# SIMULATION AND IMPLEMENTATION OF A ROBUST DIRECT SEQUENCE SPREAD SPECTRUM COMMUNICATION SYSTEM

By

Sajid Saleem

Submitted to the Department of Electrical Engineering

in Partial Fulfillment of the requirements for the Degree of

Master of Science

in

Electrical Engineering

**Thesis Advisor**

Dr. Mohammad Bilal Malik

**College of Electrical and Mechanical Engineering**

**National University of Sciences and Technology, Pakistan**

**2008**

بسم الله الرحمن الرحيم

In the name of Allah, the most Merciful and the most Beneficent

# ABSTRACT

Reed Solomon codes form an important class of linear cyclic block codes with numerous applications in communications and data storage. This thesis involves investigation and Hardware Description Language (HDL) implementation of Reed Solomon decoding algorithms and code acquisition for Direct Sequence spread spectrum (DSSS) systems. Conventional decoding algorithms which can correct errors up to half the minimum distance include Berlekamp-Massey (BM) and extended Euclidean (eE) algorithms. These algorithms are compared with respect to their hardware complexity, architecture regularity and decoding delay. A series of algorithmic transformations result in a fully systolic architecture for BM algorithm. This reformulated BM algorithm requires fewer hardware resources and reduced critical path delay when compared with architectures for eE algorithms. A parameterized Verilog code generator for Reed Solomon encoder and Berlekamp Massey architecture has been written in Matlab. Alternate RS decoding procedures based upon polynomial interpolation such as Guruswami-Sudan (GS) algorithm and Berlekamp-Welch (BW) algorithm are implemented using Matlab. GS algorithm is a list decoding algorithm which can provide error correction capabilities beyond half the minimum distance.

Second part of the thesis deals with synchronization issues in a DSSS with emphasis on Code acquisition. A baseband DSSS transmitter using a PN spreading sequence equipped with read only memory (ROM) based raised cosine filter is implemented. Correct de-spreading and decoding of data is possible only if the receiver reference sequence and received sequence are properly synchronized. Receiver coarse synchronization is done by parallel search over the code offset space. Cross correlation of these sequences is performed in the frequency domain by exploiting computational efficiency of the Fast Fourier Transform algorithm.

*To my Parents and Teachers*

# ACKNOWLEDGEMENTS

All praise to Allah Almighty the most merciful, beneficent and the source of all knowledge; Who granted the courage and knowledge to complete this research successfully.

I would like to express my sincere gratitude to my supervisor, Dr. Mohammad Bilal Malik, for being the guiding force behind the research necessary for the completion of this thesis. Without his guidance, encouragement, support, ideas, and scientific enlightenment this work would not have been possible.

I would like to thank Dr. Khalid Munawar, Dr. Shahzad Amin Sheikh and Mr. Asim Ajaz for serving as members on the committee and for their suggestions regarding this work. I would also like to thank Mr. Muhammad Salman and Mr. Abdul Wahid for providing useful suggestions and continuous support.

And finally I am thankful to my family and friends for their support, trust and patience during this research work.

# TABLE OF CONTENTS

## Table of Contents

# LIST OF FIGURES

# LIST OF TABLES

**CHAPTER 1**

# INTRODUCTION

## 1.1 Background

Modern communication systems are required to operate at high data rates with constrained power and bandwidth. These conflicting requirements lead to complex modulation and pulse shaping along with inevitable use of efficient error control coding and an increased level of signal processing at the receiver. Synchronization requirements also become more stringent at high data rates and, as a result, receivers become more complex.

This thesis investigates a special class of non-binary cyclic block codes recognized for their superior multiple error correction capability called Reed-Solomon codes. Moreover, the synchronization problem for Direct Sequence spread spectrum (DSSS) system is also considered and a parallel search scheme for DSSS acquisition is developed and implemented using Verilog Hardware description language (HDL).

Error control coding also called channel coding in the context of digital communication has a history dating back to the middle of the twentieth century [1,10,16]. In recent years, the field has been revolutionized by codes which are capable of approaching the theoretical limits of performance, the channel capacity.

Error control can be classified into Error correction and Error detection [16]. Error correction coding is the means whereby errors introduced into digital data as a result of transmission through a communication channel can be corrected based upon received data. Error detection coding is the means whereby errors can be detected based upon received information.

Error control coding can provide the difference between an operating communication system and a dysfunctional system. It has been a significant enabler in the telecommunication revolution, the internet, digital recording, and space exploration. Error control coding is ubiquitous in modern, information-based society. Every compact disc, CD-ROM, or DVD employs codes to protect the data embedded in the plastic disk. Every hard disk drive employs correction coding. Every phone call made over a digital cellular phone employs it. Every packet transmitted over the internet has a protective coding "wrapper" used to determine if the packet has been received correctly. Even everyday commerce takes advantage of error detection coding. Every consumer good and every text employs ISBN (International Standard Book Number) and UPC (Universal Product Code) respectively to uniquely identify and to ensure reliability in scanning [1].

The principle of channel codes is to represent the information being transmitted as a sequence of symbols and then add redundant symbols (parity check) in a structured manner. This encoded information is transmitted over the channel and a noisy version is received. The structural arrangement of the redundant received information is used by the channel decoder to detect and possibly correct the errors induced during transmission [15].

Reed Solomon (RS) codes are among the most extensively used error-control codes, with applications ranging from magnetic recording, through satellite and mobile communications to deep space exploration [7].

## 1.2 Basic Communication System:

A digital communication system has functionality to perform physical actions on information. A basic frame-work for a single communication link is shown in the Figure 1-1. This communication link transforms the information from the source into a form suitable for transmission over the designated channel. At the other end, reverse transformations are done to recover the data and sent to a sink. The performance of all these blocks is governed by the theorems from information theory.

There are various codes employed in a communication system. Let us take a brief overview of every block and understand the context of each type of code especially error-correction codes which is the focus of this thesis.

**Source:** Source represents data to be communicated which may represent any kind of information. They can be viewed as streams of random numbers governed by some probability distribution.

**Source Encoder:** Source encoder performs data compression by removing redundancy. Source

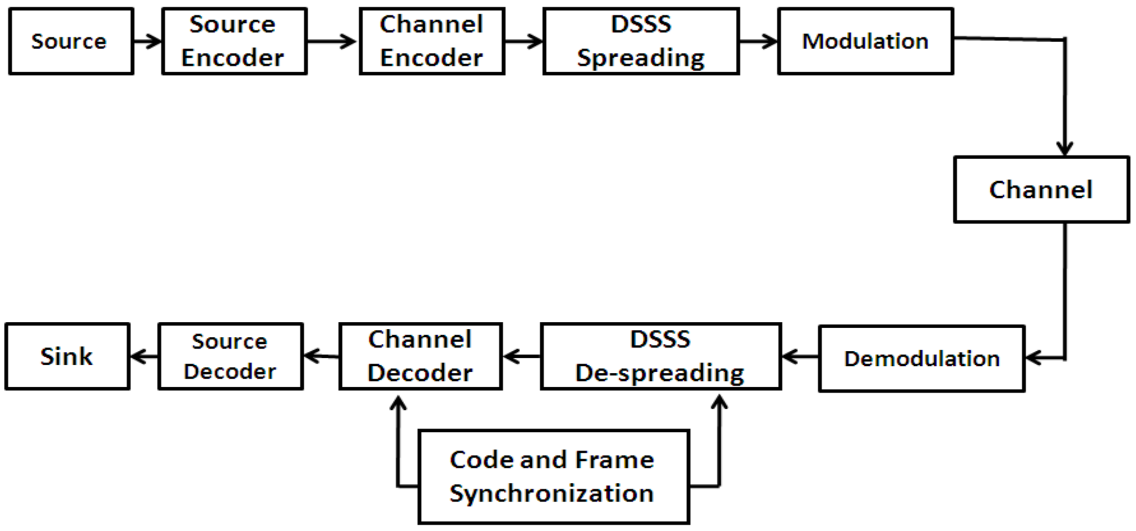


**Figure 1-1: A general frame work for a DSSS digital communications**

coding theorem puts entropy of the source as the theoretical minimum bound on the compression capabilities of source encoder.

**Channel Coder:** Channel coder adds redundant information in a structured way to the stream of input symbols that allows errors which are introduced by the channel to be corrected.

The redundancy in the source cannot be used as an alternative to channel coding because source redundancy is unstructured and thus wasteful of power and bandwidth to transmit.

Because of the redundancy introduced by the channel coder, there must be more symbols at the output of the coder than at the input. The rate R of a channel coder can be defined as

$$R = \frac{k}{n}$$

Where n is the number of output symbols produced for every k message symbols at its input.

**The Modulator:** Converts the symbol sequences from the channel encoders into signals suitable for transmission over the channel.

**Channel:** Channel is the medium over which information is conveyed. Examples of channels include telephone lines, fiber optic cable, internet cables, microwave radio channels, high frequency channels, cell phone channels, etc. These are the channels in which information is conveyed between two distinct places. Information may also be conveyed between two separate times, for example, by writing information onto a computer disk and then retrieving it at a later time. Hard disks, diskettes, CD-ROMS, DVDs, and solid state memory are other examples of channels.

**Channel Impairments:** As signals travel through a channel they may be corrupted. For example, a signal may have noise added to it; it may experience time delay or timing jitter, or suffer from attenuation due to propagation distance and/or carrier offset; it may be multiply reflected by objects in its path, resulting in constructive and/or destructive interference patterns; it may experience inadvertent interference from other channels, or be deliberately jammed. It may be filtered by channel response, resulting in interference among symbols. These sources of corruption in many cases all occur simultaneously.

For purpose of analysis, channels are frequently characterized by mathematical models, which (it is hoped) are sufficiently accurate to be representative of the attributes of the actual channel, yet are also sufficiently abstracted to yield tractable mathematics.

Channels can have different information carrying capabilities. For example, a dedicated fiber-optic cable is capable of carrying more information than a plain old telephone service (POTS) pair of copper wires. Associated with each channel is a quantity known as the capacity C, which indicates how much information it can carry reliably.

The reliable information a channel can carry is intimately related to the use of error correction coding. The governing theorem from information theory is Shannon's Channel Coding Theorem [10], which states essentially, "Provided that the rate R of the transmission is less than the capacity C, there exists a code such that the probability of error can be made arbitrarily small."

Channel encoding and modulation may be combined into Coded Modulation [1].

**The Demodulator/Equalizer:** Receives the signal from the channel and converts it into a sequence of symbols. This typically involves many functions, such as filtering, demodulation, carrier synchronization, symbol timing estimation, frame synchronization, and matched filtering, followed by a detection step in which decisions about the transmitted symbols are made.

**The Channel Decoder:** Exploits the redundancy introduced by the channel encoder to correct any errors that may have been introduced. As suggested by the figure, demodulation, equalization and decoding may be combined e.g. in a turbo equalizer.

**Source Decoder:** Provides uncompressed received data.

**The sink:** Ultimate destination of the data.

**Code and Frame Synchronization:** The synchronization block influences almost every block. The coherent demodulation of a digitally modulated signal requires that the receiver be synchronous to the transmitter. Two sequences of events are said to be synchronous relative to each other when the events in one sequence and the corresponding events in the other occur simultaneously. The process of making a situation synchronous is called Synchronization. At the receiver, the process of synchronizing the frequency and phase of the carrier is called carrier recovery and synchronizing symbol boundaries is called symbol Alignment, symbol recovery or symbol timing recovery. A coherent demodulator requires knowledge of carrier phase, carrier frequency and symbol timing for successful operation. Similarly a channel decoder block must know the boundaries of the block or frame to be decoded. This is called frame synchronization [26].

## 1.3 Motivation

Reed Solomon (RS) codes and their decoding is a very rich and growing research area even after 48 years of their introduction by Irving S. Reed and Gustave Solomon. Their outstanding error performance and diversity of application areas make them most attractive when compared with other block codes. New methods and architectures are being sought which reduce the decoding complexity, improve the error correction performance without reducing the code rate. Concatenation of RS codes with convolutional codes has made it possible to reach within half dB of the Shannon’s theoretical bound of channel capacity. Various reformulations of the decoding algorithms which reduce the architecture complexity and provide more regular systolic architectures have been derived. Recently introduced concept of soft-decision decoding for Reed Solomon codes has also met with great success. These architectures originally thought of as unpractical because of very high complexity have been made implemented by such architectural innovations.

## 1.4 Objectives of the Thesis

- This thesis involves a detailed investigation of the Reed Solomon codes, their encoding, various decoding approaches, error performance capability for various decoding procedures and their algebraic formulation.
- High speed architectures for Berlekamp Massey algorithms and their efficient reformulations have been investigated.
- Major RS decoding algorithms such as Berlekamp-Massey (BM), Extended Euclidean eE) algorithm and Berlekamp-Welch (BW) algorithm have been implemented and tested in Matlab.
- Verilog Hardware description Language (HDL) code generator for Matlab has been coded which generates all the Verilog files with more than twenty different modules, ready to be synthesized and simulated.
- Theoretical understanding of Guruswami-Sudan (GS) decoding and implementation of all the modules has been carried out in Matlab.

## 1.5 Overview of the Thesis

Thesis contents are organized into seven chapters. Chapter 2 provides the theoretical construction of Reed Solomon codes, systematic encoding, syndrome evaluation and theorems related to Berlekamp-Massey algorithm. Chapter 3 deals with derivation of second Key-equation and associated decoding technique of Berlekamp Welch. Chapter 4 introduces list decoding and interpolation based decoding algorithm introduced by Guruswami and Sudan. Koetter's interpolation and Roth-Ruckenstein's factorization algorithm are important components of the GS decoder and have been discussed thoroughly. Chapter 5 gives an overview of Spread Spectrum communications, PN sequences, acquisition for Direct sequence spread spectrum systems. Chapter 6 deals with HDL implementation details of Reed Solomon decoder and Chapter 7 concludes the thesis and provides future work recommendations. Selected references are provided at the end.

**CHAPTER 2**

# REED SOLOMON CODES AND BERLEKAMP-MASSEY DECODING

The most commonly used error correcting codes are the BCH and Reed Solomon Codes. The BCH code is named for Bose, Ray-Chaudhari, and Hocquenghem, who published work in 1959 and 1960 which revealed a means of designing codes over $GF(2)$ with a specified design distance. Decoding algorithms were then developed by Peterson and others.[1,10]

The Reed-Solomon codes are named for their inventors, who published in 1960. It was later realized that Reed Solomon (RS) codes and BCH codes are related and that their decoding algorithms are quite similar. Decoding of these codes is an extremely rich area.

## 2.1 BCH Codes

BCH codes are cyclic codes and hence may be specified by a generator polynomial. A BCH code over $GF(q)$ of length n capable of correcting at least $t$ errors is specified as follows:

1. Determine the smallest $m$ such that $GF(q^m)$ has a primitive $n$-th root of unity $\beta$.
2. Select a non-negative integer b. Frequently, $b = 1$.
3. Write down a list of $2t$ consecutive powers of $\beta$

   $$\beta^b, \beta^{b+1}, \dots, \beta^{b+2t-1}$$

   and determine the minimal polynomial with respect to $GF(q)$ of each of these powers of $\beta$.
4. The generator polynomial $g(x)$ is the least common multiple (LCM) of these minimal polynomials.
5. The code is a $(n, n - deg(g(x))$ cyclic code.

Because the code is constructed using minimal polynomials with respect to $GF(q)$, the generator $g(x)$ has coefficients in $GF(q)$ , and the code is over $GF(q)$.

**Definition 2-1**

If $\boldsymbol{b} = \mathbf{1}$ in the construction procedure, the BCH code is said to be $\boldsymbol{narrow\ sense}$. If $\boldsymbol{n} = \boldsymbol{q^m} - \mathbf{1}$, then the BCH code is said to be $\boldsymbol{primitive}$ [1].

Two fields are involved in the construction of BCH codes. The "small field" $GF(q)$ is where the generator polynomial has its coefficients and is the field where the elements of the code words are. The "big field" $GF(q^m)$ is the field where the generator polynomial has its roots. For encoding purposes, it is sufficient to work only with the small field. However, decoding requires operations in the extension field.

## 2.2 The BCH Bound

The BCH bound is the proof that the constructive procedure described above produces codes with at least the specified minimum distance.

**Theorem 2-1**

Let $C$ be a $q$-ary $(n, k)$ cyclic code with generator polynomial $g(x)$. Let $GF\ (q^{m)})$ be the smallest extension field of $GF(q)$ that contains a primitive $nth$ root of unity and let $\beta$ be a primitive $nth$ root of unity in that field. Let $g(x)$ be the minimal-degree polynomial in $GF\ (q)[x]$ having $2t$ consecutive roots of the form

$$g(\beta^b) = g(\ \beta^{b+1}) =, \dots, = \ g(\beta^{b+2t-1}) = 0 \qquad \text{Eq. ( 2-1)}$$

then the minimum distance of the code satisfies $d_{min} \geq \ \delta = 2t + 1$ ; that is, the code is capable of correcting at least $t$ errors.[1]

## 2.3 Reed Solomon Codes

There are actually two distinct constructions for Reed-Solomon codes. While these initially appear to describe different codes, it can be shown using Galois Field Fourier transform techniques that two are in fact equivalent. Most of the decoding operations are concerned with the second construction.

### 2.3.1 Reed Solomon Construction 1

**Definition 2-2**

Let $\propto$ be a primitive element in $GF(q^m)$ and let $n = q^m - 1$. Let $\boldsymbol{m} = (m_0, m_1, \dots, m_{k-1}) \in GF(q^m)^k$ be a message vector and let $m(x) = m_0 + m_1 x + \cdots + m_{k-1}x^{k-1} \in GF(q^m)[x]$ be its associated polynomial. Then the encoding is defined by the mapping $\rho: m(x) \mapsto \mathbf{c}$ by

$$(c_0, c_1, \dots, c_{n-1}) \triangleq \rho\big(m(x)\big) = (m(1), m(\propto), m(\propto^2), \dots, m(\propto^{n-1}))$$

That is $\rho\big(m(x)\big)$ evaluates $m(x)$ at all the non-zero elements of $GF(q^m)$. The Reed Solomon code of length $n = q^m - 1$ and dimension $k$ over $GF(q^m)$ is the image under $\rho$ of all polynomials in $GF(q^m)[x]$ of degree less than or equal to $k-1$.

More generally, a Reed Solomon code can be defined by taking $n \le q$, choosing $n$ distinct elements out of $(q^m)$ , $\propto_1, \propto_2, \dots, \propto_n$ known as the **Support set** , and defining the encoding operation as

$$\rho\big(m(x)\big) = (m(\propto_1), m(\propto_2), m(\propto_3), \dots, m(\propto_n))$$

The code is the image of the support set under $\rho$ of all polynomials in $GF(q^m)[x]$ of degree less than k.[1]

Following properties can be shown to be true for RS Codes.

1. The Reed Solomon code is a linear code
2. The minimum distance of an (n,k) Reed Solomon code is $d_{min} = \mathrm{n} - \mathrm{k} + 1$.
3. Reed Solomon codes achieve the singleton bound and are thus maximum distance separable codes.

This construction of RS Codes came first historically and Guruswami Sudan list decoding algorithm is based on it [8].

### 2.3.2 Reed Solomon Construction 2

In constructing BCH codes, generator polynomials over $GF(q)$ (base field) are dealt with by finding least common multiple of minimal polynomials which have all the conjugates of $\beta$ as roots. The degree of resulting generator polynomial usually exceeds the number $2t$ of roots specified. However, in Reed Solomon codes, we can operate in the extension field [1].

A Reed-Solomon code is a $q^m$ -ary BCH code of length $q^m - 1$. In $GF(q^m)$, the minimal polynomial for any element $\beta$ is $(x - \beta)$. The generator polynomial for an RS-Code is therefore

$$g(x) = (x - \alpha^b)(x - \alpha^{b+1}) \dots (x - \alpha^{b+2t-1}), \qquad \text{Eq. ( 2-2)}$$

where $\alpha$ is a primitive element. There are no extra roots of $g(x)$ included due to conjugacy in the minimal polynomials, so the degree of $g$ is exactly equal to $2t$. Thus, $n - k = 2t$ , for an RS code. The design distance is $\delta = n - k + 1$.

## 2.4 Systematic Encoding of Reed Solomon Codes

Reed Solomon codes may be encoded just as any other cyclic code (provided that the arithmetic is done in the right field). Given a message vector $\boldsymbol{m} = (m_0, m_1, \dots, m_{k-1})$, where each $m_i \in \mathrm{GF(q)}$, and its corresponding message polynomial, $m(x) = m_0 + m_1 x + \cdots + m_{k-1}x^{k-1} \in GF(q^m)[x]$ , the systematic encoding process is

$$c(x) = m(x)x^{n-k} - R_{g(x)}[m(x)x^{n-k}] \qquad \text{Eq. ( 2-3)}$$

where $R_{g(x)}[.]$ denotes the operation of taking the remainder after division by$g(x)$.

Typically, the code is over $GF(2^m)$, for some $m$. The message symbol $m_i$ can then be formed by taking $m$ bits of data, then interpreting these as the vector representation of the $GF(2^m)$ elements [16].

## 2.5 Decoding BCH and RS Code General Outline

There are many algorithms which have been developed for decoding BCH or RS codes. The algebraic decoding of BCH or RS codes has the following steps:

1. Compute the syndromes.
2. Determination of an error locator polynomial, whose roots provide an indication of where the errors are. There are several different ways of finding the locator polynomial. These methods include Peterson's algorithm for BCH codes, the Berlekamp-Massey algorithm for BCH codes; the Peterson-Gorenstein-Zierler algorithm for RS codes, the Berlekamp-Massey algorithm for RS codes, and the Euclidean Algorithm. In addition there are techniques based upon Galois-filed Fourier transforms.

3. Finding the roots of the error locator polynomial. This is usually done using the Chien search, which is an exhaustive search over all the elements in the field.
4. For RS codes or non-binary BCH codes, the error values must also be determined. This is typically accomplished using Forney's algorithm.

### 2.5.1 Computation of the Syndrome

Since $2t$ consecutive powers of $\alpha$ are roots of the generator polynomial,

$$g(\alpha^{b}) = g(\alpha^{b+1}) = \cdots = g(\alpha^{b+2t-1}) = 0$$

It follows that a codeword $c = (c_0, c_1, \ldots, c_{n-1})$ with polynomial

$c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1}$ has

$$c(\alpha^{b}) = c(\alpha^{b+1}) = \cdots = c(\alpha^{b+2t-1}) = 0$$

For a received polynomial

$$r(x) = c(x) + e(x)$$

We have $\quad S_j = r(\alpha^{j+b}) = c(\alpha^{j+b}) + e(\alpha^{j+b}) = e(\alpha^{j+b}), \quad j = 0,\ldots, 2t-1$

The values $S_0, S_1, \ldots, S_{2t-1}$, are called the Syndromes of the received data.Suppose that $r$ has $v$ errors in it which are at locations $i_1, i_2, \ldots, i_v$, with corresponding values in these locations $e_{i_j} \neq 0$. Then

$$S_j = \sum_{l=1}^{v} e_{i_l} (\propto^{j})^{i_l} = \sum_{l=1}^{v} e_{i_l} (\propto^{i_l})^{j}$$

Let $\quad X_l = \propto^{i_l}$

Then we can write $\quad S_j = \sum_{l=1}^{v} e_{i_l} (X_l)^{j}, \quad j = 0,1,\ldots,2t-1$

For binary codes, we have $e_{i_j} = 1$ ( i.e. if there is a non-zero error , it must be 1). For a moment, we restrict our attention to binary (BCH) codes. Then we have

$$S_j = \sum_{l=1}^{v} {X_l}^{j} \qquad \text{Eq. ( 2-4)}$$

If we know $X_l$ , then we know the location of the error. For example, suppose we know that $X_1 = \propto^4$. This means, by definition of $X_l$ that $i_1 = 4$; that is, the error is in the received digit $r_4$. We thus call the $X_l$ the error locators. The next stage in the decoding problem is to determine the error locators $X_l$ , given the syndrome $S_j$ [1].

### 2.5.2 The Error Locator Polynomial

From Eq (2-4), we obtain the following equations:

$$S_0 = X_1 + X_2 + \cdots + X_v \qquad \text{Eq. ( 2-5)}$$

$$S_1 = {X_1}^2 + {X_2}^2 + \cdots + {X_v}^2$$

$$\vdots$$

$$S_{2t-1} = {X_1}^{2t} + {X_2}^{2t} + \cdots + {X_v}^{2t}$$

The equations are said to be *power-sum symmetric functions*. This gives us $2t$ equations in the $v$ unknown error locators. In principle, this set of nonlinear equations could be solved by an exhaustive search, but this would be computationally unattractive [10].

Rather than attempting to solve these non-linear equations directly, a new polynomial is introduced, the error-locator polynomial, which casts the problem in a different, and more tractable, setting. The error locator polynomial is defined as

$$\Lambda(x) = \prod_{l=1}^{v}(1 - X_l x) = \Lambda_v x^v + \Lambda_{v-1}x^{v-1} + \ldots + \Lambda_1 x + \Lambda_0 \qquad \text{Eq. ( 2-6)}$$

where $\Lambda_0 = 1$. By this definition, if $x = {X_l}^{-1}$, then $\Lambda(x) = 0$; that is, the roots of the error locator polynomial are at the reciprocals (in the Galois field arithmetic) of the error locators [1].

### 2.5.3 Chein Search

If we have the error-locator polynomial, the next step is to find the roots of the error locator polynomial. The field of interest is $GF(q^m)$. Being a finite field, we can examine every element of the field to determine if it is a root [13].Suppose for example, that v = 3 and the error locator polynomial is

$$\Lambda(x) = \Lambda_0 + \Lambda_1 x + \Lambda_2 x^2 + \Lambda_3 x^3 = 1 + \Lambda_1 x + \Lambda_2 x^2 + \Lambda_3 x^3$$

We evaluate $\Lambda(x)$ at each non-zero element in the field in succession: $x = 1, x = \propto, x = \propto^2, \ldots, x = \propto^{q^m-2}$ . This gives us the following

$\Lambda(1) = 1 + \Lambda_1(1) + \Lambda_2(1)^2 + \Lambda_3(1)^3$

$\Lambda(\propto) = 1 + \Lambda_1(\propto) + \Lambda_2(\propto)^2 + \Lambda_3(\propto)^3$

$\Lambda(\propto^2) = 1 + \Lambda_1(\propto^2) + \Lambda_2(\propto^2)^2 + \Lambda_3(\propto^2)^3$

$\vdots$

$\Lambda(\propto^{q^m-2}) = 1 + \Lambda_1(\propto^{q^m-2}) + \Lambda_2(\propto^{q^m-2})^2 + \Lambda_3(\propto^{q^m-2})^3$

A set of $v$ registers are loaded initially with the coefficients of the error locator polynomial, $\Lambda_1, \dots, \Lambda_v$. The initial output is the term

$$A = \sum_{j=1}^{v} \Lambda_j = \Lambda(x) - 1|_{x=1}$$

If A = 1, then an error has been located (since then $\Lambda(x) = 0$). At the next stage, each register is multiplied by $\propto^j$ ,j=1,2,..,v, so the register contents are $\Lambda_1 \propto, \Lambda_2 \propto^2, \Lambda_3 \propto^3, \dots, \Lambda_v \propto^v$. The output is the sum

$$A = \sum_{j=1}^{v} \Lambda_j \propto^j = \Lambda(x) - 1|_{x=\propto}$$

The registers are multiplied again by successive powers of $\propto$ , resulting in evaluation at $\propto^2$. This procedure continues until $\Lambda(x)$ has been evaluated at all non-zero elements of the field.

If the roots are distinct and all lie in the appropriate field, then we use these to determine the error locations. If they are not distinct or lie in the wrong field, then the received word is not within distance $t$ of any codeword. (This condition can be observed if the error locator polynomial of degree $v$ does not have $v$ roots in the field that the operations take in; the remaining roots are either repeated or exist in an extension of the field). The corresponding error pattern is said to be an uncorrectable error pattern. An uncorrectable error pattern results in a **Decoder Failure** [11].

## 2.6 Finding the Error Locator Polynomial

Let us return to the question of finding the error locator polynomial using the syndromes. Let us examine the structure of the error locator polynomial by expanding it for the case $v = 3$.

$$\Lambda(x) = 1 - x(X_1 + X_2 + X_3) + x^2(X_1X_2 + X_1X_3 + X_2X_3) - x^3X_1X_2X_3$$

$$= \Lambda_0 + \mathrm{x}\Lambda_1 + \mathrm{x}^2\Lambda_2 + \mathrm{x}^3\Lambda_3$$

So that

$\Lambda_0 = 1$

$\Lambda_1 = -(X_1 + X_2 + X_3)$

$\Lambda_2 = X_1X_2 + X_1X_3 + X_2X_3$

$\Lambda_3 = -(X_1X_2X_3)$

In general, for an error locator of degree $v$ we find that

$\Lambda_0 = 1$ Eq. ( 2-7)

$$-\Lambda_1 = \sum_{\mathrm{i}=1}^{\mathrm{v}} X_i = X_1 + X_2 + X_3 + \ldots + X_v$$

$$\Lambda_2 = \sum_{i<j} X_iX_j = X_1X_2 + X_1X_3 + \ldots X_1X_v + \ldots + X_{v-1}X_v$$

$$-\Lambda_3 = \sum_{i<j<k} X_iX_j X_k = X_1X_2X_3 + X_1X_2X_4 + \ldots + X_1X_2X_v + \ldots + X_{v-2}X_{v-1}X_v$$

$$(-1)^{\mathrm{v}}\Lambda_{\mathrm{v}} = X_1X_2X_3 \ldots X_v$$

That is, the coefficient of the error locator polynomial $\Lambda_{\mathrm{i}}$ is the sum of the product of all combinations of the error locator taken $i$ at a time. Equations of the form (above) are referred to as the ***elementary symmetric functions*** of the error locators (so called because if the error locators $\{X_i\}$ are permuted, the same values are computed [1,10].

The power sum symmetric functions of the Eq(2-7) provides a non-linear relationship between the syndromes and the error locators. The elementary symmetric functions provide a non-linear relationship between the coefficients of the error locator polynomial and the error locators. The Key observation is that there is a linear relationship between the syndromes and the coefficients of the error locator polynomial. This relationship is provided by the Newton Identities, which apply over all fields.

**Theorem 2-2**

The syndromes and the coefficients of the error locator polynomial are related by

$$S_k + \Lambda_1 S_{k-1} + \ldots + \Lambda_{k-1} S_1 + k\Lambda_k = 0 \quad 1 \le k \le v$$

$$S_k + \Lambda_1 S_{k-1} + \ldots + \Lambda_{v-1} S_{k-v+1} + \Lambda_v S_{k-v} = 0 \qquad k > v$$

Eq. ( 2-8)

That is,

$k = 1$: $S_1 + \Lambda_1 = 0$

$k = 2$: $S_2 + \Lambda_1 S_1 + 2\Lambda_2 = 0$

$\vdots$

$k = v$: $S_v + \Lambda_1 S_{v-1} + \Lambda_2 S_{v-2} + \cdots + \Lambda_{v-1} S_1 + v\Lambda_v = 0$

$k = v + 1$ : $S_{v+1} + \Lambda_1 S_v + \Lambda_2 S_{v-1} + \cdots + \Lambda_v S_1 = 0$

$k = v + 2$ : $S_{v+1} + \Lambda_1 S_{v+1} + \Lambda_2 S_v + \cdots + \Lambda_v S_2 = 0$

$\vdots$

$k = 2t$: $S_{2t} + \Lambda_1 S_{2t-1} + \Lambda_2 S_{2t-2} + \cdots + \Lambda_v S_{2t-v} = 0$ Eq. ( 2-9)

For $k > v$ , there is linear feedback shift register relationship between the syndromes and the coefficients of the error locator polynomial.

$$S_j = -\sum_{i=1}^{v} \Lambda_i S_{j-i}$$ Eq. ( 2-10)

This equation can be expressed in a matrix form

$$\begin{bmatrix} S_1 & S_2 & & S_v \\ S_2 & S_3 & \cdots & S_{v+1} \\ S_3 & S_4 & & S_{v+2} \\ \vdots & & \ddots & \vdots \\ S_v & S_{v+1} & \cdots & S_{2v-1} \end{bmatrix} \begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \Lambda_{v-2} \\ \vdots \\ \Lambda_1 \end{bmatrix} = -\begin{bmatrix} S_{v+1} \\ S_{v+2} \\ S_{v+3} \\ \vdots \\ S_{2v} \end{bmatrix}$$

The $v \times v$ matrix, which we denote $M_v$, is a Toeplitz matrix, constant on the diagonals [1,10,6]. The number of errors $v$ is not known in advance, so it must be determined. The Peterson-Gorenstein-Zierler decoder [1] operates as follows.

