elements in the Table 4.1 is as follows. For example, the right-bottom element in Table 4.1 is [2,6]. This means that, for the Eq 4.2, when $\sigma_1^{'} = \alpha^6$ and $\sigma_2^{'} = \alpha^6$, the two roots are α^2 and α^6. This kind of elements are called solution elements. The elements marked as [x,x] means that there are no solutions for the combinations of the $\sigma_1^{'}$ and $\sigma_2^{'}$. These

	α^6	α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^0	x,x	x,x	1,5	2,3	x,x	4,6	x,x	x,x	x,x	1,5	2,3	x,x	4,6	x,x	x,x
α^1	3,5	x,x	x,x	x,x	0,4	1,2	x,x	3,5	x,x	x,x	x,x	0,4	1,2	x,x	3,5
α^2	0,1	x,x	2,4	x,x	x,x	x,x	3,6	0,1	x,x	2,4	x,x	x,x	x,x	3,6	0,1
α^3	x,x	2,5	0,6	x,x	1,3	x,x	x,x	x,x	2,5	0,6	x,x	1,3	x,x	x,x	x,x
α^4	x,x	x,x	x,x	1,4	5,6	x,x	0,2	x,x	x,x	x,x	1,4	5,6	x,x	0,2	x,x
α^5	x,x	1,6	x,x	x,x	x,x	0,3	4,5	x,x	1,6	x,x	x,x	x,x	0,3	4,5	x,x
α^6	2,6	3,4	x,x	0,5	x,x	x,x	x,x	2,6	3,4	x,x	0,5	x,x	x,x	x,x	2,6
α^0	x,x	x,x	1,5	2,3	x,x	4,6	x,x	x,x	x,x	1,5	2,3	x,x	4,6	x,x	x,x
α^1	3,5	x,x	x,x	x,x	0,4	1,2	x,x	3,5	x,x	x,x	x,x	0,4	1,2	x,x	3,5
α^2	0,1	x,x	2,4	x,x	x,x	x,x	3,6	0,1	x,x	2,4	x,x	x,x	x,x	3,6	0,1
α^3	x,x	2,5	0,6	x,x	1,3	x,x	x,x	x,x	2,5	0,6	x,x	1,3	x,x	x,x	x,x
α^4	x,x	x,x	x,x	1,4	5,6	x,x	0,2	x,x	x,x	x,x	1,4	5,6	x,x	0,2	x,x
α^5	x,x	1,6	x,x	x,x	x,x	0,3	4,5	x,x	1,6	x,x	x,x	x,x	0,3	4,5	x,x
α^6	2,6	3,4	x,x	0,5	x,x	x,x	x,x	2,6	3,4	x,x	0,5	x,x	x,x	x,x	2,6

Table 4.2: Table used to show the periodicity

elements are called non-solution elements. To explain the basic properties more clearly, Table 4.1 is re-formulated as Table 4.2.

Table 4.2 essentially consists of many copies of Table 4.1. As $\alpha^7 = \alpha^0$ when $m = 3$, this property can be used to form the Table 4.2. A line is drawn across the table on which there are N (where $N = 7$) solution elements. Each such line is called a *chain*.

Several terms are defined as follows:

1. **Solution element** — the element with solutions in Table 4.1.

P_{leader}	α^0	α^1	α^2	α^3	α^4	α^5	α^6
I_{leader}	x	x	x	2	x	1	3

Table 4.3: Leaders of the chains ($m = 3$)

2. **Non-solution element** — the [x,x] element which means no solution.

3. **Solution chain** — the chain contains only solution elements.

4. **Non-solution chain** — the chain contains only non-solution elements.

5. **Shift on chains** — movement from one element to its neighbor element on the chain. This movement skips one column to the next row on the solution table.

6. **Leaders of the chains** — the smaller values of the elements in first column in Table 4.1.

7. **Leader table** — putting the leaders into a table can form the leader table. Table 4.3 is the leader table for $m = 3$.

Looking at Table 4.2, let us put an imaginary pointer to the right-bottom element which is [2,6]. Then let us shift the pointer 2 columns left and 1 row up which is defined as 1 shift. Now the pointer is moved onto the element which

is [0,3]. If keeping moving the imaginary pointer according to the rule explained above, following chain is obtained

$$[2,6] \leftarrow [1,5] \leftarrow [0,4] \leftarrow [3,6] \leftarrow [2,5] \leftarrow [1,4] \leftarrow [0,3] \leftarrow [2,6]. \quad (4.3)$$

It should be noticed that, after N shifts, the pointer goes back onto the element [2,6], the "starting point". In other words, the chain repeats itself with a period of N shifts !

For the chain shown in Eq. 4.3, the last element is [2,6], then [0,3]. Since $\alpha^7 = \alpha^0$ for $m = 3$, [0,3] can be written as [7,3] or [3,7]. It is obvious that, from [2,6] to [3,7], the increment of both error positions is 1. Similarly, for the cases of that [0,3] to [1,4], [1,4] to [2,5], $\cdots$, the increment is still 1.

In Table 4.1, looking at each column, the sum of two solutions and the power value of σ_2' is 7 or 14. It should be noticed that $N = 7$ and $2N = 14$ when $m = 3$. It has been explained above that the values of each solution element [E_1,E_2] mean α^{E_1} and α^{E_2}. Therefore, this yields

$$\alpha^{E_1}\alpha^{E_2}\sigma_2' = \alpha^{2N} = \alpha^N = \alpha^0 = 1. \quad (4.4)$$

Eq. 4.4 is called *constant product property.* Considering that $\sigma_2' = \alpha^{i_2}$, the constant product property can be simplified as

$$E_1 + E_2 + i_2 = N. \quad (4.5)$$

In summary, there are three basic properties:

1. The error position numbers display the periodicity. The period is N shifts.

2. The increment of the error position numbers is 1 for each shift.

3. There is the constant product property. It can be simply expressed as $E_1 + E_2 + i_2 = N$.

The case of $m = 3$ has been used as the example to describe the three properties of the RS error positions. However, do those three properties hold for the other cases of m as well? The answer to this question is yes, and the verification to this conclusion will be given in section 4.3.

Before going on to the description of the algorithm , the following facts should be noted:

1. The solution chains contain only the solution elements.

2. The non-solution chains contain only the [x,x] non-solution elements.

3. All the elements in the solution table are either on the solution chains or on the non-solution chains without exception.

4.2 Description for periodicity algorithm

4.2.1 Algorithm description

Definitions for the symbols used in this algorithm are as follows:

LT: leader table which consists of the leaders of the chains

$i_{mapping}$: intermediate variable

P_{leader}: position of the leader in LT

I_{leader}: value of the leader

i_1: exponential power of the σ_1' ($\sigma_1' = \alpha^{i_1}$)

i_2: exponential power of the σ_2' ($\sigma_2' = \alpha^{i_2}$)

E_1: position of the first error

E_2: position of the second error

The LT is pre-processed off-line. The algorithm for the pre-processing is given in Fig. 4.1.

The PA error locating consists of following steps:

1. According to the given $\sigma_1 = \alpha^{i_1}$ and $\sigma_2 = \alpha^{i_2}$, find the leader of the chain.

   $i_{mapping} = i_2/2$, if i_2 even (with direct leader);

   $i_{mapping} = (i_2 + N)/2$, if i_2 odd (without direct leader);

   $P_{leader} = (i_1 - i_{mapping})_{mod\ N}$.

   According to P_{leader}, get the value of the leader I_{leader} in LT.

2. If leader of the chain is "x", a non-solution chain is met. It means that all the elements on this chain are non-solution elements. Otherwise, go

on to the step 3.

3. With the value of the leader of the chain, calculate the first error position, E_1.

   $$E_1 = (I_{leader} - i_{mapping})_{mod\ N}.$$

4. With the property of constant product, calculate the second error position.

   $$E_2 = N - (i_2 + E_1).$$

The flow chart of the algorithm is given in Fig. 4.2.

From the flow chart, it is clear that PA error locating needs fewer than 7 additions, 4 decision operations, 1 shift and 1 memory reading operations. Because the shift operation normally needs the same time as addition, it can be simply considered that the periodicity algorithm needs 8 additions and 4 decisions. There are no multiplications and divisions at all. In the flow chart, the operation $i_2/2$ is actually performed by shift-right operation. The testing operation decides whether the value is even or odd by testing the least significant bit (LSB), *i.e.* if the LSB is 0, the value is even; if it is 1, the value is odd. All those operations are simple for both the hardware and software, and can be counted as one addition.

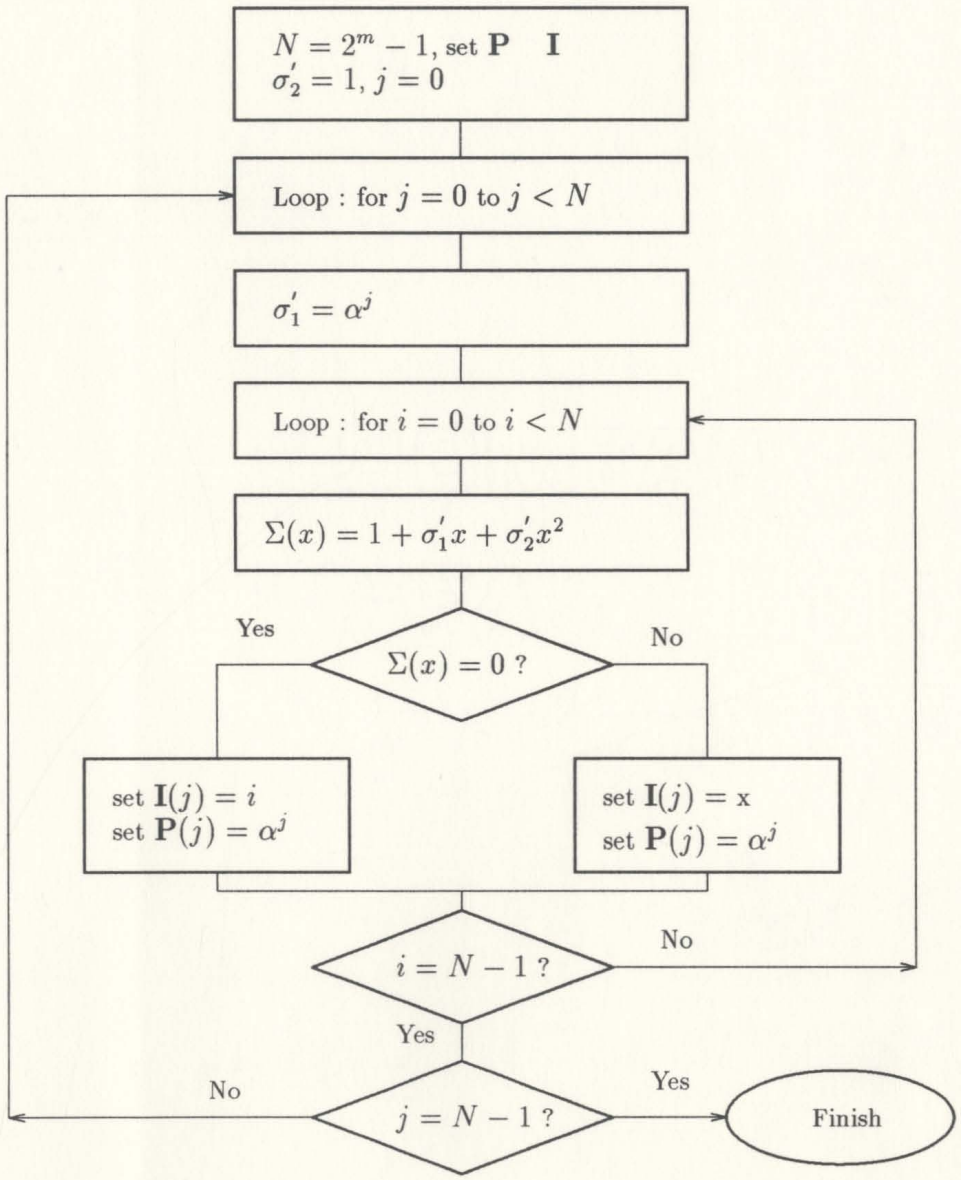

Figure 4.1: Leader table creation

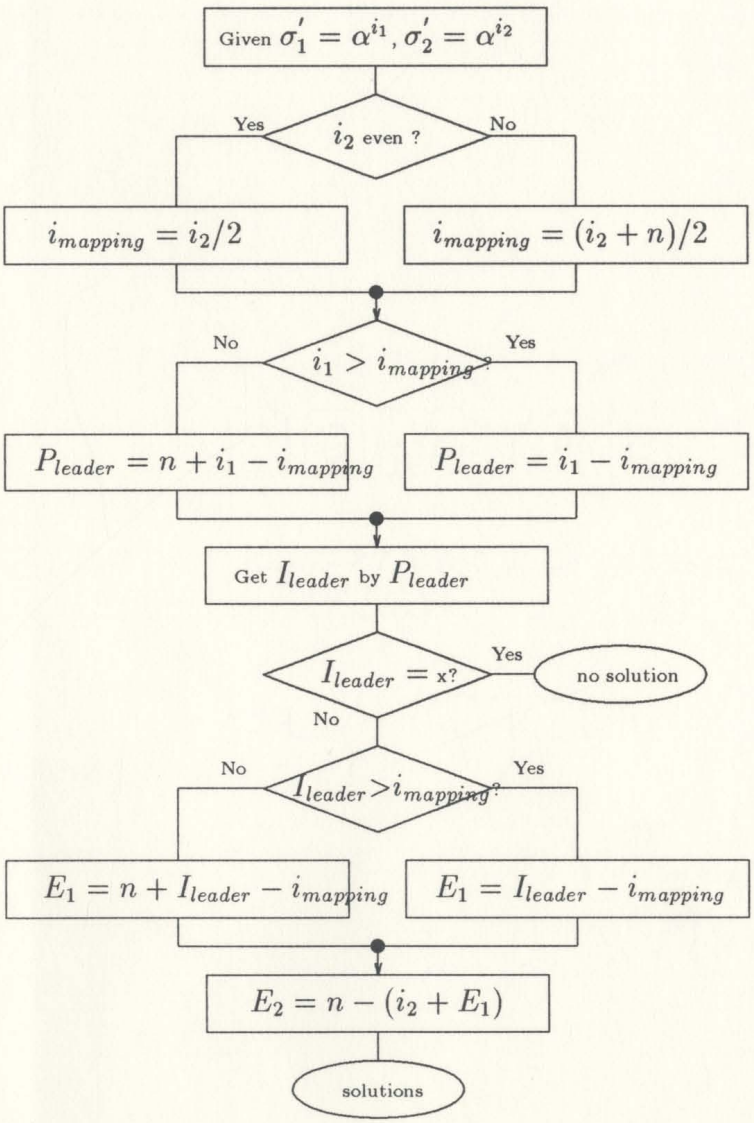

Figure 4.2: Periodicity algorithm

	α^6	α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^0	x,x	x,x	1,5	2,3	x,x	4,6	x,x	x,x	x,x	1,5	2,3	x,x	4,6	x,x	x,x
α^1	3,5	x,x	x,x	x,x	0,4	1,2	x,x	3,5	x,x	x,x	x,x	0,4	1,2	x,x	3,5
α^2	0,1	x,x	2,4	x,x	x,x	x,x	3,6	0,1	x,x	2,4	x,x	x,x	x,x	3,6	0,1
α^3	x,x	2,5	0,6	x,x	1,3	x,x	x,x	x,x	2,5	0,6	x,x	1,3	x,x	x,x	x,x
α^4	x,x	x,x	x,x	1,4	5,6	x,x	0,2	x,x	x,x	x,x	1,4	5,6	x,x	0,2	x,x
α^5	x,x	1,6	x,x	x,x	x,x	0,3	4,5	x,x	1,6	x,x	x,x	x,x	0,3	4,5	x,x
α^6	2,6	3,4	x,x	0,5	x,x	x,x	x,x	2,6	3,4	x,x	0,5	x,x	x,x	x,x	2,6
α^0	x,x	x,x	1,5	2,3	x,x	4,6	x,x	x,x	x,x	1,5	2,3	x,x	4,6	x,x	x,x
α^1	3,5	x,x	x,x	x,x	0,4	1,2	x,x	3,5	x,x	x,x	x,x	0,4	1,2	x,x	3,5
α^2	0,1	x,x	2,4	x,x	x,x	x,x	3,6	0,1	x,x	2,4	x,x	x,x	x,x	3,6	0,1
α^3	x,x	2,5	0,6	x,x	1,3	x,x	x,x	x,x	2,5	0,6	x,x	1,3	x,x	x,x	x,x
α^4	x,x	x,x	x,x	1,4	5,6	x,x	0,2	x,x	x,x	x,x	1,4	5,6	x,x	0,2	x,x
α^5	x,x	1,6	x,x	x,x	x,x	0,3	4,5	x,x	1,6	x,x	x,x	x,x	0,3	4,5	x,x
α^6	2,6	3,4	x,x	0,5	x,x	x,x	x,x	2,6	3,4	x,x	0,5	x,x	x,x	x,x	2,6

