TWO EXTENDED PROGRAMMABLE BCH SOFT DECODERS USING LEAST RELIABLE BITS REPROCESSING

by
Mohamed Tarek Abdelsadek Osman

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 COMMUNICATIONS

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2014

TWO EXTENDED PROGRAMMABLE BCH SOFT DECODERS USING LEAST RELIABLE BITS REPROCESSING

by
Mohamed Tarek Abdelsadek Osman

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 COMMUNICATIONS

Under the Supervision of

Associate Professor Ahmed Shalash
Principal Adviser

Associate Professor Hossam Aly
Hassan Fahmy
Adviser

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2014

TWO EXTENDED PROGRAMMABLE BCH SOFT DECODERS USING LEAST RELIABLE BITS REPROCESSING

by
Mohamed Tarek Abdelsadek Osman

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 COMMUNICATIONS

Approved by the
Examining Committee

Associate Professor Ahmed Shalash, Thesis Main Advisor

---

reader1, Member

---

reader2, Member

FACULTY OF ENGINEERING, CAIRO UNIVERSITY
GIZA, EGYPT
2014

Acknowledgements

In the name of Allah the most merciful the most gracious; all thanks to Allah the Lord of the Heavens and Earth and peace be upon Mohamed and his companions. I wish to express my gratitude to my advisers, Associate Prof. Ahmed Shalash and Hossam Fahmy who were helpful and offered invaluable assistance, support and guidance. I am also genuinely grateful for the support of Associate Prof. Yasmine Fahmy and Maha Elsabrouti throughout my work.

Many thanks to my colleagues at silicon vision for their support and help through the duration of this work.

My deepest gratitude to my family. Without their encouragement, I would not have gone this far.

Mohamed

Abstract

This thesis proposes two BCH soft decoders suitable for high-rate codes with medium to large word length. The proposed decoders extend the correcting capability by providing a programmable performance gain according to the choice of the extra compensated bits p, with a theoretical maximum likelihood decoding when $2t + p$ approaches the codeword size n, where t is the correcting capability of the code under algebraic decoding. Our proposed architectures for the proposed algorithms use pipelined arithmetic units, leading to a reduction in the critical paths. This allows for an increase in the operating frequency by up to $\frac{m}{2}$ times compared to algebraic decoders, where m is the Galois field size. Our proposed decoders operate only on the least reliable bits, which leads to a reduction in the decoder complexity by removing the Chien search procedure.

Synthesizing the two proposed decoders on TSMC 90 nm G technology for BCH (255, 239), our proposed EBP decoder obtains a throughput of 617.5 Mbits/sec, 509 Mbits/sec and 420 Mbits/sec at an area of 14727, 18510 and 22365 μm^2 at $p = 2, 4\, and\, 6$ obtaining a gain of 0.05, 0.2, 0.35 dB over algebraic decoding. While EHe decoder obtains a throughput of 726.4 Mbits/sec, 458.7 Mbits/sec and 185.4 Mbits/sec at an area of 14320, 19542 and 28461 μm^2 at $p = 2,\ 4\, and\, 6$ obtaining a gain of 0.75, 1, 1.2 over algebraic decoding.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

BCH codes are invented in 1959 by French mathematician Alexis Hocquenghem [1], and independently in 1960 by Raj Bose and D. K. Ray-Chaudhuri [2]. The acronym BCH comprises the initials of these inventors names. BCH codes are powerful linear block codes that are used widely in multimedia and storage applications; this is due to the fact that BCH codes acquire strong correcting capability outperforming Reed-Solomon codes [3] in binary channels [4]. The class of BCH codes employed in such systems is characterized by large word length and low redundancy length. The choice of class is because BCH codes are employed to remove the error floor of the system, either by being concatenated to a more powerful channel codes as in DVB-T2 [5] and DVB-C2 [6] systems, or in systems with low probability of error like storage applications [7].

Conventional decoding of BCH codes is simple and suitable for VLSI implementation [8]. Yet hard decision decoding has limited decoding capability as it disregards channel information which is available in most of BCH code applications. In this thesis we propose two programmable soft decoders both targeting the class of BCH codes mentioned above. These decoders acquire a programmable coding gain and speed that outperforms conventional decoding, with area comparable to conventional decoding. These decoders provide a trade off between complexity and performance, and are feasible to be implemented in programmable hardware. This allows our proposed decoders to be used in multi-rate systems, or multiple power consumption architectures where the performance can be sacrificed in a certain mode to reduce the system power consumption. The work done in this thesis was published in [9] and [10].

This rest of the thesis is organized as follows. An introduction about Galois

fields is given in chapter 2, an overview on BCH codes encoding and conventional decoding is detailed in chapter 3, then a survey on BCH soft decoding is given in chapter 4, followed by a survey on Galois field arithmetic units in chapter 5. The two proposed decoding algorithms are shown in chapter 6, and their proposed implementation are described in chapter 7.

Chapter 2

Galois Fields

Galois field, so named in honour of Evariste Galois, is a field that contains a finite number of elements. All mathematical operations in BCH codes are performed using Galois field arithmetic. In this chapter the properties of Galois field are presented, then different representation approaches of Galois field elements are explained. At the end of this chapter a mathematical foundation of arithmetic operations is presented.

2.1 Galois Field Properties

A field F is a non-empty set closed under two operations addition and multiplication, denoted by '+' and '*' respectively. For F to be a field a number of conditions must hold [11] [12]:

1. Closure: For every a, b in F,

$$c = a + b\,, \qquad d = a * b, \tag{2.1}$$

   where c, $d \in F$.

2. Associative: For every a, b, c in F

$$a + (b + c) = (a + b) + c, \qquad a * (b * c) = (a * b) * c. \tag{2.2}$$

3. Identity: For every element a in F, there exists an identity element '0' for

addition and '1' for multiplication that satisfy,

$$a + 0 = 0 + a = a, \qquad a * 1 = 1 * a = a. \tag{2.3}$$

4. Inverse: For every element a in F, there exist elements b, c in F such that,

$$a + b = 0, \qquad a * c = 1, \tag{2.4}$$

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

5. Commutative: For every a, b in F

$$a + b = b + a, \qquad a * b = b * a. \tag{2.5}$$

6. Distributive: For every a, b, c in F

$$(a + b) * c = a * c + b * c. \tag{2.6}$$

The existence of a multiplicative inverse a^{-1} leads to the presence of division operation. This is because for every a, b, c in F, $c = b/a$ is defined as $c = b * a^{-1}$. Similarly the existence of an additive inverse $(-a)$ enables the use of subtraction. In this case for every a, b, c in F, $c = b - a$ is defined as $c = b + (-a)$. It can be shown that the set of integers 0, 1, 2, ... , $p - 1$ where p is a prime, together with modulo p addition and multiplication forms a field [13]. Such a field is called the finite field of order p, or $GF(p)$. In this thesis only binary arithmetic is considered, where p is constrained to equal 2.

2.2 Binary Extension Field Arithmetic

$GF(2^m)$ is a finite field that contains 2^m different elements. This finite field is an extension of $GF(2)$ which contains 0 and 1. The extended binary field $GF(2^m)$, is associated with an irreducible polynomial of degree m over $GF(2)$, i.e.$P(z) = p_m z^m + p_{m-1} z^{m-1} + \ldots + p_1 z^1 + p_0 z^0$, where $p_i \in GF(2)$, and $p_o = p_m = 1$. A polynomial $P(z)$ of degree m is called primitive or irreducible if the smallest positive integer λ for which $P(z)$ divides $z^\lambda + 1$ is $\lambda = 2^m - 1$. Elements

in $GF(2^m)$ can be represented using different basis function, supporting the operations described above.

2.2.1 Basis Representation

Polynomial Basis

Let x be a primitive element of $P(z)$, i.e., $P(x) = 0$. An element, A, of $GF(2^m)$ can be represented as a polynomial of degree up to $m-1$ over $GF(2)$ as,

$$A = \sum_{i=0}^{m-1} a_i x^i, \tag{2.7}$$

where $a_i \in GF(2)$.

This representation is called the polynomial basis (PB) representation. In this case, the addition of any two elements is easily performed by the exclusive-or (XOR) operation. However, the multiplication and squaring operations are complicated as the intermediate product needs further reduction by $P(x)$.

Shifted Polynomial Basis

Assuming v is an integer, $0 < v \leq m-1$, and the set $1,\ x,\ x^2,\ \ldots,\ x^{m-1}$ is a polynomial basis for $GF(2^m)$, the Shifted Polynomial Basis (SPB) for $GF(2^m)$ is denoted as the set $x^{-v},\ x^{-v+1},\ \ldots,\ x^{m-v+1}$. Similar to the polynomial basis, it is possible to represent each field element using the SPB. For example, if A is an element of $GF(2^m)$, then it can be represented as

$$A = \sum_{i=0}^{m-1} a_i x^{i-v}, \tag{2.8}$$

where $a_i \in GF(2)$.

The addition of two field elements, represented in the SPB, is carried out by the XOR operation. However, the multiplication and squaring operations are complicated as the intermediate product needs further reduction by $P(x)$.

Normal Basis

It is shown that there exists a normal basis for the binary extension field $GF(2^m)$ for all positive integers m [14]. The normal basis is constructed by finding a field element β , where $\left\{\beta,\ \beta^2,\ \beta^4,\ \ldots\ ,\ \beta^{2(m-1)}\right\}$ is a basis for$GF(2^m)$. In this case, if $A \in GF(2^m)$, then it can be represented as,

$$A = \sum_{i=0}^{m-1} a_i \beta^{2^i}, \tag{2.9}$$

where $a_i \in GF(2)$.

Normal basis representations of field elements are especially attractive in situations where squaring is required, since squaring an element $A = a_0\beta^1 + a_1\beta^2 + a_2\beta^4 + \ldots + a_{m-1}\beta^{2(m-1)}$ is simply its cyclic right shift $A^2 = a_{m-1}\beta^1 + a_0\beta^2 + a_1\beta^4 + \ldots + a_{m-2}\beta^{2(m-1)}$.

