Design and Implementation of Configurable Reed Solomon Decoder Using Euclidean Algorithm

by

Hamed Salah El-Din Hamed

A Thesis Submitted to the
Faculty of Engineering at Cairo University
in Partial Fulfillment of the
Requirements for the Degree of
MASTER OF SCIENCE
in
Electronics and Electrical Communications Engineering

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
September 2010

Acknowledgment

I would like to thank my supervisors Prof. Dr. Amin Nassar, who is not only my supervisor but he is like my father, and Dr. Hossam Aly Hassan Fahmy who taught me how to be a researcher and the most important thing is how to evaluate yourself and I want to ask to him if I made anything made him unhappy, please forgive me. Also I would like to thank Dr. Tallal El-shabrawy, owner of the credit in this work. I could not achieve anything without his help. They helped me not only to complete this work but also to organize my academic life.

Also I want to thank all the faculty members in GUC Prof. Dr. Yasser Hegazy, Prof. Dr. Ahmed El mahdy, Dr Amr Tallat, Dr. Mohamed Ashour, and Dr. Hany Hammad for their sincere advices are very fruitful.

I would like also to thank my study partner Hazem A. Ahmed. Also I would like to thank Walid Galal and Soukry Ibrahim for their valuable advices and my colleague Mohamed Fattouh who taught me VHDL and also Eng. Amr Abdulzahir and Eng. Alhussein. I can not explain my gratitude to my father, and mother. They all helped me to achieve progress through my life.

Abstract

Due to the increasing of the speed of modern communications systems the detection and correction of errors in digital information have become a very important issues. Such errors almost inevitably occur after the transmission, storage or processing of information in digital form, because of noise and interference in communication channels, or imperfections in storage media, for example. So the protecting of digital information with a suitable error-control code enables the efficient detection and correction of any errors that may have occurred. In high-speed communication systems, Reed-Solomon (RS) codes have a widespread use to provide error protection especially for burst errors. This feature has been one of the important factors in adopting RS code in many practical applications such as digital audio and video, magnetic and optical recording, computer memory, cable modem, and wireless communications systems. We designed a low-energy configurable multi-channel Reed-Solomon RS(255, 239) decoder using Euclidean algorithm. The configurable structure enables the decoder to be shared with 8 or 16 channels. The proposed design presents also a configurable syndrome cell and Chien search cell which can work as two parallel or serial cells to achieve high throughput and reduce the power consumption. The reduction of the power consumption and the latency of the architecture lead to save the energy dissipation and save the battery in hand held devices.

Contents

1 INTRODUCTION **1**

1.1 Introduction to Communication System 1

1.2 Types of Codes 3

1.2.1 Binary Block Codes 3

1.2.2 Convolutional Code 4

1.3 Introduction to Reed-Solomon Codes 5

1.4 Motivation 5

1.5 Thesis Organization 5

2 REED-SOLOMON CODES **7**

2.1 Algebraic Structures 7

2.1.1 Groups 7

2.1.2 Rings 8

2.1.3 Fields 9

2.1.4 Galois Field GF(q) 11

2.1.5 The extended field GF(2^m) 11

2.1.6 Basis of field GF(2^m) 14

2.1.7 Basis Conversions 16

2.2 Finite Field Multipliers 20

2.2.1 Multiplication by a constant value 20

2.2.2 Two-Variable operands Multipliers 21

2.3 Introduction to Cyclic Codes 27

2.3.1 Polynomial Representation of Codewords 28

2.3.2 Encoding of a Cyclic Code 28

2.3.3 Encoding of Cyclic Codes in Systematic Form 30

2.4 Encoding of Reed Solomon Code 32

2.4.1 RS Codes in Systematic Form 32

2.4.2 Implementation of RS Encoder 33

3 REED-SOLOMON DECODER 35
3.1 Syndrome Calculation . . . 36
3.2 The Decoding Algorithm . . . 37
3.2.1 The Key Equation . . . 38
3.2.2 Decoding of RS Codes Using the Euclidean Algorithm . . . 39
3.2.3 Decoding of RS Codes Using Berlekamp-Massey Algorithm . 41
3.2.4 Decoding of RS Codes Using Peterson-Gorenstein-Zierler (PGZ) Algorithm . . . 46
3.3 Comparison of Decoding Algorithms . . . 49
3.3.1 Complexity . . . 49
3.3.2 Critical path . . . 49
3.3.3 Latency . . . 49
3.3.4 Power consumption . . . 50
3.4 Chien Search . . . 50
3.5 Forney Algorithm . . . 51

4 CONFIGURABLE MULTI-CHANNEL REED-SOLOMON DECODER 53
4.1 Architectures of Syndrome computation . . . 53
4.1.1 Serial architecture . . . 53
4.1.2 Two Parallel architecture . . . 54
4.1.3 The new Configurable architecture . . . 55
4.2 Hardware Implementation of Euclidean Algorithm . . . 56
4.2.1 Euclidean Division Module . . . 57
4.2.2 Euclidean Multiply Module . . . 58
4.3 Architectures of Chien search block . . . 59
4.3.1 Serial architecture . . . 59
4.3.2 Two Parallel architecture . . . 60
4.3.3 The new configurable architecture . . . 61
4.4 Architecture of Forney Algorithm . . . 62
4.4.1 Serial architecture . . . 62
4.4.2 Two Parallel architecture . . . 63
4.4.3 The new configurable architecture . . . 64
4.5 Multi-Channel Decoder . . . 64
4.5.1 Multi-Channel using serial architectures . . . 64
4.5.2 Multi-Channels using two parallel architectures . . . 65
4.5.3 Multi-Channel using configurable architectures . . . 65

4.6 Results and Comparisons 66
4.6.1 Results of configurable (255, 239) RS decoder 66
4.6.2 Comparisons 67
4.7 Conclusion 68

5 CONCLUSIONS AND FUTURE WORKS 69
5.1 Conclusions 69
5.2 Future Works 70

A The list of primitive polynomials $p(x)$ for m $\leq$ 10 75

B Conversions from standard basis to Normal basis in GF(2^4) 77

List of Tables

1.1 Binary block code with $k = 4$ and $n = 7$ 4

2.1 Binary representation of GF(2^m) 14
2.2 RS codes parameters 32

3.1 B-M algorithm table for determining the error-location polynomial . 44

4.1 Implementation Results of single channel RS(255, 239) Decoders . 67
4.2 Implementation Results of Multi-Channel RS Decoders 67

List of Figures

1.1 Block diagram of a typical data transmission or storage system. . . 2
1.2 Simplified model of a coded system. 3

2.1 Bit-parallel dual basis multiplier for GF(2^3) 23
2.2 Type A module for a bit-parallel dual basis multiplier for GF(2^3) . 24
2.3 Type B module for a bit-parallel dual basis multiplier for GF(2^3) . 24
2.4 PPBM for GF(2^4). 26
2.5 Module B of the PPBM 27
2.6 Circuit for multiplying by α in GF(2^4) 27
2.7 The LFSR architecture of RS encoder. 33

3.1 Block Diagram of RS Decoder 35

4.1 Serial syndrome 54
4.2 Serial syndrome cell 54
4.3 Two parallel Syndrome 55
4.4 Two Parallel syndrome cell 55
4.5 Configurable syndrome cell 56
4.6 Block diagram of the Euclidean architecture. 57
4.7 Overall architecture of the Euclidean divider module. 57
4.8 EAdiv A and EAdiv B of the Euclidean divider module. 58
4.9 Block diagram of the Euclidean multiply module. 58
4.10 Architecture of the EAmul C module in the Euclidean multiply operation. 59
4.11 Serial Chien search ($t = 8$). 60
4.12 Serial Chien search cell 60
4.13 Two Parallel Chien search ($t = 8$) 61
4.14 Two Parallel Chien search cell 61
4.15 Configurable Chien search cell 62

4.16 Serial Forney architecture 63
4.17 Two parallel Forney architecture 63
4.18 16 Channel (255, 239) RS decoder using serial architectures 64
4.19 8 Channels (255, 239) RS decoder using two parallel architectures . 65
4.20 8 Channels (255, 239) RS decoder configurable and serial architectures 66

Chapter 1

INTRODUCTION

1.1 Introduction to Communication System

Due to a high demand for efficient and reliable digital data transmission in communication systems and storage devices, a great effort had been exerted in this area to merge between communication and computer technology in the design of these systems.

In 1948, in his classic paper "A Mathematical Theory of Communication" C. Shannon [1] introduced the main concepts of what is known as information theory. Since Shannon's work, a great of effort has been exerted on the problem of proposing efficient encoding and decoding methods for error control in a noisy channels. The use of coding for error control has become an integral part in the design of modern communication systems and digital storage devices in computers to achieve the reliability required by today's high speed digital systems.

The transmission in communication systems and storage of digital information have many common properties. Both of them transfer data from an information source to a destination. The transmission system or storage device can be represented by the block diagram shown in figure 1.1.

The information source output can be either analog signal, in case of medical application or digital signal in case of digital computer.

The source encoder transforms the source output into a suitable sequence of **b**inary dig**its** (bits) as it may give each symbol different length of bits to minimize the size of data and its output called the information sequence u. In case of a analog information source, the source encoder contains analog to digital converter (ADC).

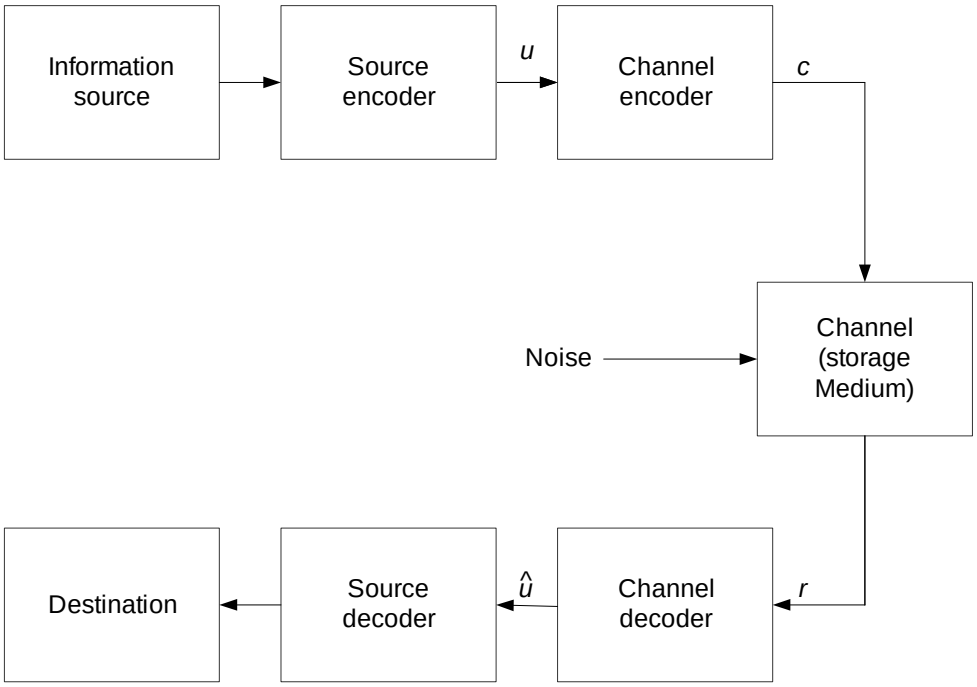

Figure 1.1: Block diagram of a typical data transmission or storage system.

The channel encoder converts the information sequence u into an encoded sequence c called a codeword or code vector. In most applications c is also a binary sequence, although in some applications non-binary codes have been used as each symbol in non-binary code is represented in binary sequence.

The channel is the medium which the data is transferred through it and its nature depends on the application . It may be wireless channel in mobile application or wired channel in ADSL. But in storage devices the channel is the device itself. The channel determines the type of errors which affects the encoded sequence c, it will be random errors or burst errors so it also determines the type of code to be used. So the channel transforms the encoded sequence v to received sequence r which may has some errors.

The channel decoder corrects the errors in the received sequence r to transform it into a binary sequence $\hat{u}$ called the estimated sequence. The decoding technique is based on the type of channel encoding. Ideally,$\hat{u}$ will be the same as the information sequence u, but actually the noise may cause some decoding errors which makes difference between the information sequence u and the estimated sequence $\hat{u}$.

The source decoder transforms the estimated sequence $\hat{u}$ into an estimate of

the source output and delivers this estimate to the destination. When the source is an analog source, it contains digital to analog converter (DAC).

In this thesis we will focus on the channel encoder and channel decoder, So the block diagram in figure 1.1 can be simplified as shown in figure 1.2.

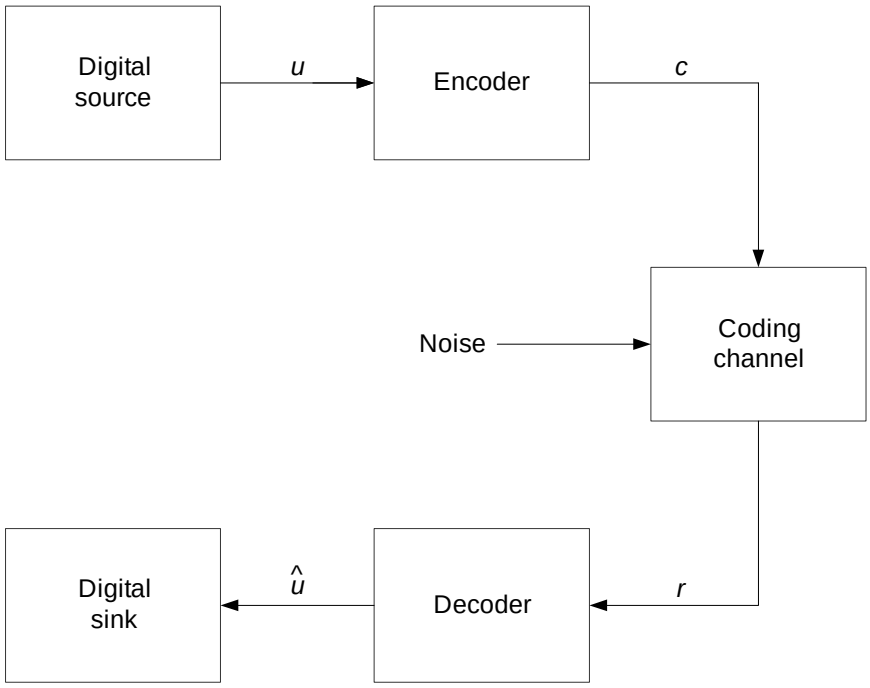

Figure 1.2: Simplified model of a coded system.

1.2 Types of Codes

There are two different types of codes in common use today, block codes and convolutional codes.

1.2.1 Binary Block Codes

The encoder for a block code divides the information sequence into message blocks, each block of k information bits. A message block is represented by k bits $u = (u_0, u_2, ..., u_{k-1})$ called a message. There are 2^k different possible messages each of them of length k bits. The encoder encodes each message independently into an n-bits sequence $c = (c_0, c_2, ..., c_{n-1})$ called a code word. Therefore, corresponding to the 2^k different possible messages, there are 2^k different possible code words of

length n bits at the encoder output. So in this case we call the code by (n, k) block code. The ratio $R = k/n$ is called the code rate, and it represents the number of information bits corresponding to the transmitted bits and it's smaller than one, $R < 1$ and $n - k$ is called redundant bits [2]. These redundant bits provide the code the capability of detecting and correcting errors in the received vectors. How to add these redundant bits to the message to achieve reliable transmission over a noisy channel is the major problem in designing the code. An example of a binary code with $k = 4$ and $n = 7$ is shown in Table 1.1.

Table 1.1: Binary block code with $k = 4$ and $n = 7$

Message	Code words
0000	0000000
1000	1101000
0100	0110100
1100	1011100
0010	1110010
1010	0011010
0110	1000110
1110	0101110
0001	1010001
1001	0111001
0101	1100101
1101	0001101
0011	0100011
1011	1001011
0111	0010111
1111	1111111

1.2.2 Convolutional Code

The encoder for a convolutional code also divides the information sequence to k-bit blocks of message u and produces a code word c of n-bits length. In convolutional coding, each code word depends not only on the corresponding k-bit message block at the same time unit, but also m previous message blocks. Hence, the encoder has m memory elements. Each bit from k bits message enters from different input and similarly we have n different outputs. So The encoded sequences produced by a k-input, n-output encoder of m memory elements is called an (n, k, m) convolutional code. The ratio $R = k/n$ also is called the code rate. Also in convolutional code the redundant bits are added to provide the code the capability of correcting errors in the received vectors so the code rate is smaller than one, $R < 1$ [2]. How to

use the memory to achieve reliable transmission over a noisy channel is the main problem in designing the convolutional encoder [2].

1.3 Introduction to Reed-Solomon Codes

In June of 1960, Irving Reed and Gus Solomon published a paper in the Journal of the Society for Industrial and Applied Mathematics with title of "Polynomial Codes over Certain Finite Fields" [3]. This paper described a new type of error-control codes that are now called Reed-Solomon codes. Reed-Solomon Codes are non-binary block codes so it has the properties of block codes. These codes are encoded by the same way of encoding the binary codes, but decoding was very complicated and a great efforts was exerted by Berlekamp [4] and others to improve its decoding algorithm. In 1971, Reed-Solomon codes used by NASA for some of their missions to Mars. The problem in this code is the decoding algorithm which was very complicated so it was implemented on computers on earth. In recent years, a great efforts have been exerted in developing this field to enhance the decoding algorithms, and nowadays Reed-Solomon codes are used for CDs, wireless communication, DVD or digital TV.

The most important advantage of Reed-Solomon codes over binary linear block codes is the ability of correcting the burst errors. As in Reed-Solomon codes the information bits are divided into message blocks of k information symbols each and each symbol consists of m bits. The encoder encodes each message independently to n symbols. The RS(n, k) can correct up to t symbols where $t = \left\lfloor \frac{n-k}{2} \right\rfloor$, and that regardless the number of error bits in each symbol.

1.4 Motivation

While the RS code is used in many digital applications, a great efforts were exerted to develop the decoding algorithms and to implement them. But all these efforts were focusing on either the speed of the implementations or the area of them. So in this thesis we are focusing on a compromising between the area and the speed and also the energy consumed by the decoder.

1.5 Thesis Organization

The organization of this thesis is as follows:

In Chapter 2, we introduce the fundamentals of different algebraic structures, Galois field properties and different structures of Galois field multipliers, then finally an introduction to cyclic codes is introduced. In Chapter 3, we introduce the RS decoder and many types of decoding algorithms and which of them we will choose. In Chapter 4, the hardware implementation of the decoder, and also a comparison between the proposed architecture and the previous architectures are introduced. Finally, some conclusions and future work will be given in Chapter 5.

Chapter 2

REED-SOLOMON CODES

2.1 Algebraic Structures

In this section we will define some of algebraic basics that we need throughout the thesis. We start with the definition of some algebraic structures [5] and why we choose one of them to work over it. The most important algebraic structures in the algebraic coding theory are groups, rings and finite fields. Also we will introduce the binary and non-binary finite fields and their properties.

2.1.1 Groups

A non-empty set G together with a binary operation ' $*$ ' is called a group if for all elements a, b, $c \in G$ the following properties must be satisfied [5]:

1- Closure :

$$a * b \in G, \tag{2.1}$$

2- Associative :

$$a * (b * c) \;\; = \;\; (a * b) * c, \tag{2.2}$$

3- Identity :

$$\exists e \in G \, : \, \forall a \in G \, : \;\; a * e \;\; = e * a = a, \tag{2.3}$$

4- Inverse :

$$\forall a \in G \, : \, \exists \grave{a} \in G \, : \, a * \grave{a} \;\; = \;\; e. \tag{2.4}$$

The element e is the identity element, and $\grave{a}$ is the inverse element of element a.