Table 4.4: Table for Example 4.1

4.2.2 Examples for PA

Several examples are given below which show how the periodicity algorithm works. The flow chart shown in Fig. 4.2 should be used to assist the understanding.

Example 4.1

- Given $\sigma_1' = \alpha^{i_1} = \alpha^5$, $\sigma_2' = \alpha^{i_2} = \alpha^4$.
- $i_1 = 5$, $i_2 = 4$.

1. Obtain $i_{mapping}$, which is a value to show that the leader of the chain is how many "shift" above current row.

   i_2 is even. It means that the leader of current chain is within current table.

   $i_{mapping} = i_2/2 = 4/2 = 2$ (shifts). The leader is 2 shifts above current row.

2. Go up 2 shifts to find the leader of the current chain.

   $P_{leader} = i_1 - i_{mapping} = 5 - 3 = 2$.

   From the leader table shown in Table 4.3, the value of the leader can be found, $I_{leader} = 2$.

3. Use increment property (for 2 shifts):

   $E_1 = I_{leader} - i_{mapping} = 2 - 2 = 0$.

4. Use constant product property:

   $E_2 = N - E_1 - i_2 = 7 - 0 - 4 = 3$.

   Therefore, $[E_1,E_2] = [0,3]$.

Example 4.2

- Given $\sigma_1' = \alpha^{i_1} = \alpha^4$, $\sigma_2' = \alpha^{i_2} = \alpha^5$.
- $i_1 = 4$, $i_2 = 5$.

	α^6	α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^0	x,x	x,x	1,5	2,3	x,x	4,6	x,x	x,x	x,x	1,5	2,3	x,x	4,6	x,x	x,x
α^1	3,5	x,x	x,x	x,x	0,4	1,2	x,x	3,5	x,x	x,x	x,x	0,4	1,2	x,x	3,5
α^2	0,1	x,x	2,4	x,x	x,x	x,x	3,6	0,1	x,x	2,4	x,x	x,x	x,x	3,6	0,1
α^3	x,x	2,5	0,6	x,x	1,3	x,x	x,x	x,x	2,5	0,6	x,x	1,3	x,x	x,x	x,x
α^4	x,x	x,x	x,x	1,4	5,6	x,x	0,2	x,x	x,x	x,x	1,4	5,6	x,x	0,2	x,x
α^5	x,x	1,6	x,x	x,x	x,x	0,3	4,5	x,x	1,6	x,x	x,x	x,x	0,3	4,5	x,x
α^6	2,6	3,4	x,x	0,5	x,x	x,x	x,x	2,6	3,4	x,x	0,5	x,x	x,x	x,x	2,6
α^0	x,x	x,x	1,5	2,3	x,x	4,6	x,x	x,x	x,x	1,5	2,3	x,x	4,6	x,x	x,x
α^1	3,5	x,x	x,x	x,x	0,4	1,2	x,x	3,5	x,x	x,x	x,x	0,4	1,2	x,x	3,5
α^2	0,1	x,x	2,4	x,x	x,x	x,x	3,6	0,1	x,x	2,4	x,x	x,x	x,x	3,6	0,1
α^3	x,x	2,5	0,6	x,x	1,3	x,x	x,x	x,x	2,5	0,6	x,x	1,3	x,x	x,x	x,x
α^4	x,x	x,x	x,x	1,4	5,6	x,x	0,2	x,x	x,x	x,x	1,4	5,6	x,x	0,2	x,x
α^5	x,x	1,6	x,x	x,x	x,x	0,3	4,5	x,x	1,6	x,x	x,x	x,x	0,3	4,5	x,x
α^6	2,6	3,4	x,x	0,5	x,x	x,x	x,x	2,6	3,4	x,x	0,5	x,x	x,x	x,x	2,6

Table 4.5: Table for Example 4.2

1. Obtain $i_{mapping}$, which is a value to show that the leader of the chain is how many "shift" above current row.

   i_2 is odd. It means that the leader of current chain is outside current table.

   $i_{mapping} = (N + i_2)/2 = (5 + 7)/2 = 6$ (shifts). The leader is 6 shifts above current row.

2. Go up 6 shifts to find the leader of the current chain.

   $P_{leader} = i_1 - i_{mapping} + N = 4 - 6 + 7 = 5.$

   From the leader table shown in Table 4.3, the value of the leader can be found, $I_{leader} = 1$.

3. Use increment property (for 6 shifts):

   $E_1 = I_{leader} - i_{mapping} + N = 1 - 6 + 7 = 2.$

4. Use constant product property:

   $E_2 = N - E_1 - i_2 = 7 - 2 - 5 = 0.$

   Therefore, $[E_1,E_2] = [2,0]$.

From Table 4.1, it is easy to find that the results of Example 4.1 and Example 4.2 are correct.

4.3 Algorithm Verification

In this section, the validity of the periodicity algorithm will be verified. Since the size of the table will increase exponentially as m increases, it is not possible to quote all the relevant tables here so that only the tables of up to $m = 6$ are given. However, as the Galois field fortunately contain only a finite number of elements, it is possible to use exhaustive computer simulation to verify the periodicity algorithm.

Verifying the validity is simple. From LT, the exhaustive test can be performed for all the combinations of σ_1 and σ_2. When the results are not "x", they can be substituted into Eq 4.2 to see whether the result is zero. When results are "x", Chien search can be used for that case to see whether the "x" is correct for that combination. This procedure can be used to verify the validity of PA for different m. This verification algorithm is given in Fig. 4.3.

Another issue to be noted is the limited verification of the periodicity algorithm. That is, the periodicity algorithm is not shown correct analytically, but by the computer exhaustive verification limited by the array size for m such that $3 \leq m \leq 10$. However, this range of m is adequate for most actual applications, *e.g.* when $m = 10$, $N = 2^m = 2^{10} = 1024$, which is longer than the length of a typical data block. The verification outputs for $7 \leq m \leq 10$ are given in Table 4.9.

Table of error positions (m = 4, T = 2)

$$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2$$

$\downarrow \sigma_1 \quad \rightarrow \sigma_2$	α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
α^0	5,10	6,8	1,12	x,x	2,9	11,14	x,x	x,x	3,4	x,x	7,13	x,x	x,x	x,x	x,x
α^1	x,x	x,x	4,9	5,7	0,11	x,x	1,8	10,13	x,x	x,x	2,3	x,x	6,12	x,x	x,x
α^2	x,x	x,x	x,x	x,x	3,8	4,6	10,14	x,x	0,7	9,12	x,x	x,x	1,2	x,x	5,11
α^3	x,x	4,10	x,x	x,x	x,x	x,x	2,7	3,5	9,13	x,x	6,14	8,11	x,x	x,x	0,1
α^4	x,x	0,14	x,x	3,9	x,x	x,x	x,x	x,x	1,6	2,4	8,12	x,x	5,13	7,10	x,x
α^5	6,9	x,x	x,x	13,14	x,x	2,8	x,x	x,x	x,x	x,x	0,5	1,3	7,11	x,x	4,12
α^6	x,x	3,11	5,8	x,x	x,x	12,13	x,x	1,7	x,x	x,x	x,x	x,x	4,14	0,2	6,10
α^7	1,14	5,9	x,x	2,10	4,7	x,x	x,x	11,12	x,x	0,6	x,x	x,x	x,x	x,x	3,13
α^8	x,x	2,12	0,13	4,8	x,x	1,9	3,6	x,x	x,x	10,11	x,x	5,14	x,x	x,x	x,x
α^9	x,x	x,x	x,x	1,11	12,14	3,7	x,x	0,8	2,5	x,x	x,x	9,10	x,x	4,13	x,x
α^{10}	3,12	x,x	x,x	x,x	x,x	0,10	11,13	2,6	x,x	7,14	1,4	x,x	x,x	8,9	x,x
α^{11}	7,8	x,x	2,11	x,x	x,x	x,x	x,x	9,14	10,12	1,5	x,x	6,13	0,3	x,x	x,x
α^{12}	x,x	x,x	6,7	x,x	1,10	x,x	x,x	x,x	x,x	8,13	9,11	0,4	x,x	5,12	2,14
α^{13}	4,11	1,13	x,x	x,x	5,6	x,x	0,9	x,x	x,x	x,x	x,x	7,12	8,10	3,14	x,x
α^{14}	2,13	x,x	3,10	0,12	x,x	x,x	4,5	x,x	8,14	x,x	x,x	x,x	x,x	6,11	7,9

Table 4.6: Solution table for $m = 4$

Table of error positions ($m = 5$, $T = 2$)

