A GENERAL PURPOSE REED-SOLOMON CODEC SIMULATOR AND NEW PERIODICITY ALGORITHM

YING YE

A General Purpose Reed-Solomon CODEC Simulator and New Periodicity Algorithm

By

© Ying Ye

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

July, 1995

St. John's Newfoundland Canada

Abstract

In most digital communication systems, if we can afford to send data below the modem transmission speed, then it is possible to achieve the system bit error rate as small as we desire by using error control codes. The Reed-Solomon codes are such error control codes, that are widely used for forward error correction due to their optimal characteristics in both Hamming distance and structure, but most of all, their capacity for correcting both random and burst errors.

Finding a suitable code for a communication channel, or trying to explain how the Reed-Solomon codes work, or comparing various decoding methods is not an easy task, hence this thesis developed a general purpose Reed-Solomon (RS) coding and decoding (codec) simulator for teaching as well as research purposes.

The RS codec simulator has two versions that can be run under Microsoft Windows and Unix operating system, respectively. A friendly and easy-to-use graphical user interface (GUI) is provided for PC. The user can define a code by selecting the symbol length m from 3 to 8 bits and the error correcting capability T of up to 20.

In the encoder, the systematic code generation and the self-reciprocal generator polynomial are used. The noisy channel can be modeled by an error pattern. This error pattern can either be entered by the user with the arbitrary weight or generated by an external program, which generates all possible error positions. In the decoding process, both the Peterson and Berlekamp-Massey algorithms are available for finding the error locator polynomial. The simulation results show that the Peterson's direct method is better when $T \leq 6$. However, the Berlekamp-Massey

algorithm is much faster when $T > 6$. Chien search is used for locating the error position in the received word. Although the error values can be obtained by using either Gauss-Jordan elimination or Forney's algorithm, Gauss-Jordan elimination is preferred when T is small, i.e. $T \leq 10$, but as T increased, i.e. $T > 10$, the Forney algorithm should be used in order to minimize decoding time.

It is found that the periodicity algorithm conceived by S.Le-Ngoc and Z.Young [1] [2] is a special case of the LeNgoc-Ye Transformation Algorithm [3]. An improved periodicity algorithm is proposed which can eliminate the division operation required by the LeNgoc-Young algorithm. The analysis shows that the periodicity algorithm is valid for all values of m. Furthermore, a new periodicity algorithm is also developed by using direct solution method to eliminate the index table required in the proposed periodicity algorithm. It is shown that the new periodicity algorithm outperforms the look-up table, Chien search, binary decision (fast Chien search) and Okano-Imai algorithms in terms of optimization of both memory space and decoding time. The new periodicity algorithm has a simple structure and therefore it is well suited for VLSI implementation.

Acknowledgements

I would like to express my deepest gratitude to my supervisor Dr. Son Le-Ngoc. Without his stimulating my interest in this exciting field and his constant encouragement and guidance throughout the course of this research, this thesis would not have been possible. I also wish to thank the examiners for their careful reading of the thesis and for their useful comments that led to the improvement of the thesis.

The financial support by Natural Sciences and Engineering Research Council of Canada, the Faculty of Engineering and Applied Science of Memorial University, and Wilson Technologies Inc. is gratefully acknowledged.

The last but not the least, I would also like to thank my parents Mr. Yulin Ye and Ms. Bingqing Chen for a joyful dawning in the quest for truth and knowledge.

Contents

List of Figures

List of Tables

List of Principal Symbols and Abbreviations

$A(x)$	Parity check polynomial
BiCMOS	Bipolar complimentary metal oxide semiconductor
$\overline{c}$	codeword vector
CCAE	Center for Computer Aided Engineering
C_j	jth Fourier transform coefficient
$C(x)$	codeword polynomial
$\hat{C}(x)$	Estimated codeword polynomial
Codec	Coding and decoding
$E(x)$	Error pattern polynomial
F	Fourier transform
FEC	Forward Error Correction
GF	Galois field
GUI	Graphic User Interface
$G(x)$	Generator polynomial
K	Number of information symbols
LSI	Large Scale Integrate
m	Symbol length of a Galois field element
$\overline{m}$	Information or message vector
$M(x)$	Message polynomial
MS	Microsoft
N	Code length

$p(x)$	Primitive polynomial
$Q(x)$	Quotient polynomial
ROM	Read-only memory
RS	Reed-Solomon code
$R(x)$	Received word polynomial
$S(x)$	Syndrome polynomal
S_j	Syndrome
T	Error correction capability
X_i	Error locator
Y_i	Error value
VLSI	Very large scale integrated
α	Primitive element of $GF(2^m)$
$\Lambda(x)$	Error locator polynomial
ν	Actual error number
$\Omega(x)$	Error evaluator polynomial
$\Sigma(x)$	Error locator polynomial

Chapter 1

Introduction

1.1 Statement of the Problem

Reed-Solomon codes are extremely powerful codes that play a major role in error control codes. This claim is illustrated by the following important applications of Reed-Solomon codes [4]:

The digital audio disc, or compact disc uses Reed-Solomon codes for error correction and error concealment to improve the signal-to-noise ratio at the output exceeding 90dB, thus assuring the high-fidelity sound quality of the compact disc.

On the Voyager space craft, Reed-Solomon and convolutional codes were used in concatenated systems, hence enormous coding gains were achieved. They were responsible for sending clear pictures from the deep space planet back to earth.

Reed-Solomon codes are used in systems with feedback such as mobile data transmission systems [5][6] and high-reliability military communication systems that allow the transmission of information from the receiver back to the transmitter.

Reed-Solomon codes are also used in spread-spectrum systems such as frequency-hopping spread spectrum (FH/SS) and direct-sequence spread spectrum (DS/SS) [7].

Codes based on Reed-Solomon codes are developed to control data flow in computers [8].

Since every particular application has its own distinct requirements such as the error correction capability and the codeword length, a general purpose Reed-Solomon codec simulator should be introduced to help the designer to evaluate the performance of various RS codes and choose the most efficient RS code for a particular application. Such a codec simulator will allow us to compare various encoding and decoding algorithms of RS codes and to investigate different properties of RS codes and thus lead to development of new encoding and decoding algorithms. Another motivation of this research is to demonstrate RS code encoding and decoding principles in the classrooms as well as in the laboratories.

Chien search is normally used for obtaining the error location numbers for syndrome based Reed-Solomon decoding. This method is the most time consuming process in the decoding procedure since it may be necessary, in the worst case, to search the entire Galois field. Therefore, it is also necessary to search for an efficient algorithm to substitute the Chien search method.

The motivations for this thesis research are therefore three-fold: 1) to design and develop a simulator to aid the designer, 2) to allow the simulator also to demonstrate RS encoding and decoding principles in the classroom as well as in the laboratories, and 3) to develop new algorithm to replace the Chien search method.

1.2 Literature Review

In March and September of 1960, Bose and Ray-Chaudhuri proposed a class of error correcting binary codes [9][10], which are now called BCH codes. The "H" in BCH is for Hocquenghem, whose 1959 paper presented independent work that included a description of BCH codes as a "generalization of Hamming's work" [11].

Reed-Solomon codes first appeared to the outside world in June 1960, in a paper entitled "Polynomial Codes over Certain Finite Fields," [12] in *JSIAM* (Journal of the Society for Industrial and Applied Mathematics). They are an important subclass of non-binary BCH codes [13]. The codes are optimal in the sense that it is impossible for any linear codes with the same length to have a Hamming distance greater than that of the Reed-Solomon codes. For decades since their discovery, Reed-Solomon codes have found countless applications, from compact disc players to deep space telecommunications.

1.2.1 Three Approaches to Constructing Reed-Solomon Codes

There are three approaches to constructing Reed-Solomon codes. The first is the original approach by Reed and Solomon [12][14]. Suppose that we have a packet of K information symbols, $m_0, m_1, \cdots, m_{K-1}$, taken from the finite field $GF(q)$.[1] These symbols can be used to construct a polynomial $P(x) = m_0 + m_1 x + \cdots + m_{K-1}x^{K-1}$. A Reed-Solomon codeword $\bar{c}$ is formed by evaluating $P(x)$ at each of

[1] Reed-Solomon codes are constructed and decoded through the use of finite field arithmetic.

the q elements in the finite field $GF(q)$.

$$\bar{c} = (c_0, c_1, \cdots, c_{q-1}) = [P(0), P(\alpha), \cdots, P(\alpha^{q-1})] \tag{1.1}$$

It was felt for quite a long time that Reed and Solomon's original approach failed to lead to efficient decoding algorithms. In 1982, Tsfasman, Vladut, and Zink, using a technique developed by Goppa, extended Reed and Solomon's construction to develop a class of codes whose performance exceeded the Gilbert-Varshamov bound [15]. The Gilbert-Varshamov bound is a lower bound on the performance of error-correcting codes that, many were beginning to believe, was also an upper bound. Tsfasman, Vladut, and Zink's work broke open an entirely new field of research that continues to attract great interest from coding theorists.

The generator polynomial construction for Reed-Solomon codes is the approach most commonly used today in the error control literature. This approach initially evolved independently from Reed-Solomon codes as a means for describing cyclic codes, which had led to the discovery of BCH codes. It was Gorenstein and Zierler who generalized Bose and Ray-Chaudhuri's work to arbitrary Galois fields of size p^m and developed a new means for describing Reed and Solomon's "polynomial codes" [16].

Cyclic Reed-Solomon codes with codeword symbols from the finite field $GF(q)$ usually have length $q-1$. The cyclic Reed-Solomon codes design criterion is as follows:

The generator polynomial for a T-error-correcting code must have as roots $2T$ consecutive powers of α, where $2T < q-1$ and α is a primitive element in $GF(q)$.

$$G(x) = \sum_{j=1}^{2T} (x - \alpha^j) \tag{1.2}$$

Any valid code polynomial must be a multiple of the generator polynomial. It follows that any valid polynomial must have as roots the same $2T$ consecutive powers of α that form the roots of $G(x)$. This provides us with a very convenient means to determine whether a received word is a valid codeword. The generator polynomial approach leads to a powerful and efficient set of decoding algorithms which are introduced later in Section 1.2.2, and discussed in detail in Chapters 2 and 3.

The third approach to Reed-Solomon codes uses the Fourier transforms technique to achieve the encoding and decoding process. Let α again be a primitive element in the Galois field $GF(q)$. The Galois field Fourier transform (GFFT) of an N-bit vector $\bar{c} = (c_0, c_1, \cdots, c_{N-1})$ is defined as follows:

$$F(c_0, c_1, \cdots, c_{N-1}) = (C_0, C_1, \cdots, C_{N-1}), \tag{1.3}$$

where $C_j = \sum_{i=0}^{N-1} c_i \alpha^{ij}, j = 0, 1, \cdots, N-1$ and $N < q$.

Unlike the conventional analysis of signals in a communication system, it is not entirely clear what is meant by the terms "time domain" and "frequency domain" when we are working with coordinate values from finite fields. 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. It can be shown that the following two conditions in the time-domain and the frequency-domain, respectively, are equivalent.

A word polynomial has $2T$ consecutive powers of α as roots if and only if the spectrum of the corresponding word has $2T$ consecutive zero coordinates.

The GFFT approach is a dual to the generator polynomial approach. The

transform relationship leads to a series of efficient encoders and decoders. The pioneering work on transform techniques can be attributed to Blahut[17].

1.2.2 Decoding Reed-Solomon Codes

In 1960 Peterson provided the first explicit description of a decoding algorithm for binary BCH codes [18]. Peterson introduced an algebraic decoding algorithm relying on the transformation of power sum symmetric functions (the syndromes) into elementary symmetric functions. This leads to a matrix equation relating the syndromes to the coefficients of an "error locator polynomial" whose roots specify the locations of erroneous coordinates in a received word. Peterson's "direct solution" algorithm is quite useful for correcting small numbers of errors but becomes computationally intractable as the number of errors increases. Peterson also redefined Reed-Solomon codes in a cyclic context to complement his work in algebraic decoding [19]. A lot of coders and their codes owe their names and fames to Peterson. An evaluation of Peterson's contribution is given in [20]. Peterson's algorithm was improved and extended to nonbinary codes by Gorenstein and Zierler (1961) [16], Chien (1964) [21], and Forney (1965) [22].

The Peterson technique of using matrix inversion to find the coefficients of the error locator polynomials was far too complicated for the decoding of large numbers of errors. In 1967, Berlekamp demonstrated his powerful iterative algorithm for decoding both nonbinary BCH and Reed-Solomon codes [23][24]. In 1969 Massey showed that Berlekamp's algorithm is equivalent to the method of synthesizing the shortest linear feedback shift register capable of generating a given sequence [25]. This shift register-based decoding approach is now commonly referred to as

the Berlekamp-Massey algorithm. Michelson and Levesque [26] pointed out that the Berlekamp-Massey 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.

In 1975 four Japanese mathematicians Sugiyama, Kasahara, Hirasawa, and Namekawa showed that Euclid's algorithm can also be used to efficiently decode BCH and Reed-Solomon codes [27]. Euclid's algorithm is a method for finding the greatest common divisor (gcd) of two polynomials. Euclid's algorithm is well suited for VLSI implementation because of its modularity. The operations needed to compute the Euclidean algorithm generally require the computation of inverse elements in the finite field. A modified Euclidean algorithm [28] can avoid the computation of inverse elements and it is very similar to the Berlekamp-Massey algorithm.

1.2.3 Syndrome-Based Decoding vs Remainder-Based Decoding

The BCH and RS decoding methods can be divided into two categories: syndrome-based decoding and remainder-based decoding. Both algebraic decoding and transform decoding belong to syndrome-based decoding.

In algebraic decoding, syndromes are evaluated using Equation (1.4):

$$S_j = R(\alpha^j) = \sum_{i=0}^{N-1} r_i \alpha^{ij}, \quad j = 1, 2, \cdots, 2T \tag{1.4}$$

where $R(x) = r_0 + r_1 x + \cdots + r_{N-1} x^{N-1}$ is referred to the received polynomial. Then, based on the syndromes, the error locator polynomial $\Lambda(x)$ is found by using one of the Peterson, Berlekamp-Massey, or Euclidean algorithms. Once the error locator

polynomial is found, the Chien search method can be used to evaluate the roots. This method simply consists of the computation of $\Lambda(\alpha^j)$ for $j = 0, 1, \cdots, N-1$ and searching for a result equal to zero. Since the number of elements in a Galois field is finite, the Chien search method is feasible for evaluating the roots of the polynomial, i.e. the locations of the errors [21].

After evaluation of the roots of $\Lambda(x)$, the error values can be obtained by solving a system of $2T$ equations. An alternative method to find the error values is called Forney's algorithm [22]. Forney's algorithm is more efficient because it eliminates the extensive computation required for the matrix inversion.

In the algebraic decoding algorithm, first the received vector is transformed to the frequency domain by evaluating the syndromes, and then, based on the syndromes, the error locations and error values are found in the time domain. This algorithm is sometimes called the hybrid decoding algorithm [13].

With transform decoding, the received vector is first transformed to the frequency domain with $2T$ syndromes as the $2T$ components of its spectrum from 1 to $2T$. According to the construction of Reed-Solomon codes in the frequency domain, out of N components of the spectrum of the error pattern, $2T$ can be directly obtained from the syndromes. For the given $2T$ frequency domain components and the additional information that at most T components of the time domain error pattern are nonzero, the decoder must find the entire transform of the error pattern. Finally an inverse Fourier transform is performed to find the time domain error vector.

A frequency domain decoder was first proposed by Mandelbaum [29]. Imple-

mentation of transform domain decoders can be also found in [30][31][32][33][34]. By using Fermat theoretic transforms and continued fractions to implement a frequency domain RS decoder. Reed found that the transformation method is faster than the conventional method [35].

Welch and Berlekamp developed an algorithm that does not require evaluation of syndromes [36]. Instead this algorithm relies on the remainder polynomial obtained from the division of the received polynomial by the generator polynomial. The error locator polynomial can be obtained using the Welch-Berlekamp algorithm. Chien search can also be performed to find the roots of the error locator polynomial. The determination of the error values is quite difficult. There are four polynomials involved in this algorithm compared to only two polynomials in other decoding algorithms. Also this algorithm can only directly correct errors that occur in the information locations. Those errors occurred in the parity-check locations can be found after correcting errors located in the information symbols, by reencoding and comparing the received word and regenerated parity-check symbols. More on remainder-based decoding can be found in [37] [38].

1.2.4 Frequency Domain decoding vs Time Domain Decoding

As mentioned in Section 1.2.3, transform decoding treats data completely in the frequency domain. Algebraic decoding also deals with syndromes, which are the Fourier transform of the received data.

In 1980 Blahut proposed time domain decoding [39]. The time domain decoder works on the received data directly and Fourier transforms are not required in the

time domain algorithms. Blahut felt that these algorithms are good candidates for universal decoders [40]. The time domain decoding is based on a time domain equivalent of the Berlekamp-Massey algorithm. The decoding algorithm involves N iterations instead of $2T$ in the Berlekamp-Massey algorithm. In the first $2T$ iterations the error locators are found. In the next $(N - 2T)$ iterations, the error pattern is calculated. Note that there seems to be no obvious advantages in decoding speed for time domain methods. However, the regular algorithm structure of time domain methods is very suitable for VLSI implementation [40].

1.2.5 Evaluation of the Roots of the Error Locator Polynomial

Syndrome-based decoding always requires finding the error locator polynomial, then solving for the roots of the error locator polynomial. One of the methods uses look-up table. A table consisting of roots of the polynomial with the coefficients of polynomial as access address is constructed in advance. When the number of the coefficients and the size of Galois field get larger, the memory size will be too large.

In 1964 Chien [21] suggested that computing of the roots of the polynomial can be realized by evaluating the polynomial at α^j for $j = 0, 1, \cdots, N-1$ and checking for results equal to zero. Since the number of elements in a Galois field is finite, the Chien search method is feasible for evaluating the roots of the polynomial.

In 1987 Shayan, Le-Ngoc, and Bhargava proposed a binary-decision fast Chien search [41] which is a mixture of Chien search and look-up table method. The approach divides the Galois field into two halves. With a table established to indicate to which part of the Galois field the roots belong, only half of the Galois

field is searched and the required memory space is less than for the look-up table method.

In 1987 Okano and Imai also proposed a root search method based on LSI implementation. By using a certain transformation, the look-up table size can be reduced by $O(N)$ for double error correction or $O(N^2)$ for mutiple error correction.

In 1993 Young and Le-Ngoc [1] [2] conceived by experimental results that the roots of a quadratic polynomial over a finite field are not randomly distributed. Consequently, they developed a periodicity algorithm based on this observation. The algorithm needs $O(N)$ size of memory for root index table.

1.2.6 VLSI Structures

Now, we have introduced to the reader the basic concept of Reed-Solomon codes, but the problem of designing a low-complexity, high-bit-rate RS encoder and decoder still remains an active area of research. Some work has already been done in developing VLSI encoders and decoders for RS codes.