Dual Basis

The dual basis is an important concept in finite field theory and was originally exploited to allow for the design of hardware efficient BCH and RS decoders [15]. For 2 basis functions $\{\lambda_i\}$ and $\{\mu_i\}$,define f a linear transformation function from $GF(2^m) \longrightarrow GF(2)$, and $\beta \in GF(2^m)$, $\beta \neq 0$. Then $\{\lambda_i\}$ and $\{\mu_i\}$ are dual to each other with respect to f and β if:

$$f\left(\beta\lambda_i\mu_j\right) = \left\{ \begin{array}{ll} 1 & if\ i = j \\ 0 & if\ i \neq j \end{array} \right\}, \tag{2.10}$$

In this case $\{\lambda_i\}$ is the standard and$\{\mu_i\}$ is the dual basis.

2.2.2 Operations

Addition

Let A and B be two field elements in $GF(2^m)$ represented by $(a_{m-1}, a_{m-2}, \ldots, a_0)$ and (b_{m-1}, b_{m-2}, $\ldots$, b_0), respectively in any basis representation. Now, C = A+B can be obtained by pair-wise addition of the coordinates of A and B over

$GF(2)$ (i.e., modulo 2 addition), then C can be represented as,

$$c_i = a_i \oplus b_i, \tag{2.11}$$

where $0 \leq i \leq m-1$, and $\oplus$ denotes XOR operation

Multiplication

The multiplication over $GF(2^m)$ is much more complicated than addition. This operation has been considered by researchers from different points of view. The most common approaches are based on the polynomial basis, normal basis, dual basis and the shifted polynomial basis. Each of these multiplication algorithms offers different time and area complexities, more details about multipliers will be given in Chapter 4

Multiplication By a Constant Field Element

It is frequently required to carry out multiplication by a constant field element. This can be accomplished using two-variable input multipliers. Alternatively it is often beneficial to employ a multiplier designed specifically for this task. Consider the operation $x^j * A$, where A is a finite field element, and x is a primitive root over the polynomial $P(z) = z^m + f_{m-1}z^{m-1} + \ldots + f_1 z^1 + f_0 z^0$, and $P(x) = 0$, thus x^m can be expressed as,

$$x^m = f_{m-1}x^{m-1} + \ldots + f_1 x^1 + f_0 x^0 \tag{2.12}$$

thus for simplicity $x^1 * A$ can be expressed as,

$$x^1 * A = a_0 x + a_1 x^2 \ldots + a_{m-2}x^{m-1} + a_{m-1}x^m, \tag{2.13}$$

substituting with the expression of x^m above thus,

$$\begin{aligned} x^1 * A = a_{m-1}f_0 * x^0 + (a_0 + a_{m-1}f_1) * x + (a_1 + \\ a_{m-1}f_2) * x^2 \ldots + (a_{m-2} + a_{m-1}f_{m-1}) * x^{m-1} \end{aligned} \tag{2.14}$$

The previous expression can be implemented using simple XOR operations, it is shown at [16] that in constant multiplication, the number of XOR gates is

bounded by $m^2 - \frac{m}{2}$, and its critical path is bounded by $log_2(m)$ the delay of an XOR gate.

Squaring

Squaring over $GF(2^m)$ is a special case of multiplication and as a result, requires less resources. The normal basis offers the best squaring operation which is performed by a circular right shift. The squaring in polynomial basis and shifted polynomial basis is more complicated and the complexity depends on the irreducible polynomial $P(z)$.

Inversion

Inversion over binary extension fields is considered an expensive operation. Assuming $A \in GF(2^m)$, the objective is to find a field element A^{-1}, where $AA^{-1} = 1$. Algorithms proposed for inversion are based on the fact that $A^{2m-2} = A^{-1}$. Various approaches to find A^{2m-2} are proposed for each basis function.

2.3 Summary

In this chapter an overview on Galois fields is given, the basic definitions of fields and Galois field extensions is elaborated, and the different basis representation of elements in Galois field are shown. This chapter is a preliminary to understand BCH codes structure which is based on Galois fields, for more details on Galois fields refer to [13]

Chapter 3

BCH Code

In this chapter Bose-Chaudhuri-Hocquenghem (BCH) code is introduced, BCH encoding and algebraic decoding algorithms are presented. The decoding is broken down into three processes: syndromes calculation, Berlekamp-Massey algorithm (BMA), and Chien search. This chapter will be divided as follows, a brief introduction on BCH code is described in section 3.1, the encoding procedure is described in section 3.2, and the decoding procedure is described in section 3.3.

3.1 BCH Codes

The first class of linear codes derived for error correction were Hamming codes. These codes are capable of correcting only a single error because of their simplicity. Later the generalised binary class of Hamming codes for multiple-errors was discovered by Hocquenghem in 1959 [1], and independently by Bose and Chaudhuri in 1960 [2]. Almost at the same time independently of the work of Bose, Chaudhuri and Hocquenghem, the important subclass of non-binary BCH codes RS codes were introduced by Reed and Solomon [17].

The class of BCH codes is a large class of error correction codes that occupies a prominent place in theory and practice of error correction. This prominence is a result of the relatively simple encoding and decoding techniques. Before considering BCH codes, some additional theory needs to be introduced [18].

- The minimum distance of a linear code is the minimum Hamming weight of any non-zero codeword, where the hamming weight is the number of different bits.

- A code with minimum distance d can correct $\lfloor (d-1)/2 \rfloor$ errors.
- A linear code C is cyclic if whenever $(c_0, c_1, ..., c_{n-1})$ is a codeword in C then $(c_{n-1}, c_0, c_1, ..., c_{n-2})$ is also a codeword.
- A codeword $(c0, c1, ..., cn-1)$ of a cyclic code can be represented as the polynomial $c(x) = c_0 + c_1 x + ... + c_{n-1} x^{n-1}$. This correspondence is very helpful as the mathematical background of polynomials is well developed, and so this representation is used here.

It is frequently convenient to define error-correcting codes in terms of the generator polynomial of that code $g(x)$. The generator polynomial of a t-error-correcting BCH code is defined to be the least common multiple (LCM) of $f_1, f_3, ...f_{2*t-1}$, that is,

$$g(x) = LCM f_1, f_3, f_5, ...f_{2*t-1} \tag{3.1}$$

where f_j is the minimal polynomial of α^j, and $(0 < j < 2t + 1)$.

Let f_j, where $(0 < j < 2t + 1)$, be a minimal polynomial of α^j then f_j is obtained by:

$$f_j(x) = \prod_{i=0}^{e-1} (x + \beta^{2^i}) \tag{3.2}$$

where $\beta = \alpha^j$, $\beta^{2^e} = \beta$ and $e \leq m$

To generate a codeword for an (n, k) t error-correcting BCH code, the k information symbols are formed into the information polynomial $i(x) = i_0 + i_1 x + ... + i_{k-1} x^{k-1}$ where $i_j \in GF(2)$. Then the codeword polynomial $c(x) = c_0 + c_1 x + ... + c_{n-1} x^{n-1}$ is formed as

$$c(x) = i(x) * g(x) \tag{3.3}$$

Since the degree of $f_j(x)$ is less or equal to m $(e \leq m)$, the degree of $g(x)$ (and consequently the number of parity bits $n - k$) is at most equal to $m * t$. For small values of t, the number of parity check bits is usually equal to $m * t$.

For any positive integer $m \geq 3$ there exists binary BCH codes (n, k) with the following parameters:

- $n = 2^m - 1$ the length of codeword in bits

- t the maximum number of error bits that can be corrected
- $k \geq n - m * t$ number of information bits in a codeword
- $d_{min} \geq 2 * t + 1$ the minimum distance.

Note that for $t = 1$, this construction of BCH codes generates Hamming codes. The number of parity bits equals m, and so $(2^m - 1, 2^m - m - 1)$ codes are obtained. In this case the generator polynomial $g(x)$ satisfies

$$g(x) = f_1(x) = p(x) \tag{3.4}$$

where $p(x)$ is the irreducible polynomial for $GF(2^m)$.

In this thesis only primitive BCH codes are considered. Non-primitive BCH codes have a generator polynomial g(x) with $\beta^l, \beta^{l+1}, \beta^{l+2}, \beta^{l+d-2}$

as roots, where β is an element in $GF(2^m)$ and l is a non-negative integer. Non-primitive BCH codes obtained in this way have a minimum distance of at least d. When $l = 1$, $d = 2 * t + 1$ and $\beta = \alpha$ where α is a primitive element of $GF(2^m)$, primitive BCH codes are obtained.

3.2 Encoding BCH codes

If BCH codewords are encoded the data bits do not appear explicitly in the codeword. To overcome this let

$$c(x) = x^{n-k} * i(x) + b(x) \tag{3.5}$$

where $c(x) = c_0 + c_1x + ... + c_{n-1}x^{n-1}$, $i(x) = i_0 + i_1x + ... + i_{k-1}x^{k-1}$, $b(x) = b_0 + b_1x + ... + b_{m-1}x^{m-1}$ and c_j, i_j, $b_j \in GF(2)$. Then if $b(x)$ is taken to be the polynomial such that

$$x^{n-k} * i(x) = q(x) * g(x) - b(x) \tag{3.6}$$

the k data bits will be present in the codeword [14].