$$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2$$

$\downarrow \sigma_1 \quad \sigma_2 \rightarrow$	α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	α^{15}	α^{16}	α^{17}	α^{18}	α^{19}	α^{20}	α^{21}	α^{22}	α^{23}	α^{24}	α^{25}	α^{26}	α^{27}	α^{28}	α^{29}	α^{30}
α^0	x,x	2,28	4,25	x,x	8,19	x,x	x,x	26,29	7,16	x,x	x,x	x,x	x,x	x,x	21,27	6,10	1,14	x,x	x,x	13,30	x,x	x,x	x x	3,5	x x	15,22	x,x	17,18	11,23	9,24	12,20
α^1	8,23	11,19	x,x	1,27	3,24	x,x	7,18	x,x	x,x	25,28	6,15	x,x	x,x	x,x	x,x	x,x	20,26	5,9	0,13	x,x	x,x	12,29	x,x	x x	x,x	2,4	x,x	14,21	x x	16,17	10,22
α^2	15,16	9,21	7,22	10,18	x,x	0,26	2,23	x,x	6,17	x,x	x,x	24,27	5,14	x,x	x,x	x,x	x,x	x,x	19,25	4,8	12,30	x,x	x x	11,28	x x	x x	x x	1,3	x x	13,20	x x
α^3	12,19	x,x	14,15	8,20	6,21	9,17	x,x	25,30	1,22	x,x	5,16	x,x	x,x	23,26	4,13	x,x	x,x	x,x	x,x	x,x	18,24	3,7	11,29	x x	x,x	10,27	x x	x x	x x	0,2	x x
α^4	1,30	x,x	11,18	x,x	13,14	7,19	5,20	8,16	x,x	24,29	0,21	x,x	4,15	x,x	x,x	22,25	3,12	x,x	x,x	x,x	x,x	x,x	17,23	2,6	10,28	x,x	x,x	9,26	x x	x,x	x,x
α^5	x,x	x,x	0,29	x,x	10,17	x,x	12,13	6,18	4,19	7,15	x,x	23,28	20,30	x,x	3,14	x,x	x,x	21,24	2,11	x,x	x,x	x,x	x x	x x	16,22	1,5	9,27	x x	x,x	8,25	x,x
α^6	7,24	x,x	x,x	x,x	28,30	x,x	9,16	x,x	11,12	5,17	3,18	6,14	x,x	22,27	19,29	x,x	2,13	x,x	x,x	20,23	1,10	x,x	x x	x x	x x	x x	15,21	0,4	8,26	x,x	x,x
α^7	x,x	x,x	6,23	x,x	x,x	x,x	27,29	x,x	8,15	x,x	10,11	4,16	2,17	5,13	x,x	21,26	18,28	x,x	1,12	x,x	x,x	19,22	0,9	x x	x,x	x,x	x,x	x,x	14,20	3,10	7,25
α^8	2,29	6,24	x,x	x,x	5,22	x,x	x,x	x,x	26,28	x,x	7,14	x,x	9,10	3,15	1,16	4,12	x,x	20,25	17,27	x,x	0,11	x,x	x x	18,21	8,30	x x	x,x	x,x	x x	x,x	13,19
α^9	x,x	12,18	1,28	5,23	x,x	x,x	4,21	x,x	x,x	x,x	25,27	x,x	6,13	x,x	8,9	2,14	0,15	3,11	x,x	19,24	16,26	x x	10,30	x x	x x	17,20	7,29	x x	x x	x,x	x x
α^{10}	x,x	x,x	x,x	11,17	0,27	4,22	x,x	x,x	3,20	x,x	x,x	x,x	24,26	x,x	5,12	x,x	7,8	1,13	14,30	2,10	x x	18,23	15,25	x x	9,29	x x	x,x	16,19	6,28	x x	x,x
α^{11}	x,x	x,x	x,x	x,x	x,x	10,16	26,30	3,21	x,x	x,x	2,19	x,x	x,x	x,x	23,25	x,x	4,11	x,x	6,7	0,12	13,29	1,9	x x	17,22	14,21	x x	8,28	x,x	x x	15,18	5,27
α^{12}	14,17	4,26	x,x	x,x	x,x	x x	x,x	9,15	25,29	2,20	x,x	x,x	1,18	x,x	x,x	x,x	22,24	x,x	3,10	x,x	5,6	11,30	12,28	0,8	x x	16,21	13,23	x,x	7,27	x x	x x
α^{13}	x,x	x,x	13,16	3,25	x,x	x,x	x,x	x,x	x,x	8,14	24,28	1,19	x,x	x,x	0,17	x,x	x x	x,x	21,23	x,x	2,9	x,x	4,5	10,29	11,27	7,30	x x	15,20	12,22	x x	6,26
α^{14}	x,x	5,25	x,x	x,x	12,15	2,24	x,x	x,x	x,x	x,x	x,x	7,13	23,27	0,18	x x	x x	16,30	x,x	x,x	x x	20,22	x x	1,8	x x	3,4	9,28	10,26	6,29	x x	14,19	11,21
α^{15}	13,18	10,20	x,x	4,24	x,x	x,x	11,14	1,23	x,x	x,x	x,x	x,x	x,x	6,12	22,26	17,30	x,x	x,x	15,29	x,x	x x	x x	19,21	x x	0,7	x x	2,3	8,27	9,25	5,28	x x
α^{16}	4,27	x,x	12,17	9,19	x,x	3,23	x,x	x,x	10,13	0,22	x,x	x,x	x,x	x,x	x x	5,11	21,25	16,29	x,x	x x	14,28	x,x	x x	x x	18,20	x x	6,30	x,x	1,2	7,26	8,23
α^{17}	6,25	7,23	3,26	x,x	11,16	8,18	x,x	2,22	x,x	x,x	9,12	21,30	x,x	x,x	x x	x,x	x,x	4,10	20,24	15,28	x x	x x	13,27	x x	x x	x x	17,19	x,x	5,29	x,x	0,1
α^{18}	x,x	0,30	5,24	6,22	2,25	x,x	10,15	7,17	x,x	1,21	x,x	x,x	x,x	8,11	20,29	x x	x x	x,x	x,x	x,x	3,9	19,23	14,27	x x	x x	12,26	x x	x x	x x	16,18	x x	4,28
α^{19}	x,x	3,27	x,x	29,30	4,23	5,21	1,24	x,x	9,14	6,16	x x	0,20	x,x	x x	7,10	19,28	x x	x x	x,x	x,x	x x	2,8	18,22	13,26	x x	x x	11,25	x x	x x	x x	15,17
α^{20}	x,x	14,16	x,x	2,26	x,x	28,29	3,22	4,20	0,23	x,x	8,13	5,15	x,x	19,30	x x	x,x	6,9	18,27	x,x	x,x	x x	x x	x x	1,7	17,21	12,25	x x	x x	10,24	x x	x x
α^{21}	x,x	x,x	x,x	13,15	x,x	1,25	x,x	27,28	2,21	3,19	22,30	x,x	7,12	4,14	x,x	18,29	x x	x,x	5,8	17,26	x x	x,x	x x	x x	x x	0,6	16,20	11,24	x x	x x	9,23
α^{22}	x,x	8,22	x,x	x,x	x,x	12,14	x,x	0,24	x,x	26,27	1,20	2,18	21,29	x,x	6,11	3,13	x x	17,28	x x	x x	4,7	16,25	x x	x x	x x	x x	x x	5,10	15,19	10,23	x x
α^{23}	9,22	x,x	x,x	7,21	x,x	x,x	x,x	11,13	x,x	23,30	x,x	25,26	0,19	1,17	20,28	x,x	5,10	2,12	x x	16,27	x x	x x	3,6	15,24	x x	x x	x x	x,x	x x	4,22	14,18
α^{24}	3,28	13,17	8,21	x,x	x,x	6,20	x,x	x,x	x,x	10,12	x,x	22,29	x,x	24,25	18,30	0,16	19,27	x x	4,9	1,11	x x	15,26	x x	x x	2,5	11,23	x x	x x	x x	x x	x x
α^{25}	x,x	x,x	2,27	12,16	7,20	x,x	x,x	5,19	x,x	x,x	x,x	9,11	x,x	21,28	x,x	23,24	17,29	15,30	18,26	x x	3,8	0,10	x x	14,25	x x	x x	1,4	13,22	x x	x x	x x
α^{26}	x,x	x,x	x,x	x,x	1,26	11,15	6,19	x,x	x,x	4,18	x,x	x,x	x,x	8,10	x,x	20,27	x,x	22,23	16,28	14,29	17,25	x x	2,7	9,30	x x	13,21	x x	x x	0,3	12,21	x x
α^{27}	11,20	x,x	x,x	x,x	x,x	x,x	0,25	10,14	5,18	x,x	x,x	3,17	x,x	x,x	x,x	7,9	x,x	19,26	x,x	21,22	15,27	13,28	16,24	x x	1,6	8,29	x x	12,23	x x	x x	2,30
α^{28}	x,x	1,29	10,19	x,x	x,x	x,x	x,x	24,30	9,13	4,17	x,x	x,x	2,16	x x	x x	x x	x x	6,8	x x	18,25	x x	20,21	14,26	12,27	15,23	x x	0,5	7,28	x x	11,22	x x
α^{29}	10,21	x,x	x,x	0,28	9,18	x,x	x,x	x,x	x,x	x,x	23,29	8,12	3,16	x x	x,x	1,15	x,x	x x	x,x	5,7	x,x	17,24	x x	19,29	13,25	11,26	14,22	x x	4,30	6,27	x x
α^{30}	5,26	x,x	9,20	x,x	x,x	27,30	8,17	x,x	x,x	x,x	x,x	x,x	22,28	7,11	2,15	x x	x,x	0,14	x x	x,x	x x	4,6	x x	16,23	x x	18,19	12,21	10,25	13,21	x x	3,29

Table 4.7: Solution table for $m = 5$

Table of error positions (m = 6, T = 2)

$\sigma(z) = 1 + \sigma_1 z + \sigma_2 z^2$

$\downarrow \sigma_1 \quad \sigma_2 \rightarrow$	α^0	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9
α^0	21,42	16,46	29,32	11,49	1,58	x,x	22,35	57,62	2,53	13,41
α^1	x,x	x,x	20,41	15,45	28,31	10,48	0,57	x,x	21,34	56,61
α^2	x,x	x,x	x,x	x,x	19,40	14,44	27,30	9,47	56,62	x,x
α^3	x,x	x,x	x,x	x,x	x,x	x,x	18,39	13,43	26,29	8,46
α^4	x,x	11,51	x,x	x,x	x,x	x,x	x,x	x,x	17,38	12,42
α^5	x,x	13,49	x,x	10,50	x,x	x,x	x,x	x,x	x,x	x,x
α^6	x,x	0,62	x,x	12,48	x,x	9,49	x,x	x,x	x,x	x,x
α^7	23,40	x,x	x,x	61,62	x,x	11,47	x,x	8,48	x,x	x,x
α^8	x,x	20,42	22,39	x,x	x,x	60,61	x,x	10,46	x,x	7,47
α^9	27,36	x,x	x,x	19,41	21,38	x,x	x,x	59,60	x,x	9,45
α^{10}	x,x	x,x	26,35	x,x	x,x	18,40	20,37	x,x	x,x	58,59
α^{11}	1,62	x,x	x,x	x,x	25,34	x,x	x,x	17,39	19,36	x,x
α^{12}	x,x	x,x	0,61	x,x	x,x	x,x	24,33	x,x	x,x	16,38
α^{13}	x,x	27,35	x,x	x,x	60,62	x,x	x,x	x,x	23,32	x,x
α^{14}	17,46	24,38	x,x	26,34	x,x	x,x	59,61	x,x	x,x	x,x
α^{15}	22,41	x,x	16,45	23,37	x,x	25,33	x,x	x,x	58,60	x,x
α^{16}	x,x	7,55	21,40	x,x	15,44	22,36	x,x	24,32	x,x	x,x
α^{17}	x,x	x,x	x,x	6,54	20,39	x,x	14,43	21,35	x,x	23,31
α^{18}	9,54	21,41	x,x	x,x	x,x	5,53	19,38	x,x	13,42	20,34
α^{19}	x,x	15,47	8,53	20,40	x,x	x,x	x,x	4,52	18,37	x,x
α^{20}	x,x	5,57	x,x	14,46	7,52	19,39	x,x	x,x	x,x	3,51
α^{21}	x,x	x,x	x,x	4,56	x,x	13,45	6,51	18,38	x,x	x,x
α^{22}	2,61	x,x	x,x	x,x	x,x	3,55	x,x	12,44	5,50	17,37
α^{23}	x,x	3,59	1,60	x,x	x,x	x,x	x,x	2,54	x,x	11,43
α^{24}	x,x	19,43	x,x	2,58	0,59	x,x	x,x	x,x	x,x	1,53
α^{25}	8,55	26,36	x,x	18,42	x,x	1,57	58,62	x,x	x,x	x,x
α^{26}	x,x	18,44	7,54	25,35	x,x	17,41	x,x	0,56	57,61	x,x
α^{27}	14,49	x,x	x,x	17,43	6,53	24,34	x,x	16,40	x,x	55,62
α^{28}	29,34	25,37	13,48	x,x	x,x	16,42	5,52	23,33	x,x	15,39
α^{29}	x,x	6,56	28,33	24,36	12,47	x,x	x,x	15,41	4,51	22,32
α^{30}	19,44	28,34	x,x	5,55	27,32	23,35	11,46	x,x	x,x	14,40
α^{31}	15,48	1,61	18,43	27,33	x,x	4,54	26,31	22,34	10,45	x,x
α^{32}	x,x	10,52	14,47	0,60	17,42	26,32	x,x	3,53	25,30	21,33
α^{33}	x,x	x,x	x,x	9,51	13,46	59,62	16,41	25,31	x,x	2,52
α^{34}	x,x	x,x	x,x	x,x	x,x	8,50	12,45	58,61	15,40	24,30
α^{35}	20,43	x,x	x,x	x,x	x,x	x,x	x,x	7,49	11,44	57,60
α^{36}	18,45	x,x	19,42	x,x	x,x	x,x	x,x	x,x	x,x	6,48
α^{37}	31,32	x,x	17,44	x,x	18,41	x,x	x,x	x,x	x,x	x,x
α^{38}	x,x	x,x	30,31	x,x	16,43	x,x	17,40	x,x	x,x	x,x
α^{39}	11,52	8,54	x,x	x,x	29,30	x,x	15,42	x,x	16,39	x,x
α^{40}	x,x	x,x	10,51	7,53	x,x	x,x	28,29	x,x	14,41	x,x
α^{41}	x,x	4,58	x,x	x,x	9,50	6,52	x,x	x,x	27,28	x,x
α^{42}	x,x	x,x	x,x	3,57	x,x	x,x	8,49	5,51	x,x	x,x
α^{43}	x,x	30,32	x,x	x,x	x,x	2,56	x,x	x,x	7,48	4,50
α^{44}	4,59	x,x	x,x	29,31	x,x	x,x	x,x	1,55	x,x	x,x
α^{45}	7,56	x,x	3,58	x,x	x,x	28,30	x,x	x,x	x,x	0,54
α^{46}	x,x	14,48	6,55	x,x	2,57	x,x	x,x	27,29	x,x	x,x
α^{47}	24,39	9,53	x,x	13,47	5,54	x,x	1,56	x,x	x,x	26,28
α^{48}	x,x	x,x	23,38	8,52	x,x	12,46	4,53	x,x	0,55	x,x
α^{49}	10,53	x,x	x,x	x,x	22,37	7,51	x,x	11,45	3,52	x,x
α^{50}	16,47	22,40	9,52	x,x	x,x	x,x	21,36	6,50	x,x	10,44
α^{51}	26,37	x,x	15,46	21,39	8,51	x,x	x,x	x,x	20,35	5,49
α^{52}	x,x	x,x	25,36	x,x	14,45	20,38	7,50	x,x	x,x	x,x
α^{53}	x,x	x,x	x,x	x,x	24,35	x,x	13,44	19,37	6,49	x,x
α^{54}	28,35	29,33	x,x	x,x	x,x	x,x	23,34	x,x	12,43	18,36
α^{55}	12,51	x,x	27,34	28,32	x,x	x,x	x,x	x,x	22,33	x,x
α^{56}	5,58	x,x	11,50	x,x	26,33	27,31	x,x	x,x	x,x	x,x
α^{57}	13,50	23,39	4,57	x,x	10,49	x,x	25,32	26,30	x,x	x,x
α^{58}	x,x	x,x	12,49	22,38	3,56	x,x	9,48	x,x	24,31	25,29
α^{59}	6,57	17,45	x,x	x,x	11,48	21,37	2,55	x,x	8,47	x,x
α^{60}	25,38	2,60	5,56	16,44	x,x	x,x	10,47	20,36	1,54	x,x
α^{61}	3,60	x,x	24,37	1,59	4,55	15,43	x,x	x,x	9,46	19,35
α^{62}	30,33	12,50	2,59	x,x	23,36	0,58	3,54	14,42	x,x	x,x