In 1982 Berlekamp developed a bit-serial Reed-Solomon encoder [43]. In his scheme, Berlekamp introduced a bit-serial multiplier algorithm, which required only shifting and exclusive OR operations for multiplication of two field elements by using a dual basis over a Galois field. A single VLSI (255, 223) RS encoder chip using Berlekamp's bit-serial multiplier, was realized in 1984 [44]. The encoder structure is more regular and simpler than the conventional architecture.

In 1989 Seroussi proposed a hypersystolic Reed-Solomon encoder to achieve very high sustained data rates, in the gigabit per second order of magnitude [45][46]. In

1986 Berlekamp presented a conceptual model of hypersystolic architectures [47]. A systolic array [48] is an array of computing cells with a regular interconnection pattern in which every cell communicates only with physically adjacent cells. In principle, there could be one global clock signal distributed to all the cells, and data transfer between adjacent cells could occur simultaneously throughout the array at each clock cycle. In a hypersystolic array [47][49], clocking signals are passed from cell to adjacent cell along with the data rather than being globally distributed. The resulting architecture is more practical than that of the systolic array since it can avoid clock skews if there are large number of cells in the array, and thus achieve higher data throughput. Berlekamp, Seroussi, and Tong had also patented the design of the hypersystolic Reed-Solomon decoder (HRSD) in [49].

Based on the idea of Brent and Kung that a pipeline architecture could be used to compute the greatest common divisor (gcd) of two polynomials, a new pipeline architecture for a transform decoder using a modified Euclidean algorithm was developed in 1985 [50]. In 1988 Shao and Reed presented a time domain RS decoding algorithm and its detailed VLSI architecture. It was shown that time domain decoding is more efficient than transform decoding in terms of VLSI implementation, since it can maintain the same throughput rate with less circuitry [51].

Reed-Solomon codes utilize the arithmetic of the finite field. The operations of addition and subtraction in the Galois field are simply bit-wise XOR operations. However, multiplication and division in Galois field are more complex and difficult. In 1984 Yeh, Reed, and Truong presented serial-in-serial-out systolic architectures for performing multiplication in finite fields $GF(2^m)$ [52]. Afterwards, several meth-

ods have been proposed to realize multiplication and division in finite fields. Wang *et al* implemented a Massey-Omura normal basis multiplier in 1985 [53]. In 1986 Scott, Tavares, and Peppard developed a fast VLSI multiplier using the standard basis [54]. A comparison of the VLSI architectures of three different finite field multipliers: the dual basis multiplier due to Berlekamp [43], the Massey-Omura normal basis multiplier, and the Scott-Tavares-Peppard standard basis multiplier is presented in [55].

Inverter structures are very complex. Wang *et al* presented a recursive pipeline inverter using Massey-Omura parallel-type multiplier based on the normal basis representation [53]. In 1992 Hasan and Bhargava proposed a bit-serial systolic divider over $GF(2^m)$ [56]. The structure is independent of the primitive polynomial used to generate finite field and the basis used to represent the field element.

1.3 Scope of the Work

The main objective of this research is first to develop a general purpose RS codec simulator. Since it was shown in [57] that time domain algorithms are slower than syndrome-based algorithms and furthermore transform decoding requires computing inverse Fourier transforms, we decide to adopt an algebraic decoding scheme. A software RS codes simulator [58] is developed in C language under both UNIX and MS-Windows operating systems. It can simulate an RS code with length between 7 and 255, and with the error correcting capability of up to $T = 20$. Different decoding algorithms are also investigated and compared.

For syndrome-based RS decoding, Chien search is the widely used method for

determining error locations. This method will take a great deal of time to locate error depending on the error position in the Galois field table. Periodicity properties of the distribution of the roots of the error locator polynomials are shown based on the relationship between coefficients and roots. A new periodicity algorithm is proposed for double error correction based on [2][1] to replace the Chien search method.

1.4 Organization of the Thesis

This thesis is organized as follows:

Chapter 2 first briefly reviews the encoding and decoding of RS codes. Then various RS decoding algorithms are presented.

Chapter 3 describes the structure of the software RS codes simulator.

Chapter 4 first demonstrates the periodicity properties that exist in the distribution of the roots of Galois field polynomials. Then a new decoding algorithm for finding roots of the error locator polynomial is proposed. Finally a hardware implementation of this algorithm is proposed.

Chapter 5 compares different decoding algorithms, Peterson's and Berlekamp-Massey's methods, Forney's and Gauss-Jordan's methods, the periodicity algorithm and other methods.

Chapter 6 concludes the thesis with a summary of results and suggestions for further research.

Chapter 2

Reed-Solomon Codes Overview

In this chapter, the Reed-Solomon codes are described. Several important decoding techniques are introduced.

2.1 General Description of Reed-Solomon Codes

The Reed-Solomon (RS) codes are a special subclass of nonbinary BCH codes, obtained by choosing the error locator field to be the same as the symbol field. The definition of an RS (N, K, T) code is as follows [58][61]:

A T-error-correcting Reed-Solomon code with symbols from the Galois field $GF(2^m)$ has the following parameters:

Block length $= N = 2^m - 1$

Number of parity-check symbols $= N - K = 2T$

Minimum distance $= d_{min} = 2T + 1$

where K is the number of information or message symbols. The RS codes are capable of correcting T random errors and one of the following error bursts:

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

  $\vdots$
- 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.

The generator polynomial $G(x)$ of a T-error-correcting RS code of length $2^m - 1$ is the polynomial of degree $N-K$ with coefficients from $GF(2^m)$. To save memory space, the self-reciprocal generator polynomial is preferred. It has

$$\alpha^{2^{m-1}-T}, \alpha^{2^{m-1}-T+1}, \cdots, \alpha^{2^{m-1}+T-1}$$

as its roots, where α is a primitive element in $GF(2^m)$.

$$G(x) = (x - \alpha^{2^{m-1}-T})(x - \alpha^{2^{m-1}-T+1})...(x - \alpha^{2^{m-1}+T-1})$$

2.2 Encoding and Decoding of RS Codes

Let $\overline{m} = (m_0, m_1, \cdots, m_{K-1})$ (K is the number of data symbols) be the data vector to be encoded, then the data polynomial can be defined as: $M(x) = m_0 + m_1 x + \cdots + m_{K-1}x^{K-1}$. A simple way to form the codeword polynomial $C(x)$ is $C(x) = M(x)G(x)$. However, this code is non-systematic because K data symbols are not explicitly present in the codeword and thus necessitating an extra step to extract the information from the corrected code word in the decoding process.

To increase the decoding speed, a systematic code word is generated by:

$$\frac{x^{2T}M(x)}{G(x)} = Q(x) + \frac{A(x)}{G(x)} \tag{2.1}$$

$$C(x) = Q(x)G(x) = x^{2T}M(x) + A(x) \tag{2.2}$$

where $A(x)$ is the parity check polynomial of degree $2T-1$.

When a codeword $C(x)$ is sent from the transmitter to the receiver, errors occur due to channel noise, distortion and fading. These errors can be modeled and presented as an error pattern

$$E(x) = e_0 + e_1x + \cdots + e_{N-1}x^{N-1}$$

Then the received word is given by

$$\begin{aligned} R(x) &= C(x) + E(x) \\ &= r_0 + r_1x + r_2x^2 + \cdots + r_{N-k}x^{N-1} \end{aligned}$$

The coefficients of $C(x)$, $E(x)$, and $R(x)$ are elements from $GF(2^m)$. The partial syndrome values of a received word can be obtained from

$$S_j = R(\alpha^j) = E(\alpha^j) \in GF(2^m), \qquad j = 2^{m-1}-T, 2^{m-1}-T+1, \cdots, 2^{m-1}+T-1$$

The error pattern polynomial $E(x)$ can be rewritten as follows:

$$E(x) = Y_1x^{i_1} + Y_2x^{i_2} + \cdots + Y_\nu x^{i_\nu}$$

where i_l is the actual location of the lth error and Y_l is the error value, $Y_l \in GF(2^m)$. Let $X_l = \alpha^{i_l}$ be the field element associated with this location, then its syndrome can be written as

$$S_j = \sum_{l=1}^{\nu} Y_l X_l^j, \qquad j = 2^{m-1}-T, 2^{m-1}-T+1, \cdots, 2^{m-1}+T-1$$

where X_l is the error locator of the lth error symbol and Y_l is the corresponding error value.

Once $E(x)$ is known, the estimated codeword $\hat{C}(x)$ can be obtained from:

$$\hat{C}(x) = R(x) + E(x)$$

In the following sections, we will present several RS decoding schemes.

2.3 Error Locator Polynomials

For a given received word $R(x)$, the syndromes S_j for $j = 2^{m-1}-T, \cdots, 2^{m-1}+T-1$ are given by

$$S_j = R(\alpha^j) = C(\alpha^j) + E(\alpha^j) = E(\alpha^j)$$

One of the essential RS decoding issues is to find the error locators $X_1, X_2, \cdots, X_\nu$, from $S_1, S_2, \cdots, S_{2T}$, where ν is the number of actual errors.

2.3.1 Peterson-Gorenstein-Zierler Decoding

Consider the polynomial in x,

$$\begin{aligned}\Sigma(x) &= x^\nu + \sigma_1 x^{\nu-1} + \cdots + \sigma_{\nu-1}x + \sigma_\nu \\ &= (x + X_1)(x + X_2)\cdots(x + X_\nu)\end{aligned}$$

known as the error-locator polynomial, where X_l for $l = 1, \cdots, \nu$ are error locators.

Another alternative representation of the error locator polynomial is

$$\Lambda(x) = 1 + \Lambda_1 x + \cdots + \Lambda_{\nu-1}x^{\nu-1} + \Lambda_\nu x^\nu \tag{2.3}$$

defined to have zeros at the inverse error locators X_l^{-1} , for $l = 1, \cdots, \nu$. That is,

$$\begin{aligned}\Lambda(x) &= (1 - xX_1)(1 - xX_2)\cdots(1 - xX_\nu) \\ &= \prod_{i=1}^{\nu}(1 - xX_i)\end{aligned}$$

In the following derivation, we adopt the error locator polynomial in the form of $\Lambda(x)$. Multiply both sides of Equation(2.3) by $Y_l X_l^{j+\nu}$ and set $x = X_l^{-1}$, then the left side of Equation(2.3) is zero and we have

$$0 = Y_l X_l^{j+\nu}(1 + \Lambda_1 X_l^{-1} + \cdots + \Lambda_{\nu-1} X_l^{-(\nu-1)} + \Lambda_\nu X_l^{-\nu}) \tag{2.4}$$

and

$$Y_l(X_l^{j+\nu} + \Lambda_1 X_l^{j+\nu-1} + \cdots + \Lambda_{\nu-1} X_l^{j+1} + \Lambda_\nu X_l^j) = 0 \tag{2.5}$$

$$l = 1, 2, \cdots, \nu$$

Sum up these equations from $l = 1$ to $l = \nu$. This gives, for each j

$$\sum_{l=1}^{\nu} Y_l X_l^{j+\nu} + \Lambda_1 \sum_{l=1}^{\nu} Y_l X_l^{j+\nu-1} + \cdots + \Lambda_{\nu-1} \sum_{l=1}^{\nu} Y_l X_l^{j+1} + \Lambda_\nu \sum_{l=1}^{\nu} Y_l X_l^j = 0 \tag{2.6}$$

The individual sums are known as syndromes, and thus the equation becomes

$$S_{j+\nu} + \Lambda_1 S_{j+\nu-1} + \Lambda_2 S_{j+\nu-2} + \cdots + \Lambda_\nu S_j = 0 \tag{2.7}$$

or

$$\Lambda_1 S_{j+\nu-1} + \Lambda_2 S_{j+\nu-2} + \cdots + \Lambda_\nu S_j = -S_{j+\nu} \tag{2.8}$$

$$j = 2^{m-1} - \nu, \cdots, 2^{m-1} - 1$$

We can write these equations in matrix form:

$$\begin{bmatrix} S_{2^{m-1}-\nu} & S_{2^{m-1}-\nu+1} & \cdots & S_{2^{m-1}-2} & S_{2^{m-1}-1} \\ S_{2^{m-1}-\nu+1} & S_{2^{m-1}-\nu+2} & \cdots & S_{2^{m-1}-1} & S_{2^{m-1}} \\ & & \vdots & & \\ S_{2^{m-1}-1} & S_{2^{m-1}} & \cdots & S_{2^{m-1}+\nu-3} & S_{2^{m-1}+\nu-2} \end{bmatrix} \begin{bmatrix} \Lambda_\nu \\ \Lambda_{\nu-1} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{2^{m-1}} \\ -S_{2^{m-1}+1} \\ \vdots \\ -S_{2^{m-1}+\nu-1} \end{bmatrix} \tag{2.9}$$

These equations can be solved by using the ordinary algebra except that multiplication, division, and addition are done based on rules of $GF(2^m)$. First we

find the appropriate value of ν as follows. Set $\nu = \mathrm{T}$, which is the error correcting capability, and compute the determinant of the above matrix. If it is nonzero, then we have $\nu \geq T$. If it is zero then $\nu < T$, reduce the trial value of ν by 1 and reduce the trial order of the matrix by 1. Repeat it until a nonzero determinant is obtained. We might assume that the actual number of errors occurred is ν. Then the reduced equations can be solved using the ordinary algebra over $GF(2^m)$. This is called Peterson's direct solution method [16][18].

2.3.2 The Berlekamp-Massey Algorithm

For correction of moderate to large numbers of errors (i.e. $\nu > 6$) with an RS code, Peterson's direct method of solving for the coefficients of error locator polynomial from the syndrome becomes cumbersome and inefficient due to the large number of multiplications and divisions that must be performed. It is better to use Berlekamp-Massey algorithm to solve error locators.

For the simplicity of the notation, we start to solve the problem as follows:

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_{\nu-1} & S_\nu \\ S_2 & S_3 & \cdots & S_\nu & S_{\nu+1} \\ S_3 & S_4 & \cdots & S_{\nu+1} & S_{\nu+2} \\ & & \vdots & & \\ S_\nu & S_{\nu+1} & \cdots & S_{2\nu-2} & S_{2\nu-1} \end{bmatrix} \begin{bmatrix} \Lambda_\nu \\ \Lambda_{\nu-1} \\ \Lambda_{\nu-2} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ -S_{\nu+3} \\ \vdots \\ -S_{2\nu} \end{bmatrix} \tag{2.10}$$

The Berlekamp-Massey algorithm can be derived as a problem in the design of a linear feedback shift register (LFSR) with initial states $S_1, S_2, \cdots, S_\nu$ and tap connections $-\Lambda_1, -\Lambda_2, \cdots, -\Lambda_\nu$ [25]. A diagram of an LFSR is shown in Figure 2.1. The length of the LFSR is ν.

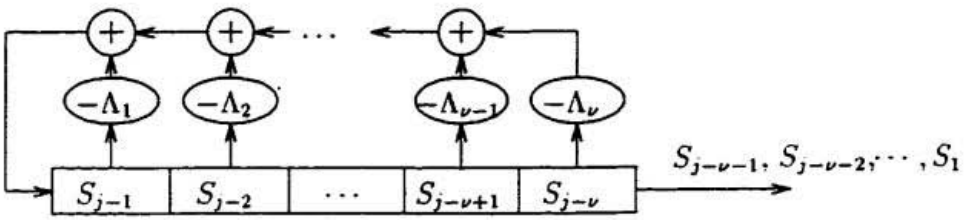

Figure 2.1: Linear feedback shift register for generating a sequence of syndrome values.

In LFSR, we also call the equation

$$\Lambda(x) = 1 + \Lambda_1 x + \Lambda_2 x^2 + \cdots + \Lambda_\nu x^\nu$$

the connection polynomial. For a given sequence of syndrome values, there are a determinable number of connection polynomials of various lengths that will generate the syndromes. This corresponds to the fact that there are a number of error patterns that can account for a given set of syndome values. However, the task of bounded-distance decoding is to find the lowest weight error pattern corresponding to the given syndrome. Therefore, in the design of LFSR we seek the lowest degree of connection polynomial $\Lambda(x)$ that generates the syndromes.

Now we are going to derive the recursive algorithm for producing a minimum length LFSR, which generates $S_1, S_2, \cdots, S_{2T}$. We define L_r as the length of the LFSR that generate $S_1, S_2, \cdots, S_r$. There is a Lemma by Massey [25] showing the change of the length of LFSR with the generation of a sequence of syndromes.

Lemma 1. If some LFSR of length L_{r-1} generates $S_1, S_2, \cdots, S_{r-1}$, but not $S_1, S_2, \cdots, S_r$ then

$$L_r \geq max[L_{r-1}, r - L_{r-1}]$$

If we can find a design that satisfies the inequality of the Lemma 1 with equality,

then it must be of the shortest length.

As an inductive hypothesis, assume a set of LFSRs which produce $S_1, S_2, \cdots, S_j$, with length L_j and connection polynomial $\Lambda^{(j)}(x)$ are found with equality

$$L_j = max[L_{j-1}, j - L_{j-1}], \qquad j = 1, 2, \cdots, r-1$$

We seek to find the LFSR that generates the syndrome values $S_1, S_2, \cdots, S_r$.