BCH codes are implemented as cyclic codes, that is, the digital logic implementing the encoding and decoding algorithms is organised into shift-register circuits that mimic the cyclic shifts and polynomial arithmetic required in the

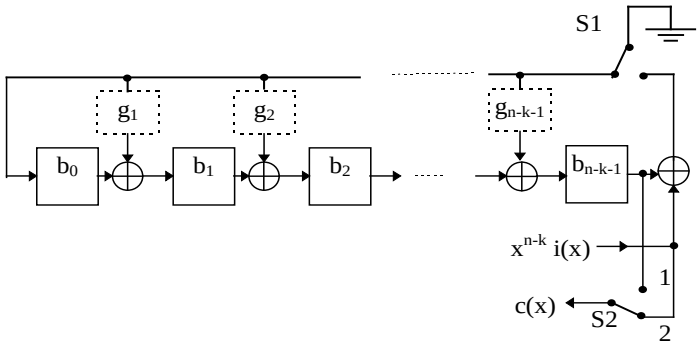

Figure 3.1: Encoding Circuit for a (n, k) BCH Code

Table 3.1: Polynomials for $(15, 5)$ 3-Error Correcting BCH Code

roots	minimal polynomial
α, α^2, α^4	$f_1(x) = (x+\alpha)(x+\alpha^2)(x+\alpha^4)(x+\alpha^8) = 1+x+x^4$
α^3, α^6	$f_3(x) = (x+\alpha^3)(x+\alpha^6)(x+\alpha^{12})(x+\alpha^9) = 1+x+x^2+x^3+x^4$
α^5	$f_5(x) = (x+\alpha^5)(x+\alpha^{10}) = 1+x+x^2$

description of cyclic codes. Using the properties of cyclic codes [18], the remainder $b(x)$ can be obtained in a linear $(n-k)$-stage shift register with feedback connections corresponding to the coefficients of the generator polynomial $g(x) = 1 + g_1 x + g_2 x^2 + ... + g_{n-k-1} x^{n-k-1} + x^{n-k}$. Such a circuit is shown in Fig. 3.1. Encoding circuit for a (n, k) BCH code.

The encoder operates as follows:

- For clock cycles 1 to k, the information bits are transmitted in unchanged form (switch $S2$ in position 2) and the parity bits are calculated in the Linear Feedback Shift Register (LFSR) (switch $S1$ is on).
- For clock cycles $k+1$ to n, the parity bits in the LFSR are transmitted (switch $S2$ in position 1) and the feedback in the LFSR is switched off ($S1$ - off).

As an example, the $(15, 5)$ 3-error correcting BCH code is considered. The generator polynomial with $\alpha, \alpha^2, \alpha^3, ..., \alpha^6$ as the roots is obtained by multiplying the minimal polynomials given in Table 3.1.

Thus the generator polynomial g(x) is given by $g(x) = f_1(x) * f_3(x) * f_5(x) = 1 + x + x^2 + x^4 + x^5 + x^8 + x^{10}$.

3.3 Decoding BCH Codes

The decoding process is far more complicated than the encoding process. As a general rule, decoding can be broken down into three separate steps:

1. Calculating the syndromes
2. Solving the key equation
3. Finding the error locations.

3.3.1 Calculation of the Syndromes

Let $c(x) = c_0 + c_1x + ... + c_{n-1}x^{n-1}$, $r(x) = r_0 + r_1x + ... + r_{n-1}x^{n-1}$ and $e(x) = e_0 + e_1x + ... + e_{n-1}x^{n-1}$ be the transmitted polynomial, the received polynomial and the error polynomial respectively so that

$$r(x) = c(x) + e(x) \tag{3.7}$$

The first step of the decoding process is to store the received polynomial $r(x)$ in a buffer register and to calculate the syndromes S_j for $1 \leq j \leq 2t$. The most important feature of the syndromes is that they do not depend on transmitted information but only on error locations.

Define the syndromes S_j as

$$S_j = \sum_{i=0}^{n-1} r_i \alpha^{i-j} \tag{3.8}$$

for $1 \leq j \leq 2t$.

Since $r_j = c_j + e_j$, for $j = 0, 1, ..., n-1$, thus

$$S_j = \sum_{i=0}^{n-1} (c_i + e_i)\alpha^{i-j} = \sum_{i=0}^{n-1} c_i \alpha^{i-j} + \sum_{i=0}^{n-1} e_i \alpha^{i-j} \tag{3.9}$$

By the definition of BCH codes $\sum_{i=0}^{n-1} c_i \alpha^{i-j} = 0$ for $1 \leq j \leq 2t$, thus

$$S_j = \sum_{i=0}^{n-1} e_i \alpha^{i-j} \tag{3.10}$$

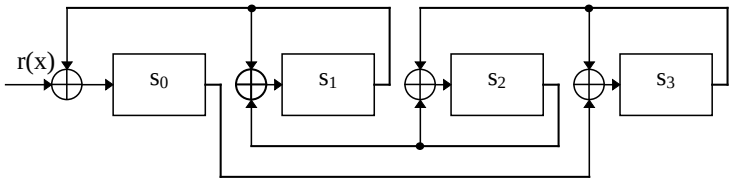

Figure 3.2: Circuit Computing S_3 for $m = 4$.

It is therefore observed that the syndromes S_j depends only on the error polynomial $e(x)$, and so if no errors occur, the syndromes will all be zero.

To generate the syndromes, express the syndrome equation as

$$Sj = (...((r_{n-1} * \alpha^j + r_{n-2}) * \alpha^j + r_{n-3}) * \alpha^j + ...) * \alpha^j + r_0 \tag{3.11}$$

Thus a circuit calculating the syndrome S_j carries out $(n-1)$ multiplications by the constant value α^j and $(n-1)$ single bit summations. Note that because $r_j \in GF(2)$ the equation $S_{2i} = S_i^2$ is met [19].

For example a circuit calculating S_3 for $m = 4$ and $p(x) = x^4 + x + 1$ is presented in Fig. 3.2. Initially the register s_i $(0 \leq i \leq 3)$ is set to zero. Then the register $s_0 - s_3$ is shifted 15 times and the received bits r_i $(0 \leq i \leq 14)$ are clocked into the syndrome calculation circuit. Then the S_3 is obtained in the $s_0 - s_3$ registers.

3.3.2 Solving the Key Equation

The second stage of the decoding process is finding the coefficients of the error location polynomial $\sigma(x) = \sigma_0 + \sigma_1 x + ... + \sigma_t x^t$ using the syndromes S_j. The relationship between the syndromes and these values of σ_j is given by

$$\sum_{j=0}^{t} S_{t+i-j}\sigma_j = 0 \tag{3.12}$$

for $i = 1, ..., t$, and the roots of $\sigma(x)$ give the error positions. The coefficients of $\sigma(x)$ can be calculated by methods such as the Peterson-Gorenstein-Zieler algorithm [20] or Euclid's algorithm [21] or Berlekamp-Massey Algorithm (BMA) [22] which is the most efficient method in practice [20].

In the BMA, the error location polynomial $\sigma(x)$ is found by $t-1$ recursive

iterations. During each iteration n, the degree of $\sigma(x)$ is usually incremented by one. Through this method the degree of $\sigma(x)$ is exactly the number of corrupted bits, as the roots of $\sigma(x)$ are associated with the transmission errors. The BMA is based on the property that for a number of iterations r greater or equal the number of errors t_a that have actually occurred ($n \geq t_a$), the discrepancy $d_n = 0$, where d_n is expressed as

$$d_n = \sum_{j=0}^{t} S_{2n-j-1}\sigma_j \tag{3.13}$$

On the other hand if $n < t_a$, the discrepancy d_n is non zero and is used to modify the degree and coefficients of $\sigma(x)$. What the BMA essentially does therefore is compute the shortest degree $\sigma(x)$ such that Eq. 3.13 holds.

The BMA with inversion is given below. Initial values:

$$\begin{aligned} d_p &= \begin{cases} 1 & , S_1 = 0 \\ S_1 & , S_1 \neq 0 \end{cases} \\ \sigma^0(x) &= 1 + S_1 x \\ \beta^1(x) &= \begin{cases} x^3 & , S_1 = 0 \\ x^2 & , S_1 \neq 0 \end{cases} \\ l_1 &= \begin{cases} 0 & , S_1 = 0 \\ 1 & , S_1 \neq 0 \end{cases} \\ n &= 1 \end{aligned} \tag{3.14}$$

The error location polynomial $\sigma(x)$ is then generated by the following set of recursive equations:

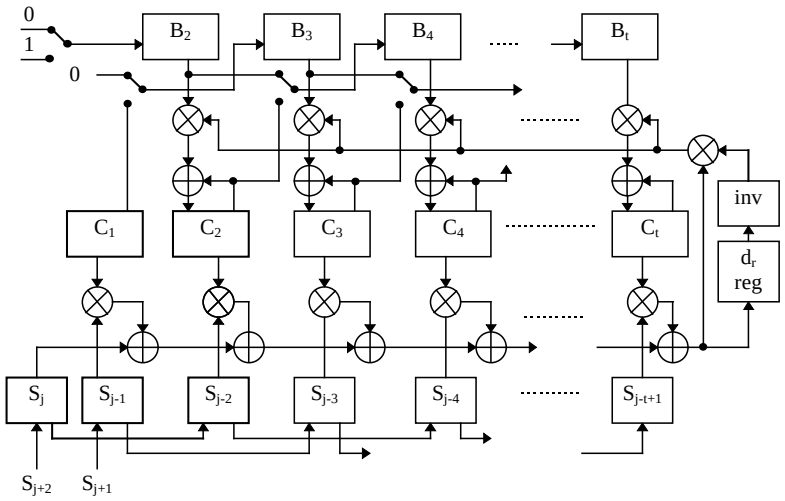

Figure 3.3: Berlekamp Massey Algorithm with Inversion

$$
\begin{aligned}
d_n &= \sum_{i=0}^{t} \sigma_i^n . S_{2n-i+1} \\
\sigma^n(x) &= \sigma^{n-1}(x) - d_p^{-1} . d_n . \beta^n(x) \\
bsel &= \begin{cases} 0 & , d_n = 0 \; or \; n < l_n \\ 1 & , d_n \neq 0 \; or \; r \geq l_n \end{cases} \\
\beta^{n+1}(x) &= \begin{cases} x^3 . \beta^n(x) & , bsel = 0 \\ x^2 . \sigma^{n-1}(x) & , bsel \neq 0 \end{cases} \\
l_{n+1} &= \begin{cases} l_n & , bsel = 0 \\ 2 * l_n - l_n + 1 & , bsel \neq 0 \end{cases} \\
d_p &= \begin{cases} d_{p-1} & , bsel = 0 \\ d_n & , bsel \neq 0 \end{cases} \\
n &= n + 1
\end{aligned}
\tag{3.15}
$$

These calculation are carried out for $n = 1, ..., t-1$. A circuit implementing the BMA is given in Fig. 3.3. The error location polynomial $\sigma(x)$ is obtained in the C registers after $t-1$ iterations.