1. Set $v = t$.
2. Form $M_v$, and compute the determinant $\det(M_v)$ to determine if $M_v$ invertible. If it is not invertible, set $v = v - 1$ and repeat this step.
3. If $M_v$ is invertible, solve the coefficients $\Lambda_1, \Lambda_2, \dots, \Lambda_v$.

### 2.6.1 Simplifications for Binary Codes and Peterson's Algorithm

For binary codes, Newton's identities are subject to further simplifications $nS_j = 0$ if $n$ is even and $nS_j = S_j$ if $n$ is odd. Furthermore, we have $S_{2j} = S_j{}^2$. We can thus write Newton's Identities as,

$$S_1 + \Lambda_1 = 0$$

$$S_3 + \Lambda_1 S_2 + \Lambda_2 S_1 + \Lambda_3 = 0$$

$$\vdots$$

$$S_{2t-1} + \Lambda_1 S_{2t-2} + \dots + \Lambda_t S_{t-1} = 0,$$

which can be expressed in the matrix form as

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ S_2 & S_2 & 1 & 0 & \dots 0 & 0 \\ S_4 & S_3 & S_2 & S_1 & 0 & 0 \\ & & \vdots & & & \\ & & & & & \end{bmatrix} \begin{bmatrix} \Lambda_1 \\ \Lambda_2 \\ \Lambda_3 \\ \vdots \\ \Lambda_t \end{bmatrix} = - \begin{bmatrix} S_1 \\ S_2 \\ S_3 \\ \vdots \\ S_{2t-1} \end{bmatrix} \qquad \text{Eq. ( 2-11)}$$

or $A\Lambda = -S$. If there is in fact t errors, the matrix is invertible , as we can determine by computing the determinant of the matrix. If it is not invertible, remove two rows and columns and then try again. Once $A$ is found, we find its roots. This matrix based approach for solving for the error-locator polynomial is called Peterson's algorithm for decoding binary BCH codes [1].

For large number of errors, Peterson's algorithm is quite complex. Computing the sequence of determinants to find the number of errors is costly. So is solving the system of equations, once the number of errors is determined. We therefore look for more efficient techniques.

## 2.7 Berlekamp Massey Algorithm

While Peterson's method involves straightforward linear algebra, it is computationally complex in general. Starting with matrix A in Eq (2-11) it is examined to see if it is singular.

This involves either attempting to solve the equations (e.g., by Gaussian Elimination or equivalent), or computing the determinant to see if the solution can be found. If A is singular, then the last two rows and columns are dropped to form a new A matrix. Then the attempted solution must be re-computed starting over with the new A matrix.

The Berlekamp-Massey algorithm takes a different approach. Starting with a small problem, it works up to increasingly longer problems until it obtains an overall solution. However, at each stage, it is able to reuse information it has already learnt. Whereas, as the computational complexity of the Peterson method is $O(v^3)$, the computational complexity of the Berlekamp-Massey algorithm is $O(v^2)$ [12].

We have observed from the Newton's Identity, Eq. ( 2-10), that ,

$$S_j = -\sum_{i=1}^{v} \Lambda_i S_{j-i} \quad j = v+1, v+2, \dots, 2t \qquad \text{Eq. ( 2-12)}$$

This formula describes the output of a linear feedback shift register (LFSR), with coefficients $\Lambda_1, \Lambda_2, \dots, \Lambda_v$. In order for this formula to work, we must find the $\Lambda_j$ coefficients in such a way that the LFSR generates the known sequence of Syndromes $S_1, S_2, .., S_{2t}$. Furthermore, by the Maximum-likelihood principle; the number of errors $v$ determined must be the smallest that is consistent with the observed syndromes. We therefore want to determine the shortest such LFSR.

In the Berlekamp-Massey algorithm, we build the LFSR that produces the entire sequence $\{S_1, S_2, .., S_{2t}\}$ by successively modifying an existing LFSR, if necessary, to produce increasingly longer sequences. We start with an LFSR that could produce $S_1$. We determine if that LFSR could also produce the sequence $\{S_1, S_2\}$; if it can, then no modifications are necessary. If the sequence cannot be produced using the current LFSR configuration, we determine a new LFSR that can produce the longer sequence.

Proceeding inductively in this way, we start from an LFSR capable of producing a sequence $\{S_1, S_2, .., S_{k-1}\}$ and modify it if necessary, so that it can also produce the sequence$\{S_1, S_2, .., S_k\}$. At each stage, the modifications to the LFSR are accomplished so that the LFSR is the shortest possible. By this means after completion of the algorithm, an LFSR has been found that is able to produce $\{S_1, S_2, .., S_{2t}\}$ and its coefficients correspond to the error locator polynomial $\Lambda(x)$ of smallest degree [1].

Since we build up the LFSR using information from prior computations, we need a notation to represent the Λ(x) used at different stages of the algorithm. Let $L_k$ denote the length of the LFSR produced at stage k of the algorithm. Let

$$\Lambda^{[k]}(x) = 1 + \Lambda_1{}^{[k]} x + \Lambda_2{}^{[k]} x^2 + \cdots + \Lambda_{L_k}{}^{[k]} x^{L_k}$$

be the ***connection polynomial*** at stage k, indicating the connections for the LFSR capable of producing the output sequence$\{S_1, S_2, .., S_k\}$. That is,

$$S_j = -\sum_{i=1}^{L_k} \Lambda_i{}^{[k]} S_{j-i} \quad j = L_k + 1, .., k. \qquad \text{Eq. ( 2-13)}$$

It is important to realize that some of the coefficients in $\Lambda^{[k]}(x)$ may be zero, so that $L_k$ may be different from the degree of $\Lambda^{[k]}(x)$. In realizations which use polynomial arithmetic, it is important to keep in mind what the length is as well as the degree.

At some intermediate step, suppose we have a connection polynomial $\Lambda^{[k-1]}(x)$, of length $L_{k-1}$ that produces $\{S_1, S_2, .., S_{k-1}\}$ for some $k - 1 < 2t$. We check if this connection polynomial also produces $S_k$. By computing the output,

$$\hat{S}_k = -\sum_{i=1}^{L_{k-1}} \Lambda_i{}^{[k-1]} S_{k-i}$$

If $\hat{S}_k$ is equal to $S_k$, then there is no need to update the LFSR, so $\Lambda^{[k]}(x) = \Lambda^{[k-1]}(x)$, and $L_k = L_{k-1}$. Otherwise, there is some non-zero discrepancy associated with $\Lambda^{[k-1]}(x)$,

$$d_k = S_k - \hat{S}_k = S_k + \sum_{i=1}^{L_{k-1}} \Lambda_i{}^{[k-1]} S_{k-i} = \sum_{i=0}^{L_{k-1}} \Lambda_i{}^{[k-1]} S_{k-i} \qquad \text{Eq. ( 2-14)}$$

In this case, we update the connection polynomial using the formula,

$$\Lambda^{[k]}(x) = \Lambda^{[k-1]}(x) + Ax^l \Lambda^{[m-1]}(x), \qquad \text{Eq. ( 2-15)}$$

where A is some element in the field, $l$ is an integer, and $\Lambda^{[m-1]}(x)$, is one of the prior connection polynomials produced by our process associated with non-zero discrepancy$d_m$. Using this new connection polynomial, we compute the new discrepancy denoted by $d_{\acute{k}}$ , as

$$d_{\acute{k}} = \sum_{i=0}^{L_k} \Lambda_i{}^{[k]} S_{k-i} = \sum_{i=0}^{L_{k-1}} \Lambda_i{}^{[k-1]} S_{k-i} + A \sum_{i=0}^{L_m - 1} \Lambda_i{}^{[m-1]} S_{k-i-l} \qquad \text{Eq. ( 2-16)}$$

Now, let $l = k - m$ . Then, by comparison with the definition of the discrepancy in Eq. ( 2-14) the second summation gives

$A\sum_{i=0}^{L_m-1} \Lambda_i^{[m-1]} S_{m-i} = A d_m.$

Thus, if we choose $A = -d_m{}^{-1} d_k$, then the summation in Eq. (2-16), gives

$$d_{\acute{k}} = d_k - d_m{}^{-1} d_k \, d_m = 0$$

So the new connection polynomial produces the sequence $\{S_1, S_2, .., S_k\}$ with no discrepancy.

### 2.7.1 Characterization of LFSR Length in Massey's Algorithm

The update in Eq. (2.15) is, in fact, the heart of Massey's algorithm. If all we need is an algorithm to find a connection polynomial, no further analysis is necessary. However, the problem was to find the shortest LFSR, but have no indication yet that it is the shortest. Following two theorems provide results about it [1].

**Theorem 2-3**

Suppose that an LFSR with connection polynomial $\Lambda^{[k-1]}(x)$ of length $L_{k-1}$ produces the sequence $\{S_1, S_2, .., S_{k-1}\}$, but not the sequence $\{S_1, S_2, .., S_k\}$ then any connection polynomial that produces the latter sequence must have a length $L_k$ satisfying

$$L_k \geq k - L_{k-1}$$

Since the shortest LFSR that produces the sequence $\{S_1, S_2, .., S_k\}$ must also produce the first part of that sequence, we must have $L_k > L_{k-1}$. Combining this with the result of the theorem, we obtain,

$$L_k \geq \max(L_{k-1}, k - L_{k-1}) \qquad \text{Eq. ( 2-17)}$$

We observe that the shift register cannot become shorter as more outputs are produced.

We have seen how to update the LFSR to produce a longer sequence using Eq. ( 2-15) and have also seen that there is a lower bound on the length of the LFSR. We now show that this lower bound can be achieved with equality, thus providing the shortest LFSR which produces the desired sequence.

**Theorem 2-4**

In the update procedure, if $\Lambda^{[k]}(x) \neq \Lambda^{[k-1]}(x)$, then a new LFSR can be found whose length satisfies

$$L_k = \max(L_{k-1}, k - L_{k-1})$$

In the update step, we observe that the new length is the same as the old length if $L_{k-1} \geq k - L_{k-1}$, that is, if

$$2L_{k-1} \geq k$$

In this case, the connection polynomial is updated but there is no change in length. The shift register synthesis algorithm, known as Massey's algorithm, is presented first in pseudo code as Algorithm 2-1 where we use the notations.

$$c(x) = \Lambda^{[k]}(x)$$

to indicate the "Current" connection polynomial and

$$p(x) = \Lambda^{[m-1]}(x)$$

to indicate the "previous" connection polynomial. Also, N is the number of input symbols $N = 2t$ for many decoding problems.

**Algorithm 2-1**

Berlekamp-Massey Algorithm[1]

```
$\boldsymbol{Input\ S_1, S_2, \dots, S_N}$
$Initialize$:
$L = 0$ $(the\ current\ length\ of\ the\ LFSR)$
$c(x) = 1$ $(the\ current\ connectoin\ polynomial)$
$p(x) = 1$ $(the\ previous\ connection\ polynomial)$
$l = 1$          $(l\ is\ k - m, the\ amount\ of\ shift\ in\ update)$
$d_m = 1$        (Previous discrepancy)
$for\ k = 1\ to\ n$
    $d = S_k + \sum_{i=1}^{L} c_i S_{k-i}$          $(Compute\ the\ discrepancy)$
    $if\ (d == 0)$          $(no\ change\ in\ update)$
      $l = l + 1$
    $else$
        $if(\ 2L\ \geq k)\ \ then\ (no\ length\ change\ in\ the\ update)$
            $c(x) = c(x) - dd_m^{-1}x^l p(x)$
            $l = l + 1$
        $else$        $(update\ c\ with\ length\ change)$
            $t(x) = c(x)$        $(temporary\ storage)$
            $c(x) = c(x) - dd_m^{-1}x^l p(x)$
            $L = k - L$
            $p(x) = t(x)$
            $d_m = d$
            $l = 1$
        $end$
    $end$
$end$
```

## 2.8 Non-Binary BCH and RS Decoding

For non-binary BCH or RS decoding, some additional work is necessary. Some extra care is needed to find the error locators, and then the error values must be determined. As the syndromes are related to the error-values as:

$$S_0 = e_{i_1}X_1 + e_{i_2}X_2 + \cdots + e_{i_v}X_v$$

$$S_1 = e_{i_1}{X_1}^2 + e_{i_2}{X_2}^2 + \cdots + e_{i_v}{X_v}^2$$

$$\vdots$$

$$S_{2t-1} = e_{i_1}{X_1}^{2t} + e_{i_2}{X_2}^{2t} + \cdots + e_{i_v}{X_v}^{2t}$$

Because of the $e_{i_l}$ coefficients these are not power-sum symmetric functions as was the case for the binary codes. Nevertheless, in a similar manner, it is possible to make use of an error locator polynomial [4,6].

**Lemma 2-1**

The syndromes and the coefficients of the error locator polynomial $\Lambda(x) = \Lambda_v x^v + \Lambda_{v-1}x^{v-1} + \ldots + \Lambda_1 x + \Lambda_0$ are related by

$$\Lambda_v S_{j-v} + \Lambda_{v-1} S_{j-v+1} + \ldots + \Lambda_1 S_{j-1} + S_j = 0 \qquad \text{Eq. ( 2-18)}$$

Because Eq (2-18) holds, the Berlekamp-Massey algorithm (in its non-binary formulation), can be used to find the coefficients of the error locator polynomial, just as for binary codes.

## 2.9 Forney's Algorithm

Having formed the error locator polynomial and its roots, there is still one more step for the non-binary BCH or RS codes: we have to find the error values. Let us return to the syndrome,

$$S_j = \sum_{l=1}^{v} e_{i_l} X_l^j, \quad j = 0,1,..,2t-1$$

Knowing the error-locators (obtained from the roots of the error locator polynomial) it is straightforward to setup and solve a set of linear equations:

$$\begin{bmatrix} X_1 & X_2 & & X_v \\ {X_1}^2 & {X_2}^2 & \cdots & {X_v}^2 \\ {X_1}^3 & {X_2}^3 & & {X_v}^3 \\ \vdots & & \ddots & \vdots \\ {X_1}^{2t} & {X_2}^{2t} & \cdots & {X_v}^{2t} \end{bmatrix} \begin{bmatrix} e_{i_1} \\ e_{i_2} \\ e_{i_3} \\ \vdots \\ e_{i_v} \end{bmatrix} = \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ \vdots \\ S_{2t-1} \end{bmatrix} \qquad \text{Eq. ( 2-19)}$$

However, there is a method which is computationally easier and in addition provides us a key insight for another way of doing the decoding. It may be observed that the matrix in Eq. ( 2-19) is essentially a Vandermonde matrix. There exist fast algorithms for solving Vandermonde systems. One of these which apply specifically to this problem is known as Forney's Algorithm.

Let us define the syndrome polynomial as

$$S(x) = S_0 + S_1\, x + \cdots + S_{2t-1} x^{2t} = \sum_{j=0}^{2t-1} S_j\, x^j \qquad \text{Eq. ( 2-20)}$$

and the error-evaluator polynomial as

$$\Omega(x) = S(x)\Lambda(x)\ (mod\ x^{2t}\ ) \qquad \text{Eq. ( 2-21)}$$

This equation is called the Key Equation. [1,6,10]

**Theorem 2-5**

(Forney's Algorithm) The error values for a narrow-sense Reed-Solomon code are computed by

$$e_{i_k} = -\frac{\Omega(X_k^{-1})}{\acute{\Lambda}(X_k^{-1})}$$

Where $\acute{\Lambda}(x)$ is the formal derivative of $\Lambda(x)$ [1].

## 2.10 Euclidean Algorithm for the Error Locator Polynomial

We have seen that the Berlekamp-Massey algorithm can be used to construct the error locator polynomial. An alternative algorithm called extended Euclidean algorithm can also be used for the same purpose. This approach to decoding is often called the Sugiyama algorithm [1,4]. We return to the key equation:

$$\Omega(x) = S(x)\Lambda(x)\ (mod\ x^{2t}\ )$$

Given only $S(x)$ and $t$, we desire to determine the error locator polynomial $\Lambda(x)$ and the error evaluator polynomial $\Omega(x)$. From the statement of the problem it looks hopefully unconstrained. However, we can re-write the Key equation above as

$$\Theta(x)(x^{2t}) + \Lambda(x)S(x) = \Omega(x)$$

for some polynomial $\Theta(x)$. Also, the extended Euclidean algorithm returns, for a pair of elements $(a, b)$ from a Euclidean domain, a pair of elements $(s, t)$ such that

$$as + bt = c$$

where c is the GCD of $a$ and $b$. In our case, we run the extended Euclidean algorithm to obtain a sequence of polynomials $\Theta^{[k]}(x), \Lambda^{[k]}(x)$ and $\Omega^{[k]}(x)$ satisfying

$$\Theta^{[k]}(x)\ (x^{2t}) + \Lambda^{[k]}(x)S(x) = \Omega^{[k]}(x)$$

And the stopping criterion is when the polynomial $\Omega^{[k]}(x)$ has a degree less than $t$.

The steps to decode using the Euclidean algorithm are summarized as follows:

1. Compute the syndromes and the syndrome polynomial

$$S(x) = S_0 + S_1 x + \cdots + S_{2t-1} x^{2t} = \sum_{j=0}^{2t-1} S_j x^j$$

2. Run the Euclidean algorithm with $a(x) = x^{2t}$ and $b(x) = S(x)$, until $\deg\big(r_i(x)\big) < t$. Then $\Omega(x) = r_i(x)$ and $\Lambda(x) = t_i(x)$.
3. Find the roots of $\Lambda(x)$ and the error locator $X_i$.
4. Solve for the error values using Forney's formula.

In terms of computational efficiency, it appears that the Berlekamp-Massey algorithm procedure may be slightly better than the Euclidean algorithm for binary codes, since the Berlekamp-Massey deals with polynomials no longer than the error locator polynomial, while the Euclidean algorithm may have intermediate polynomials of higher degree [6]. The computational complexity for Euclidean algorithm is probably quiet smaller. Also, the error evaluator polynomial $\Omega(x)$ is automatically obtained as useful by product of the Euclidean algorithm method. However, there are inversion-less reformulations of Berlekamp Massey algorithm which have considerably lower critical path delay and have comparable complexity [2,3].

CHAPTER 3

# BERLEKAMP-WELCH ALGORITHM

In this chapter, we discuss another decoding method for Reed Solomon codes. It is based upon a new Key equation and is called Remainder Decoding [1].

## 3.1 Workload for Reed-Solomon Decoding

A primary motivation between the remainder decoder is that its implementation may have lower decoder complexity. The decode complexity for a conventional decoding algorithm for an $(n, k)$ code having redundancy $\rho = n - k$ is summarized by the following steps:

1. Compute the syndromes. $\rho$ Syndromes must be computed, each with a computational cost of $O(n)$, for a total cost of $O(\rho n)$. Furthermore, all the syndromes must be computed, regardless of the number of errors.
2. Find the error locator polynomial and the error evaluator. This has a computation cost of $O(\rho^2)$, (depending on the approach).
3. Find the roots of the error locator polynomial. This has a computation cost of $O(\rho n)$ using the Chien Search.
4. Compute the error values, with a cost of $O(\rho^2)$.

Thus, if $\rho < n/2$, the most expensive steps are computing the syndrome and finding the roots. In remainder decoding, decoding takes place by computing remainders instead of syndromes; the remaining steps retain similar complexity. This results in potentially faster decoding. Furthermore, it is possible to find the error locator polynomial using a highly-parallelizable algorithm. The general outline for the new decoding algorithm is as follows [20]:

1. Compute the remainder polynomial $r(x) = R(x)\, mod\big(g(x)\big)$, with complexity $O(n)$ (using very simple hardware).
2. Compute an error-locator polynomial W(x) and an associated polynomial $N(x)$. The complexity is $O(\rho^2)$, Architectures exist for parallel processing.
3. Find the roots of the error locator polynomial, complexity $O(\rho n)$.

4. Compute the error values, complexity $O(n)$.

## 3.2 Derivations of the Welch-Berlekamp Key Equation

Welch-Berlekamp (WB) Key equation can be derived using two separate methods. The first derivation uses the definition of the remainder polynomial. The second definition shows that the WB Key equation can be obtained from Conventional Reed-Solomon Key equation [1].

### 3.2.1 The Welch-Berlekamp Derivation of the Key Equation

The generator polynomial for an $(n, k)$ RS code can be written as

$$g(x) = \prod_{i=b}^{b+d-2} (x - \alpha^i)$$

which is a polynomial of degree $d - 1$, where $d = d_{min} = 2t + 1 = n - k + 1$. We denote the received polynomial as $R(x) = c(x) + E(x)$. We designate the first $d - 1$ symbols of $R(x)$ as *check symbols*, and the remaining k symbols as the *Message symbols*. This designation applies naturally to systematic encoding of codewords, but we use it even in the case that non-systematic coding is employed. Let $L_c = \{0,1, \ldots, d-2\}$ be the index of set of the check locations with corresponding check locators $L_{\alpha^c} = \{\alpha^k, 0 \leq k \leq d-2\}$. Also $L_m = \{d-1, d, \ldots, n-1\}$ denote the index set of the message locations, with corresponding message locators $L_{\alpha^m} = \{\alpha^k, d-1 \leq k \leq n-1\}$

We define the remainder polynomial as

$$r(x) = R(x)\, mod(g(x))$$

and write $r(x)$ in terms of its coefficients as $r(x) = \sum_{i=0}^{d-2} r_i\, x^i$

The degree of $r(x)$ is $\leq d - 2$ . This remainder can be computed using conventional LFSR hardware that might be used for the encoding operation, with computational complexity $O(n)$.

**Lemma 3-1**

$$r(x) \equiv E(x)\, mod\, g(x)$$

And $r(\propto^k) = E(\propto^k)$ for $k \in \{b, b+1, \ldots, b+d-2\}$. [1,21]

#### 3.2.1.1 Single Error in a Message Location

To derive the WB key equation, we assume initially that a single error occurs. We need to make a distinction between whether the error location $e$ is a message location or a check location. Initially we assume that $e \in L_m$ with error value $Y$. We thus take $E(x) = Yx^e$, or the (error position, error location) = $(\propto^e, Y) = (X, Y)$. The notation $Y = Y[X]$ is also used to indicate the error value at the error locator $X$.

When $e \in L_m$ , then modulo operation $Yx^e \; mod \; g(x)$ "folds" the polynomial back into the lower order terms, as pictured in **Error! Reference source not found.**. Evaluating $r(x)$ at generator root locations we have by Lemma 3-1,

$$r(\propto^k) = E(\propto^k) = Y(\propto^k)^e = YX^k, \qquad k \in \{b, b+1, \dots, b+d-2\} \qquad \text{Eq. ( 3-1)}$$

where $X = \propto^e$ is the error locator. It follows that

$$r(\propto^k) - Xr(\propto^{k-1}) = YX^k - XYX^{k-1} = 0, \quad k \in \{b+1, b+2, \dots, b+d-2\}$$

Define the polynomial,

$$u(x) = r(x) - Xr(\propto^{-1}),$$

which has the degree less than $d-1$. Then $u(x)$ has roots at $\propto^{b+1}, \propto^{b+2}, \dots, \propto^{b+d-2}$, so that $u(x)$ is divisible by the polynomial

$$p(x) = \prod_{i=b+1}^{b+d-2} (x - \alpha^i) = \sum_{i=0}^{d-2} p_i x^i$$

which has degree $d-2$. Thus $u(x)$ must be a scalar multiple of $p(x)$,

$$u(x) = ap(x), \qquad \text{Eq. ( 3-2)}$$

For some $a \in GF(q)$. Equating coefficients between $u(x)$ and $p(x)$ we obtain,

$$r_i(1 - X \propto^{-i}) = ap_i. \qquad i = 0,1, \dots, d-2$$

That is,

$$r_i(\propto^i - X) = a \propto^i p_i. \qquad i = 0,1, \dots, d-2 \qquad \text{Eq. ( 3-3)}$$

We define the error locator polynomial as $W_m(x) = x - X = x - \propto^e$. (This definition is different from the error locator we defined for the conventional decoding algorithm, since the roots of $W_m(x)$ are the message locators, not the reciprocals of message locators.) Using $W_m(x)$, we see from Eq (3-3) that

$$r_i W_m(\propto^i) = a \propto^i p_i. \qquad i = 0,1,\dots,d-2 \qquad \text{Eq. ( 3-4)}$$

Since the error is in the message location, $e \in L_m, W_m(\propto^i)$ is not zero for $i = 0,1,\dots,d-2$. We can solve for $r_i$ as

$$r_i = {}^{a \propto^i p_i}\Big/_{W_m(\propto^i)} \qquad \text{Eq. ( 3-5)}$$

We can now eliminate the coefficient $a$ from Eq (3-5) The error value $Y$ can be computed using Eq. ( 3-1) choosing $k = b$:

$$Y = Y[X] = X^{-b} r(\propto^b) = X^{-b} \sum_{i=0}^{d-2} r_i \propto^{ib} = X^{-b} \sum_{i=0}^{d-2} \frac{a \propto^i p_i}{W_m(\propto^i)} \propto^{ib} = aX^{-b} \sum_{i=0}^{d-2} \frac{\propto^{i(b+1)} p_i}{(\propto^i - X)}$$

Define

$$f(x) = x^{-b} \sum_{i=0}^{d-2} \frac{\propto^{i(b+1)} p_i}{(\propto^i - x)}, \qquad x \in L_{\alpha^m}$$

which can be pre-computed for all the values of $x \in L_{\alpha^m}$. Then

$Y = af(X)$

or $a = Y/f(X)$. We thus write Eq (3-4) as

$$r_i = \frac{Y \propto^i p_i}{f(X) W_m(\propto^i)} \qquad \text{Eq. ( 3-6)}$$

### 3.2.1.2 Multiple errors in the Message Locations

Now assume that there are $v \geq 1$errors, with error locators, $X_i \in L_{\alpha^m}$ and corresponding error values $Y_i = Y[X_i]\ for\ i = 1,2,\dots,v$. Corresponding to each error there is a "mode" yielding a relationship $(\propto^{k}) = Y_i X_i^k$ , each of which has a solution of the form Eq. ( 3-6). Thus by linearity we can write

$$r_k = r[\propto^k] = p_k \propto^k \sum_{i=1}^{v} \frac{Y_i}{f(X_i)(\propto^k - X_i)} \quad ,k = 0,1,..,d-2 \qquad \text{Eq. ( 3-7)}$$

Now define the function,

$$F(x) = \sum_{i=1}^{v} \frac{Y_i}{f(X_i)(x - X_i)} \qquad \text{Eq. ( 3-8)}$$

having poles at the error locations. This function can be written as

$$F(x) = \sum_{i=1}^{v} \frac{Y_i}{f(X_i)(x - X_i)} = \frac{N_m(x)}{W_m(x)}$$

where $W_m(x) = \prod_{i=1}^{v}(x - X_i)$ is the error locator polynomial for the errors among the symbol locations and where $N_m(x)$ is the numerator obtained by adding together the terms in $F(x)$. It is clear that $\deg(N_m(x)) < \deg(W_m(x))$ . Note that the representation in Eq. ( 3-8) corresponds to a partial fraction expansion of $\frac{N_m(x)}{W_m(x)}$. Using this notation, Eq. ( 3-7) can be written as

$$r_k = p_k \propto^k F(\propto^k) = p_k \propto^k \frac{N_m(\propto^k)}{W_m(\propto^k)}$$

or
$$N_m(\propto^k) = \frac{r_k}{p_k \propto^k} W_m(\propto^k), k \in L_c = \{0,1,..,d-2\} \qquad \text{Eq. ( 3-9)}$$

$N_m(x)$ and $W_m(x)$ have the degree constraints $\deg(N_m(x)) < \deg(W_m(x))$ and $\deg(W_m(x)) \leq \lfloor (d-1)/2 \rfloor = t$, since no more than $t$ errors can be corrected, Eq (3-9) has the form of the Key equation we seek [1,21].

### 3.2.1.3 Errors in Check Locations

For a single error occurring in a check location $e \in L_c$, then $r(x) = E(x)$ since there is no "folding" by modulo operation [20]. Then $u(x) = r(x) - Xr(\propto^{-1} x)$ must be identically 0, so the coefficient a in Eq (3-2) is equal to zero. We can write

$$r_k = \begin{cases} Y & k = e \\ 0 & therwise \end{cases}$$

If there are errors in both check locations and message locations, let $E_m = \{i_1, i_2, \dots, i_{v_1}\} \subset L_m$ denote the error locations among the message locations and let $E_c = \{i_{v_1+1}, \dots, i_v\} \subset L_c$ denote the error locations among the check locations. Let $E_{\propto^m}=\{\propto^{i_1}, \propto^{i_2}, \dots, \propto^{i_{v_1}}\}$ and $E_{\propto^c}=\{\propto^{i_{v_1+1}}, \dots, \propto^{i_v}\}$ denote the corresponding error locators. The (error location, error value) pairs for the errors in message locations are $(X_i, Y_i), i = 1,2,\dots, v_1$. The pairs for errors in check locations are $(X_i, Y_i), i = v_1 + 1, \dots, v$. Then by linearity,

$$r_k = p_k \propto^k \sum_{i=1}^{v_1} \frac{Y_i}{f(X_i)(\propto^k - X_i)} + \begin{cases} Y_j & if\ error\ locator\ X_j = \propto^k \ is\ in\ a\ check\ locatoin \\ 0 & otherwise \end{cases}$$

Eq. ( 3-10)

Because of the extra terms added on in Eq. ( 3-10) , equation Eq (3-9) does not apply when $k \in E_c$, so we have

$$N_m(\propto^k) = \frac{r_k}{p_k \propto^k} W_m(\propto^k), k \in L_c \backslash E_c \qquad \text{Eq. ( 3-11)}$$

To account for errors among the check symbols, let $W_c(x) = \prod_{i \in E_c}(x - \propto^i)$ be the error locator polynomial for errors in check locations. Let

$N(x) = N_m(x)W_c(x)$ and $W(x) = W_m(x)W_c(x)$.

Since $N(\propto^k) = W(\propto^k) = 0$ for $k \in E_c$, we can write

$$N(\propto^k) = \frac{r_k}{p_k \propto^k} W(\propto^k), k \in L_c = \{0,1,.., d-2\} \qquad \text{Eq. ( 3-12)}$$

That is, the equation is now satisfied for all values of $k\ in\ L_c$. Eq (3-12) is the Welch-Berlekamp (WB) Key equation, to be solved subject to the conditions

$\deg(N(x)) < \deg(W(x))$ $\qquad$ $\deg(W(x)) < (\mathrm{d}-1)/2$

The polynomial $W(x)$ is the error locator polynomial, having roots at all the error locators. We write Eq (3-12) as

$$N(x_i) = W(x_i)\mathrm{y_i} \qquad \mathrm{i} = 1,2,\dots,\mathrm{m} = 2\mathrm{t} = \mathrm{d}-1 \qquad \text{Eq. ( 3-13)}$$

For points $(x_i, \mathrm{y_i}) = \left(\propto^{\mathrm{i}-1}, \mathrm{r_{i-1}}/\left(\mathrm{p_{i-1}} \propto^{\mathrm{i}-1}\right)\right), \quad \mathrm{i} = 1,2,\dots,\mathrm{m} = 2\mathrm{t}$

Hereafter we will refer to the $N(x)$ and $W(x)$ as $N_1(x)$ and $W_1(x)$, referring to the first (WB) derivation.