5- Commutative :

$$a * b \;\; = \;\; b * a \tag{2.5}$$

The number of elements of a group G determines the group order, ord (G). If the number of elements is finite, the group is a finite group. From previous definition and properties of group we can note that one operation is defined in the structure. So we can either add or multiply but in our application we need the two operations so this structure is not suitable for RS codes.

2.1.2 Rings

A ring S is a non-empty set of elements with two operators usually called addition and multiplication, denoted ' $+$ 'and ' $*$ ' respectively. For S to be a ring a number of conditions must hold for all elements a, b, $c \in S$ [5]:

1- Closure :

$$\begin{aligned} a * b \;\; &\in \;\; S, \\ a + b \;\; &\in \;\; S \end{aligned} \tag{2.6}$$

2- Associative :

$$\begin{aligned} a + (b + c) \;\; &= \;\; (a + b) + c \\ a * (b * c) \;\; &= \;\; (a * b) * c, \end{aligned} \tag{2.7}$$

3- Identity :

$$\begin{aligned} \exists 1 \in S : \forall a \in S : \quad a * 1 \quad &= 1 * a = a \\ \exists 0 \in S : \forall a \in S : \quad a + 0 \quad &= 0 + a = a, \end{aligned} \tag{2.8}$$

4- Inverse :

$$\forall a \in S : \exists -a \in S : a + (-a) \quad = \quad 0. \tag{2.9}$$

The element 0 is the identity element of addition, the element 1 is the identity element of multiplication, and $-a$ is the additive inverse element of element a.

5- Commutative :

$$\begin{aligned} a + b &= b + a \\ a * b &= b * a \end{aligned} \tag{2.10}$$

6- Distributive :

$$(a + b) * c = a * c + b * c \tag{2.11}$$

From previous properties there is no condition to satisfy that the multiplicative inverse of each element is unique. So we couldn't make division operation. But the decoding algorithm of RS codes needs the division operation so this structure is not suitable to work over it.

2.1.3 Fields

A field F is a non-empty set of elements with two binary operations called addition and multiplication, denoted ' $+$ ' and ' $*$ ' respectively. For F to be a field the following conditions must be satisfied for all elements a, b, $c \in F$ [5]:

1- Closure :

$$\begin{aligned} c &= a + b, \\ d &= a * b. \end{aligned} \tag{2.12}$$

2- Associative :

$$\begin{aligned} a + (b + c) &= (a + b) + c, \\ a * (b * c) &= (a * b) * c. \end{aligned} \tag{2.13}$$

3- Identity :

$$\begin{aligned} \exists 1 \in F \, : \, \forall a \in F \, : \quad a * 1 &= 1 * a = a, \\ \exists 0 \in F \, : \, \forall a \in F \, : \quad a + 0 &= 0 + a = a. \end{aligned} \tag{2.14}$$

The element 0 is the identity element of addition, the element 1 is the identity element of multiplication

4- Inverse :

$$\begin{aligned} \forall a \in F \, : \, \exists b \in F \, : \, a + b &= 0. \\ \forall a \in F \, : \, \exists c \in F \, : \, a * c &= 1. \end{aligned} \tag{2.15}$$

Element b is called the additive inverse, $b = (-a)$, element c is called the multiplicative inverse, $c = a^{-1}(a \neq 0)$.

5- Commutative :

$$\begin{aligned} a + b &= b + a \\ a * b &= b * a \end{aligned} \tag{2.16}$$

6- Distributive :

$$(a + b) * c = a * c + b * c \tag{2.17}$$

It's clear from previous properties, the multiplicative inverse and additive inverse of each element are unique. So these properties enable the use of division and subtraction this is because the division can be performed as follows:

$$\forall a, \, b, \, c \in F \, : \, c = b/a = b * a^{-1} \tag{2.18}$$

Also the subtraction can be performed as follows:

$$\forall a,\, b,\, c \in F \,:\, c = b - a = b + (-a) \tag{2.19}$$

Also the number of elements in the field determines the order of it. So this structure is suitable for RS codes which needs addition, subtraction, multiplication, and division to decode it.

2.1.4 Galois Field GF(q)

Galois field GF(q) can be defined as a set of integers $\{0,\, 1,\, 2, ...,\, q-1\}$ where q is a prime, with modulo q addition and multiplication [6]. The name of Galois is in honour of Evariste Galois [7].

For example consider GF(2) $=\{0,\, 1\}$ is a finite field of order 2 under modulo 2 addition and multiplication.

Modulo 2 addition

+	0	1
0	0	1
1	1	0

Modulo 2 multiplication

*	0	1
0	0	0
1	0	1

From previous example it is clear that the additive inverse of zero is zero and also the additive inverse of one is one. This property is very important in GF(2) or in general in GF(2^m) and we can generalize this property by saying that the additive inverse of any element in GF(2^m) is itself. Because of this property we used GF(2^m) to represent RS codes as by using this algebraic structure we can represent each element in the field in binary form. So we can implement the encoder and decoder using regular digital circuits.

The previous field is known as a binary field which is used in binary error control coding. But in this thesis we use non-binary error control coding so we will introduce non-binary field which called GF(2^m).

2.1.5 The extended field GF(2^m)

In this subsection we introduce the properties of GF(2^m) and how to generate the field and how the arithmetic operation can be performed over it. Before introducing the properties of GF(2^m), we will see how to generate this field.

From the definition of Galois field GF(q), the number of elements are q and

the field must contain the identity element of addition and the identity element of multiplication which are zero and one respectively. So GF(2^m) has the same properties but GF(2^m) can not be generated like GF(q) as 2^m is not a prime number. So the field will be generated in a different way, we will define a new element α and the field will be:

$$\mathrm{GF}(2^m) = \left\{0,\ 1,\ \alpha,\ \alpha^2, \ldots,\ \alpha^{2^m-2}\right\} \tag{2.20}$$

By defining the field in this way, the multiplication can be performed easily. For example GF(2^3) = $\{0,\ 1,\ \alpha,\ \alpha^2, \ldots, \alpha^6\}$, the multiplication be performed as follows:

$$\begin{aligned} 0 * \alpha &= 0 \\ \alpha * \alpha &= \alpha^2 \\ \alpha^3 * \alpha^6 &= \alpha^9 = \alpha^{9 mod\, 7} = \alpha^2 \end{aligned}$$

Or in general the multiplication of any two elements in the field α^i and α^j belong to GF(2^m) can be defined as:

$$\alpha^i * \alpha^j = \alpha^k$$

where $k = (i+j) mod\, (2^m - 1)$. In other words $\alpha^k = \frac{\alpha^{i+j}}{\alpha^{2^m-1}}$, so this condition must be satisfied:

$$\alpha^{2^m-1} = 1 \tag{2.21}$$

In the following part we will explain how to represent the elements in binary form, hence addition can be performed and also satisfies equation 2.21.

Before introducing the binary representation of GF(2^m) some definitions should be introduced.

A primitive polynomial $p(x)$ of degree m over GF(2) is presented as

$$p(x) = p_0 + p_1 x + p_2 x^2 + ... + p_m x^m \tag{2.22}$$

where the coefficients $p_i \in$ GF(2).

The notion of an primitive polynomial is now introduced. But before introducing the primitive polynomial we will define the irreducible polynomial.

A polynomial $p(x)$ defined over GF(2), of degree m, is said to be *irreducible*,

if $p(x)$ can not be factorized to polynomials of degree higher than zero and lower than m [2].

An *irreducible* polynomial of degree m is a *primitive* polynomial if the smallest positive integer n for which $p(x)$ divides $x^n + 1$ is $n = 2^{m-1}$ [2].

So we write this equation,

$$x^n + 1 = p(x)q(x) \tag{2.23}$$

To generate the extended field GF(2^m), the element α is assumed to be a root of $p(x)$ which is a primitive, monic polynomial of degree m. By substituting by α in equation 2.23

$$\begin{aligned} \alpha^n + 1 &= p(\alpha)q(\alpha) = 0 \\ \alpha^n &= 1 \end{aligned} \tag{2.24}$$

So now equation 2.21 is satisfied. By this primitive polynomial the field elements can be represented in binary form.

For example, we want to represent the elements which belong to GF(2^3) in binary form and the primitive polynomial $p(x) = 1 + x + x^3$. So

$$\begin{aligned} p(\alpha) &= 1 + \alpha + \alpha^3 = 0 \\ \alpha^3 &= 1 + \alpha \\ \alpha^4 &= \alpha + \alpha^2 \\ \alpha^5 &= 1 + \alpha + \alpha^2 \\ \alpha^6 &= 1 + \alpha^2 \end{aligned}$$

So we can represent all elements in terms of 1, α, α^2 as follows:

Table 2.1: Binary representation of GF(2^m)

Field elements	Standard Basis 1, α, α^2
0	000
1	100
α	010
α^2	001
α^3	110
α^4	011
α^5	111
α^6	101

As shown in Table 2.1 the binary representation of elements depends on the primitive polynomial and the basis of the field [7]. We have a different types of basis to represent the field elements so we can change the field basis to enhance the performance of multiplier as addition can be performed simply using XOR gate. The primitive polynomial $p(x)$ for different values of m is shown in Appendix A.

2.1.6 Basis of field GF(2^m)

The Galois field GF(2^m) elements can be expressed as a linear combination of a set of m linearly independent elements $\gamma = \{\gamma_0, \gamma_1, ..., \gamma_{m-1}\}$ belong to GF(2^m) and these elements called a basis of GF(2^m) [8].

Let any element $a\in$ GF(2^m) can be represented as a linear combination of these basis elements over GF(2). That is

$$a = a_o\gamma_0 + a_1\gamma_1 + ... + a_{m-1}\gamma_{m-1} \qquad a_i \in \text{GF}(2). \tag{2.25}$$

Hence the field element a can be expressed by the binary vector (a_0, a_1, ..., a_{m-1}). The type of basis which is used to represent the field depends on the application as the speed is the important issue of some applications and the area is the important issue of other applications so now we will introduce the types of basis.

Standard Basis

Let a be an element in GF(2^m). In the standard basis the element can be represented as follows [8]:

$$a = a_0 + a_1\alpha + a_2\alpha^2 + \cdots + a_{m-1}\alpha^{m-1} \tag{2.26}$$

Where $a_i \in \mathrm{GF}(2)$.

For example consider $\mathrm{GF}(2^3)$ with a primitive polynomial $p(x) = x^3 + x + 1$, then $A = \{1, \alpha, \alpha^2\}$ forms the standard basis for this field and all 8 elements can be represented as

$$a = a_0 + a_1\alpha + a_2\alpha^2 \tag{2.27}$$

where the $a_i \in \mathrm{GF}(2)$. The relationship between the elements and the standard basis representation of $\mathrm{GF}(2^3)$ is shown in Table 2.1.

Dual Basis

A field element a can be expressed in dual basis as follows [8]:

$$a = a_0\lambda_0 + a_1\lambda_1 + a_2\lambda_2 + \cdots + a_{m-1}\lambda_{m-1} \tag{2.28}$$

Where $a_i \in \mathrm{GF}(2)$ is the trace of a, i.e. $a_i = Tr(a\alpha^i)$ and $\lambda_i \in \mathrm{GF}(2^m)$. The definition of the trace function is given below:

Let the trace of an element $\gamma \in \mathrm{GF}(2^m)$ be defined as follows:

$$Tr(\gamma) = \sum_{k=0}^{m-1} \gamma^{2^k} \tag{2.29}$$

Let two basis λ_k and μ_j are said to be dual to each other if:

$$Tr(\mu_j\lambda_k) = \begin{cases} 1 & if\ j = k \\ 0 & if\ j \neq k \end{cases} \tag{2.30}$$

For convenience, the basis μ_j is called the original basis and λ_k is called its dual basis. So we can convert from any basis to its dual basis using equation 2.30 which will be explained later.

Using The dual basis field element representation, an efficient and simple hardware implementation of a multiplier [9] can be implemented as it can be implemented using addition and shift operations only.

For example consider $\mathrm{GF}(2^4)$ with a primitive polynomial $p(x) = x^4 + x + 1$. Then $\{1, \alpha, \alpha^2, \alpha^3\}$ is the standard basis for the field. $\{1, \alpha^3, \alpha^2, \alpha\}$ forms the dual basis to the standard basis [9]. So the dual basis can be obtained from the standard basis with a simple linear transformation [10].

Normal Basis

The normal basis also can be used to represent an element $a \in \mathrm{GF}(2^m)$ as follows [8]:

$$a = a_0\gamma + a_1\gamma^2 + a_2\gamma^4 + \cdots + a_{m-1}\gamma^{2^{(m-1)}} \tag{2.31}$$

Where $a_i \in \mathrm{GF}(2)$ and $\gamma \in \mathrm{GF}(2^m)$.

The main advantage of using the normal basis is the easy implementation of squarer. Assume that $\left\{\alpha, \alpha^2, \alpha^4, \ldots, \alpha^{2^{(m-1)}}\right\}$is the normal basis of $\mathrm{GF}(2^m)$ so an element a can be represented as follows [8]:

$$a = a_0\alpha + a_1\alpha^2 + a_2\alpha^4 + \cdots + a_{m-1}\alpha^{2^{(m-1)}} \tag{2.32}$$

So the square of a can be written as follows:

$$a^2 = a_0\alpha^2 + a_1\alpha^4 + a_2\alpha^8 + \cdots + a_{m-1}\alpha^{2^m} \tag{2.33}$$

But by multiplying equation 2.21 by α, we can get

$$\alpha^{2^m} = \alpha \tag{2.34}$$

So equation 2.33 can be written as

$$a^2 = a_{m-1}\alpha + a_0\alpha^2 + a_1\alpha^4 + a_2\alpha^8 + \cdots + a_{m-2}\alpha^{2^{(m-1)}} \tag{2.35}$$

From equations 2.33 and 2.35 we can notice that a^2 is a cyclic shift of a.

This property is important as it allows for hardware efficient Massey-Omura multipliers to be designed [11]. The normal basis representation of $\mathrm{GF}(2^4)$ is given in Appendix B.

2.1.7 Basis Conversions

Conversion from one basis to another one is required to have a homogeneous system, as the communicating parts must have a common representation. But they may choose different representation when implementing finite field arithmetic, eg., data encryption and error control coding. To take the advantage of each basis representation, a system can be partitioned and an appropriate technique applied to each sub-system. Hence, the need for conversion circuits and techniques are necessary [8].

Standard Basis to Dual Basis Conversion

Let the primitive polynomial of GF(2^8) be

$p(x) = 1 + x + x^3 + x^5 + x^8$

From the definition of the trace function in equation 2.29 we get:

$$Tr(\alpha^k) = \sum_{i=0}^{7} (\alpha^k)^{2^i} \tag{2.36}$$

Where α is the root of $p(x)$.

In standard basis any element $Z \in$GF(2^8) can be written as :

$$Z = \sum_{k=0}^{7} z_k \alpha^k \tag{2.37}$$

Also can represent the element Z by a vector $[z_0, z_1, \ldots, z_7]$.

And in dual basis element Z can be represented as:

$$Z = \sum_{k=0}^{7} \grave{z}_k \lambda_k \tag{2.38}$$

Where λ_k and α^k are dual to each others. So from equations 2.30, 2.37, and 2.38 we can get $\grave{z}_k$ if we know z_k as follows:

$$\begin{aligned} \grave{z}_k &= Tr(Z\alpha^k) \\ &= Tr((z_0\alpha^0 + z_1\alpha + z_2\alpha^2 + z_3\alpha^3 \\ &\quad + z_4\alpha^4 + z_5\alpha^5 + z_6\alpha^6 + z_7\alpha^7)\alpha^k \\ &= z_0 Tr(\alpha^k) + z_1 Tr(\alpha^{k+1}) + z_2 Tr(\alpha^{k+2}) + z_3 Tr(\alpha^{k+3}) \\ &\quad + z_4 Tr(\alpha^{k+4}) + z_5 Tr(\alpha^{k+5}) + z_6 Tr(\alpha^{k+6}) + z_7 Tr(\alpha^{k+7}) \end{aligned} \tag{2.39}$$

From equation 2.40, the conversion from standard basis to dual basis is completed once $Tr(\alpha^i)$, for $0 \leq i \leq 14$ are known.

Dual Basis to Standard Basis Conversion

Also from equations 2.30, 2.37, and 2.38 we can get z_k if we know $\grave{z}_k$ as follows:

$$\begin{aligned} z_k &= Tr(Z\alpha^k) \\ &= Tr((\grave{z}_0\lambda_0 + \grave{z}_1\lambda_1 + \grave{z}_2\lambda_2 + \grave{z}_3\lambda_3 \\ &\quad +\grave{z}_4\lambda_4 + \grave{z}_5\lambda_5 + \grave{z}_6\lambda_6 + \grave{z}_7\lambda_7)\lambda_k \qquad (2.40) \\ &= \grave{z}_0 Tr(\lambda_0\lambda_k) + \grave{z}_1 Tr(\lambda_1\lambda_k) + \grave{z}_2 Tr(\lambda_2\lambda_k) + \grave{z}_3 Tr(\lambda_3\lambda_k) \\ &\quad +\grave{z}_4 Tr(\lambda_4\lambda_k) + \grave{z}_5 Tr(\lambda_5\lambda_k) + \grave{z}_6 Tr(\lambda_6\lambda_k) + \grave{z}_7 Tr(\lambda_7\lambda_k) \end{aligned}$$

If the trace values of equation 2.40 the basis conversion from dual to standard basis is completed.

Normal Basis to Standard Basis Conversion

Consider the element $Z \in$GF(2^m) and its representation in standard basis will be as follows:

$$Z = \begin{bmatrix} z_0 & z_1 & \dots & z_{m-1} \end{bmatrix} \begin{bmatrix} \alpha^0 \\ \alpha^1 \\ \vdots \\ \alpha^{m-1} \end{bmatrix} \qquad (2.41)$$

Also the element Z can be represented in normal basis as follows:

$$\grave{Z} = \begin{bmatrix} \grave{z}_0 & \grave{z}_1 & \dots & \grave{z}_{m-1} \end{bmatrix} \begin{bmatrix} \alpha \\ \alpha^2 \\ \vdots \\ \alpha^{2^{(m-1)}} \end{bmatrix} \qquad (2.42)$$

$\grave{Z}$ is simply an alternative representation of Z therefore $\grave{Z}$ can be converted from Z with a particular mapping known as a conversion matrix. Let C be the conversion matrix and can be written as follows:

$$\begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ z_{m-1} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1(m-1)} \\ c_{21} & c_{22} & \dots & c_{2(m-2)} \\ \vdots & \vdots & \ddots & \vdots \\ c_{(m-1)1} & c_{(m-1)2} & \dots & c_{(m-1)(m-1)} \end{bmatrix} \begin{bmatrix} \grave{z}_0 \\ \grave{z}_1 \\ \vdots \\ \grave{z}_{m-1} \end{bmatrix} \qquad (2.43)$$

Since $\{z_0, z_1, z_2, \dots, z_{m-1}\}$and $\{\grave{z}_0, \grave{z}_1, \grave{z}_2, \dots, \grave{z}_{m-1}\}$elements belong to GF(2)

therefore their addition can be implemented by an array of XOR gates.