We have

$$S_j + \sum_{i=1}^{L_{r-1}} \Lambda_i^{(r-1)} S_{j-i} = \begin{cases} 0 & j = L_{r-1}, \cdots, r-1 \\ d_r & j = r \end{cases} \tag{2.11}$$

where the second term of Equation(2.11) computes the jth output of the $(r-1)th$ LFSR, and d_r, *called the rth discrepancy, is the difference between* S_r and the rth output of the $(r-1)$th LFSR, which we have found to generate the first $r-1$ syndrome values. If $d_r = 0, L_r = L_{r-1}$, then $\Lambda^{(r)}(x) = \Lambda^{(r-1)}(x)$. If $d_r \neq 0$, a new LFSR must be found to generate the first r syndrome values. Let $m-1$ be the syndrome sequence length before the last length change in the minimal length register. i.e.

$$L_{m-1} < L_{r-1}$$

$$L_m = L_{r-1}$$

We have

$$S_j + \sum_{i=1}^{L_{m-1}} \Lambda_i^{(m-1)} S_{j-i} = \begin{cases} 0 & j = L_{m-1}, \cdots, m-1 \\ d_m \neq 0 & j = m \end{cases} \tag{2.12}$$

By the induction hypothesis, for m Lemma 1 holds with equality

$$L_m = L_{r-1} = max[L_{m-1}, m - L_{m-1}]$$

Because of $L_{m-1} < L_{r-1}$, this gives

$$L_{r-1} = m - L_{m-1} \tag{2.13}$$

We claim that the connection polynomial

$$\Lambda^{(r)}(x) = \Lambda^{(r-1)}(x) - d_r d_m^{-1} x^{r-m} \Lambda^{(m-1)}(x) \tag{2.14}$$

is a valid choice for $\Lambda^{(r)}(x)$. Moreover, it follows from Equation(2.14) that

$$S_j + \sum_{i=1}^{L_r} \Lambda_i^{(r)} S_{j-i} = S_j + \sum_{i=1}^{L_{r-1}} \Lambda_i^{(r-1)} S_{j-i} - d_r d_m^{-1} [S_{j-r+m} + \sum_{i=1}^{L_{m-1}} \Lambda_i^{(m-1)} S_{j-r+m-i}] \tag{2.15}$$

The first two terms of the right-hand side sum up to:

$$S_j + \sum_{i=1}^{L_{r-1}} \Lambda_i^{(r-1)} S_{j-i} = \begin{cases} 0 & j = L_{r-1}, \cdots, r-1 \\ d_r & j = r \end{cases} \tag{2.16}$$

as described in Equation(2.11). We have from Equation(2.12) that

$$S_{j-r+m} + \sum_{i=1}^{L_{m-1}} \Lambda_i^{(m-1)} S_{j-r+m-i} = \begin{cases} 0 & j = L_{m-1} + r - m, \cdots, r-1 \\ d_m & j = r \end{cases} \tag{2.17}$$

From Equation(2.13), $L_{m-1} = m - L_{r-1}$

$$L_{m-1} + r - m = m - L_{r-1} + r - m = r - L_{r-1} \leq L_r,$$

we obtain

$$S_{j-r+m} + \sum_{i=1}^{L_{m-1}} \Lambda_i^{(m-1)} S_{j-r+m-i} = \begin{cases} 0 & j = L_r, \cdots, r-1 \\ d_m & j = r \end{cases} \tag{2.18}$$

Summing up the right-hand side of Equation(2.15), we get

$$S_j + \sum_{i=1}^{L_r} \Lambda_i^{(r)} S_{j-i} = \begin{cases} 0 & j = L_r, \cdots, r-1 \\ d_r - d_r d_m^{-1} d_m & j = r \end{cases} \tag{2.19}$$

Also from Equation(2.14) the degree of $\Lambda^{(r)}(x)$ is at most

$$\begin{aligned} max[L_{r-1}, r-m+L_{m-1}] &= max[L_{r-1}, r-m+m-L_{r-1}] \\ &= max[L_{r-1}, r-L_{r-1}] \end{aligned}$$

From the above induction, an LFSR algorithm for synthesizing a shortest LFSR to generate the syndrome sequence $S_1, S_2, \cdots, S_{2T}$ is described in detail below.

Massey LFSR Synthesis Algorithm (Berlekamp Algorithm)

1. Initialize algorithm variables

   Let $L = 0, r = 1, \Lambda(x) = 1, D(x) = x$

2. Take in new syndrome value and compute discrepancy

   $d = S_r + \sum_{i=1}^{L} \Lambda_i S_{r-i}$

3. Test discrepancy

   If $d = 0$, go to step 8. Otherwise, go to step 4.

4. Modified connection polynomial

   If $d \neq 0$, Let $\Lambda^*(x) = \Lambda(x) - dD(x)$

5. Test register length

   If $2L \geq r$, go to step 7(i.e. do not extend register). Otherwise, go to step 6.

6. Change register length and update correction term

   Let $L = r - L$ and $D(x) = \Lambda(x)/d$

7. Update connection polynomial

   Let $\Lambda(x) = \Lambda^*(x)$

8. Update correction term

   Let $D(x) = xD(x)$

9. Update syndrome counter

   Let $r = r + 1$

10. Test syndrome counter

    If $r < 2T + 1$, where T represent for error correction capability, go to step 2.

11. Otherwise stop.

In the algorithm, for every stage r when step 2 has just been reached, then the quantities produced by the algorithm bear the following relations to the quantities appearing in the developing procedure:

$$\begin{aligned} \Lambda(x) &= \Lambda^{r-1}(x) \\ L &= L_{r-1} \\ d &= d_r \\ D(x) &= d_m^{-1} x^{r-m} \Lambda^{(m-1)}(x) \end{aligned}$$

The algorithm stops after $2T$ iterations. The length L of the LFSR reflects the actual error number ν, i.e. $\nu = L$. If the algorithm terminates with an LFSR connection polynomial of degree greater than T, that is, $L > T$, then we are not assured that the corresponding error-locator polynomial is correct, and error detection is announced.

2.3.3 Improved Berlekamp-Massey Algorithm

Rewrite the same matrix equation as Equation (2.10) for convenience.

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_{\nu-1} & S_\nu \\ S_2 & S_3 & \cdots & S_\nu & S_{\nu+1} \\ S_3 & S_4 & \cdots & S_{\nu+1} & S_{\nu+2} \\ & & \vdots & & \\ S_\nu & S_{\nu+1} & \cdots & S_{2\nu-2} & S_{2\nu-1} \end{bmatrix} \begin{bmatrix} \Lambda_\nu \\ \Lambda_{\nu-1} \\ \Lambda_{\nu-2} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ -S_{\nu+3} \\ \vdots \\ -S_{2\nu} \end{bmatrix} \tag{2.20}$$

The matrix is nonsingular if the number of errors is ν, and singular if the number of errors is less than ν.

Now assume the actual number of errors occurred is ν, $\nu < T$, we have

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_{\nu-1} & S_\nu \\ S_2 & S_3 & \cdots & S_\nu & S_{\nu+1} \\ & & \vdots & & \\ S_T & S_{T+1} & \cdots & S_{T+\nu-2} & S_{T+\nu-1} \end{bmatrix} \begin{bmatrix} \Lambda_\nu \\ \Lambda_{\nu-1} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ \vdots \\ -S_{T+\nu} \end{bmatrix} \tag{2.21}$$

Let D be matrix

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_{\mu-1} & S_\mu \\ S_2 & S_3 & \cdots & S_\mu & S_{\mu+1} \\ & & \vdots & & \\ S_\mu & S_{\mu+1} & \cdots & S_{2\mu-2} & S_{2\mu-1} \end{bmatrix}$$

where $\nu + 1 \leq \mu \leq T$, and

$$d_j = \begin{bmatrix} S_j \\ S_{j+1} \\ \vdots \\ S_{j+\mu-1} \end{bmatrix}, \qquad j = 1, 2, \cdots, \mu$$

be the column vectors of D. From Equation(2.21), we have

$$\begin{aligned} d_1\Lambda_\nu + d_2\Lambda_{\nu-1} + \cdots + d_\nu\Lambda_1 &= d_{\nu+1} \\ d_2\Lambda_\nu + d_3\Lambda_{\nu-1} + \cdots + d_{\nu+1}\Lambda_1 &= d_{\nu+2} \\ &\vdots \\ d_{\mu-\nu}\Lambda_\nu + d_{\mu-\nu+1}\Lambda_{\nu-1} + \cdots + d_{\mu-1}\Lambda_1 &= d_\mu \end{aligned}$$

Since $d_j, j = \nu+1, \cdots, \mu$ can be formed by $d_1, d_2, \cdots, d_\nu$, hence, if $\Lambda_1, \Lambda_2, \cdots, \Lambda_\nu$ satisfy all T equations of (2.21), they would satisfy the set of equations:

$$\begin{aligned} S_1\Lambda_\nu + S_2\Lambda_{\nu-1} &+ \cdots + S_\nu\Lambda_1 = S_{\nu+1} \\ S_2\Lambda_\nu + S_3\Lambda_{\nu-1} &+ \cdots + S_{\nu+1}\Lambda_1 = S_{\nu+2} \\ &\vdots \\ S_{\mu-\nu}\Lambda_\nu + S_{\mu-\nu+1}\Lambda_{\nu-1} &+ \cdots + S_{\mu-1}\Lambda_1 = S_\mu \end{aligned} \tag{2.22}$$

and $\Lambda_1, \Lambda_2, \cdots, \Lambda_\nu$ are the coefficients of the error location polynomial.

We first review Berlekamp-Massey's algorithm. In each stage of iteration, a new syndrome is used in the calculation of the discrepancy. At stage r the coefficients of the $\Lambda^{(r)}(x)$ satisfy a subset of Equation(2.21) that contains $S_j, 1 \leq j \leq r$. Since $\nu = L_r$ at that stage, the number of equations satisfied is $r - \nu = r - L_r$. If $r - L_r = T$, the coefficients of $\Lambda^{(r)}(x)$ satisfy all T equations of (2.21). According to the statement above, $\Lambda^{(r)}(x)$ would be the error locator polynomial. Therefore, we have the following modified algorithm [60]:

1. Initialize algorithm variables

Let $L = 0, r = 1, \Lambda(x) = 1, D(x) = x$

2. Take in new syndrome value and compute discrepancy

$d = S_r + \sum_{i=1}^{L} \Lambda_i S_{r-i}$

3. Test discrepancy

If $d = 0$, go to step 8. Otherwise, go to step 4.

4. Modified connection polynomial

If $d \neq 0$, Let $\Lambda^*(x) = \Lambda(x) - dD(x)$

5. Test register length

If $2L \geq r$, go to step 7(i.e. do not extend register). Otherwise, go to step 6.

6. Change register length and update correction term

Let $L = r - L$ and $D(x) = \Lambda(x)/d$

7. Update connection polynomial

Let $\Lambda(x) = \Lambda^*(x)$

8. Update correction term

Let $D(x) = xD(x)$

9. Update syndrome counter

Let $r = r + 1$

10.Test syndrome counter

If $r < 2T + 1$, and $r < T + L$ go to step 2; otherwise, stop.

All steps are the same as in the Berlekamp-Massey algorithm except step 10.

The modified algorithm causes the iteration procedure to stop at an early stage when the number of errors that have occurred is less than T. The modified algorithm requires a total of $(T + \nu)$ iterations, where ν is the actual number of errors, as compared to $2T$ iterations required in Berlekamp-Massey's algorithm. The reduction of $T - \nu$ iterations for ν errors results in the increase of decoding speed. The overall performance of the improved algorithm depends on the probability distribution of the errors.

Both the Peterson and Berlekamp-Massey algorithm can be used for finding error locator polynomials. A time comparison between Peterson's and Berlekamp-Massey's algorithm will be made in Section 5.1. We will find that Berlekamp-Massey's algorithm is faster than Peterson's for the correction of more than 6 errors,

2.4 Error Locators

The roots of the error locator polynomial can be found using Chien search. One simply substitutes each element α^j of $GF(2^m)$ into $\Lambda(x)$ and checks for zero. Thus the error-locator polynomial can be decomposed as follows:

$$\Lambda(x) = \prod_{l=1}^{\nu}(1 - xX_l) \qquad l = 1, \cdots, \nu \tag{2.23}$$

where X_l is the error locator of the lth error symbol.

2.5 Magnitude of Error Pattern

Once the error locations have been obtained, the next decoding procedure is to compute the error magnitude.

2.5.1 The Gauss-Jordan Elimination Method

We return to the equations defining the syndromes.

$$S_1 = Y_1X_1 + Y_2X_2 + \cdots + Y_\nu X_\nu$$
$$S_2 = Y_1X_1^2 + Y_2X_2^2 + \cdots + Y_\nu X_\nu^2$$

$$\vdots$$

$$S_{2T} = Y_1 X_1^{2T} + Y_2 X_2^{2T} \quad + \quad \cdots + Y_\nu X_\nu^{2T}$$

The first ν equations can be solved for the error magnitudes if the determinant of the matrix of coefficients is nonzero. Actually the matrix does have a nonzero determinant if ν errors occur because $X_1, X_2, \cdots, X_\nu$ are nonzero and distinct. If this is the case, the Gauss-Jordan elimination method can be used for solving this linear matrix equation. In the following subsection, we shall introduce an alternative method for determining error values, thereby eliminating the need for solving simultaneous equations.

2.5.2 The Forney Algorithm

We have error-locator polynomial

$$\Lambda(x) = \prod_{l=1}^{\nu}(1 - xX_l) \qquad l = 1, \cdots, \nu.$$

Define the syndrome polynomial

$$S(x) = \sum_{j=1}^{2T} S_j x^{j-1} = \sum_{j=1}^{2T}\sum_{i=1}^{\nu} Y_i X_i^j x^{j-1}$$

and define the error-evaluator polynomial $\Omega(x)$ in terms of these known polynomials:

$$\Omega(x) = S(x)\Lambda(x) \pmod{x^{2T}}.$$

Expand each term by the definition, we get

$$\begin{aligned}\Omega(x) &= [\sum_{j=1}^{2T}\sum_{i=1}^{\nu} Y_i X_i^j x^{j-1}] \cdot [\prod_{l=1}^{\nu}(1 - X_l x)] \pmod{x^{2T}} \\ &= \sum_{i=1}^{\nu} Y_i X_i [\sum_{j=1}^{2T}(X_i x)^{j-1}(1 - X_i x)] \prod_{l \neq i}(1 - X_l x) \pmod{x^{2T}}.\end{aligned}$$

The bracketted term is a factorization of $(1 - X_i^{2T} x^{2T})$. Therefore

$$\Omega(x) = \sum_{i=1}^{\nu} Y_i X_i x (1 - X_i^{2T} x^{2T}) \prod_{l \neq i} (1 - X_l x) \pmod{x^{2T}}. \tag{2.24}$$

After this is modulo x^{2T}, we get

$$\Omega(x) = \sum_{i=1}^{\nu} Y_i X_i \cdot \prod_{l \neq i} (1 - X_l x) \tag{2.25}$$

Substitute X_l^{-1} in Equation(2.25) to get

$$\Omega(X_l^{-1}) = Y_l X_l \prod_{j \neq l} (1 - X_j X_l^{-1}) + \prod_{i \neq l} Y_i X_i \prod_{j \neq i} (1 - X_j X_l^{-1}) \tag{2.26}$$

Since X_l^{-1} is the root of $\Lambda(x)$, the second term in the Equation (2.26) is zero, and we get

$$\Omega(X_l^{-1}) = Y_l X_l \cdot \prod_{j \neq l} (1 - X_j X_l^{-1})$$

Hence, the error magnitude can be given by

$$Y_l = \frac{X_l^{-1} \Omega(X_l^{-1})}{\prod_{j \neq l} (1 - X_j X_l^{-1})} \tag{2.27}$$

Moreover, the derivative of $\Lambda(x)$ is

$$\Lambda'(x) = -\sum_{i=1}^{\nu} X_i \cdot \prod_{j \neq i} (1 - x X_j)$$

Hence, another form of the Equation(2.27) is

$$Y_l = -\frac{\Omega(X_l^{-1})}{\Lambda'(X_l^{-1})}$$

The Forney algorithm provides a considerable improvement over matrix inversion. We will make a decoding time comparison between Forney algorithm and Gauss-Jordan elimination method in Section 5.2. We will find that for correction of more than 10 errors, Forney algorithm is faster than Gauss-Jordan elimination.

Chapter 3

Reed-Solomon Simulator

In this chapter, a software RS codes simulator is discussed. First, the overall structure of the simulator is described. Then, the data structures used in this simulator are introduced. Finally, some flowcharts of major subroutines are also given.

3.1 General description

The RS code simulator has been implemented in C under both UNIX operating system and MS-WINDOWS. A graphic user interface (GUI) is also provided for PC user using Visual Basic. This software package can be used to correct any random errors occurring in an $N = 2^m - 1$ symbol codeword, where $3 \leq m \leq 8$ and $T \leq min((N-1)/2, 20)$. The overall basic structure of the simulator is given in Figure 3.1. It is divided into 12 blocks.

Block 1: This subroutine allows users to input the number m of bits per symbol, error correcting capability T, and a data word through a GUI.

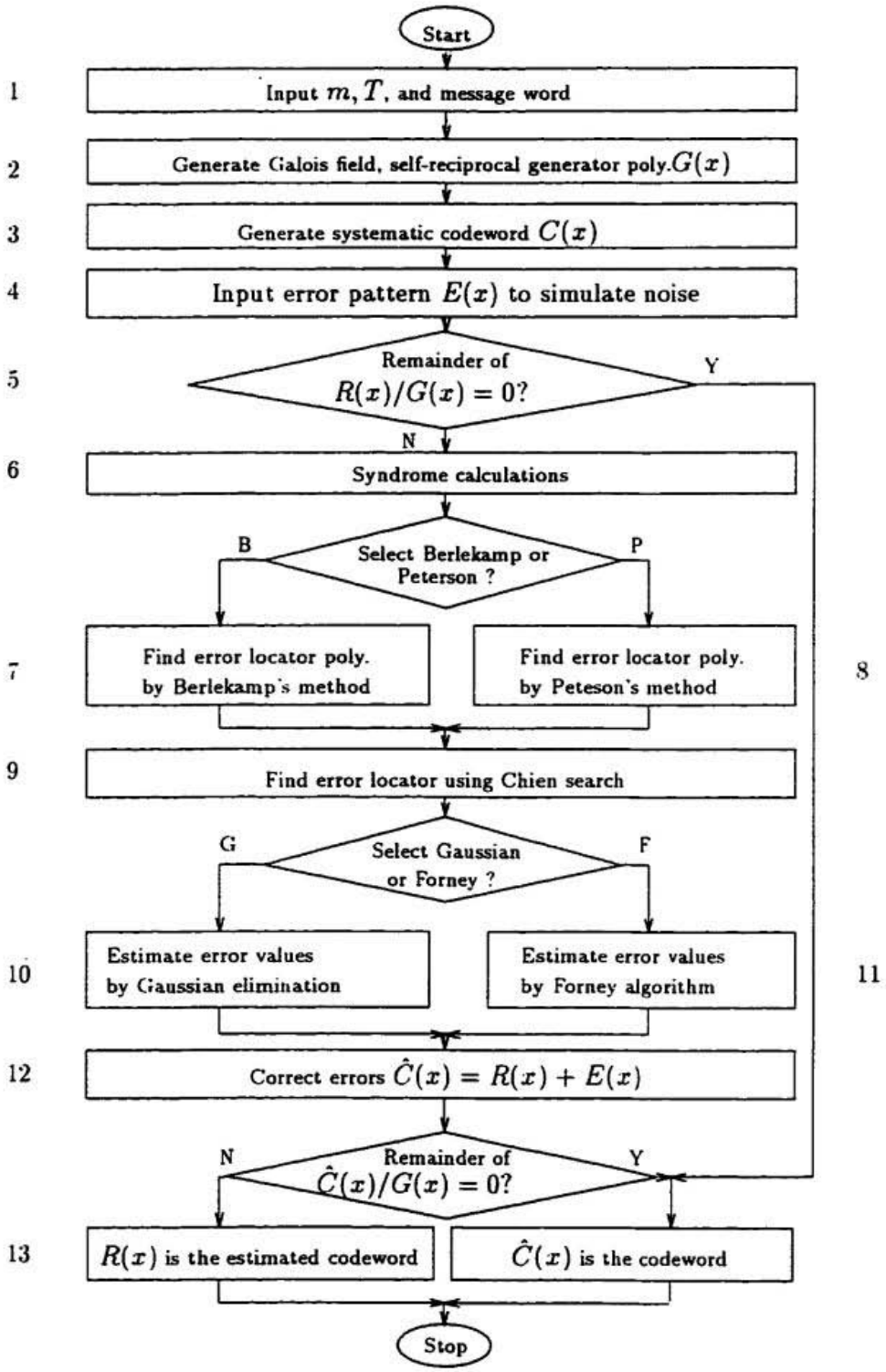

Figure 3.1: Overall flowchart of simulator

Block 2: For the software implementation, the necessary Galois field multiplications are performed using the address pointing approach. Therefore, a table of both binary and power representations of the Galois field $GF(2^m)$ is formed here. The self-reciprocal generator polynomial $G(x)$ is also generated in this block.