3.3.3 Finding the Error Locations

The last step in decoding BCH codes is to find the error location numbers. These values are the reciprocals of the roots of $\sigma(x)$ and may be found simply

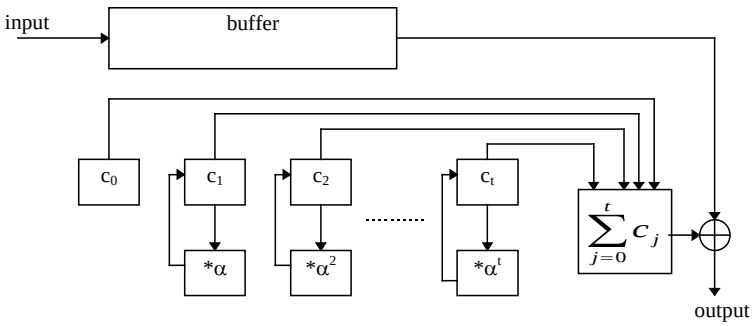

Figure 3.4: Chien's Search Circuit.

by substituting $1, \alpha, \alpha^2, ..., \alpha^{n-1}$ into $\sigma(x)$. A method of achieving this using sequential substitution has been presented by Chien [23]. In the Chien search the sum

$$\sigma 0 + \sigma_1 \alpha^j + \sigma_2 \alpha^{2j} + ... + \sigma_t \alpha^{tj} \tag{3.16}$$

for $j = 0, 1, ..., k-1$, is evaluated every clock. It can be noticed that if $\sigma(\alpha^j) = 0$, the received bit r_{n-1-j} is corrupted. Therefore if for clock cycle j the sum equals zero the received bit r_{n-j-1} should be corrected.

A circuit implementing the Chien search is shown in Fig. 3.4. The operation of this circuit is as follows, the registers $c_0, c_1, ..., c_t$ are initialized by the coefficients of the error location polynomial $\sigma_0, \sigma_1, ..., \sigma_t$. Then the sum $\sum_{j=0}^{t} c_j$ is calculated and if this equals zero, the error has been found and after being delayed in a buffer, the faulty received bit is corrected using an XOR gate. On the next clock cycle each value in the c_i register is multiplied by α^i (using a constant multiplier), and the sum is calculated again. The above operations are carried out for every transmitted information bit (that is k times).

3.4 Summary

In this chapter an overview on BCH code is given. In this thesis BCH codes soft decoders are proposed; accordingly, this chapter contains the definition of BCH codes, and basic encoding and decoding procedures, which will be later on used to indicate the performance of our proposed decoders.

Chapter 4

Soft Decoding Algorithms Survey

BCH codes [24] are powerful linear block codes that are used widely in multimedia and storage applications; this is due to the fact that BCH codes acquire strong correcting capability and can be easily encoded and decoded.

Conventional decoding of BCH codes [25] is simple and suitable for VLSI implementation. Modifications to the Berlekamp algorithm were proposed [8,26] to have an efficient implementation of the decoding algorithm. Yet hard decision decoding has limited decoding capability as it disregards channel information which is available in most of BCH code applications. Efficient usage of the channel information was studied through the years [27–38] and can be divided into three main classes. The choice of a suitable class of algorithms from these classes depends on the codeword length, message length, correcting capability and code rate. The previous factors play a major role in both the performance and complexity of the decoding procedure, in our work we focus on medium to large codes with codeword length $n > 200$ and with high code rates, $n/k > 0.9$, due to their various applications [5, 6]. The first class of decoding algorithms is called least reliable bits reprocessing techniques and will be explained in section 4.1. The second class of decoding algorithms rely on the most reliable bits reprocessing, and will be detailed in section 4.2. The third class is belief propagation decoding algorithms, and will be explained in section 4.3.

4.1 Least Reliable bits Reprocessing Decoding Algorithms

The first class of algorithms relies on the bits with the least reliable channel information. GMD [27], SEW [28], and Chase [29] algorithms produce a list

of candidate codewords by flipping the least reliable bits and then algebraically decoding the resultant bit sequence. The original algorithm proposed by Chase was modified in [30] to compute the candidate codewords in only one run. The Chase algorithm and its modification offer a trade off between performance and complexity according to the number of flipped bits in the codeword; however, the performance gain obtained is insufficient compared to its complexity.

Chase Algorithm

$Input\ :\ e\ : extra\ correctable\ bits,\ \underline{S}\ :\ recieved\ bits,$

$:\ min\ error\ weight\ =\ MAX$

$I.\ sort\ input\ reliabilities\ in\ \underline{L}$

$II.\ for\ (j = 1\ ; j \leq e\ ; j++)$

$a.\ flip\ bit\ at\ location\ l_j\ in\ \underline{S}$

$b.\ Decode\ \underline{S}\ algebraiclly\ to\ \underline{S_{decoded}}$

$c.\ caclulatethe\ error\ vector,\ the\ bit - wise\ xor$
$between\ \underline{S_{decoded}}\ in\ \underline{X}$

$d.\ Calculate\ the\ sum\ of\ reliabilities\ at\ error$
$locations\ in\ error\ weight$

$e.\ if\ error\ weight\ <\ min\ error\ weight$

$min\ error\ weight\ =\ error\ weight,\ \underline{err} = \underline{X}$

The soft decoding algorithms [31,32] are based on obtaining a valid codeword which varies from the received one at the least reliable positions only. A modification [33] to the previous algorithms allows an extra error compensation. The algorithm with one extra error compensation offers acceptable performance with low complexity for codes with high rates and medium length. Suppose an (n, k, t) BCH code has an n-bits codeword, k-bits message and t correctable bits using algebraic decoders, operating under Galois field $GF(2^m)$, where $m = \log_2(n+1)$, and α is a primitive root over the primitive polynomial $f(x)$.

Consider a message $\underline{M} = [\, m_o\ m_1\ m_2\ \ldots\ m_{k-1}\,]$ encoded to a codeword $\underline{C} = [\, c_o\ c_1\ c_2\ \ldots\ c_{n-1}\,]$, and transmitted through a channel, the received signal at the decoder is $\underline{R_l} = [\, R_{l_0}\ R_{l_1}\ R_{l_2}; \ldots\ R_{l_{n-1}}\,]$, where the magnitude of R_{l_i} represents the reliability of the received bit i. The hard decision of the received

sequence is $\underline{R} = [\, r_o \, r_1 \, r_2 \, \ldots \, r_{n-1} \,]$.

The Syndrome polynomial, $S(x) = S_1 + S_2X + S_3X^2 \cdots S_{2t}X^{2t-1}$, can be expressed as

$$S_i = R(\alpha^i) = \sum_{j=1}^{v} \left(\alpha^i\right)^{e_j} = \sum_{j=1}^{v} \left(\beta_{e_j}\right)^i \tag{4.1}$$

for $i = 1, 2...t$, where e_i is the ith error location, and $\beta_{e_j} = \alpha^{e_j}$ indicates the corresponding error locator.

The input bits reliabilities are sorted into the set $\underline{L} = [\, l_1 \; l_2 \; \ldots \; l_{2t} \,]$, where the absolute value of the reliability of bit l_i is less than that of bit l_{i+1}. From the error location vector $\underline{L}$, the error locator vector $\underline{\beta} = [\, \beta_0 \;\; \beta_1 \;\; \ldots \;\; \beta_{2t-1} \,]$ is calculated as $\beta_i = \alpha^{l_i}$.

The relation between the error locator matrix $\underline{B} = [\, \underline{\beta} \;\; \underline{\beta}^2 \;\; \ldots \;\; \underline{\beta}^{2t} \,]$, the syndrome $\underline{S}$ and the discrepancy vector $\underline{\Delta} = [\, \delta_1 \, \delta_2 \, \ldots \, \delta_{2t} \,]$ is $\underline{\Delta} = \underline{B} \times \underline{\Gamma} + \underline{S}$, which is expressed as

$$\begin{bmatrix} \beta_{l_1} & \beta_{l_2} & \cdots & \beta_{l_{2t}} \\ \beta_{l_1}^2 & \beta_{l_2}^2 & \cdots & \beta_{l_3}^2 \\ \vdots & \vdots & \cdots & \vdots \\ \beta_{l_1}^{2t} & \beta_{l_2}^{2t} & \cdots & \beta_{l_{2t}}^{2t} \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_{2t} \end{bmatrix} + \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{2t} \end{bmatrix} = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_{2t} \end{bmatrix} \tag{4.2}$$

where $\underline{\Gamma} = [\gamma_1 \; \gamma_2 \; \ldots \; \gamma_{2t} \,]$ is the error magnitude set corresponding to error bits at $\underline{L}$. In other words, γ_i represents whether an error exists at bit location l_i or not.

In [32], two decoding algorithms are proposed to solve Eq.(4.2). The first is by iterating on all error locations $\underline{\Gamma}$ and then test the resultant vector $\underline{\Delta}$, if $\underline{\Delta} = \underline{0}$, then $\underline{\Gamma}$ corresponds to error locations at $\underline{L}$, i.e if $\gamma_i = 1$ then there is an error at bit location i in $\underline{L}$. This technique is referred to as the heuristic error magnitude solver algorithm.

The second technique is to obtain the inverse of the $\underline{B}$ matrix; thus, $\underline{\Gamma}$ is calculated by solving $\underline{\Gamma} = \underline{B}^{-1}\underline{S}$. If $\underline{\Gamma}$ is a binary sequence, then $\underline{\Gamma}$ corresponds to error locations at $\underline{L}$. This technique is referred to as the Bjorck–Pereyra error magnitude solver algorithm.