## 3.2.2 Derivation from the Conventional Key Equation

A WB-type key equation may also be obtained starting from the conventional key equation and syndromes [1]. Let us denote the syndromes as

$$S_i = \mathrm{R}\left(\propto^{\mathrm{b+i}}\right) = \mathrm{r}\left(\propto^{\mathrm{b+i}}\right) = \sum_{\mathrm{j}=0}^{\mathrm{d}-2} \mathrm{r_j}\left(\propto^{\mathrm{b+i}}\right)^{\mathrm{j}}, \mathrm{i} = 0,1,\dots,\mathrm{d}-2$$

The conventional error locator polynomial $\Lambda(x) = \prod_{i=1}^{v}(1 - X_i x) = \Lambda_0 + \Lambda_1 \mathrm{x} + \cdots + \Lambda_{\mathrm{v}} \mathrm{x}^{\mathrm{v}}$ where $\Lambda_0 = 1$; the Welch-Berlekamp error locator polynomial is $\mathrm{W}(x) = \prod_{i=1}^{v}(x - X_i) = \mathrm{W_0} + \mathrm{W_1 x} + \cdots + \mathrm{x^v}$. These are related by $\Lambda_{\mathrm{i}} = \mathrm{W_{v-i}}$. The conventional key equation can be written as

$$\sum_{i=0}^{v} \Lambda_{\mathrm{i}} \mathrm{S_{k-i}} = 0 \quad ; \qquad k = v, v+1, \dots, \quad d-2.$$

Writing this in terms of coefficients of W we have

$$\sum_{i=0}^{v} \mathrm{W_i}\, \mathrm{S_{k+i}} = 0 \quad ; \qquad k = 0,1,\dots, \quad d-2-v.$$

or $$\textstyle\sum_{i=0}^{v} \mathrm{W_i} \sum_{j=0}^{d-2} r_j\, \alpha^{j(b+k+i)} = 0$$

Rearranging,

$$\textstyle\sum_{j=0}^{d-2} r_j \left(\sum_{i=0}^{v} \mathrm{W_i} \alpha^{\mathrm{ji}}\right) \alpha^{j(k+b)} = 0 \quad , \qquad \mathrm{k} = 0,1,\dots,\mathrm{d}-2-\mathrm{v}. \qquad \text{Eq. ( 3-14)}$$

Letting

$$f_j = r_j W\left(\alpha^{\mathrm{j}}\right)\alpha^{jb}, \qquad \text{Eq. ( 3-15)}$$

Eq (3-14) can be written as

$$\sum_{j=0}^{d-2} f_j \alpha^{jk} = 0, \qquad k = 0,1,\dots,d-2-v.$$

which corresponds to the Vandermonde set of equations

$$\begin{bmatrix} 1 & 1 & & 1 & 1 \\ 1 & \alpha & \cdots & \alpha^{d-3} & \alpha^{d-2} \\ 1 & \alpha^2 & & \alpha^{2(d-3)} & \alpha^{2(d-2)} \\ \vdots & & \ddots & \vdots & \\ 1 & \alpha^{d-2-v} & \cdots & \alpha^{(d-2-v)(d-3)} & \alpha^{(d-2-v)(d-2)} \end{bmatrix}$$

with $(d-1-v) \times (d-1)$ matrix $V$. The bridge to the WB key equation is provided by the following lemma.

**Lemma 3-2[1]**

Let $V$ a $m \times r$ matrix $r > m$ having Vandermonde structure

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ u_1 & u_2 & & u_r \\ \vdots & & \ddots & \vdots \\ {u_1}^{m-1} & {u_2}^{m-1} & \cdots & {u_r}^{m-1} \end{bmatrix}$$

with the $\{u_i\}$ all distinct. For any vector **z** in the nullspace of V (satisfying $Vz = 0$), there exists a unique polynomial $N(x)$ of degree less than $r - m$ such that

$z_i = \frac{N(u_i)}{F'(u_i)}$, $i = 1,2,\dots,r$,

where $F(x) = \prod_{i=1}^{r}(x - u_i)$.

Hereafter we will refer $N(x)$ and $W(x)$ as $N_2(x)$ and $W_2(x)$, for the DB (Dabiri- Black) method derivation.

## 3.3 Finding the Error Values

We begin with the key equation in the WB-form Eq (3-12). Assuming that the error locator W(x) has been found it can be shown [1] that for WB Key equation error values $Y_i$ corresponding to an error locator $X_i$ can be computed as

$$Y[X_j] = r_k - p_k X_j \frac{N_1'(X_j)}{W_1'(X_j)}$$

For the DB form of WB equation, error values can be computed as follows

$$Y[X_k] = -\frac{N_2(\propto^k) \propto^{b(d-k-1)}}{W_2'(\propto^k) g(\propto^{b+k})} = -\frac{N_2(X_k){X_k}^{-b} \propto^{b(d-1)}}{W_2'(X_k) g(X_k \propto^b)} \quad (message\ locations)$$

$$Y_k[X_k] = r_k - \frac{N_2'(\propto^k) \propto^{b(d-k-2)}}{W_2'(\propto^k) g'(\propto^{b+k})} = r_k - \frac{N_2'(X_k){X_k}^{-b} \propto^{b(d-2)}}{W_2'(X_k) g'(X_k \propto^b)} \quad \text{(check locations)}$$

## 3.4 Rational Interpolation Problem

The key equation problem can be expressed as follows:

Given a set of points $(x_i, y_i), i = 1,2,..,m$ over some field $\mathcal{F}$, the problem of finding polynomial N(x) and W(x) with $deg(N(x)) < deg(W(x))$ satisfying

$$N(x_i) = W(x_i)y_i, \quad i = 1,2,\ldots,m. \qquad \text{Eq. ( 3-16)}$$

is called a rational interpolation problem[1,20]. Since in the case that $W(x_i) \neq 0$, we have

$$y_i = \frac{N(x_i)}{W(x_i)}$$

A solution to the rational interpolation problem provides a pair $[N(x), W(x)]$, satisfying Eq (3-16)

## 3.5 The Welch-Berlekamp Algorithm

Rational interpolation problem is structurally similar to the Berlekamp-Massey algorithm, in that it provides a sequence of solution pairs which are updated in the event that there is a discrepancy when a new point is considered. We are interested in a solution satisfying $deg(N(x)) < deg(W(x))$ and $deg(W(x)) \leq m/2$.

**Definition 3-1[1]**

The rank of a solution $[N(x), W(x)]$, is defined as

$rank[N(x), W(x)] = \max\{2\deg(\mathrm{W(x)}), 1 + 2\deg(\mathrm{N(x)})\}$

We construct a solution to the rational interpolation problem of rank $\leq$ m and show that it is unique. By the definition of the rank, the $deg(N(x)) < deg(W(x))$ .

The polynomial expression for the interpolation problem is useful . Let $P(x)$ be an interpolating polynomial such that $P(x_i) = y_i$ , i = 1,2, ..., m. For example, $P(x)$ could be the Lagrange interpolating polynomial ,

$$P(x) = \sum_{i=1}^{m} y_i \frac{\prod_{k=1,k\neq i}^{m}(x - x_k)}{\prod_{k=1,k\neq i}^{m}(x_i - x_k)}$$

By the evaluation Homomorphism the equation $N(x_i) = W(x_i)y_i$ is equivalent to

$$N(x) = W(x)P(x) \pmod{(x - x_i)}$$

Since it is true for each point $(x_i, y_i)$, and since the polynomials $(x - x_i)$, i = 1,2, ..., m are pairwise relatively prime, by the ring Isomorphism we can write using Chinese remainder theorem,

$$N(x) = W(x)P(x) \pmod{\prod(x)}, \qquad \text{Eq. ( 3-17)}$$

where $\prod(x) = \prod_{i=1}^{m}(x - x_i)$

**Definition 3-2**

Suppose $[N(x), W(x)]$, is a solution to the rational interpolation problem, and that $N(x)$ and $W(x)$ share a common factor $f(x)$ , such that $N(x) = n(x)f(x)$ *and* $W(x) = w(x)f(x)$. If $[n(x), w(x)]$, is also a solution to this problem, the solution $[N(x), W(x)]$, is said to be **Reducible.** A solution which has no common factors of $degree > 0$ which may be factored out leaving a solution is said to be **Irreducible** [1].

**Lemma 3-3**

There exists at least one irreducible solution to eq. Eq. ( 3-17) with rank $\leq$ m.

The Welch-Berlekamp algorithm finds a rational interpolation of minimal rank by building successive interpolants for increasingly larger set of points. First a minimal rank rational interpolant is found for the single point $(x_1, y_1)$. This is used to construct a minimal polynomial for the pair of points $\{(x_1, y_1), (x_2, y_2)\}$, and so on, until a minimal rank interpolant for the entire set of points $\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$ is found [1].

**Definition 3-3**

We say that $[N(x), W(x)]$ satisfy the interpolation problem if

$$N(x_i) = W(x_i)y_i \quad i = 1,2,\dots,k$$

The Welch Berlekamp finds a sequence of solutions $[N^{[k]}, W^{[k]}]$ of minimum rank satisfying the interpolation (k) problem, for $k = 1,2,\dots,m$. We can express the interpolation (k) problem as

$N(x) = W(x)P_k(x) \qquad (mod\, \Pi_{\mathrm{k}}(x))$

where $\Pi_{\mathrm{k}}(x) = \prod_{i=1}^{k}(x - x_i)$ and $P_k(x)$ is polynomial that interpolates (at least) the first $k$ points $P(x_i) = y_i,\ i = 1,2,\dots,k$.

As with Berlekamp-Massey algorithm, the Welch-Berlekamp algorithm propagates two solutions, using one of them in the update of the other. For the Welch-Berlekamp algorithm, the two sets of solution maintain the property that they are complements of each other.

**Definition 3-4**

Let $[N(x), W(x)]$ and $[M(x), V(x)]$ be two solutions of interpolation (k) such that

$$rank\, [N(x), W(x)] + rank[M(x), V(x)] = 2k + 1$$

And

$$N(x)V(x) - M(x)V(x) = f\Pi(x)$$

For some scalar f. Then $[N(x), W(x)]$ and $[M(x), V(x)]$ are **Complementary**. The key results to construct the algorithm are presented in following Lemmas [1].

**Lemma 3-4**

Let $[N(x), W(x)]$ be an irreducible solution to the interpolation (k) problem with $rank\ \leq k$. Then there exists at least one solution to the interpolation (k) problem which is a complement of $[M(x), V(x)]$ [1].

**Lemma 3-5**

If $[N(x), W(x)]$ is an irreducible solution to the interpolation (k) problem and $[M(x), V(x)]$ is another solution such that $rank[N(x), W(x)] + rank[M(x), V(x)] \leq 2k$, then $[M(x), \mathrm{V}(x)]$ can be reduced to $[N(x), W(x)]$.

This Lemma implies that there exists only one irreducible solution to the interpolation (k) problem with $rank \leq k$, and that this solution must have at least one complement [1,20].

**Lemma 3-6**

If $[\boldsymbol{N(x)}, \boldsymbol{W(x)}]$ is an irreducible solution to the interpolation (k) problem and $[\boldsymbol{M(x)}, \boldsymbol{V(x)}]$ is one of its complements, then for any $\boldsymbol{a}, \boldsymbol{b} \in \boldsymbol{\mathcal{F}}$, with $\boldsymbol{n} \neq \boldsymbol{0}$, $[\boldsymbol{bM(x)} - \boldsymbol{aN(x)}, \boldsymbol{bV(x)} - \boldsymbol{aW(x)}]$ is also one of its complements [1].

We are now ready to state and prove the theorem describing the Welch-Berlekamp algorithm.

**Theorem 3-1**

Suppose that $\left[N^{[k]}, W^{[k]}\right]$ and $\left[M^{[k]}, V^{[k]}\right]$ are two complementary solutions of the interpolation (k) problem. Suppose also that $\left[N^{[k]}, W^{[k]}\right]$ is the solution of lower rank. Let

$$b_k = N^{[k]}(x_{k+1}) - y_{k+1}W^{[k]}(x_{k+1})$$

$$a_k = M^{[k]}(x_{k+1}) - y_{k+1}V^{[k]}(x_{k+1})$$

(These are analogous to the discrepancies of the Berlekamp-Massey algorithm). If $b_k = 0$, (the discrepancy is zero, so no update is necessary) then

$\left[N^{[k]}, W^{[k]}\right]$ and $\left[(x - x_{k+1})M^{[k]}(x), (x - x_{k+1})V^{[k]}(x)\right]$

Are two complementary solutions of the interpolation (k+1) problem, and $\left[N^{[k]}, W^{[k]}\right]$ is the solution of the lower rank.

If $b_k \neq 0$, ( the discrepancy is not zero, so an update is required), then

$\left[(x - x_{k+1})N^{[k]}(x), (x - x_{k+1})W^{[k]}(x)\right]$ and $[b_kM^{[k]}(x) - a_kN^{[k]}(x), b_kN^{[k]}(x) - a_kW^{[k]}(x)]$

are two complementary solutions. The solution with lower rank is the solution to the interpolation (k+1) problem [1,20,21].

Based on this theorem, the Welch-Berlekamp algorithm is shown in the figure below.

**Algorithm 3-1[1,20]**

```
Welch Berlekamp Interpolation
$Input$: $(x_i, y_i)$,    $i = 1, \dots, m$
$Returns$: $[N^{[m]}(x), W^{[m]}(x)]$ $of\ minimal\ rank\ satisfying\ the\ interpolation\ problem$
$Initialize$: $N^{[0]}(x) = 0$; $V^{[0]}(x) = 0$; $W^{[0]}(x) = 1$; $M^{[0]}(x) = 1$;
$for\ i = 0\ to\ m-1$
    $b_i = N^{[i]}(x_{i+1}) - y_{i+1}W^{[i]}(x_{i+1})$            $(Compute\ discrepancy)$
    $if(b_i == 0)$                    $(then\ no\ change\ in\ the\ [N,W]\ solution)$
            $N^{[i+1]}(x) = N^{[i]}(x)$;     $W^{[i+1]}(x) = W^{[i]}(x)$;
            $M^{[i+1]}(x) = (x - x_{i+1})M^{[i]}(x)$ ;   $V^{[i+1]}(x) = (x - x_{i+1})V^{[i]}(x)$
    $else$            $(Update\ to\ account\ for\ discrepancy)$
            $a_i = M^{[i]}(x_{i+1}) - y_{i+1}V^{[i]}(x_{i+1})$ ;
            $M^{[i+1]}(x) = (x - x_{i+1})N^{[i]}(x)$ ; $V^{[i+1]}(x) = (x - x_{i+1})W^{[i]}(x)$ ;
            $N^{[i+1]}(x) = b_iM^{[i]}(x) - a_iN^{[i]}(x)$; $W^{[i+1]}(x) = b_iV^{[i]}(x) - a_iW^{[i]}(x)$ ;
            $If(\ rank\ [N^{[i+1]}(x),\ W^{[i+1]}(x)] >\ rank\ [M^{[i+1]}(x),\ V^{[i+1]}(x)])$
                            (swap for minimal rank)
                    $swap\ ([N^{[i+1]}(x),\ W^{[i+1]}(x)], [M^{[i+1]}(x),\ V^{[i+1]}(x)])$
            $end$ (if)
    $end$  (else)
$end$  $(for)$
```

# CHAPTER 4

# THE GURUSWAMI-SUDAN DECODING ALGORITHM

In 1997 Madhu Sudan [19], building on previous work of Welch-Berlekamp[20] and others, discovered a polynomial-time algorithm for decoding certain low-rate Reed-Solomon codes beyond the classical $d/2$ error-correcting bound. Two years later Guruswami and Sudan [22] published a significantly improved version of Sudan's algorithm, which was capable of decoding virtually every RS code at least somewhat, and often significantly, beyond the $d/2$ limit. The main focus of these seminal papers was to establish the existence of polynomial-time decoding algorithms, and not on devising practical implementations. However, several later authors, notably Koetter [23,24] and Roth-Ruckenstein[25] , were able to find low-complexity (no worse than $O(n^2)$ ) realizations for the key steps in the GS algorithm, thus making GS a genuinely practical engineering alternative in storage and transmission systems requiring RS codes [7,8].

An $(n, k)$ Reed-Solomon code over $F = GF(q)$, as given by Reed Solomon in their original paper is defined as follows. Let $(\alpha_1, \dots, \alpha_n)$ be a fixed list of $n$ distinct elements of $F$, called the ***support set*** of the code[1,10]. The encoding process is that of mapping a vector $(f_0, f_1, \dots, f_{k-1})$ of $k$ information symbols into an $n$-symbol codeword $(x_1, x_2, \dots, x_n)$ by *polynomial evaluation*, i.e.,

$$(x_1, x_2, \dots, x_n) = \quad (f(\alpha_1), \dots, f(\alpha_n)), \qquad \text{Eq. ( 4-1)}$$

where

$$f(x) = f0 + f1x + \cdot \cdot \cdot + f_{k-1}x^{k-1}. \qquad \text{Eq. ( 4-2)}$$

The corresponding *Reed-Solomon* code consists of all $n$-vectors of the form in Eq. ( 4-1) where $f(x)$ is a polynomial of degree $< k$. It is well-known that this code has minimum Hamming distance $d = n - k + 1$ and is therefore capable of correcting up to

$$t_0 = \left\lfloor \frac{n-k}{2} \right\rfloor \qquad \text{Eq. ( 4-3)}$$

errors. Conceptually, this may be accomplished as follows. The decoder searches the Hamming sphere of radius $t_0$ centered at the received word for codewords. If the sphere contains a unique codeword, that is the decoder's output. Otherwise, the decoder reports failure. (This strategy is called *bounded distance decoding*, (BDD) and dates back to Shannon's proof of the noisy-channel coding theorem. The conventional RS decoding algorithms, e.g., Berlekamp, Berlekamp-Massey, Continued Fractions, or Euclidean Algorithm-based are all BDD algorithms.) The decoding sphere cannot contain more than one codeword, since the minimum distance of the code is $> 2t_0$. If we attempt to correct more than $t_0$ errors by increasing the decoding radius, it is possible for the decoding sphere to contain more than one codeword, in which case the decoder will fail. For this reason, conventional wisdom asserts that the code is not capable of correcting more than $t_0$ errors. Nevertheless, if we examine the *probability* that the decoding sphere will contain multiple codewords, rather than the *possibility*, we may reach a different conclusion [7].

The Guruswami Sudan Decoder is capable of correcting more than $t_0$ errors[7,22]. It is a polynomial-time algorithm for correcting (in a certain sense) up to $t_{GS}$ errors, where $t_{GS}$ is the largest integer strictly less than $n - \sqrt{(k-1)n}$, i.e.,

$$t_{GS} = n - 1 - \lfloor (k-1)n \rfloor. \qquad \text{Eq. ( 4-4)}$$

It is easy to show that $t_{GS} \geq t_0$ , and often $t_{GS}$ is considerably greater than $t_0$. Asymptotically, for RS codes of rate $R$, the conventional decoding algorithms will correct a fraction $\tau_0 = (1-R)/2$ of errors, while the GS algorithm can correct up to $\tau_{GS} = 1 - \sqrt{R}$. The GS decoder has an adjustable integer parameter $m \geq 1$ called the *interpolation multiplicity*[1]. Associated with the interpolation multiplicity $m$ is positive integer $t = t_m$ called the *designed decoding radius*. Given a received word, the $GS(m)$ decoder returns a list which includes all codewords with Hamming distance $t_m$ or less from the received word, and perhaps a few others. The exact formula for $t_m$ is a bit complicated, however following relation holds

$$t_0 \leq t_1 \leq t_2 \leq \cdots ,$$

and there exists an integer $m_0$ such that

$$t_{m_0} = t_{m_0+1} = \cdot \ \cdot \ \cdot = t_{GS}.$$

## 4.1 An Overview of the GS(m) Algorithm

Suppose $C = (f(\alpha_1),\ldots,f(\alpha_n))$, is the transmitted codeword, where $f(x)$ is a polynomial of degree $< k$, and that $C$ is received as $R = (\beta_1,\ldots,\beta_n)$. Let $p(x)$ be any polynomial of degree $< k$ which maps to an RS codeword with Hamming distance $\leq t_m$ from $R$, i.e.,

$$|\{i : p(\alpha_i) \neq \beta_i\}| \leq t_m.$$

The $GS(m)$ decoder "finds" $p(x)$ as follows [8].

1. **The interpolation step**

   Given the received vector $R = (\beta_1,\ldots,\beta_n)$, the decoder constructs a two-variable polynomial $Q(x,y) = \sum_{i,j} a_{ij}x^i y^j$

   with the property that $\mathrm{Q(x,y)}$ has a zero of multiplicity m at each of the points $(\alpha_i, \beta_i)$, and for which the $(1, \mathrm{k}-1)$ weighted degree of $\mathrm{Q(x,y)}$ is as small as possible.

2. **The factorization step**

   The decoder then finds all factors of $Q(x,y)$ of the form $y - p(x)$, where $p(x)$ is a polynomial of degree $k-1$ or less. Let

   $$\mathcal{L} = \{p_1(x),\ldots,p_L(x)\}$$

   be the list of polynomials produced by this step. The polynomials (codewords) $\mathrm{p(x)} \in \mathrm{L}$ are of three possible types:

- Type 1. The transmitted, or *causal*, codeword.
- Type 2. Codewords with Hamming distance ≤ *tm* from *R*, which we call *plausible* codewords.
- Type 3. Codewords with distance $> t_m$ from $R$, which we call *implausible codewords*.

**Theorem 4-1**

*If the $GS(m)$ decoding algorithm is used, all plausible codewords will be in $\mathcal{L}$. In particular, the transmitted codeword will be in $\mathcal{L}$ if the number of channel errors is $\leq t_m$. The list may also contain implausible codewords, but the total number of codewords in the list, plausible and implausible, will satisfy $L \leq L_m$, where $L_m$ is conservatively estimated by*

$$L_m < (m + \frac{1}{2})\sqrt{\frac{n}{k-1}}$$

Let $\bar{L}(t)$ is the average number of codewords in a randomly chosen sphere of radius *t*, and which gives a heuristic upper bound on the probability that the decoding sphere will contain a non-causal codeword [1,8,22].

## 4.2 Monomial Orders and Generalized Degree

This section provides an introduction to the algebraic fundamentals of two-variable polynomials. These fundamentals include weighted monomial orderings, generalized degree functions, and certain related combinatorial results [1].

If $\mathcal{F}$ is a field, we denote by $F[x, y]$ the ring of polynomials in $x$ and $y$ with coefficients from $\mathcal{F}$. A polynomial $Q(x, y) \in F[x, y]$ is, by definition, a finite sum of monomials,

$$Q(x, y) = \sum_{i,j\geq 0} a_{ij}x^i y^j \qquad \text{Eq. ( 4-5)}$$

where only a finite number of the coefficients $a_{ij}$ are nonzero. The summation in Eq. ( 4-5) is two-dimensional, but often it is desirable to have a one-dimensional representation instead. To do this, we need to have a linear ordering of set of monomials $\mathcal{M}[x, y] = \{x^i y^j : i, j \geq 0\}$

It can be observed that the set $\mathcal{M}[x, y]$ is isomorphic to the set $\mathbb{N}^2$ of pairs of nonnegative integers under the bijection $x^i y^j \leftrightarrow (i, j)$. A *monomial ordering* is a relation " $<$ " on $\mathcal{M}[x, y]$ (equivalently, on $\mathbb{N}^2$) with the following three properties:

1. If $a1 \leq b1$ and $a2 \leq b2$, then $(a1, a2) \leq (b1, b2)$.
2. The relation " $<$ " is a total ordering, i.e., if $\boldsymbol{a}$ and $\boldsymbol{b}$ are distinct monomials, either $\boldsymbol{a} < \boldsymbol{b}$ or $\boldsymbol{b} < \boldsymbol{a}$.
3. If $\boldsymbol{a} \leq \boldsymbol{b}$ and $\boldsymbol{c} \in \mathbb{N}^2$, then $\boldsymbol{a} + \boldsymbol{c} \leq \boldsymbol{b} + \boldsymbol{c}$.

There are many possible monomial orderings, but the most important ones are the *weighted degree* monomial orders [1]. A WD monomial order is characterized by a pair $\boldsymbol{w} = (u, v)$ of nonnegative integers, not both zero. For a fixed $\boldsymbol{w}$, the $\boldsymbol{w}$-degree of the monomial $x^i y^j$ is defined as

$$deg_{\boldsymbol{w}} x^i y^j = ui + vj.$$

If we order $\mathcal{M}[x, y]$ by **w**-degree, i.e., declare that $\phi(x, y) < \phi'(x, y)$ if $deg_{\boldsymbol{w}}\, \phi(x, y) < deg_{\boldsymbol{w}}\, \phi'(x, y)$, we only get a partial order, since monomials with equal **w**-degree are incomparable. It turns out that there are just two ways to break such ties so that Property (3) is satisfied: ***w**-lexicographic (**w**-lex) order*, and ***w**-reverse lexicographic (**w**-revlex) order* [8].

**Definition.**

**w**-lex order is defined as follows:

$$x^{i_1}y^{j_1} < x^{i_2}y^{j_2}$$

if either $ui_1 + vj_1 < ui_2 + vj_2$ or $ui_1 + vj_1 = ui_2 + vj_2$ and $i_1 < i_2$.

**w**-revlex order is similar, except that the rule for breaking ties is $i_1 > i_2$.

(In the special case **w** = (1, 1), these orderings are called graded-lex, or grlex, and reverse graded-lex, or grevlex, respectively) [1].
Let "<" be a fixed monomial ordering:

$$1 = \phi_0(x,y) < \phi_1(x,y) < \phi_2(x,y) < \cdots$$

With respect to this ordering every nonzero polynomial in $F[x,y]$ can be expressed uniquely in the form

$$Q(x,y) = \sum_{j=0}^{J} a_j \phi_j\,(x,y)$$

for suitable coefficients $a_j \in \mathcal{F}$ with $a_j \neq 0$. The integer $J$ is called the $rank\ of\ Q(x,y)$, and the monomial $\phi_J$ is called the *leading monomial* of $Q(x,y)$. We indicate this notationally by writing $Rank(Q) = J$ and $LM(Q) = \phi_J\,(x,y)$ . The relation $LMP = LMQ$ is an equivalence relation, which we denote by $P \equiv Q$. We can extend the order " $<$ " to all of $F[x,y]$ by declaring $P < Q$ to mean LM$P$ < LM$Q$. In this way, "<," which is a total order on the set of monomials, becomes a **partial order** on $F[x,y]$ and a total order on the equivalence classes under LM [1,8].

In the case of a WD order, the weighted degree of the leading monomial $\phi_J$ is also called the weighted degree, or ***w****-degree*, of $Q(x, y)$, denoted $\deg_w Q$. Thus

$$deg_w\, Q(x,y) = max\,\{deg_w\, \phi(x,y) : a_j \neq 0\}$$

The ***w***-degree function has the following basic properties:

$$deg_w\, 0 = -\infty$$
$$deg_w(PQ) = deg_w\, P + deg_w\, Q$$
$$deg_w(P + Q) \leq max(deg_w\, P, deg_w\, Q)$$
$$deg_w(P + Q) = max(deg_w\, P, deg_w\, Q), if\ LMP \neq LMQ.$$

If $\phi_0$ $(x, y) < \phi_1(x, y) <$... is a fixed monomial ordering, and $\phi = x^i y^j$ is a particular monomial, the *index* of $\phi$, denoted **Ind($\phi$)**, is defined as the unique integer $K$ such that $\phi_K(x, y) = \phi$.

For (1, $v$) revlex order, the numbers $Ind(x^K)$ and $Ind(y^L)$are especially important, so we introduce a special notation for them:

$$A(K,v) \triangleq Ind(x^K)$$
$$B(L,v) \triangleq Ind(y^L)$$

it being understood that the underlying monomial order is (1, $v$)-revlex [8].

We note that $x^K$ is the first monomial of (1, $v$)-degree $K$, and $y^L$ is the last monomial of (1, $v$)-degree $vL$, so that

$$A(K,v) = |\{(i,j) : i + vj < K\}|$$
$$B(L,v) = |\{(i,j) : i + vj \le Lv\}| - 1.$$

## 4.3 Zeros and Multiple Zeros

In this section we consider bivariate polynomials, and focus on their notion of a zero, or a multiple zero.

If $Q(x,y) \in F[x,y]$, and $Q(\alpha,\beta) = 0$, we say that Q has a zero at $(\alpha,\beta)$.

**Definition 4-1**

We say that $Q(x,y) = \sum_{i,j} a_{ij}x^i y^j \in F[x,y]$ has a zero of multiplicity, or order m at (0, 0), and write

$$ord(Q : 0,0) = m,$$

If $Q(x,y)$ involves no term of total degree less than $m$, i.e., $a_{ij} = 0$ if $i + j < m$. Similarly, we say that $Q(x,y)$ has a zero of order $m$ at $(\alpha,\beta)$, and write

$$ord(Q : \alpha,\beta) = m,$$

if $Q(x + \alpha, y + \beta)$ has a zero of order m at (0, 0).

To calculate $ord(Q : \alpha,\beta)$, we need to be able to express $Q(x + \alpha, y + \beta)$ as a polynomial in *x* and *y*. The following theorems, due to H. Hasse tell us one way to do this. We begin with the one-variable version of Hasse's theorem [1].

**Theorem 4-2**

If $Q(x) = \sum_i a_i x^i \in F[x]$, then for any $\alpha \in F$, we have

$$Q(x + \alpha) = \sum_r Q_r(\propto)x^r,$$

where $Q_r(x) = \sum_i \binom{i}{r} a_i x^{i-r}$

which is called the $rth$ Hasse derivative of $Q(x)$. Also,

$$Q_r(\propto) = Coeff_{x^r}\, Q(x + \propto) = \sum_i \binom{i}{r} a_i \propto^{i-r}$$

and $$Q(x) = \sum_{r\ge 0} Q_r(\propto)\,(x - \propto)^r$$

**Theorem 4-3**

Let $Q(x,y) = \sum_{i,j} a_{i,j}x^i y^j \in F[x,y]$. For any $(\alpha,\beta) \in \mathcal{F}^2$, we have

$Q(x + \alpha, y + \beta) = \sum_{r,s} Q_{r,s}(\alpha,\beta) x^r y^s$

where

$$Q_{r,s}(x,y) = \sum_{i,j} \binom{i}{r}\binom{j}{s} a_{i,j} x^{i-r} y^{j-s}$$

which is called the $(r,s)th$ Hasse (mixed partial) derivative of $Q(x,y)$.

Also, an alternative equivalent formula is

$$Q_{r,s}(\alpha,\beta) = Coeff_{x^r y^s}\, Q(x+\propto, y+\beta)$$

and

$$Q(x,y) = \sum_{r,s} Q_{r,s}(\propto,\beta)\,(x-\propto)^r (y-\beta)^s$$

**Corollary:**

The polynomial $Q(x,y)$ has a zero of order $m$ at $(\alpha,\beta)$ if and only if $Q_{r,s}(\alpha,\beta) = 0$ for all $r$ and $s$ such that $0 \le r + s < m$ [8].

## 4.4 The Interpolation and Factorization Theorems

Two basics theorems of GS algorithm are stated as below.

### 4.4.1 The Interpolation Theorem

Suppose a nonnegative integer $m(\alpha)$ is assigned to each element $\alpha \in \mathcal{F}$, and we are asked to construct a polynomial $f(x)$ of least degree which has a zero of multiplicity $m(\alpha)$, at $x = \alpha$, for all $\alpha \in \mathcal{F}$. Clearly a minimum degree solution to this one-dimensional interpolation problem is

$$f(x) = \prod_{\alpha \in \mathcal{F}} (x - \alpha)^{m(\alpha)}$$

$$\deg(f(x)) = \sum_{\alpha \in \mathcal{F}} m(\alpha)$$

We are interested in the analogous two-dimensional interpolation problem: Given a required multiplicity $m(\alpha, \beta)$ for each $(\alpha, \beta) \in \mathcal{F}^2$, construct a low-degree polynomial $Q(x,y)$ which has zeros of the required multiplicity. This is a much harder problem, in general, but the following theorem gives a useful upper bound on the minimum required degree [7,8].

**Theorem 4-4**

Let $\{m(\alpha,\beta) : (\alpha,\beta) \in \mathcal{F}^2\}$ be a multiplicity function as above and let $\phi_0 < \phi_1 < \cdots$ be an arbitrary monomial order. Then there exists a non-zero polynomial $Q(x,y)$ of the form

$$Q(x,y) = \sum_{i=0}^{C} a_i \, \phi_i(x,y)$$

*where* $$C = \sum_{\alpha,\beta} \binom{m(\alpha,\beta)+1}{2}$$

which has a zero of multiplicity *m*(*α*, *β*), *at* (*x*, *y*) = (*α*, *β*), *for all* (*α*, *β*) $\in \mathcal{F}^2$.

For any (u, v), there is a nonzero polynomial $Q(x,y)$ with the required zero multiplicities whose (u, v)-degree is strictly less than $\sqrt{2uvC}$ [22].

### 4.4.2 The Factorization Theorem

If $Q(x,y) \in \mathcal{F}[x,y]$, and $f(x) \in \mathcal{F}[x]$, define the $Q$-score of $f$ as

$$S_Q(f) = \sum_{\alpha\in\mathcal{F}} ord(\,Q:\alpha, f(\alpha))$$

Suppose $f(x) \in \mathcal{F}_v[x], Q(x,y) \in \mathcal{F}[x,y]$, and
$S_Q(f) > deg_{1,v}Q$. Then $y - f(x)$ is a factor of $Q(x, y)$.