Block 3: In this step, the simulator encodes the data and forms a systematic codeword. It also presents the user with the list of the data, the codeword, and the generator polynomial $G(x)$.

Block 4: Users can form the error pattern and add it to the generated codeword to generate the corrupted received word.

Block 5: In case the corrupted received word is a codeword, the block will output the message directly without going through the error correcting process. In this case the decoder is blind if the error pattern is a codeword.

Block 6: The first $2T$ syndromes of the received word are calculated in this step.

Block 7,8: The error locator polynomial is determined by using either Berlekamp's or Peterson's method depending on the user's option.

Block 9: The error locators $X_1, X_2, \cdots, X_\nu$, where ν is the actual number of errors, are evaluated using Chien search by substituting each element of the $GF(2^m)$ into the error locator polynomial until a zero is found. The process repeats until the end of the Galois field or all the roots come out.

Block 10,11: The error values $Y_1, Y_2, \cdots, Y_\nu$, where ν is the actual number of errors, are obtained using either Gauss-Jordan elimination or Forney's method depending on user's choice. Thus the error pattern $E(x)$ becomes known.

Block 12: This step directly obtains the estimated received codeword $\hat{C}(x)$ by adding the error pattern $E(x)$ to the received word $R(x)$. Finally the estimated codeword is obtained.

Block 13: The decoding procedure is finally verified by checking the remainder of the estimated word $\hat{C}(x)$ with the generator polynomial $G(x)$. In case the estimated codeword $\hat{C}(x)$ is not a codeword, the block will output the information, "Uncorrectable errors have occurred. Resend the data." This helps to confirm the correctness of the program, as well as to indicate that the number of errors have exceeded the limits allowable by the decoder, i.e. $\nu > T$.

3.2 Data Structure

Each element of the Galois field $GF(2^m)$ has two representations, that is, the binary and power representation. The addition or subtraction of two elements can be easily performed in modulo-2 operation by using the binary representation. The power representation makes multiplication become addition.

For example, let $m = 4$ and let the primitive polynomial $P(x) = x^4 + x + 1$ be selected to construct the Galois field. The elements of $GF(2^4)$ are given in Table 3.1.

The field element α^5 has m-tuple binary representation of (0 1 1 0) which corresponds to $\alpha^2 + \alpha$ and α^7 has of (1 1 0 1), where m is 4.

$$\alpha^5 + \alpha^7 = (\alpha^2 + \alpha) + (\alpha^3 + \alpha + 1) = \alpha^3 + \alpha^2 + 1 = \alpha^{13}$$

Obviously, the addition of the two field elements can be carried out by simply

Table 3.1: Galois field of $GF(2^4)$ generated by $p(x) = 1 + x + x^4$

Power representation	Polynomial representation α^0	α^1	α^2	α^3
-	0	0	0	0
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	1	0	0
5	0	1	1	0
6	0	0	1	1
7	1	1	0	1
8	1	0	1	0
9	0	1	0	1
10	1	1	1	0
11	0	1	1	1
12	1	1	1	1
13	1	0	1	1
14	1	0	0	1

adding the corresponding components of their m-tuple binary representations in modulo-2 addition. To multiply two elements, we can simply add their exponents. For example,

$$\alpha^5 \cdot \alpha^7 = \alpha^{5+7} = \alpha^{12}$$

Therefore, the following structure *gf_element* is used to describe a Galois field element in the software design.

```
struct {
        int power_rep
        int binary_rep
} gf_element
```

Two look-up tables, *log*[] and *alog*[] table are built such that

$$log[\alpha^i] = i, \quad \text{for } 0 \le i \le 2^m - 2$$
$$log[0] = -1$$
$$alog[i] = \alpha^i, \quad \text{for } 0 \le i \le 2^m - 2$$
$$alog[-1] = 0$$

where α is the primary element of the Galois field $GF(2^m)$, and i and α^i refer to power and binary representation of Galois field element. The detailed construction of these tables will be described in the next section.

For any given two elements $\beta = \alpha^i$ and $\gamma = \alpha^j$, where $i, j = -1, 0, \cdots, 2^m - 2$, let $\theta = \beta \pm \gamma$ and $\phi = \beta \cdot \gamma$. The binary representation of θ is $\beta + \gamma$, in which bit-wise modulo-2 operation is performed on the binary representations of β and γ. The power representation of θ can be obtained from *log*[] table.

The power representation of ϕ is

$$\begin{cases} -1 & \text{if } i = -1 \text{ or } j = -1 \\ (i+j) \bmod (2^m - 1) & \text{otherwise} \end{cases}$$

and the binary representation of ϕ is *alog*[]. Thus Galois field multiplication is simply modulo-$(2^m - 1)$ summation of the powers of the multiplicands with some conditions. This method for Galois field arithmetic is widely used in software decoders in order to increase the speed of the multiplication in Galois fields.

3.3 Generation of Galois Field

In the proceeding section, a look-up table method was introduced for Galois field arithmetic. In this section, the generation of *log*[] and *alog*[] table of Galois field is considered.

In this codec simulator, we use the six primitive polynomials illustrated in Table 3.2. We store these polynomials in a 6-element primitive polynomial array. Each element stores all the coefficients of the primitive polynomial, except the highest-order coefficients, in binary form with the lowest-order coefficient at the left. For example, let $m = 8$ and the primitive polynomial $P(x) = x^8 + x^7 + x^2 + x + 1$, the corresponding array element in binary form is 10000111. The primitive polynomial array is also shown in Table 3.2.

The construction of the *alog*[] table is in agreement with the construction of the $GF(2^m)$ elements, i.e. determining the binary representation for each α^i, $i = -1, 0, \cdots, 2^m - 2$. It can be implemented as follows:

1. Initialize $alog[-1] = 0$, $alog[0] = 1$, and $i = 1$.

Table 3.2: Primitive polynomials for generation of $GF(2^m)$

m	Primitive polynomials	Binary store representation
3	$x^3 + x + 1$	11
4	$x^4 + x + 1$	11
5	$x^5 + x^2 + 1$	101
6	$x^6 + x + 1$	11
7	$x^7 + x^3 + 1$	1001
8	$x^8 + x^7 + x^2 + x + 1$	10000111

2. Left shift the binary representation of α^i by 1 bit to get α^{i+1}.

3. If $\alpha^{i+1} < 2^m$, then go to 6.

4. Otherwise if $\alpha^{i+1} \geq 2^m$, then XOR the result with the primitive polynomial array element defined in column 3, Table 3.2.

5. AND the result with $2^m - 1$ to get m-bit binary representation, i.e. $alog[i+1]$.

6. $i = i + 1$

7. If $i < 2^m - 2$, go to 2.

8. Otherwise stop.

Figure 3.2 gives the flowchart for the generation of the Galois field.

The construction of the *log*[] table, i.e. determine the power representation of each Galois field element from its binary representation, can be obtained by exchanging the contents of the index and the entry of the *alog*[] table.

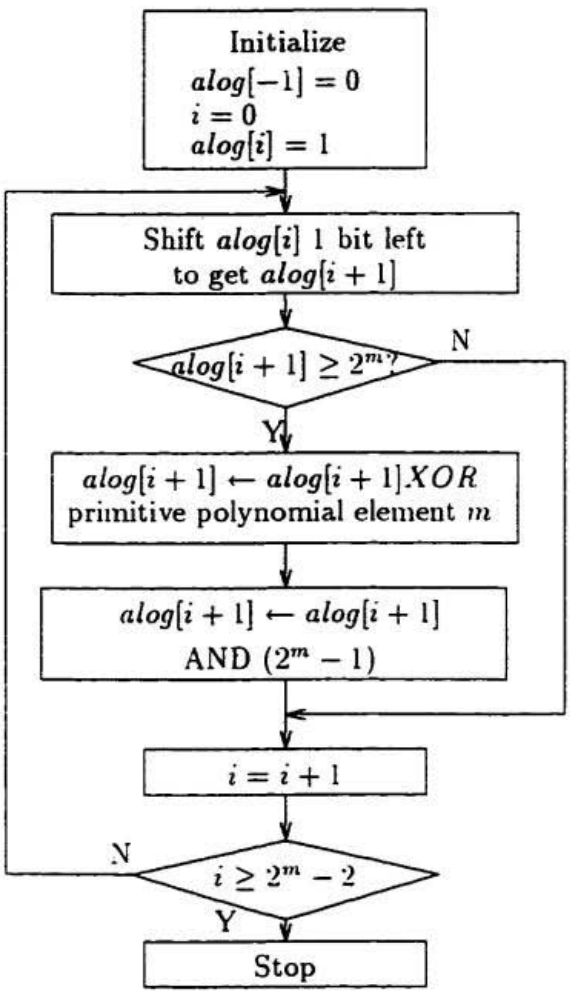

Figure 3.2: The generation of finite field $GF(2^m)$

3.4 Flowchart of Main Subroutines

In this section, the main subroutines of the RS codes simulator, such as the Peterson and Berlekamp algorithm to compute error locator polynomial, Chien search to find error locators, and the Gauss-Jordan elimination and Forney algorithm to compute error magnitudes, are treated in detail.

3.4.1 The Peterson's Direct Method

The evaluation of the coefficients of the error locator polynomial involves solving the following matrix equations [16] [18]:

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_{\nu-1} & S_\nu \\ S_2 & S_3 & \cdots & S_\nu & S_{\nu+1} \\ S_3 & S_4 & \cdots & S_{\nu+1} & S_{\nu+2} \\ & & \vdots & & \\ S_\nu & S_{\nu+1} & \cdots & S_{2\nu-2} & S_{2\nu-1} \end{bmatrix} \begin{bmatrix} \Lambda_\nu \\ \Lambda_{\nu-1} \\ \Lambda_{\nu-2} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ -S_{\nu+3} \\ \vdots \\ -S_{2\nu} \end{bmatrix} \tag{3.1}$$

The method for solving these linear equations involves two steps [13]. Firstly, the size of the matrix, which is equal to the actual number of errors ν is determined. Secondly, the coefficients of $\Lambda(x)$ can then be computed using the value of ν and Equation 3.1. The detailed description is treated in Section 2.3.1. The flowchart is shown in Figure 3.3.

3.4.2 The Berlekamp-Massey Algorithm

An alternate technique for obtaining the error locator polynomial $\Lambda(x)$ is the Berlekamp-Massey algorithm. This method was explained previously in Section 2.3.2. Figure. 3.4 illustrates the flowchart of the Berlekamp-Massey algorithm.

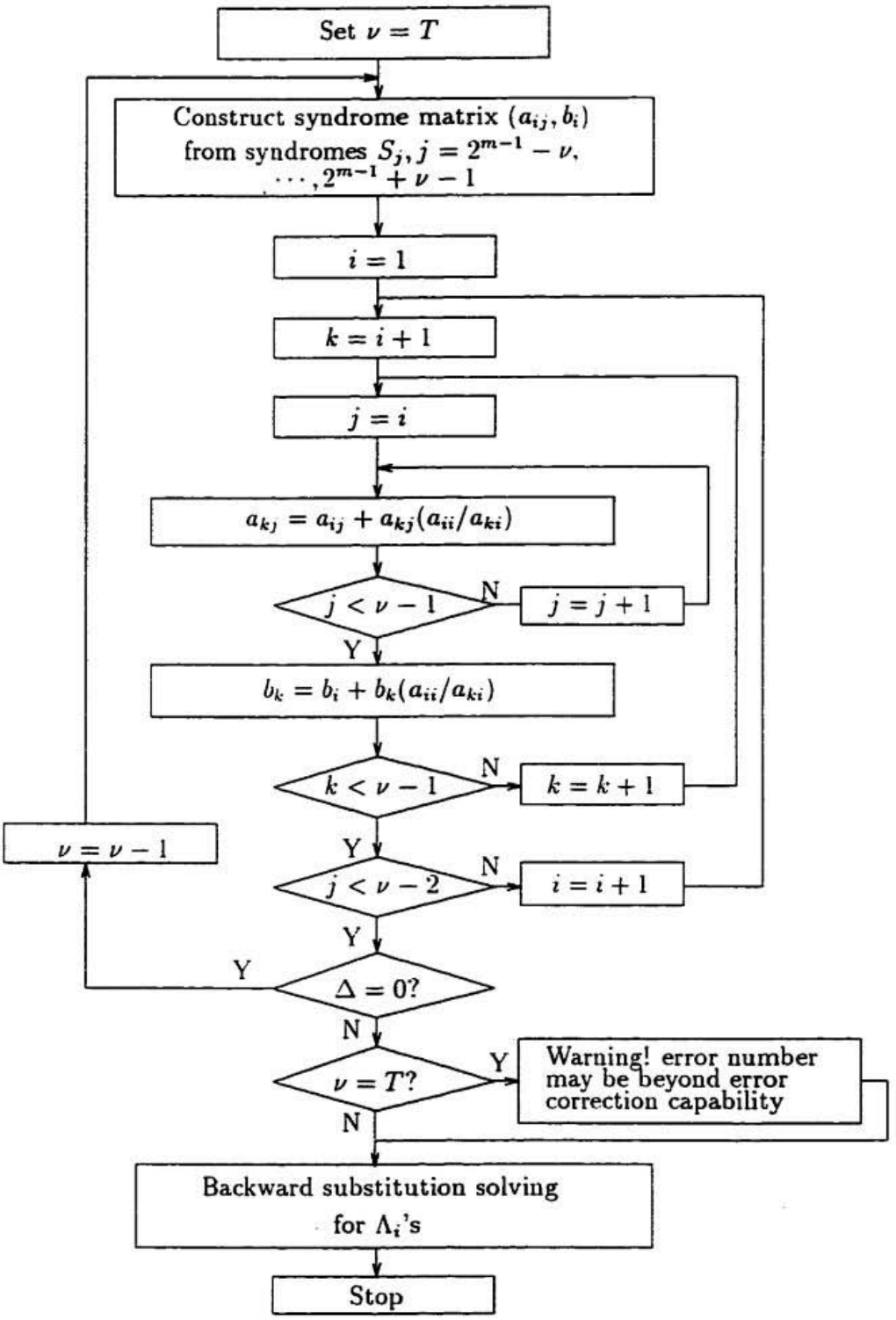

Figure 3.3: Peterson's direct method solving for error locator polynomial

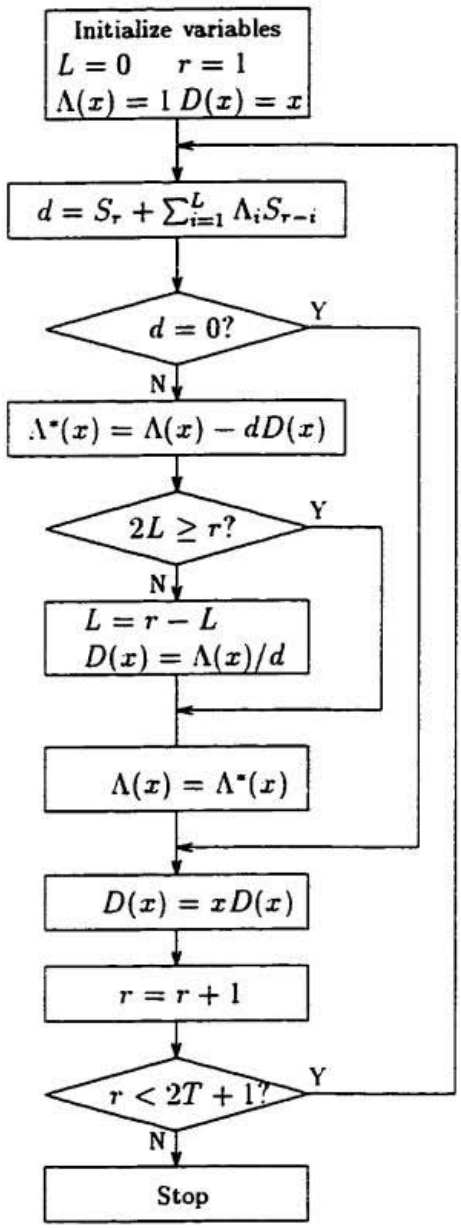

Figure 3.4: Flow chart of Berlekamp-Massey Algorithm

3.4.3 Chien Search

Once the error locator polynomial is found, the Chien search method can be used to evaluate the roots, and hence the error locations can be determined. This method involves computations of $\Lambda(\alpha^j)$ for $j = 0, 1, \cdots, N-1$ and checking for results equal to zero [21]. The method is shown as a flowchart in Figure 3.5.

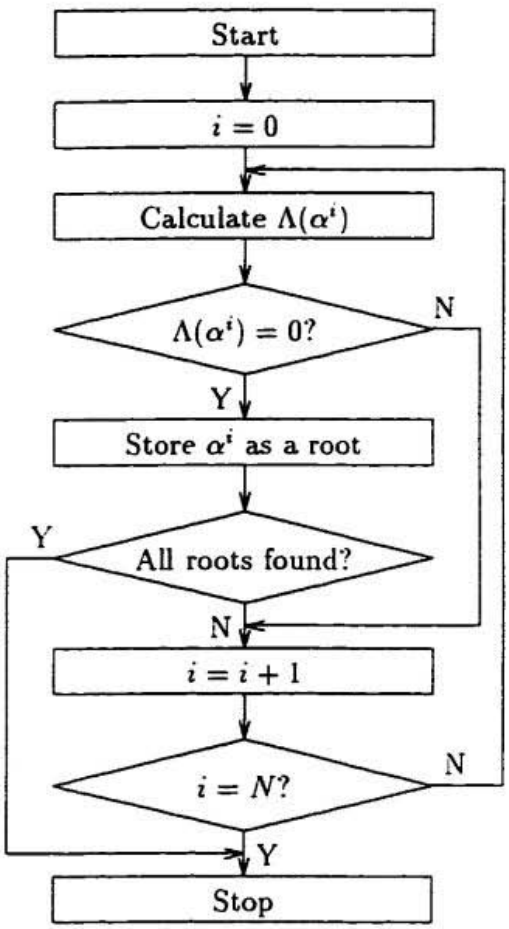

Figure 3.5: The flowchart for the Chien search

3.4.4 The Gauss-Jordan Elimination Method

After computation of the roots of $\Lambda(x)$, the error locator $X_l = \alpha^{i_l}$ can be substituted into the following equations:

$$S_j = Y_1 X_1^j + Y_2 X_2^j + \cdots + Y_\nu X_\nu^j, \; j = 1, 2, \cdots, 2T$$

This is a system of $2T$ linear equations and can be solved for the error values $Y_j, j = 1, 2, \cdots, \nu$ by the Gauss-Jordan elimination method. The flowchart is shown in Figure 3.6.