Table 4.8: Solution table for $m = 6$

Table of error positions (m = 6, T = 2)

$$\sigma(z) = 1 + \sigma_1 z + \sigma_2 z^2$$

α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	α^{15}	α^{16}	α^{17}	α^{18}	α^{19}
x,x	x,x	7,44	17,33	51,61	x,x	4,43	x,x	19,26	20,24
1,52	12,40	x,x	x,x	6,43	16,32	50,60	x,x	3,42	x,x
20,33	55,60	0,51	11,39	x,x	x,x	5,42	15,31	49,59	x,x
55,61	x,x	19,32	54,59	50,62	10,38	x,x	x,x	4,41	14,30
25,28	7,45	54,60	x,x	18,31	53,58	49,61	9,37	x,x	x,x
16,37	11,41	24,27	6,44	53,59	x,x	17,30	52,57	48,60	8,36
x,x	x,x	15,36	10,40	23,26	5,43	52,58	x,x	16,29	51,56
x,x	x,x	x,x	x,x	14,35	9,39	22,25	4,42	51,57	x,x
x,x	x,x	x,x	x,x	x,x	x,x	13,34	8,38	21,24	3,41
x,x	6,46	x,x	x,x	x,x	x,x	x,x	x,x	12,33	7,37
x,x	8,44	x,x	5,45	x,x	x,x	x,x	x,x	x,x	x,x
x,x	57,58	x,x	7,43	x,x	4,44	x,x	x,x	x,x	x,x
18,35	x,x	x,x	56,57	x,x	6,42	x,x	3,43	x,x	x,x
x,x	15,37	17,34	x,x	x,x	55,56	x,x	5,41	x,x	2,42
22,31	x,x	x,x	14,36	16,33	x,x	x,x	54,55	x,x	4,40
x,x	x,x	21,30	x,x	x,x	13,35	15,32	x,x	x,x	53,54
57,59	x,x	x,x	x,x	20,29	x,x	x,x	12,34	14,31	x,x
x,x	x,x	56,58	x,x	x,x	x,x	19,28	x,x	x,x	11,33
x,x	22,30	x,x	x,x	55,57	x,x	x,x	x,x	18,27	x,x
12,41	19,33	x,x	21,29	x,x	x,x	54,56	x,x	x,x	x,x
17,36	x,x	11,40	18,32	x,x	20,28	x,x	x,x	53,55	x,x
x,x	2,50	16,35	x,x	10,39	17,31	x,x	19,27	x,x	x,x
x,x	x,x	x,x	1,49	15,34	x,x	9,38	16,30	x,x	18,26
4,49	16,36	x,x	x,x	x,x	0,48	14,33	x,x	8,37	15,29
x,x	10,42	3,48	15,35	x,x	x,x	x,x	47,62	13,32	x,x
x,x	0,52	x,x	9,41	2,47	14,34	x,x	x,x	x,x	46,61
x,x	x,x	x,x	51,62	x,x	8,40	1,46	13,33	x,x	x,x
56,60	x,x	x,x	x,x	x,x	50,61	x,x	7,39	0,45	12,32
x,x	54,61	55,59	x,x	x,x	x,x	x,x	49,60	x,x	6,38
x,x	14,38	x,x	53,60	54,58	x,x	x,x	x,x	x,x	48,59
3,50	21,31	x,x	13,37	x,x	52,59	53,57	x,x	x,x	x,x
x,x	13,39	2,49	20,30	x,x	12,36	x,x	51,58	52,56	x,x
9,44	x,x	x,x	12,38	1,48	19,29	x,x	11,35	x,x	50,57
24,29	20,32	8,43	x,x	x,x	11,37	0,47	18,28	x,x	10,34
x,x	1,51	23,28	19,31	7,42	x,x	x,x	10,36	46,62	17,27
14,39	23,29	x,x	0,50	22,27	18,30	6,41	x,x	x,x	9,35
10,43	56,59	13,38	22,28	x,x	49,62	21,26	17,29	5,40	x,x
x,x	5,47	9,42	55,58	12,37	21,27	x,x	48,61	20,25	16,28
x,x	x,x	x,x	4,46	8,41	54,57	11,36	20,26	x,x	47,60
x,x	x,x	x,x	x,x	x,x	3,45	7,40	53,56	10,35	19,25
15,38	x,x	x,x	x,x	x,x	x,x	x,x	2,44	6,39	52,55
13,40	x,x	14,37	x,x	x,x	x,x	x,x	x,x	x,x	1,43
26,27	x,x	12,39	x,x	13,36	x,x	x,x	x,x	x,x	x,x
x,x	x,x	25,26	x,x	11,38	x,x	12,35	x,x	x,x	x,x
6,47	3,49	x,x	x,x	24,25	x,x	10,37	x,x	11,34	x,x
x,x	x,x	5,46	2,48	x,x	x,x	23,24	x,x	9,36	x,x
x,x	53,62	x,x	x,x	4,45	1,47	x,x	x,x	22,23	x,x
x,x	x,x	x,x	52,61	x,x	x,x	3,44	0,46	x,x	x,x
x,x	25,27	x,x	x,x	x,x	51,60	x,x	x,x	2,43	45,62
54,62	x,x	x,x	24,26	x,x	x,x	x,x	50,59	x,x	x,x
2,51	x,x	53,61	x,x	x,x	23,25	x,x	x,x	x,x	49,58
x,x	9,43	1,50	x,x	52,60	x,x	x,x	22,24	x,x	x,x
19,34	4,48	x,x	8,42	0,49	x,x	51,59	x,x	x,x	21,23
x,x	x,x	18,33	3,47	x,x	7,41	48,62	x,x	50,58	x,x
5,48	x,x	x,x	x,x	17,32	2,46	x,x	6,40	47,61	x,x
11,42	17,35	4,47	x,x	x,x	x,x	16,31	1,45	x,x	5,39
21,32	x,x	10,41	16,34	3,46	x,x	x,x	x,x	15,30	0,44
x,x	x,x	20,31	x,x	9,40	15,33	2,45	x,x	x,x	x,x
x,x	x,x	x,x	x,x	19,30	x,x	8,39	14,32	1,44	x,x
23,30	24,28	x,x	x,x	x,x	x,x	18,29	x,x	7,38	13,31
7,46	x,x	22,29	23,27	x,x	x,x	x,x	x,x	17,28	x,x
0,53	x,x	6,45	x,x	21,28	22,26	x,x	x,x	x,x	x,x
8,45	18,34	52,62	x,x	5,44	x,x	20,27	21,25	x,x	x,x

Table 4.8: Solution table for $m = 6$ (continue)

Table of error positions (m = 6, T = 2)

$\sigma(z) = 1 + \sigma_1 z + \sigma_2 z^2$

α^{20}	α^{21}	α^{22}	α^{23}	α^{24}	α^{25}	α^{26}	α^{27}	α^{28}	α^{29}
x,x	x,x	x,x	x,x	14,25	x,x	3,34	9,27	39,59	x,x
18,25	19,23	x,x	x,x	x,x	x,x	13,24	x,x	2,33	8,26
2,41	x,x	17,24	18,22	x,x	x,x	x,x	x,x	12,23	x,x
48,58	x,x	1,40	x,x	16,23	17,21	x,x	x,x	x,x	x,x
3,40	13,29	47,57	x,x	0,39	x,x	15,22	16,20	x,x	x,x
x,x	x,x	2,39	12,28	46,56	x,x	38,62	x,x	14,21	15,19
47,59	7,35	x,x	x,x	1,38	11,27	45,55	x,x	37,61	x,x
15,28	50,55	46,58	6,34	x,x	x,x	0,37	10,26	44,54	x,x
50,56	x,x	14,27	49,54	45,57	5,33	x,x	x,x	36,62	9,25
20,23	2,40	49,55	x,x	13,26	48,53	44,56	4,32	x,x	x,x
11,32	6,36	19,22	1,39	48,54	x,x	12,25	47,52	43,55	3,31
x,x	x,x	10,31	5,35	18,21	0,38	47,53	x,x	11,24	46,51
x,x	x,x	x,x	x,x	9,30	4,34	17,20	37,62	46,52	x,x
x,x	x,x	x,x	x,x	x,x	x,x	8,29	3,33	16,19	36,61
x,x	1,41	x,x	x,x	x,x	x,x	x,x	x,x	7,28	2,32
x,x	3,39	x,x	0,40	x,x	x,x	x,x	x,x	x,x	x,x
x,x	52,53	x,x	2,38	x,x	39,62	x,x	x,x	x,x	x,x
13,30	x,x	x,x	51,52	x,x	1,37	x,x	38,61	x,x	x,x
x,x	10,32	12,29	x,x	x,x	50,51	x,x	0,36	x,x	37,60
17,26	x,x	x,x	9,31	11,28	x,x	x,x	49,50	x,x	35,62
x,x	x,x	16,25	x,x	x,x	8,30	10,27	x,x	x,x	48,49
52,54	x,x	x,x	x,x	15,24	x,x	x,x	7,29	9,26	x,x
x,x	x,x	51,53	x,x	x,x	x,x	14,23	x,x	x,x	6,28
x,x	17,25	x,x	x,x	50,52	x,x	x,x	x,x	13,22	x,x
7,36	14,28	x,x	16,24	x,x	x,x	49,51	x,x	x,x	x,x
12,31	x,x	6,35	13,27	x,x	15,23	x,x	x,x	48,50	x,x
x,x	45,60	11,30	x,x	5,34	12,26	x,x	14,22	x,x	x,x
x,x	x,x	x,x	44,59	10,29	x,x	4,33	11,25	x,x	13,21
44,62	11,31	x,x	x,x	x,x	43,58	9,28	x,x	3,32	10,24
x,x	5,37	43,61	10,30	x,x	x,x	x,x	42,57	8,27	x,x
x,x	47,58	x,x	4,36	42,60	9,29	x,x	x,x	x,x	41,56
x,x	x,x	x,x	46,57	x,x	3,35	41,59	8,28	x,x	x,x
51,55	x,x	x,x	x,x	x,x	45,56	x,x	2,34	40,58	7,27
x,x	49,56	50,54	x,x	x,x	x,x	x,x	44,55	x,x	1,33
x,x	9,33	x,x	48,55	49,53	x,x	x,x	x,x	x,x	43,54
45,61	16,26	x,x	8,32	x,x	47,54	48,52	x,x	x,x	x,x
x,x	8,34	44,60	15,25	x,x	7,31	x,x	46,53	47,51	x,x
4,39	x,x	x,x	7,33	43,59	14,24	x,x	6,30	x,x	45,52
19,24	15,27	3,38	x,x	x,x	6,32	42,58	13,23	x,x	5,29
x,x	46,59	18,23	14,26	2,37	x,x	x,x	5,31	41,57	12,22
9,34	18,24	x,x	45,58	17,22	13,25	1,36	x,x	x,x	4,30
5,38	51,54	8,33	17,23	x,x	44,57	16,21	12,24	0,35	x,x
x,x	0,42	4,37	50,53	7,32	16,22	x,x	43,56	15,20	11,23
x,x	x,x	x,x	41,62	3,36	49,52	6,31	15,21	x,x	42,55
x,x	x,x	x,x	x,x	x,x	40,61	2,35	48,51	5,30	14,20
10,33	x,x	x,x	x,x	x,x	x,x	x,x	39,60	1,34	47,50
8,35	x,x	9,32	x,x	x,x	x,x	x,x	x,x	x,x	38,59
21,22	x,x	7,34	x,x	8,31	x,x	x,x	x,x	x,x	x,x
x,x	x,x	20,21	x,x	6,33	x,x	7,30	x,x	x,x	x,x
1,42	44,61	x,x	x,x	19,20	x,x	5,32	x,x	6,29	x,x
x,x	x,x	0,41	43,60	x,x	x,x	18,19	x,x	4,31	x,x
x,x	48,57	x,x	x,x	40,62	42,59	x,x	x,x	17,18	x,x
x,x	x,x	x,x	47,56	x,x	x,x	39,61	41,58	x,x	x,x
x,x	20,22	x,x	x,x	x,x	46,55	x,x	x,x	38,60	40,57
49,57	x,x	x,x	19,21	x,x	x,x	x,x	45,54	x,x	x,x
46,60	x,x	48,56	x,x	x,x	18,20	x,x	x,x	x,x	44,53
x,x	4,38	45,59	x,x	47,55	x,x	x,x	17,19	x,x	x,x
14,29	43,62	x,x	3,37	44,58	x,x	46,54	x,x	x,x	16,18
x,x	x,x	13,28	42,61	x,x	2,36	43,57	x,x	45,53	x,x
0,43	x,x	x,x	x,x	12,27	41,60	x,x	1,35	42,56	x,x
6,37	12,30	42,62	x,x	x,x	x,x	11,26	40,59	x,x	0,34
16,27	x,x	5,36	11,29	41,61	x,x	x,x	x,x	10,25	39,58
x,x	x,x	15,26	x,x	4,35	10,28	40,60	x,x	x,x	x,x

Table 4.8: Solution table for $m = 6$ (continue)

$\sigma(z) = 1 + \sigma_1 z + \sigma_2 z^2$

α^{30}	α^{31}	α^{32}	α^{33}	α^{34}	α^{35}	α^{36}	α^{37}	α^{38}	α^{39}
x,x	x,x	8,23	37,56	x,x	31,60	38,52	x,x	40,48	x,x
38,58	x,x	x,x	x,x	7,22	36,55	x,x	30,59	37,51	x,x
1,32	7,25	37,57	x,x	x,x	x,x	6,21	35,54	x,x	29,58
11,22	x,x	0,31	6,24	36,56	x,x	x,x	x,x	5,20	34,53
x,x	x,x	10,21	x,x	30,62	5,23	35,55	x,x	x,x	x,x
x,x	x,x	x,x	x,x	9,20	x,x	29,61	4,22	34,54	x,x
13,20	14,18	x,x	x,x	x,x	x,x	8,19	x,x	28,60	3,21
36,60	x,x	12,19	13,17	x,x	x,x	x,x	x,x	7,18	x,x
43,53	x,x	35,59	x,x	11,18	12,16	x,x	x,x	x,x	x,x
35,61	8,24	42,52	x,x	34,58	x,x	10,17	11,15	x,x	x,x
x,x	x,x	34,60	7,23	41,51	x,x	33,57	x,x	9,16	10,14
42,54	2,30	x,x	x,x	33,59	6,22	40,50	x,x	32,56	x,x
10,23	45,50	41,53	1,29	x,x	x,x	32,58	5,21	39,49	x,x
45,51	x,x	9,22	44,49	40,52	0,28	x,x	x,x	31,57	4,20
15,18	35,60	44,50	x,x	8,21	43,48	39,51	27,62	x,x	x,x
6,27	1,31	14,17	34,59	43,49	x,x	7,20	42,47	38,50	26,61
x,x	x,x	5,26	0,30	13,16	33,58	42,48	x,x	6,19	41,46
x,x	x,x	x,x	x,x	4,25	29,62	12,15	32,57	41,47	x,x
x,x	x,x	x,x	x,x	x,x	x,x	3,24	28,61	11,14	31,56
x,x	,59	x,x	x,x	x,x	x,x	x,x	x,x	2,23	27,60
x,x	4,61	x,x	35,58	x,x	x,x	x,x	x,x	x,x	x,x
x,x	47,48	x,x	33,60	x,x	34,57	x,x	x,x	x,x	x,x
8,25	x,x	x,x	46,47	x,x	32,59	x,x	33,56	x,x	x,x
x,x	5,27	7,24	x,x	x,x	45,46	x,x	31,58	x,x	32,55
12,21	x,x	x,x	4,26	6,23	x,x	x,x	44,45	x,x	30,57
x,x	x,x	11,20	x,x	x,x	3,25	5,22	x,x	x,x	43,44
47,49	x,x	x,x	x,x	10,19	x,x	x,x	2,24	4,21	x,x
x,x	x,x	46,48	x,x	x,x	x,x	9,18	x,x	x,x	1,23
x,x	12,20	x,x	x,x	45,47	x,x	x,x	x,x	8,17	x,x
2,31	9,23	x,x	11,19	x,x	x,x	44,46	x,x	x,x	x,x
7,26	x,x	1,30	8,22	x,x	10,18	x,x	x,x	43,45	x,x
x,x	40,55	6,25	x,x	0,29	7,21	x,x	9,17	x,x	x,x
x,x	x,x	x,x	39,54	5,24	x,x	28,62	6,20	x,x	8,16
39,57	6,26	x,x	x,x	x,x	38,53	4,23	x,x	27,61	5,19
x,x	0,32	38,56	5,25	x,x	x,x	x,x	37,52	3,22	x,x
x,x	42,53	x,x	31,62	37,55	4,24	x,x	x,x	x,x	36,51
x,x	x,x	x,x	41,52	x,x	30,61	36,54	3,23	x,x	x,x
46,50	x,x	x,x	x,x	x,x	40,51	x,x	29,60	35,53	2,22
x,x	44,51	45,49	x,x	x,x	x,x	x,x	39,50	x,x	28,59
x,x	4,28	x,x	43,50	44,48	x,x	x,x	x,x	x,x	38,49
40,56	11,21	x,x	3,27	x,x	42,49	43,47	x,x	x,x	x,x
x,x	3,29	39,55	10,20	x,x	2,26	x,x	41,48	42,46	x,x
34,62	x,x	x,x	2,28	38,54	9,19	x,x	1,25	x,x	40,47
14,19	10,22	33,61	x,x	x,x	1,27	37,53	8,18	x,x	0,24
x,x	41,54	13,18	9,21	32,60	x,x	x,x	0,26	36,52	7,17
4,29	13,19	x,x	40,53	12,17	8,20	31,59	x,x	x,x	25,62
0,33	46,49	3,28	12,18	x,x	39,52	11,16	7,19	30,58	x,x
x,x	37,58	32,62	45,48	2,27	11,17	x,x	38,51	10,15	6,18
x,x	x,x	x,x	36,57	31,61	44,47	1,26	10,16	x,x	37,50
x,x	x,x	x,x	x,x	x,x	35,56	30,60	43,46	0,25	9,15
5,28	x,x	x,x	x,x	x,x	x,x	x,x	34,55	29,59	42,45
3,30	x,x	4,27	x,x	x,x	x,x	x,x	x,x	x,x	33,54
16,17	x,x	2,29	x,x	3,26	x,x	x,x	x,x	x,x	x,x
x,x	x,x	15,16	x,x	1,28	x,x	2,25	x,x	x,x	x,x
37,59	39,56	x,x	x,x	14,15	x,x	0,27	x,x	1,24	x,x
x,x	x,x	36,58	38,55	x,x	x,x	13,14	x,x	26,62	x,x
x,x	43,52	x,x	x,x	35,57	37,54	x,x	x,x	12,13	x,x
x,x	x,x	x,x	42,51	x,x	x,x	34,56	36,53	x,x	x,x
x,x	15,17	x,x	x,x	x,x	41,50	x,x	x,x	33,55	35,52
44,52	x,x	x,x	14,16	x,x	x,x	x,x	40,49	x,x	x,x
41,55	x,x	43,51	x,x	x,x	13,15	x,x	x,x	x,x	39,48
x,x	33,62	40,54	x,x	42,50	x,x	x,x	12,14	x,x	x,x
9,24	38,57	x,x	32,61	39,53	x,x	41,49	x,x	x,x	11,13