For example, let us consider the case of an element in GF(2^4) in which $p(x) = 1+x+x^4$ is the primitive polynomial. In standard basis, let element A be expressed as

$$A = a_0\alpha^0 + a_1\alpha^1 + a_2\alpha^2 + a_3\alpha^3 \tag{2.44}$$

Similarly, $\grave{A}$ is an alternative representation of A in normal basis, so it can be represented as follows:

$$\grave{A} = \grave{a_0}\alpha^3 + \grave{a_1}\alpha^6 + \grave{a_2}\alpha^{12} + \grave{a_3}\alpha^9 \tag{2.45}$$

So by equating equation 2.44 by equation 2.45, we can get the conversion matrix as follows:

$$\begin{aligned} a_0\alpha^0 + a_1\alpha^1 + a_2\alpha^2 + a_3\alpha^3 &= \grave{a_0}\alpha^3 + \grave{a_1}\alpha^6 + \grave{a_2}\alpha^{12} + \grave{a_3}\alpha^9 \\ &= \grave{a_2}\alpha^0 + (\grave{a_3} + \grave{a_2})\alpha^1 + (\grave{a_1} + \grave{a_2})\alpha^2 + (\grave{a_0} + \grave{a_1} + \grave{a_2} + \grave{a_3})\alpha^3 \end{aligned} \tag{2.46}$$

So by equating the coefficient we can get the following set of equations:

$$\begin{aligned} a_0 &= \grave{a_2} \\ a_1 &= \grave{a_3} + \grave{a_2} \\ a_2 &= \grave{a_1} + \grave{a_2} \\ a_3 &= \grave{a_0} + \grave{a_1} + \grave{a_2} + \grave{a_3} \end{aligned} \tag{2.47}$$

So the conversion matrix can be represented as:

$$\begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \grave{a_0} \\ \grave{a_1} \\ \grave{a_2} \\ \grave{a_3} \end{bmatrix} \tag{2.48}$$

Therefore, $[A] = [C][B]$, where C is the conversion matrix.

Standard Basis to Dual Basis Conversion

The matrix C which is shown above describes the conversion from normal basis to standard basis. Therefore, to convert from standard basis to normal basis the inverse of matrix is needed which is C^{-1}. The matrix C^{-1} for previous example can be expressed as:

$$C^{-1} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \tag{2.49}$$

2.2 Finite Field Multipliers

The most important operations of finite field operations are multiplication and addition. Multiplication is considered to be much more complicated than addition as addition can be implemented easily using XOR gate. A huge effort has been exerted in research to reduce the hardware complexity and the latency of multipliers. In the following subsections Three types of multipliers will be explained and each of them is based on different basis.

2.2.1 Multiplication by a constant value

A multiplication by a constant value is used a lot either in encoding or in decoding process of RS codes. A multiplication by a constant value can be performed using two-variable operands multiplier of any types which will be described later. But it is usually efficiently to design a multiplier specifically for this task to save the area [2, 7].

Let a be an element in GF(2^m) and it can be expressed in standard basis as:

$$a = a_0 + a_1\alpha + ... + a_{m-1}\alpha^{m-1} \tag{2.50}$$

After multiplying a by α equation 2.50 can be written as

$$a * \alpha = a_0\alpha + a_1\alpha^2 + ... + a_{m-1}\alpha^m \tag{2.51}$$

But the primitive polynomial $p(x)$ can be written as follows:

$$p(x) = 1 + p_1x + p_2x^2 + \cdots + p_{m-1}x^{m-1} + x^m \tag{2.52}$$

Where $p_i \in \mathrm{GF}(2)$ and $p(\alpha) = 0$ so

$$p(\alpha) = 1 + p_1\alpha + p_2\alpha^2 + \cdots + p_{m-1}\alpha^{m-1} + \alpha^m = 0 \tag{2.53}$$

hence,

$$\alpha^m = 1 + p_1\alpha + p_2\alpha^2 + \cdots + p_{m-1}\alpha^{m-1} \tag{2.54}$$

So equation 2.51 can be written as follows:

$$a*\alpha = a_0\alpha + a_1\alpha^2 + ... + a_{m-2}\alpha^{m-1} + a_{m-1}(1 + p_1\alpha + p_2\alpha^2 + \cdots + p_{m-1}\alpha^{m-1}) \tag{2.55}$$

Which is equivalent to $a * \alpha \bmod p(\alpha)$.

For example consider, let $a \in \mathrm{GF}(2^4)$ and be presented in standard basis as,

$$a = a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3 \tag{2.56}$$

multiplication by α in $\mathrm{GF}(2^4)$, where the primitive polynomial $p(x) = x^4 + x + 1$. Then

$$a * \alpha = a_0\alpha + a_1\alpha^2 + a_2\alpha^3 + a_3\alpha^4 \tag{2.57}$$

hence,

$$a * \alpha = a_3 + (a_3 + a_0)\alpha + a_1\alpha^2 + a_2\alpha^3 \tag{2.58}$$

and this multiplication can be carried out with one XOR gate as shown in figure 2.6.

2.2.2 Two-Variable operands Multipliers

Finite field addition and multiplication can be performed serially or in parallel, that depends on that m bits representing field elements are processed in series or in parallel. Bit-serial multipliers generally require less area than bit-parallel multipliers, but also they usually need m clock cycles to generate the output rather than one. Hence in time critical applications bit-parallel multipliers are the best choice, in spite of the increased hardware overheads, so in this thesis we will focus on bit-parallel multipliers.

Bit-parallel Dual basis multipliers (PDBM)

The bit-parallel dual basis multiplier (PDBM) was presented in [9].

Let a, b, $c \in$GF(2^m), where a and c are presented in dual basis form as

$$a = a_0\lambda_0 + a_1\lambda_1 + \cdots + a_{m-1}\lambda_{m-1} \tag{2.59}$$
$$c = c_0\lambda_0 + c_1\lambda_1 + \cdots + c_{m-1}\lambda_{m-1} \tag{2.60}$$

where a_i, $c_i \in$ GF(2), and $\lambda_i \in$ GF(2^m), but b is presented in standard basis form as

$$b = b_0 + b_1\alpha + \cdots + b_{m-1}\alpha^{m-1} \tag{2.61}$$

where $b_i \in$ GF(2).

From the definition of trace function in equation 2.30 the coefficients a_i and c_i can be obtained as follows:

$$a_i = Tr(a\alpha^i) \tag{2.62}$$
$$c_i = Tr(c\alpha^i) \tag{2.63}$$

Where $i = 0, 1, \ldots, m-1$

Therefore the multiplication $c = a * b$ can be expressed in the matrix form [9].

$$\begin{bmatrix} a_0 & a_1 & \ldots & a_{m-1} \\ a_1 & a_2 & \ldots & a_m \\ \vdots & \vdots & \ddots & \vdots \\ a_{m-1} & a_m & \ldots & a_{2m-2} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{m-1} \end{bmatrix} = \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_{m-1} \end{bmatrix} \tag{2.64}$$

The coefficients a_i ($i = m, m+1, \ldots, 2m-1$) can be computed as follows:

$$a_{m+k} = Tr(a\alpha^{m+k}) = \sum_{j=0}^{m-1} p_j * a_{j+k} \tag{2.65}$$

Where the p_j are the coefficient of the primitive polynomial $p(x)$ and $k = 0, 1, m-2$.

Berlekamp used the above properties to design a bit serial-multiplier [12], and also the PDBM can be easily derived [9] as a circuit implementing the equation.

$$c_j = a_j b_0 + a_{j+1} b_1 + a_{j+2} b_2 + \ldots + a_{j+m-1} b_{m-1} \tag{2.66}$$

Where $j = 0, 1, \ldots, m-1$.

In general a PDBM for GF(2^m) contains module A that generates a_{m+i} ($i = 0, 1, \ldots, m-1$) from equation 2.66 and m modules B each of them generates the inner product of c_i by performing equation 2.66.

As an example, the PDBM for GF(2^3) using a primitive polynomial $p(x) = 1 + x + x^3$ is shown in figure 2.1 and modules A and B are shown in figures 2.2 and 2.3 respectively.

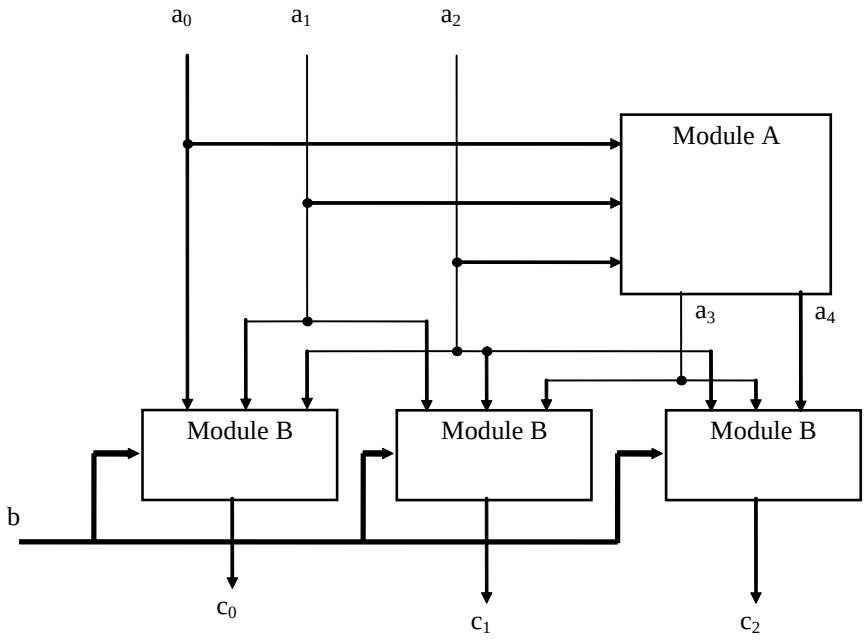

Figure 2.1: Bit-parallel dual basis multiplier for GF(2^3)

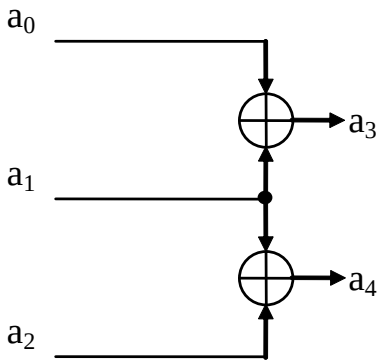

Figure 2.2: Type A module for a bit-parallel dual basis multiplier for GF(2^3)

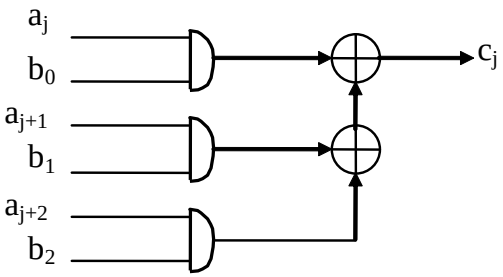

Figure 2.3: Type B module for a bit-parallel dual basis multiplier for GF(2^3)

From previous discussion it is clear that the polynomial basis multiplier is more efficient in serial form [12]. So we don't use it in this thesis.

Bit-parallel normal basis multipliers

A normal basis multiplier was proposed by Massey and Omura [11]. Massey-Omura multiplier [11] operates over the normal basis and so no basis converters are required as the two operands are expressed in normal basis. The idea of Massey-Omura multiplier is that the coefficient c_{i+1} of the output can be obtained by increasing the indices of the inner products of c_i by one, but from equation 2.68 we can notice that we the index of a_{m-1} and b_{m-1} is increased by one they are transformed to a_0 and b_0. So the same function of c_i can be used to generate the coefficient c_{i+1} after one cyclic shift. So with each cyclic shift a product bit is generated. Hence instead of m Boolean functions to generate the product, one Boolean function is required to generate all m product bits. Each Boolean function

needs $2m-1$ AND gates and $2m-2$ XOR gates and the latency of the multiplier is m clock cycles.

For an example, consider a Massey-Omura bit-serial multiplier for GF(2^4). Let a, b, and $c \in$ GF(2^4) and the primitive polynomial is $p(x) = 1 + x + x^4$ and let a normal basis for the field is $\{\alpha^3, \alpha^6, \alpha^{12}, \alpha^9\}$, where $\alpha^{24} = \alpha^9$. Let such that $c = a * b$ and these elements are represented over normal basis. Then

$$\begin{aligned} c &= c_0\alpha^3 + c_1\alpha^6 + c_2\alpha^{12} + c_3\alpha^9 \\ &= (a_0\alpha^3 + a_1\alpha^6 + a_2\alpha^{12} + a_3\alpha^9).(b_0\alpha^3 + b_1\alpha^6 + b_2\alpha^{12} + b_3\alpha^9) \end{aligned} \quad (2.67)$$

Where

$$\begin{aligned} c_0 &= a_0b_2 + a_1b_2 + a_1b_3 + a_2b_0 + a_2b_1 + a_3b_1 + a_3b_3 \\ c_1 &= a_1b_3 + a_2b_3 + a_2b_0 + a_3b_1 + a_3b_2 + a_0b_2 + a_0b_0 \\ c_2 &= a_2b_0 + a_3b_0 + a_3b_1 + a_0b_2 + a_0b_3 + a_1b_3 + a_1b_1 \\ c_3 &= a_3b_1 + a_0b_1 + a_0b_2 + a_1b_3 + a_1b_0 + a_2b_0 + a_2b_2 \end{aligned} \quad (2.68)$$

From equation 2.68 the Boolean function which is required to generate c_0 can be used to generate c_1, c_2, and c_3 by adding one to all the indices. .

So Massey-Omura multiplier is more efficient in serial architecture. A bit-parallel Massey-Omura multiplier (PMOM) requires at least $m(2m-2)$ 2-input XOR gates and $m(2m-1)$ 2-input AND gates [13]. Accordingly, this multiplier is more complex than the PDBM and is not used in this thesis.

Bit-parallel standard basis multipliers (PSBM)

Standard basis multipliers operate over the standard basis and require no basis converters (as the two operands use the same basis). These multipliers are easily implemented, reasonably hardware efficient. The bit-parallel standard basis multiplier (PSBM) was proposed by Laws [14].

Let a, b, $c \in$ GF(2^m) and can be represented in standard basis as follows:

$$\begin{aligned} a &= a_0 + a_1\alpha + \cdots + a_{m-1}\alpha^{m-1} \\ b &= b_0 + b_1\alpha + \cdots + b_{m-1}\alpha^{m-1} \\ c &= c_0 + c_1\alpha + \cdots + c_{m-1}\alpha^{m-1} \end{aligned} \tag{2.69}$$

The multiplication $c = a * b$ (where a, b, c are as given in equation 2.69) can be expressed

$$\begin{aligned} c &= a * b = (a_0 + a_1\alpha + \cdots + a_{m-1}\alpha^{m-1}) * b \\ c &= (\cdots(((a_0 b) + a_1 b\alpha) + a_2 b\alpha^2 + \cdots)\alpha + a_{m-1} b\alpha^{m-1} \end{aligned} \tag{2.70}$$

Now represent $b * \alpha^j$ as

$$b * \alpha^j = b_{j,0} + b_{j,1}\alpha + b_{j,2}\alpha^2 + \cdots + b_{j,m-1}\alpha^{m-1} \tag{2.71}$$

Therefore using 2.70 and 2.71

$$c_j = a_0 b_{0,j} + a_1 b_{1,j} + a_2 b_{2,j} + \cdots + a_{m-1} b_{m-1,j} \tag{2.72}$$

Using equations 2.71 and 2.72 it is possible to construct a modular and regular bit-parallel standers basis multiplier. A PPSM for GF(2^4) is presented in figures 2.4, 2.5, and 2.6.

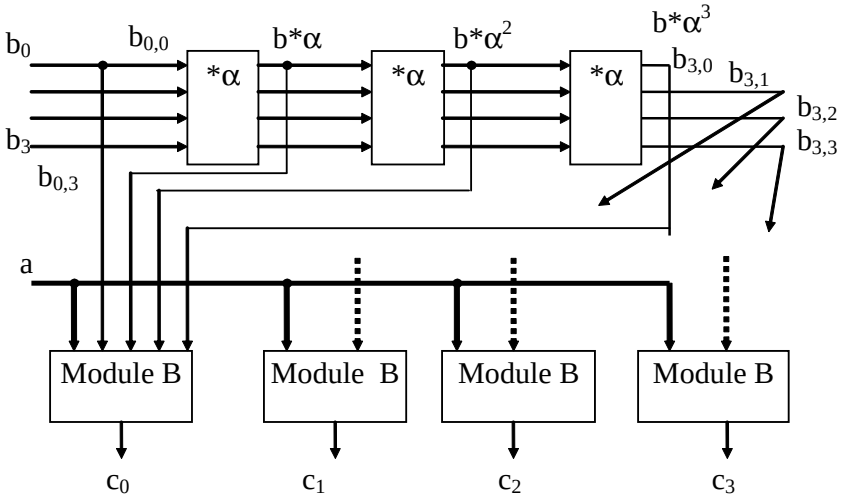

Figure 2.4: PPBM for GF(2^4).

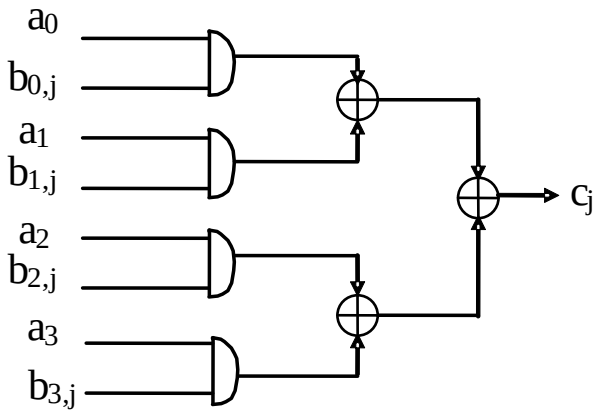

Figure 2.5: Module B of the PPBM

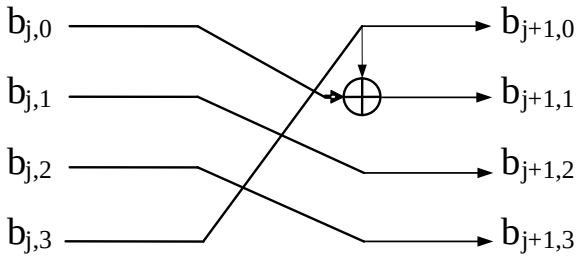

Figure 2.6: Circuit for multiplying by α in GF(2^4)

2.3 Introduction to Cyclic Codes

Cyclic codes are an important type of linear block codes, but they has an important properties which makes them easily implemented using sequential logic or shift registers and also by using these properties the hardware complexity can be reduced [2].

A linear block code is said to be cyclic if this condition is satisfied:

For any code vector of n components, $c = (c_0, c_1, ..., c_{n-1})$, a right-shift rotation of its components generates also a code vector [6, 15, 16]. If this right-shift rotation is done i times, a cyclically rotated version of the original vector is obtained as follows:

$$c^{(i)} = (c_{n-i}, c_{n-i+1}, \ldots, c_{n-1}, c_0, c_1, \ldots, c_{n-i-1})$$

Also, a linear block code means that, the sum of any two code vectors is also a code vector. As an example, the cyclic code $C_b(7, 4)$ described in Table 1.1.