**Lemma 4-1**

*If* $f(x) \in \mathcal{F}_v[x]$, *then* $\deg\,(Q(x,f(x)) \le deg_{1,v}\,Q(x,y)$.

**Lemma 4-2**

$Q(x,f(x)) = 0$ *if and only if* $(y - f(x))|Q(x,y)$.

**Lemma 4-3**

*If* $ord(Q : \alpha,\beta) = K$, *and* $f(\alpha) = \beta$, *then* $(x - \alpha)^K \,|Q\big(x, f(x)\big)$ [8].

## 4.5 A Second Look at the Guruswami-Sudan Algorithm

Given a $(n,k)$ RS code over the finite field $\mathcal{F}$, with support set $(\alpha_1,\ldots,\alpha_n)$, and a positive integer $m$, the *GS*(*m*) decoder accepts a vector $\beta = (\beta_1,\ldots,\beta_n) \in \mathcal{F}^n$ as input, and produces a list of polynomials $\{f_1,\ldots,f_L\}$, as output. Here's how:

**The *GS*(*m*) Decoder.** *The GS*(*m*) *decoder constructs a nonzero two-variable polynomial of the form* $Q(x,y) = \sum_{j=0}^{C(n,m)} a_j \phi_j(x,y) = 0$ *where* $\phi_0 < \phi_1 <\ldots$ *is* (1, *v*)*-revlex monomial order, such that Q*(*x, y*) *has a zero of order m at each of the n points* $(\alpha_i, \beta_i)$*, for* $i = 1, \ldots, n$. *(The Interpolation Theorem guarantees that such a polynomial exists.) The output of the algorithm is the list of y-roots of Q*(*x, y*)*, i.e.,*

$$L = \{f(x) \in F[x] : (y - f(x)) \mid Q(x,y)\}$$

**Theorem 4-5**

*The output list contains every polynomial of degree* $\leq v$ *such that* $K(f, \beta) \geq K_m$. *Furthermore, the number of polynomials in the list is at most* $L_m$.

## 4.6 Koetter's Solution to the Interpolation Problem

In general terms, the interpolation problem is to construct a bivariate polynomial *Q*(*x, y*) with minimal (1, *v*)-degree that satisfies a number of constraints of the form

$$D_{r,s}Q(\alpha,\beta) = 0,$$

where $(r,s) \in \mathbb{N}^2$ and $(\alpha,\beta) \in \mathcal{F}^2$. It turns out that the mapping

$$Q(x,y) \to D_{r,s}Q(\alpha,\beta)$$

is an example of what is called a *linear functional* on $F[x,y]$. We consider the more general problem of constructing a bivariate polynomial *Q*(*x, y*) of minimal weighted-degree that satisfies a number of constraints of the form

$$D_i Q(x,y) = 0, for\ i = 1,2,\ldots,$$

where each $D_i$ is a linear functional. The goal of this section is to describe an algorithm for solving the more general problem [23].

### 4.6.1 Linear Functionals *F*[*x, y*]

A mapping $D : F[x,y] \to F$ is called a *linear functional* if

$D(\alpha P + \beta Q) = \alpha D(P) + \beta D(Q)$

for all $P, Q \in F[x,y]$ and $all\ \alpha, \beta \in \mathcal{F}$. The primary example of a linear functional is the mapping that evaluates a Hasse derivative:

$$Q(x,y) \to D_{r,s}Q(\alpha,\beta),$$

for fixed values of $(r, s) \in \mathbb{N}^2$ and $(\alpha, \beta) \in \mathcal{F}^2$.

If we agree on a particular monomial order, say

$$\phi_0(x,y) < \phi_1(x,y) < \cdots ,$$

so that any polynomial $Q(x,y)$ has a unique expansion of the form

$$Q(x,y) = \sum_{j=0}^{J} a_j \phi_j(x,y)$$

where $a_J \neq 0$, then any linear functional can be expressed as

$$D(Q) = \sum_{i=0}^{J} a_j d_j$$

where $d_j = D(\phi_j(x,y))$. The *kernel* of $D$ is defined to be the set

$$K = kerD = \{Q : D(Q) = 0\}$$

If $D$ is a linear functional with kernel $K$, the corresponding *bilinear mapping* $[P,Q]_D$ is defined as

$$[P,Q]_D \triangleq D(Q)P - D(P)Q$$

This simple mapping is a crucial part of the algorithms presented below; its key properties are given in the following lemma [23,24].

**Lemma 4-4**

For all P, Q in $F[x,y]$, $[\mathrm{P},\mathrm{Q}]_\mathrm{D} \in \mathrm{kerD}$. Furthermore, if $\mathrm{P} > \mathrm{Q}$ and $\mathrm{Q} \notin \mathrm{K}$, then $\mathrm{Rank}[\mathrm{P},\mathrm{Q}]_\mathrm{D} = \mathrm{Rank}\,\mathrm{P}$.

### 4.6.2 Problem Statement

Let $F_L[x,y]$ denote the set of polynomials from $F[x,y]$ whose *y*-degree is $\leq L$, i.e., those of the form

$$Q(x,y) = \sum_{k=0}^{L} q_k(x) y^k$$

where each $q_k(x) \in F[x]$. We note that $F_L[x,y]$ is an *F*[*x*]-module, i.e., if $Q(x,y) \in F_L[x,y]$,
and $p(x) \in F[x]$, then $p(x)Q(x,y) \in F_L[x,y]$ as well.

Let $D_1, \ldots, D_C$ be $C$ linear functionals defined on $F_L$ [*x*, *y*], and let $K_1, \ldots, K_C$ be the corresponding kernels, i.e.,

$$Ki = \{Q(x,y) \in F_L[x,y] : Di(Q) = 0\}$$

The *cumulative kernels* $\overline{K_0}, \ldots, \overline{K_C}$ are defined as follows: $\overline{K_0} = F_L[x, y]$ and for $i = 1, \ldots, C$,

$$\begin{aligned}\overline{K}_i &= \overline{K}_{i-1} \cap \overline{K}_i \\ &= K_1 \cap \ldots \cap K_i \\ &= \{Q(x, y)\} \in F_L[x, y]: D_1(Q) = \cdots = D_i(Q) = 0\}\end{aligned}$$

### 4.6.3 Generalized Interpolation Problem

*Construct a minimal element from* $\overline{K}_C = K_1 \cap \ldots \cap K_C$, *i.e., calculate*

$Q_0(x, y) \in min \{Q(x, y)\} \in F_L[x, y]: D_1(Q) = \cdots = D_i(Q) = 0\}$

**Koetter's Algorithm:**

Koetter [12,13] noticed, in effect, that if *the cumulative kernels are F*[*x*]*-modules*, generalized interpolation problem admits a less complex solution than the one afforded by the Feng-Tzeng algorithm [1,7,23].

This observation applies to the GS interpolation problem, since if we enforce the conditions $D_{r,s}(\alpha, \beta) = 0 \, for \, s + r < min$ an order in which ($r$−1, $s$) always precedes ($r$, $s$), the cumulative kernels will be $F$[$x$]-modules. For example, ($m$ − 1, 1) lex order, which orders the pairs as (0, 0), (0, 1), . . . , (0,$m$ − 1), (1, 0), (1, 1), . . . , (1,$m$ − 2), . . . , ($m$ − 1, 0) has the desired property.

In Koetter's algorithm, the set of monomials from $F_L[x, y]$,

$$\mathcal{M}_L[x, y] = \{ x^i y^j \ : \ 0 \leq i, 0 \leq j \leq L_, \}$$

is partitioned according to the exponent of $y$: $\mathcal{M}_L[x, y] = \cup_{j=0}^{L} \mathcal{M}_j$

where $\qquad \mathcal{M}_j = \{x^i y^j : i \geq 0\}$

This partition of $\mathcal{M}_L$ induces a partition on $F_L[x, y]$,: $F_L[x, y]$, $= S_0 \cup \cdot \cdot \cdot \cup S_L$, where

$$S_j = \{Q \in F_L[x, y] : LM(Q) \in \mathcal{M}_j \}$$

Koetter's algorithm generates a sequence of lists $G_0, G_1, \ldots, G_C$, with

$$G_i = (g_{i,0}, \ldots, g_{i,L}),$$

where $g_{i,j}$ is a minimal element of $\overline{K}_\iota \cap S_j$ . The algorithm's output is the polynomial

$$Q_0(x, y) = \min_{0 \leq j \leq L} g_{C,j}(x, y)$$

which is a minimal rank element of $\overline{K}_C$.

Koetter's algorithm is initialized as follows:

$$g_{0,j} = y^j, \quad i = 0, \ldots, L$$

Given $G_i, G_{i+1}$ is defined recursively:

$$J_0 = \{j : D_{i+1}(g_{i,j}) = 0\}$$

$$J_1 = \{j : D_{i+1}(g_{i,j}) \neq 0\}$$

If $J_1$ is not empty, among the polynomials $g_{i,j}$ with $j \in J_1$, let $g_{i,j^*}$ be the one with minimal rank; and temporarily denote $g_{i,j^*}$ by *f*:

$$f = \min_{j \in J_1} g_{i,j}$$

$$j^* = \operatorname{argmin}_{j \in J_1} g_{i,j}$$

Then using the notation of linear functionals, $g_{i+1,j}$ is defined for $j = 0, \ldots, L$,

$$g_{i+1,j} = \begin{cases} g_{i,j} & if\ j \in J_0 \\ [g_{i,j}, f]_{D_i+1} & if\ j \in J_1\ but\ j \neq j^* \\ [xf, f]_{D_i+1} & if\ j = j^* \end{cases}$$

**Theorem 4-6**

*For* $i = 0, \ldots, C$, *we have* $g_{i,j} = min\{g : g \in \overline{K}_i \cap Sj\} \qquad for\ j = 0, \ldots, L.$ [1]

**Algorithm 4-1 : Koetter's Interpolation for Guruswami-Sudan Decoder [1,8]**

```
$Input$: $Points\ (x_i, y_i), i = 1, \ldots, n. The\ interpolation\ order\ m_i; a\ (1, v) monomial\ order; L$
$Returns$: $Q_0(x, y) satisfying\ the\ interpolation\ problem$
$Initialize$: $g_j = y^j for\ j = 0, \ldots, L.$
    $for\ i = 1\ to\ n \quad (go\ from\ i-1\ st\ stage\ to\ ith\ stage)$
        $C = \frac{(m_i + 1)m_i}{2}$    $(Compute\ no. of\ derivatives\ involved)$
        $for\ (r, s) = (0,0)\ to\ (m_i - 1,0) by\ (m_i - 1,1) lex\ order\ from\ 1\ to\ C$
            $for\ j = 0\ to\ L$
                $\Delta_j = D_{(r,s)} g_j(x_i, y_i)$
            $end \quad (for\ j)$
            $J = \{j : \Delta_j \neq 0\}$    $(Set\ of\ non-zero\ discrepancies)$
            $if\ (J \neq \emptyset)$
$j^* = \arg\min\{g_j : j \in J\}$    $(polynomial\ of\ least\ weighted\ degree)$
                $f = g_{j^*} \quad ; \quad \Delta = \Delta_{j^*}$
            $for \quad j \in J$
                $if \ (j \neq j^*)$
                    $g_j = \Delta g_j - \Delta_j f$        $(update\ without\ change\ in\ rank)$
                $else\ if\ (j = j^*)$
                    $g_j = (x - x_i) f$        $(update\ with\ change\ in\ rank)$
                $end(if)$
            $end\ (for\ J)$
        $end\ \ (if\ J)$
    $end\ (for\ (r,s)\ )$
  $end\ (for\ i)$
$Q_0(x,y) = \min_j \{ g_j(x,y) \}$        $(least\ weighted\ degree)$
```

## 4.7 The Roth-Ruckenstein Solution to the Factorization Problem

The most efficient algorithm currently known for solving the factorization problem is due to Roth and Ruckenstein [25].

The factorization problem is this: given a polynomial $Q(x,y) \in F[x,y]$, find all polynomials $f(x)$ of degree $\leq v$ such that $(y - f(x)) \mid Q(x,y)$. Alternatively, find all $f(x) \in F_v[x]$ such that

$$Q(x, f(x)) \equiv 0.$$

If this condition holds, we call $f(x)$ a *y-root* of $Q(x,y)$. This section describes an algorithm due to Roth and Ruckenstein [1] for finding $y$-roots.

If $Q(x, y)$ is a two-variable polynomial such that $x^m \mid Q(x,y)$, but $x^{m+1} \nmid Q(x,y)$,define

$$\langle Q(x,y) \rangle \triangleq \frac{Q(x,y)}{x^m}$$

Although $Q(0,y)$ might be identically zero, nevertheless $\langle Q(0,y) \rangle$ is a nonzero polynomial in $y$.

Suppose

$$f(x) = a_0 + a_1 x + \cdot \cdot \cdot + a_v x^v$$

is a $y$-root of $Q(x, y)$. We will see that the coefficients $a_0, a_1 \ldots, a_v$ can be “picked off,” one at a time. As a start, the following lemma shows how to determine $a_0$ [1,8].

**Lemma 4-5**

If $(y - f(x)) \mid Q(x,y)$ then $y = f(0) = a_0$ is a root of the equation $Q_0(0,y) = 0$, where

$$Q_0(x,y) = \langle Q(x,y) \rangle.$$

We now proceed by induction, defining three sequences of polynomials $f_j(x), T_j(x,y),$ $and\ Q_j(x,y)$,for $j = 0,1,\dots,v,$ as follows.

Initially, $f_0 := f(x), Q_0(x,y) := \langle Q(x,y) \rangle$.

For $j \geq 1$ define

$$f_j(x) := \frac{f_{j-1}(x) - f_{j-1}(0)}{x} = a_j + \cdot \cdot \cdot + a_v x^{v-j}$$

$$T_j(x,y) := Q_{j-1}(x, xy + a_{j-1})$$

$$Q_j(x,y) := \langle T_j(x,y) \rangle$$

**Theorem 4-7**

*Given* $f(x) = a_0 + a_1 x + \cdots + a_v x^v \in F_v[x]$, *and* $Q(x,y) \in F[x,y]$, *define the sequences* $f_j(x)$ *and* $Q_j(x,y)$ *as above. Then for any* $j \geq 1, (y - f(x)) \mid Q(x,y)$ *if and only if* $(y - f_j(x)) \mid Q_j(x,y)$.

Here is the "picking off" theorem. [7,8]

**Corollary**

*If* $(y - f(x)) \mid Q(x,y)$ then $y = a_j$ is a root of the equation

$$Q_j(0,y) = 0, \quad for\ j = 0,\dots,v.$$

**Corollary**

If $y \mid Q_{v+1}(x,y)$, i.e., if $Q_{v+1}(x,0) = 0$,, then $f(x) = a_0 + a_1 x + \cdots + a_v x^v$ is a y-root of $Q(x,y)$.

The following Lemma provides some insight into the all-important transformation $Q(x, y) \rightarrow Q(x, xy + a)$

**Lemma 4-6**

If
$$Q(x,y) = \sum_i x^i g_i(y)$$
$$= \sum_{i,j} x^i y^j D_j g_i(0)$$

Then
$$Q(x, xy + a) = \sum_{i,j} x^i y^j D_j g_{i-j}(a)$$

where $D_i$ denotes the $i$th one-dimensional Hasse derivative.

Symbolically, this lemma can be summarized as follows:

$$Q(x,y) = \begin{bmatrix} g_0(0) & g_1(0) & g_2(0) & g_3(0) & \\ D_1g_0(0) & D_1g_1(0) & D_1g_2(0) & D_1g_3(0) & \cdots \\ D_2g_0(0) & D_2g_1(0) & D_2g_2(0) & D_2g_3(0) & \\ & & \vdots & & \end{bmatrix}$$

$$Q(x,xy+a) = \begin{bmatrix} g_0(a) & g_1(a) & g_2(a) & g_3(a) & \\ 0 & D_1g_0(a) & D_1g_1(a) & D_1g_2(a) & \cdots \\ 0 & 0 & D_2g_0(a) & D_2g_1(a) & \\ & & \vdots & & \end{bmatrix}$$

In words: if the entries of column *j* of *Q*(*x, y*) are interpreted as the coefficients of a polynomial, say $g_j(z)$, then the entries of the *j*th diagonal of *Q*(*x, xy*+*a*) are the coefficients of the polynomial $g_j(z+a)$ [1,23].

A pseudocode representation of the RR algorithm is given below. It takes as input a bivariate polynomial *Q*(*x, y*) and positive integer *D*, and returns as output the set of all *y*-roots of *Q*(*x, y*) of degree ≤ *D*. The strategy adopted by the algorithm is "depth-first search," as described in [1,25].

## 4.8 Roth-Ruckenstein Pseudo code for Finding y-roots of Q(x,y) [1]

```
Input : $Q(x,y), D$   (where $D$ is the maximum degree of $p(x)$ )
Output: List of polynomials $p(x)$ of degree $\le D$ such that $(y - p(x)) | Q(x,y)$
Initialization:
Set $p(x) = 0$ , $u = \deg(p) = -1$, $D$ = maximum degree (set as internal global)
Set up linked list where polynomials are saved.
Set $v = 0$ (the number of the node; global variable).
Call rothrucktree ( $Q(x,y), u, p$ )

Function rothrucktree ( $Q, u, p$ )
Input: $Q(x,y), p(x)$ and $u$ (degree of $p$)
Output: List of polynomials
 $v = v + 1$     (increment node number)
If ( $Q(x,y) = 0$ )
  Add $p(x)$ to the output list
end (if)
```

```
$else\ if\ (u < D)\quad (try\ another\ branch\ of\ the\ tree)$
  $R = list\ of\ roots\ of\ Q(0,y)$
  $for\ each\quad \propto \in R$
     $Q_{new}(x,y) = Q(x, xy+\propto)\qquad (shift\ the\ polynomial)$
     $p_{u+1} = \propto\qquad (new\ coefficient\ of\ p(x)\ )$
      $Call\ rothrucktree\ (\langle Q_{new}(x,y)\rangle, u+1, p)\qquad (\ recursive\ call)$
  $end\ (for)$
$else\ (leaf\ of\ tree\ reached\ with\ non-zero\ polynomial)$
  $(no\ output)$
$end\ if$
$end$
```

**CHAPTER 5**

# DIRECT SEQUENCE SPREAD SPECTRUM SYSTEMS AND CODE ACQUISITION

Spread spectrum is a communication technique which is widely used in the radar, navigation and telecommunication systems and playing a dominant role in the philosophy of the forthcoming generation of systems and networks. The amount of interest and research effort invested in this area is growing constantly especially after successful commercial success of Code division multiple access (CDMA) mobile telephone (IS-95) and the use of CDMA as the basic platform of 3G mobile radio [27].

The term Spread spectrum is today one of the most popular in the radio engineering and communication community. At the same time, it appears difficult to formulate an unequivocal and precise definition which distinguishes clearly between a spread spectrum and non-spread spectrum system.

A rather frequent way to explain the concept consists in the statement that a system or a signal is of spread spectrum type if its bandwidth significantly exceeds the minimum bandwidth necessary to send the information.

The very idea of a minimum bandwidth of information or message is full of ambiguity because there is no standard definition of bandwidth. A better definition is the one which incorporates Gabor's uncertainty principle [27].

A signal for which product of signal duration and bandwidth are of the order of 1 i.e. they are tightly linked together is called a "Plain" or "Non-spread spectrum signal". The only way to widen the bandwidth of a plain signal is to increase its bandwidth. On the other hand a deterministic signal for which time-bandwidth product is very greater than 1 and bandwidth can be governed independently of duration is called Spread Spectrum one. A system employing spread spectrum signals is a spread spectrum system.

An important difference that a spread spectrum modulated signal has over other conventional modulation techniques is that in spread spectrum modulation, the most precious resources of the communication channel i.e. bandwidth and power are sacrificed in order to achieve the goal of secure communications [15].

An important advantage of a spread-spectrum communication system is that it can provide immunity against externally generated (interfering) signals with finite power. The interference can be intentional as well as un-intentional. Protection against jamming waveforms is provided by purposely making the information bearing signal occupy a bandwidth far in excess of the minimum bandwidth necessary to transmit it. This has the effect of making the transmitted signal assume a noise-like appearance so as to blend into the background. The transmitted signal is thus enabled to propagate through the channel undetected by anyone who may be listening.

Spread spectrum systems were initially developed for military application, where resistance to jamming was of major concern. However, there are non-tactical applications which make use of beneficial attributes of a spread spectrum system. For example, it can be used to provide multipath rejection in ground-based mobile radio environment. Another application is in multiple-access communications in which a number of independent users are required to share a common channel without an external synchronizing mechanism [14].

## 5.1 Pseudo-Noise Sequences

All spread spectrum signals utilize some kind of a code which is independent of the data to spread the spectrum before transmission. These codes have special auto-correlation and cross-correlation properties and are called Pseudo-random noise (PN) codes because these sequences have white-noise like statistical properties while being obviously deterministic [15,16]. Thus, the sequence is "nearly random". The method most frequently used to generate pseudo-random codes is based on a feedback shift register.

Various spread spectrum systems can be classified based upon the exact point of usage of the PN sequence. On the transmitter end, they are used to increase the signal spectrum and hence called **Spreading**. On the receiver end, they are used to reduce the signal spectrum to its original bandwidth and hence called **Despreading**. The factor by which bandwidth of the signal is increased is called the **Processing Gain** of the system.

There are two categories regarding the length of codes:

### 5.1.1 Short codes

In this category, same PN-sequence is used for each data symbol i-e

$$N_c \,.\, T_c \;=\; T_S$$

Where

$N_c$ is the length of sequence

$T_c$ is the chip period

$T_S$ is the symbol period

### 5.1.2 Long codes

For long codes, the PN-sequence period is much longer than that of the data symbol so that a different chip pattern is associated with each symbol.

$$N_c \,.\, T_c \;\gg\; T_S$$

## 5.2 Properties of PN-sequences

PN-sequences of maximal length have a number of special properties possessed by a truly random binary sequence. A random binary sequence is a sequence in which the presence of binary symbol 1 or 0 is equally probable [14, 15]. Some properties of such sequences are as follows:

### 5.2.1 Balance property

In each period of the sequence the number of binary ones differs from the number of binary zeros by at most one digit (for $N_c$ odd).

$$Pn = +1 \;\; +1 \;\; +1 \;\; -1 \;\; +1 \;\; -1 \;\; -1 \qquad \sum = +1$$

### 5.2.2 Run length Property

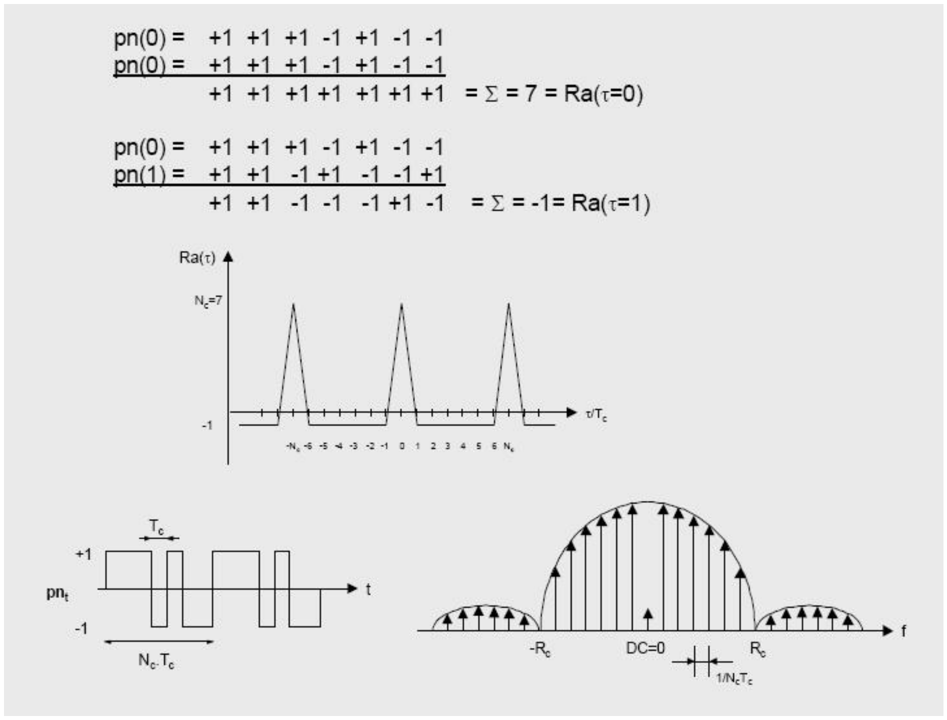
A “run” means a subsequence of identical symbols (1s or 0s) within one period of the sequence. The length of these subsequences is the length of the run. For maximal length PN sequences; among the runs of 1s and 0s in each period of a maximal-length sequence, one half the runs of each kind are of length one, one fourth are of length two, one eighth are of length three, and so on as long as these fractions represent positive number of runs. This property is called Run property. For a maximal length sequence generated by a linear feedback shift register of length $m$, the total number of runs is $(N+1)/2$ where $N = 2^m - 1$.

### 5.2.3 Autocorrelation

The auto-correlation function of a maximal-length PN sequence is periodic and binary-valued. The origin of the name pseudo-noise is that the sequence has an autocorrelation function which is very similar to that of a white noise signal. The autocorrelation function for the periodic sequence PN is defined as the number of agreements less the number of disagreements in a term by term comparison of one full period of the sequence with a cyclic shift (position τ) of the sequence itself.

$$R(\tau) = \int_{-\mathrm{N_c T_c}/2}^{\mathrm{N_c T_c}/2} \mathrm{p_n(t)p_n(t+\tau)dt}$$

The autocorrelation has a large peaked maximum only for perfect synchronization of two identical sequences. For a period of the maximal-length sequence, the auto-correlation function is somewhat similar to that of a random binary wave. The synchronization of the receiver is based on this property.



$$\begin{array}{lccccccc}
pn(0) = & +1 & +1 & +1 & -1 & +1 & -1 & -1 \\
\underline{pn(0) =} & \underline{+1} & \underline{+1} & \underline{+1} & \underline{-1} & \underline{+1} & \underline{-1} & \underline{-1} \\
 & +1 & +1 & +1 & +1 & +1 & +1 & +1 \quad = \Sigma = 7 = Ra(\tau=0)
\end{array}$$

$$\begin{array}{lccccccc}
pn(0) = & +1 & +1 & +1 & -1 & +1 & -1 & -1 \\
\underline{pn(1) =} & \underline{+1} & \underline{+1} & \underline{-1} & \underline{+1} & \underline{-1} & \underline{-1} & \underline{+1} \\
 & +1 & +1 & -1 & -1 & -1 & +1 & -1 \quad = \Sigma = -1 = Ra(\tau=1)
\end{array}$$

**Figure 5-1: Autocorrelation and Time/frequency domain representation of PN-sequence [14]**

### 5.2.4 Frequency spectrum

Periodicity of the PN sequence in the time domain is transformed into uniform sampling in the frequency domain. Its frequency spectrum has spectral lines which become closer to each other with increasing sequence length $N_c$. Each line is further smeared by data scrambling, which spreads each spectral line and further fills in between the lines to make the spectrum more nearly continuous. The DC component is determined by the zero-one balance of the PN-sequence.

### 5.2.5 Cross-correlation

Cross-correlation describes the interference between codes $\mathrm{p_{ni}}$ and $\mathrm{p_{nj}}$.

$$R_c(\tau) = \int_{-N_cT_c/2}^{N_cT_c/2} \mathrm{p_{ni}}(t)\mathrm{p_{nj}}(t+\tau)dt$$

It is a measure of agreement between two different codes $\mathrm{p_{ni}}$ and $\mathrm{p_{nj}}$. When the cross-correlation $R_c(\tau)$ is zero for all $\tau$, the codes are called orthogonal. In multi-user environment, users occupy the same RF bandwidth and transmit simultaneously. When the user codes are orthogonal, there is no interference between the users after de-spreading and the privacy of the communication of each user is protected.

In practice, codes are not perfectly orthogonal, hence the cross-correlation between user codes introduces performance degradation (increased noise power after de-spreading), which limits the maximum number of simultaneous users.

The construction or selection of proper sequences is not trivial. To guarantee efficient Spread Spectrum communications, the sequences must respect certain rules, such as length, auto-correlation, cross-correlation and bits balancing. The popular sequences include Barker, M-Sequence, Gold, Walsh etc. Every sequence has its own characteristics like gold codes have better cross-correlation properties so they are good for multi-user environment.

## 5.3 Types of Spread Spectrum Systems

Different Spread Spectrum techniques are distinguished according to the point in the system at which a pseudo-random code is inserted in the communication channel. This is illustrated in the figure below.

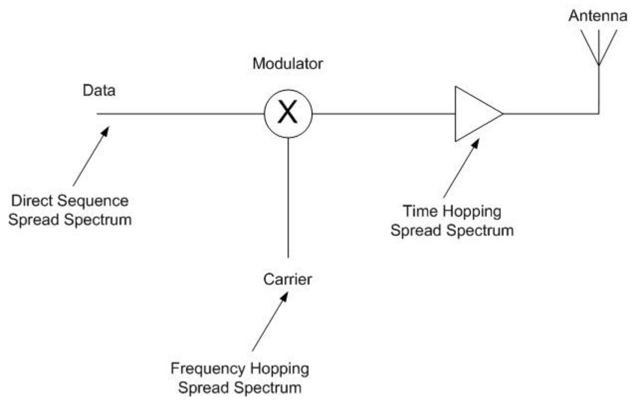

Antenna
Modulator
Data
X
Direct Sequence Spread Spectrum
Time Hopping Spread Spectrum
Carrier
Frequency Hopping Spread Spectrum

**Figure 5-2: Spreading techniques [14]**

If the PN sequence is inserted at the data level, we have the direct sequence form of spread spectrum (DSSS). If the PRN acts at the carrier-frequency level, we have the frequency hopping form of spread spectrum (FHSS). Applied at the local oscillator (LO) stage, FHSS PN codes force the carrier to change or hop according to the pseudo-random sequence. If the PRN acts as an on/off gate to the transmitted signal, we have a time hopping spread spectrum technique (THSS). There is also the chirp technique, which linearly sweeps the carrier frequency in time. Our topic of discussion for the rest of this chapter is the acquisition of DSSS signals in the receiver.

## 5.4 Direct Sequence Spread Spectrum (DSSS)

Direct Sequence Spread Spectrum transmissions multiply the data being transmitted by a "noise" signal. This noise signal is a pseudorandom sequence of 1 and −1 values, at a frequency much higher than that of the original signal, thereby spreading the energy of the original signal into a much wider band.

The resulting signal resembles white noise which can be filtered out at the receiving end to recover the original data, by again multiplying the same pseudorandom sequence to the received signal. Spreading operation can be summarized as:

The binary data $d_t$with symbol rate $R_s = 1/T_s$ is multiplied with the pseudo-noise code $pn_t$ with chip rate $R_c = 1/T_c$ to produce the transmitted baseband signal $tx_b$.

$$tx_b = d_t \cdot pn_t$$

The effect of multiplication of $d_t$ with the PN-sequence is to spread the baseband bandwidth $R_s$ of $d_t$ to a baseband bandwidth of $R_c$. Following figure illustrates this phenomenon.