Table 4.8: Solution table for $m = 6$ (continue)

Table of error positions (m = 6, T = 2)

$$\sigma(x) = 1 + \sigma_1 x + \sigma_2 x^2$$

α^{40}	α^{41}	α^{42}	α^{43}	α^{44}	α^{45}	α^{46}	α^{47}	α^{48}	α^{49}
x,x	10,12	x,x	x,x	x,x	36,45	x,x	x,x	28,50	30,47
39,47	x,x	x,x	9,11	x,x	x,x	x,x	35,44	x,x	x,x
36,50	x,x	38,46	x,x	x,x	8,10	x,x	x,x	x,x	34,43
x,x	28,57	35,49	x,x	37,45	x,x	x,x	7,9	x,x	x,x
4,19	33,52	x,x	27,56	34,48	x,x	36,44	x,x	x,x	6,8
x,x	x,x	3,18	32,51	x,x	26,55	33,47	x,x	35,43	x,x
33,53	x,x	x,x	x,x	2,17	31,50	x,x	25,54	32,46	x,x
27,59	2,20	32,52	x,x	x,x	x,x	1,16	30,49	x,x	24,53
6,17	x,x	26,58	1,19	31,51	x,x	x,x	x,x	0,15	29,48
x,x	x,x	5,16	x,x	25,57	0,18	30,50	x,x	x,x	x,x
x,x	x,x	x,x	x,x	4,15	x,x	24,56	17,62	29,49	x,x
8,15	9,13	x,x	x,x	x,x	x,x	3,14	x,x	23,55	16,61
31,55	x,x	7,14	8,12	x,x	x,x	x,x	x,x	2,13	x,x
38,48	x,x	30,54	x,x	6,13	7,11	x,x	x,x	x,x	x,x
30,56	3,19	37,47	x,x	29,53	x,x	5,12	6,10	x,x	x,x
x,x	x,x	29,55	2,18	36,46	x,x	28,52	x,x	4,11	5,9
37,49	25,60	x,x	x,x	28,54	1,17	35,45	x,x	27,51	x,x
5,18	40,45	36,48	24,59	x,x	x,x	27,53	0,16	34,44	x,x
40,46	x,x	4,17	39,44	35,47	23,58	x,x	x,x	26,52	15,62
10,13	30,55	39,45	x,x	3,16	38,43	34,46	22,57	x,x	x,x
1,22	26,59	9,12	29,54	38,44	x,x	2,15	37,42	33,45	21,56
x,x	x,x	0,21	25,58	8,11	28,53	37,43	x,x	1,14	36,41
x,x	x,x	x,x	x,x	20,62	24,57	7,10	27,52	36,42	x,x
x,x	x,x	x,x	x,x	x,x	x,x	19,61	23,56	6,9	26,51
x,x	31,54	x,x	x,x	x,x	x,x	x,x	x,x	18,60	22,55
x,x	29,56	x,x	30,53	x,x	x,x	x,x	x,x	x,x	x,x
x,x	42,43	x,x	28,55	x,x	29,52	x,x	x,x	x,x	x,x
3,20	x,x	x,x	41,42	x,x	27,54	x,x	28,51	x,x	x,x
x,x	0,22	2,19	x,x	x,x	40,41	x,x	26,53	x,x	27,50
7,16	x,x	x,x	21,62	1,18	x,x	x,x	39,40	x,x	25,52
x,x	x,x	6,15	x,x	x,x	20,61	0,17	x,x	x,x	38,39
42,44	x,x	x,x	x,x	5,14	x,x	x,x	19,60	16,62	x,x
x,x	x,x	41,43	x,x	x,x	x,x	4,13	x,x	x,x	18,59
x,x	7,15	x,x	x,x	40,42	x,x	x,x	x,x	3,12	x,x
26,60	4,18	x,x	6,14	x,x	x,x	39,41	x,x	x,x	x,x
2,21	x,x	25,59	3,17	x,x	5,13	x,x	x,x	38,40	x,x
x,x	35,50	1,20	x,x	24,58	2,16	x,x	4,12	x,x	x,x
x,x	x,x	x,x	34,49	0,19	x,x	23,57	1,15	x,x	3,11
34,52	1,21	x,x	x,x	x,x	33,48	18,62	x,x	22,56	0,14
x,x	27,58	33,51	0,20	x,x	x,x	x,x	32,47	17,61	x,x
x,x	37,48	x,x	26,57	32,50	19,62	x,x	x,x	x,x	31,46
x,x	x,x	x,x	36,47	x,x	25,56	31,49	18,61	x,x	x,x
41,45	x,x	x,x	x,x	x,x	35,46	x,x	24,55	30,48	17,60
x,x	39,46	40,44	x,x	x,x	x,x	x,x	34,45	x,x	23,54
x,x	23,62	x,x	38,45	39,43	x,x	x,x	x,x	x,x	33,44
35,51	6,16	x,x	22,61	x,x	37,44	38,42	x,x	x,x	x,x
x,x	24,61	34,50	5,15	x,x	21,60	x,x	36,43	37,41	x,x
29,57	x,x	x,x	23,60	33,49	4,14	x,x	20,59	x,x	35,42
9,14	5,17	28,56	x,x	x,x	22,59	32,48	3,13	x,x	19,58
x,x	36,49	8,13	4,16	27,55	x,x	x,x	21,58	31,47	2,12
24,62	8,14	x,x	35,48	7,12	3,15	26,54	x,x	x,x	20,57
28,58	41,44	23,61	7,13	x,x	34,47	6,11	2,14	25,53	x,x
x,x	32,53	27,57	40,43	22,60	6,12	x,x	33,46	5,10	1,13
x,x	x,x	x,x	31,52	26,56	39,42	21,59	5,11	x,x	32,45
x,x	x,x	x,x	x,x	x,x	30,51	25,55	38,41	20,58	4,10
0,23	x,x	x,x	x,x	x,x	x,x	x,x	29,50	24,54	37,40
25,61	x,x	22,62	x,x	x,x	x,x	x,x	x,x	x,x	28,49
11,12	x,x	24,60	x,x	21,61	x,x	x,x	x,x	x,x	x,x
x,x	x,x	10,11	x,x	23,59	x,x	20,60	x,x	x,x	x,x
32,54	34,51	x,x	x,x	9,10	x,x	22,58	x,x	19,59	x,x
x,x	x,x	31,53	33,50	x,x	x,x	8,9	x,x	21,57	x,x
x,x	38,47	x,x	x,x	30,52	32,49	x,x	x,x	7,8	x,x
x,x	x,x	x,x	37,46	x,x	x,x	29,51	31,48	x,x	x,x

Table 4.8: Solution table for $m = 6$ (continue)

Table of error positions (m = 6, l = 2)

$\sigma(z) = 1 + \sigma_1 z + \sigma_2 z^2$

α^{50}	α^{51}	α^{52}	α^{53}	α^{54}	α^{55}	α^{56}	α^{57}	α^{58}	α^{59}	α^{60}	α^{61}	α^{62}
x,x	x,x	5,6	x,x	18,54	x,x	15,55	x,x	x,x	x,x	x,x	x,x	x,x
27,49	29,46	x,x	x,x	4,5	x,x	17,53	x,x	14,54	x,x	x,x	x,x	x,x
x,x	x,x	26,48	28,45	x,x	x,x	3,4	x,x	16,52	x,x	13,53	x,x	x,x
x,x	33,42	x,x	x,x	25,47	27,44	x,x	x,x	2,3	x,x	15,51	x,x	12,52
x,x	x,x	x,x	32,41	x,x	x,x	24,46	26,43	x,x	x,x	1,2	x,x	14,50
x,x	5,7	x,x	x,x	x,x	31,40	x,x	x,x	23,45	25,42	x,x	x,x	0,1
34,42	x,x	x,x	4,6	x,x	x,x	x,x	30,39	x,x	x,x	22,44	24,41	x,x
31,45	x,x	33,41	x,x	x,x	3,5	x,x	x,x	x,x	29,38	x,x	x,x	21,43
x,x	23,52	30,44	x,x	32,40	x,x	x,x	2,4	x,x	x,x	x,x	28,37	x,x
14,62	28,47	x,x	22,51	29,43	x,x	31,39	x,x	x,x	1,3	x,x	x,x	x,x
x,x	x,x	13,61	27,46	x,x	21,50	28,42	x,x	30,38	x,x	x,x	0,2	x,x
28,48	x,x	x,x	x,x	12,60	26,45	x,x	20,49	27,41	x,x	29,37	x,x	x,x
22,54	15,60	27,47	x,x	x,x	x,x	11,59	25,44	x,x	19,48	26,40	x,x	28,36
1,12	x,x	21,53	14,59	26,46	x,x	x,x	x,x	10,58	24,43	x,x	18,47	25,39
x,x	x,x	0,11	x,x	20,52	13,58	25,45	x,x	x,x	x,x	9,57	23,42	x,x
x,x	x,x	x,x	x,x	10,62	x,x	19,51	12,57	24,44	x,x	x,x	x,x	8,56
3,10	4,8	x,x	x,x	x,x	x,x	9,61	x,x	18,50	11,56	23,43	x,x	x,x
26,50	x,x	2,9	3,7	x,x	x,x	x,x	x,x	8,60	x,x	17,49	10,55	22,42
33,43	x,x	25,49	x,x	1,8	2,6	x,x	x,x	x,x	x,x	7,59	x,x	16,48
25,51	14,61	32,42	x,x	24,48	x,x	0,7	1,5	x,x	x,x	x,x	x,x	6,58
x,x	x,x	24,50	13,60	31,41	x,x	23,47	x,x	6,62	0,4	x,x	x,x	x,x
32,44	20,55	x,x	x,x	23,49	12,59	30,40	x,x	22,46	x,x	5,61	3,62	x,x
0,13	35,40	31,43	19,54	x,x	x,x	22,48	11,58	29,39	x,x	21,45	x,x	4,60
35,41	x,x	12,62	34,39	30,42	18,53	x,x	x,x	21,47	10,57	28,38	x,x	20,44
5,8	25,50	34,40	x,x	11,61	33,38	29,41	17,52	x,x	x,x	20,46	9,56	27,37
17,59	21,54	4,7	24,49	33,39	x,x	10,60	32,37	28,40	16,51	x,x	x,x	19,45
x,x	x,x	16,58	20,53	3,6	23,48	32,38	x,x	9,59	31,36	27,39	15,50	x,x
x,x	x,x	x,x	x,x	15,57	19,52	2,5	22,47	31,37	x,x	8,58	30,35	26,38
x,x	x,x	x,x	x,x	x,x	x,x	14,56	18,51	1,4	21,46	30,36	x,x	7,57
x,x	26,49	x,x	x,x	x,x	x,x	x,x	x,x	13,55	17,50	0,3	20,45	29,35
x,x	24,51	x,x	25,48	x,x	x,x	x,x	x,x	x,x	x,x	12,54	16,49	2,62
x,x	37,38	x,x	23,50	x,x	24,47	x,x	x,x	x,x	x,x	x,x	x,x	11,53
15,61	x,x	x,x	36,37	x,x	22,49	x,x	23,46	x,x	x,x	x,x	x,x	x,x
x,x	17,58	14,60	x,x	x,x	35,36	x,x	21,48	x,x	22,45	x,x	x,x	x,x
2,11	x,x	x,x	16,57	13,59	x,x	x,x	34,35	x,x	20,47	x,x	21,44	x,x
x,x	x,x	1,10	x,x	x,x	15,56	12,58	x,x	x,x	33,34	x,x	19,46	x,x
37,39	x,x	x,x	x,x	0,9	x,x	x,x	14,55	11,57	x,x	x,x	32,33	x,x
x,x	x,x	36,38	x,x	x,x	x,x	8,62	x,x	x,x	13,54	10,56	x,x	x,x
x,x	2,10	x,x	x,x	35,37	x,x	x,x	x,x	7,61	x,x	x,x	12,53	9,55
21,55	13,62	x,x	1,9	x,x	x,x	34,36	x,x	x,x	x,x	6,60	x,x	x,x
16,60	x,x	20,54	12,61	x,x	0,8	x,x	x,x	33,35	x,x	x,x	x,x	5,59
x,x	30,45	15,59	x,x	19,53	11,60	x,x	7,62	x,x	x,x	32,34	x,x	x,x
x,x	x,x	x,x	29,44	14,58	x,x	18,52	10,59	x,x	6,61	x,x	x,x	31,33
29,47	16,59	x,x	x,x	x,x	28,43	13,57	x,x	17,51	9,58	x,x	5,60	x,x
x,x	22,53	28,46	15,58	x,x	x,x	x,x	27,42	12,56	x,x	16,50	8,57	x,x
x,x	32,43	x,x	21,52	27,45	14,57	x,x	x,x	x,x	26,41	11,55	x,x	15,49
x,x	x,x	x,x	31,42	x,x	20,51	26,44	13,56	x,x	x,x	x,x	25,40	10,54
36,40	x,x	x,x	x,x	x,x	30,41	x,x	19,50	25,43	12,55	x,x	x,x	x,x
x,x	34,41	35,39	x,x	x,x	x,x	x,x	29,40	x,x	18,49	24,42	11,54	x,x
x,x	18,57	x,x	33,40	34,38	x,x	x,x	x,x	x,x	28,39	x,x	17,48	23,41
30,46	1,11	x,x	17,56	x,x	32,39	33,37	x,x	x,x	x,x	x,x	27,38	x,x
x,x	19,56	29,45	0,10	x,x	16,55	x,x	31,38	32,36	x,x	x,x	x,x	x,x
24,52	x,x	x,x	18,55	28,44	9,62	x,x	15,54	x,x	30,37	31,35	x,x	x,x
4,9	0,12	23,51	x,x	x,x	17,54	27,43	8,61	x,x	14,53	x,x	29,36	30,34
x,x	31,44	3,8	11,62	22,50	x,x	x,x	16,53	26,42	7,60	x,x	13,52	x,x
19,57	3,9	x,x	30,43	2,7	10,61	21,49	x,x	x,x	15,52	25,41	6,59	x,x
23,53	36,39	18,56	2,8	x,x	29,42	1,6	9,60	20,48	x,x	x,x	14,51	24,40
x,x	27,48	22,52	35,38	17,55	1,7	x,x	28,41	0,5	8,59	19,47	x,x	x,x
x,x	x,x	x,x	26,47	21,51	34,37	16,54	0,6	x,x	27,40	4,62	7,58	18,46
x,x	x,x	x,x	x,x	x,x	25,46	20,50	33,36	15,53	5,62	x,x	26,39	3,61
18,58	x,x	x,x	x,x	x,x	x,x	x,x	24,45	19,49	32,35	14,52	4,61	x,x
20,56	x,x	17,57	x,x	x,x	x,x	x,x	x,x	x,x	23,44	18,48	31,34	13,51
6,7	x,x	19,55	x,x	16,56	x,x	x,x	x,x	x,x	x,x	x,x	22,43	17,47