2.3.1 Polynomial Representation of Codewords

Instead of vectors, Code vectors of a given cyclic code $C_{cyc}(n,\ k)$ can be expressed by polynomials. These polynomials are defined over a GF(2^m), and they can be defined over the binary field GF(2) as a special case. A polynomial representation $c(x)$ of a code vector $c = (c_0,\ c_1, ...,\ c_{n-1})$ is then of the form

$$c(x) = c_0 + c_1 x + \ldots + c_{n-1} x^{n-1} \tag{2.73}$$

Hence The ith-position right-shift rotation of a code vector c can be expressed as follows:

$$c^{(i)}(x) = c_{n-i} + c_{n-i+1} x + \cdots + c_{n-1} x^{i-1} + c_0 x^i + c_1 x^{i+1} + \cdots + c_{n-i-1} x^{n-1} \tag{2.74}$$

Dealing with these polynomials mathematically is much easier with dealing with vectors as they can be multiplied and divided easily.

2.3.2 Encoding of a Cyclic Code

The relationship between the ith-position right-shift rotated polynomial $c^{(i)}(x)$ and the original code polynomial $c(x)$ is of the form

$$x^i c(x) = q(x)(x^n + 1) + c^{(i)}(x) \tag{2.75}$$

hence,

$$c^{(i)}(x) = x^i c(x)\, mod\, (x^n + 1) \tag{2.76}$$

From all the code polynomials of a given cyclic code $C_{cyc}(n,\ k)$, there is a certain code polynomial of minimum degree [2]. Let this minimum degree equals to r, so that in its polynomial expression the coefficient of x^r will be exist. Therefore, this polynomial will be of the form $g(x) = g_0 + g_1 x + \cdots + x^r$.

The minimum degree polynomial is unique as if there is another polynomial with the same degree, this polynomial would be of the form $g_1(x) = g_{10} + g_{11} x + \cdots + x^r$. But because the cyclic code $C_{cyc}(n,\ k)$ is a linear block code, the sum of these two code polynomials should be a code polynomial, and this sum will result in a polynomial of degree $(r - 1)$, which contradicts the initial assumption that the minimum possible degree is r. Hence, the minimum-degree code polynomial

of a given cyclic code $C_{cyc}(n,\, k)$ is unique.

Also it is possible to prove that the minimum-degree polynomial of a given cyclic code $C_{cyc}(n,\, k)$ is monic polynomial, $g_0 = 1$. If the minimum degree polynomial has the form $g_2(x) = g_{21}x + g_{22}x^2 + \ldots + g_{2(r-1)}x^{r-1} + x^r$. So if we cyclically shift $g_2(x)$ by one place to the left we get another code polynomial $\grave{y}_2(x) = g_{21} + g_{22}x + \ldots + g_{2(r-1)}x^{r-1}$ with degree less than r and that contradicts the definition that $g_2(x)$ is the non-zero code polynomial of minimum degree.

Then, the expression for such a non-zero minimum-degree polynomial of a given cyclic code $C_{cyc}(n,\, k)$ is

$$g(x) = 1 + g_1 x + \cdots + g_{r-1}x^{r-1} + x^r \tag{2.77}$$

On the other hand, polynomials of the form $xg(x)$, $x^2g(x)$, ..., $x^{n-r-1}g(x)$ are equivalent to shift the polynomial $g(x)$ cyclically so they can be written as follows

$$\begin{aligned} xg(x) &= g^{(1)}(x) \\ x^2g(x) &= g^{(2)}(x) \\ &\vdots \\ x^{n-r-1}g(x) &= g^{(n-r-1)}(x) \end{aligned} \tag{2.78}$$

So these polynomials are also code polynomials. Since a cyclic code $C_{cyc}(n,\, k)$ is also a linear block code, linear combinations of code polynomials are also code polynomials, and therefore

$$\begin{aligned} c(x) &= m_0 g(x) + m_1 x g(x) + \cdots + m_{n-r-1}x^{n-r-1}g(x) \\ c(x) &= (m_0 + m_1 x + \cdots + m_{n-r-1}x^{n-r-1})g(x) \end{aligned} \tag{2.79}$$

Equation 2.79 determines that any code polynomial $c(x)$ is a multiple of the minimum-degree polynomial $g(x)$. This property is the most important for the encoding and decoding of a cyclic code $C_{cyc}(n,\, k)$. For simplicity we will assume that the coefficients m_i , $i = 0,\, 1,\, 2,\, ...,\, n - r - 1$, in equation 2.79 belong to GF(2). Then there will be 2^{n-r} code polynomials of degree $n - 1$ or less that are multiples of $g(x)$. But in $C_{cyc}(n,\, k)$ there is k bits message so we have 2^kpossible linear combinations of message. For a dual assignment between the message and the coded vector spaces, there should be 2^k possible linear combinations of code polynomials. Therefore, $2^{n-r} = 2^k$ or $r = n - k$. So in general the degree of. The

minimum degree polynomial equals to the number of redundant symbols which are added to the message in encoding process. The minimum-degree polynomial is then of the form

$$g(x) = 1 + g_1 x + \cdots g_{n-k-1} x^{n-k-1} + x^{n-k} \tag{2.80}$$

Also there another important property for $g(x)$ this property is that the generator polynomial $g(x)$ is factor from $x^n + 1$ to prove that we will use the following procedure; multiply $g(x)$ by x^k so the polynomial $x^k g(x)$ has degree of n then divide $x^k g(x)$ over $x^n + 1$ so we will have the following

$$x^k g(x) = (x^n + 1) + g^{(k)}(x) \tag{2.81}$$

where $g^{(k)}(x)$ is kth cyclic shift of $g(x)$ so it is a code polynomial and also it is multiple of $g(x)$ so we can rewrite equation 2.81 as follows

$$\begin{aligned} x^k g(x) &= (x^n + 1) + a(x)g(x) \\ (x^k + a(x))g(x) &= x^n + 1 \end{aligned} \tag{2.82}$$

from equation 2.82 we can say $g(x)$ is factor of $x^n + 1$.

Summarizing, in a linear cyclic code $C_{cyc}(n,\ k)$, there is a unique minimum-degree code polynomial, and any other code polynomial is a multiple of this polynomial. The non-zero minimum-degree polynomial is of degree $n - k$, and any other code polynomial of the linear cyclic code $C_{cyc}(n,\ k)$ is of degree $n-1$ or less, and so

$$c(x) = m(x)g(x) = (m_0 + m_1 x + \cdots + m_{k-1} x^{k-1})g(x) \tag{2.83}$$

where m_i, $i = 0,\ 1,\ 2, ...,\ k - 1$, are the symbols of the message vector. Since the minimum-degree code polynomial generates the linear cyclic code $C_{cyc}(n,\ k)$, so it is called the generator polynomial.

2.3.3 Encoding of Cyclic Codes in Systematic Form

the encoding process for a linear cyclic code $C_{cyc}(n,\ k)$ has been introduced in equation 2.83 as a multiplication between the message polynomial $m(x)$ and the generator polynomial $g(x)$, and this operation is suitable to generate any code polynomial of the code, but this type of encoding has a big problem. This prob-

lem is summarized in that the message could not be obtained directly from the code polynomial as the code polynomial should be divided over the generator $g(x)$ to get the message $m(x)$ and that makes the decoding process more complicated. So we encode to make the message part of code polynomial and this encoding procedure is called encoding in systematic form. Given a polynomial that satisfies the conditions to be the generator polynomial $g(x)$ of a linear cyclic code $C_{cyc}(n,\, k)$, and if the message polynomial is of the form

$$m(x) = m_0 + m_1 x + \cdots + m_{k-1} x^{k-1} \tag{2.84}$$

then the systematic version of the linear cyclic code $C_{cyc}(n,\, k)$ can be obtained by performing the following operations [2, 15, 16]:

The polynomial $x^{n-k} m(x) = m_0 x^{n-k} + m_1 x^{n-k} + \cdots + m_{k-1} x^{n-1}$ is first formed, and then divided by the generator polynomial $g(x)$:

$$x^{n-k} m(x) = a(x) g(x) + p(x) \tag{2.85}$$

Here $p(x)$ is the remainder polynomial of the division of equation 2.85, which has degree $n - k - 1$ or less, since the degree of $g(x)$ is $n - k$. By rearranging equation 2.85, we obtain $x^{n-k} m(x) + p(x) = a(x) g(x)$ where that can be noticed the polynomial $x^{n-k} m(x) + p(x)$ is a code polynomial because it is a multiple of $g(x)$. In this polynomial, the term $x^{n-k} m(x)$ represents the message polynomial but it is right shifted by $n - k$ positions, and $p(x)$ is the remainder polynomial of this division and acts as the redundancy polynomial, and reserving the lower degree terms of the code polynomial $c(x)$. This procedure achieves the objective of encoding in the systematic form

$$\begin{aligned} c(x) \;=\; & x^{n-k} m(x) + p(x) = p_0 + p_1 x + \cdots + p_{n-k-1} x^{n-k-1} + \\ & m_0 x^{n-k} + m_1 x^{n-k+1} + \cdots + m_{k-1} x^{n-1} \end{aligned} \tag{2.86}$$

that when expressed as a code vector is equal to

$$c = (p_0,\, p_1,\, \ldots,\, p_{n-k-1},\, m_0,\, m_1,\, \ldots\, m_{k-1}) \tag{2.87}$$

2.4 Encoding of Reed Solomon Code

RS codes are a type of linear, non-binary, cyclic block codes [3]. This type is a subfamily of the family of the linear, non-binary, cyclic BCH (Bose, Chaudhuri [17], Hocquenghem [18]) codes which are working over the GF(q). Here q is a power of a prime number p_{prime}, $q = p_{prime}^m$, where m is a positive integer. These codes are different from binary codes, which work over GF(2). This is why these codes are also called non-binary codes. All the concepts and properties valid for binary codes are also valid for these non-binary codes.

An RS code $C_{RS}(n, k)$ able to correct any error pattern of size t or less is defined over the Galois field GF(2^m), and it has parameters as shown in Table 2.2.

Table 2.2: RS codes parameters

Code length	$n = 2^m - 1$
Number of parity check elements	$n - k = 2t$
Minimum distance	d_{min}= $2t + 1$
Error-correction capability	t element errors per code vector

An RS code $C_{RS}(n, k)$ of length n and dimension k is the linear, cyclic, block RS code generated by the polynomial

$$\begin{aligned} g(x) &= (x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{n-k}) \\ &= (x-\alpha)(x-\alpha^2)\cdots(x-\alpha^{2t}) \\ &= g_0 + g_1 x + g_2 x^2 + \cdots + g_{2t-1}x^{2t-1} + g_{2t}x^{2t} \end{aligned} \tag{2.88}$$

An RS code can be equivalently defined as the set of code polynomials $c(x)$ over GF(2^m) of degree $\deg\{c(x)\} \leq n-1$ that have α, $\alpha^2, \ldots, \alpha^{n-k}$ as their roots [17]. Therefore $c(x) \in C_{RS}$ if and only if

$$c(\alpha) = c(\alpha^2) = c(\alpha^3) = \cdots = c(\alpha^{2t}) = 0 \,\text{where}\, \deg\{c(x)\} \leq n-1 \tag{2.89}$$

2.4.1 RS Codes in Systematic Form

An RS code generated by a given generator polynomial of the form of 2.88 is a linear and cyclic block RS code $C_{RS}(n, n-2t)$ consisting of code polynomials $c(x)$ of degree $n-1$ or less. All these polynomials coefficients belong to GF(2^m). Code

polynomials are multiples of the generator polynomial $g(x)$, thus containing all its roots. A message polynomial is of the form

$$m(x) = m_0 + m_1 x + \cdots + m_{k-1} x^{k-1} \tag{2.90}$$

This message polynomial is also formed with coefficients that are elements of GF(2^m). The systematic form for these codes is obtained in the same way as for binary cyclic codes, that is, by obtaining the remainder $p(x)$ of the division of $x^{n-k} m(x)$ by $g(x)$ as stated in equation 2.85

2.4.2 Implementation of RS Encoder

In hardware implementation, equation 2.86 is accomplished by using division circuit as shown in figure 2.7. As soon as the message $m(x)$ has entered to the circuit, the parity-check symbols are in the register.

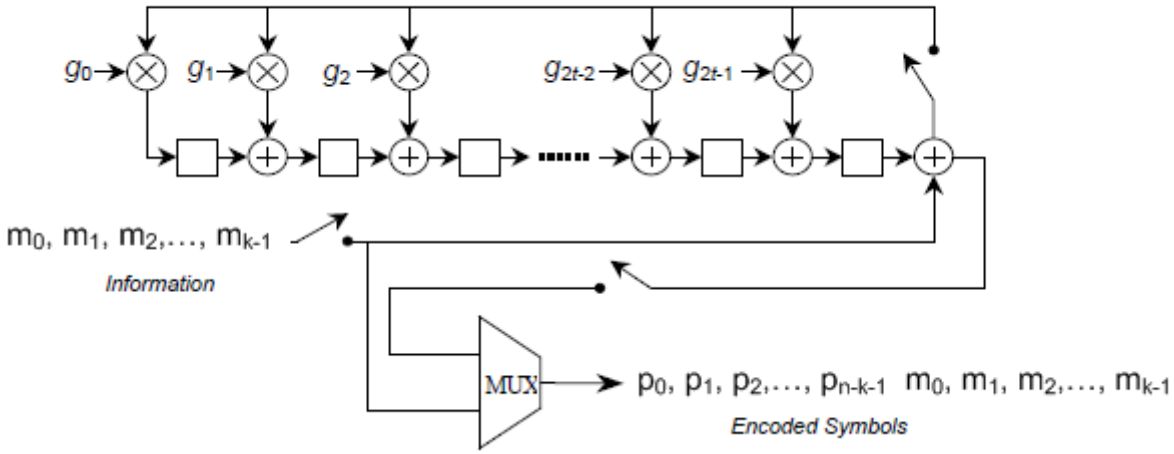

Figure 2.7: The LFSR architecture of RS encoder.

Chapter 3

REED-SOLOMON DECODER

When a received codeword (code vector) is fed to the RS decoder at the receiver for processing, the decoder first tries to verify that is this codeword valid codewords or not. If it dose not, errors must have occurred during transmission over a communication channel. This part of the decoder processing is called error detection. If errors are detected, the decoder attempts to correct it. This called error correction.

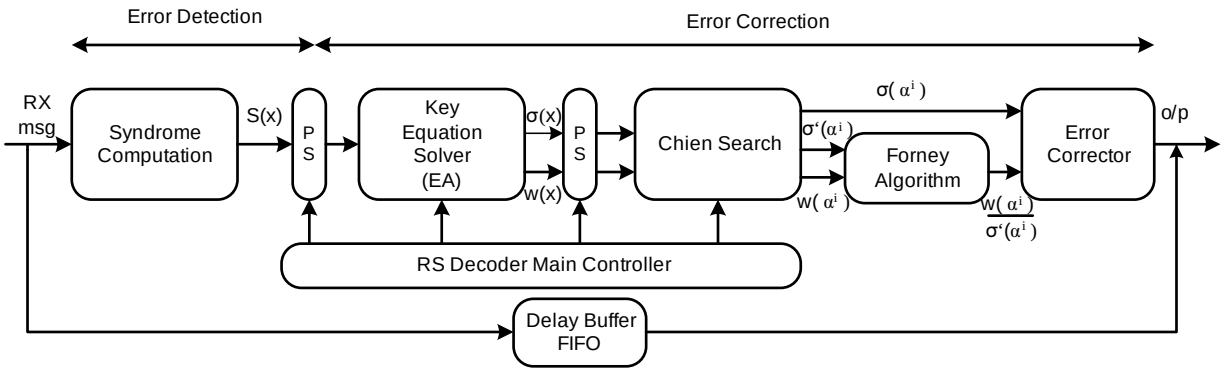

Figure 3.1: Block Diagram of RS Decoder

Figure 3.1 shows the basic block diagram of a decoder for RS codes. The decoder consists of digital circuits and processing elements to accomplish the following tasks:

1- Syndrome computation used for error detection.

2- Key equation solver used to find the coefficients of error-location polynomial $\sigma(x)$ and error-evaluation polynomial $W(x)$.

3- Find the roots of the error location polynomial $\sigma(x)$ using Chien search algorithm.

4- Find the error values using Forney algorithm.

5-Correct the received codeword with the error locations and values found.

3.1 Syndrome Calculation

The syndrome calculation is the first step in RS decoding process. The function of syndrome block is to detect if there are any errors in the received codeword. Assume that the transmitted codeword polynomial $c(x)$ is :

$$c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1} \tag{3.1}$$

This codeword is transmitted and affected by noise in channel, and modeled as a received polynomial $r(x)$:

$$r(x) = r_0 + r_1 x + \cdots + r_{n-1} x^{n-1} \tag{3.2}$$

which is related to the error polynomial $e(x)$ and the codeword polynomial $c(x)$ as follows:

$$r(x) = c(x) + e(x) \tag{3.3}$$

where the error pattern $e(x)$ added by the channel is expressed as:

$$e(x) = e_0 + e_1 x + \ldots + e_{n-1} x^{n-1} \tag{3.4}$$

where $e_i = r_i - c_i$ is a symbol form GF(2^m).

From equation 2.79 that can be noticed every valid codeword polynomial $c(x)$ is a multiple of the generator polynomial $g(x)$. Therefore, the roots of $g(x)$ must also be the roots of $c(x)$. From equation 3.5, the received polynomial $r(x)$ evaluated at each of the roots of $g(x)$ should equals to zero only when it is a valid codeword. Any errors will result in one or more of the computations yielding a non-zero result. So the computation of syndrome symbol can be described as follows:

$$S_i = r(\alpha^i),\ i = 1, 2, \ldots, 2t \tag{3.5}$$

where $\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{2t}$ are the roots of $g(x)$. If $r(x)$ was a valid codeword,

it would cause each syndrome symbol S_i to equal to zero, or, if one or more syndromes are non-zero, errors have been detected.

3.2 The Decoding Algorithm

The relation between the code polynomial $c(x)$, received polynomial $r(x)$, and the error polynomial $e(x)$ can be stated as follows [2]:

$$c(x) = r(x) + e(x) \tag{3.6}$$

All these polynomials are defined with coefficients over GF(2^m). Let us assume that the error vector contains τ non-zero elements, representing an error pattern of τ errors placed at positions x^{j_1}, $x^{j_2}, \ldots$, x^{j_τ} , where $0 \leq j_1 < j_2 < \cdots < j_\tau \leq n-1$. The error-location number is then defined as

$$\beta_l = \alpha^{j_l} \tag{3.7}$$

where $l = 1, 2, 3, \ldots, \tau$.

The syndrome vector components are calculated, as was stated in equation 3.5, by replacing the variable x in the received polynomial $r(x)$ with the roots α^i, $i = 1, 2, \ldots, 2t$. It is then true that

$$s_i = r(\alpha^i) = c(\alpha^i) + e(\alpha^i) = e(\alpha^i) \tag{3.8}$$

Thus, a system of $2t$ equations can be formed as follows:

$$\begin{aligned} s_1 &= r(\alpha) = e(\alpha) = e_{j_1}\beta_1 + e_{j_2}\beta_2 + \cdots + e_{j_\tau}\beta_\tau \\ s_2 &= r(\alpha^2) = e(\alpha^2) = e_{j_1}\beta_1^2 + e_{j_2}\beta_2^2 + \cdots + e_{j_\tau}\beta_\tau^2 \\ &\vdots \\ s_{2t} &= r(\alpha^{2t}) = e(\alpha^{2t}) = e_{j_1}\beta_1^{2t} + e_{j_2}\beta_1^{2t} + \cdots + e_{j_\tau}\beta_\tau^{2t} \end{aligned} \tag{3.9}$$

So now we have 2τ unknowns which are $\beta_1, \beta_2, \ldots, \beta_\tau, e_{j_1}, e_{j_2}, \ldots, e_{j_\tau}$ and at worst case $\tau = t$. These unkowns are the positions and the values of errors. So any algorithm solves the set of equations 3.9 can be considered as a decoding algorithm for RS codes.