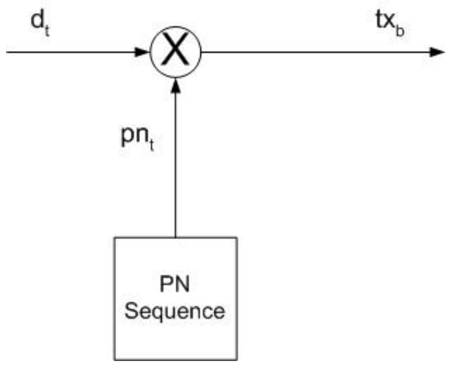


**Figure 5-3: DSSS transmitter end [14]**

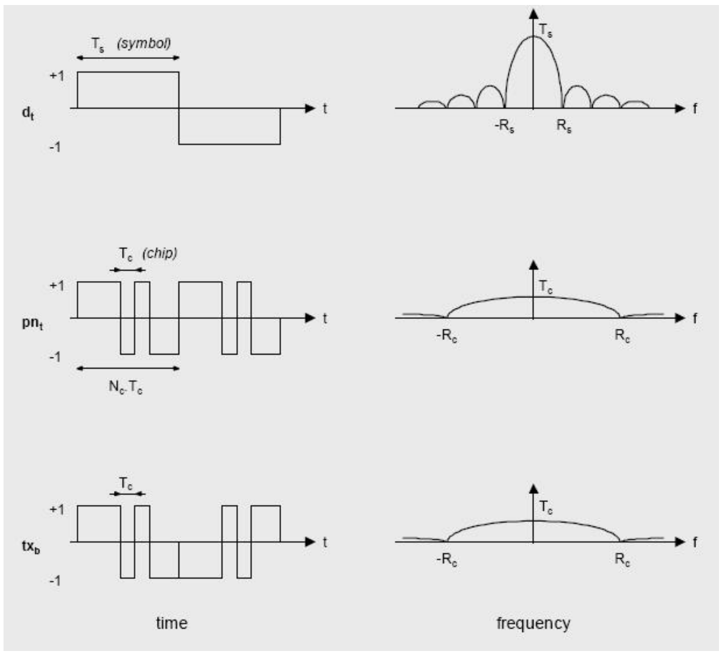


**Figure 5-4: DSSS Spreading [14]**

The bandwidth expansion factor or processing gain, being the ratio of chip rate Rc and the data symbol rate Rs, is usually selected to be an integer in practical spread spectrum systems.

$$G_p = \frac{BW_{ss}}{BW_d} = \frac{R_c}{R_s} = \frac{T_s}{T_c} = N_c$$

The de-spreading operation can be summarized as:

At the receiver, the received baseband signal $r_{xb}$ is multiplied with the PN-sequence $pn_r$. If $pn_r = pn_t$ and synchronized to the PN-sequence in the received data, then the recovered binary data is produced on $d_r$.

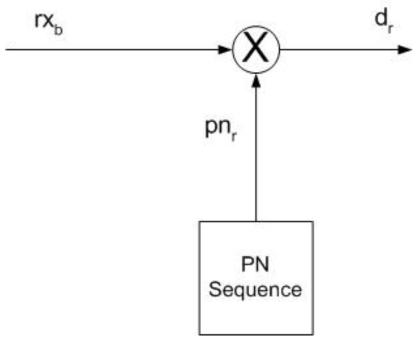


**Figure 5-5: DSSS Receiver end [14]**

$$\begin{aligned}
d_r &= rx_b.pn_r \\
d_r &= (d_t.pn_t).pn_r \\
d_r &= (d_t.pn_t).pn_t \because pn_r = pn_t \\
d_r &= d_t \because pn_t.pn_t = 1
\end{aligned}$$

Last equation holds only if the sequences are perfectly synchronized with each other. The effect of multiplication of the spread spectrum signal $rx_b$ with the PN-sequence $pn_t$ used in the transmitter is to de-spread the bandwidth of $rx_b$ to $R_s$. This is illustrated in the following figure.

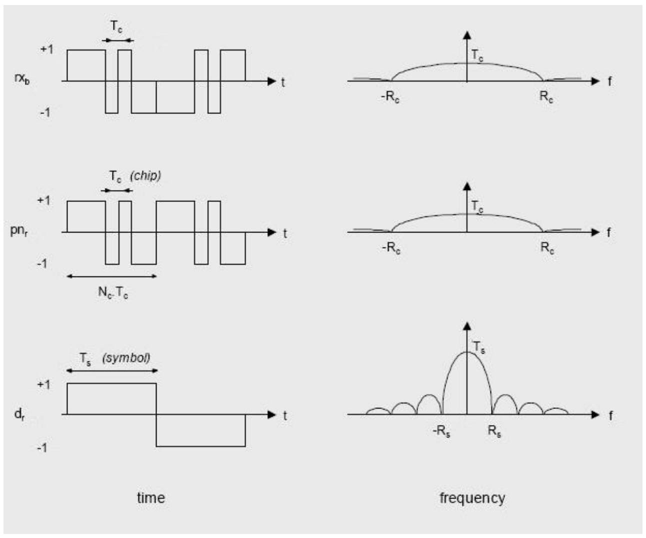


**Figure 5-6: Direct-Sequence de-spreading [14]**

If $pn_r \neq pn_t$, then there is no de-spreading action. The multiplier output becomes:

$$\begin{aligned}
d_r &= rx_b.pn_r \\
d_r &= (d_t.pn_t).pn_r
\end{aligned}$$

In the receiver, detection of the desired signal is achieved by correlation against a local reference PN-sequence. For secure communications in a multi-user environment, the transmitted data $d_t$ may not be recovered by a user that doesn't know the PN-sequence $pn_t$ used at the transmitter. Therefore the cross-correlation between all PN-sequences used for multi-user transmission should be ideally zero. If this is achieved then the output of the correlator used in the receiver is approximately zero for all except the desired transmission.

## 5.5 DSSS acquisition

One of the most characteristic problems in spread spectrum technology is measuring the time of arrival and frequency of the received signal [27]. In the systems where spread spectrum signals are used for ranging and measurement of object motion parameters (radar, sonar, and navigation), time-frequency estimation is the main task. In spread spectrum communications, it is the core of the timing recovery procedure. In fact to correctly demodulate the transmitted data, a receiver of every digital communication system must know with sufficient accuracy the border of symbols, frames etc. in the received data stream. In other words, the local receiver clock should be properly synchronized with the received data stream. The initial acquisition of the correct phase offset is referred to as the coarse acquisition. The subsequent tracking, once coarse acquisition has been achieved is sometimes called fine acquisition. Coarse acquisition consists of searching the time/frequency space illustrated in the figure below.

**Figure 5-7 Time/Frequency Search Space associated with coarse sync/ acquisition**

In case of DSSS systems, fortunately, the carrier frequency is generally known in advance and the search is needed only in the time dimension.[26] Coarse acquisition attempts to adjust the phase offset of the locally generated pseudo random sequence to within a large fraction of one chip time, $T_C$.

### 5.5.1 Search Strategies for Acquisition:

Irrespective of the code used, the code space must be searched in some fashion to find the correct phase offset. There are several ways to accomplish this.

#### 5.5.1.1 Serial Search

The simplest is a serial approach where one phase offset at a time is attempted and the comparison with the threshold is made. If the sequence length is large, however, this approach can be very slow if bounds on the search space are not available.[16]

#### 5.5.1.2 Parallel Search

On the other hand, a fully parallel search which is the fastest way can also be performed. In this architecture, N parallel matched filters would simultaneously search the code space, one offset for each matched filter. The filter with the largest output would correspond to the correct phase offset. If N is large, implementation would be prohibitive, however [16].

#### 5.5.1.3 Multi-dwell Approach

Between these approaches, fully serial and fully parallel, there are compromises that can be made. Instead of a fully parallel implementation, for example, some smaller number of parallel matched filters could also be included.[26] Multi-dwell search is one such approach. The first correlator implements a relatively low threshold with a short integration time. Its purpose is to quickly eliminate offsets that are not acceptable. This stage would have a relatively high false alaram rate but corresponding high probability of detection. The second correlator implements a small false alarm rate and small probability of miss, and therefore a large acquisition time. The goal is to have the first stage hand off to the second infrequently so the overall acquisition time is minimized.

## 5.6 Correlation in Frequency Domain:

Time average cross-correlation of two sequences $r(n)$ and $p(n)$ for a lag of $\tau$ is defined as follows $$c(\tau) = \sum r(n)p(n+\tau)$$

where the summation is taken over all the non-zero values of $r(n)$ and $p(n)$. Let the number of such indices be $N$. However, this correlation can be done with much less computations in the frequency domain by following relationship

$$c(\tau) = IFT\ (\ \ FFT\ (r(n)) * \overline{FT}\big(p(n)\big)\ \ )$$

where IFFT denotes operation of inverse Fourier transform and FT means Forward Fourier transform and a bar over it represents its complex conjugate. In order to compute correlation for a single lag in time domain, we have to perform $N^2$ multiplications and $N-1$ additions while in the case of FFT based approach we can compute correlation for all the lags with only $N\log_2 N$ complex additions and multiplications. So, computing correlation in frequency domain is much simpler computationally when compared with time domain calculations. Following section explains how this scheme is implemented.

## 5.7 Implementation Details:

For HDL implementation of DSSS acquisition, parallel search technique is used. Correlation of the local reference signal with the received signal is performed in the frequency domain. Length of the selected PN sequence is 512 chips. Received signal is stored in a read only memory (ROM) having a depth of 1024 (corresponding of two periods of the PN sequence) and width of 16 bits. Both real and imaginary parts of the conjugate of the Fourier Transform of the local reference PN noise are also stored in two separate ROMs. Each of these ROMs has a depth of 512 and precision of 16 bits. Correlation is performed for two periods of the PN sequence. Overall block diagram is shown in Figure 5-8. FFT is computed by using the built-in Xilinx Intellectual Property (IP) core in the Radix-2 Burst I/O mode of operation. For the product specifications see [11]. The FFT result for the received sequence is shown in the Figure 5-9.

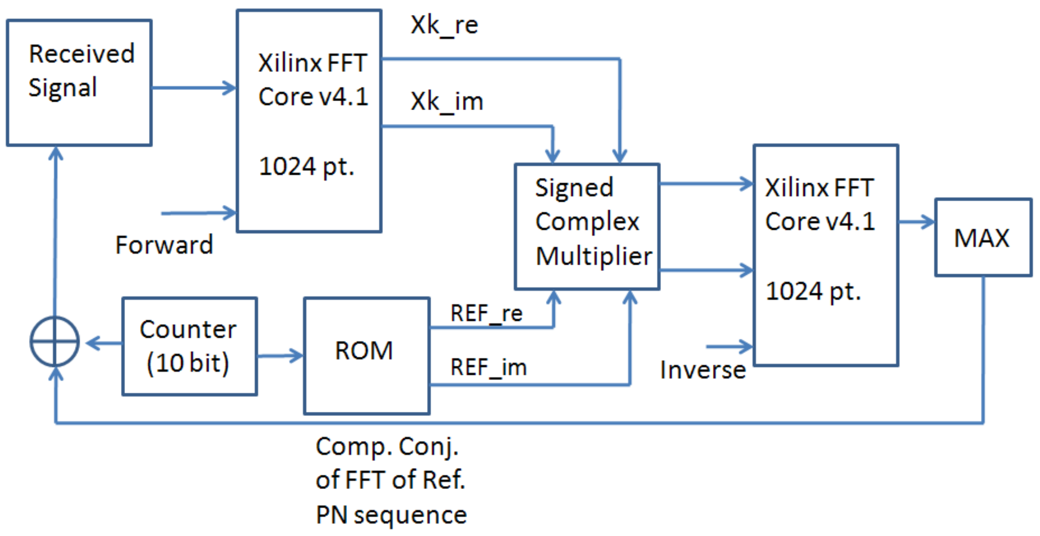
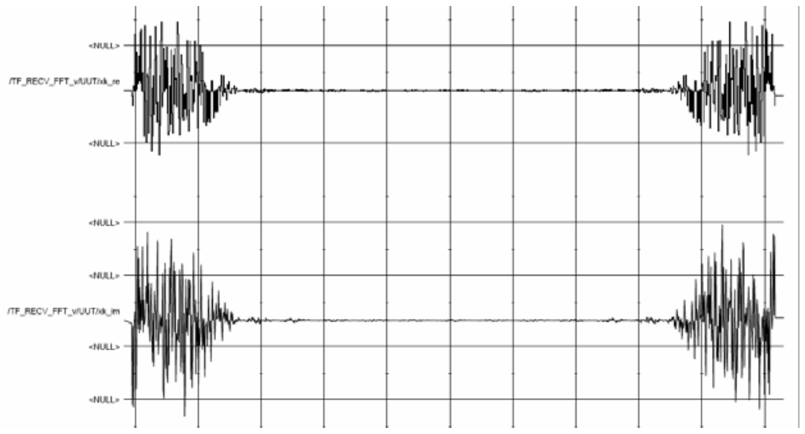

Received Signal
Xilinx FFT Core v4.1 1024 pt.
Xk_re
Xk_im
Forward
Signed Complex Multiplier
Xilinx FFT Core v4.1 1024 pt.
MAX
Counter (10 bit)
ROM
REF_re
REF_im
Inverse
Comp. Conj. of FFT of Ref. PN sequence

**Figure 5-8: Parallel search acquisition architecture for DSSS systems.**



**FFT (Real part)**

**FFT (Imag. part)**

**Figure 5-9 Fast Fourier Transform of the Received signal**

**Product (Real part)**

**Product (Imag. part)**

**Figure 5-10 Real and Imaginary Parts of the Complex Multiplier output**

The computed FFT of the received sequence and stored FFT of the local reference sequence are multiplied together using a complex multiplier. Modelsim simulation waveform for the product is shown in the Figure 5-10.

Then we take the inverse Fourier Transform of this product to get the result for the received sequence cross-correlation with the local reference PN sequence. Modelsim waveform for this computed correlation and its maximum correlation lag is shown in the Figure 5-11.

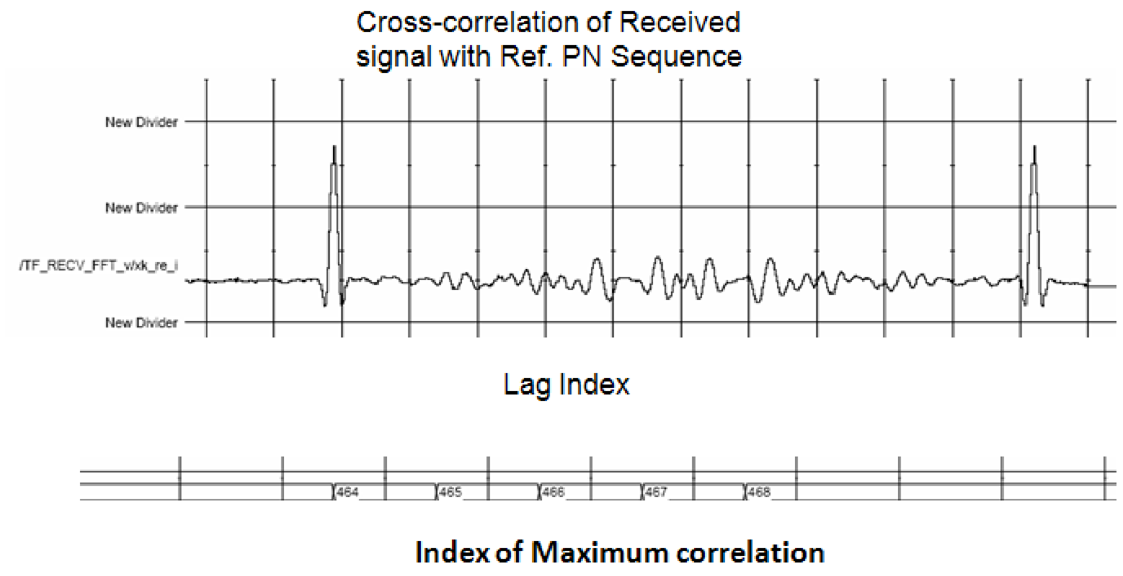


**Figure 5-11: Cross-correlation of Received signal with Reference PN sequence**

Once the maximum index of correlation is available, we add it to the address counter of the received signal ROM to remove the offset of the two signals. This completes the process of code acquisition.

**CHAPTER 6**

# SIMULATION AND IMPLEMENTATION OF REED SOLMON CODEC ARCHITECTURES

Area efficient and high speed VLSI architectures for encoding and decoding Reed–Solomon codes with the Berlekamp–Massey algorithm are presented in this chapter. The speed bottleneck in the Berlekamp–Massey algorithm is in the iterative computation of *discrepancies* followed by the updating of the error-locator polynomial [6]. This bottleneck can be eliminated via a series of algorithmic transformations that result in a fully systolic architecture in which a single array of processors computes both the error-locator and the error-evaluator polynomials. In contrast to conventional Berlekamp–Massey architectures in which the critical path passes through two multipliers and $1 + \lceil \log_2 t + 1) \rceil$ adders, the critical path in reformulated inversion-less Berlekamp Massey architectures passes through only one multiplier and one adder, which is comparable to the critical path in architectures based on the extended Euclidean algorithm [3,4].

## 6.1 Arithmetic Operations in Galois Field

Before discussing Reed Solomon codec architecture, we discuss how addition and multiplication is performed in the Galois Field GF($2^m$).

### 6.1.1 Addition in Galois Field GF($2^m$)

Addition and subtraction are same in GF($2^m$). Addition is performed by expressing both the operands in the polynomial representation. Then we take bit- by bit exclusive-or (XOR) of the corresponding bits to get the result of addition [1,10].

### 6.1.2 Multiplication in Galois Field GF($2^m$)

Multiplication of GF($2^m$) is bit more completed. We define the primitive polynomial of the field and its root is known as the primitive element (we can express all the non-zero elements of the field as powers of the primitive element.) We express both the multiplier and the multiplicand as the powers of the primitive element. Let $\alpha$ be the primitive element and $\alpha^i$ and $\alpha^j$ be the two operands. Then the product is defined as follows:

$$\alpha^l = \alpha^i \alpha^j = \alpha^{i+j} = \alpha^{(i+j) mod (2^m - 1)}$$

We present an example of the design of a GF($2^4$) multiplier. Let $\alpha$ be the primitive element of the field corresponding to the primitive polynomial

$$g(x) = 1 + x + x^4$$

As $\alpha$ is a root of the primitive polynomial and addition and subtraction are same operation in the GF($2^m$), $\alpha^4 = \alpha + 1$

In order to develop an architecture for a GF($2^4$) multiplier , we first consider an arbitrary field element $\beta \epsilon GF(2^m)$ which is to multiplied with the primitive element $\alpha$ of the field. Polynomial representation of $\beta$ in terms of $\alpha$ is as follows

$$\beta = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3$$

Multiplying it with $\alpha$, we get

$$\alpha\beta = \alpha * (b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3)$$

$$= b_0\alpha + b_1\alpha^2 + b_2\alpha^3 + b_3\alpha^4$$

But $\alpha^4 = \alpha + 1$

So, $\alpha\beta = b_0\alpha + b_1\alpha^2 + b_2\alpha^3 + b_3(\alpha + 1)$

$$\alpha\beta = b_3 + (b_0 + b_3)\alpha + b_1\alpha^2 + b_2\alpha^3$$

This alpha–gain block is shown in Figure 6-1.

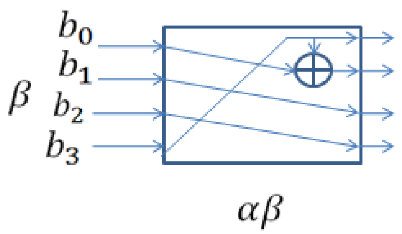


**Figure 6-1: An alpha-gain block for GF($2^4$)**

Now we consider multiplying two arbitrary field elements $\beta$ and $\gamma \in GF(2^4)$ expressed in the polynomial form as follows:

$$\beta = b_0 + b_1\alpha + b_2\alpha^2 + b_3\alpha^3$$

$$\gamma = c_0 + c_1\alpha + c_2\alpha^2 + c_3\alpha^3$$

Their product can be expressed in the Horner notation as follows:

$$\beta\gamma = (((c_3\beta)\alpha + c_2\beta)\alpha + c_1\beta)\alpha + c_0\beta$$

This expression and the alpha-gain multiplier can be used to design the multiplier for $GF(2^4)$ shown in the Figure 6-2.

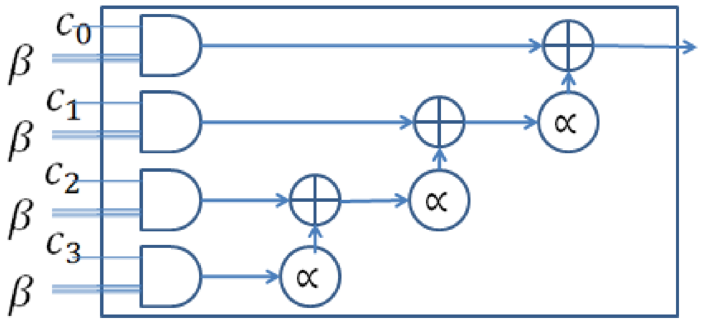


**Figure 6-2: Parallel-in parallel-out GF ($2^4$) Multiplier**

Addition and Multiplication tables for GF ($2^4$) are shown in Table 6-1.

| | zlog: | | – | – | 4 | 1 | 8 | 2 | 10 | 5 | 14 | 3 | 7 | 9 | 13 | 6 | 12 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | log: | | – | 0 | 1 | 4 | 2 | 8 | 5 | 10 | 3 | 14 | 9 | 7 | 6 | 13 | 11 | 12 |
| | | | 0 | 1 | $\alpha$ | $\alpha^4$ | $\alpha^2$ | $\alpha^8$ | $\alpha^5$ | $\alpha^{10}$ | $\alpha^3$ | $\alpha^{14}$ | $\alpha^9$ | $\alpha^7$ | $\alpha^6$ | $\alpha^{13}$ | $\alpha^{11}$ | $\alpha^{12}$ |
| | | | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |
| 0 | **0** | × | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | **1** | + | 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $\alpha$ | **2** | | 2 | 3 | 4 | 6 | 8 | 10 | 12 | 14 | 3 | 1 | 7 | 5 | 11 | 9 | 15 | 13 |
| $\alpha^4$ | **3** | | 3 | 2 | 1 | 5 | 12 | 15 | 10 | 9 | 11 | 8 | 13 | 14 | 7 | 4 | 1 | 2 |
| $\alpha^2$ | **4** | | 4 | 5 | 6 | 7 | 3 | 7 | 11 | 15 | 6 | 2 | 14 | 10 | 5 | 1 | 13 | 9 |
| $\alpha^8$ | **5** | | 5 | 4 | 7 | 6 | 1 | 2 | 13 | 8 | 14 | 11 | 4 | 1 | 9 | 12 | 3 | 6 |
| $\alpha^5$ | **6** | | 6 | 7 | 4 | 5 | 2 | 3 | 7 | 1 | 5 | 3 | 9 | 15 | 14 | 8 | 2 | 4 |
| $\alpha^{10}$ | **7** | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 6 | 13 | 10 | 3 | 4 | 2 | 5 | 12 | 11 |
| $\alpha^3$ | **8** | | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 12 | 4 | 15 | 7 | 10 | 2 | 9 | 1 |
| $\alpha^{14}$ | **9** | | 9 | 8 | 11 | 10 | 13 | 12 | 15 | 14 | 1 | 13 | 5 | 12 | 6 | 15 | 7 | 14 |
| $\alpha^9$ | **10** | | 10 | 11 | 8 | 9 | 14 | 15 | 12 | 13 | 2 | 3 | 8 | 2 | 1 | 11 | 6 | 12 |
| $\alpha^7$ | **11** | | 11 | 10 | 9 | 8 | 15 | 14 | 13 | 12 | 3 | 2 | 1 | 9 | 13 | 6 | 8 | 3 |
| $\alpha^6$ | **12** | | 12 | 13 | 14 | 15 | 8 | 9 | 10 | 11 | 4 | 5 | 6 | 7 | 15 | 3 | 4 | 8 |
| $\alpha^{13}$ | **13** | | 13 | 12 | 15 | 14 | 9 | 8 | 11 | 10 | 5 | 4 | 7 | 6 | 1 | 14 | 10 | 7 |
| $\alpha^{11}$ | **14** | | 14 | 15 | 12 | 13 | 10 | 11 | 8 | 9 | 6 | 7 | 4 | 5 | 2 | 3 | 11 | 5 |
| $\alpha^{12}$ | **15** | | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 10 |

**Table 6-1: Addition and Multiplication Tables for GF(16) [1]**

## 6.2 An overview of Reed Solomon Codes

Let $(d_{k-1}, d_{k-2}, \dots \dots \dots \dots, d_1, d_0)$ denote $k$ $m$-bit *data symbols* (bytes) that are to be transmitted over a communication channel (or stored in memory). These bytes are regarded as elements of the finite field (also called Galois field), GF $(2^m)$, and encoded into a *codeword* $(c_{n-1}, c_{n-2}, \dots \dots \dots \dots .., c_1, c_0)$ of n > k bytes.

For Reed–Solomon codes over GF $(2^m)$, $n = 2^m - 1, k \; is \; odd$ and the code can correct $t = (n-k)/2$ byte errors. The encoding process is best described in terms of the *data polynomials*

$$D(z) = d_{k-1}z^{k-1} + d_{k-2}z^{k-2} + \dots \dots d_1 z + d_0$$

being transformed into a *codeword polynomial*

$$C(z) = c_{n-1}z^{n-1} + c_{n-2}z^{n-2} + \dots \dots c_1 z + c_{0.}$$

All codeword polynomials *C (z)* are polynomial multiples of G $(z)$, the *generator polynomial* of the code, which is defined as

$$G(z) = \prod_{i=0}^{2t-1}(z - \propto^{m_0+i}) \qquad \text{Eq. ( 6-1)}$$

where $m_0$ is typically zero or one [1,6]. Since $2t$ consecutive powers $\propto^{m_0}, \propto^{m_0+1}, \dots \dots, \propto^{m_0+2t-1}$ of $\propto$ are roots of $G(z)$, and *C(z)* is a multiple of *G(z)*, it follows that $C(\propto^{m_0+i}) = 0, \quad 0 \le i \le 2t-1$ Eq. ( 6-2)

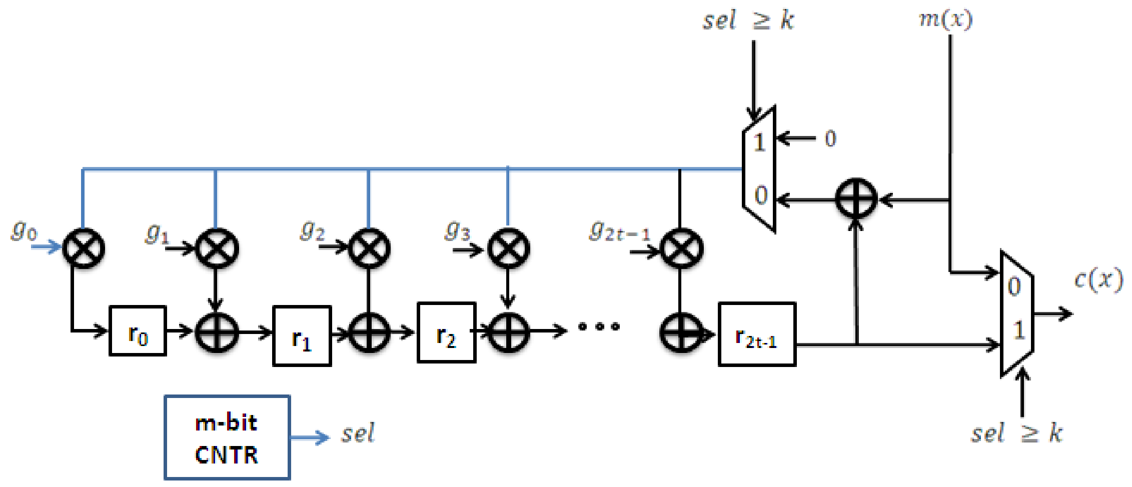


**Figure 6-3: Reed Solomon Systematic Encoder Architecture [1]**

for all codeword polynomials *C(z)* . In fact, an arbitrary polynomial of degree less than *n* is a codeword polynomial if and only if it satisfies Eq. ( 6-2).

A *systematic* encoding produces codewords that are comprised of data symbols followed by *parity-check symbols* and is obtained as follows. Let *Q(z)* and *P(z)* denote the quotient and remainder respectively when the polynomial $z^{n-k}D(z)$ of degree *n-1* is divided by *G(z)* of degree *2t = n - k* . Thus, $z^{n-k}D(z) = Q(z)G(z) + P(z)$ where $\deg(P(z)) < n-k$. Clearly, $Q(z)G(z) = z^{n-k}D(z) - P(z) = C(z)$ is a multiple of $G(z)$. Furthermore, since the lowest degree term in $z^{n-k}D(z)$ is $d_0 z^{n-k}$ while $P(z)$ is of degree at most $n-k+1$ , it follows that the codeword is given by

$$(c_{n-1}, c_{n-2}, \dots \dots, c_1, c_0)$$
$$= (d_{k-1}, d_{k-2}, \dots \dots \dots \dots .., d_1, d_0,$$
$$-p_{n-k-1}, -p_{n-k-2}, \dots \dots \dots \dots .., -p_1, -p_0)$$

and consists of the data symbols followed by the parity-check symbols.

### 6.2.1 Decoding of Reed–Solomon Codes

Let *C(z)* denote the transmitted codeword polynomial and let *R(z)* denote the received word polynomial. The input to the decoder is *R(z)* , and it assumes that

$$R(z) = C(z) + E(z)$$

where, if *e≥0* errors have occurred during transmission, the error polynomial *E(z)* can be written as

$$E(z) = Y_1 z^{i_1} + Y_2 z^{i_2} + \dots \dots + Y_e z^{i_e}$$

It is conventional to say that the *error values* $Y_1, Y_2, \dots \dots, Y_e$, occurred at the error locations $X_1 = \propto^{i_1}, X_2 = \propto^{i_2}, \dots\dots, X_e = \propto^{i_e}$. Note that the decoder does not know *E(z)* ; in fact, it does not even know the value of *e*. The decoder's task is to determine *E(z)* from its input *R(z)* , and thus correct the errors by subtracting off *E(z)* from *R(z)* . If $e \leq t$, then such a calculation is always possible, that is *t*, or fewer errors can always be corrected [10].

The decoder begins its task of error correction by computing the *syndrome values*

$$s_i = R(\propto^{m_0+i}) = C(\propto^{m_0+i}) + E(\propto^{m_0+i}) = E(\propto^{m_0+i}) \qquad 0 \leq i \leq 2t-1$$

Eq. ( 6-3)

If all $2t$ syndrome values are zero, then *R(z)* is a codeword and it is assumed that *C(z)* = *R(z)* that is, no errors have occurred. Otherwise, the decoder knows that and uses the *syndrome polynomial* , which is defined to be

$$S(z) = s_0 + s_1 z + \dots + s_{2t-1} z^{2t-1}$$

to calculate the error values and error locations. Define the *error-locator* polynomial $\Lambda(z)$ of degree $e$ and the *error evaluator* polynomial $\Omega(z)$ of degree $e-1$ at most to be

$$\Lambda(z) = \prod_{j=1}^{e} (1 - X_j z) = 1 + \lambda_1 z + \lambda_2 z^2 + \dots + \lambda_e z^e \qquad \text{Eq. ( 6-4)}$$

$$\boldsymbol{\Omega(z) = \sum_{i=1}^{e} Y_i X_i^{m_0} \prod_{j=1, j\neq i}^{e} (1 - X_j z)} = 1\ \boldsymbol{\omega_0 + \omega_1 z + \omega\lambda_2 z^2 + \cdots + \omega_{e-1} z^{e-1}}$$

Eq. ( 6-5)

These polynomials are related to *S(z)* through the *key equation* [1], [3]:

$$\boldsymbol{\Lambda}(z)\ S(z) \equiv \boldsymbol{\Omega(z) mod\ \ z^{2t}} \qquad \text{Eq. ( 6-6)}$$

Solving the key equation to determine both $\Lambda(z)$ and $\Omega(z)$ from *S(z)* is the hardest part of the decoding process. The **BM** algorithm and the **eE** algorithm can be used to solve Eq. ( **6-6**). If $e \leq t$, these algorithms find $\Lambda(z)$ and $\Omega(z)$, but if $e > t$ , then the algorithms almost always fail to find $\Lambda(z)$ and $\Omega(z)$. Fortunately, such failures are usually easily detected [6].