These two proposed algorithm correct up to $2t$ errors in the received sequence. Figure 4.1 shows the performance of soft decoding versus the soft decoding algorithm explained at [32], while figure 4.2 shows the performance of the GMD

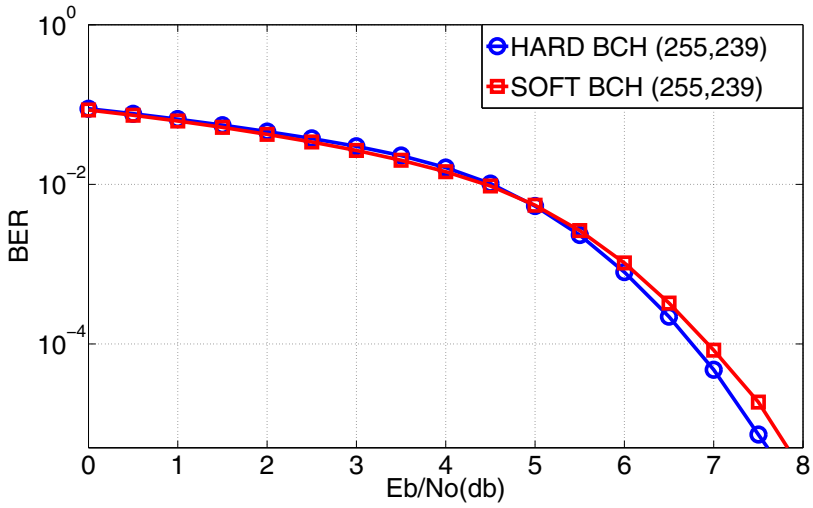

Figure 4.1: BCH code(255,239) Hard Decoding versus Soft Decoding

decoding algorithm at [27], the algorithm proposed at [33] and hard decoding.

4.2 Most Reliable Bits Reprocessing Decoding Algorithms

The second class of algorithms relies on the bits with the most reliable channel information. The motivation behind this category of algorithms is that these bits have few error bits. The correction of these few error bits would result in an error-free section of the codeword that could be used to generate the received codeword. Fossorier and Lin [34] first proposed this class of algorithms and later modifications [35, 36] decrease their time and space complexity, but the complexity of such algorithms remains prohibitively large and more suitable for short codes with low rates.

All most reliable bits reprocessing techniques have one step in common. This step is to perform permutation on the generator matrix $\underline{G}$ and the received data $\underline{r} = [r_0\, r_1 \ldots r_{n-1}]$ according to their reliability in descending order of reliability $\underline{r'} = [r'_0\, r'_1 \ldots r'_{n-1}]$ such that $|r_i| > |r_j|$ if $i > j$. The resultant generator matrix $\underline{G'}$ generated by permuting the columns of $\underline{G}$ by π_1 the reordering function performed on $\underline{r}$ to obtain $\underline{r'}$. The first k columns are not necessarily independent; thus, to obtain a generator matrix $\underline{G''}$ with the first k columns independent Gaussian elimination operation is performed on G'. The resultant function is

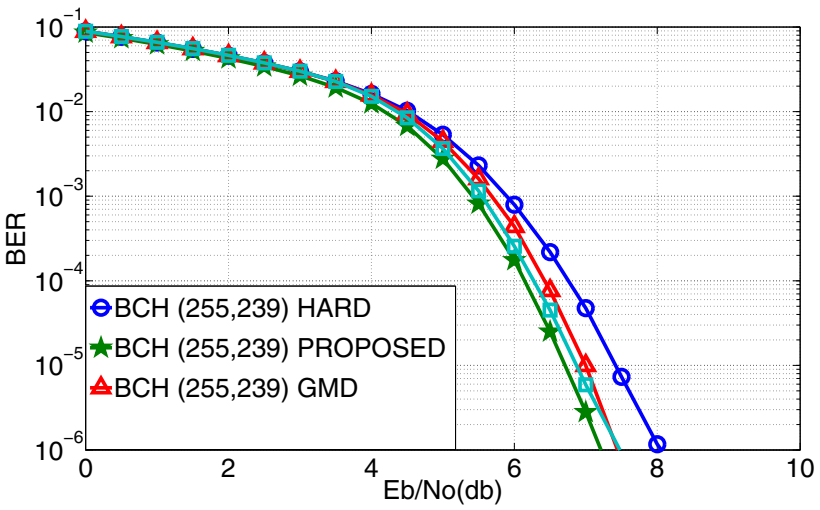

Figure 4.2: BCH code(255,239) Hard Decoding versus Soft Decoding and GMD

π_2. The generator matrix $\underline{G''}$ is in the form

$$G = [I\ P] = \begin{pmatrix} 1 & 0 & 0 & p_{11} & \cdots & p_{1(n-k)} \\ \vdots & \cdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & 1 & p_{k1} & \cdots & p_{k(n-k)} \end{pmatrix} \tag{4.3}$$

All algorithms based on most reliable bits reprocessing are variation of the one suggested at [34]. After the preprocessing stage explained earlier, the ordered statistics algorithm re-encodes the $k-$most reliable bits in $\underline{r}$ based on the fact that these location are with least probability of error; thus, the generated codeword is most likely the optimum received codeword. An order i ordered statistics algorithm flips all combination of i bits, then the resultant codeword discrepancy from the received codeword is calculated. The optimum codeword is the one with the least error weight, as shown below.

Ordered Statistics Algorithm

Input : i : *order of reprocessing*, $\underline{s}$ *recieved ordered reliabilities*, *min error weight* $= MAX$

I. sort input reliabilities into $\underline{r}$

II. hard decode $\underline{r}$ *into* $\underline{x}$

III. itterate on all error patterns $\underline{e}$ *in* i *bits of the first* k *locations of* $\underline{x}$

a. recode $\underline{x}$ *to* $\underline{v}$ *using* $\underline{G''}$

b. calculate descripancy weight of $\underline{v}$ *from* $\underline{r}$

c. if descripancy weight $<$ *min error weight*

$\quad$ *min error weight* $=$ *descripancy weight*,

$\quad$ $\underline{error} = \pi_1^{-1}[\pi_2^{-1}[\underline{e}]]$

4.3 Belief Propagation (BP) Based Algorithms

The third class of algorithms was motivated by the superior performance of belief propagation decoding for LDPC codes. The application of this algorithm directly to BCH codes resulted in poor performance due to the dense nature of their parity check matrices which resulted in large number of short cycles in their Tanner graph. This motivated the modification of the parity check matrix which was proposed in [37, 38]. The parity check matrix of the code is modified according to the reliability of the receive sequence in ascending order, so that least reliable positions correspond to sparse columns in the parity check matrix.

Adaptive Belief Propagation Algorithm

$Input\ :\ num\ : number\ of\ iterations,\ \underline{S}\ :\ recieved\ bits$

$I.\ sort\ input\ reliabilities\ in\ \underline{L}$

$II.\ for\ (j = 1\ ; j \leq num\ ; j++)$

$a.\ adapt\ the\ parity\ check\ matrix\ according\ to$
$output\ of\ stage\ j-1$

$b.\ Perform\ Belief\ propagation\ decoding\ using$
$the\ current\ parity\ check\ matrix$

$e.\ if\ all\ parity\ checks\ pass$

$S_{decoded}\ =\ output\ of\ BP\ decoder$

The resultant decoders are more complex than the belief propagation decoders, which makes them inappropriate for VLSI implementation at such code rate and word length.

4.4 Summary

In this chapter BCH soft decoding algorithms were proposed. The algorithms proposed can be classified into three major types, least significant bits reprocessing algorithms, most significant bits reprocessing algorithms and belief propagation based algorithms. As we target decoders for long codes with high rates most significant bits reprocessing algorithms and belief propagation algorithms show large complexity compared to its coding gain; thus, least reliable bits reprocessing algorithms was the most suitable class of decoding algorithms. In our work we extend the work in [31] into a programmable decoder, where the coding gain can be traded off with the decoder complexity.

Chapter 5

Galois Field Arithmetic Units Survey

Galois field operations in BCH decoding are addition, multiplication, exponentiation and inversion. With inversion being viewed as a special case of exponentiation, as will be shown later, and addition being a simple *xor* operation. The arithmetic operations performed can be classified to multiplication and exponentiation operations.

This chapter will survey the algorithms and architectures used in multiplication and exponentiation operations, and will be divided as follows, multiplication operations will be discussed in section 5.1, and exponentiations will be discussed in section 5.2.

5.1 Galois Field $GF(2^m)$ Multipliers

Multipliers can be classified according to the basis function of the elements. Another way would be according to the method of bit processing whether it is serial processed (bit serial) or parallel processed (bit parallel). A category of multipliers that exists between bit serial and bit parallel operates on groups of bits (digits). Digit serial processing of bits is a processing technique where bits are grouped into digits which are processed together as one unit. In this section we focus on basis classification, and we point out its variations according to other types of classifications.

5.1.1 Polynomial Basis Multipliers

Polynomial basis multipliers form a popular category of finite field multipliers. These multipliers are easily implemented, reasonably hardware efficient. Let A and B are two elements over $GF(2^m)$ represented using polynomial basis, their multiplication $C = AB \ mod \ F(x)$. As A and B can be represented in the polynomial basis then:

$$A = \sum_{i=0}^{m-1} a_i x^i, \ and \qquad B = \sum_{i=0}^{m-1} b_i x^i, \tag{5.1}$$

the product C is given by:

$$C = \sum_{i=0}^{m-1} c_i x^i = A \ (\ b_{m-1} \ b_{m-2} \ \ldots \ , b_o \ ), \tag{5.2}$$

this can be formulated based on the way the bits are processed, there are two kinds of polynomial basis bit-serial multipliers. They are called the Least significant bit first LSB-first and the most significant bit first MSB-first polynomial basis multipliers. In the next subsection we give detailed explanation of the operation of both these algorithms for bit serial case. For bit parallel and digit serial case, similar classification is made with a different problem formulation.