But In order to decode RS code, the following polynomials will be defined:

The error-location polynomial $\sigma(x)$ is defined as

$$\begin{aligned}\sigma(x) &= (x-\alpha^{-j_1})(x-\alpha^{-j_2})\cdots(x-\alpha^{-j_\tau}) = \prod_{l=1}^{\tau}(x-\alpha^{-j_l}) \\ &= \sigma_0+\sigma_1 x+\cdots+\sigma_\tau x^\tau \end{aligned} \tag{3.10}$$

and the error-evaluation polynomial $W(x)$ also can be defined as

$$\begin{aligned}W(x) &= \sum_{l=1}^{\tau} e_{j_l} \prod_{i=1, i\neq l}^{\tau}(x-\alpha^{-ji}) \\ &= W_0+W_1 x+\cdots+W_{\tau-1}x^{\tau-1} \end{aligned} \tag{3.11}$$

The error values can be calculated from

$$e_{j_l} = \frac{W(\alpha^{j_l})}{\grave{\sigma}(\alpha^{j_l})} \tag{3.12}$$

where $\grave{\sigma}(x)$ is the first derivative of the polynomial $\sigma(x)$ with respect to x. Polynomials $\sigma(x)$ and $W(x)$ are relative prime, since from the way they are defined, they do not have common roots.

By using polynomials $\sigma(x)$ and $W(x)$ the algorithm can calculate the positions of the errors and their values by using equation 3.12. Also, the syndrome polynomial of degree $2t-1$ is defined as

$$S(x) = s_1 + s_2 x + s_3 x^2 + \cdots + s_{2t}x^{2t-1} = \sum_{j=0}^{2t-1} s_{j+1}x^j \tag{3.13}$$

If $S(x) = 0$, the received polynomial is a code polynomial or contains an undetectable error pattern.

3.2.1 The Key Equation

The key equation is that equation relates polynomials $\sigma(x)$, $S(x)$, and $W(x)$. The solution of this equation is a decoding algorithm for a RS code [2]. This equation can be written as follows:

$$\sigma(x)S(x) = -W(x) + \mu(x)x^{2t} \tag{3.14}$$

Also it can be written like that

$$\{\sigma(x)S(x) + W(x)\} \bmod (x^{2t}) = 0 \tag{3.15}$$

where $\mu(x)$ is an auxiliary polynomial.

This key equation can be proved as follows:

By rewriting the syndrome polynomial $S(x)$,

$$S(x) = \sum_{j=0}^{2t-1} s_{j+1}x^j = \sum_{j=0}^{2t-1}(\sum_{i=1}^{\tau} e_{ji}\alpha^{ji(j+1)})x^j = \sum_{i=1}^{\tau} e_{ji}\alpha^{ji}\sum_{j=0}^{2t-1}(\alpha^{ji}x)^j$$

$$S(x) = \sum_{i=1}^{\tau} e_{ji}\alpha^{ji}\frac{(\alpha^{ji}x)^{2t}-1}{(\alpha^{ji}x)-1} = \sum_{i=1}^{\tau} e_{ji}\frac{(\alpha^{ji}x)^{2t}-1}{x-\alpha^{-ji}} \tag{3.16}$$

and then

$$\begin{aligned}\sigma(x)S(x) &= \sum_{i=1}^{\tau} e_{ji}\sigma^{(\mu)}(x)\frac{(\alpha^{ji}x)^{2t}-1}{x-\alpha^{-ji}}\prod_{l=1}^{\tau}(x-\alpha^{-jl}) \\ &= \sum_{i=1}^{\tau} e_{ji}\left[(\alpha^{ji}x)^{2t}-1\right]\prod_{i=1,i\neq l}^{\tau}(x-\alpha^{-jl})\end{aligned}$$

$$= \sum_{i=1}^{\tau} e_{ji}\prod_{i=1,i\neq l}^{\tau}(x-\alpha^{-jl}) + \left[\sum_{i=1}^{\tau} e_{ji}\alpha^{ji(2t)}\prod_{i=1,i\neq l}^{\tau}(x-\alpha^{-jl})\right] \tag{3.17}$$

$$= -W(x) + \mu(x)x^{2t}$$

For convenience some algorithms deals with equation 3.17 in the following form

$$(S(x)\sigma(x) + W(x)) \bmod x^{2t} = 0 \tag{3.18}$$

Many algorithms are used, to solve this equation, Euclidean algorithm, Berlerkamp-Massey algorithm, and PGZ algorithm which will be described in the following sections.

3.2.2 Decoding of RS Codes Using the Euclidean Algorithm

The Euclidean algorithm is an algorithm that calculates the greatest common divisor (GCD) r of any two numbers a and b, $r = \mathrm{GCD}(a, b)$ [2]. It also gets two integer numbers, or for our application, two polynomials s and t , such that

$$r = sa + tb \tag{3.19}$$

Where s, and t are two polynomials are calculated by the algorithm to get the

GCD. So Euclidean algorithm is suitabel for solving the key equation 3.18, but we will rewrite this equation to be suitable for the algorithm

$$-\mu(x)x^{2t} + \sigma(x)S(x) = -W(x) \tag{3.20}$$

So we can say that $W(x) = \text{GCD}(S(x), x^{2t})$, $s = \mu(x)$, and $t = \sigma(x)$.

Now we will write the algorithm in recursive form to be easy implemented.

Let there are two polynomials a and b such that $\deg(a) \geq \deg(b)$ and we want to apply Euclidean algorithm on them to get the GCD r and s and t polynomials to saitsfy the following equation 3.19.

Initialization:

$r_{-1} = a$ and $r_0 = b$, where the value r_i is obtained as the remainder of the division of r_{i-2} by r_{i-1}

$$r_{i-2} = q_i r_{i-1} + r_i \tag{3.21}$$

where $\deg(r_i) < \deg(r_{i-1})$.

By rearranging equation 3.21

$$r_i = r_{i-2} - q_i r_{i-1} \tag{3.22}$$

The algorithm usually tries to satisfy the following equation

$$r_i = s_i a + t_i b \tag{3.23}$$

where s_i and t_i represents s and t polynomials in ith step.

The recursion 3.22 is also valid for coefficients s_i and t_i:

$$s_i = s_{i-2} - q_i s_{i-1} \tag{3.24}$$

$$t_i = t_{i-2} - q_i t_{i-1} \tag{3.25}$$

So initially

$$r_{-1} = a = (1)a + (0)b \tag{3.26}$$

$$r_0 = b = (0)a + (1)b \tag{3.27}$$

and the initial conditions are

$$s_{-1} = 1, \qquad t_{-1} = 0$$

By applying Euclidean algorithm on the key equation 3.20.

The given polynomials are x^{2t} and $S(x)$, and the ith recursion is of the form

$$r_i(x) = s_i(x)x^{2t} + t_i(x)S(x) \tag{3.28}$$

Multiplying equation 3.28 by a constant $\lambda \in \mathrm{GF}(2^m)$, we obtain

$$\begin{aligned} \lambda r_i(x) &= \lambda s_i(x)x^{2t} + \lambda t_i(x)S(x) \\ &= -W(x) = -\mu(x)x^{2t} + \sigma(x)S(x) \end{aligned} \tag{3.29}$$

where

$$\deg\left(r_i(x)\right) < t \tag{3.30}$$

Thus,

$$W(x) = -\lambda r_i(x)$$

$$\sigma(x) = \lambda t_i(x) \tag{3.31}$$

where λ is a constant that makes the resulting polynomial be a monic polynomial.

3.2.3 Decoding of RS Codes Using Berlekamp-Massey Algorithm

The Berlekamp-Massey (B-M) algorithm [2], [4] is another algorithm to decode RS and BCH codes. First, this algorithm assumes binary errors. Then it gets the error location polynomial as this polynomial depends only on the error locations so there is no need to take the error values into consideration. So equations 3.9 will be modified to

$$\begin{aligned} s_1 &= r(\alpha) = e(\alpha) = \beta_1 + \beta_2 + \cdots + \beta_\tau \\ s_2 &= r(\alpha^2) = e(\alpha^2) = \beta_1^2 + \beta_2^2 + \cdots + \beta_\tau^2 \\ &\vdots \\ s_{2t} &= r(\alpha^{2t}) = e(\alpha^{2t}) = \beta_1^{2t} + \beta_2^{2t} + \cdots + \beta_\tau^{2t} \end{aligned} \tag{3.32}$$

The error location polynomial has a different definition with respect to expression 3.10, which is the following:

$$\begin{aligned} \sigma_{BM}(x) &= (1 - \beta_1 x)(1 - \beta_2 x) \cdots (1 - \beta_\tau x) \\ &= \sigma_0 + \sigma_1 x + \cdots + \sigma_\tau x^\tau \end{aligned} \tag{3.33}$$

The roots of this polynomial are $\beta_1^{-1}, \beta_2^{-1}, \ldots, \beta_\tau^{-1}$. This modified definition is more suitable for the description of the B-M algorithm.

From equation 3.33 the coefficients of the error location polynomial $\sigma(x)$ can be written as:

$$\begin{aligned} \sigma_0 &= 1 \\ \sigma_1 &= \beta_1 + \beta_2 + \cdots + \beta_\tau \\ \sigma_2 &= \beta_1\beta_2 + \beta_2\beta_3 + \cdots + \beta_\tau\beta_{\tau-1} \\ &\vdots \\ \sigma_\tau &= \beta_1\beta_2 \ldots \beta_\tau \end{aligned} \tag{3.34}$$

The set of equations 3.34 can be related to the equations 3.32 as follows:

$$\begin{aligned} s_1 &+ \sigma_1 = 0 \\ s_2 &+ \sigma_1 s_1 = 0 \\ s_3 &+ \sigma_1 s_2 + \sigma_2 s_1 + \sigma_3 = 0 \\ &\vdots \\ s_\tau &+ \sigma_1 s_\tau + \cdots + \sigma_{\tau-1} s_2 + \sigma_\tau s_1 = 0 \end{aligned} \tag{3.35}$$

These equations are called Newton identities [17]. Thus, for example,

$$s_2 + \sigma_1 s_1 = (\beta_1)^2 + (\beta_2)^2 + \cdots + (\beta_\tau)^2 + (\beta_1 + \beta_2 + \cdots + \beta_\tau)(\beta_1 + \beta_2 + \cdots + \beta_\tau) = 0$$

since in GF(2^m) the products $\beta_i\beta_j + \beta_j\beta_i = 0$. The remaining Newton identities can be derived in the same way.

B-M algorithm gets the error location polynomial $\sigma(x)$ first, then it subtitutes by it in the key equation to get the error evaluator polynomial $W(x)$.

The Error-Location Polynomial $\sigma(x)$

The B-M algorithm mainly finds the coefficients of the error-location polynomial, $\sigma_1, \sigma_2, \ldots, \sigma_\tau$, then substitutes by these coefficients in the key equation to find the coefficients of error evaluation polynomial. The B-M algorithm uses the syndrome vector components $S = (s_1, s_2, \ldots, s_{2t})$ to find the coefficients $\sigma_1, \sigma_2, \ldots, \sigma_\tau$ of the error-location polynomial, whose roots are the inverses of the error-location numbers $\beta_1, \beta_2, \ldots, \beta_\tau$. The error values at these positions should be calculated, to correct the error. As said above, the core of the B-M algorithm is an iterative method for calculating the error-location polynomial coefficients $\sigma_1, \sigma_2, \ldots, \sigma_\tau$ then it gets the error evaluation polynomial coefficients $W_0, W_1, \ldots, W_{\tau-1}$ directly from the key equation.

The algorithm mainly has two steps as follows [2]:

The first step is calculating a minimum-degree polynomial $\sigma_{BM}^{(\mu)}(x)$ that satisfies the μth Newton identity described in equation 3.35. The second step is substituting by $\sigma_{BM}^{(\mu)}(x)$ in the $(\mu+1)$th Newton identity in equation 3.35. If this polynomial satisfies the $(\mu+1)$th Newton identity, then $\sigma_{BM}^{(\mu)}(x) = \sigma_{BM}^{(\mu+1)}(x)$. Oth-

erwise the decoding algorithm adds a correction term which is called discrepancy d_μ to $\sigma_{BM}^{(\mu)}(x)$ to get the polynomial $\sigma_{BM}^{(\mu+1)}(x)$, which satisfies the first μ Newton identities.

After $2t$ iteration steps the algorithm is halted and the final version of error location polynomial $\sigma_{BM}(x)$ equals to $2t$ th iteration polynomial $\sigma_{BM}^{(2t)}(x)$.

Since the last polynomial satisfies the whole set of Newton identities described in equation 3.35. This algorithm can be implemented in iterative form. The minimum-degree polynomial obtained in the μth iteration and satisfies the first μ Newton identities, is $\sigma_{BM}^{(\mu)}(x)$, and can be written in the following form

$$\sigma_{BM}^{(\mu)}(x) = 1 + \sigma_1^{(\mu)}x + \sigma_2^{(\mu)}x^2 + \cdots + \sigma_{l_\mu}^{(\mu)}x^{l\mu} \tag{3.36}$$

where l_μ is the degree of the polynomial $\sigma_{BM}^{(\mu)}(x)$. The μth discrepancy d_μ can be obtained by using the following equation [2]:

$$d_\mu = s_{\mu+1} + \sigma_1^{(\mu)}s_\mu + \sigma_2^{(\mu)}s_{\mu-1} + \cdots + \sigma_{l_\mu}^{(\mu)}s_{\mu+1-l_\mu} \tag{3.37}$$

If the discrepancy d_μ does not equal to zero, $d_\mu \neq 0$, the minimum-degree polynomial $\sigma_{BM}^{(\mu)}(x)$ does not satisfy the $(\mu+1)$th Newton identity, and the discrepancy d_μ is used to get $\sigma_{BM}^{(\mu+1)}(x)$ which satisfies the $(\mu+1)$th Newton identity. In the calculation of $\sigma_{BM}^{(\mu+1)}(x)$, the algorithm comes back to a previous step ρ of the iteration, with respect to μ, such that its discrepancy d_ρ does not equal to zero, $d_\rho \neq 0$ and $\rho - l_\rho$ is a maximum. Where the number l_ρ is the degree of the polynomial $\sigma_{BM}^{(\rho)}(x)$.

Then

$$\sigma_{BM}^{(\mu+1)}(x) = \sigma_{BM}^{(\mu)}(x) + d_\mu d_\rho^{-1}x^{(\mu-\rho)}\sigma_{BM}^{(\rho)}(x) \tag{3.38}$$

The B-M algorithm can be stated in the form of a table, as given in Table 3.1.

Table 3.1: B-M algorithm table for determining the error-location polynomial

μ	$\sigma_{BM}^{(\mu)}$	d_μ	l_μ	$\mu - l_\mu$
-1	1	1	0	-1
0	1	s_1	0	0
1				
$\vdots$				
$2t$				

So calculation of minimum-degree polynomial $\sigma_{BM}^{(\mu+1)}(x)$ in iteration $\mu+1$ can be summarized as follows:

$$\text{If } d_\mu = 0 \text{ then } \sigma_{BM}^{(\mu+1)}(x) = \sigma_{BM}^{(\mu)}(x),\ l_{\mu+1} = l_\mu.$$

If $d_\mu \neq 0$, the algorithm comes back to a previous row ρ, such that $d_\rho \neq 0$ and $\rho - l_\rho$ is maximum.

Then

$$\begin{aligned} \sigma_{BM}^{(\mu+1)}(x) &= \sigma_{BM}^{(\mu)}(x) + d_\mu d_\rho^{-1} x^{(\mu-\rho)} \sigma^{(\rho)}(x), \\ l_{\mu+1} &= \max(l_\mu,\, l_\rho + \mu - \rho), \\ d_{\mu+1} &= s_{\mu+2} + \sigma_1^{(\mu+1)} s_{\mu+1} + \sigma_2^{(\mu+1)} s_\mu + \cdots + \sigma_{l_{\mu+1}}^{(\mu+1)} s_{\mu+2-l_\mu} \end{aligned} \quad (3.39)$$

In general, if the degree of $\sigma_{BM}^{(2t)}(x)$ is larger than t, this polynomial has no meaning as its roots do not correspond to the inverses of the real error-location numbers, since the number of error elements is more than t elements which is over the error-correction capability of the code.

After the calculation of the error-location polynomial, the roots of this polynomial are calculated by applying the Chien search, which will be explained later.

The Error-Evaluation Polynomial $W(x)$

RS codes are non-binary codes, where the error values are non-binary, so we need to calculate not only the position but also the value of an error, to perform error correction. Once the B-M algorithm calculates the error-location polynomial, it substitutes by this polynomial in equation 3.18 to get the error evaluation polynomial, then error values can be calculated using equation 3.12. But there is an another method [18] for evaluating error values. This method requires the definition of the error evaluation polynomial in a different way [17]:

$$\begin{aligned} W(x) &= 1 + (s_1 + \sigma_1)x + (s_2 + \sigma_1 s_1 + \sigma_2)x^2 + \cdots \\ &\quad + (s_\tau + \sigma_1 s_{\tau-1} + \sigma_2 s_{\tau-2} + \cdots + \sigma_\tau)x^\tau \end{aligned} \quad (3.40)$$

Equation 3.12 is modified to

$$e_{j_i} = \frac{W(\beta_i^{-1})}{\prod_{k=1,\, k\neq i}^{\tau}(1+\beta_k\beta_i^{-1})} \tag{3.41}$$

So B-M algorithm called serial architecture [19] as the error locator polynomial is calculated first then the error evaluator polynomial is computed. After the explanation of B-M algorithm, we can conclude that this algorithm is suitable for low or moderate rate applications like CDs, DVDs, HDTV,.....etc. But it's not suitable for high rat applications like optical communication so this algorithm is not used in this thesis.

3.2.4 Decoding of RS Codes Using Peterson-Gorenstein-Zierler (PGZ) Algorithm

The PGZ algorithm is one of the decoding algorithms which solves equation 3.18. The algorithm includes two main steps [20]. Solving Newton identity is the first step, where equation 3.35 can be written in matrix form:

$$\begin{bmatrix} s_2 & s_3 & \cdots & s_{t+1} \\ s_3 & s_4 & \cdots & s_{t+2} \\ \vdots & \vdots & \ddots & \vdots \\ s_{t+1} & s_{t+2} & \cdots & s_{2t} \end{bmatrix} \begin{bmatrix} \sigma_{t-1} \\ \sigma_{t-2} \\ \vdots \\ \sigma_0 \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_t \end{bmatrix} \tag{3.42}$$

That is, the syndrome values are used to solve equation 3.42.