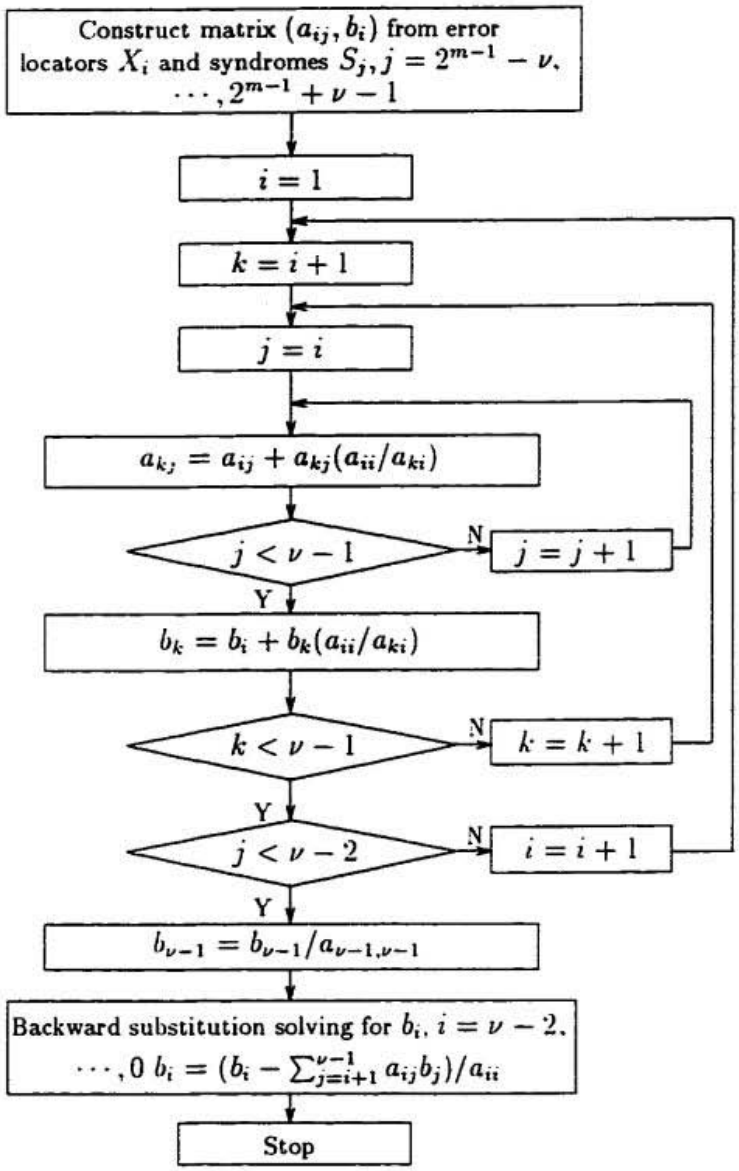

Figure 3.6: The flowchart of the Gauss-Jordan elimination method

3.4.5 The Forney Algorithm

A more efficient method to find the error values is given by using Forney's algorithm [22]. The flowchart of the Forney algorithm is shown in Figure 3.7.

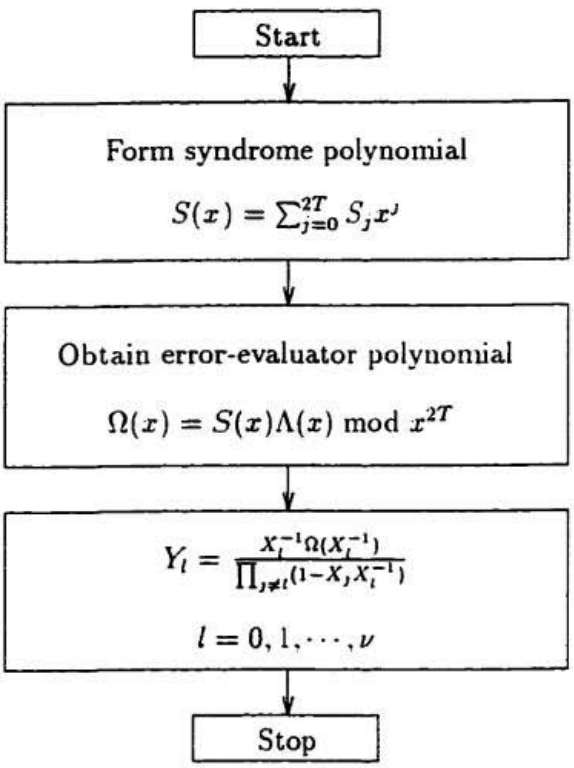

Figure 3.7: The flowchart of Forney algorithm

3.5 Simulation Results

Several examples are listed below to show the encoding and decoding of the RS codes in different cases when $T = \nu, T < \nu$, and $T > \nu$.

Example 3.1

- Given: $m = 3, T = 2$, and $\nu = 2$.
- Peterson's method is selected.
- Forney's algorithm is selected.
- The inputs and outputs are listed below.

Simulation results of Example 3.1

```
RS CODEC for 3 <= m <= 8 and T <= 20
================================================
Input bits per symbol m  or  code length N , (m or N):  m
Please select m (3 <= m <= 8): 3
Please select T (T <=  3) :2
Generator polynomial:

  alpha^0
+ alpha^4 x^1
+ alpha^2 x^2
+ alpha^4 x^3
+ alpha^0 x^4

Roots of the generator polynomial:
  alpha^2
, alpha^3
, alpha^4
```

```
, alpha^5

Select K symbols data based on the following format:
================================================

1 -> From an existing input file.

2 -> From keyboard : Enter symbol numbers

3 -> All zeros.

The inputs represent the powers of alpha, valid inputs are -1 to 6.

Enter your choice (1, 2 or 3) : 2
enter Message value, less than -1 stop
Message M(0) = 0
enter Message value, less than -1 stop
Message M(1) = 6
enter Message value, less than -1 stop
Message M(2) = 4

Encoding process: (in Hex)
==================
M[  6] = 6
```

```
M[  5] = 5
M[  4] = 1
M[  3] = 3
M[  2] = 2
M[  1] = 7
M[  0] = 0

Noisy channel modeling by creating the error pattern
====================================================
Enter decimal numbers for error location,
valid values are 0 to 6, -1 to stop

The error values input in Hex form,
valid inputs are 0 to 6.

Error location = 2
Error value    = 5
Error location = 3
Error value    = 6
Error location = -1

Error pattern:
error[6]=0
error[5]=0
```

```
error[4]=0

error[3]=6

error[2]=5

error[1]=0

error[0]=0

Is it the right pattern, if not try again:

1 for Yes, 2 for No : 1

Print the binary representation of the following:

 Codeword   Error pattern Received word (in Hex)

M[ 6]= 6    E[ 6]= 0    R[ 6]= 6
M[ 5]= 5    E[ 5]= 0    R[ 5]= 5
M[ 4]= 1    E[ 4]= 0    R[ 4]= 1
M[ 3]= 3    E[ 3]= 6    R[ 3]= 5
M[ 2]= 2    E[ 2]= 5    R[ 2]= 7
M[ 1]= 7    E[ 1]= 0    R[ 1]= 7
M[ 0]= 0    E[ 0]= 0    R[ 0]= 0

Syndrome calculation results:

============================
```

```
 syndrome[  0] = alpha^ -1
 syndrome[  1] = alpha^  1
 syndrome[  2] = alpha^  6
 syndrome[  3] = alpha^  3

 Select decoding methods :
1.Berlekamp
2.Peterson
 Enter your choice : 2
 Warning: Error pattern may beyond errorcorrection capability,
 since 2 by 2 matrix is non zero !

 Error locator polynomial:
 =========================
       1
  +   alpha^  5 x^  1
  +   alpha^  5 x^  2

 Chien search result: (error locations)
 ======================================
root[  1] =alpha^   2
root[  2] =alpha^   3

 Select error value calculation methods :
```

```
1.Forney
2.Gaussian elimination
 Enter your choice : 1

 Error values
 ============
 error[2] =   5
 error[3] =   6

 Estimated error pattern
 =======================
   alpha^  6 x^  2
 + alpha^  4 x^  3

  Codeword   Error pattern  Received word  Decoded word (in Hex)
 M[  6]=  6   E[  6]=  0     R[  6]=  6     D[  6]=  6
 M[  5]=  5   E[  5]=  0     R[  5]=  5     D[  5]=  5
 M[  4]=  1   E[  4]=  0     R[  4]=  1     D[  4]=  1
 M[  3]=  3   E[  3]=  6     R[  3]=  5     D[  3]=  3
 M[  2]=  2   E[  2]=  5     R[  2]=  7     D[  2]=  2
 M[  1]=  7   E[  1]=  0     R[  1]=  7     D[  1]=  7
 M[  0]=  0   E[  0]=  0     R[  0]=  0     D[  0]=  0
```

Example 3.2

- Given: $m = 4, T = 4$, and $\nu = 2$.
- Berlekamp-Massey's method is selected.
- Gauss-Jordan elimination is selected.
- The inputs and outputs are listed below.

Simulation results of Example 3.2

```
RS CODEC for 3 <= m <= 8 and T <= 20
==========================================
Input bits per symbol m  or  code length N , (m or N):  n
Please select N (N = 2^m -1): 15
Please select T (T <=  7) :4
Generator polynomial:

  alpha^0
+ alpha^2 x^1
+ alpha^8 x^2
+ alpha^13 x^3
+ alpha^14 x^4
+ alpha^13 x^5
+ alpha^8 x^6
+ alpha^2 x^7
+ alpha^0 x^8
```

```
Roots of the generator polynomial:
  alpha^4
, alpha^5
, alpha^6
, alpha^7
, alpha^8
, alpha^9
, alpha^10
, alpha^11

Select K symbols data based on the following format:
==================================================

1 -> From an existing input file.

2 -> From keyboard : Enter symbol numbers

3 -> All zeros.

The inputs represent the powers of alpha, valid inputs are -1 to 14.

Enter your choice (1, 2 or 3) : 3

Encoding process: (in Hex)
```

```
================

M[ 14] = 0

M[ 13] = 0

M[ 12] = 0

M[ 11] = 0

M[ 10] = 0

M[  9] = 0

M[  8] = 0

M[  7] = 0

M[  6] = 0

M[  5] = 0

M[  4] = 0

M[  3] = 0

M[  2] = 0

M[  1] = 0

M[  0] = 0


Noisy channel modeling by creating the error pattern

================================================

Enter decimal numbers for error location,

valid values are 0 to 14, -1 to stop


The error values input in Hex form,

valid inputs are 0 to e.
```

```
Error location = 7
Error value    = 9
Error location = 2
Error value    = a
Error location = -1

Error pattern:
error[14]=0
error[13]=0
error[12]=0
error[11]=0
error[10]=0
error[9]=0
error[8]=0
error[7]=9
error[6]=0
error[5]=0
error[4]=0
error[3]=0
error[2]=a
error[1]=0
error[0]=0
```

```
Is it the right pattern, if not try again:

1 for Yes, 2 for No : 1

Print the binary representation of the following:

 Codeword    Error pattern Received word (in Hex)

M[ 14]=  0    E[ 14]=  0    R[ 14]=  0
M[ 13]=  0    E[ 13]=  0    R[ 13]=  0
M[ 12]=  0    E[ 12]=  0    R[ 12]=  0
M[ 11]=  0    E[ 11]=  0    R[ 11]=  0
M[ 10]=  0    E[ 10]=  0    R[ 10]=  0
M[  9]=  0    E[  9]=  0    R[  9]=  0
M[  8]=  0    E[  8]=  0    R[  8]=  0
M[  7]=  0    E[  7]=  9    R[  7]=  9
M[  6]=  0    E[  6]=  0    R[  6]=  0
M[  5]=  0    E[  5]=  0    R[  5]=  0
M[  4]=  0    E[  4]=  0    R[  4]=  0
M[  3]=  0    E[  3]=  0    R[  3]=  0
M[  2]=  0    E[  2]=  a    R[  2]=  a
M[  1]=  0    E[  1]=  0    R[  1]=  0
M[  0]=  0    E[  0]=  0    R[  0]=  0
```

```
Syndrome calculation results:
=============================
syndrome[  0] = alpha^  7
syndrome[  1] = alpha^ -1
syndrome[  2] = alpha^  1
syndrome[  3] = alpha^ 13
syndrome[  4] = alpha^ -1
syndrome[  5] = alpha^  7
syndrome[  6] = alpha^  4
syndrome[  7] = alpha^ -1

Select decoding methods :
1.Berlekamp
2.Peterson
Enter your choice : 1

Error locator polynomial:
=========================
      1
 +   alpha^ 12 x^  1
 +   alpha^  9 x^  2

Chien search result: (error locations)
======================================
```

```
root[  1] =alpha^   2
root[  2] =alpha^   7

 Select error value calculation methods :
1.Forney
2.Gaussian elimination
 Enter your choice : 2

 Error values
 ============
 error[2] =   a
 error[7] =   9

 Estimated error pattern
 =======================
   alpha^  9 x^  2
 + alpha^ 14 x^  7

  Codeword   Error pattern  Received word  Decoded word (in Hex)
 M[ 14]=  0   E[ 14]=  0     R[ 14]=  0     D[ 14]=  0
 M[ 13]=  0   E[ 13]=  0     R[ 13]=  0     D[ 13]=  0
 M[ 12]=  0   E[ 12]=  0     R[ 12]=  0     D[ 12]=  0
 M[ 11]=  0   E[ 11]=  0     R[ 11]=  0     D[ 11]=  0
 M[ 10]=  0   E[ 10]=  0     R[ 10]=  0     D[ 10]=  0
```

```
M[ 9]= 0   E[ 9]= 0    R[ 9]= 0    D[ 9]= 0
M[ 8]= 0   E[ 8]= 0    R[ 8]= 0    D[ 8]= 0
M[ 7]= 0   E[ 7]= 9    R[ 7]= 9    D[ 7]= 0
M[ 6]= 0   E[ 6]= .0   R[ 6]= 0    D[ 6]= 0
M[ 5]= 0   E[ 5]= 0    R[ 5]= 0    D[ 5]= 0
M[ 4]= 0   E[ 4]= 0    R[ 4]= 0    D[ 4]= 0
M[ 3]= 0   E[ 3]= 0    R[ 3]= 0    D[ 3]= 0
M[ 2]= 0   E[ 2]= a    R[ 2]= a    D[ 2]= 0
M[ 1]= 0   E[ 1]= 0    R[ 1]= 0    D[ 1]= 0
M[ 0]= 0   E[ 0]= 0    R[ 0]= 0    D[ 0]= 0
```

Example 3.3

- Given: $m = 4, T = 3$, and $\nu = 4$.
- Berlekamp-Massey's method is selected.
- Forney's algorithm is selected.
- The inputs and outputs are listed below.

Simulation results of Example 3.3

```
RS CODEC for 3 <= m <= 8 and T <= 20
=============================================
Input bits per symbol m  or  code length N , (m or N):  m
Please select m (3 <= m <= 8): 4
```

```
Please select T (T <=  7) :3
Generator polynomial:

  alpha^0
+ alpha^14 x^1
+ alpha^7 x^2
+ alpha^1 x^3
+ alpha^7 x^4
+ alpha^14 x^5
+ alpha^0 x^6

Roots of the generator polynomial:
  alpha^5
, alpha^6
, alpha^7
, alpha^8
, alpha^9
, alpha^10

Select K symbols data based on the following format:
==================================================

1 -> From an existing input file.
```

```
2 -> From keyboard : Enter symbol numbers

3 -> All zeros.

 The inputs represent the powers of alpha, valid inputs are -1 to 14.

 Enter your choice (1, 2 or 3) : 3

 Encoding process: (in Hex)
 ==========
 M[ 14] = 0
 M[ 13] = 0
 M[ 12] = 0
 M[ 11] = 0
 M[ 10] = 0
 M[  9] = 0
 M[  8] = 0
 M[  7] = 0
 M[  6] = 0
 M[  5] = 0
 M[  4] = 0
 M[  3] = 0
 M[  2] = 0
 M[  1] = 0
```

```
M[  0] = 0

Noisy channel modeling by creating the error pattern
====================================================
Enter decimal numbers for error location,
valid values are 0 to 14, -1 to stop

The error values input in Hex form,
valid inputs are 0 to e.

Error location = 2
Error value    = 4
Error location = 5
Error value    = 7
Error location = 9
Error value    = c
Error location = 4
Error value    = 6
Error location = -1

Error pattern:
error[14]=0
error[13]=0
error[12]=0
```

```
error[11]=0

error[10]=0

error[9]=c

error[8]=0

error[7]=0

error[6]=0

error[5]=7

error[4]=6

error[3]=0

error[2]=4

error[1]=0

error[0]=0

Is it the right pattern, if not try again:

1 for Yes, 2 for No : 1

Print the binary representation of the following:

 Codeword   Error pattern Received word (in Hex)

M[ 14]=  0    E[ 14]=  0    R[ 14]=  0

M[ 13]=  0    E[ 13]=  0    R[ 13]=  0

M[ 12]=  0    E[ 12]=  0    R[ 12]=  0
```

```
M[ 11]=  0    E[ 11]=  0    R[ 11]=  0
M[ 10]=  0    E[ 10]=  0    R[ 10]=  0
M[  9]=  0    E[  9]=  c    R[  9]=  c
M[  8]=  0    E[  8]=  0    R[  8]=  0
M[  7]=  0    E[  7]=  0    R[  7]=  0
M[  6]=  0    E[  6]=  0    R[  6]=  0
M[  5]=  0    E[  5]=  7    R[  5]=  7
M[  4]=  0    E[  4]=  6    R[  4]=  6
M[  3]=  0    E[  3]=  0    R[  3]=  0
M[  2]=  0    E[  2]=  4    R[  2]=  4
M[  1]=  0    E[  1]=  0    R[  1]=  0
M[  0]=  0    E[  0]=  0    R[  0]=  0

Syndrome calculation results:
=============================
syndrome[  0] = alpha^  1
syndrome[  1] = alpha^  5
syndrome[  2] = alpha^  0
syndrome[  3] = alpha^ 13
syndrome[  4] = alpha^ -1
syndrome[  5] = alpha^ 10

Select decoding methods :
1.Berlekamp
```

```
2.Peterson
 Enter your choice : 1

 Error locator polynomial:
 ========================

       1
  +   alpha^ 14 x^  1
  +   alpha^ 12 x^  2
  +   alpha^ -1 x^  3

 Chien search result: (error locations)
 ======================================

 No roots!

 Error number beyond error correction capability!!

 Resend message
```

Figure 3.8 shows an example of running the simulator on PC. In this case $m = 8, T = 4$, all message bits are set to zero, errors are located at position 6, 7, 8, 9, and the error values are 50, 100, 150, 200, respectively. The upper half of the screen displays the message word, encoded word, received word. and decoded word. The generator polynomial, syndromes, error locator polynomial, and error values are shown in the lower one.