Once $\Lambda(z)$ and $\Omega(z)$ have been found, the decoder can find the error locations by checking whether $\Lambda(\propto^{-j}) = 0$ for each $j$, $0 \leq j \leq n-1$ . Usually, the decoder computes the value of $\Lambda(\propto^{-j})$ just before the $j$-th received symbol leaves the decoder circuit. This process is called a *Chien search* [1]. If $\Lambda(\propto^{-j}) = 0$ , then $\propto^{j}$ is one of the error locations (say $X_i$). In other words, $r_j$ is in error, and needs to be corrected before it leaves the decoder. The decoder can calculate the error value $Y_i$ to be subtracted from $r_j$ via Forney's error value formula [1]

$$Y_i = -\frac{X_i^{-(m_0-1)}\Omega(X_i^{-1})}{\Lambda'(X_i^{-1})} = -\left.\frac{z^{m_0}\Omega(z)}{z\Lambda'(z)}\right|_{z=\alpha^{-j}} \qquad \text{Eq. ( 6-7)}$$

where $\Lambda'(z) = \lambda_1 + 2\lambda_2 z + 3\lambda_3 z^2 \ldots + e\lambda_e z^{e-1}$ denotes the formal derivative of $\Lambda(z)$ . Note that the formal derivative simplifies to $\Lambda'(z) = \lambda_1 + \lambda_3 z^2 \ldots$ since we are considering codes over GF ($2^m$). Thus, $z\Lambda'(z) = \lambda_1 z + \lambda_3 z^3 + \cdots$ which is just the terms of odd degree in $\Lambda(z)$ . Hence, the value of $z\Lambda'(z)$ at $z = \alpha^{-j}$ can be found during the evaluation of $\Lambda(z)$ at $z = \alpha^{-j}$ and does not require a separate computation. Note also that Eq. ( 6-7) can be simplified by choosing $m_0 = 0$ .

## 6.3 Reed–Solomon Decoder Structure

In summary, a Reed–Solomon decoder consists of three blocks which are shown in Figure 6-4:

1. the syndrome computation (**SC**) block
2. the key-equation solver (**KES**) block
3. the Chien search and error evaluator (**CSEE**) block

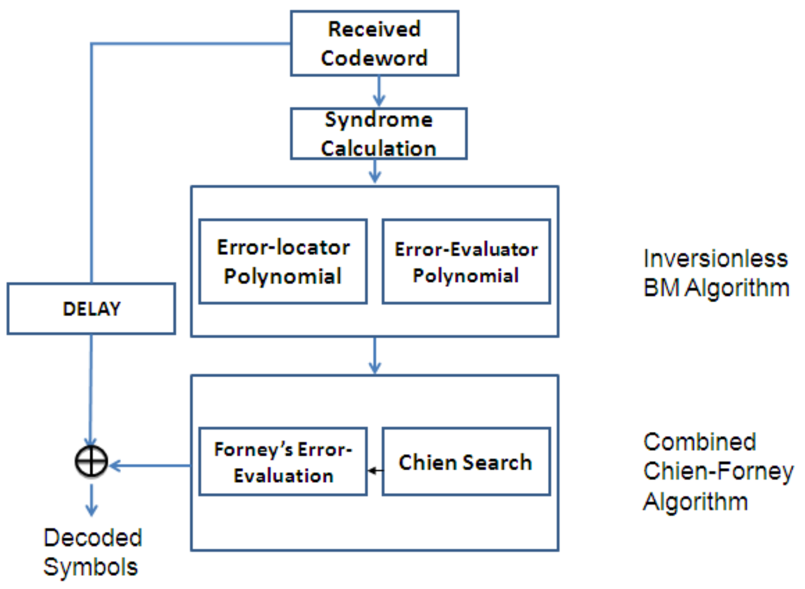

Received Codeword
Syndrome Calculation
Error-locator Polynomial
Error-Evaluator Polynomial
Inversionless BM Algorithm
DELAY
Forney's Error-Evaluation
Chien Search
Combined Chien-Forney Algorithm
Decoded Symbols

**Figure 6-4: Reed Solomon Decoder Block Diagram**

These blocks usually operate in pipelined mode in which three blocks are separately and simultaneously working on three successive received words.

### 6.3.1 Syndrome Computation Block

The **SC** block computes the syndromes via **Eq. ( 6-3)** usually as the received word is entering the decoder. The SC architecture is shown in the Figure 6-5 which uses multiply accumulate blocks. The incoming received word enters serially symbol by symbol and gets multiplied with the roots of the generator polynomial and the result is accumulated for each clock cycle. At the end of n clock cycles, last symbol of the received word enters the SC block and the result is 2t syndrome values.

The syndromes are passed to the **KES** block which solves Eq (6-6) to determine the error locator and error evaluator polynomials. KES block and its various architectures will be discussed in detail in the following section.

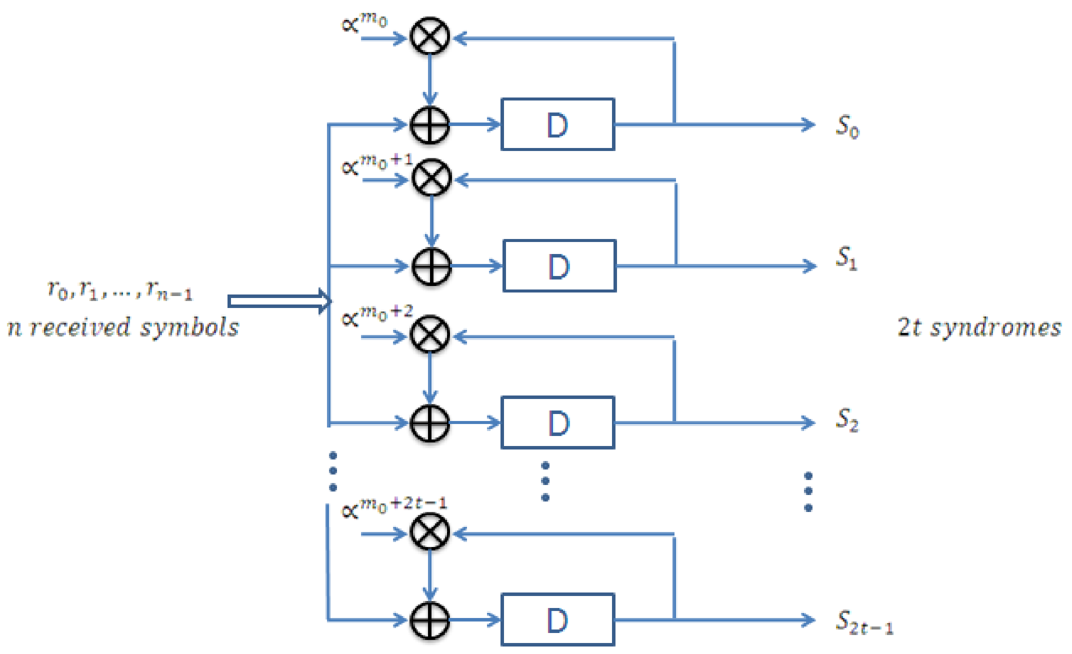


**Figure 6-5: Syndrome Computation (SC) Block [13]**

### 6.3.2 Chein Search and Error-Evaluator Block

These polynomials are then passed to the **CSEE** block, which calculates the error locations and error values via Eq. ( 6-7) and corrects the errors as the received word is being read out of the decoder. Chein search block can be implemented as shown in the Figure 6-6

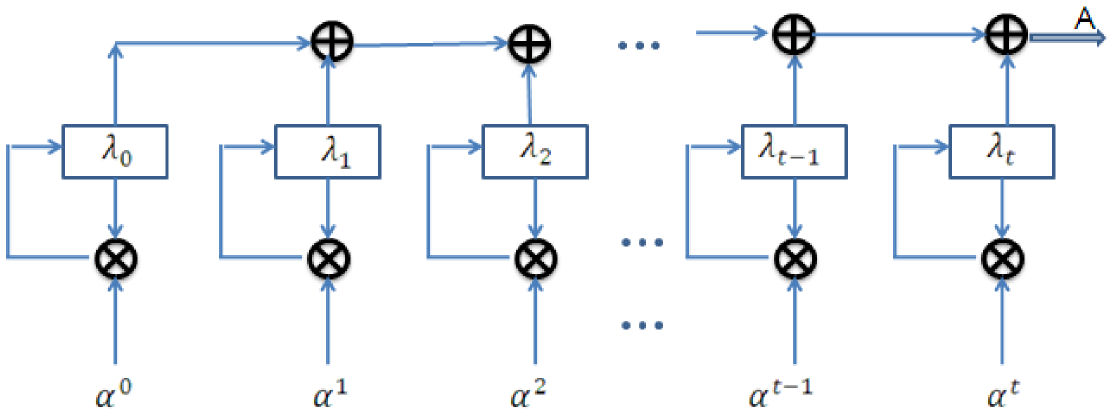
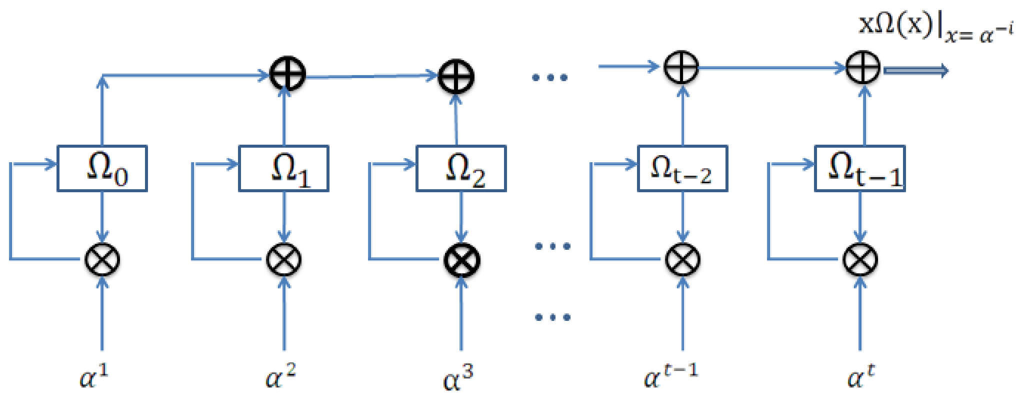


**Figure 6-6: Chein Search (CS) Block [3]**

Error-values are evaluated using Forney's Formula. The numerator of the Forney's formula can be computed using block diagram of Figure 6-7.

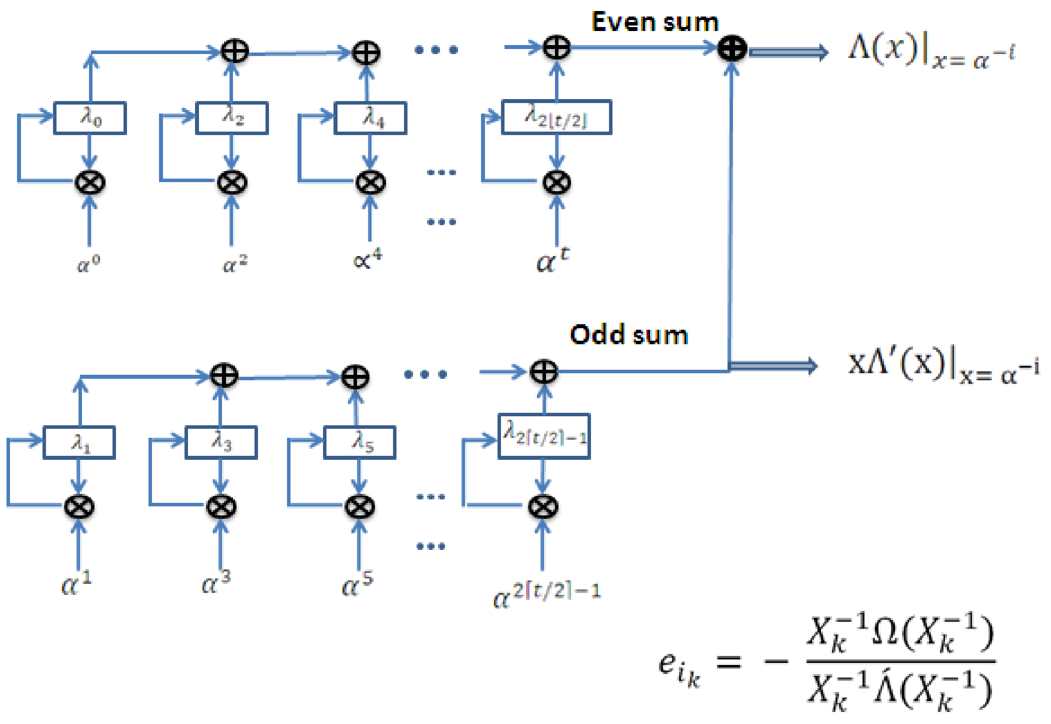


$$e_{i_k} = -\frac{X_k^{-1}\Omega(X_k^{-1})}{X_k^{-1}\acute{\Lambda}(X_k^{-1})}$$

**Figure 6-7: Evaluation of Error-evaluator at reciprocal of Error-location [13]**

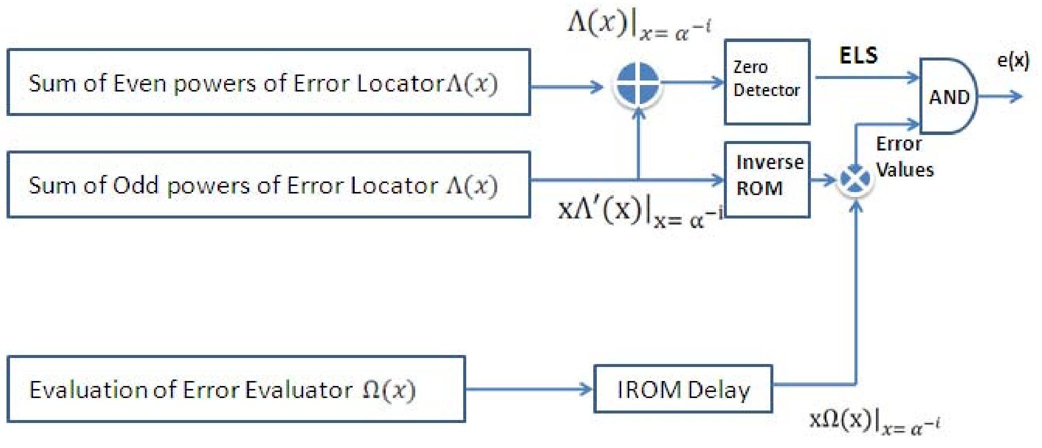
It can be shown that both Chein search and Forney's Formula computation can share certain calculations. For example, the computation of the error locator polynomial's formal derivative is same as that of the odd powered terms of the Chein search if we both multiply and divide Forney's formula with inverse of the error location. Architecture of Figure 6-8 shows this computation.

Overall architecture for both Chein Search and Error-evaluation unit is shown in Figure 6-9. Zero-detector is simply a NOR gate whose output goes high if it detects a zero at the output of the Chein search unit i.e. a root and hence an error-location is found. It output is thus called Error Locator Sequence (ELS) . The the reciprocal of the computed formal derivative of the error-locator polynomial is taken by using an IROM look up table which has inverses of GF elements tabulated. This is then multiplied with the error-evaluator polynomial computed at inverse of the error-locators. This product represents the estimated error-values which is then 'anded' with ELS (working as an enable signal) to get the error polynomial e(x). e(x) may be added to a delayed version of the received word to get an estimate of the transmitted codeword.



Even sum
$\Lambda(x)|_{x=\alpha^{-i}}$
$\lambda_0$
$\lambda_2$
$\lambda_4$
$\lambda_{2\lfloor t/2 \rfloor}$
$\alpha^0$
$\alpha^2$
$\alpha^4$
$\alpha^t$
Odd sum
$x\Lambda'(x)|_{x=\alpha^{-i}}$
$\lambda_1$
$\lambda_3$
$\lambda_5$
$\lambda_{2\lceil t/2 \rceil-1}$
$\alpha^1$
$\alpha^3$
$\alpha^5$
$\alpha^{2\lceil t/2 \rceil-1}$
$e_{i_k} = -\frac{X_k^{-1}\Omega(X_k^{-1})}{X_k^{-1}\acute{\Lambda}(X_k^{-1})}$

**Figure 6-8: Combined Chein-Search and Formal Derivative of Error-locator Polynomial [13]**



$\Lambda(x)|_{x=\alpha^{-i}}$
Sum of Even powers of Error Locator $\Lambda(x)$
Zero Detector
ELS
AND
e(x)
Sum of Odd powers of Error Locator $\Lambda(x)$
Inverse ROM
Error Values
$x\Lambda'(x)|_{x=\alpha^{-i}}$
Evaluation of Error Evaluator $\Omega(x)$
IROM Delay
$x\Omega(x)|_{x=\alpha^{-i}}$

**Figure 6-9: Combined Chein-Search and Error-evaluator Block [13]**

### 6.3.3 Key Equation Solver (KES) Block

The throughput bottleneck in Reed–Solomon decoders is in the **KES** block which solves Eq. ( **6-6**). In contrast, the **SC** and **CSEE** blocks are relatively straightforward to implement. Now we focus on high-speed architectures for the **KES** block. As mentioned earlier, the key equation can be solved via the **eE** algorithm or via the **BM** algorithm. We discuss high-speed architectures for a reformulated version of the **BM** algorithm because this reformulated algorithm can be used to achieve much higher speeds than can be achieved by other implementations of the **BM** and **eE** algorithms. Furthermore, these new architectures also have lower gate complexity and a simpler control structure than architectures based on the **eE** algorithm [6].

### 6.4 Berlekamp-Massey (BM) Architectures

In this section, we give a brief description of different versions of the Berlekamp–Massey (**BM**) algorithm and then discuss a generic architecture for implementation of the algorithm.

#### 6.4.1 The Berlekamp–Massey Algorithm

The **BM** algorithm is an iterative procedure for solving the Key equation. In the form originally proposed by Berlekamp [1,10], the algorithm begins with polynomials $\Lambda(0,z) = 1$, $\Omega(0,\mathrm{z}) = 0$ and iteratively determines polynomials $\Lambda(r,z)$, *and* $\Omega(\mathrm{r},\mathrm{z})$ satisfying the polynomial congruence $\Lambda(r,z)\ S(z) \equiv \Omega(r,z) mod\ z^{2t}$ *For* $r = 1,2,\ldots,2t$ and, thus, obtains a solution $\Lambda(2t,z)$ and $\Omega(2t,z)$ to the key equation. Two "scratch" polynomials $\mathrm{B}(r,z)$ and $\mathrm{H}(r,z)$ with initial values $\mathrm{B}(0,z)= 1$ and $\mathrm{H}(0,z) = -1$ are used in the algorithm. For each successive value of $r$, the algorithm determines $\Lambda(r,z)$, *and* $\mathrm{B}(\mathrm{r},\mathrm{z})$ from $\Lambda(r-1,z)$, *and* $\mathrm{B}(\mathrm{r}-1,\mathrm{z})$. Similarly, the algorithm determines $\Omega(\mathrm{r},\mathrm{z})$ and $\mathrm{H}(r,z)$ from $\Omega(\mathrm{r}-1,\mathrm{z})$ and $\mathrm{H}(r-1,z)$. Since $\mathrm{S}(\mathrm{z})$ has degree *2t-1*, and the other polynomials can have degrees as large as *t*, the algorithm needs to store roughly *6t* field elements. If each iteration is completed in one clock cycle, then *2t* clock cycles are needed to find the error-locator and error-evaluator polynomials.

In recent years, most researchers have used the formulation of the **BM** algorithm given by Blahut in which only $\Lambda(r,z)$, and $\mathrm{B}(\mathrm{r},\mathrm{z})$ are computed iteratively. Following the completion of the *2t* iterations, the error-evaluator polynomial $\Omega(2\mathrm{t},\mathrm{z})$ is computed as the terms of degree *t-1* or less in the polynomial product $\Omega(2\mathrm{t},\mathrm{z})\mathrm{S}(\mathrm{z})$. An implementation of this version thus needs to store only *4t* field elements, but the computation of $\Omega(2\mathrm{t},\mathrm{z})$ requires an additional $t$ clock cycles. Although this version of the **BM** algorithm trades off space against time, it also suffers from the same problem as the Berlekamp version, viz. during some of the iterations, it is necessary to divide each coefficient of $\Lambda(r,z)$, by a quantity $\delta_r$. These divisions are most efficiently handled by first computing $\delta_r{}^{-1}$, the inverse of $\delta_r$, and then multiplying each coefficient of $\Lambda(r,z)$ by ${\delta_r}^{-1}$ . Unfortunately, regardless of whether this method is used or whether one constructs separate divider circuits for each coefficient of $\Lambda(r,z)$ , these divisions, which occur inside an iterative loop, are more time consuming than multiplications. Obviously, if these divisions could be replaced by multiplications, the resulting circuit implementation would have a smaller critical path delay and higher clock speeds would be usable.

The inversion-nless **BM** (**iBM**) algorithm [3,4] is described by the pseudocode shown below. The **iBM** algorithm actually finds scalar multiples $\beta.\Lambda(z) and\ \beta.\Omega(z)$ instead of the $\Lambda(z) and\ \Omega(z)$. However, it is obvious that the Chien search will find the same error locations and it follows from Forney's formula that the same error values are obtained. Hence, we continue to refer to the polynomials computed by the **iBM** algorithm as $\Lambda(z) and\ \Omega(z)$.

**Algorithm 6-1**

***The iBM Algorithm* [6]**

***Initialization***:

```
$\lambda_0(0) = b_0(0) = 1,\ \lambda_i(0) = b_i(0) = 0,\ for\ i = 1,2,\dots,t,\quad k(0) = 0,\quad \gamma(0) = 1$
$Input$:    $s_i, i = 0,1\dots,2t-1.$
$\boldsymbol{for}\ r = 0:1:2t-1\ \boldsymbol{do}$
$\boldsymbol{begin}$
$\boldsymbol{Step\ iBM.1}$  $\boldsymbol{\delta}(r) = s_r\,\lambda_0(r) + s_{r-1}\,\lambda_1(r) + \cdots + s_{r-t}\,\lambda_t(r)$
$\boldsymbol{Step\ iBM.2}$  $\lambda_i(r+1) = \gamma(r)\lambda_i(r) - \delta(r)b_{i-1}(r);\ (i = 0,1,\dots,t)$
$\boldsymbol{Step\ iBM.3}$  $if\ (\delta(r) \neq 0\ and\ k(r) \geq 0)$
      $\boldsymbol{then}$
      $\boldsymbol{begin}$
          $b_i(r+1) = \lambda_i(r),\quad (i = 0,1,\dots,t)$
          $\gamma(r+1) = \delta(r)\quad k(r+1) = -k(r) - 1$
      $\boldsymbol{end}$
      $\boldsymbol{else}$
      $\boldsymbol{begin}$
          $b_i(r+1) = b_{i-1}(r),\ (i = 0,1,\dots,t)$
          $\gamma(r+1) = \gamma(r)\quad k(r+1) = k(r) + 1$
      $\boldsymbol{end}$
```

```
end
for $i = 0 : 1 : t-1$ do
Step iBM.4  $\omega_i(2t) = s_i\,\lambda_0(2t) + s_{i-1}\,\lambda_1(2t) + \ldots + s_0\,\lambda_i(2t)$
Output: $\lambda_i(2t)$; $i = 0, 1, \ldots, t.$  $\omega_i(2t), i = 0, 1, \ldots, t-1$
```

For $r < t$, Step **iBM.1** includes terms $s_{-1}.\lambda_{r+1}(r), s_{-2}.\lambda_{r+2}(r), \ldots, s_{r-t}.\lambda_t(r)$ involving unknown quantities. Fortunately, it is known that $\deg(\Lambda\,(r,z)) \leq r$, so that $\lambda_{r+1}(r) = \lambda_{r+2}(r) = \cdots = \lambda_t(r)=0$ and therefore the unknown $s_i$ do not affect the value of $\delta(r)$ .Notice also the similarity between Steps **iBM.1** and **iBM.4**.These facts simplify the architecture that we describe next.

## *6.4.2* Architectures Based on the iBM Algorithm

Due to the similarity of Steps **iBM.1** and **iBM.4**, architectures based on the **iBM** algorithm need only two major computational structures as shown in Figure 6-10.

1. The *discrepancy computation* (**DC**) block for implementing Step **iBM.1**.
2. The *error locator update* (**ELU**) block which implements Steps **iBM.2** and **iBM.3** in parallel.

The **DC** block contains latches for storing the syndromes $s_i$, the GF $(2^m)$ arithmetic units for computing the discrepancy $\delta(r)$ and the control unit for the entire architecture. It is connected to the **ELU** block, which contains latches for storing for Λ*(r,z)*and B*(r,z)* as well as arithmetic units for updating these polynomials, as shown in Figure 6-10. During a clock cycle, the **DC** block computes the discrepancy $\delta(r)$ and passes this value together with $\gamma(r)$ and a control signal $MC(r)$ to the **ELU** block which updates the polynomials during the same clock cycle.

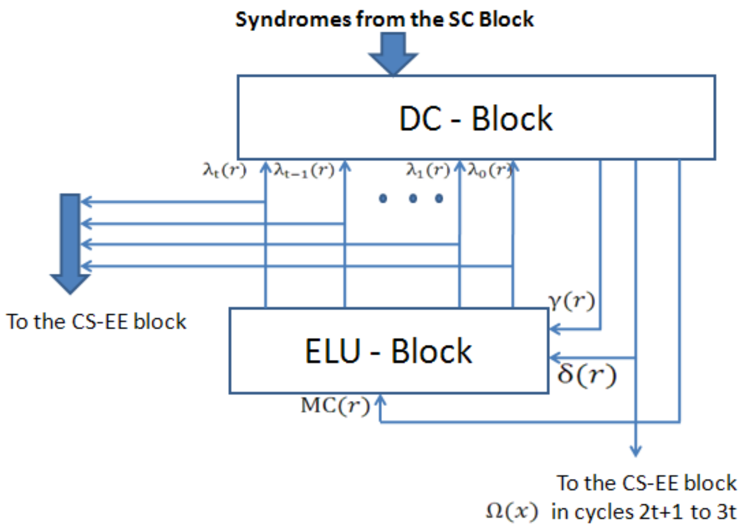


**Figure 6-10: The iBM Architecture [6]**

### 6.4.2.1 DC Block Architecture:

The **DC** block architecture shown in Figure 6-11 has $2t$ latches constituting the DS shift register that are initialized such that the latches $DS_1, DS_2, \dots \dots, DS_{2t-1}, DS_0$ contain the syndromes $s_1, s_2, \dots \dots, s_{2t-1}, s_0,$ respectively. In each of the first $2t$ clock cycles, the $t+1$ multipliers compute the products in Step **iBM.1**. These are added in a binary adder tree of depth $\lceil \log_2 t+1) \rceil$ to produce the discrepancy $\delta(r)$. Thus, the delay in computing $\delta(r)$ is $T_\delta = T_{mult} + \lceil \log_2 t + 1) \rceil . T_{add}$.

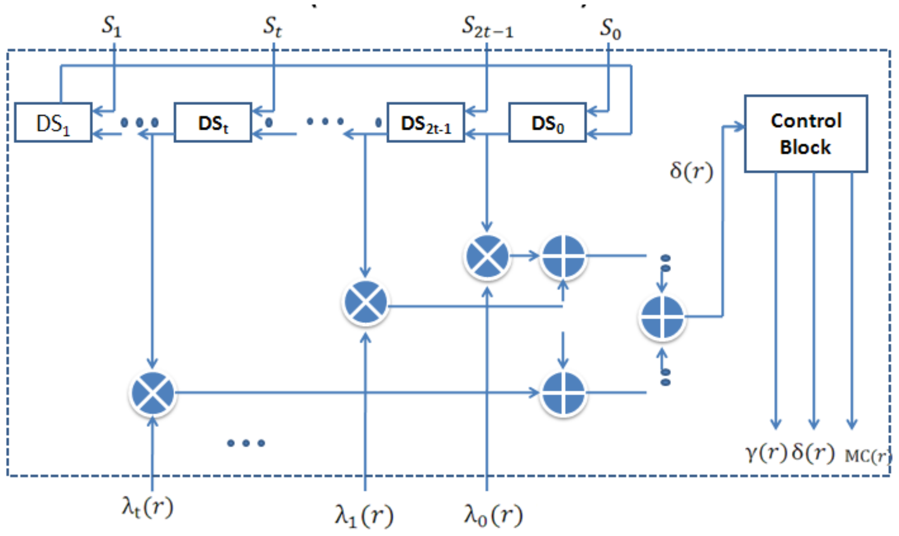
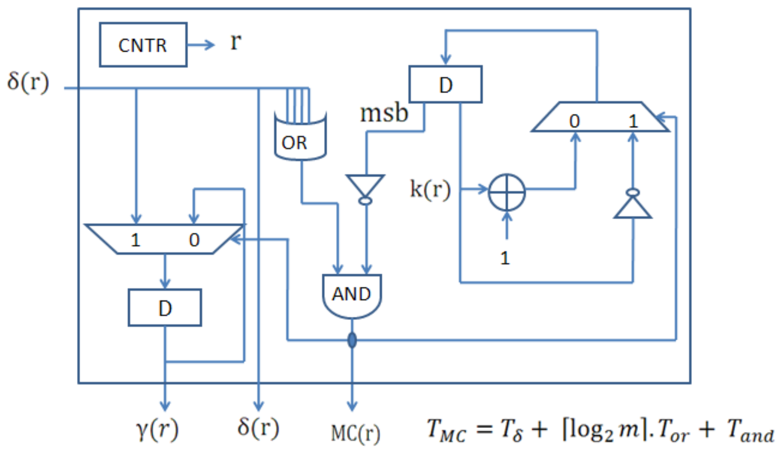


**Figure 6-11: The Discrepancy Computation Block [6]**



CNTR
r
δ(r)
D
msb
0
1
OR
k(r)
1
0
1
D
AND
γ(r)
δ(r)
MC(r)
$T_{MC} = T_{\delta} + \lceil \log_2 m \rceil . T_{or} + T_{and}$

**Figure 6-12: Control Block [6]**

A typical control unit such as the one illustrated in Figure 6-12 has counters for the variables $r$ and $k(r)$, and storage for $\gamma(r)$. Following the computation of $\delta(r)$, the control unit computes the OR of the bits in order to determine whether $\delta(r)$ is nonzero. This requires $m-1$ two-input OR gates arranged in a binary tree of depth $\lceil \log_2(m) \rceil$. If the counter for $k(r)$ is implemented in two's-complement representation, then $k(r) \geq 0$ if and only if the most significant bit in the counter is 0. The delay in generating $MC(r)$ signal is thus $T_{MC} = T_{\delta}$ $+\lceil \log_2 m) \rceil . T_{or}$ $+T_{and}$. Finally, once the signal $MC(r)$ is available, the counter for $k(r)$ can be updated. Notice that a twos-complement arithmetic addition is needed if $k(r+1) = \ k(r) + 1$ .On the other hand, negation in two's-complement representation complements all the bits and then adds one and, hence, the update $k(r+1) = -k(r) + 1$ requires only the complementation of all the bits in the counter $k(r)$. We note that it is possible to use *ring counters* for $r$ and $k(r)$, in which case $k(r)$ is updated just $T_{mux}$ seconds after the signal $MC(r)$ has been computed.

Following the $2t$ clock cycles for the **BM** algorithm, the **DC** block computes the error-locator polynomial $\Omega(z)$ in the next $t$ clock cycles. To achieve this, the $DS_t, DS_{t+1}$, … …, $DS_{2t-1}$ latches are reset to zero during the 2t-th clock cycle, so that, at the beginning of the (2t+1)-th clock cycle, the contents of the DS register (see Figure 6-11) are $s_1$, $s_2$, … …, $s_{t-1}$, 0,0, … …, $s_0$. Also, the outputs of the **ELU** block are frozen so that these do not change during the computation of $\Omega(z)$. From Step **iBM.4**, it follows that the "discrepancies" computed during the next $t$ clock cycles are just the coefficients $\omega_0(2t), \omega_1(2t)$ , … …, $\omega_{t-1}(2t)$ of $\Omega(z)$.

Note that the total hardware requirements of the **DC** block are $2t$ m-bit latches, $t+1$ multipliers, $t$ adders, and miscellaneous other circuitry (counters, arithmetic adder or ring counter, OR gates, inverters and latches), in the control unit. The critical path delay of the **DC** block is

$$T_{DC} = T_{mult} + (1 + \lceil \log_2 t + 1) \rceil ).T_{add} + \lceil \log_2 m) \rceil .T_{or} + T_{and}.$$

### 6.4.2.2 ELU Block Architecture:

Following the computation of the discrepancy $\delta(r)$ and the signal $MC(r)$ in the **DC** block, the polynomial coefficient updates of Steps **iBM.2** and **iBM.3** are performed simultaneously in the **ELU** block. The processor element **PE0** (hereinafter the **PE0** *processor*) that updates one coefficient of $\lambda(z)$ and $\mathrm{B}(z)$ is illustrated in

Figure 6-13.