LSB-First Algorithm

In the LSB-first bit-serial algorithm, the bits are processed starting from the LSB. To obtain the LSB-first bit-serial PB multiplier, the product C can be formulated as:

$$C = b_{m-1}(Ax^m \ mod \ F(x)) + \ldots + b_1(Ax \ mod \ P(x)) + b_0(A \ mod \ P(x)). \tag{5.3}$$

The previous relation can be formulated using the LSB-first algorithm below. Equations $2 - a$, and $2 - b$ in the algorithm are depicted in figure 5.1, where $2 - a$ is calculated at the right part, and $2 - b$ is calculated at the left part.

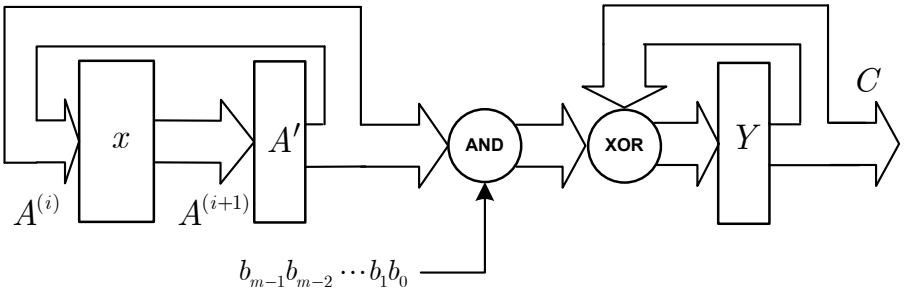

Figure 5.1: LSB-First Bit-Serial Polynomial Basis Multiplier

LSB-first bit-serial Polynomial Basis algorithm
Inputs : $A,\ B \in GF(2^m),\ F(x)$
Output : $C = A.B\ mod\ F(x)$
1 - $A^{'} = A,\ Y = 0$
2 - For $i = 0 : m-1$
a - $Y = b_i A^{'} + Y$
b - $A^{'} = A^{'}.x\ mod\ F(x)$
3 - $C = Y$

MSB-First Algorithm

The other bit-serial polynomial basis multiplier is the MSB-first bit-serial multiplier. To obtain this bit-serial multiplier, one can use the Horner's rule to calculate the product C can be formulated as:

$$C = (\ldots(b_{m-1}Ax\ mod\ P(x) + b_{m-2}A)x\ mod\ P(x) + \ldots b_1 A) \\ x\ mod\ F(x) + b_o A. \quad (5.4)$$

In this multiplication algorithm, as shown in the algorithm below, the bits are processed starting from the MSB of B. Equation $1-a$ is divided into three parts, a constant multiplication by x part, an AND gate level to perform $b_i A$, and an XOR level to perform the addition as shown in Fig. 5.2.

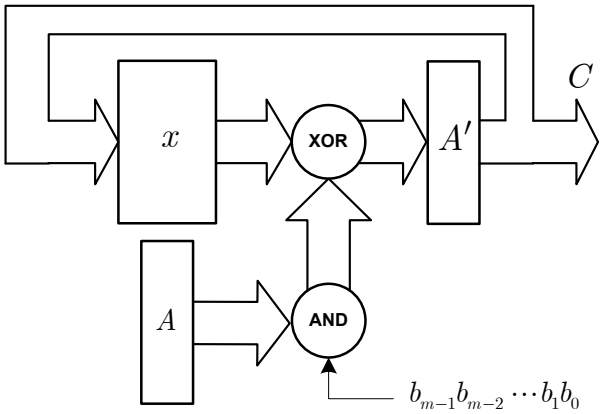

Figure 5.2: MSB-First Bit-Serial Polynomial Basis Multiplier

MSB-first bit-serial Polynomial Basis algorithm
Inputs : $A,\ B \in GF(2^m),\ F(x)$
Output : $C = A.B\ mod\ F(x)$
1 - For $i = m-1\ down\ to\ 0$
a- $A^{'} = A^{'}.x\ mod\ F(x) + b_iA$
2 - $C = A^{'}$

Logic level Implementation of Polynomial Basis Multipliers

Various polynomial basis multipliers were proposed through the years [39], [40], [41], [42] and [43]. The most efficient of those is the LSB-first algorithm proposed in [39]. Figure 5.3 shows the implementation of a $GF(2^4)$, and figure 5.4 shows a cell of the multiplier.

As shown in figure 5.4, the multiplier consists of m^2 cells, with each cell containing 2 AND gates, 2 XOR gates, and 3 1-bit latches. The critical path of the multiplier T is $T = T_{AND} + T_{XOR}$, where T_{AND} denotes the delay of an AND gate, and T_{XOR} denotes the delay of an XOR gate. The latency of this unit is m cycles.

5.1.2 Shifted Polynomial Basis Multipliers

Shifted Polynomial Basis Multipliers are similar to Polynomial Basis Multipliers, with a shift in the basis representation of the elements by x^{-v} . With the proper

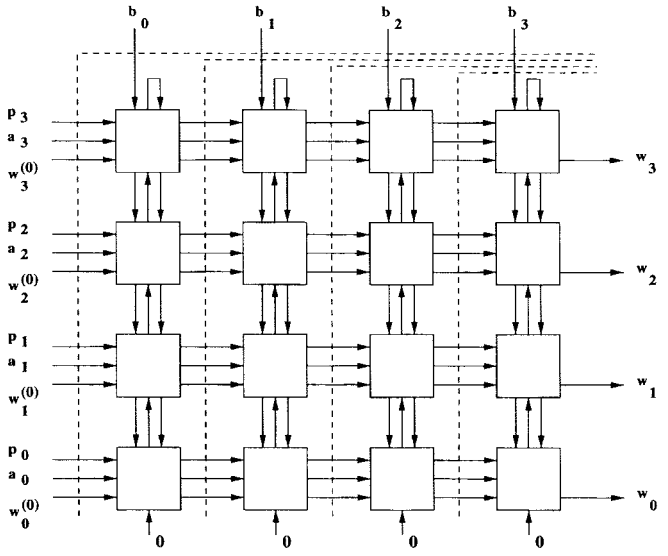

Figure 5.3: Logic Level Diagram of the LSB-First Semisystolic Array Multiplier over $GF(2^4)$

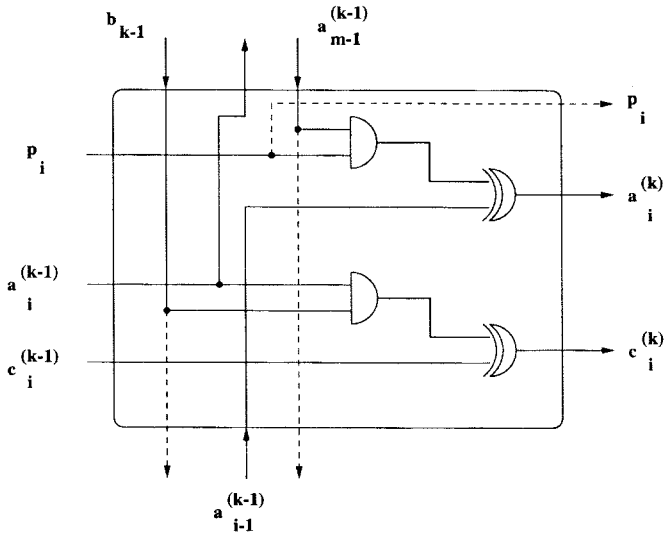

Figure 5.4: Basic Cells in the LSB-First Semisystolic Array Multiplier.

choice of v Shifted Polynomial Basis Multipliers can be formulated in a manner to reduce the latency of polynomial basis multipliers by half, as will be seen.

Let A and B be two elements over $GF(2^m)$ represented using shifted polynomial basis, their multiplication $C = AB \ mod \, P(x)$. As A and B can be represented in the shifted polynomial basis then:

$$A = \sum_{i=0}^{m-1} a_i x^{i-v}, \ and \qquad B = \sum_{i=0}^{m-1} b_i x^{i-v}, \tag{5.5}$$

the product C is given by:

$$C = \sum_{i=0}^{m-1} c_i x^i = b_o A x^{-v} + b_1 A x^{1-v} \ldots + b_{m-2} A x^{m-2-v} + b_{m-1} A x^{m-1-v} \, mod \, F(x) \tag{5.6}$$

with the choice of $v = \lfloor \frac{m}{2} \rfloor$ thus C is given by:

$$C = b_o A x^{-\lfloor \frac{m}{2} \rfloor} + b_1 A x^{1-\lfloor \frac{m}{2} \rfloor} \ldots + b_{m-2} A x^{m-2-\lfloor \frac{m}{2} \rfloor} + b_{m-1} A x^{m-1-\lfloor \frac{m}{2} \rfloor} \, mod \, F(x). \tag{5.7}$$

It is clear from the equation above that C includes two parts. One part is based on the positive powers of x and the other part is based on the negative powers of x. Thus C is given by

$$C = C' + C", \tag{5.8}$$

where

$$C' = b_o A x^{-\lfloor \frac{m}{2} \rfloor} + b_1 A x^{1-\lfloor \frac{m}{2} \rfloor} \ldots + b_{\lfloor \frac{m}{2} \rfloor - 1} A x^{-1} \, mod \, F(x), \tag{5.9}$$

and

$$C" = b_{\lfloor \frac{m}{2} \rfloor} A x^0 + b_{\lfloor \frac{m}{2} \rfloor + 1} A x^1 \ldots + b_{m-1} A x^{m-1-\lfloor \frac{m}{2} \rfloor} \, mod \, F(x). \tag{5.10}$$

The previous equations can be formulated based in a hybrid bits processing manner as will be shown in the next subsection.