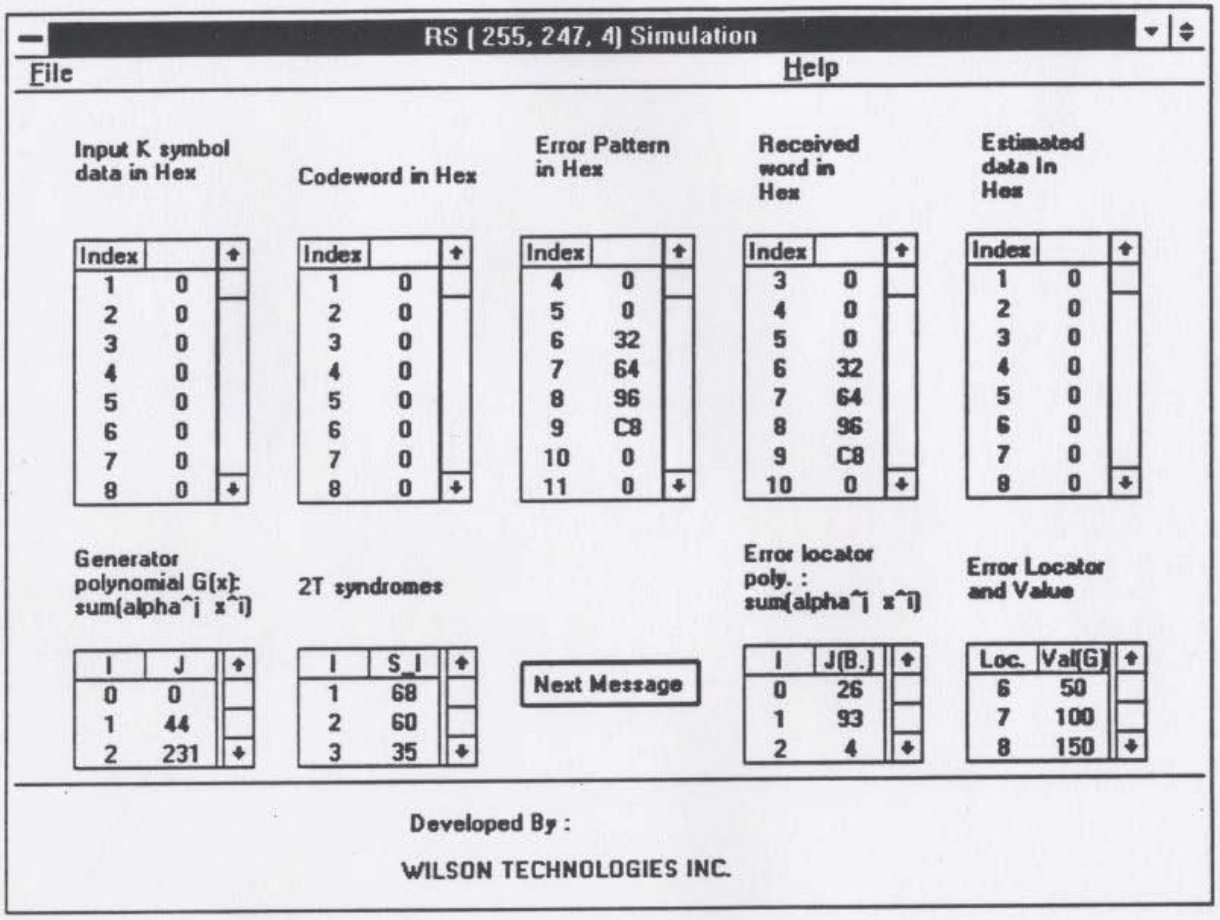

Figure 3.8: Example of RS (255,247,4) simulator

Chapter 4

New Periodicity Algorithm

In [2] and [1], a periodicity algorithm was introduced by Le-Ngoc and Young to eliminate the conventional Chien search method in finding the roots of the error locator polynomial. In this chapter, we prove that the periodicity features which exist between the roots and their coefficients in Galois field. Furthermore, the properties illustrated in Section 4.1 suggest that the equation transform in Young's algorithm [2] [1] is unnecessary. Therefore an improved periodicity algorithm is proposed in Section 4.2.1. In the last section, a new periodicity algorithm using XOR operations to eliminate the index table required for storing the head of the roots chain is discussed.

4.1 Basic Properties

To simplify the notation, let us start with the quadratic equation[1]

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

in $GF(2^m)$. After both sides of Equation(4.1) are divideds by σ_2, the following equation is obtained:

$$1 + \sigma_{y1} x + \sigma_{y2} x^2 = 0 \tag{4.2}$$

After listing the roots of all the possible σ_{y1} and σ_{y2}, σ_{y1}, for all $\sigma_{y2} \in GF(2^m)$, Young and Le-Ngoc found that the roots of Equation(4.2) are not randomly distributed. Therefore, they developed the periodicity algorithm. In this section, it will be shown that periodicity properties exist in the general form of Equation(4.1) because of the internal relationship between roots and their coefficients. Thus the transformation from Equation(4.1) to Equation(4.2) is not necessary.

Suppose $x_1 = \alpha^{i_1}$ and $x_2 = \alpha^{i_2}$ are the roots of Equation (4.1), where α is the primitive element of $GF(2^m)$. The following relations always exist between roots and their coefficients:

$$\begin{aligned} \sigma_1 &= x_1 + x_2 \\ \sigma_2 &= x_1 \cdot x_2 \end{aligned}$$

If we have roots $x_1' = \alpha \cdot x_1 = \alpha^{i_1+1}$ and $x_2' = \alpha \cdot x_2 = \alpha^{i_2+1}$, the corresponding coefficients will be:

$$\sigma_1' = x_1' + x_2' = \alpha \cdot x_1 + \alpha \cdot x_2 = \alpha\sigma_1$$

[1] An error locator polynomial has two forms:
1. $x^2 + \sigma_1 x + \sigma_2 = (x - X_1)(x - X_2)$ or
2. $1 + \lambda_1 x + \lambda_2 x^2 = (1 - X_1 x)(1 - X_2 x)$
where $\lambda_1 = \sigma_1$ and $\lambda_2 = \sigma_2$

$$\sigma_2' = x_1' \cdot x_2' = \alpha \cdot x_1 \cdot \alpha \cdot x_2 = \alpha^2 \sigma_2$$

In Galois field, multiplication of two operands can be easily realized by modulo-$N(N = 2^m - 1)$ addition of their powers. Therefore, we expect when the powers of two roots increase by one individually, the powers of σ_1 and σ_2 will increase by one and two respectively. All the increment operations are modulo N. The above mentioned property is further illustrated by Table 4.1. Table 4.1 lists all the roots of Equation (4.1) with respect to all possible values of σ_1 and σ_2 in the case of $m = 3$ and primitive polynomial $p(x) = x^3 + x + 1$.

Table 4.1: The root table over $GF(2^3)$ and $p(x) = x^3 + x + 1$

σ_1 \ σ_2	α^0	α^1	α^2	α^3	α^4	α^5	α^6
α^0	x	2,6	4,5	x	1,3	x	x
α^1	x	x	x	0,3	5,6	x	2,4
α^2	x	3,5	x	x	x	1,4	0,6
α^3	2,5	0,1	x	4,6	x	x	x
α^4	x	x	3,6	1,2	x	0,5	x
α^5	1,6	x	x	x	0,4	2,3	x
α^6	3,4	x	0,2	x	x	x	1,5

Each element in Table 4.1 represents for the power of a Galois field element. For example, the bottom right corner element in Table 4.1 is [1,5], which means that, for Equation (4.1), when $\sigma_1 = \alpha^6$ and $\sigma_2 = \alpha^6$, the two roots are α^1 and α^5. If Equation (4.1) has no root for given values of σ_1 and σ_2, it is marked as [x].

As shown in Table 4.1, when the root pair of [1,5] becomes [2,6], its corresponding σ's will change from (α^6, α^6) to (α^0, α^1), i.e.$(\alpha^{6+1}, \alpha^{6+2})$. If we keep increasing both of the root power values by 1, we can get a root chain shown below:

$[1,5] \longrightarrow [2,6] \longrightarrow [0,3] \longrightarrow [1,4] \longrightarrow [2,5] \longrightarrow [3,6] \longrightarrow [0,4] \longrightarrow [1,5]$.

The chain will be closed because of modulo-N addition and there must always be N distinct elements in the chain.

According to the relationship between the roots and their corresponding coefficients, a list of (σ_1, σ_2) pairs of the above root chain is shown below:

$(\alpha^6, \alpha^6) \longrightarrow (\alpha^0, \alpha^1) \longrightarrow (\alpha^1, \alpha^3) \longrightarrow (\alpha^2, \alpha^5) \longrightarrow (\alpha^3, \alpha^7) \longrightarrow (\alpha^4, \alpha^2) \longrightarrow (\alpha^5, \alpha^4) \longrightarrow (\alpha^6, \alpha^6)$.

Due to the uniqueness of the roots, the (σ_1, σ_2) pairs also form a closed chain and it also has N distinct elements. In the (σ_1, σ_2) pair chain, σ_1 increases by one, while σ_2 increases by two. Thus, if we know one element in the root chain and its corresponding σ_1 and σ_2, the other roots in the same chain can be obtained by using shifting operations.

We assumed that $(\sigma_1, \sigma_2) = (\alpha^0, x)$ is the leader of the chain since each (σ_1, σ_2) pair chain passes through $\sigma_1 = \alpha^0$ and different chains contain $(\sigma_1, \sigma_2) = (\alpha^0, \alpha^j)$ with distinct j, where $j \in [0, N-1]$.

There are N number of σ pair chains altogether, where $N = 2^{m-1}$. Each chain has N σ pairs. Because there are altogether $N(N-1)/2$ possible root pairs and they are distributed in $(N-1)/2$ number of root chains, all those (σ_1, σ_2) pairs with roots will form $(N-1)/2 = 3$ chains which we call root chains. The other $(N+1)/2 = 4$ chains without roots are called no-root chains.

Another useful property is the addition of the powers of two roots (modulo-N) equals the power of σ_2. These properties are summarized as follows:

1. There are N numbers of chains in total, of which $(N-1)/2$ chains contain

roots, while the other $(N+1)/2$ chains contain no roots.

2. There are N distinct elements in each chain, and each element in the chain repeats with a period of N, where $N = 2^m - 1$.

3. In each (σ_1, σ_2) pair chain, powers of σ_1's increase by one and σ_2's increase by two. In the root chains, both of the powers of the two corresponding roots increase by one.

4. Each (σ_1, σ_2) pair chain contains a (σ_1, σ_2) pair of (α^0, α^j) with unique j, where $j \in [0, N-1]$.

5. The addition of the powers of the two roots (modulo-N) equals the power of the σ_2 in the Equation (4.1).

In the above discussion, we ignore the case $\sigma_1 = 0$ or $\sigma_2 = 0$, because in RS decoding procedure the quadratic error locator polynomial must have two distinct non-zero error locators.

4.2 Improved Periodicity Algorithm

4.2.1 Improved Periodicity Algorithm

From the properties described in the previous section, an improved periodicity algorithm for solving the quadratic equation $x^2 + \sigma_1 x + \sigma_2 = 0$ in $GF(2^m)$ can be obtained below.

First, for any given Galois field, establish an index table containing the head of each root chain and its corresponding (σ_1, σ_2) pairs. Since the corresponding

(σ_1, σ_2) pairs of each root chain pass through a $(\sigma_1, \sigma_2) = (\alpha^0, \alpha^j)$, the table of the roots of

$$x^2 + x + \alpha^j = 0, j = 0, 1, \cdots, 2^m - 2 \tag{4.3}$$

needs to be found. Each entry in this table is only one of the roots of Equation (4.3) and the access to these entries can be obtained by the index j. For example, the index table for $m = 3$ and $p(x) = x^3 + x + 1$ is given in Table 4.2. Here, x stands for no root chain. The flowchart for the construction of the index table is shown in Figure 4.1.

Table 4.2: The index table over $GF(2^3)$ and $p(x) = x^3 + x + 1$

α^j	α^0	α^1	α^2	α^3	α^4	α^5	α^6
$I[j]$	x	2	4	x	1	x	x

Now, the improved periodicity algorithm based on the preprocessed index table is given as follows:

1. For any given σ_1 and σ_2, determine the chain to which they belong. Let $\sigma_1 = \alpha^{i_1}$ and $\sigma_2 = \alpha^{i_2}$, then we try to map the $(\sigma_1, \sigma_2) = (\alpha^{i_1}, \alpha^{i_2})$ into $(\alpha^0, \alpha^{i_{map}})$ with the mapping function given by:

$$i_{map} = (i_2 - 2i_1) \pmod{N}$$

2. Look up the index table to get the root value $I[i_{map}]$

3. If it happens to belong to a no-root chain, then the given error locator polynomial has no root.

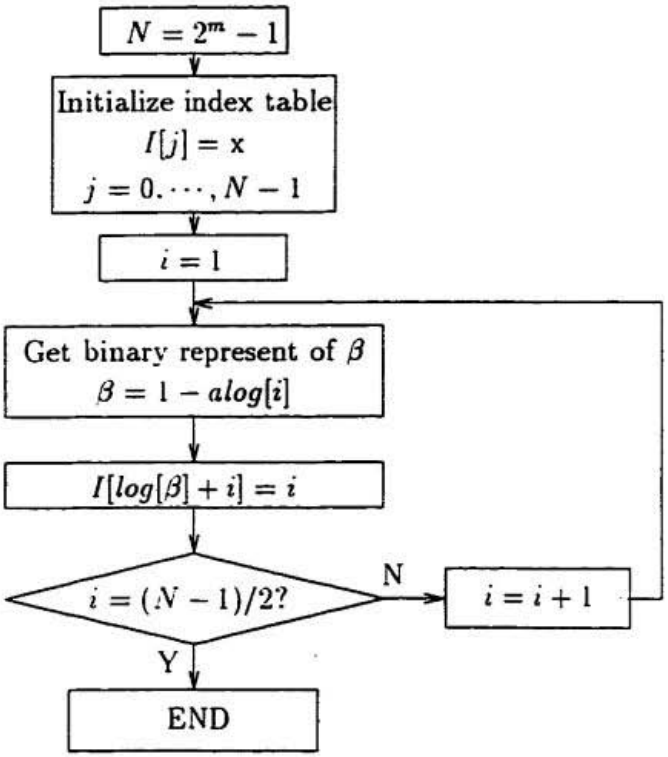

Figure 4.1: The construction of the root index table over $GF(2^m)$

4. Otherwise, if it is on the root chain, find the actual roots according to the relationship among the coefficients of each chain. Assume that the roots can be expressed as $X_1 = \alpha^f$ and $X_2 = \alpha^s$. Then the exponent of one root X_1 can be obtained by:

$$f = (I[i_{map}] + i_1) \pmod{N}$$

where i_1 is the exponent of σ_1.

5. By applying property (5), the other root $X_2 = \alpha^s$ can be found by:

$$s = (i_2 - f) \pmod{N}$$

where i_2 is the exponent of σ_2.

Figure 4.2 shows the flow chart of the periodicity algorithm.

4.2.2 The Okano-Imai Method

In this section, we introduce the Okano-Imai ROM root search method [42] for comparison.

Let us consider the quadratic polynomial over $GF(2^m)$:

$$\Sigma(x) = x^2 + \sigma_1 x + \sigma_2 \tag{4.4}$$

The appropriate linear translation

$$x = \sigma_1 y, \quad \sigma_1 \neq 0 \tag{4.5}$$

will yield

$$\Sigma'(y) = y^2 + y + c, \quad c = \sigma_2/\sigma_1^2 \tag{4.6}$$

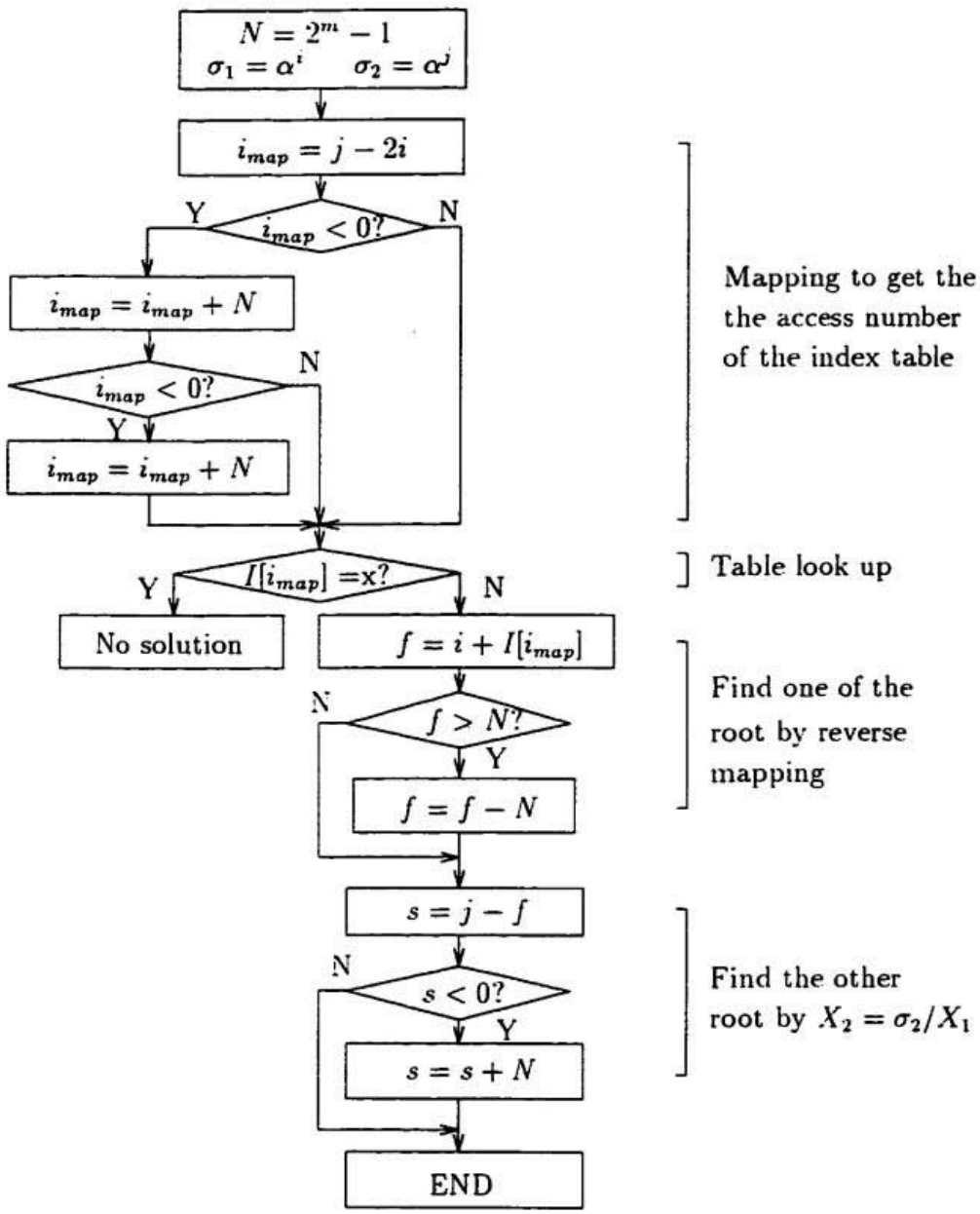

Figure 4.2: The flow chart of the periodicity algorithm

Since the polynomial $\Sigma'(y)$ has just one coefficient c, the size of the table for finding each root of $\Sigma'(y)$ is $N \cdot m$ bits.