Define the error location polynomial as

$$\sigma(x) = \sigma_0 + \sigma_1 x + \cdots + \sigma_{t-1}x^{t-1} + x^t \tag{3.43}$$

Then, we can solve the Key equation 3.18, where the error value polynomial is defined as

$$W(x) = W_0 + W_1 x + \cdots + W_{t-1}x^{t-1} \tag{3.44}$$

PGZ Algorithm for $t = 1$

Given $t = 1$, from equation 3.42, we have

$$[s_2][\sigma_0] = [s_1] \tag{3.45}$$

and

$$\sigma_0 = \frac{s_1}{s_2} \tag{3.46}$$

Next, we can solve the key equation 3.18 for $t = 1$.

$$W(x) = -(\sigma_0 + x)(s_1 + s_2 x) \, mod \, x^2 \tag{3.47}$$

where the error value polynomial is

$$\begin{aligned} W(x) &= W_0 && (3.48) \\ W_0 &= \sigma_0 s_1 && (3.49) \end{aligned}$$

PGZ Algorithm for $t = 2$

Given $t = 2$, equation 3.42 is reduced to

$$\begin{bmatrix} s_2 & s_3 \\ s_3 & s_4 \end{bmatrix} \begin{bmatrix} \sigma_0 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} \tag{3.50}$$

Then, we have

$$\begin{aligned} \sigma_0 &= \frac{s_1 s_2 + s_2^2}{s_2 s_4 + s_3^2} \\ \sigma_1 &= \frac{s_2 s_3 + s_1 s_4}{s_2 s_4 + s_3^2} \end{aligned} \tag{3.51}$$

Then the error location polynomial can be written as

$$\sigma(x) = \sigma_0 + \sigma_1 x + x^2 \tag{3.52}$$

Solving the key equation for $t = 2$ yields

$$W(x) = -(\sigma_0 + \sigma_1 x + x^2)(s_1 + s_2 x + s_3 x^2) \, mod \, x^4 \tag{3.53}$$

Then, the error value polynomial can be represented as

$$W(x) = W_0 + W_1 x \tag{3.54}$$

Where

$$\begin{aligned} W_0 &= \sigma_0 s_1 \\ W_1 &= \sigma_0 s_2 + \sigma_1 s_1 \end{aligned} \tag{3.55}$$

PGZ Algorithm for $t = 3$

Similarly, for $t = 3$, we have

$$\begin{bmatrix} s_2 & s_3 & s_4 \\ s_3 & s_4 & s_5 \\ s_4 & s_5 & s_6 \end{bmatrix} \begin{bmatrix} \sigma_2 \\ \sigma_1 \\ \sigma_0 \end{bmatrix} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} \tag{3.56}$$

The coefficients of the error location polynomial can be solved as

$$\begin{aligned} \sigma_0 &= \frac{s_1 s_3 s_5 + s_1 s_4^2 + s_3^3}{s_2 s_4 s_6 + s_4^3 + s_3^2 s_6 + s_2 s_5^2} \\ \sigma_1 &= \frac{s_2^2 s_6 + s_1 s_4 s_5 + s_3^2 s_4 + s_2 s_4^2 + s_1 s_3 s_6 + s_2 s_3 s_5}{s_2 s_4 s_6 + s_4^3 + s_3^2 s_6 + s_2 s_5^2} \\ \sigma_2 &= \frac{s_1 s_4 s_6 + s_2 s_4 s_5 + s_3^2 s_5 + s_1 s_5^2 + s_2 s_3 s_6 + s_3 s_4^2}{s_2 s_4 s_6 + s_4^3 + s_3^2 s_6 + s_2 s_5^2} \end{aligned} \tag{3.57}$$

Then, the error location polynomial can be written as

$$\sigma(x) = \sigma_0 + \sigma_1 x + \sigma_2 x^2 + x^3 \tag{3.58}$$

The key equation for $t = 3$ can be written as

$$W(x) = -(\sigma_0 + \sigma_1 x + \sigma_2 x^2 + x^3)(s_1 + s_2 x + s_3 x^2 + s_4 x^3 + s_5 x^4 + s_6 x^5) mod\, x^6 \tag{3.59}$$

Then, the error value polynomial can be represented as

$$W(x) = W_0 + W_1 x + W_2 x^2 \tag{3.60}$$

Where

$$\begin{aligned} W_0 &= \sigma_0 s_1 \\ W_1 &= \sigma_0 s_2 + \sigma_1 s_1 \\ W_2 &= \sigma_0 s_3 + \sigma_1 s_2 + \sigma_2 s_3 \end{aligned} \tag{3.61}$$

That is clear, equation 3.57 is very complicated in compared with equations 3.46 and 3.51. So the direct implementation of equation 3.57 will be complicated. Hence, PGZ algorithm is very efficient to correct a small number of errors ($t < 3$). However, it becomes too complicated when the number of errors is relatively large.

3.3 Comparison of Decoding Algorithms

3.3.1 Complexity

In this chapter, algorithms of Reed-Solomon decoders have been considered. We compare the different algorithms with regard to circuit complexity. For $t \leq 2$, the PGZ has the best choice. But the hardware complexity of Euclidean algorithm is larger than Berlekamp-Massey algorithm as Euclidean algorithm needs divider module to implement equation 3.22 and multiplier module to implement equation 3.25. Then we can make the conclusion:

The circuit complexity : PGZ>Euclidean>Berlekamp-Massey

3.3.2 Critical path

We also compare the different algorithms with regard to the timing delay for minimum delay, Berlekamp-Massey algorithm has the best choice as it has lower complexity. So we can conclude that :

The critical path : Euclidean>PGZ>Berlekamp-Massey

3.3.3 Latency

If we compare between the different algorithms with regard to the latency in clock cycles. Berlekamp-Massey algorithm has larger latency in clock cycles in compared with Euclidean algorithm. As it is iterative algorithm so its latency is $O(t^2)$ So

Euclidean algorithm is suitable for multi-channels application as it can be shared between more than one channel. So we can make this conclusion :

The latency (clock cycles) : Berlekamp-Massey> Euclidean>PGZ

3.3.4 Power consumption

We also compare the different algorithms with regard to the power consumption. For minimum power consumption, the berlekamp-Massey algorithm has the the best choice. We can make the conclusion :

The power consumption : PGZ >Euclidean>Berlekamp-Massey

For comparisons of Euclidean algorithm and Berlekamp-Massey algorithm, of the chip size is a important factor, Berlekamp algorithm is the best choice. If the processing speed is an important factor, we suggest to choose the euclidean algorithm. So as we are targeting multi-channel application, we used Euclidean algorithm.

3.4 Chien Search

The next step in RS decoding process is to find the roots of the error location polynomial $\sigma(x)$, which is a polynomial whose roots are constructed to be the inverses of the locations where the errors occurred. We can not get the roots of the error location polynomial directly. So, we will perform an exhaustive search by substituting by all the finite field elements in the error location polynomial $\sigma(x)$ and checking for the condition $\sigma(\alpha^i) = 0$ is satisfied or not . The Chien search is effective algorithm is to do this exhaustive search in an efficient manner.

For example, assume that the error location polynomial is

$$\sigma(x) \quad = \quad 1 + \sigma_1 x + \sigma_2 x^2 + \cdots + \sigma_t x^t \tag{3.62}$$

We evaluate $\sigma(x)$ at each non-zero element in GF(2^m) in succession:

$$x = \alpha, \;\; x = \alpha^2, \ldots, x = \alpha^{2^m - 1}$$

This gives us the following

$$\begin{aligned}
\sigma(\alpha) &= 1 + \sigma_1(\alpha) + \sigma_2(\alpha)^2 + \cdots + \sigma_t(\alpha)^t \\
\sigma(\alpha^2) &= 1 + \sigma_1(\alpha^2) + \sigma_2(\alpha^2)^2 + \cdots + \sigma_t(\alpha^2)^t \\
&\vdots \\
\sigma(\alpha^{2^m-1}) &= 1 + \sigma_1(\alpha^{2^m-1}) + \sigma_2(\alpha^{2^m-1})^2 + \cdots + \sigma_t(\alpha^{2^m-1})^t
\end{aligned} \tag{3.63}$$

If in the above performed substitutions we obtained $\sigma(\alpha^i) = 0$, then the exponent of the inverse of the root α^i is equal to the error location index i.

3.5 Forney Algorithm

After calculating the error location polynomial by the key equation solver and its roots using Chien search , there is still one more step in the RS decoding [2]:

we have to find the error values. In general, this is done using the Forney algorithm as shown in equation 3.12.

So after calculating the error locations and error values, we can form the error polynomial $e(x)$ and correct the received polynomial $r(x)$ just by adding (with XOR operation) these two polynomials together, as shown in figure 3.1.

Chapter 4

CONFIGURABLE MULTI-CHANNEL REED-SOLOMON DECODER

4.1 Architectures of Syndrome computation

4.1.1 Serial architecture

As explained in the previous chapter the received vector can be presented as a polynomial as expressed in equation 3.2. So The syndrome polynomial $S(x)$ is defined as

$$S(x) = \sum_{i=1}^{2t} S_i x^{i-1} \tag{4.1}$$

Where

$$S_i = r(\alpha^i) = r_0 + r_1(\alpha^i)^1 + r_2(\alpha^i)^2 + \cdots + r_{n-1}(\alpha^i)^{n-1} \tag{4.2}$$

If all $2t$ syndromes are zeros, then the received polynomial $r(x)$ is a valid codeword $c(x)$, that is, no detectable occurred. So from equation 4.2 the syndrome polynomial can be computed in n clock cycles as the received codeword enters serially to syndrome block from highest coefficient to the lowest coefficient, i.e. r_{n-1} enters first then r_{n-2} and so on. So we will have $2t$ syndrome cells and each cell substitutes by α^i in the received polynomial as shown in figure 4.1. Each syndrome cell has a multiplier by constant multiplying by α^i so after n clock cycles r_{n-1} is multiplied by $(\alpha^i)^{n-1}$ as shown in figure 4.2 which illustrates the

serial syndrome cell.

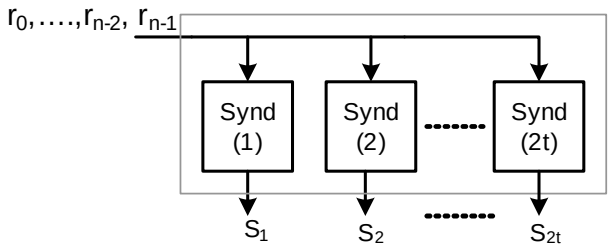

Figure 4.1: Serial syndrome

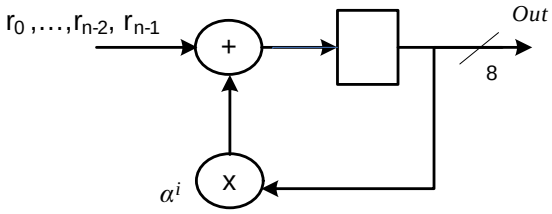

Figure 4.2: Serial syndrome cell

4.1.2 Two Parallel architecture

The syndrome coefficient can be computed in $\frac{n}{2}$ clock cycles if the received codeword enters two symbols by two symbols but this needs a modification to equation 4.2 as follows [21]:

$$\begin{aligned} S_i &= (...(((r_{n-1}(\alpha^i)^2 + r_{n-2}(\alpha^i) + r_{n-3})(\alpha^i)^2 \\ &+ (r_{n-4}(\alpha^i) + r_{n-5}))(\alpha^i)^2 + (r_{n-6}(\alpha^i) + r_{n-7}))(\alpha^i)^2 \\ &+ ... + r_1\alpha^i + r_0 \end{aligned} \tag{4.3}$$

From equation 4.3 it is clear that if n is odd number, the odd symbols are multiplied by α^i, the even symbols are multiplied by one except r_{n-1} is multiplied by $(\alpha^i)^2$, and the result of the summation is multiplied by $(\alpha^i)^2$. So we have two paths for the received polynomial one for the even symbols and the other for odd symbols and usually we have a pair of symbol one odd and the other even except r_{n-1}will enter with zero as shown in figure 4.3.

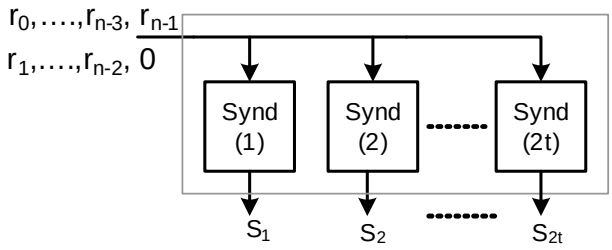

Figure 4.3: Two parallel Syndrome

Figure 4.4 shows that, each syndrome cell has two multipliers by constant one of them multiplies by α^i and the other multiplies by $(\alpha^i)^2$. Initially the selection signal Sel equals to zero for first clock cycle to multiply r_{n-1} by (α^2) and then it equals to one for the remaining clock cycles to multiply $(r_{odd}\alpha^i + r_{even})$ by $(\alpha^i)^2$ [21].

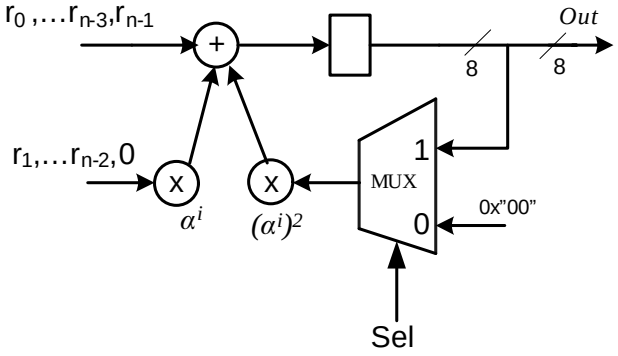

Figure 4.4: Two Parallel syndrome cell

4.1.3 The new Configurable architecture

The two parallel syndrome architecture can be configured to work either in serial mode or in two parallel mode by designing a new configurable syndrome cell as shown in figure 4.5.

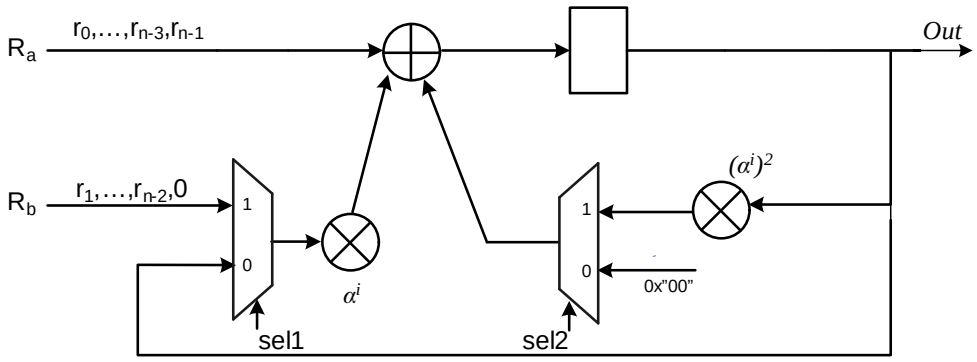

Figure 4.5: Configurable syndrome cell

If the selection signals sel1 and sel2 are low, the received codeword enters serially through $\mathrm{R_a}$, the latency of syndrome block is n clock cycles, as the architecture will be like figure 4.2. But if the selection signal sel1 is high and sel2 is low for the first clock cycle, then both of them are high for the remaining clock cycles, and the received codeword enters two symbols by two symbols through $\mathrm{R_a}$and $\mathrm{R_b}$, the latency will be $\frac{n}{2}$ clock cycles as the cell will be converted to two parallel cell as shown in figure 4.4.

4.2 Hardware Implementation of Euclidean Algorithm

The overall block diagram of RS decoder using Euclidean algorithm is shown in figure 4.6. The architecture consists of two parts:

1- Euclidean Divider Module to implement ,

$$r_i = r_{i-2} - q_i r_{i-1} \tag{4.4}$$

2- Euclidean Multiply Module to implement,

$$t_i = t_{i-2} - q_i t_{i-1} \tag{4.5}$$

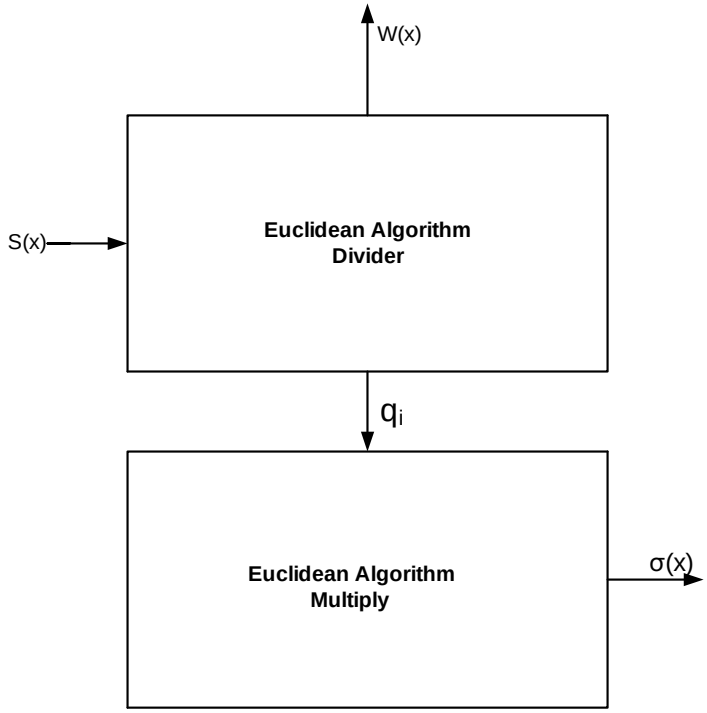

Figure 4.6: Block diagram of the Euclidean architecture.

4.2.1 Euclidean Division Module

The Euclidean division module performs the division r_{i-2}/r_{i-1} in each iteration in equation 4.4. It generates the quotient q_{i-1} and stores the new remainder r_i. The quotient q_{i-1} is used in the Euclidean multiply module to compute equation 4.5. An Euclidean divider module is shown in figure 4.7. This polynomial division architecture includes two major components, the EAdiv A and EAdiv B modules respectively [22]. These architectures are showed in figure 4.8.

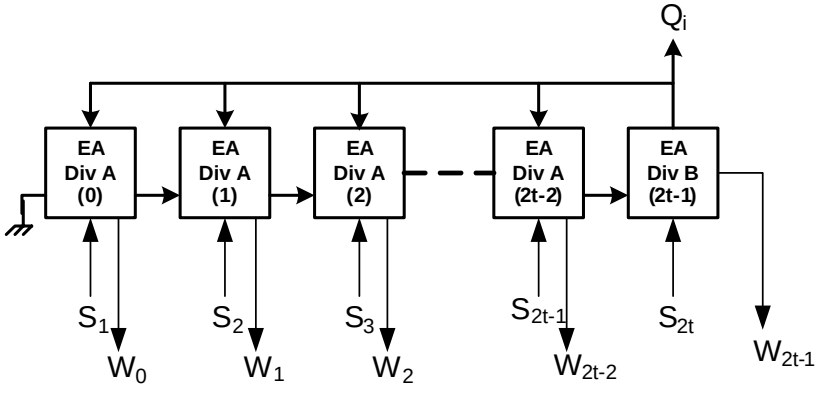

Figure 4.7: Overall architecture of the Euclidean divider module.