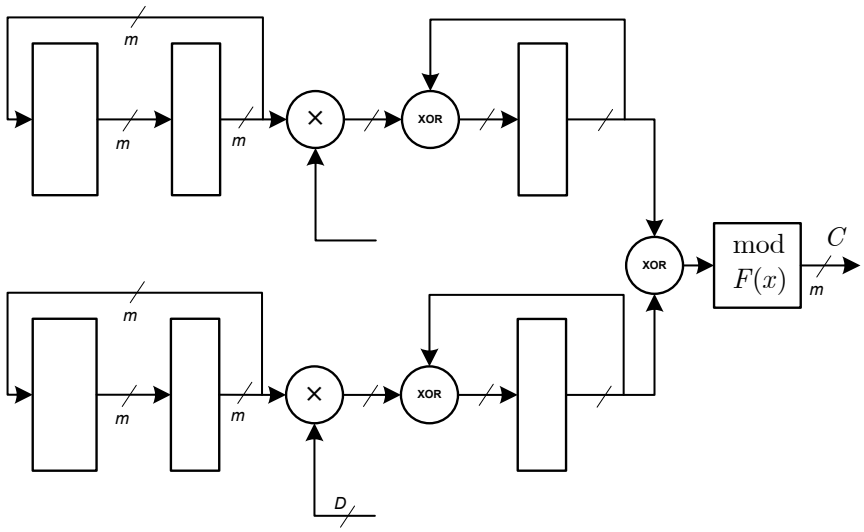

Figure 5.5: Hybrid Bit Serial Shifted Polynomial Basis Multiplier

Hybrid Bit Serial Algorithm

Using the same formulation of the equations above. The product $C = ABmod\,F(x)$ is calculated as in the algorithm below, equations 2-a, and 2-b in the algorithm are calculated in the upper part of figure 5.5, while equations 2-c, and 2-d in the algorithm are calculated in the lower part.

Hybrid bit serial Shifted Polynomial Basis algorithm
Inputs: $A,\ B \in GF(2^m),\ F(x),\ v = \lfloor \frac{m}{2} \rfloor$
Output:$C = A.B\ mod\,F(x)$
1-$A^{'} = A,\ C^{'} = 0,\ C^{"} = 0,\ A^{"} = A$
2-For $i = 0\ :\ \frac{m}{2} - 1$
a-$A^{'} = A^{'}.x^{-1}\ mod\,F(x)$
b-$C^{'} = C^{'} + b_{\frac{m}{2}-i-1}.A^{'}$
c-$C^{"} = C^{"} + b_{\frac{m}{2}+i}.A^{"}$
d-$A^{"} = A^{"}.x\ mod\,F(x)$
3-$C = C^{'} + C^{"}$

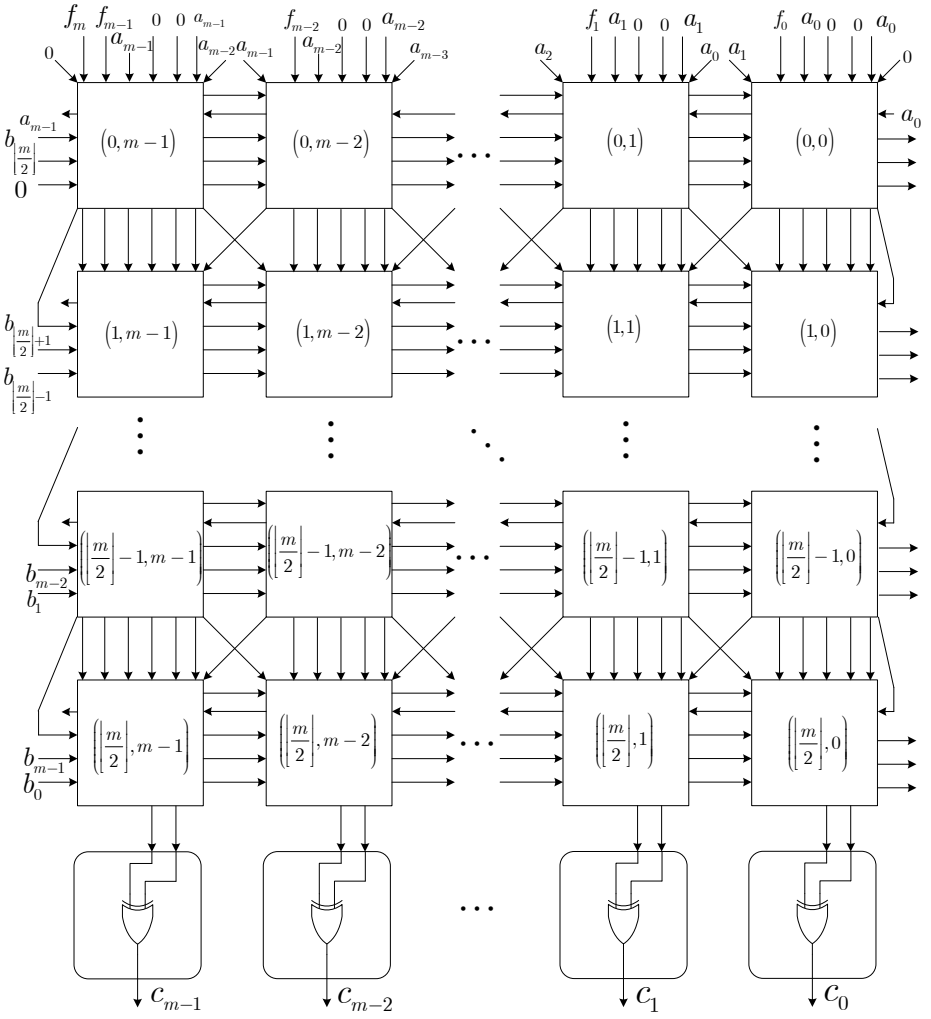

Figure 5.6: Semi-Systolic Shifted Polynomial Basis Multiplier

Logic level Implementation of Shifted Polynomial Basis Multipliers

The multiplier discussed here was proposed in [44], where the product $C = AB \, mod \, FP(x)$ is calculated using the formulation proposed above. The multiplier consists of $m * \frac{m}{2}$ cells and a final XOR level as shown in figure 5.6. Each unit consists of 4 AND, 4 XOR gates, and 5 1-bit latches for general cells. While contains 3 AND, 3 XOR gates and 5 1-bit latches for left most cells, as shown in figure 5.7. The critical path of the unit is $T = T_{AND} + T_{XOR}$, and a latency of $\frac{m}{2}$ clock cycles.

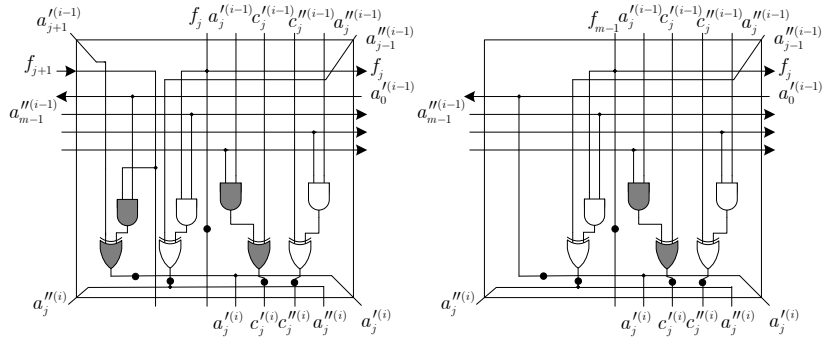

Figure 5.7: Shifted Polynomial Basis Cell (i, j), and the Leftmost Column Cells

5.1.3 Normal Basis Multipliers

Let A, and B be two elements of $GF(2^m)$ represented in normal basis. Let C be their product $C = AB \; mod \; F(x)$. Then, any coordinate of C, say c_{m-1}, is a function u of A and B which can be obtained by a matrix multiplication, i.e., $c_{m-1} = u\,(A,\, B) = A\,.\,M\,.\,B^T$, where M is a binary $m \times m$ matrix known as the multiplication matrix [45], and T denotes matrix transposition.

Massey and Omura [46] have shown that if the function $u\,(A,\, B)$ is implemented to generate c_{m-1}, then the other coordinates of C can be obtained from the same implementation with inputs appropriately shifted in cyclic fashion, more precisely, $c_{m-1-i} = u\,(A^{2^i},\, B^{2^i})$ as shown in Fig. 5.8, Which was previously shown to be equivalent to a cyclic shift. The M matrix is constant for a given field size m, and a normal basis β. The number of $1's$ in M is known as the complexity of the normal basis N and is denoted as C_N. The latter determines the gate counts and, hence, time delay for a normal basis multiplier.

Logic level Implementation Of Massey Omura Multiplier

As the implementation of Massey Omura multipliers [46] depends on the function u, an example of Massey Omura multipliers is given for $GF(2^5)$ and $\beta = x^5$. From [45]

$$M = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix},$$

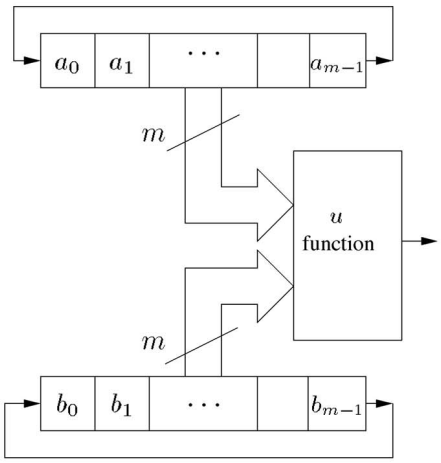

Figure 5.8: Massey Omura Multiplier

$c_4 = a_3b_3 + (a_0b_2 + a_2b_0) + (a_0b_4 + a_4b_0) + (a_1b_2 + a_2b_1) + (a_1b_3 + a_3b_1)$. Which is implemented as in figure 5.9. In general, the unit consists of m 1-bit latches, the number of AND gates and XOR gates are C_N and $C_N - 1$, respectively. Also, its critical path delay is $T_{AND} + \lceil log_2 C_N \rceil T_{XOR}$, and its latency is m clock cycles.