Table 4.8: Solution table for $m = 6$ (continue)

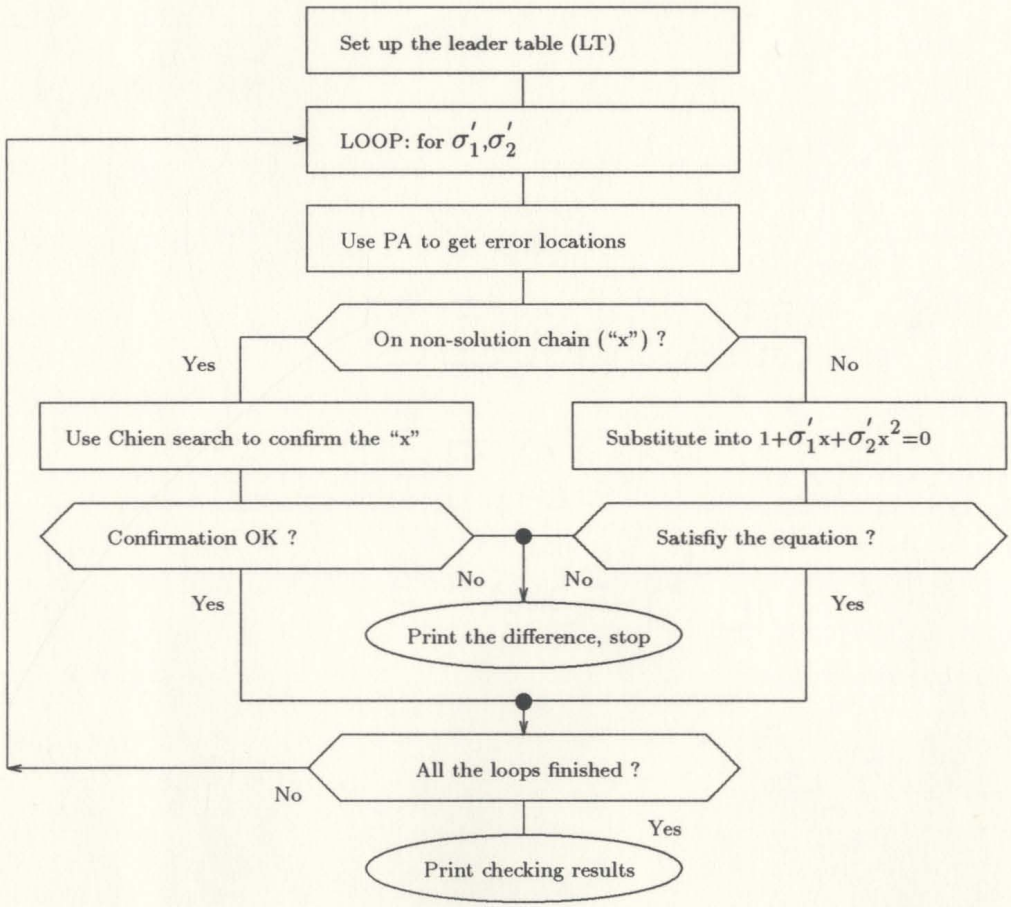

Figure 4.3: Verification of the periodicity algorithm

```
ee
m = 7
Last pair of SIGMA's : 126-126 OK

ee
m = 8
Last pair of SIGMA's : 254-254 OK

ee
m = 9
Last pair of SIGMA's : 510-510 OK

ee
m = 10
Last pair of SIGMA's : 1022-1022 OK
```

Table 4.9: Verification outputs for $7 \leq m \leq 10$

4.4 Discussion and summary

A new algorithm called periodicity algorithm to locate error positions is introduced. This algorithm now can only be used for double error correcting. The periodicity algorithm is based on the fact that the roots of $1 + \sigma_1' x + \sigma_2' x^2 = 0$ are not randomly distributed. All of the possible roots are distributed according to certain patterns. The roots are on the solution chains. The roots have the periodicity of N shifts on the solution chains. Increment is 1 for each shift on the chain. From the leader table, the error location numbers can be calculated with fewer than 8 additions. No multiplication is needed.

Chapter 5

Comparison with other methods

5.1 General discussion

In terms of monetary expenses, microprocessor implementation is cheaper than VLSI, especially for low volume production. Furthermore, the microprocessor implementation is easily programmable. With the development of specialized manufacturing techniques for microprocessors[1], high-speed implementations are also available using assembly language. As long as the microprocessor speed meets the requirements of the applications, microprocessor implementation seems to be the better choice.

In this chapter, the discussion is mainly about the microprocessor implementation of RS error locating. The look-up table, binary decision fast Chien search and Okano's methods will be studied in detail, and an implementa-

[1]For example, 486 CPU can be run at 50 MHz clock [38][39]. TMS320C30 can provide the speed at 50 nS single cycle execution [40].

tion in 486 assembly language of the periodicity algorithm will be described. Comparisons among those methods are also included.

5.1.1 Look-up table method

As mentioned in chapter 3, Chien search needs lots of multiplications and additions to locate all of the errors. Avoiding or reducing the heavy computational load of Chien search has been a hot topic in the RS decoding research community for a long time.

The look-up table method is the easiest way to avoid the computational load. This methods performs Chien search in advance and stores the results in a ROM. Thus the table in the ROM contains the mapping relationship between all the possible values of the coefficients and the corresponding error positions, and the coefficients can be mapped onto the error positions directly. The table forming operations are performed off-line. When locating errors, the coefficients of the error location polynomial will be used as the address to the table in the ROM, and the error positions will be given at the data output of the ROM. The only operation needed is to read the table. This method is the fastest method for RS error locating. It needs only the table (ROM) access delay time to give out the error positions.

However, as the error locator polynomial of a T-error correcting RS code will have N^T possible vectors of the coefficient values, the memory space of such table will be N^T symbols. While N increases, the memory

space requirements increases very fast. For example, if $m = 8$, $T = 2$, then $N = 255$, the memory space of the table is $N^T = 255^2 = 65025$ symbols, or almost 65 Kbytes. For a 2-error correcting RS decoder, such memory space is not affordable. Therefore, even though the look-up table method can provide the fastest speed for RS error locating, it is not feasible for many practical applications, especially for large value of m.

5.1.2 Binary decision fast Chien search

In 1987, Shayan [30] proposed a binary decision approach to fast Chien search. In his approach, a binary table was designed for double error correcting RS codes, in which each address of the table has an one bit value that is either 0 or 1. When the value is 0, Chien search is performed for a specified half of the Galois field, otherwise, Chien search is performed on the other half of the field. As soon as the first error position is found, the second error position can be calculated by the formula $\sigma_2 = x_1 x_2$. Clearly, this approach can save half of Chien search time because Chien search is performed over only half of the Galois field for the first error position. Moreover, there is no search for the second error position. Since Chien search takes so much time and the time is increasing quadratically with N, the time saving can be significant. This approach needs one bit for each address instead of one symbol needed by direct look-up table. However, the space factor of N^2 bits is still large. This issue is a big concern for applications with large N.

Shayan [4] introduced the microprocessor implementation of the binary decision fast Chien search, and implemented the method on Intel 8086/286 as well as TMS320C25 microprocessors. With the binary decision fast Chien search, the time required to decode N symbols was about 17 ms on Intel 8086/286 microprocessor.

5.1.3 Okano's ROM method

Okano and Imai [31] gave a VLSI implementation of decoder using a multi-ROM method. In this method, several ROM's were used for the parallel processing to achieve high speed. The length of the memory is N for each ROM so that the total memory space depends on the number of the ROM's used. Because it uses multi-ROM's to get the parallel processing, this method has relatively high speed. Although Okano and Imai implemented the RS decoders using VLSI, the method is also suitable for microprocessor implementations. Here, the Okano's ROM method is described for the case of double error correcting codes ($T = 2$).

For

$$x^2 + \sigma_{21}x + \sigma_{22} = 0,$$

let $x = \sigma_{21}y$.

Then

$$y^2 + y + C_i = 0 \tag{5.1}$$

is obtained,

where $C_i = \sigma_{22}/\sigma_{21}^2$.

Performing exhaustive calculations on Eq. 5.1 yields all of the possible roots for Y_1 and Y_2. Originally Okano and Imai put the roots into two ROM's which need $2N$ symbols. By using the ROM's, according to a given C_i, two roots Y_1 and Y_2 can be obtained directly, and the real roots are then given by $X_1 = \sigma_{21}Y_1$ and $X_2 = \sigma_{21}Y_2$. Above discussion is based on it that both roots Y_1 and Y_2 can be obtained from the ROM table. For comparison purpose, let us consider Okano's method in another way, which uses only N memory to store all of the possible values for one of the roots, Y_1, and then calculates the corresponding X_1. One can then apply $X_1X_2 = \sigma_2$ to obtain the other root, X_2. For this alternative, only N symbols of memory space will be needed, but it will need one more multiplication.

To be compared with the periodicity algorithm, let us consider the operations required by Okano's method in another way. The Galois field multiplication can be considered as the addition of the power numbers. Based on this, the Okano's method can be expressed as in Fig. 5.1. From the Fig. 5.1, it is clear that Okano's ROM method needs 8 additions and 5 decision operations for the worst case.

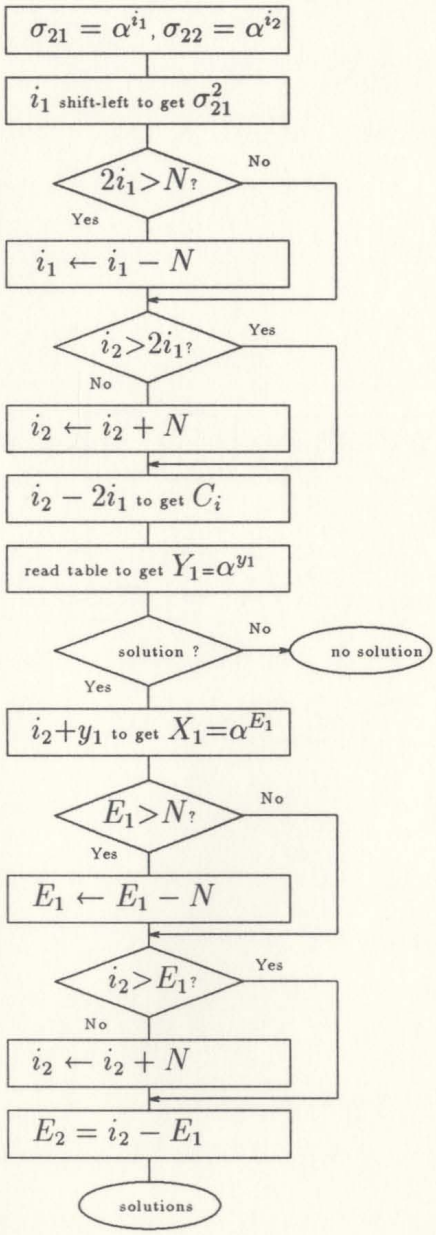

Figure 5.1: Operations needed by Okano's ROM method

5.2 Microprocessor implementation of periodicity algorithm

In this thesis, the periodicity algorithm described in chapter 3, was implemented in assembly language on a 486 PC, though, the given design idea is also valid for other microprocessors. The discussion here will focus on the implementation of the periodicity algorithm itself. Time estimation is made based on the count of the 486 microprocessor cycles. Information about the clock cycles used by the 486 microprocessors is given in [38][39].

Fig. 5.2 gives the flow chart for the implementation. To make the flow chart match with assembly language, corresponding source code is given in each block. Before calling the PA routine shown in Fig. 5.2, some preparations have to be done. They are

1. Off-line leader table (LT) calculation,
2. Store LT into the memory starting at the address in [DI],
3. Put the value of the code length into CX register,
4. Put the value of i_1 into BX register,
5. Put the value of i_1 into AX register.