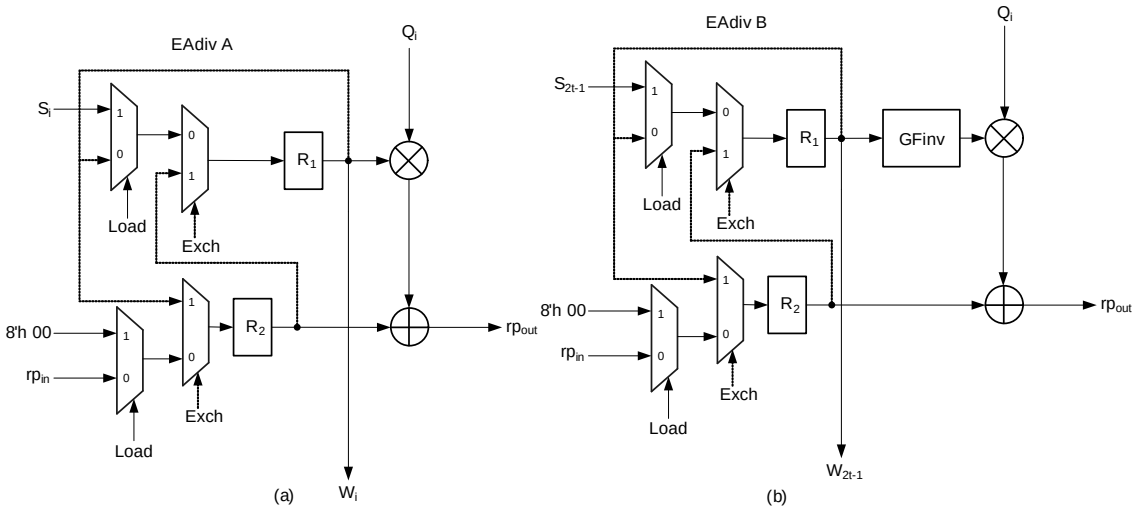

Figure 4.8: EAdiv A and EAdiv B of the Euclidean divider module.

4.2.2 Euclidean Multiply Module

The Euclidean multiply achieves the multiplication and accumulation in the polynomial in equation 4.5. It is used to obtain error location polynomial $\sigma(x)$. The Euclidean multiply module is shown in figure 4.9. The coefficients of the quotient $Q(x)$ are input sequentially from the higher coefficients. The polynomial multiply architecture includes one major component of the EAmulC module, as shown in figure 4.10.

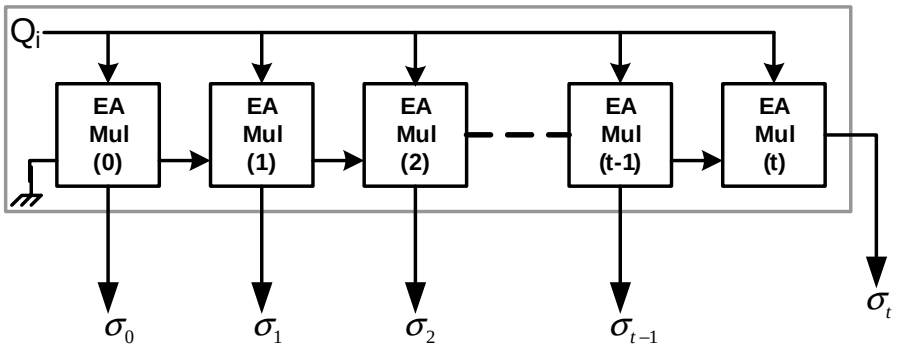

Figure 4.9: Block diagram of the Euclidean multiply module.

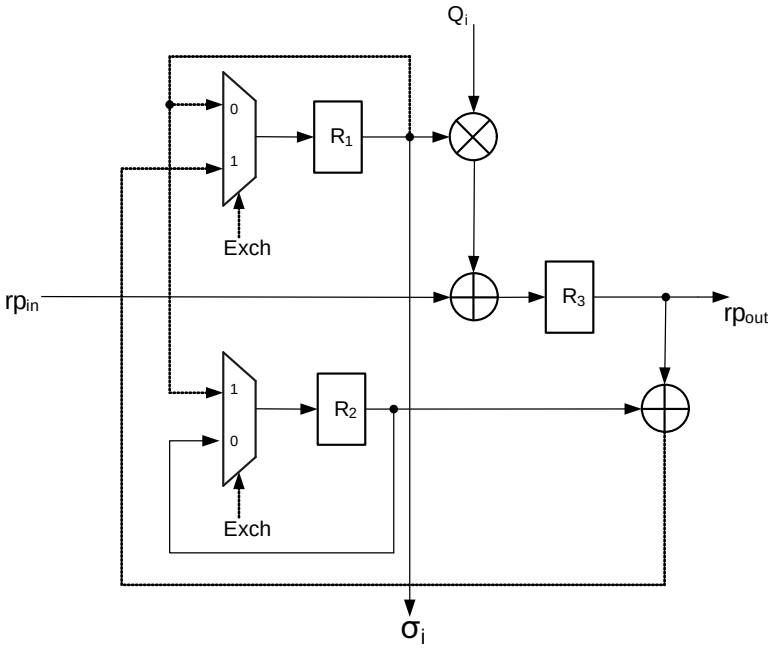

Figure 4.10: Architecture of the EAmul C module in the Euclidean multiply operation.

4.3 Architectures of Chien search block

4.3.1 Serial architecture

The chien search block substitutes in the error locator polynomial and error evaluator polynomial by all the field elements as shown in equation 3.63. We have t cells to get $\sigma(\alpha^i)$ and $t-1$ cells to get $W(\alpha^i)$ as $\sigma(x)$ has order t and $W(x)$ has order $t-1$ as shown in figure 4.11.

Also the role of Chien search block is to get $\grave{\sigma}(\alpha^i)$ but

$$\begin{aligned} \grave{\sigma}(x) &= 0\sigma_0 + \sigma_1 + 2\sigma_2 x + \cdots + t\sigma_t x^{t-1} \\ &= \sigma_1 + \sigma_3 x^2 + \cdots + (t-1)\sigma_{t-1} x^{t-2} \\ &= \sigma_{odd}(x)/x \end{aligned} \tag{4.6}$$

So the Chien search block gets also $\sigma_{odd}(\alpha^i)$. The serial Chien search cell is similar to the serial syndrome cell as it has the same function as shown in fig. 4.12

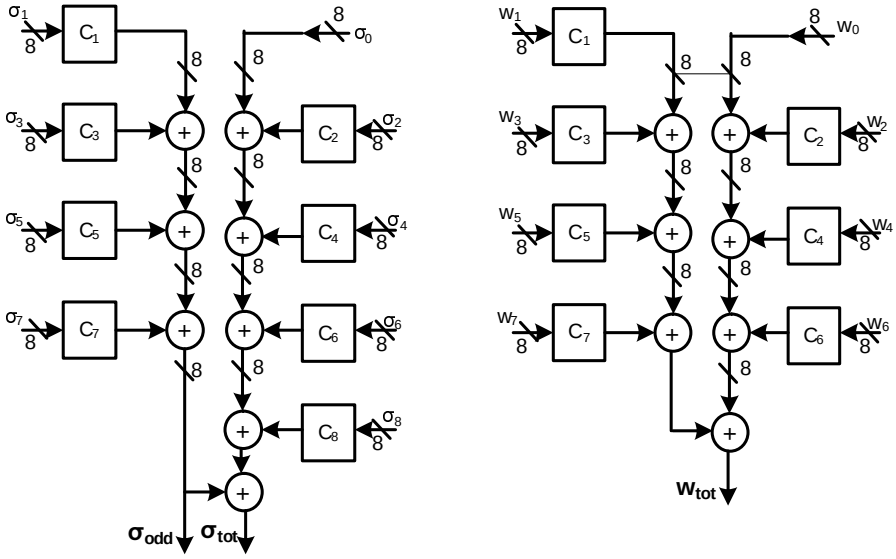

Figure 4.11: Serial Chien search ($t = 8$).

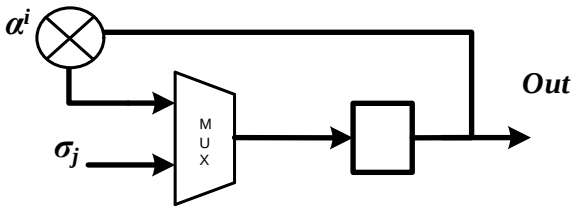

Figure 4.12: Serial Chien search cell

4.3.2 Two Parallel architecture

In the serial architecture of Chien search at each clock cycle we get a value for $\sigma(\alpha^i)$, $\grave{\sigma}(\alpha^i)$, and $W(\alpha^i)$ so we need n clock cycles as we have n elements in GF(2^m). In the two parallel architecture we get two values at each clock cycle, i.e. $\sigma(\alpha^i)$ and $\sigma(\alpha^{i+1})$. So the we need $n/2$ clock cycles as the output of each cell is $\sigma_j(\alpha^i)^j$and $\sigma_j(\alpha^{i+1})^j$ as shown in figure 4.13.

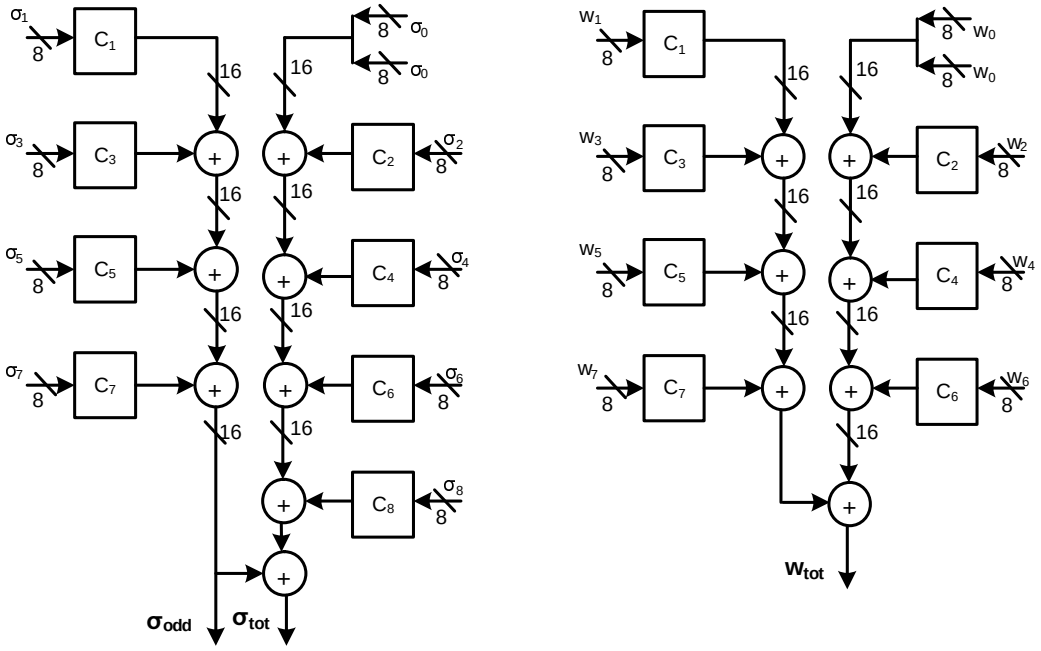

Figure 4.13: Two Parallel Chien search ($t = 8$)

The two parallel Chien search cell has four finite field multipliers by constant [21] one of them multiplies by α^i and the others multiply by $(\alpha^i)^2$. We need the multiplier which multiplies by α^i in the first clock cycle only then we need the others multipliers as shown in figure 4.15.

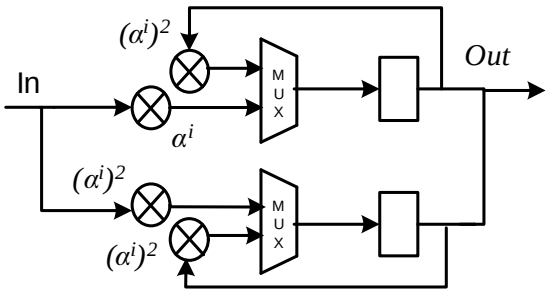

Figure 4.14: Two Parallel Chien search cell

4.3.3 The new configurable architecture

The Chien search can be configured to work either in two parallel mode or in serial mode and that's by three finite field multipliers only so the number of multipliers is reduced than the two parallel architecture [21] as shown in figure 4.15.

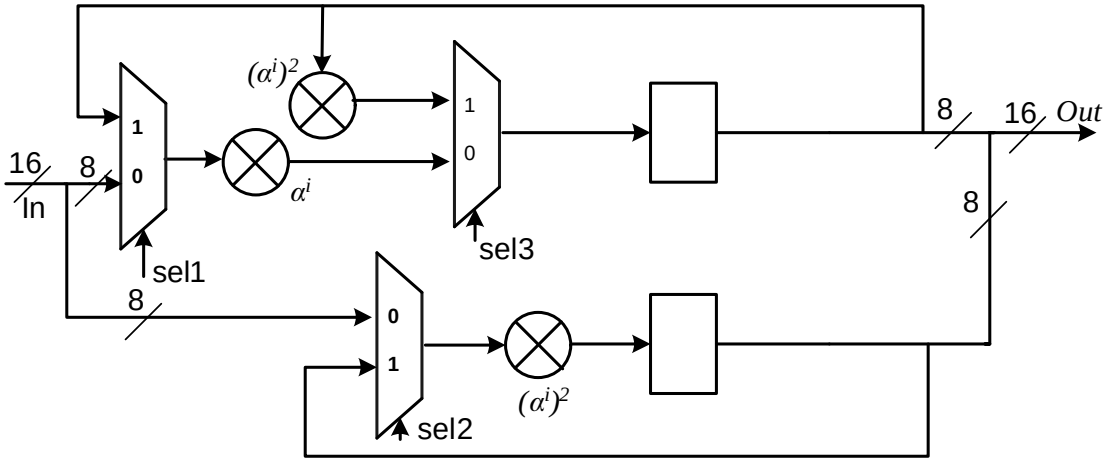

Figure 4.15: Configurable Chien search cell

To activate the cell in the two parallel mode with latency equals to $\frac{n}{2}$ clock cycles, set the selection lines sel1, sel2, and sel3 to low in 1^{st} clock cycles then set them to high, but to activate the cell in serial mode with latency equals to n clock cycles, set the selection line sel1 to low in 1^{st} clock cycles then set it to high, while the selection lines sel2 and sel3 are always low.

4.4 Architecture of Forney Algorithm

4.4.1 Serial architecture

The main function of the Forney algorithm is to get the error values from equation 3.12. But from equation 3.12 and 4.6 we can rewrite equation 3.12 as follows :

$$e_{j_l} = \frac{W(\beta_l^{-1})\beta_l^{-1}}{\sigma_{odd}(\beta_l^{-1})} \tag{4.7}$$

Where β_l^{-1} is the root of $\sigma(x)$ so in the architecture of forney algorithm we need inverse ROM to get the inverse of the $\sigma_{odd}(\beta_l^{-1})$ and a look up table (LUT) to get β_l^{-1} at each location but we replaced the LUT by a specific cell takes an integer number i and its output is α^i which reduces the area significantly as shown in figure 4.16. The serial Forney block takes n clock cycles to produce n error values as in each clock cycle we have one value for $W(\beta_l^{-1})$ and $\sigma_{odd}(\beta_l^{-1})$.

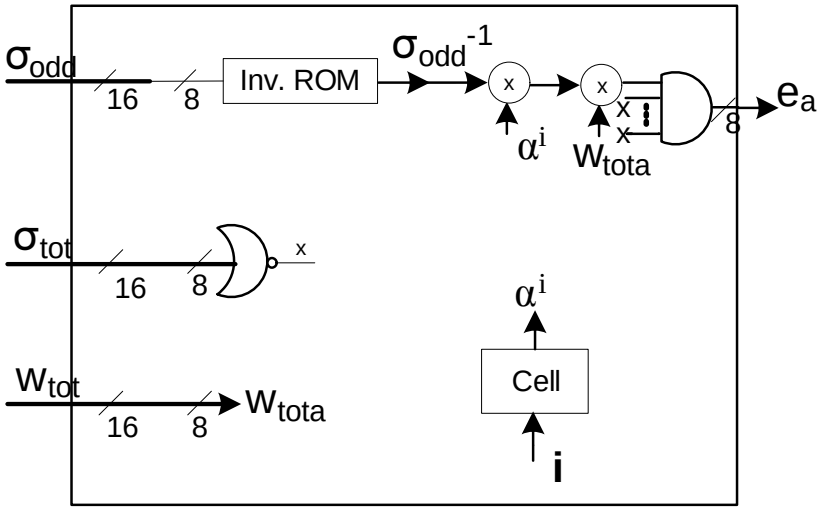

Figure 4.16: Serial Forney architecture

4.4.2 Two Parallel architecture

In the two parallel architecture two error values can be computed in each clock cycle as in this case we have two values for $W(\beta_l^{-1})$ and $\sigma_{odd}(\beta_l^{-1})$ so the architecture will be like figure 4.17. So the two parallel architecture tackles $n/2$ clock cycles to calculate the n error values.

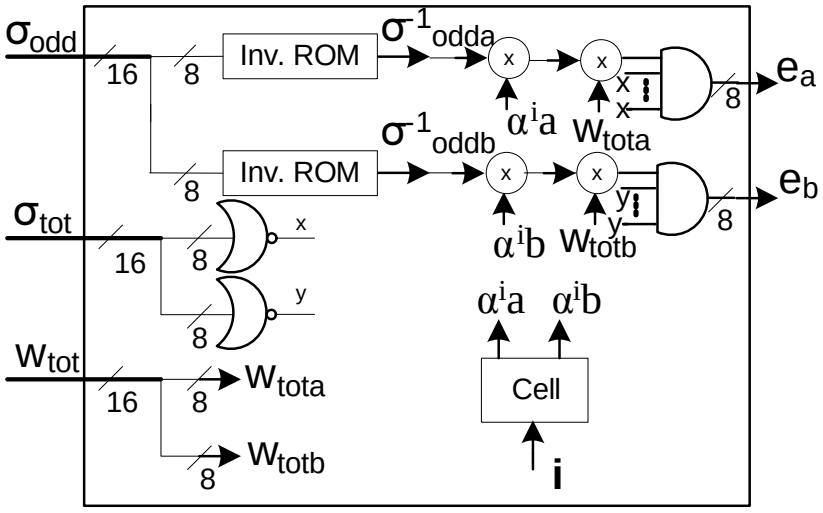

Figure 4.17: Two parallel Forney architecture

4.4.3 The new configurable architecture

Forney algorithm architecture can be configured easily by using the two parallel architecture. If we enable the two paths, we have two parallel architecture. If we disable one of them, we have serial architecture.

4.5 Multi-Channel Decoder

4.5.1 Multi-Channel using serial architectures

As the latency of Euclidean algorithm block equals to $2t$ clock cycles so in RS(255, 239) we can share the key equation solver with 16 channels with serial architectures as shown in figure 4.18. So the maximum throughput per channel equals to $\frac{\text{n}*\#\text{of bits per symbol}}{\text{n}} * F_{max}$ bit per second (pbs) ,where number of bits per symbol equals to 8 and F_{max} is the maximum operating frequency of the decoder..