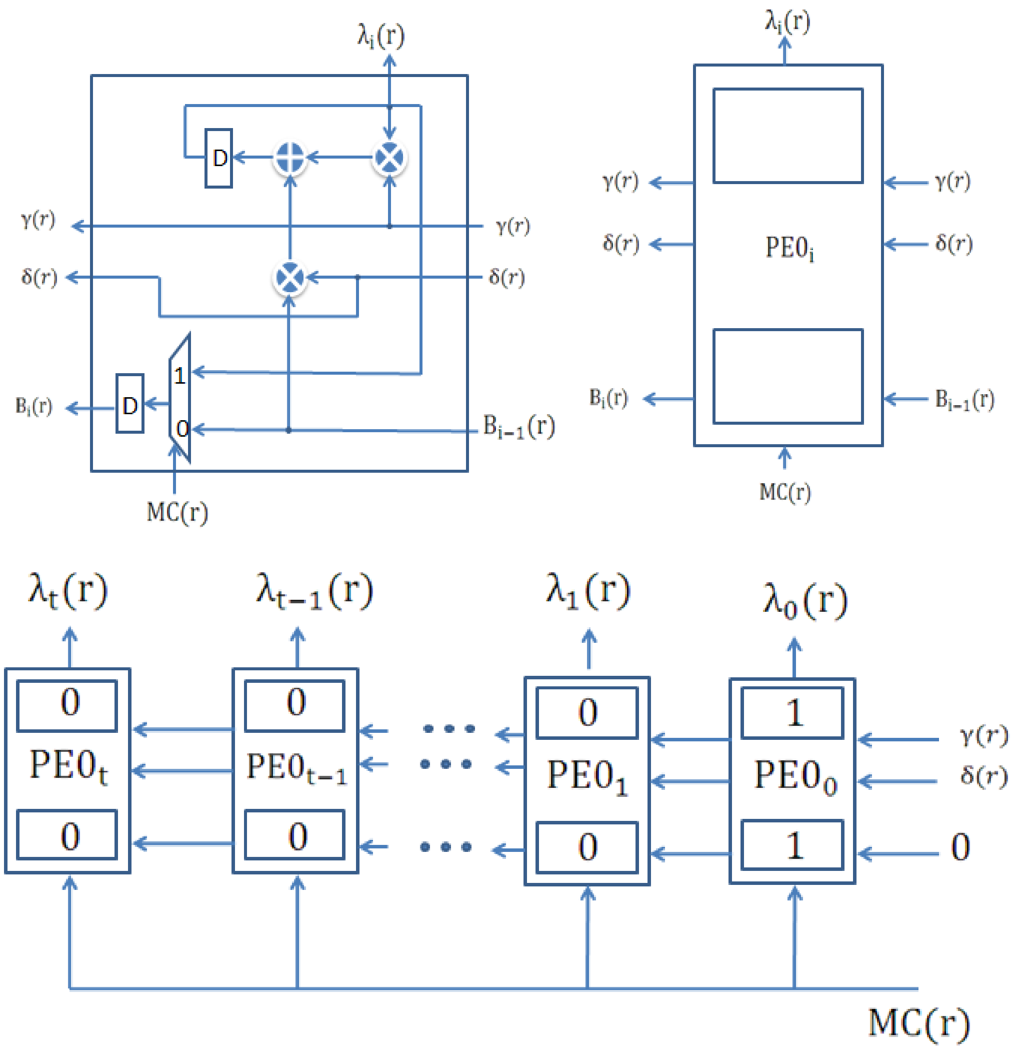


Figure 6-13: The PE0 processor and ELU Block Diagram [6]

The complete **ELU** architecture is also shown in

Figure 6-13 where we see that signals $\delta(r)$, $\gamma(r)$ and $MC(r)$ are broadcast to all the **PE0** processors. In addition, the latches in all the **PE0** processors are initialized to zero except for $\mathbf{PE0_0}$, which has its latches initialized to the element $1 \epsilon \text{ GF } (2^m)$ . Notice that $2t + 1$ latches and multipliers, and $t + 1$ adders and multiplexers are needed. The critical path delay of the **ELU** block is given by

$$T_{\text{ELU}} = T_{mult} + T_{add}$$

### 6.4.2.3 iBM Architecture

Ignoring the hardware used in the control section, the total hardware needed to implement the **iBM** algorithm is $4t + 2$ latches, $3t + 3$ multipliers, $2t + 1$ adders, and $t + 1$ multiplexers. The total time required to solve the key equation for one codeword is $3t$ clock cycles. Alternatively, if $\Omega(2t, z)$ is computed iteratively, the computations require only $2t$ clock cycles. However, since the computations required to update $\Omega(r, z)$ are the same as that of $\Lambda(r, z)$ , a near-duplicate of the **ELU** block is needed. This increases the hardware requirements to $6t + 2$ latches, $5t + 3$ multipliers, $3t + 1$ adders, and $2t + 1$ multiplexers. In either case, the critical path delay of the **iBM** architecture can be obtained as

$$T_{\text{IBM}} = 2. T_{mult} + (1 + \lceil \log_2 t + 1)\rceil ). T_{add} > 2. ( T_{mult} + \ T_{add})$$

which is the delay of the direct path that begins in the **DC** block starting from the $\boldsymbol{DS_i}$ latches, through a multiplier, an adder tree of height $\lceil \log_2 t + 1)\rceil$ (generating the signal $\delta(r)$ ), feeding into the **ELU** block multiplier and adder before being latched. We have assumed that the indirect path taken by through the control unit (generating signal $MC(r)$) feeding into the **ELU** block multiplexer is faster than the direct path, i.e.,

$$T_{mult} > \ \lceil \log_2 m)\rceil T_{or} + \ T_{and}$$

This is a reasonable assumption in most technologies. Note that more than half of $T_{\text{IBM}}$ is due to the delay in the **DC** block, and that this contribution increases logarithmically with the error correction capability. Thus, reducing the delay in the **DC** block is the key to achieving higher speeds. In the next section, we describe algorithmic reformulations of the

**iBM** algorithm that lead to a systolic architecture for the **DC** block and reduce its critical path delay to $T_{ELU}$.

## 6.5 Reformulated Reed-Solomon Decoder Architectures

The critical path in **iBM** architecture passes through *two* multipliers as well as the adder tree structure in the **DC** block. The multiplier units contribute significantly to the critical path delay and hence reduce the throughput achievable with the **iBM** architecture. In this section, we discuss decoder architectures that have a smaller critical path delay. These architectures are derived via algorithmic reformulation of the **iBM** algorithm. This reformulated **iBM** (**riBM**) algorithm computes the *next* discrepancy $\delta(r+1)$ at the same time that it is computing the *current* polynomial coefficient updates, that is, the $\lambda_i(r+1)$ 's and the $b_i(r+1)$'s. This is possible because the reformulated discrepancy computation does not use the $\lambda_i(r+1)$'s explicitly. Furthermore, the discrepancy is computed in a block which has the *same* structure as the **ELU** block, so that both blocks have the same critical path delay $T_{mult} + T_{add}$.

### 6.5.1 Reformulation of the iBM Algorithm

#### 6.5.1.1 Simultaneous Computation of Discrepancies and Updates

Viewing Steps **iBM.2** and **iBM.3** in terms of polynomials, we see that Step **iBM.2** computes

$$\boldsymbol{\Lambda}(\boldsymbol{r}+\mathbf{1},\boldsymbol{z}) = \boldsymbol{\gamma}(\boldsymbol{r}).\boldsymbol{\Lambda}(\boldsymbol{r},\boldsymbol{z}) - \boldsymbol{z}.\boldsymbol{\delta}(\boldsymbol{r}).\mathbf{B}(\boldsymbol{r},\boldsymbol{z}) \quad \text{Eq. ( 6-8)}$$

while Step **iBM.3** sets $\mathrm{B}(r+1,z)$ either to $\Lambda(r,z)$ or to $z.B(r,z)$ . Next, note that the discrepancy $\delta(r)$ computed in Step **iBM.1** is actually $\delta_r(r)$ , the coefficient of $z^r$ in the polynomial product

$$\Lambda(r,z).S(z) = \Delta(r,z) = \delta_0(r) + \delta_1(r).z + \cdots + \delta_r(r).z^r + \cdots \quad \text{Eq. ( 6-9)}$$

Much faster implementations are possible if the decoder computes *all* the coefficients of $\Delta(r,z)$ (and of $\Theta(r,z) = B(r,z).S(z)$) even though only $\delta_r(r)$ is needed to compute $\Lambda(r+1,z)$ and to decide whether $B(r+1,z)$ is to be set to $\Lambda(r,z)$ or to $z.B(r,z)$.

Suppose that at the beginning of a clock cycle, the decoder has available to it all the coefficients of $\Delta(r,z)$ and $\Theta(r,z)$ (and, of course, of $\Lambda(r,z)$ and $B(r,z)$ as well). Thus, $\delta(r) = \delta_r(r)$is available at the beginning of the clock cycle, and the decoder can compute

$\Lambda(r+1,z)$ and $B(r+1,z)$ . Furthermore, it follows from $\boldsymbol{\Lambda r{+}1,z} = \boldsymbol{\gamma(r).\Lambda(r,z)} - \boldsymbol{z.\delta(r).\mathrm{B}(r,z)}$ Eq. ( **6-8**) and Eq. ( 6-9) that

$$\Delta(r+1,z) = \Lambda(r+1,z).S(z) = [\gamma(r).\Lambda(r,z) - z.\delta_r(r).B(r,z)].S(z)$$
$$= \gamma(r).\Delta(r,z) - z.\delta_r(r).\Theta(r,z)$$

while $\Theta(r+1,z) = \mathrm{B}(r+1,z).S(z)$ is set to either $\Delta(r,z) = \Lambda(r,z).S(z)$ or to $z.\Theta(r,z) = z.B(r,z).S(z)$. In short, $\Delta(r+1,z)$and $\Theta(r+1,z)$ are computed in *exactly* the same manner as are $\Lambda(r+1,z)$and $\mathrm{B}(r+1,z)$ . Furthermore, all four polynomial updates can be computed simultaneously, and all the polynomial coefficients as well as $\delta_{r+1}(r+1)$are thus available at the beginning of the *next* clock cycle.

#### 6.5.1.2 A New Error-Evaluator Polynomial

The **riBM** algorithm simultaneously updates four polynomials $\Lambda(r,z), \mathrm{B}(r,z), \Delta(r,z)$, and $\Theta(r,z)$ with initial values $\Lambda(0,z) = \mathrm{B}(0,z) = 1$ and $\Delta(0,z) = \Theta(0,z) = S(z)$ . The 2t iterations thus produce the error-locator polynomial $\Lambda(2t,z)$ and also the polynomial $\Delta(2t,z)$. Note that since $\Omega(2t,z) \equiv \Lambda(2t,z).S(z) mod\, z^{2t}$ it follows from Eq. ( 6-9) that the low-order coefficients of $\Delta(2t,z)$ are just $\Omega(2t,z)$ , that is, the 2t iterations compute *both* the error-locator polynomial $\Lambda(2t,z)$ and the error-evaluator polynomial $\Omega(2t,z)$ — the additional $t$ iterations of Step **iBM.4** are not needed. The high-order coefficients of $\Delta(2t,z)$ can also be used for error evaluation. Let $\Delta(2t,z) = \Omega(2t,z) + z^{2t}.\Omega^{(\mathrm{h})}(z)$ , where $\Omega^{(\mathrm{h})}(z)$ of degree at most $\mathrm{e}-1$ contains the high-order terms. Since $\mathrm{X}_\mathrm{i}^{-1}$ is a root of $\Lambda(2t,z)$, it follows from Eq. ( 6-9) that $\Delta(2t,\mathrm{X}_\mathrm{i}^{-1}) = \Omega(2t,\mathrm{X}_\mathrm{i}^{-1}) + \mathrm{X}_\mathrm{i}^{-2t}\Omega^{(\mathrm{h})}(\mathrm{X}_\mathrm{i}^{-1}) = 0$. Thus, Forney's error evaluation formula can be rewritten as

$$Y_i = -\frac{X_i^{-(m_0+2t-1)}\Omega^{(\mathrm{h})}\left(X_i^{-1}\right)}{\Lambda'\left(X_i^{-1}\right)} = -\frac{z^{m_0+2t}\Omega^{(\mathrm{h})}(z)}{\mathrm{z}\Lambda'(z)}\Bigg|_{z=X_i^{-1}}$$

Eq. ( 6-10)

This variation of the error evaluation formula has certain architectural advantages. Note that the choice $m_0 = -2t = n - 2t$ is preferable if **Eq. ( 6-10**) is to be used.

### 6.5.1.3 Further Reformulation

Since the updating of all four polynomials is identical, the discrepancies can be calculated using an **ELU** block. Unfortunately, for $r = 0,1, \dots\dots, 2t-1,$ , the discrepancy $\delta_r(r)$ is computed in processor **PE0$_\mathbf{r}$** . Thus, multiplexers are needed to route the appropriate latch contents to the control unit and to the **ELU** block that computes $\Lambda(r+1, z)$and $\mathrm{B}(r+1, z)$ . Additional reformulation of the **iBM** algorithm, as described next, eliminates these multiplexers [6]. We use the fact that for any $i < r, \delta_i(r)$ and $\theta_i(r)$ cannot affect the value of any later discrepancy $\delta_{r+j}(r+j)$. Consequently, we need not store $\delta_i(r)$ and $\theta_i(r)$ for $i < r$. Thus, for $= 0,1, \dots\dots, 2t-1$ , define $\hat{\delta}_i(r) = \delta_{i+r}(r)$ and $\hat{\theta}_i(r) = \theta_{i+r}(r)$ and the polynomials

$$\widehat{\Delta}(\mathrm{r}, \mathrm{z}) = \sum_{\mathrm{i}=0}^{2\mathrm{t}-1} \hat{\delta}_i(r) z^r$$

And

$$\widehat{\Theta}(\mathrm{r}, \mathrm{z}) = \sum_{\mathrm{i}=0}^{2\mathrm{t}-1} \hat{\theta}_i(r) z^i$$

with initial values $\widehat{\Delta}(0, \mathrm{z}) = \widehat{\Theta}(0, \mathrm{z}) =$ S(z). It follows that these polynomial coefficients are updated as $\hat{\delta}_i(r+1) = \delta_{i+1+r}(r+1) = \gamma(r).\delta_{i+1+r}(r) - \delta_r(r)\theta_{i+r}(r) = \gamma(r).\hat{\delta}_{i+1}(r) - \hat{\delta}_0(r)\hat{\theta}_i(r)$ while $\widehat{\theta_\imath}(\mathrm{r}+1) = \theta_{\mathrm{i}+1+\mathrm{r}}(\mathrm{r}+1)$ is set either to $\delta_{i+1+r}(r) = \hat{\delta}_{i+1}(r)$ or to $\theta_{i+1}(r) = \hat{\theta}_i(r)$ . Note that the discrepancy $\delta_r$ (r) = $\hat{\delta}_0$ (r) is always in a fixed (zero-th) position with this form of update. As a final comment, note this form of update ultimately produces

$$\widehat{\Delta}(2\mathrm{t}, \mathrm{z}) = \delta_{2t}(2t) + \delta_{2t+1}(2t)z + \cdots = \Omega^{(\mathrm{h})}(2t, z)$$

and, thus, Eq. ( 6-10) can be used for error evaluation in the **CSEE** block. The **riBM** algorithm is described by the following pseudo code.

**Algorithm 6-2**

***The riBM Algorithm*** **[6]**

***Initialization***:

$\lambda_0(0) = b_0(0) = 1, \ \lambda_i(0) = b_i(0) = 0, \ for \ i = 1,2, \dots, t, \qquad k(0) = 0, \qquad \gamma(0) = 1$

***Input***: $\quad s_i, \quad i = 0, 1 \dots, 2t-1.$

$\hat{\delta}_i(0) = \hat{\theta}_i(0) = \ s_i \ (i = 0; \dots; 2t-1)$

```
$\boldsymbol{for}\ r = 0\ \boldsymbol{step}\ 1\ \boldsymbol{until}\ 2t\ \ 1\ \boldsymbol{do}$
$\boldsymbol{begin}$
$\boldsymbol{Step\ riBM.1}$    $\lambda_i(r+1) = \gamma(r).\lambda_i(r) - \hat{\delta}_0(r)b_{i-1}(r);\ (i = 0,\dots,2t-1)$
                  $\hat{\delta}_i(r+1) = \gamma(r).\hat{\delta}_{i+1}(r) - \hat{\delta}_0(r).\hat{\theta}_i(r);\ (i = 0,\dots,2t-1)$
$\boldsymbol{Step\ riBM.2}$    $\boldsymbol{if}\ (\ \hat{\delta}_0(r) \neq 0\ \boldsymbol{and}\ k(r) \geq 0\ )$
$\boldsymbol{then}$
$\boldsymbol{begin}$
        $b_i(r+1) = \lambda_i(r);\ (i = 0,1,\dots,t)$
        $\hat{\theta}_i(r+1) = \hat{\delta}_{i+1}(r);\ (i = 0,1,\dots,2t-1)$
        $\gamma(r+1) = \hat{\delta}_0(r)$
        $k(r+1) = -k(r) - 1$
$\boldsymbol{end}$
$\boldsymbol{else}$
$\boldsymbol{begin}$
        $b_i(r+1) = b_{i-1}(r);\ (i = 0,1,\dots,t)$
        $\hat{\theta}_i(r+1) = \hat{\theta}_i(r);\ (i = 0,1,\dots,2t-1)$
        $\gamma(r+1) = \gamma(r)$
        $k(r+1) = k(r) + 1$
$\boldsymbol{end}$
$\boldsymbol{end}$
$\boldsymbol{Output}$: $\lambda_i(2t);\ (i = 0,1,\dots,t);\ \omega^{(h)}(2t) = \hat{\delta}_i(2t);\ (i = 0,1,\dots,t-1)$
```

Next, we consider architectures that implement the **riBM** algorithm.

## 6.6 High-Speed Reed–Solomon Decoder Architectures

As in the **iBM** architecture described in Section **Error! Reference source not found.**, the **riBM** architecture consists of a reformulated discrepancy computation (**rDC**) block connected to an **ELU** block.

### 6.6.1 The rDC Architecture

The **rDC** block uses processor **PE1** shown in Figure 6-14 and the **rDC** architecture shown in Figure 6-15. Notice that processor **PE1** is very similar to processor **PE0** of
Figure 6-13.

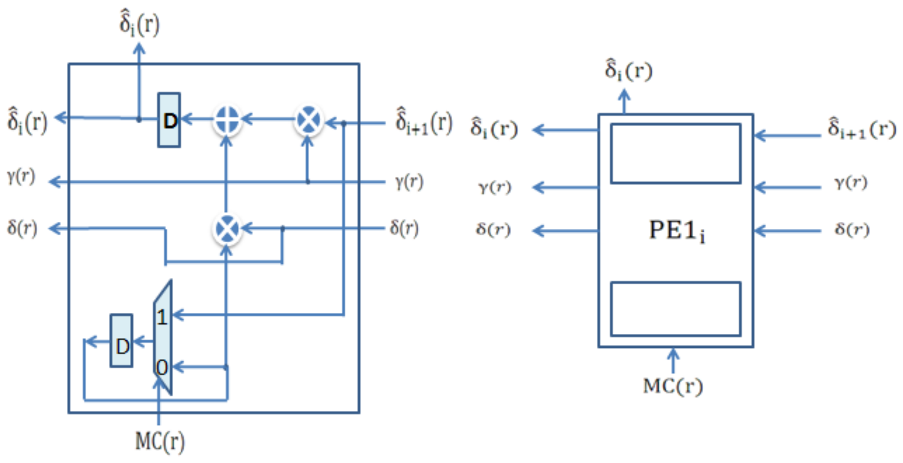


**Figure 6-14: Processor Element 1 (PE1) [6]**

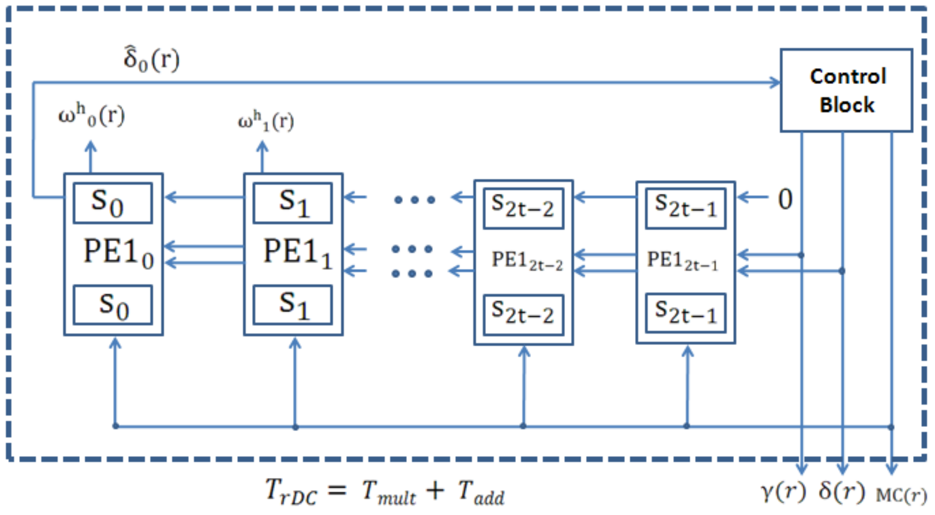


**Figure 6-15: The reformulated Discrepancy Computation (rDC) Architecture [6]**

Obviously, the hardware complexity and the critical path delays of processors **PE0** and **PE1** are identical, we get that $T_{rDC} = T_{mult} + T_{add}$. Note that the delay is independent of the error-correction capability $t$ of the code. The hardware requirements of the architecture in Figure 6-15 are $2t$ **PE1** processors, that is, $4t$ latches, $4t$ multipliers, $2t$ adders, and $2t$ multiplexers, in addition to the control unit, which is the same as that in iBM.

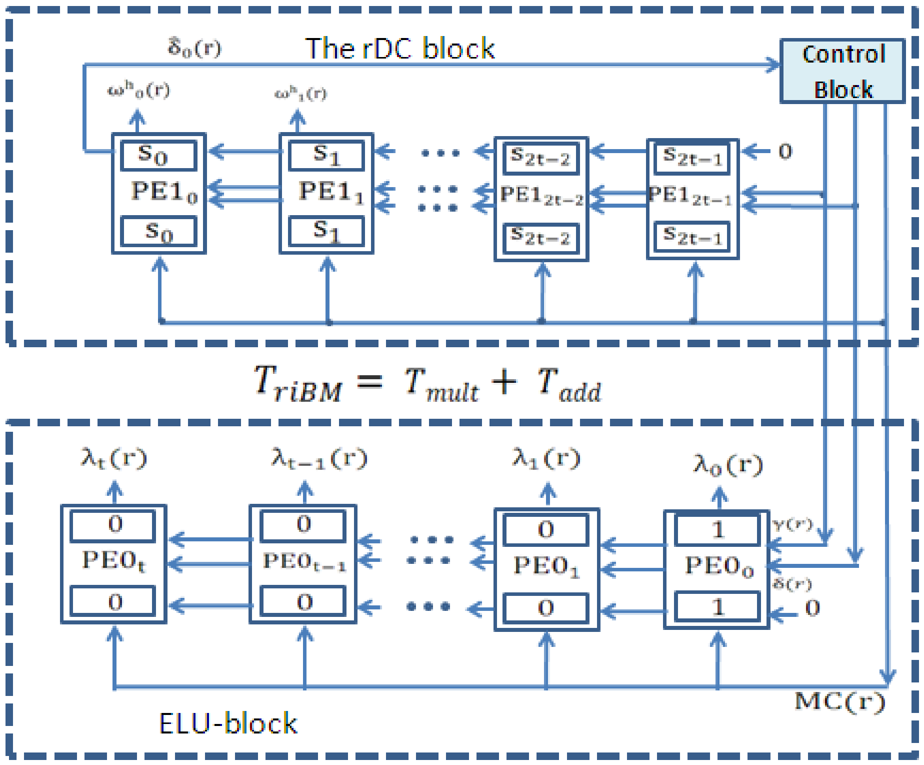


**Figure 6-16: The Systolic riBM Architecture [6]**

### 6.6.2 The riBM Architecture

The overall **riBM** architecture is shown in Figure 6-16 . It uses the **rDC** block of Figure 6-15 and the **ELU** block in

Figure 6-13. Note that the outputs of the **ELU** block do not feed back into the **rDC** block. Both blocks have the same critical path delay of $T_{rDC} = T_{ELU} = T_{mult} + T_{add}$ and since they operate in parallel, **riBM** architecture achieves the same critical path delay:

$$T_{riBM} = T_{mult} + T_{add}$$

which is less than half the delay $T_{iBM}$ of the enhanced **iBM** architecture [6].

At the end of the $2t$ -th iteration,the $\boldsymbol{PE1}$ s, contain the coefficients of $\Omega^{(h)}(2t, z)$ which can be used for error evaluation. Thus, $2t$ clock cycles are used to determine both $\Lambda(z)$ and $\Omega^{(h)}(z)$. Ignoring the control unit, the hardware requirement of this architecture is $3t + 1$ processors, that is, $6t + 2$ latches, $6t + 2$ multipliers, $3t + 1$ adders, and $3t + 1$ multiplexers. This compares very favorably with the $6t + 2$ latches, $5t + 3$ multipliers, $3t + 1$ adders, and $2t + 1$ multiplexers needed to implement the enhanced **iBM** architecture in which both the error-locator and the error-evaluator polynomial are computed in $2t$ clock cycles. Using only $t-1$ additional multipliers and $t$ additional multiplexers, we have reduced the critical path delay by more than 50%. Furthermore, the **riBM** architecture consists of two systolic arrays and is thus very regular [6].

### 6.6.3 The RiBM Architecture

It is possible to eliminate the **ELU** block entirely, and to implement the **BM** algorithm in an enhanced **rDC** block in which the array of $2t$ **PE1** processors are lengthened into an array of $\mathbf{3t+1}$ **PE1** processors as shown in Figure 6-17.

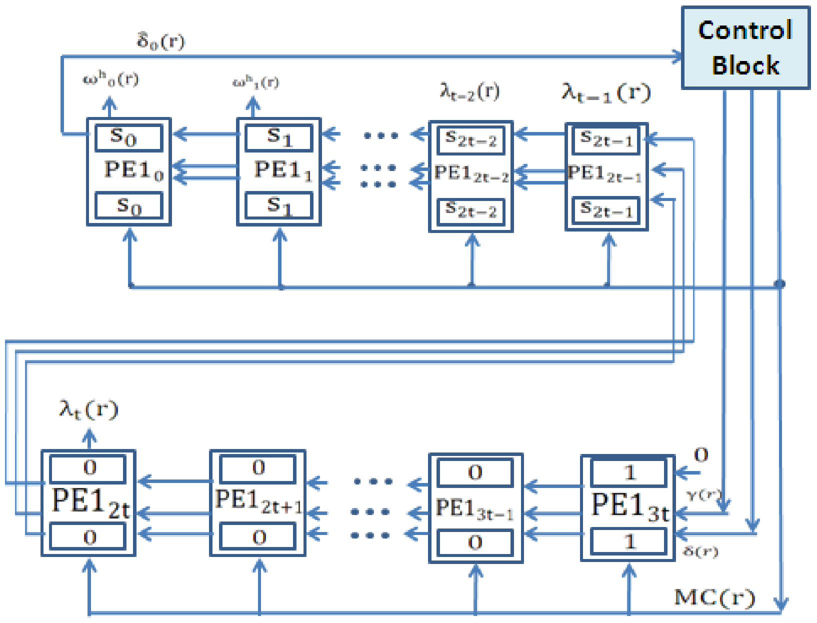


**Figure 6-17: The homogeneous Systolic RiBM architecture [6]**

In this completely systolic architecture, a *single* array computes *both* $\Lambda(z)$ and $\Omega^{(h)}(z)$. Since the $t+1$ **PE0** processors eliminated from the **ELU** block re-appear as the $t+1$ additional **PE1** processors, the **RiBM** architecture has the same critical path delay as the **riBM** architecture. However, its extremely regular structure offers some advantage in VLSI circuit layouts.

An array of **PE0** processors in the **riBM** architecture (see Figure 6-16) carries out the same p*olynomial* computation as an array of **PE1** processors in the **RiBM** architecture (see Figure 6-17), but in the latter array, the polynomial coefficients shift left with each clock pulse. Thus, in the **RiBM** architecture, suppose that the initial loading of $\boldsymbol{PE1_0}$, $\boldsymbol{PE1_1}$, …,$\boldsymbol{PE1_{2t-1}}$, is as in Figure 6-15, while $\boldsymbol{PE1_{2t}}$, $\boldsymbol{PE1_{2t+1}}$, …,$\boldsymbol{PE1_{3t-1}}$,are loaded with zeros, and the latches in $\boldsymbol{PE1_{3t}}$ are loaded with $1 \in GF(2^m)$. Then, as the iterations proceed, the polynomials $\widehat{\Delta}(r,z)$and $\widehat{\Theta}(r,z)$ are updated in the processors in the left-hand end of the array (effectively $\Delta(r,z)$, and $\Theta(r,z)$, get updated and shifted leftwards).

After $2t$ clock cycles, the coefficients of $\Omega^{(h)}(z)$are in processors $\boldsymbol{PE1_0}$–$\boldsymbol{PE1_{t-1}}$. Next, note that $\boldsymbol{PE1_{3t}}$ contains $\Lambda(0,z)$ and $B(0,z)$, and as the iterations proceed, $\Lambda(r,z)$and $\mathrm{B}(r,z)$shift leftwards through the processors in the right-hand end of the array, with $\lambda_i(r)$ and $b_i(r)$ being stored in processor $\boldsymbol{PE1_{3t-r+1}}$. After $2t$ clock cycles, processor $\boldsymbol{PE1_{t+i}}$ contains $\lambda_i(2t)$ and $b_i(2t)$ for $i = 0,1,\dots,t$. Thus, the same array is carrying out two separate computations. These computations donot interfere with one another. On the other hand, since $\deg(\Delta(r,z)) = \deg(S(z)) + \deg(\Lambda(r,z))$, it follows that $\deg\left(\widehat{\Delta}(r,z)\right) \leq 2t - 1 + r + l(r)$ where $l(r) = \frac{r-k(r)}{2}$ is known to be an upper bound on $\deg(\Lambda(r,z))$. It is known that $l(r)$ is a non-decreasing function of $r$ and that it has maximum value $l(2t) = e$ if $e \leq t$ errors have occurred. Hence, $2t - 1 + r + l(r) < 3t - r$ for all $r$, and thus, as $\Lambda(r,z)$ and $B(r,z)$ shift leftwards, they do not over-write the coefficients of $\widehat{\Delta}(r,z)$ and $\widehat{\Theta}(r,z)$. We denote the contents of the array in the **RiBM** architecture as polynomials $\widehat{\Delta}(r,z)$ and $\widehat{\Theta}(r,z)$ with initial values $\widehat{\Delta}(0,z)$ and $\widehat{\Theta}(0,z) = S(z) + z^{3t}$. The **RiBM** architecture [6] implements the following pseudo code.