Those preparations are the tasks of the calling routine. For the PA routine, most operations except table looking-up are performed within the CPU registers to increase running speed. The 486 assembly language source codes of

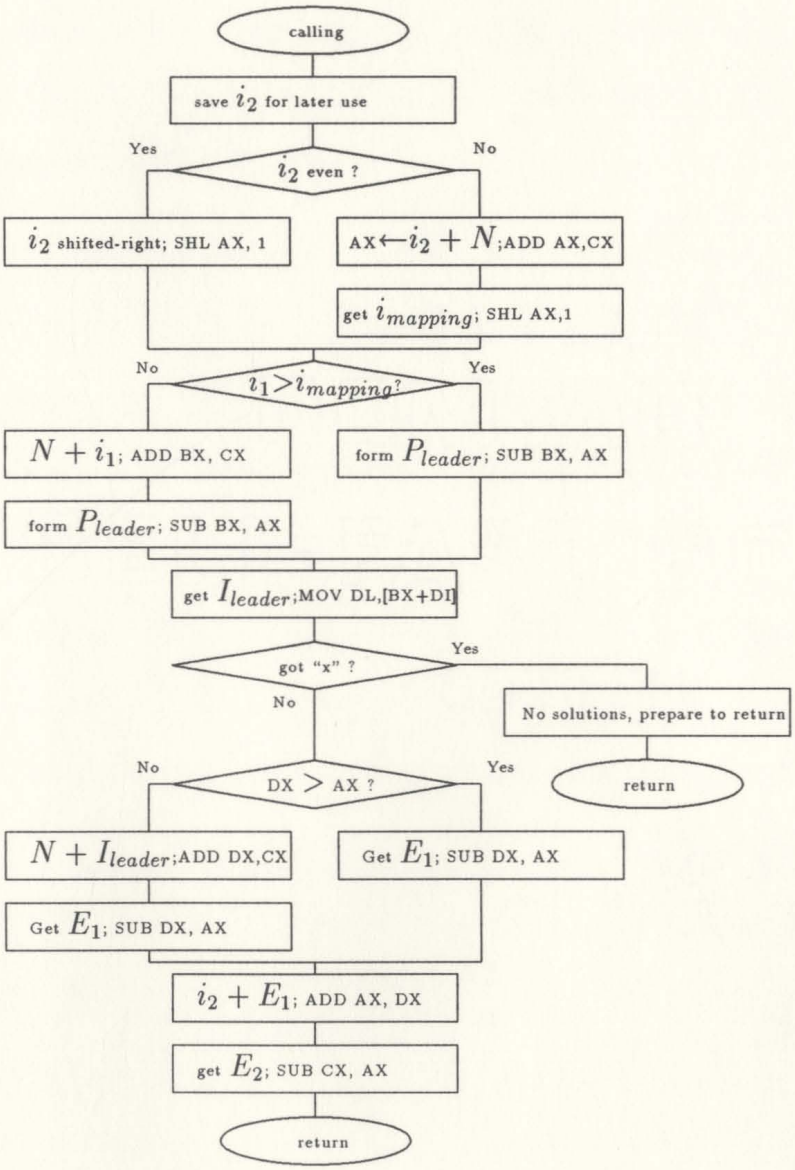

Figure 5.2: Microprocessor implementation of periodicity algorithm

the PA routine are listed in the Table 5.1.

The source codes in Table 5.1 have comments for every line. The basic idea of the assembly routine is described by those comments. Here, some key statements are explained as follows:

- "`PUSH AX`" is to save the value of i_2 in AX register because the value will be used to calculate the second error position E_2.
- "`MOV DI, LT_`" sets the starting point of the leader table. The calling routine has arranged the off-line calculation of the LT. All the values of the leaders are stored sequentially after the starting point.
- "`MOV DX, 1`" and "`AND DX, AX`" are to test whether i_2 is even or not.
- "`SHR AX, 1`" is to perform dividing by 2. This operation has $i_2 \leftarrow i_2/2$. At this point, AX register contains the value of $i_{mapping}$.
- The preparation procedure sets the value of i_1 into BX register. "`CMP BX, AX`" performs the comparison between i_1 and $i_{mapping}$. "`JG`" gives branching.
- The table reading operation is "`MOV DL, [BX+DI]`". DI register contains the starting pointer of LT (base pointer). The content of the BX register is used as the offset of the pointer. Then "`[BX+DI]`" gives the combined pointer to the value of the leader in LT.

```
        PUSH  AX           ; save AX (i2) for later use
        MOV   DI, LT_      ; set LT starting point
        MOV   DX, 1        ; prepare to test even or not
        AND   DX, AX       ; if even, last bit is 0
        JZ    EVEN         ; whether even ?
        ADD   AX, CX       ; (i2) + n
EVEN:   SHR   AX, 1        ; i2 shifted right
        CMP   BX, AX       ; (i1) > (i mapping) ?
        JG    GREAT1       ;
        ADD   BX, CX       ; (i1) + n
GREAT1: SUB   BX, AX       ; (i1) - (i mapping)
        MOV   DL, [BX+DI]  ; table reading
        CMP   DX, 0FFH     ; "x" got ?
        JNE   OK           ; if not, then jump to "OK"
        MOV   AX, 0        ; if yes, return. "x" is in DX
        RET                ; return with no solution
OK:     CMP   DX, AX       ; (i leader) > (i mapping) ?
        JG    GREAT2       ;
        ADD   DX, CX       ; n + (i leader)
GREAT2: SUB   DX, AX       ; (i leader) - (i mapping)
        POP   AX           ; restore (i2) into AX
        ADD   AX, DX       ; (i2) + E1
        CMP   CX, AX       ; add n ? or not ?
        JG    NO_ADD       ;
        SHL   CL, 1        ; n + n
NO_ADD: SUB   CX, AX       ; n - ( (i2) + E1 ), E2 is in CX
        MOV   AX, 0        ; ready to return
        RET                ; return with solutions
```

Table 5.1: Source codes of the PA implementation

- The value of the leader is moved into DX register. Some of the leaders represent "no solution". In the PA routine, such cases are marked by the value of "0FFH". If the hexadecimal integer "0FFH" is given, it means that a non-solution chain is met. Operation "CMP DX, 0FFH" tests whether a non-solution chain is met or not. If yes, the hexadecimal value 0FFH is in DX register to represent no solution. If not, the statement labeled "OK" will be executed.

- The "POP AX" operation restores the value of i_2 into AX register. The PUSH and POP must be paired.

When the PA routine returns, the values of E_1 and E_2 are in DX and CX registers.

5.3 Time estimation

Table 5.2 shows the clock cycles needed by each operation. The time parameters are given in [41]. Note that Table. 5.2 is same as Table. 5.1 except the comments are replaced by the timing estimates for each operation. Considering the worst case during the execution, the time of those statements marked by (*) should be accumulated. That result will be the worst case time, that is, 72 clock cycles. This has included the address calculations. However, if the code fetch is considered, more cycles have to be used. The total cycles will be less than 100 cycles. The Intel 486 can work at 50 MHz

```
        PUSH  AX           ; 2 clock cycles (*)
        MOV   DI, LI_      ; 2 clock cycles (*)
        MOV   DX, 1        ; 2 clock cycles (*)
        AND   DX, AX       ; 2 clock cycles (*)
        JZ    EVEN         ; 3 for no jump, 7 for jump (*)
        ADD   AX, CX       ; 2 clock cycles
EVEN:   SHR   AX, 1        ; 3 clock cycles (*)
        CMP   BX, AX       ; 2 clock cycles (*)
        JG    GREAT1       ; 3 for no jump, 7 for jump (*)
        ADD   BX, CX       ; 2 clock cycles
GREAT1: SUB   BX, AX       ; 2 clock cycles (*)
        MOV   DL, [BX+DI]  ; 4 clock cycles (*)
        CMP   DX, 0FFH     ; 2 clock cycles (*)
        JNE   OK           ; 3 for no jump, 7 for jump (*)
        MOV   AX, 0        ; 2 clock cycles
        RET                ; 10 clock cycles
OK:     CMP   DX, AX       ; 2 clock cycles (*)
        JG    GREAT2       ; 3 for no jump, 7 for jump (*)
        ADD   DX, CX       ; 2 clock cycles
GREAT2: SUB   DX, AX       ; 2 clock cycles (*)
        POP   AX           ; 4 clock cycles (*)
        ADD   AX, DX       ; 2 clock cycles (*)
        CMP   CX, AX       ; 2 clock cycles (*)
        JG    NO_ADD       ; 3 for no jump, 7 for jump (*)
        SHL   CX, 1        ; 3 clock cycles
NO_ADD: SUB   CX, AX       ; 2 clock cycles (*)
        MOV   AX, 0        ; 2 clock cycles (*)
        RET                ; 10 clock cycles (*)
```

Table 5.2: Time estimation of the PA assembly routine

clock [38]. Hence, if the PA routine is run on Intel 486 CPU, the time needed will be about 2 μs, yielding a throughput of double error locating of 0.5 million/s. For a general purpose microprocessor, this throughput is rather fast.

An estimation of the error locating time using PA and Chien search for Eq 4.2 is shown in Fig. 5.3 and Fig. 5.4 for $m = 3, 4, 5, 6, 7$ and 8. Note that Fig. 5.4 is in expanded scale. It is noted that the expanded scale shown in Fig. 5.4 is the segment of 0 to 1ms on Fig. 5.3. On this expanded scale chart, the relationship between the curves can be seen more clearly.

The above running time test is done on DEC 3100 workstation, with ULTRIX operating system V4.0 (Rev. 179) and C language. By using timing profiles provided by the system command "profile", the factor of time-sharing can be eliminated, and the exact CPU time used by each function can be determined. To be more precise, 100,000 loops were arranged for each test. A sample output of profile (for Chien search, $m = 7$) is listed in Table 5.3.

It is clear that as m increases, Chien search time increases quadratically with N while the PA time remains almost constant. From those two figures, it can be estimated that for $m = 8$, Chien search might need 800 times more time than that used by the periodicity algorithm.