After one of the roots of $\Sigma'(y)$ is obtained, the root of $\Sigma(x)$ can be calculated using Equation (4.5). The other root can be obtained by applying $X_2 = \sigma_2/X_1$. After we investigate the flowchart of the Okano-Imai method, we find that it is a similar algorithm to the improved periodicity algorithm.

4.3 New Periodicity Algorithm

The periodicity algorithm described before requires a root index table of equation

$$x^2 + x + \sigma_2 = 0 \tag{4.7}$$

which needs a $(2^m - 1) \times m$-bit memory. This table can now be eliminated by the following direct solution method.

Any element x of $GF(2^m)$ can be represented as:

$$x = x_{m-1}\beta^{m-1} + x_{m-2}\beta^{m-2} + \cdots + x_1\beta + x_0$$

where $\beta^0, \beta^1, \cdots, \beta^{m-1}$ is the standard basis of $GF(2^m)$ and $x_i \in GF(2)$. Hence, x can also be denoted as vector $(x_{m-1}, x_{m-2}, \cdots, x_0)$ under the basis $\beta^0, \beta^1, \cdots, \beta^{m-1}$. The Equation (4.7) can be rewritten as follows:

$$(x_{m-1}, x_{m-2}, \cdots, x_0)^2 + (x_{m-1}, x_{m-2}, \cdots, x_0) + (c_{m-1}, c_{m-2}, \cdots, c_0) = 0 \tag{4.8}$$

where $\sigma_2 = c_{m-1}\beta^{m-1} + \cdots + c_0\beta^0$.

To further illustrate the direct solution method, we limit our discussions to $GF(2^8)$, $p(x) = x^8 + x^7 + x^2 + x + 1$ and standard basis $\alpha^0, \cdots, \alpha^7$. However,

the method can be extended to other cases accordingly. Based on Galois field properties, the square of the element of the Galois field equals the sum of the square of each coordinate, which can be written as:

$$(x_7, x_6, \cdots, x_0)^2 = x_7\alpha^{14} + x_6\alpha^{12} + \cdots + x_0$$

$$= x_7(\alpha^6 + \alpha^5 + \alpha^4 + \alpha^3 + \alpha^1) + x_6(\alpha^7 + \alpha^6 + \alpha^4 + \alpha^3 + \alpha^2 + 1)$$

$$+x_5(\alpha^7 + \alpha^4 + \alpha^2 + 1) + x_4(\alpha^7 + \alpha^2 + \alpha + 1) + x_3\alpha^6 + x_2\alpha^4 + x_1\alpha^2 + x_0$$

$$= \alpha^7(x_6 + x_5 + x_4) + \alpha^6(x_7 + x_6 + x_3) + \alpha^5 x_7 + \alpha^4(x_7 + x_6 + x_5 + x_2)$$

$$+ \alpha^3(x_7 + x_6) + \alpha^2(x_6 + x_5 + x_4 + x_1) + \alpha^1(x_7 + x_4) + \alpha_0(x_6 + x_5 + x_4 + x_0) \quad (4.9)$$

Equation (4.9) can be rewritten in matrix form:

$$(x_7, x_6, \cdots, x_0)^2 = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix} \quad (4.10)$$

The quadratic equation (4.8) becomes

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix} = \begin{bmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} \quad (4.11)$$

After performing some matrix operations, we get

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_7 \\ x_6 \\ x_5 \\ x_4 \\ x_3 \\ x_2 \\ x_1 \\ x_0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} c_7 \\ c_6 \\ c_5 \\ c_4 \\ c_3 \\ c_2 \\ c_1 \\ c_0 \end{bmatrix} \quad (4.12)$$

As we can see, $x_7, x_6, \cdots, x_1$ can be determined from $c_7, c_6, \cdots, c_0$ directly. The two roots of Equation(4.7) are $(x_7, x_6, \cdots, x_1, 0)$ and $(x_7, x_6, \cdots, x_1, 1)$. The solution only exists when $c_7 + c_6 + c_5 + c_4 + c_3 + c_2 + c_1 = 0$. An implementation circuit solving x from Equation (4.7) can be shown in Figure 4.3. It is clear that the index table is not required as shown in the periodicity algorithm. This table elimination advantage can be also utilized for Okano-Imai's ROM method [42].

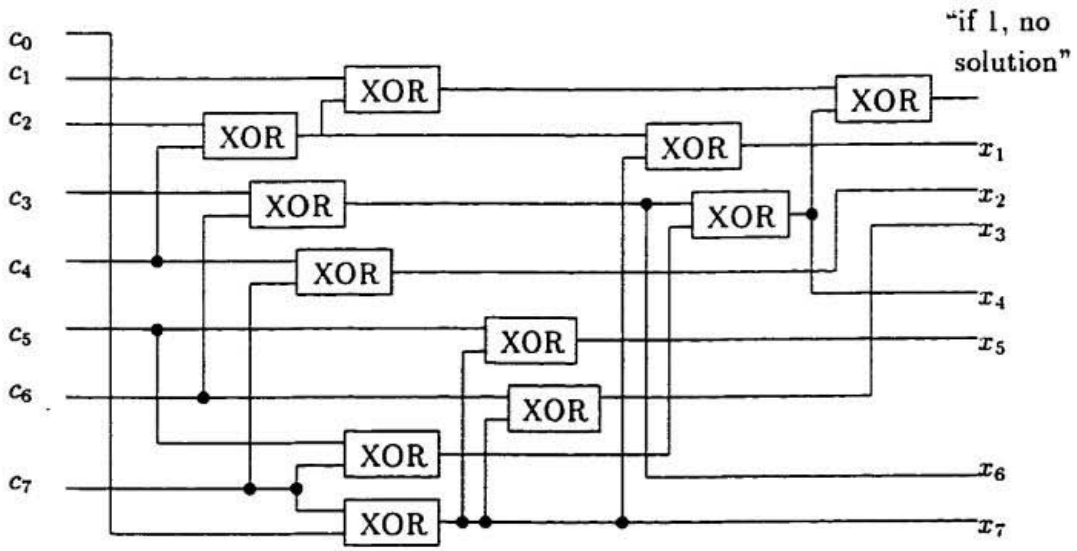

Figure 4.3: Hardwired connections for finding roots of $x^2 + x + \sigma_2 = 0$ over $GF(2^8)$

A corresponding flowchart for the software implementation is shown in Fig-

ure 4.4.

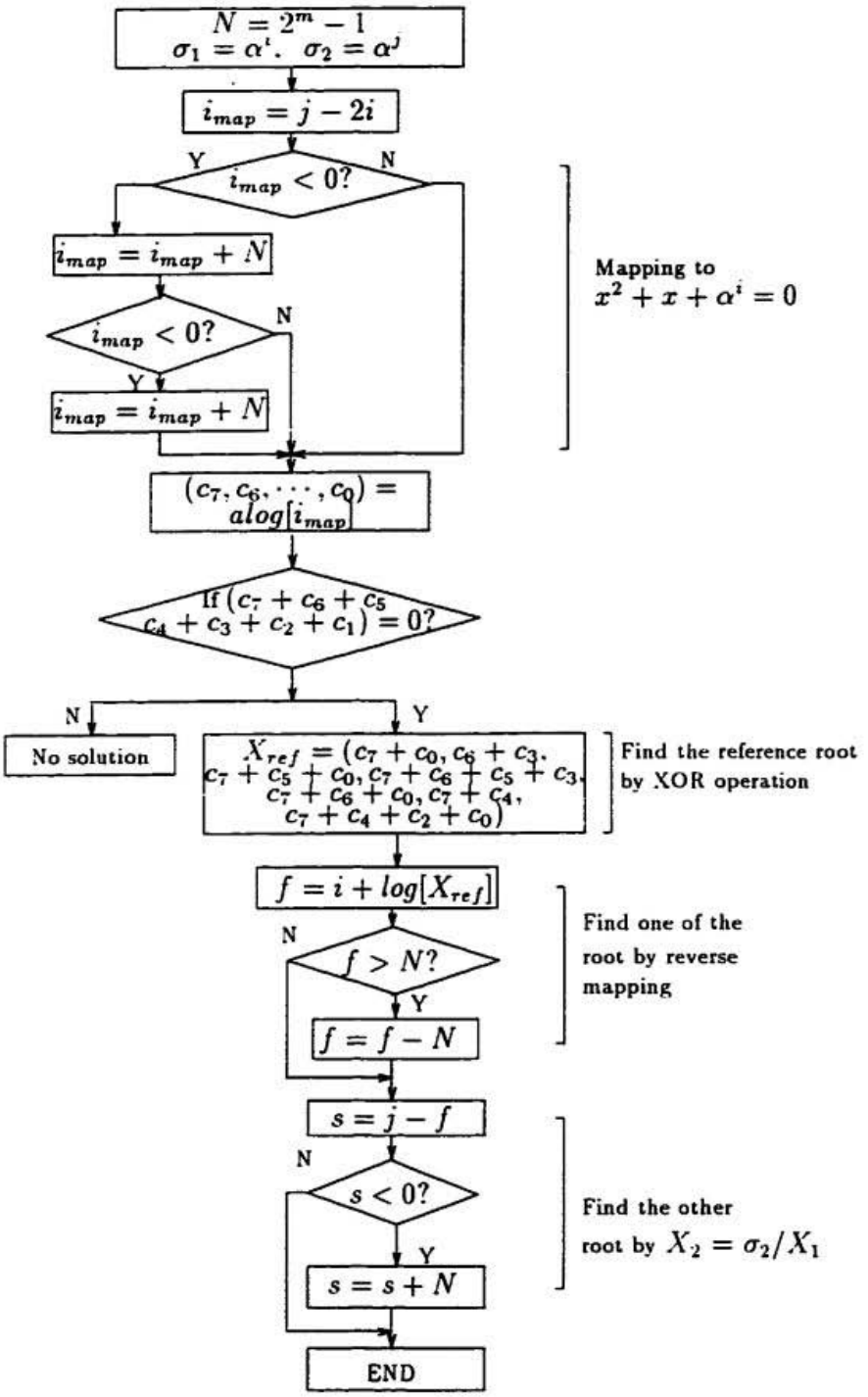

Figure 4.4: The flowchart of new periodicity algorithm for $m = 8$ and $p(x) = x^8 + x^7 + x^2 + x + 1$

Chapter 5

Comparison of Different Algorithms

In this chapter, different decoding schemes are compared. Solving the error locator polynomial can be achieved by either the Peterson method or Berlekamp-Massey algorithm. A time comparison of these two algorithms is made based on the simulation program. The comparison of different techniques of determining error values, Gauss-Jordan elimination and the Forney algorithm are also made. At last, five different root search techniques are compared for time and ROM complexity. These include Chien search, look-up table, binary-decision fast Chien search [57] [41], the periodicity algorithm and new periodicity algorithm.

5.1 The Peterson Method vs the Berlekamp-Massey Algorithm

In Chapter 2, we discussed the Peterson-Gorenstein-Zierler decoder and the Berlekamp-Massey algorithm for finding error locator polynomials. Now, run our simulation

program on a UNIX workstation Pico[1]. Assume that $m = 8$ and the actual error number ν equal to T and let the error occur at $1, 2, \cdots, T$, respectively. The results are plotted in Figure 5.1.

It is clear that when $T \leq 6$, the execution time of Peterson's algorithm is slightly better than that of the Berlekamp-Massey algorithm. However, when $T > 6$, the execution time of the Peterson algorithm exponentially increases, while the execution time of the Berlekamp-Massey algorithm linearly increases. Hence, for any $T > 6$, the Berlekamp-Massey algorithm should be used to reduce the decoding time.

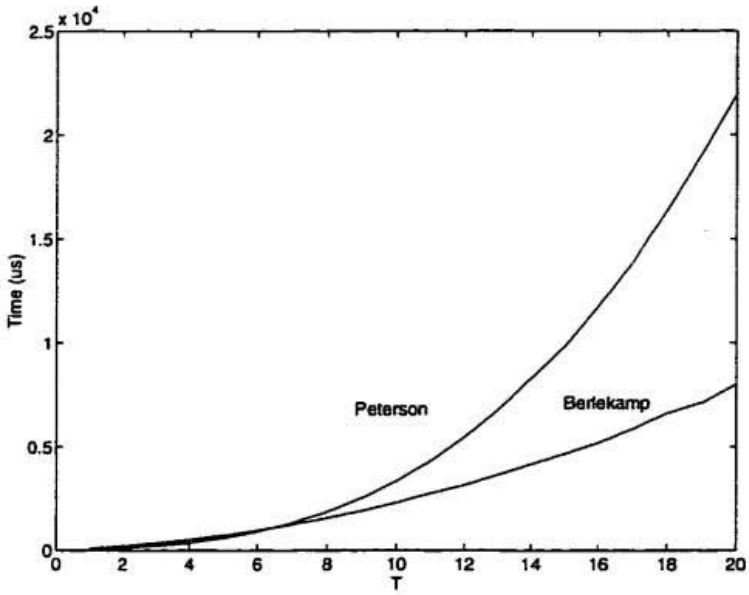

Figure 5.1: Time comparison between the Peterson and Berlekamp-Massey method for $m = 8$

[1]Pico is a 5000/120 DEC station under OS Ultrix 4.4, in C-CAE, Faculty of Engineering and Applied Science, Memorial University of Newfoundland.

5.2 Gauss-Jordan Elimination vs Forney's Algorithm

As shown in Chapter 2, after having determined the error locator by using Chien search, the error values can be obtained by using either the Gauss-Jordan elimination or Forney algorithm. Simulations were run under the same running conditions stated in Section 5.1. The results are plotted as shown in Figure 5.2. It is shown that when $T \leq 10$ the Gauss-Jordan method is a preferred method, while when $T > 10$ the Forney algorithm should be used to cut down decoding time.

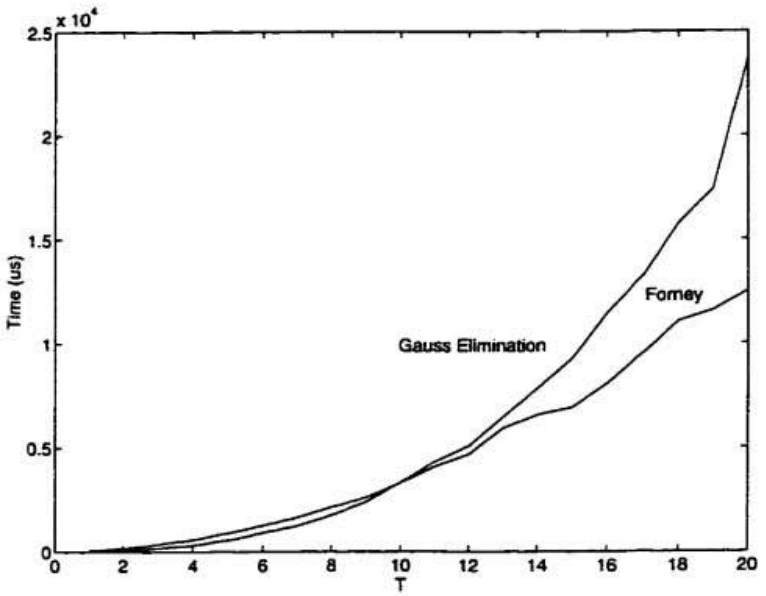

Figure 5.2: Time comparison between the Gauss-Jordan elimination and Forney method for $m = 8$

5.3 Comparison Among Different Root Search Methods

In Chapter 4, the new periodicity algorithm was presented for solving the error locators. In Section 2.4, the traditional Chien search method was also discussed. Recently, two other methods for finding roots of Galois field polynomial have appeared in the literature. One is the Okano-Imai ROM method [42] and the other is the binary-decision fast Chien search proposed by Shayan, Le-Ngoc, and Bhargava [41]. Another conventional way to evaluate the roots of polynomials over $GF(2^m)$ is to use a look-up table. The Okano-Imai ROM method is identical to the improved periodicity algorithm and has been introduced in Section 4.2.2. The last two methods will be briefly introduced before we proceed with further comparisons of all these methods.

5.3.1 The Look-Up Table Method

Let us again consider the following polynomial:

$$\Sigma(x) = x^2 + \sigma_1 x + \sigma_2 \tag{5.1}$$

In the look-up table method, a table which contains the mapping relationship between all the possible values of the coefficients and the corresponding error locators is constructed in advance. Since coefficients of the polynomial contain all the information about the roots, we can use these coefficients as addresses and the roots of error locator polynomial as entries of a look-up table. This approach is the fastest, however, it also needs a large look-up table of size $O(N^2)$ bytes (N is the length of $GF(2^m)$). Therefore, in most cases, it is not feasible for many applications,

particularly high order finite fields such as $GF(2^8)$.

5.3.2 Binary-Decision Fast Chien Search

This approach is also called segmented search algorithm [41][57]. The algorithm can be explained as follows.

Let us consider the $GF(2^m)$. We can divide the field into l equal segments, where $l = 2^i$. For each combination of two coefficients in Equation (5.1), a Segment-Identifier (SI) value is assigned which indicates the segment with most of roots. Each SI entry in the table is i bits long. Hence the size of SI table is $N^2 \cdot i$ bits.

To solve for the roots of the polynomial, first the two coefficients of the polynomial are applied to the SI table and the segment which has most of the errors is identified. Then a Chien search is applied to that segment. As soon as the first root is found, the second root can be calculated by the formula $\sigma_2 = X_1 X_2$. Clearly, this approach only takes approximately $1/i$ of Chien search time.