**Algorithm 6-3**

```
$\boldsymbol{The\ RiBM\ Algorithm}$ [6]
$\boldsymbol{Initialization}$:
$\hat{\delta}_{3t}(0) = 1.\ \hat{\delta}_i(0) = 0\ for\ i = 2t,\dots,3t-1.\quad k(0) = 0,\quad \gamma(0) = 1$
$\boldsymbol{Input}: s_i, i = 0,1,\dots,2t-1.$
$\hat{\delta}_i(0) = \hat{\theta}_i(0) = s_i;\ (i = 0,\dots,2t\ 1)$
$\boldsymbol{for}\ r = 0:\mathbf{1}:2t-1\ \boldsymbol{do}$
$\boldsymbol{begin}$
$\boldsymbol{Step\ RiBM.1}\ \hat{\delta}_i(r+1) = \gamma(r).\hat{\delta}_{i+1}(r) - \hat{\delta}_0(r).\hat{\theta}_i(r)\qquad i = (0,\dots,3t)$
$\boldsymbol{Step\ RiBM.2\ if}\ (\hat{\delta}_0(r) \neq 0\ \boldsymbol{and}\ k(r) \geq 0$
$\boldsymbol{then}$
$\boldsymbol{begin}$
$\hat{\theta}_i(r+1) = \hat{\delta}_{i+1}(r)\qquad (i = 0;\ 1;\ \dots;\ 3t)$
```

```
$\gamma(r + 1) = \hat{\delta}_0(r)$
$k(r + 1) = -k(r) - 1$
$\boldsymbol{end}$
$\boldsymbol{else}$
$\boldsymbol{begin}$
$\hat{\theta}_i(r+1) = \hat{\theta}_i(r), \quad (i = 0; 1; \ldots; 3t)$
$\gamma(r + 1) = \gamma(r)$
$k(r + 1) = k(r) + 1$
$\boldsymbol{end}$
$\boldsymbol{end}$
```

## *6.7* Comparison of Architectures

Table 6-2 summarizes the complexity of the various architectures described so far. It can be seen that, in comparison to the conventional **iBM** architecture (Berlekamp's version), the reformulated **riBM** and **RiBM** systolic architectures require more $t - 1$ multipliers and $t$ more multiplexers. All three architectures require the same numbers of latches and adders and all three architectures require $2t$ cycles to solve the key equation for a $t$ -error-correcting code. The **riBM** and **RiBM** architectures require considerably more gates than the conventional **iBM** architecture (Blahut's version), but also require only $2t$ clock cycles as compared to the $3t$ clock cycles required by the latter. Furthermore, since the critical path delay in the **riBM** and **RiBM** architectures is less than half the critical path delay in either of the **iBM** architectures, the reformulated architectures significantly reduce the total time required to solve the key equation (and thus achieve higher throughput) with only a modest increase in gate count. More important, the regularity and scalability of the **riBM** and **RiBM** architectures creates the potential for automatically generating regular layouts (via a core generator) with predictable delays for various values of $t$ and $m$ . Nonetheless, a rough comparison is that the **riBM** and **RiBM** architectures require three times as many gates as the hypersystolic **eE** architecture, but solve the key equation in one-sixth the time.

| Architectures | Adders | Multipliers | Latches | Muxes | Clocks Cycles | Critical Path Delay |
|---|---|---|---|---|---|---|
| iBM (Blahut) | $2t+1$ | $3t+3$ | $4t+2$ | $t+1$ | $3t$ | $> 2.(T_{mult}+T_{add})$ |
| iBM (Berlekamp) | $3t+1$ | $5t+3$ | $6t+2$ | $2t+1$ | $3t$ | $> 2.(T_{mult}+T_{add})$ |
| riBM | $3t+1$ | $6t+2$ | $6t+2$ | $3t+1$ | $2t$ | $T_{mult}+T_{add}$ |
| RiBM | $3t+1$ | $6t+2$ | $6t+2$ | $3t+1$ | $2t$ | $T_{mult}+T_{add}$ |
| Euclidean | $4t+2$ | $8t+8$ | $4t+4$ | $8t+8$ | $2t$ | $T_{mult}+T_{add}+T_{mux}$ |
| Euclidean (folded) | $2t+1$ | $2t+1$ | $10t+5$ | $14t+7$ | $12t$ | $T_{mult}+T_{add}+T_{mux}$ |

**Table 6-2: Comparison of Hardware complexity and Path Delay [6]**

It is possible to implement the **eE** algorithm with complex processor elements, as described by Shao et al. [4]. Here, the four multiplications in each processor are computed using four separate multipliers. The architecture described in [4] uses only $2t+1$ processors as compared to the $3t+1$ **PE0** or **PE1** processors needed in the **riBM** and **RiBM** architectures, but each processor in [4] has 4 multipliers, four multiplexers, and two adders. As a result, the **riBM** and **RiBM** architectures compare very favorably to the **eE** architecture of [4]—the reformulated iBM architectures achieve the same (actually slightly higher) throughput with much smaller complexity.

All the multiplexers in the **riBM** and **RiBM** architectures receive the same signal and the computations in these architectures is purely systolic in the sense that all processors carry out exactly the same computation in each cycle, with all the multiplexers set the same way in all the processors—there are *no* cell-specific control signals.

## 6.8 Simulation and Synthesis of iBM,riBM and RiBM Architectures

We have used Xilinx Integrated Simulation Environment (ISE) v9.2 for the design process of RS codec. Simulation and synthesis are done using Xilinx ISE Simulator and Xilinx Synthesis Tool(XST) respectively. Target selected was Spartan-3 Xc3s500 with a speed grade -5.

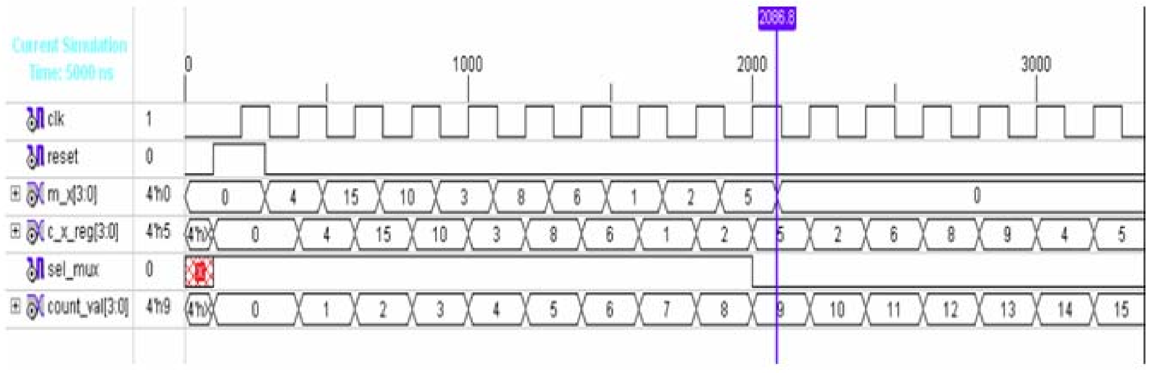
### 6.8.1 Simulation Results for Reed Solomon Codec

Process of systematic Reed Solomon encoding for (15,9) code (where n = 15, k = 9 and m = 4) over GF($2^4$) are shown in Figure 6-18. The first and second waveforms of the clock and reset signals respectively. On the positive edge o f the clock after the reset goes low, message symbols (3rd waveform) start emerging and transition at the negative edge of the clock. The message polynomial used in this example simulation is as follows:

$$m(x) = 5 + 2x + x^2 + 6x^3 + 8x^4 + 3x^5 + 10x^6 + 15x^7 + 4x^8$$

And the encoded polynomial is

$$c(x) = 5 + 4x + 9x^2 + 8x^3 + 6x^4 + 2x^5 + 5x^6 + 2x^7 + x^8 + 6x^9 + 8x^{10} + 3x^{11} + 10x^{12} + 15x^{13} + 4x^{14}$$



Time (ns)

**Figure 6-18: Systematic RS Encoding simulation waveform**

For simulation purposes, it is assumed that 3 errors have occurred during the transmission of the code-word as the received word is as follows:

$$r(x) = 5 + 4x + \overline{13x^2} + 8x^3 + 6x^4 + 2x^5 + 5x^6 + 2x^7 + \overline{2x^8} + 6x^9 + 8x^{10} + 3x^{11} + 10x^{12} + 15x^{13} + \overline{6x^{14}}$$

Symbols in errors are shown with a bar above them. Now, the task of the decoder is to find out both the locations of the error and their corresponding values. i.e. the error polynomial

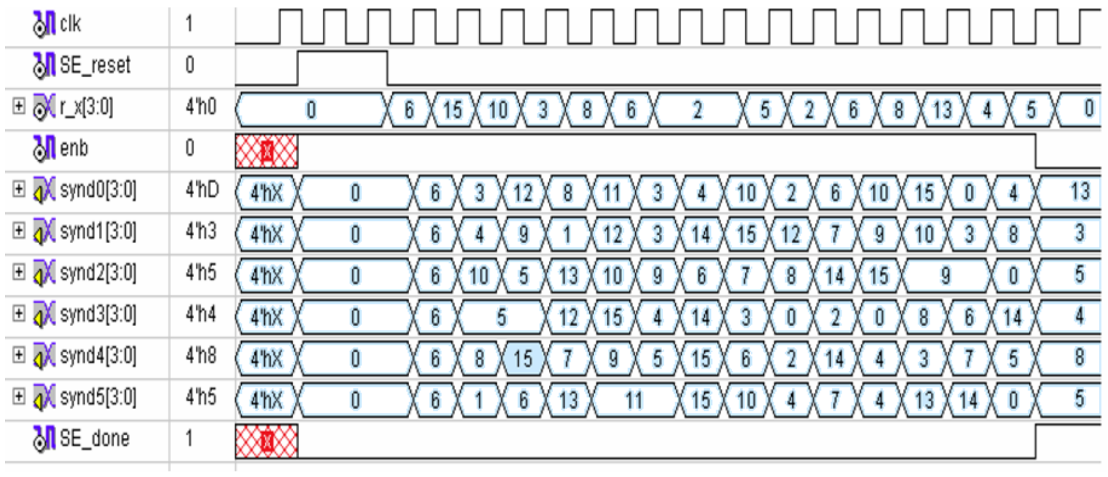
$$e(x) = 4x^2 + 3x^8 + 2x^{14}$$

Syndrome computation for this example is shown in simulation wave-form in Figure 6-19. The received word r(x) shown in the third row is input to the Syndrome Computation module. As t = 3, for (15,9) code, there are 2t = 6 syndromes. Syndromes get evaluated one cycle after the received word has entered completely into SC block. Computed syndromes are

$$S = \{13,3,5,4,8,5\}$$

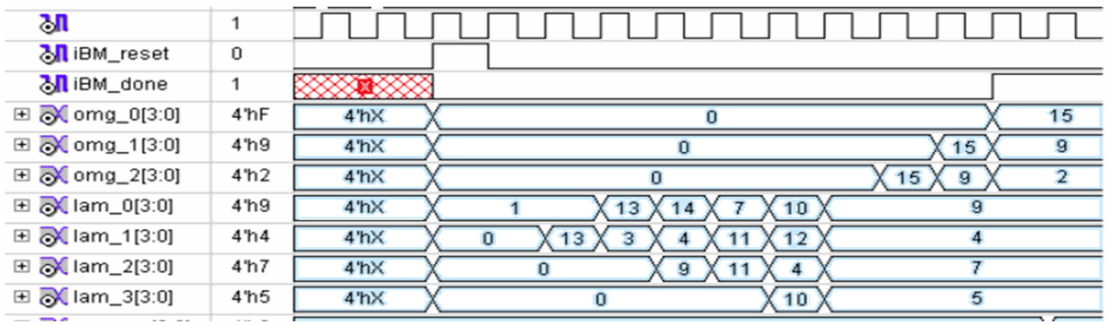
SE_done goes high as the computation of the syndrome completes.

After the syndromes are calculated, the Key Equation Solver block computes the error-locator and the error-evaluator polynomial. Simulation results for both iBM architecture and RiBM architectures are presented for comparison in Figure 6-20 and Figure 6-21respectively. It can be observed from the simulation waveforms that the error-locator and error-evaluator polynomials are computed in about 50% less time (2t cycles) in RiBM and riBM as compared with iBM (2t cycles). Coefficients of error-locator polynomial are indicated by lam_i and those of error-evaluator polynomial as omg_i.



Time (ns)

**Figure 6-19: Syndrome Computation**



Time (ns)

**Figure 6-20: Key Equation Solver Simulation for iBM Architecture**

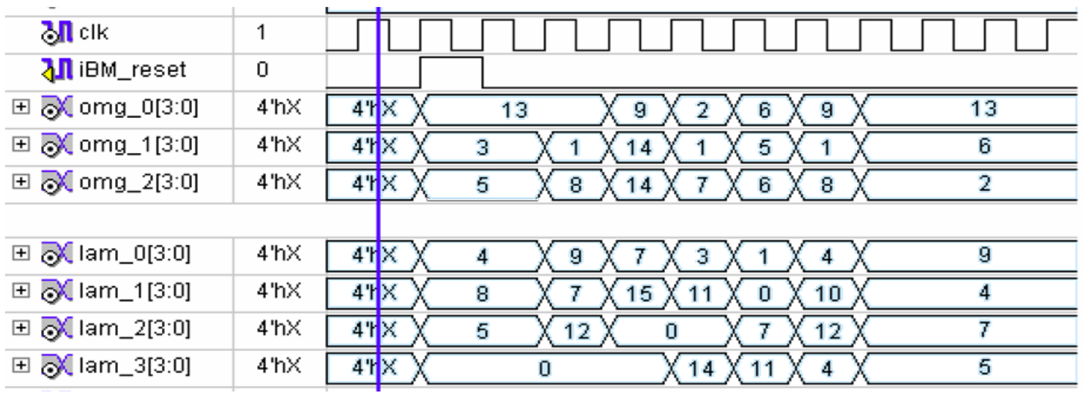

clk 1
iBM_reset 0
omg_0[3:0] 4'hX 13 9 2 6 9 13
omg_1[3:0] 4'hX 3 1 14 1 5 1 6
omg_2[3:0] 4'hX 5 8 14 7 6 8 2
lam_0[3:0] 4'hX 4 9 7 3 1 4 9
lam_1[3:0] 4'hX 8 7 15 11 0 10 4
lam_2[3:0] 4'hX 5 12 0 7 12 7
lam_3[3:0] 4'hX 0 14 11 4 5

**Figure 6-21: Key equation solver simulation for RiBM and riBM architecture**

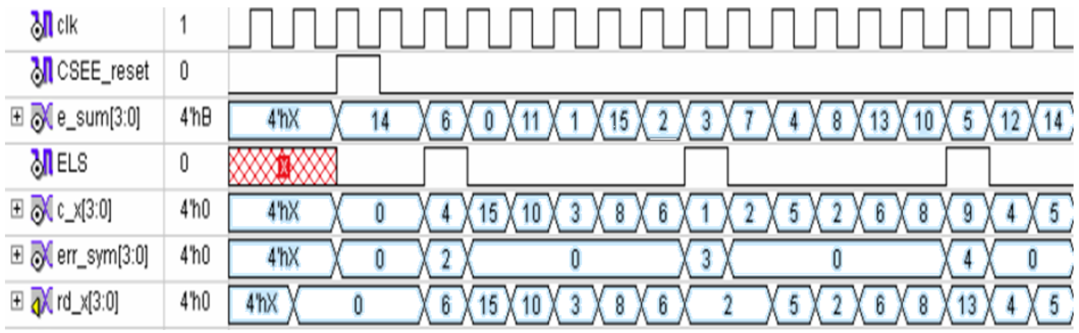

clk 1
CSEE_reset 0
e_sum[3:0] 4'hB 4'hX 14 6 0 11 1 15 2 3 7 4 8 13 10 5 12 14
ELS 0
c_x[3:0] 4'h0 4'hX 0 4 15 10 3 8 6 1 2 5 2 6 8 9 4 5
err_sym[3:0] 4'h0 4'hX 0 2 0 3 0 4 0
rd_x[3:0] 4'h0 4'hX 0 6 15 10 3 8 6 2 5 2 6 8 13 4 5

**Figure 6-22: Error-correction**

Figure 6-22 shows the results of Chein-search and Error-evaluator block. Error location sequence indicates the location of error and err_sym are the values of the errors. CSEE_reset enables the block and a delayed version of received signal rd_x is added with the error sequence to get the decoded code-word symbols c_x.

### 6.8.2 Synthesis Results for Reed Solomon Codec:

Reed Solomon codec was synthesized for Spartan-3 speed grade-5 with various values of the parameters n,k and t. This was accomplished by writing the Matlab code for a Verilog HDL code generator application. Values of the parameters n and k are specified to the application and it generates the required Verilog files in a directory ready to be synthesized by Xilinx ISE.

Synthesis report generated by XST contains details about the resource usage and maximum attainable clock frequency which is the inverse of the critical path delay.

### 6.8.2.1 Synthesis results with changing Error-correction capability ‘t’:

Synthesis results for 4 different RS codec parameters are shown in Table 6-3 for RiBM.

| RS Code (n,k) | Slices | Flip_Flops | 4-input LUTs | Max.Freq in MHz. |
|---|---|---|---|---|
| (15,11) | 164 | 118 | 300 | 198 |
| (15,9) | 234 | 162 | 427 | 204 |
| (15,7) | 302 | 205 | 551 | 206 |
| (15,5) | 368 | 247 | 672 | 208 |

**Table 6-3: Area vs. Speed Comparison with increasing Error-correction capability (n=15)**

It can be observed that as we increase the error-correction capability of the code without changing the code-size (and hence the underlying Galois Field) there is an increase in the area (resource consumption) however, the speed remains almost the same because we are using same GF(16) and the critical path delay is the sum of the delays of adder and multiplier. Results in the table are shown in the graph in Figure 6-23.

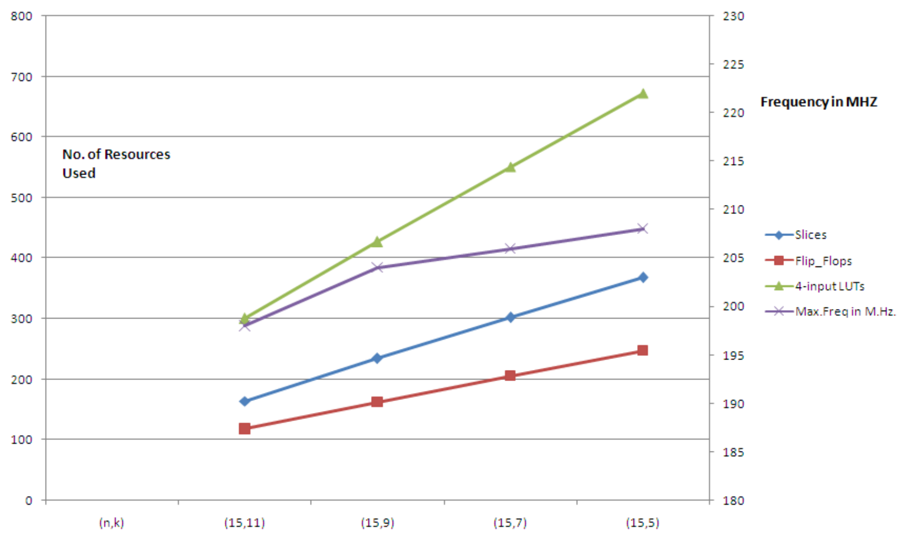


**Figure 6-23: Area vs. Speed Comparison for (n,k) RS Code (n =15) for RiBM**

Similar results are observed for n = 63. Tabulated data is as follows :

| RS Code (n,k) | Slices | Flip_Flops | 4-input LUTs | Max.Freq(MHz ) |
|---|---|---|---|---|
| (63,55) | 574 | 290 | 1057 | 164 |
| (63,47) | 1107 | 537 | 2034 | 160 |
| (63,31) | 2179 | 1123 | 4017 | 155 |

**Table 6-4: Area vs. Speed Comparison with increasing Error-correction capability (n=63)**

These results are displayed in the graph in Figure 6-24.

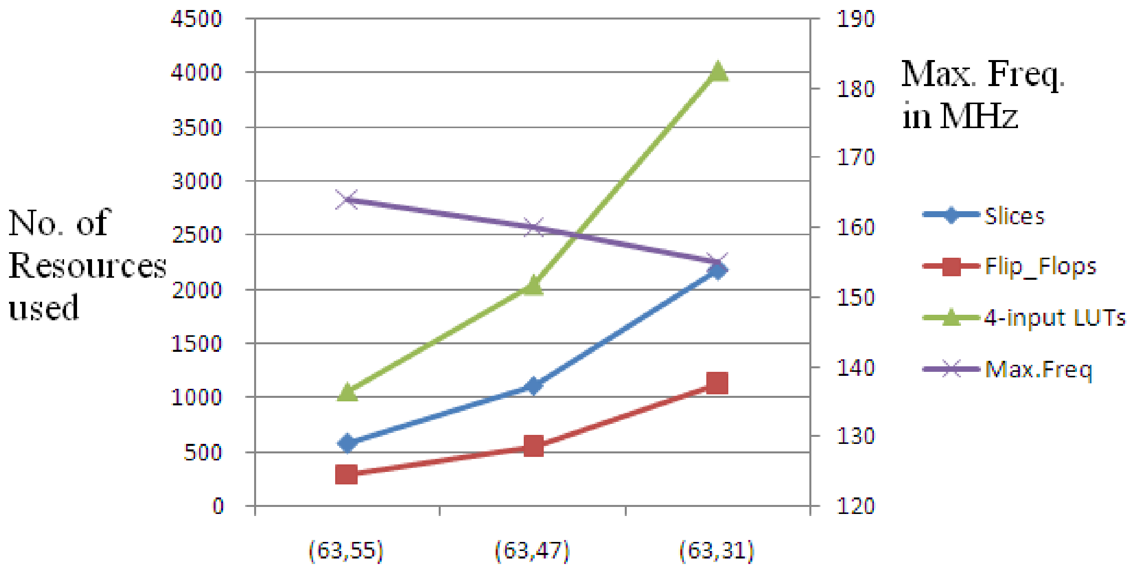


**Figure 6-24: Area vs. Speed Comparison for (n,k) RS Code (n =63) for RiBM**

As we increase the value of n, we operate in larger Galois Fields with multipliers having greater critical path delays and lower maximum clock period. This can be observed by the graph in Figure 6-25

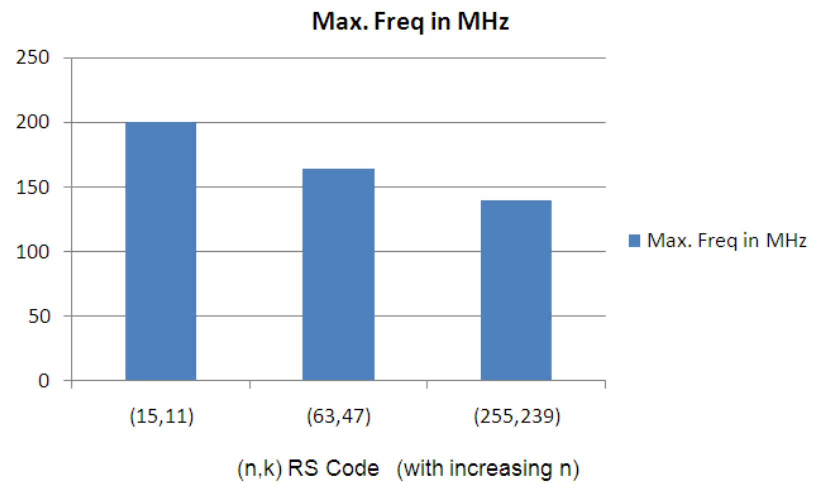


**Figure 6-25: Increase in critical path delay with increasing code size**

Figure 6-26 compares three different inversion-less BM architectures. These architectures when synthesized for (15,9) show that they use almost the same number of slices, flip-flops and LUT (look up tables). However, the systolic and homogeneous architecture of RiBM makes it the fastest i.e. with the minimum critical path delay.

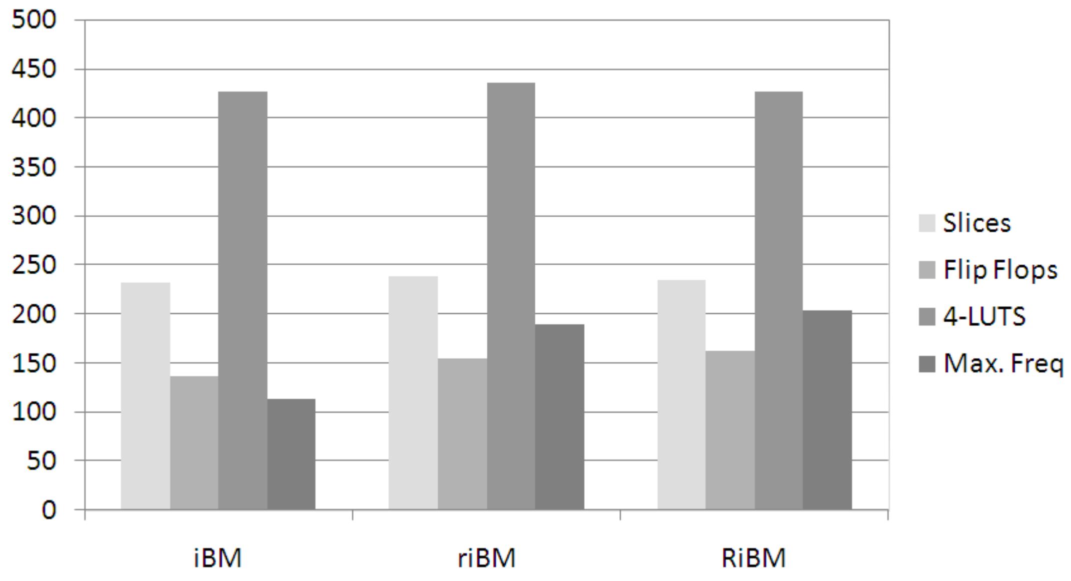


**Figure 6-26: Area-Speed comparison of iBM, riBM and RiBM architectures**

This chapter concludes that application of algorithmic transformations to the Berlekamp–Massey algorithm result in the **riBM** and **RiBM** architectures whose critical path delay is less than half that of conventional architectures such as the **iBM** architecture. The **riBM** and **RiBM** architectures use systolic arrays of identical processor elements.

# CHAPTER 7

# CONCLUSIONS AND FUTURE WORK SUGGESTIONS

## 7.1 Conclusions

This work aimed at investigation, simulation and implementation of various Reed Solomon encoding and decoding architectures as well as development of a code acquisition system for Direct sequence spread spectrum communication systems. Simulation and programming was done in Matlab while HDL implementation and synthesis were carried out in Xilinx Integrated Simulation environment.

Reed Solomon decoding algorithms implemented in Matlab include Berlekamp-Massey(BM) algorithm, Extended Euclidean (eE) algorithm, Berlekamp-Welch(BW) modular decoding approaches, Guruswami-Sudan(GS) list decoding algorithm, Inversion-less Berlekamp Massey algorithm (iBM), Reformulated versions of iBM i.e. riBM and RiBM. Algorithms are compared based upon their structure, operational complexity, and critical path delay and error-correction capability.

Guruswami-Sudan decoding algorithm can correct errors beyond half the minimum bound but involves high computational cost. We observed that BM algorithm which requires division in each iteration and results in irregular architectures can be transformed to an inversionless form (iBM). However, iBM algorithm's critical path delay is dependent upon the error-correction capability of the code which is highly undesirable. Reformulated versions of iBM i.e. riBM and RiBM have a very regular and systolic architecture having critical path delay lowest among all the current decoding approaches. iBM and its reformulated forms were implemented in Verilog HDL and their simulation was carried out using Modelsim and Xilinx ISE. A Reed Solomon Codec depends upon three parameters n (block length), k (message length) and t (no. of correctable errors) only two of which are independent. These parameters may be required to change in any step during the receiver design. So, in order to add flexibility for the system designer, a scalable and parameterizable code must be written. Adding this flexibility in Verilog language directly is very tough if not impossible. Solution to this problem is obtained by writing a Code generator in Matlab which is capable of generating all the Verilog source files for the required set of parameters.

Second part of the work involves the implementation of a code acquisition system for direct sequence spread spectrum (DSSS) systems. A parallel search acquisition strategy is adopted and correlation is performed in frequency domain for computational efficiency. Built-in efficient Xilinx cores for Fast-Fourier Transform and Complex Multiplier are used by this design.

## 7.2 Future Work Suggestions

This thesis involved work on two very important components of a digital communication receiver, that is, Synchronization and Channel Coding. The implemented schemes can be used with other receiver modules to integrate into a functional receiver. For example, a DSSS tracking system should accompany the Code acquisition system for fine synchronization. Similarly, a concatenated channel coding scheme utilizing both Convolutional codes and Reed Solomon codes can be implemented to get greater coding gains.

The critical path delay of Reed Solomon decoding architectures depends upon the delay of the Galois Field (GF) multiplier. Use of an efficient and fast GF multiplier can increase the speed of the implemented architectures substantially.

VLSI architectures for interpolation based Reed Solomon decoder architectures can be derived and implemented for better error-correction capability. This work can also be extended by utilizing soft-decision reliability information from the channel for better decoding performance.

# REFERENCES

[1] T.K. MOON, *Error correction coding, mathematical methods and algorithms*, New Jersey, John Wiley and Sons, Inc., 2005.

[2] H. O. BURTON, *Inversion-less Decoding of Binary BCH codes*, IEEE Transactions on Information Theory, Vol. IT-17, No. 4, July 1971, pp. 464-466.

[3] I.S. REED, M.T. SHIH, and T.K. TRUONG, *VLSI design of inverse-free Berlekamp-Massey algorithm*, IEEE proceedings-E, Vol 138, No. 5. Sep. 1991, pp. 295-298.

[4] 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 transactions on Computers, Vol. c-34, No. 5, May 1985.

[5] M. MEHNERT, D.F. DROSTE, and D. Schiel, *VHDL implementation of a (255,191) Reed Solomon Coder for DVB-H,* IEEE 1-4244-0216-6/06, 2006.

[6] D.V. SARWATE and N.R. SHANBHAG, *High-speed Architectures for Reed-Solomon Decoders*, IEEE Transactions on VLSI systems, Vol. 9, No. 5, Oct. 2001. pp. 641-655.

[7] W.J. GROSS, F.R. KSCHINCHANG, R. KOETTER, and P.G. GULAK, *Towards a VLSI architecture for Intepolation-based Soft-decision Reed-Solomon Decoders,* Springer Science, Journal of VLSI signal processing 39, 2005, pp. 93-111.

[8] R.J. McEleice, *The Guruswami-Sudan Decoding Algorithm for Reed-Solomon Codes,* IPN progress report 42-153, May 2003.

[9] A. AHMED, R. KOETTER, and N. SHANBHAG, *VLSI architectures for Soft-decision decoding of Reed-Solomon Codes,* Coordinated Science Lab., UIUC, Draft,Feb. 2003

[10] R. BOSE, *Information Theory, Coding and Cryptography*, Tata Mc Graw-Hill, New Delhi, 2003.

[11] Fast Fourier Transform v4.1, Product Specification, www.xilinx.com, DS260 Apr. 2007.

[12] N.A. MIR, F. ABBAS, Implementation of Reed Solomon Decoder and Optimization for Tri-media Processor, Thesis, College of E&ME, Rawalpindi, 1997.

[13] A. IQBAL, *Scalable VLSI architecture for Reed Solomon Codec*, Thesis report, College of E&ME, Rawalpindi, 2003.

[14] M. A. ATTA, *Design and Implementation of Digital Transmitters*, M.S. Thesis, College of E&ME, Rawalpindi ,2001

[15] S. HAYKIN, *Communication Systems*, 4th Edition, John Wiley and Sons, New Jersey, 2001.

[16] B. SKLAR, *Digital Communications Fundamentals and Applications,* Pearson Education Private Limited.

[17] W.H. TRANTER, K.S. SHANMUGAN, T.S. RAPPAPORT, and K.L. KOSBAR, *Principles of Communication Systems Simulation*, Pearson Education, New Delhi, 2004.

[18] Wikipedia Encyclopedia, *www.wikipedia.org*.

[19] M. SUDAN, "*Decoding of Reed-Solomon codes beyond the Error-correction bound*," J. Complexity, vol.13, pp. 180-193, 1997.

[20] L.R. WELCH and E.R. BERLEKAMP, "*Error correction for Algebraic block codes*", U.S. Patent No. 4,633,470, Dec. 30, 1986.

[21] L.R. WELCH and R.A. SCHOLTZ, "*Continual Fractions and Berlekamp's Algorithm* ," IEEE Trans. On Information Theory, vol. 25, no. 1, pp. 19-27, Jan. 1979.

[22] V. GURUSWAMI and M.SUDAN, "*Improved Decoding of Reed-Solomon codes and Algebraic Geometry codes", *IEEE Trans. Info. Theory, vol. 45, no. 6, pp. 1757-1767,Sep. 1999.

[23] R. KOETTER, *On Algebraic Decoding of Algebraic-Geometric and Cyclic Codes, Linkoping Studies in Science and Technology*, no. 419 (Ph.D. Dissertation, Department of Electrical Engineering), Linkoping U.,1996.

[24] R. KOETTER, "*Fast Generalized Minimum-Distance Decoding of Algebraic-Geometry and Reed-Solomon Codes",* IEEE Trans. Info. Theory, vol. 42, no. 3. pp. 721-736, May 1996.

[25] R. ROTH and G. RUCKENSTEIN, "*Efficient Decoding of Reed-Solomon Codes beyond Half the Minimum Distance,"* IEEE Trans. Inform. Theory, vol. 46, no. 1, pp. 246-257, Jan. 2000.

[26] S. GLISIC and B. VUCETIC, “*Spread spectrum CDM A systems for wireless communications,*” Artech house publications, London, 1997.

[27] V.P. IPATOV, “*Spread spectrum and CDMA principles and applications*”, John Wiley & Sons, West Sussex, 2005.