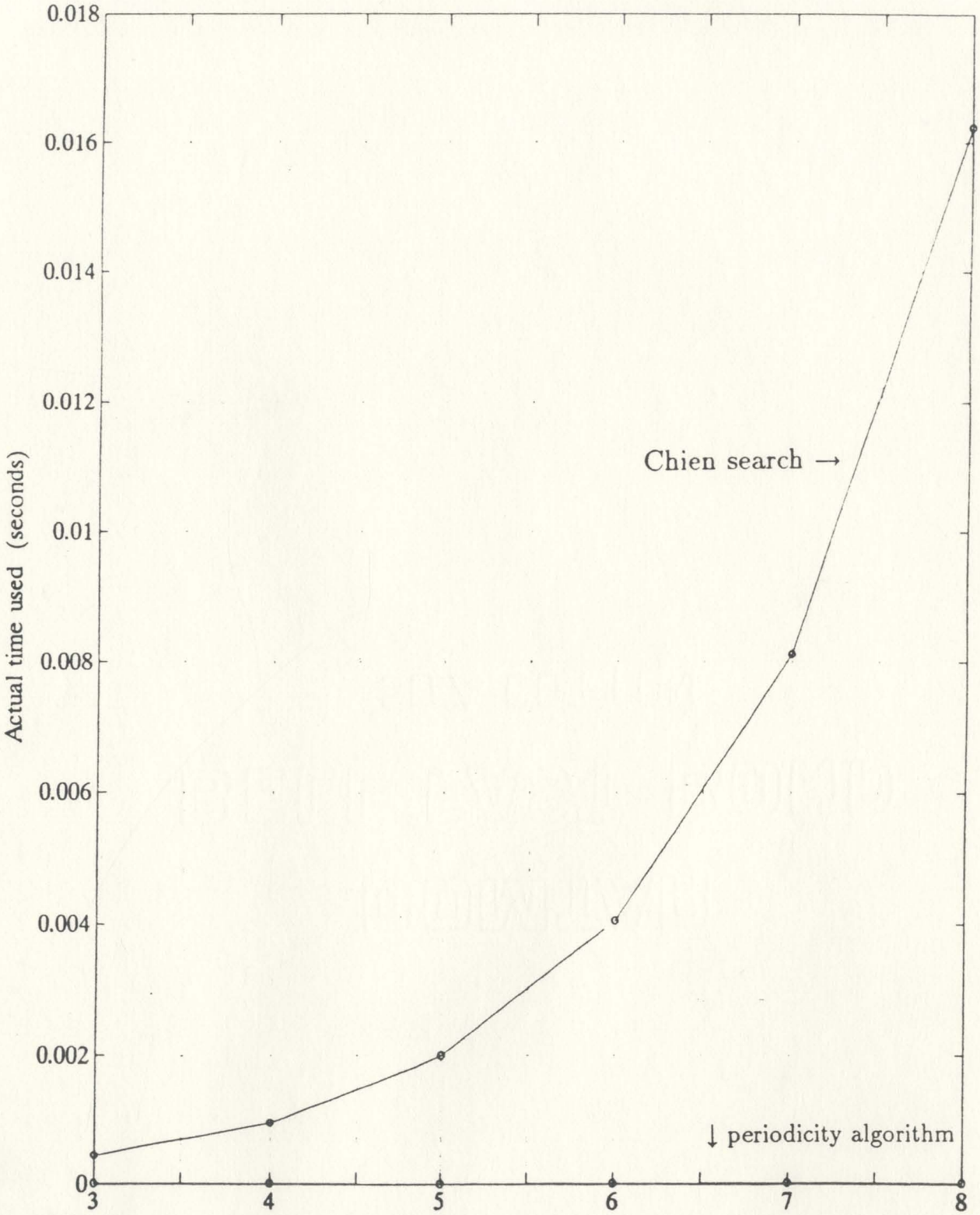

Figure 5.3: Time comparison between Chien search and PA

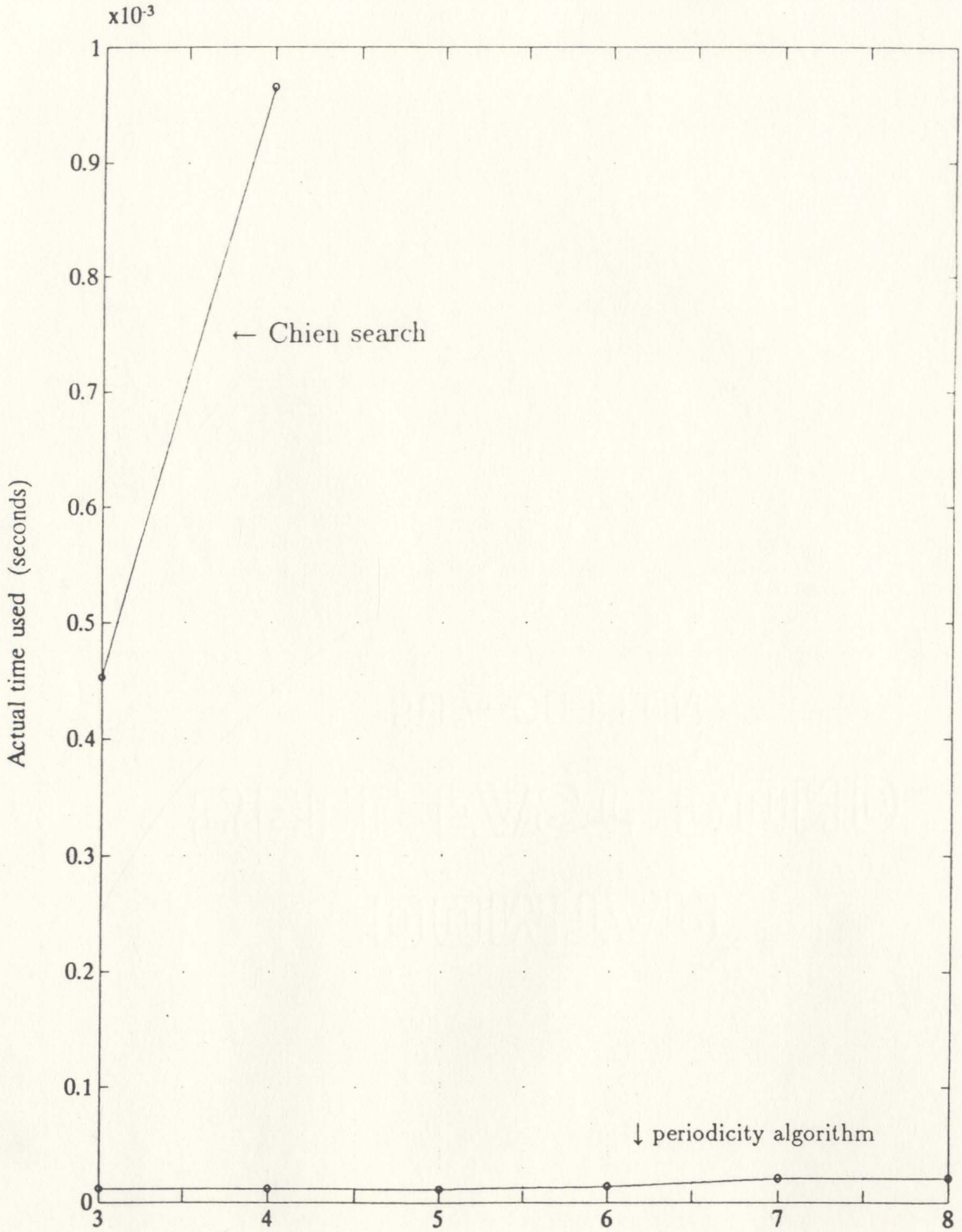

Figure 5.4: Time comparison between Chien search and PA (expanded scale)

```
File Name: TC7.time

testing the exact time used by Chien search method for m = 7.
( 100,000 loops set for the test )
-------------------------------------------------------------------
Profile listing generated Sat Jul 31 14:29:56 1993 with:
 prof tc mon.out

-------------------------------------------------------------------
 -p[rocedures] using pc-sampling;
 sorted in descending order by total time spent in each procedure;
 unexecuted procedures excluded
-------------------------------------------------------------------

Each sample covers 8.00 byte(s) for 0.0012% of 811.9500 seconds

%time     seconds  cum %   cum sec  procedure (file)

 46.2    375.2900   46.2    375.29 chien_p (chien_p.c)
 41.0    332.6900   87.2    707.98 gf_multp_1 (add.c)
 12.8    103.7900  100.0    811.77 gf_add (add.c)
  0.0      0.1700  100.0    811.94 main (tc.c)
  0.0      0.0100  100.0    811.95 write (../write.s)
```

Table 5.3: Output of profile for measuring time

5.4 Comparisons among discussed methods

Since the periodicity algorithm can only be used for double error correcting ($T = 2$), for convenience, the comparisons are made only for the discussed methods on case of $T = 2$.

1. Chien search needs lots of multiplications and additions, but no memory is needed. The disadvantage is that the operations will quadratically increase when N increases.

2. Look-up table method needs N^2 memory space, but no operation is needed. This is the fastest method. The time delay is only the ROM access time. However, when N or T increases, the memory space will terribly increase. This is the big concern of the feasibility of the method.

3. Okano's ROM method needs N memory space to store intermediate roots Y_1 or Y_2. To obtain the desired roots X_1 and X_2, it needs eight additions and five decision operations.

4. The periodicity algorithm needs N memory space to store the leaders of the chains. With fewer than eight additions and four decision operations, two error location numbers can be calculated.

Upon above discussion, it can be concluded that for the cases of double error correcting, the periodicity algorithm needs much fewer operations than

needed by Chien search ($N^2 : 8$), and much less memory space than needed by look-up table and the binary decision fast Chien search ($N^2 : N$). Compared with Okano's method, the periodicity algorithm needs same memory as Okano's method. However, Okano's method needs one more decision operation than the periodicity algorithm. Therefore, the periodicity algorithm is the best one among those methods.

5.5 Summary

Several schemes for error locating are compared in this chapter. The periodicity algorithm is implemented on a 486 PC. The executing time is estimated. Based on the time curves for Chien search and the periodicity algorithm, it is concluded that the time required by the periodicity algorithm is much shorter than the time needed by Chien search. If the case of $m = 8$ is used as the example, the periodicity algorithm needs 1800 of the time used by Chien search. According to the comparisons, the periodicity algorithm is the best one among those methods.

Chapter 6

Conclusion and Future Work

This thesis presents an RS code simulator which is implemented in C language under UNIX operating system. To find an optimal RS code for a certain application, the user can select:

1. Peterson's or Berlekamp's method for decoding,
2. the symbol length (m bits) where $3 \leq m \leq 8$, and
3. the correcting capability (T symbols), where $1 \leq T \leq 16$.

The simulator starts encoding the user's message into a systematic codeword. An error pattern of arbitrary weight (ν) randomly chosen by the user is added to the codeword and the resulted received word is formed. The simulator decodes the received word by using Peterson's or Berlekamp's method according to the user selection, to construct the error locator polynomial. The Chien search technique is used to obtain the error location numbers

which are the roots of the error locator polynomial. The simulator produces a decoded word which is subject to:

1. if $\nu \leq T$, the decoded word is the codeword when it is divisible by the generator polynomial $G(x)$, and

2. if $\nu > T$, the decoded word may not be the codeword even it is divisible by $G(x)$.

This can be explained by the fact that when ν is exactly equal to T, the decoded word is the codeword. However, when $\nu > T$ the error pattern may be a codeword, then, the decoded word is divisible by $G(x)$. But it is not a codeword. Furthermore, when $\nu > T$, the decoded word always contains more errors than the actual error pattern because the decoder makes more errors. This new error pattern may sometimes be a codeword. Hence, the decoded word is still divisible by $G(x)$ but it is not the codeword.

By using Chien search to obtain all the possible roots of the error locator polynomial, it is found that for the double error correcting case ($T = 2$), those roots are not randomly distributed, but they follow certain rules. Based on these rules, the periodicity algorithm is introduced and its validity is verified by exhaustive computer simulations.

The periodicity algorithm states:

1. All the possible roots of the double error locator polynomial always exists on solution chains.

2. Each solution chain repeats itself in a period of N shifts.

3. On each solution chain, the error position numbers are changed by an increment or decrement of 1, depending on the row position index is increased or decreased by 1, respectively.

4. In regard to the second root, the constant product property is used.

With fewer than 8 additions, 4 decision operations and only N symbols of memory space required, the periodicity algorithm outperforms Chien search and the binary decision fast Chien search techniques in terms of the decoding time. Most of all it also outperforms the Okano's analytical solutions by a decision operation. Of course, the look-up table is the fastest in terms of the decoding time but its memory space required is N^2. This will limit its use when N is large. Therefore, it is concluded that the periodicity algorithm is the optimal solution for both decoding time and memory space. This algorithm is found to be very suitable for use in microprocessor based decoders.

The periodicity algorithm is now available for $T \leq 2$. For the cases of $T > 2$, it may be anticipated that the error location numbers may not be distributed randomly. This may be the future work one may want to look into.

Bibliography

[1] Hamming, R.W., "Error detecting and error correcting codes", *Bell System Tech. J.*, 29, pp.147-160 (1950).

[2] Reed, I.S., and Solomon, G., "Polynomial codes over certain finite fields", *J. Soc. Ind. Appl. Math.*, 8, pp.300-304 (1960). Reprinted in [3].

[3] Blake, I.F., "Algebraic coding theory: history and development", Dowden, Hutchingon & Ross Inc., (1973).

[4] Shayan, Y.R., Le-Ngoc, T., "Design of Reed-Solomon (16,12) CODEC for North American advanced train control system", *IEEE Trans. Vehicular Tech.*, vol.39, no.4, pp.400-409, Nov. 1990.

[5] Le-Ngoc, S., Le-Ngoc, T. and Bhargawa, V.K., "Design aspects and performance evaluation of acts mobile data link", *IEEE Trans. Consumer Electronics*, vol.38, pp.842-849, Nov. 1992.

[6] Le-Ngoc, S., "Design of an adaptive coding and retransmission strategy for a multipath fading channel", presented at 40th Anniversary of Vehic-

ular Technology Conference, May 6-9, 1990, Florida, USA. The revised version will appear on *IEEE Journal of Ocean Engineering.*

[7] Whitaker, S.R., Canaris, J.A., "Reed Solomon VLSI Codec for advanced television", *IEEE Trans. Circuits and Systems for Video Tech.*, vol.1, no.2, pp.230-236, June 1991.

[8] Hocquenghem, A., "Code correcteurs d'erreurs", *Chiffres*, 2, pp.147-156 (1959). Reprinted in [3].

[9] Bose, R.C., and Ray-Chaudhuri, D.K. "On a class of error correcting binary group codes", *Inform. Contr.*, 3, pp.68-79 (1960). Reprinted in [3].

[10] Bose, R.C., and Ray-Chaudhuri, D.K. "Further results on error correcting binary group codes", *Inform. Contr.*, 3, pp.279-290 (1960). Reprinted in [3].

[11] Blahut, R.E., "Theory and practice of error control codes", Addison-Wesley publishing comp., 1st Ed., p.210, 1983.

[12] Peterson, W.W., "Encoding and error correction procedures for the Bose-Chaudhuri codes", *IRE Trans. Inf. Theory*, vol.IT-6, pp.459-470, Sept. 1960. Reprinted in [3].

[13] Berlekamp, E.R., "On decoding binary Bose-Chaudhuri-Hocquenghem codes", *IEEE Trans. Inf. Theory*, IT-11, pp.577-580, Oct. 1965.

[14] Michelson, A.M. and Levesque, A.H., "Error-control techniques for digital communication", John Wiley & Sons Inc., 1985.

[15] Mandelbaum, D., "On decoding of Reed-Solomon codes", *IEEE Trans. Inform. Theory*, IT-17, pp.707-712, 1971.

[16] Blahut, R.E., "Transform techniques for error-control codes", *IBM J. Res. Develop.* , vol.23, pp.299-315, 1979.

[17] Rader, C.M., "Discrete convolution via mersenne transforms", *IEEE Trans. Comput.*, vol.C-21, pp.1269-1273, Dec. 1972.

[18] Gore, W.C., "Transmitting binary symbols with Reed-Solomon code", Johns-Hopkins, EE report, No.72-5, April 1973.

[19] Michelson, A.A., "A new decoder for the Reed-Solomon codes using a fast transform technique", *Systems Engineering Technical Memorandum No.52*, Electronic Systems Group, Eastern Division GTE Sylvania, Waltham, MA, Aug. 1975.

[20] Michelson, A.A., "A fast transform in some Galois fields and an application to decoding Reed-Solomon codes", *IEEE Abstr. of Papers: IEEE International Symposium on Information Theory*, Ronneby, Sweden, 1976.

[21] Reed, I.S., *et al*, "The fast decoding of Reed-Solomon codes using Fermat theoretic transforms and continued fractions", *IEEE Trans. Inform. Theory*, vol.IT-24, no.1, pp.100-106, Jan. 1978.

[22] Miller, R.L., Truong, T.K., *et al*, "Efficient program for decoding the (255,233) Reed-Solomon code over $GF(2^8)$ with both errors and erasures, using transform decoding", *IEE Proc.*, vol.127, Pt.E, no.4, pp.136-142, July 1980.

[23] Berlekamp, E.R. "Bit-serial Reed-Solomon encoders" *IEEE Trans. Inform. Theory*, vol.28, no.6, pp.869-874, Nov. 1982.

[24] Hsu, I.S., "The VLSI implementation of a Reed-Solomon encoder using Berlekamp's bit-serial multiplier algorithm" *IEEE Trans. Compt.*, vol.33, no.10, Oct. 1984.

[25] Seroussi, G., "A systolic Reed-Solomon encoder", *IEEE Trans. Inform. Theory*, vol.37, no.4, p.1217-1220, July 1991.

[26] Truong, T.K., "Fast technique for computing syndromes of BCH and Reed-Solomon codes", *Electronics Letters*, vol.15, no.22, pp.720-721, Oct. 1979.

[27] Cooper, A.B., "Direct solution of BCH decoding equations", in Arikan,E. (ed.):"*Communication, control and signal processing*", pp.281-286, (Elsevier, Amsterdam, 1990).

[28] Cooper, A.B., "Finding BCH error locator polynomials in one step", *Electronics Letters*, vol.27, no.22, pp.2090-2091, Oct. 1991.

[29] Chien, R.T., "Cyclic decoding procedure for the Bose-Chaudhuri-Hocqwenghem codes", *IEEE Trans. Inf. Theory*, vol.IT-10, pp.357-363,October 1964.

[30] Shayan, Y.R. and Le-Ngoc, T., "Binary decision approach to fast Chien search for software decoding of BCH codes", *IEE Proc.*, vol.134, no.6, Oct. 1987.

[31] Okano, H., Imai. H., "A construction method of high-speed decoders using ROM's for Bose-Chaudhuri-Hocquenghem and Reed-Solomon codes", *IEEE Trans. Comput.*, Vol.36, No.10, pp.1165-1171, (Oct. 1987).

[32] Blahut, R.E., "Transform decoding without transforms", *presented at the Tenth IEEE Communication Theory Workshop.* Cypress Gardens, FL, 1980.

[33] Blahut, R.E., "A universal Reed-Solomon decoder", *IBM J. Res. Develop.*, vol.28, no.2, pp.150-158, March 1984.

[34] Sorger, U.K., " A new Reed-Solomon code decoding algorithm based on Newton's interpolation", *IEEE Trans. Inform. Theory*, vol.IT-39, no.2, pp.358-365, (Mar. 1993).

[35] Lin, S. and Costello, D.J., "Error control coding: fundamentals and applications", Prentice Hall, New Jersey, USA (1983).

[36] Le-Ngoc, S., "Information theory and coding", Lecture notes, Faculty of Engineering, MUN, 1992.

[37] Peterson, W.W., Weldon, E.J., "Error-correcting codes", 2nd Ed., MIT Press, Cambridge, Mass., USA, 1970.

[38] Intel Corporation, "Intel Microprocessors", Volumn 1, 1992.

[39] Intel Corporation, "Intel Microprocessors", Volumn 2, 1992.

[40] Texas Instruments Incorporation, "TMS320C3x User's Guide", 1992.

[41] Microsoft Corperation, "Microsoft Macro Assembler 5.1 Reference", 1989.