5.3.3 Comparison Among Different Root Search Methods

The comparisons of execution times and memory sizes of the Chien search, binary-decision search, look-up table method, the periodicity algorithm, and the new periodicity algorithm are made for decoding double error correcting RS codes of any length, which can be shown in Table 5.1. For comparison, we consider the worst case only. In Chien search's worst case, the polynomial should be evaluated and compared with zero, $2^m - 1$ times. In the segmented search algorithm, we consider that the Galois field is divided by 2. In the worst case, the half Galois field needs

to be searched. All the other methods go through the longest path in the execution process.

Time estimation is made based on the clock period of the Intel microprocessor. Information about the clock cycles each operation takes can be found in [59]. In the last two rows of Table 5.1, we list the CPU cycles required by the serial and parallel implementations of the new periodicity algorithm. The serial implementation refers to go through the flowchart of Figure 4.4 one operation after another with one processor only. The parallel implementation time is estimated by assuming that multiple processors exist and no communication overhead between the processors. The number of the processors will depend on the code length and the chosen primitive polynomial. For the case introduced in Section 4.3, 7 adders are required. From Table 5.1, we can see that with parallel implementation, the new periodicity algorithm requires no memory space and only 62 CPU cycles inferior to the look-up table method. It is efficient in terms of both memory size and decoding time. In fact, since no ROM is required and only additions involve in the new periodicity algorithm, LSI architecture can be easily developed to achieve the parallel processing and thus high speed decoding.

Table 5.1: CPU cycles and memory size of different root search methods

	Worst-case number of CPU cycles	Look-up memory size bytes
Chien search	$48N$	0
Binary-decision fast Chien search	$24N + 8$	$O(N^2)$
Look-up table	10	$O(N^2)$
Periodicity algorithm	49	$O(N)$
New Periodicity algorithm - serial	258	0
New Periodicity algorithm - parallel	72	0

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Communication channels are normally affected by various kinds of noises, interferences, distortions and fadings. As a result, errors occur in data transmission. Forward error correction (FEC) is a common method to counteract these problems.

Due to its optimal structure, highest correction capability and most of all its ability of correcting both random and burst errors, Reed-Solomon codes are widely used today for many applications. As the demand for error control coding rapidly increases, for example in cellular telephone systems, satellite communications, high definition television, compact disc etc., the understanding of RS codes is becoming essential. Hence, in this thesis, we examine two major aspects of RS codes:

1. A general purpose Reed-Solomon codec simulator

2. New periodicity algorithm

6.1.1 RS Codec Simulator

The overview of RS codes presented in Chapter 2 introduced and discussed the encoding and decoding processes. In the encoding process, the emphasis was placed on systematic encoder with the use of self-reciprocal generator polynomial. While in the decoding process, the error locator polynomial was obtained by using three methods: the Peterson-Gorenstein-Zierler, Berlekamp-Massey and improved Berlekamp-Massey algorithms. After having determined the error locator polynomial, Chien search was used to find the roots, i.e. error locators. The error values were calculated by two different approaches: the Gauss-Jordan elimination and Forney methods.

Chapter 2 described the implementation of RS codec simulator which is ideally suited for teaching and the study of a wide range of RS codes.

The codec simulator starts encoding the user's data message into a systematic codeword by entering the symbol length m from 3 to 8 bits and the error correcting capability T of up to 20. The data message can be selected as: all zeros, or an input file, or entering through keyboard. The noisy channel is modeled by an error pattern. This error pattern can either be entered by the user with the arbitrary weight or generated by an external program, which provides all possible error positions. Both the Peterson and Berlekamp-Massey algorithms are available for users to construct the error locator polynomial. The Chien search technique is used to obtain the error locators which are the roots of the error locator polynomial. After having determined the error locators, the error values can be obtained by using Gauss-Jordan elimination or Forney's algorithm. A decoded word is estimated if

$\nu < T$ and the decoded word (or estimated codeword) is a codeword. However, when $\nu \geq T$ the decoded word must be checked by dividing it by $G(x)$. If the remainder is zero then $\nu = T$ or the error pattern is a codeword. The latter case may happen when $\nu > T$ and the decoder is blind. If the decoded word is not divisible by $G(x)$, the generator polynomial, then $\nu > T$ and the received word will not be corrected. It will be passed to the data sink untouched because the decoder will not correct the errors properly.

The RS codec simulator was implemented in C language and can be run under Microsoft Windows and UNIX operating system. A friendly and easy-to-use graphical user interface is provided for PC.

6.1.2 New Periodicity Algorithm

Using analyses, in Chapter 4, we have shown and generalized the periodicity algorithms conceived by Le-Ngoc and Young [1][2] as follows:

(i) The new form of error locator polynomial,

$$\Sigma''(x) = \Sigma(x)/\sigma_2 = 1 + \sigma_{y1}x + \sigma_{y2}x^2$$

introduced by Young is not necessary and therefore this also means an decrease in decoding time.

(ii) Using the standard Peterson's error locator polynomial form:

$$\Sigma(x) = x^2 + \sigma_1 x + \sigma_2$$

it was clearly demonstrated that there is a specific relationship between the roots ($X_1 = \alpha^{i_1}$ and $X_2 = \alpha^{i_2}$) and the corresponding coefficients ($\sigma_1 = \alpha^{j_1}$ and $\sigma_1 = \alpha^{j_1}$).

The new periodicity algorithm states:

1. If The root power values (i_1 and i_2) of X_1 and X_2 increase by 1, then the power value of σ_1 also increases by 1, and the power value of σ_2 increases by 2. Thus they form a close root chain.

2. There are $(N-1)/2$ root chains and $(N+1)/2$ no root chains.

3. There are always $N = 2^m - 1$ distinct roots in a root chain.

4. Whenever the coefficients (σ_1, σ_2) are known, one of the two roots can be determined by mapping the current $(\sigma_1, \sigma_2) = (\alpha^{i_1}, \alpha^{i_2})$ into the index σ pair $(\alpha^0, \alpha^{i_{map}})$ with $i_{map} = (i_2 - 2i_1) \pmod N$. This is the reason that an index table of $N \times m$ bit memory is required [1] [2] to store the roots of $x^2 + x + \alpha^{i_{map}} = 0$. However, by using the direct solution method, we have eliminated the $N \cdot m$ bit memory space. This is one of the major contributions of this thesis.

5. The second root is easily obtained by $X_2 = \sigma_2 / X_1$.

(iii) Our analytical derivation also proves that the periodicity algorithm is valid for all values of m.

Chapter 5 has devoted to the comparison of different algorithms with respect to the decoding time. For determining error locator polynomial, the Berlekamp-Massey algorithm is preferably used when the error correcting capability $T > 6$, while the Peterson's method is recommended when $T \leq 6$.

For finding error values, the Forney's algorithm is normally suggested to be used

when $T > 10$, whereas the Gauss-Jordan elimination method is recommended when $T \leq 10$.

In searching for roots of the error locator polynomial, one normally trades off between execution time and memory sizes in various techniques. Table 5.1 in Chapter 5 listed the number of CPU cycles and memory sizes among the different methods. It is clear that with parallel hardware implementation, the new periodicity algorithm requires no memory space as Chien search and only 62 CPU cycles inferior to the look-up table method. It is in fact the most optimal algorithm so far as we know. It can be predicted that the new algorithm will take the place of Chien search, binary-decision (fast Chien search), look-up table, and Okano-Imai method in the future VLSI design. This is the main contribution in this thesis.

6.2 Future Work

As mentioned from the start of the thesis, our work concentrated on algebraic decoding. However, there are other kinds of decoding, such as transform decoding and time domain decoding. Therefore, one direction for future work would be to include these decoding methods into the RS codec simulator.

Since the new periodicity algorithm is an optimal one in both memory sizes and decoding time for double error correcting RS codes, our next step should be further developing a algorithm to adapt to larger error-correting capabilities. Three approaches can be considered to solve this problem. One way is to decompose the high order error locator polynomial into quadratic ones. Then the new periodicity algorithm will still be effective. Another approach is to combine Chien search with

the periodicity algorithm to solve the polynomial. The third one is to develop a new algorithm which can determine the roots directly from the coefficients by using some memory,

The new periodicity algorithm has no memory requirement and only additions. Therefore, it is easy to convert to hardware implementation. However, a more efficient circuit design will be involved in order to fabricate the algorithm into VLSI chip.

Bibliography

[1] S. Le-Ngoc and Z. Young, "An Approach to Double Error Correcting Reed-Solomon Decoding without Chien Search", Proceedings of Midwest Symposium on Circuits and Systems, Detroit, Michigan, U.S.A, pp. 534-537, Aug. 1993.

[2] Z. Young, "A Reed-Solomon Code Simulator and Periodicity Algorithm," *M.Eng. thesis*, Memorial University of Newfoundland, St.John's, Newfoundland, Canada, 1994, pp. 87-111.

[3] S. Le-Ngoc, Y. Ye, and T. Banerjee, "A Novel Approach to Quadruple Error Correcting Reed-Solomon Decoding without Chien Search". Submitted to Seabright Corporation Limited, (Patent Pending), Memorial University of Newfoundland, July, 1994.

[4] S.B. Wicker and V.K. Bhargava, *Reed-Solomon Codes and their Applications*, New York: The Institute of Electrical and Electronics Engineers, Inc., 1994, pp. 8-12.

[5] Y.R. Shayan, T. Le-Ngoc, "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.

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

[7] S.B. Wicker and V.K. Bhargava, *Reed-Solomon Codes and their Applications*, New York: The Institute of Electrical and Electronics Engineers, Inc., 1994, pp. 175-204.

[8] S.B. Wicker and V.K. Bhargava, *Reed-Solomon Codes and their Applications*, New York: The Institute of Electrical and Electronics Engineers, Inc., 1994, pp. 272-291.

[9] R.C. Bose and D.K. Ray-Chaudhuri, "On a Class of Error Correcting Binary Group Codes," *Information and Control*, vol.3, pp. 68-79, Mar. 1960.

[10] R.C. Bose and D.K. Ray-Chaudhuri, "Further Results on Error Correcting Binary Group Codes," *Information and Control*, vol.3, pp. 279-290, Sept. 1960.

[11] A. Hocquenghem, "Codes correcteurs d'erreurs," *Chiffres*, vol.2, pp. 147-156, 1959.

[12] I.S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," *J. Soc. indust. Appl. Math.*, vol.8, pp. 300-304, 1960.

[13] R.E. Blahut, *Theory and Practice of Error Control Codes*, Reading, Mass: Addison-Wesley, 1983.

[14] S.B. Wicker and V.K. Bhargava, *Reed-Solomon Codes and their Applications*, New York: The Institute of Electrical and Electronics Engineers, Inc., 1994, pp. 2-6.

[15] M.A. Tsfasman, S.G. Vladut, and T. Zink, "Modular Curves, Shimura Curves and Goppa Codes Which Are Better Than the Varshamov-Gilbert Bound," *Mathematische Nachrichten*, no.109, pp. 21-28, 1982.

[16] D.C. Gorenstein, and N. Zierler, "A Class of Error-Correcting Codes in p^m Symbols," *J. Soc. Indust. Appl. Math.* vol.9, pp. 207-214, 1961.

[17] R.E. Blahut, "Transform Techniques for Error Control Codes," *IBM Journal of Research and Development*, vol.23, pp. 299-315, 1979.

[18] W.W. Peterson, "Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes," *IRE Trans. Inf. Theor.*, IT-16, pp. 459-470, Sept. 1960.

[19] W.W. Peterson, *Error-Correcting Codes*, Cambridge, Mass: The M.I.T. Press, 1961.

[20] I.F. Blake, *Algebraic Coding Theory: History and Development*, Dowden: Hutchingon & Ross Inc., 1973.

[21] R.T. Chien, "Cyclic Decoding Procedures for Bose-Chaudhuri-Hocquenghem Codes," *IEEE Trans. Inf. Theor.* IT-10, pp. 357-363, Oct. 1964.

[22] G.D. Forney, "On Decoding BCH Codes", *IEEE Trans. Inf. Theor.* IT-11, pp.549-557, Oct. 1965.

[23] E.R. Berlekamp, "Nonbinary BCH Decoding," paper presented at the 1967 International Synposium on Information Theory, San Remo, Italy.

[24] E.R. Berlekamp, *Algebraic Coding Theory*, New York: McGraw-Hill, 1968.

[25] J.L. Massey, "Shift-Register Synthesis and BCH Decoding," *IEEE Trans. Inf. Theor*, IT-15, pp. 122-127, Jan. 1969.

[26] A.M. Michelson, and A.H. Levesque, *Error-Control Techniques for Digital Communication*, John Wiley & Sons, New York, 1984.

[27] Y. Sugiyama, Y. Kasahara, S. Hirasawa, and T. Namekawa, "A MEthod for Solving Key Equation for Goppa Codes," *Information and Control*, vol.27, pp. 87-99, 1975.

[28] H.M. Shao, T.K. Truong, L.J. Deutsch, J.H. Yuen, and I.S. Reed, "A VLSI Design of a Pipeline Reed-Solomon Decoder," *IEEE Trans. Comput.*, vol.C-34, pp.393-403, May 1985.

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

[30] C.M. Rader, "Discrete Convolution via Mersenne Transforms," *IEEE Trans. Comput.*, vol.C-21, pp. 1269-1273, Dec. 1972.

[31] W.C. Gore, "Transmitting Binary Symbols with Reed-Solomon Code," Johns-Hopkins, EE report, no. 72-5, Apr. 1973.

[32] A.A. Michelson, "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.

[33] A.A. Michelson, "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.

[34] R.L. Miller, T.K. Truong, em et al, "Efficient Program for Decoding the (255, 233) Reed-Solomon Code over $GF(2^m)$ with Both Errors and Erasures Using Transform Decoding," *IEE Proc.*, vol.127, Pt.E, no.4, pp 136-142, Jul. 1980.

[35] I.S. Reed, R.A. Scholtz, T.K. Truong, and L.R. Welch, "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.

[36] D.L. Whiting, "Bit-Serial Reed-Solomon Decoder in VLSI," *Ph.D. Thesis,* School of Engineering and Applied Science, University of California, Los Angeles, 1985.

[37] T. Yaghoobian, "On Reed-Solomon and Algebraic Geometry Codes," *Ph.D. Thesis*, Department of Electrical and Computer Engineering, University of Waterloo, Ontario, 1993.

[38] M. Morii and M. Kasahara, "Generalized Key-Equation of Remainder Decoding Algorithm for Reed-Solomon Codes," *IEEE Trans. Inform. Theory,* vol.IT-38, no.6, pp. 1801-1807, Nov. 1992.

[39] R.E. Blahut, "Transform Decoding without Transforms," *Presented at the Tenth IEEE Communication Theory Workshop*, Cypress Gardens, FL, 1980.

[40] R.E. Blahut, "A Universal Reed-Solomon Decoder," *IBM J. Res. Develop.*, vol.28, no.2, pp. 150-158, Mar. 1984.

[41] Y.R. Shayan, T. Le-Ngoc, and V.K. Bhargava, "A Binary-Decision Approach to Fast Chien Search for Software Decoding of BCH Codes," *IEE Proc.*, vol.134, pt. F, pp. 629-632, Oct. 1987.

[42] H. Okano and H. Imai, "A Construction Method for 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.

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

[44] T.K. Truong, L.J. Deutsch, I.S. Reed, J.S. Hsu, K. Wang, and C.S. Yeh, "The VLSI design of a Reed-Solomon Encoder Using Berlekamp's Bit Serial Algorithm," *IEEE Trans. Comput.*, vol. C-33, pp. 906-911, Oct. 1984.

[45] G. Seroussi, "Hypersystolic Reed-Solomon Encoder," U.S. Patent No. 4,958,348, issued May 30, 1989.

[46] G. Seroussi, "A Systolic Reed-Solomon Encoder," *IEEE Trans. Inform. Theory*, vol.37, no.4, pp. 1217-1220, Jul. 1991.

[47] E.R. Berlekamp, "Hypersystolic Computers," JASON Workshop on Advanced Computer Architectures, La Jolla, Calif., 1986.

[48] H.T. Kung, "Why Systolic Architectures?" *IEEE Computer Magzine*, vol.15, pp. 972-980, 1992.

[49] E.R. Berlekamp, G. Seroussi, and P. Tong, "Hypersystolic Reed-Solomon Decoder," U.S. Patent No. 4,958,348, issued Sept.18, 1990.

[50] H.M. Shao, T.K. Truong, L.J. Deutsch, J.H. Yuen, and I.S. Reed, "A VLSI Design of a Pipeline Reed-Solomon Decoder," *IEEE Trans. on Comput.*, vol.C-34, no.5, May 1985.

[51] H.M. Shao, I.S. Reed, "On the VLSI Design of a Pipeline Reed-Solomon Decoder Using Systolic Arrays," *IEEE Trans. Comput.*, vol.37, pp.1273-1280, Oct. 1988.

[52] C.S. Yeh, I.S. Reed, and T.K. Truong, "Systolic Multipliers for Finite Fields $GF(2^m)$," *IEEE Trans. Comput.*, vol.C-33, pp.357-360, Apr. 1984.

[53] C.C. Wang, T.K. Truong, H.M. Shao, L.J. Deutsch, J.K. Omura, and I.S. Reed, "VLSI Architectures for Computing Multiplications and Inverses in $GF(2^m)$," *IEEE Trans. Comput.*, vol.C-34, pp.709-717, Aug. 1985.

[54] P.A. Scott, S.E. Tavares, and L.E. Peppard, "A Fast Multiplier for $GF(2^m)$," *IEEE J. Select. Areas Commun.*, vol.SAC-4, Jan. 1986.

[55] I.S. Hsu, T.K. Truong, L.J. Deutsch, and I.S. Reed, "A Comparison of VLSI Architecture of Finite Field Multipliers Using Dual, Normal, or Standard Bases", *IEEE Trans. Comput.*, vol.37, no.6, Jun. 1988.

[56] M.A.Hasan and V.K.Bhargava, "Bit-Serial systolic division and multiplier for finite fields $GF(2^m)$," *IEEE Tran. Comput.*, vol.41, no.8, Aug.1992.

[57] Y.R. Shayan, "Versatile Reed-Solomon Decoders," *Ph.D. Thesis*, Concordia University, Montreal, Canada, 1990.

[58] S. Le-Ngoc, T. Banerjee, and Y. Ye, "A PC-Based General Purpose Reed-Solomon Codec Simulator", Proceeding of Canadian Conference on Electrical and Computer Engineering, Halifax, Canada. pp. 751-754, Sept. 1994.

[59] Intel Corpotation, *8086/8088 User's Manual - Programmer's and Hardware Reference*,, 1989.

[60] C.L. Chen, "High-Speeding Decoding of BCH Codes," *IEEE Tran. Inf. Theor.* IT-27, pp. 254-256, 1981.

[61] S. Le-Ngoc,"Information Theory and Coding", Lecture notes, Faculty of Engineering, MUN, Canada, 1994.