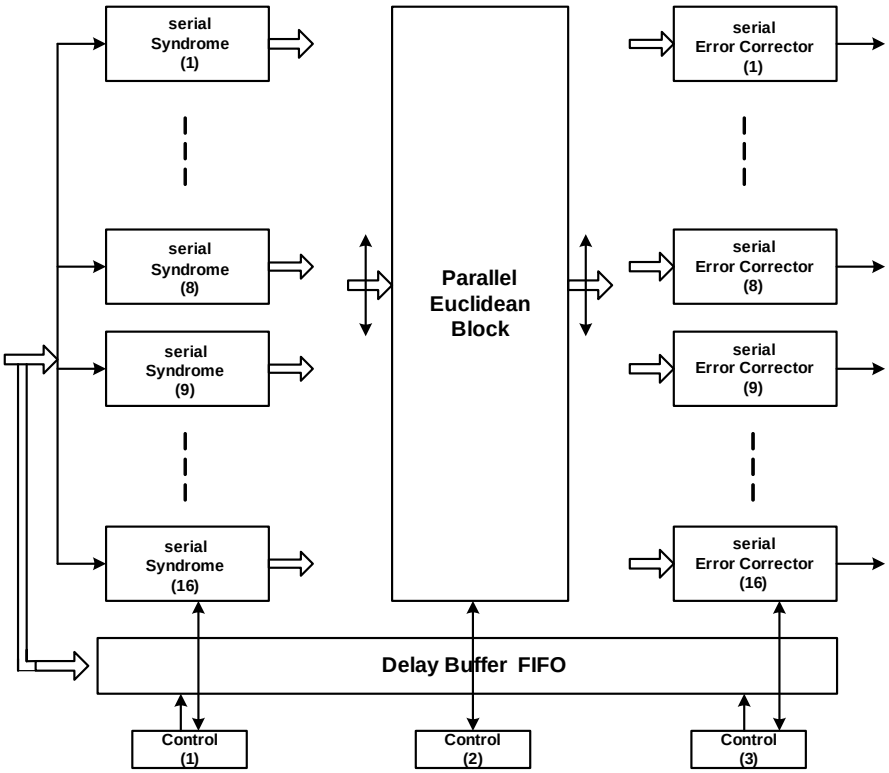

Figure 4.18: 16 Channel (255, 239) RS decoder using serial architectures

4.5.2 Multi-Channels using two parallel architectures

Also Euclidean block can be shared between 8 channels and each of them is two parallel architecture so the maximum throughput equals to $\frac{2*n*8}{n} * F_{max}$bps. So the throughput per channel is doubled but the number of channels is reduced to 8 channels as shown in figure 4.19.

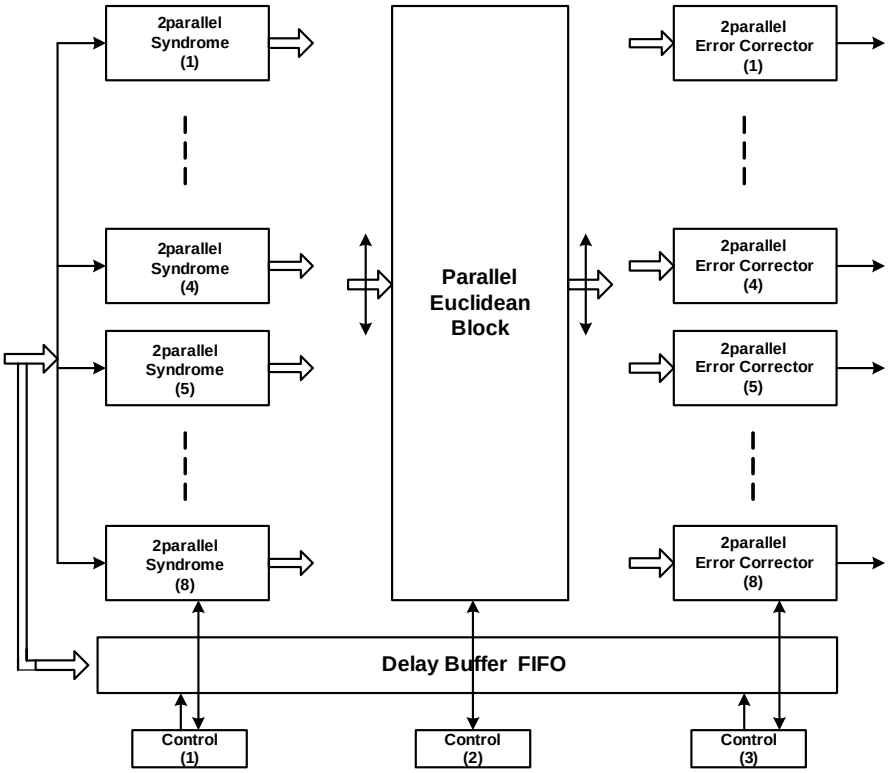

Figure 4.19: 8 Channels (255, 239) RS decoder using two parallel architectures

4.5.3 Multi-Channel using configurable architectures

We use the configurable syndrome, Chien search, and Forney algorithm architectures in configurable multi-channels to share the Euclidean algorithm between 16 channels, half of them are serial architectures and the others are configurable. By this architecture we can configure the architecture to be shared between 16 serial channels or between 8 two parallel channels as shown in figure 4.20.

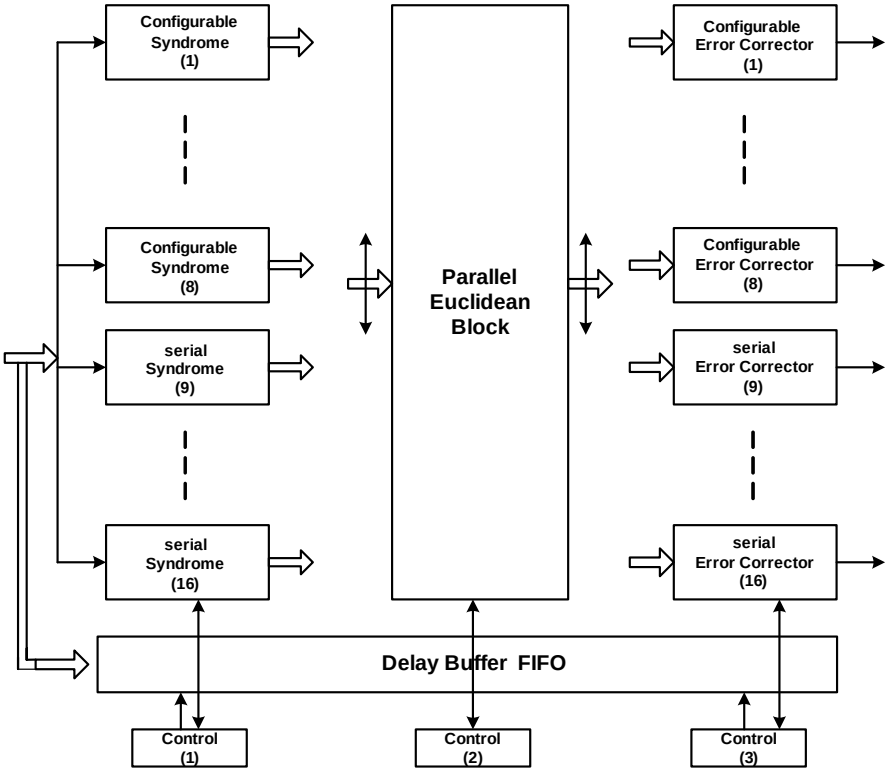

Figure 4.20: 8 Channels (255, 239) RS decoder configurable and serial architectures

4.6 Results and Comparisons

4.6.1 Results of configurable (255, 239) RS decoder

The proposed configurable architecture is modeled in VHDL and simulated to verify its functionality. After complete verification of the design functionality, it was then synthesized using appropriate time and area constraints. Both simulation and synthesis steps were carried out on 0.13μm CMOS technology and optimized for a $1.35V$ supply voltage, we used this technology to make our comparison fair with the previously published architectures. The total number of gates of single channel decoder using configurable cells is $49,000$ from the synthesized results excluding the FIFO memory, and the maximum clock frequency is $500\,MHz$. The total power dissipation for the multi-channel decoder is 568.4 mW. The latency of the architecture is 249 ns which leads to an energy per symbol of 8.84 nJ.

Table 4.1: Implementation Results of single channel RS(255, 239) Decoders

Architecture	Technology (μm)	Total # of Gates	Max. Freq. (MHz)	Latency (clocks)	Latency (ns)	Throughput (Mb/s)
Proposed	0.13	49,000	500	146	294	8,000
ME [23]	0.13	115,500	770	355	461	6,160
pDCME [24]	0.13	53,200	660	355	537.9	5,300
EA [25]	0.13	44,700	300	287	956.7	2,400
ME [26]	0.18	20,614	400	512	1280	3,200
DCME [27]	0.25	42,213	200	288	1440	1,600
BM [28]	0.25	32,900	84	192	2285.7	2,500

Table 4.2: Implementation Results of Multi-Channel RS Decoders

Architecture	Technology (μm)	Total # of Gates	Max. Freq. (MHz)	Latency (clocks)	Latency (ns)	Throughput (Gb/s)
8-Two parallel channels	0.13	200,000	500	146	294	64
16-Serial channels	0.13	200,000	500	274	548	64
PrME [29]	0.13	393,000	625	522	830	80
ME [23]	0.13	589,000	625	355	570	80
ME [30]	0.16	364,000	112	168	1500	40

4.6.2 Comparisons

Table 4.1 shows a comparison between different architectures of RS(255, 239) decoders. The Modified Euclidean (ME) algorithm is used in [26]. The results show that [26] made a modification to the euclidean algorithm to have an area efficient architecture so its area is lower than our architecture. But it has higher latency and lower throughput. The Degree Computationless Modified Euclidean (DCME) architecture is used in [24] and [27] to reduce the gate count compared to ME [23]. However, pDCME [24] has a larger area compared to our proposal and also higher maximum frequency, but it has higher latency and lower throughput. While the area of [27] is lower than our proposal, but it has lower throughput and higher latency. The Euclidean Algorithm is used in [25]. It uses the same architecture but it is implemented on different technology and it has lower area and lower latency and throughput. The Modified Berlekamp-Massey (MBM) algorithm is used in [28]. This algorithm is used to reduce the latency in clock cycles and also to reduce the power consumption (68.5mW at 84MHz), which is lower then our power per channel which is 108 mW. But this architecture has higher latency than our proposal in ns which make our proposal has lower energy per symbol.

Table 4.2 shows a comparison between different architectures of 16 channel

RS(255, 239) decoders. The proposed architecture is a compromise between the Pipelined recursive Modified Euclidean (PrME) [29], ME [23], and ME [30] in throughput, but our proposal has the least area and latency.

4.7 Conclusion

In This chapter an architecture for a low energy area efficient configurable syndrome/Chien search RS(255, 239) decoder is presented. In our proposal configurable syndrome and Chien search cells are designed to work in two modes. The first mode is the serial mode as each of them needs n clock cycles to finish. The second mode is the two parallel mode as they need $\frac{n}{2}$ clock cycles to finish which double the rate of the serial mode and decrease the energy per symbol. We used the Euclidean algorithm as a KES to give our decoder a chance to serve 16 channels to have higher throughput. The configurable cells makes the decoder works over 16 or 8 channels by duplication of the throughput of each channel without extra hardware which saves the area. Our architecture has high throughput with low power consumption and optimum area.

Chapter 5

CONCLUSIONS AND FUTURE WORKS

5.1 Conclusions

In this thesis a new architecture for RS(255, 239) decoder has been proposed by designing a new configurable syndrome and Chien search cells with Euclidean algorithm as a key equation solver. Using these configurable cells the decoder can work in two modes, one of them is serial mode and the other mode is two parallel mode. The proposed architecture has been simulated using VHDL and verified using MATLAB. This architecture is used in multi-channel decoder to serve 8 or 16 channels as the syndrome and Chien search blocks can be configured to work in serial mode or two parallel mode.

The decoder is synthesized on $0.13\mu m$ CMOS IBM library and 1.35V power supply. We used this technology to give the ability to compare between our architecture and other architectures. The total number of gates of single channel decoder using the proposed cells is $49,000$ from the synthesized results excluding the FIFO memory, and the maximum clock frequency is $500\,MHz$. The total power dissipation for the multi-channel decoder is 568.4 mW. The latency of the architecture is 249 ns which leads to an energy per symbol of 8.84 nJ.

As RS is non-binary linear cyclic code and operates over GF(2^m) so in Chapter 2, the basic theory of finite fields has been introduced and also a good overview on other algebraic structures is introduced. Also three types of finite field multipliers have been introduced and a good overview on binary and non-binary cyclic codes is introduced.

Then in Chapter 3, the RS decoder block diagram is introduced as a mathe-

matical concept and three types of decoding algorithms are introduced and a good comparison between these algorithms also are introduced.

Finally in Chapter 4, the hardware implementation of the decoder is introduced and a comparison between the proposed design and the previous work is introduced.

5.2 Future Works

In this thesis, the decoder is simulated before and after synthesis but it is not implemented to see its performance in a real application. So the next step is to convert the VHDL code to ASIC and put it an a real application to see its performance. Also the new configurable cells can be tested with another key equation solver like Modified Euclidean algorithm [23] or Reformulated inversionless Berlekamp Massey (RiBM) [31] algorithm to see the performance of these cells. Also the Reed-Solomon codes can be concatenated with convolutional code to enhance the ability of correcting random errors. Because RS codes detects the error first by using syndrome calculation block and convolutional code corrects the error directly without detection [2], some relations can be made between the two codes to save the energy of the decoder, hence saving the battery of the receiver.

Bibliography

[1] C.E. Shannon, "A Mathematical Theory introduced of Communication," Bell Syst.error control Tech. Journal, Vol. 27, Part I, July, Part II, October 1948.

[2] J. C. Moriera, P. G. Farell, Essentials of Error-Control Coding. John Wiley & Sons, Ltd, 2006.

[3] I. S. Reed and G. Solomon, "Polynomial Codes over Certain Finite Fields," SI AM Journal of Applied Mathematics, Vol. 8, June 1960.

[4] E. Berlekamp, Algebraic Coding Theory. McGraw-Hill, 1968.

[5] A. Neubauer, J. Freudenberger, V. Kuhn, Coding Theory - Algorithms, Architectures, and Applications. John Wiley & Sons, Ltd, 2007.

[6] I. Stewart, Galois theory. Chapman & Hall, 1973.

[7] F.J. Mac Williams, N. J. A. Sloane, The theory of error correcting codes. Elsevier Science Ltd, 1977.

[8] S. Choomchuay, "On the Implementation of Finite Field Basis Conversions," Ladkrabang Engineering Journal, Vol. 11, No. 1 June 1994.

[9] S. T. J. Fenn, M. Benaissa, D. Taylor, "GF(2^m) Multiplication and division over the dual field," IEEE Trans. on computers. Vol. 45, No. 3, March 1996.

[10] M. Morii, M. Kasahara, D.L Whiting, "Efficient bit-serial multiplication and the discrete-Time Wiener-Hopft Equation over finite field," IEEE Trans. on-Information Theory, Vol. 35, No. 6, November. 1989.

[11] J.L. Massey, J.K. Omura, "Computational method and apparatus for finite field arithmetic," United States Patent, No. 4587627, May 1986.

[12] E. R. Berlekamp, "Bit-serial Reed-Solomon encoders," IEEE Trans. on Information Theory, Vol. 28, No. 6, November 1982.

[13] E. Mastrovito, "VLSI design for multiplications over finite fields GF(2^m)," 6th International Conference Applied Algebra, Algebric Algorithms and Error-correcting codes, Vol. 357, 1988.

[14] B. A. Laws FR., C.K. Rushforth, "A cellular-array multiplier for GF(2^m)," IEEE Trans. Computers, Vol. C-20, No. 12, December 1971.

[15] A. B. Carlson, P. B. Crilly, and J. C. Rutledge, Communication Systems: An Introduction to Signals and Noise in Electrical Communication, McGraw-Hill, 1986.

[16] B. Sklar, Digital Communications, Fundamentals and Applications, Prentice Hall, 1993.

[17] R. C. Bose, and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," Information Control, Vol. 3, March 1960.

[18] A. Hocquenghem, "Codes correcteurs d'erreurs," Chiffres, Vol. 2, December, 1959.

[19] H. Chia Chang and C. Bernard Shung, "New Serial Architecture for the Berlekamp-Massey Algorithm," IEEE Trans. on communication, Vol. 47, No. 4, April 1999.

[20] S. Feng Wang, H. Yi Hsu, and A. Yeu Wu, "A very low-cost multi-mode Reed-Solomon decoder based on Peterson-Gorenstein-Zierler algorithm," IEEE Workshop on Signal Processing Systems, 2001.

[21] S. Lee, C. Choi, and H. Lee, "Two-parallel Reed-Solomon based FEC architecture for optical communications," IEICE Electronics Express, Vol. 5, No. 10, May 2008.

[22] H. Lee, M.-L. Yu, and L. Song, "VLSI design of Reed–Solomon decoder architectures," IEEE Int. Symp. Circuits and Syst., Vol. 5, 2000.

[23] H. Lee, "High-Speed VLSI Architecture for Parallel Reed-Solomon Decoder," IEEE Trans. on VLSI Systems, Vol. 11, No. 2, April 2003.

[24] S. Lee, H. Lee, J. Shin and J. Ko, "A High-Speed Pipelined Degree- Computationless Modified Euclidean Algorithm Architecture for Reed- Solomon Decoders," IEEE International symposium on Circuits and System, 2007.

[25] H. Lee, "An Area-Efficient Euclidean Algorithm Block for Reed-Solomon Decoder," IEEE Annual Symposium on VLSI, 2003.

[26] H. Yi Hsu, A. Yeu (Andy) Wu, and J. Yeo, "Area-Efficient VLSI Design of Reed–Solomon Decoder for 10GBase-LX4 Optical Communication Systems," IEEE Trans. on Circuits and Systems-II: express briefs, Vol. 53, No. 11, November 2006.

[27] J. H. Baek and M. H. Sunwoo, "New Degree Computationless Modified Euclidean Algorithm and Architecture for Reed-Solomon Decoder," IEEE Trans. on VLSI Systems,Vol. 14, No. 8, August 2006.

[28] H. Chang, C. Ching Lin and C. Yi Lee, "A low-power Reed-Solomon decoder for stm-16 optical communications," IEEE Asian-Pasific Conference on ASIC 2002.

[29] H. Lee, "Ultra High-Speed Reed-Solomon Decoder," IEEE International Symposium on Circuits and Systems, Vol. 2, 2005.

[30] L. Song, M-L. Yu and M. S. Shaffer, "10 and 40-Gb/s Forward Error Correction Devices for Optical Communications," IEEE Journal of Solid-State Circuits, Vol. 37, No. 11, November 2002.

[31] D. V. Sarwate and N. R. Shanbhag, "High-Speed Architectures for Reed-Solomon Decoders," IEEE Trans. on VLSI Systems, Vol. 9, No. 5, October 2001.

Appendix A

The list of primitive polynomials $p(x)$ for m $\leq$ 10

m	$p(x)$
3	$1 + x + x^3$
4	$1 + x + x^4$
5	$1 + x^2 + x^5$
6	$1 + x + x^6$
7	$1 + x + x^7$
8	$1 + x^2 + x^3 + x^4 + x^8$
9	$1 + x^4 + x^9$
10	$1 + x^3 + x^{10}$

Appendix B

Conversions from standard basis to Normal basis in GF(2^4)

power of α	Standard basis $1, \alpha, \alpha^2, \alpha^3$	Normal basis $\alpha^3, \alpha^6, \alpha^{12}, \alpha^9$
-	0000	0000
0	1000	1111
1	0100	1001
2	0010	1100
3	0001	1000
4	1100	0110
5	0110	0101
6	0011	0100
7	1101	1110
8	1010	0011
9	0101	0001
10	1110	1010
11	0111	1101
12	1111	0010
13	1011	1011
14	1001	0111