5.1.4 Dual Basis Multipliers

Dual basis multipliers rely on performing the multiplication of two $GF(2^m)$ elements A, B. the first A is represented in a general basis representation, and B in the dual basis of A. Two types of dual basis multipliers exist. The first is general dual basis multiplier, where A is represented in the polynomial basis representation, B, and C are in its dual basis. The second is optimal normal basis multiplier, where A,B, and C are represented in a special class of normal basis called optimal normal basis [47]. Optimal normal basis representation has a special property that it is self dual, i.e the dual basis of the polynomial representation is a normal representation of low complexity of multiplication matrix $C_N = 2m - 1$.

Let B be represented in polynomial basis, thus $B = \sum_{i=0}^{m-1} b_i x^i$, A be represented in dual basis representation, which is normal basis in case of optimal normal basis multiplier. Thus $A = \sum_{i=0}^{m-1} a_i^* x^i$. It was shown in [48] that $a_i^* = f\left(a\beta x^i\right)$ and the

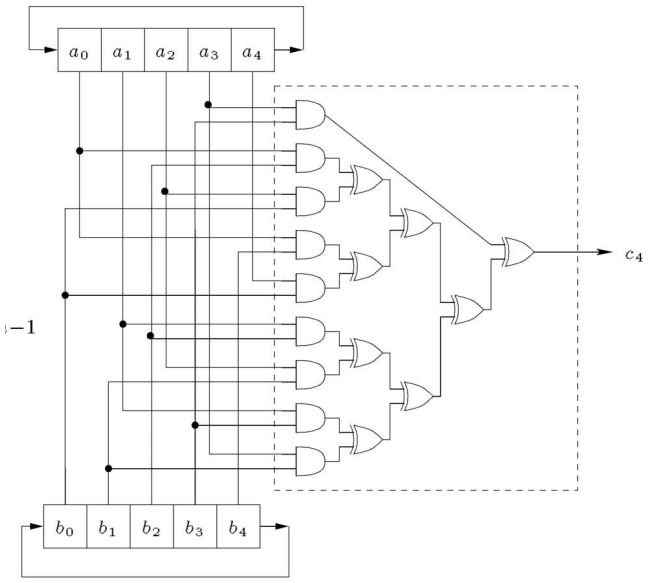

Figure 5.9: Massey Omura Multiplier for $GF(2^5)$

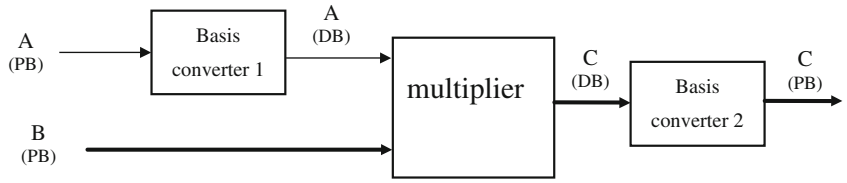

Figure 5.10: Structure of General Dual Basis Multiplier

product, $C = AB \; mod \, P(x)$ can be formulated as:

$$\begin{bmatrix} a_o & a_1 & \cdots & a_{m-1} \\ a_1 & a_2 & \cdots & a_m \\ \vdots & \vdots & \cdots & \vdots \\ a_{m-1} & a_m & \cdots & a_{2m-2} \end{bmatrix} \begin{bmatrix} b_o \\ b_1 \\ \vdots \\ b_{m-1} \end{bmatrix} = \begin{bmatrix} c_o \\ c_1 \\ \vdots \\ c_{m-1} \end{bmatrix},$$

where $a_{m+k} = f\left(a\beta x^{m+k}\right) = \sum_{i=0}^{m-1} P_i a_{i+k}$, for $k = 0, 1, \ldots, m-1$. Fig. 5.10, and Fig. 5.11 show the structure of a general dual basis multiplier, and optimal normal basis multiplier.

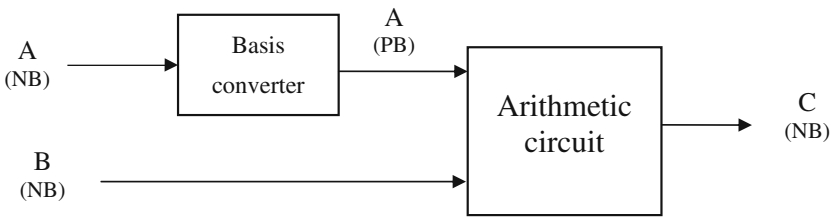

Figure 5.11: Structure of Optimal Normal Basis Multiplier

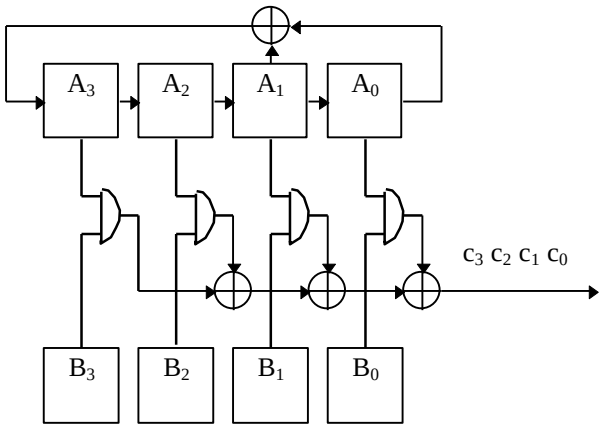

Figure 5.12: General Dual Basis Multiplier

General Dual Basis Multiplier

The matrix multiplication is performed using an AND level followed by an XOR level. the transformation of the elements of a_i to $a_i^*$is performed using a linear feedback shift register. Fig. 5.12 shows the structure of a $GF(2^4)$ multiplier from [48].

5.1.5 Comparison and Discussion

In this section we show a comparison between various pipelined multipliers, with throughput 1 operation per cycle, proposed in open literature and discuss their performance. As shown in table 5.1 various multipliers are proposed, shifted polynomial basis multiplier proposed in [15] is superior compared to other multipliers. As it has almost the same gate count compared to the polynomial basis multiplier proposed in [39], with almost half its latency. Even though multiplier in [41] has the best performance, yet, it is not widely used in cryptography due

to the absence of inversion, exponentiation, division units in shifted polynomial basis, this requires a basis conversion circuit which causes additional hardware and latency. Normal basis multipliers are widely used in cryptographic applications where squaring is of frequent manner, due to its very low cost. A less pipelined version of these multipliers can be used in cases the frequency of operation of the design is bottlenecked by another unit, thus, increasing the critical path of the multiplier unit, by removing latch levels from the design, reduces the hardware complexity of the multiplier and reduces its latency.

5.2 Galois Field $GF(2^m)$ Exponentiation

The exponentiation operation $Y = \alpha^N$ can be considered as the main operation for inversion, division and exponentiation where in case of inversion $N = -1$, and in case of division $C = AB^{-1}$. Conventionally, exponentiation is performed using the following operations, let the exponent N be expressed as

$$N = n_0 + n_1 2^1 + n_2 2^2 ... n_{m-1} 2^{m-1} \tag{5.11}$$

where the maximum exponent is 2^{m-1} as $\alpha^{2^{m-1}} = 1$; thus, N is represented in m bits. Thus $Y = \alpha^N$ can be expressed as

$$\begin{aligned} Y &= (\alpha^{n_0})(\alpha^{n_1})^{2^1}(\alpha^{n_2})^{2^2}...(\alpha^{n_{m-1}})^{2^{m-1}} \\ &= (\alpha^{n_0})[(\alpha^{n_1})(\alpha^{n_2})^{2^1}...(\alpha^{n_{m-1}})^{2^{m-2}}]^2 \\ &= (\alpha^{n_0})[(\alpha^{n_1})[(\alpha^{n_2})...(\alpha^{n_{m-1}})^{2^{m-3}}]^2]^2 \\ &= .... \\ &= (\alpha^{n_0})[(\alpha^{n_1})[(\alpha^{n_2})[...(\alpha^{n_{m-2}})(\alpha^{n_{m-1}})^2]...^2]^2 \end{aligned} \tag{5.12}$$

Accordingly, the exponentiation can be performed using the algorithm below, as shown in figure 5.13

Table 5.1: Comparison Between Multipliers

Basis Representation	# of cells	cell component	critical path	latency
Polynomial basis [39]	m^2	XOR : 2 AND : 2 $Latch$: 3	T_{AND}+T_{XOR}	m
Polynomial basis [40]	m^2	$3XOR$: 2 AND : 1 $Latch$: 7	T_{AND}+T_{3XOR}	$3m$
Polynomial basis [41]	m^2	XOR : 2 AND : 2 $Latch$: 7	T_{AND}+T_{XOR}	$3m$
Polynomial basis [42]	m^2	XOR : 1 AND : 1 $Latch$: 5	T_{AND}+T_{XOR}	$4m$
Polynomial basis [43]	m^2	$3XOR$: 1 AND : 2 $Latch$: 3	T_{AND}+T_{3XOR}	m
Shifted polynomial basis [44]	m ($\lfloor \frac{m}{2} \rfloor$ + 1)plus m XOR gates	XOR : 4 AND : 4 $Latch$: 5	T_{AND}+T_{XOR}	$\lfloor \frac{m}{2} \rfloor + 2$
Normal basis [46]	1	XOR : $C_N - 1$ AND : C_N $Latch$: $2m$	T_{AND} + $\lceil log_2 C_N \rceil T_{XOR}$	m
Normal basis [49]	1	XOR : m AND : C_N $Latch$: $2m$	T_{AND} + $\lceil log_2 C_N \rceil T_{XOR}$	m
General Dual basis [48]	m^2	XOR : 2 AND : 2 $Latch$: 8	T_{AND} + T_{XOR}	$3m$
Optimal Normal basis [50]	m^2 plus m XOR gates	$3XOR$: 1 AND : 2 $Latch$: 5	T_{AND}+T_{3XOR}	$m+1$

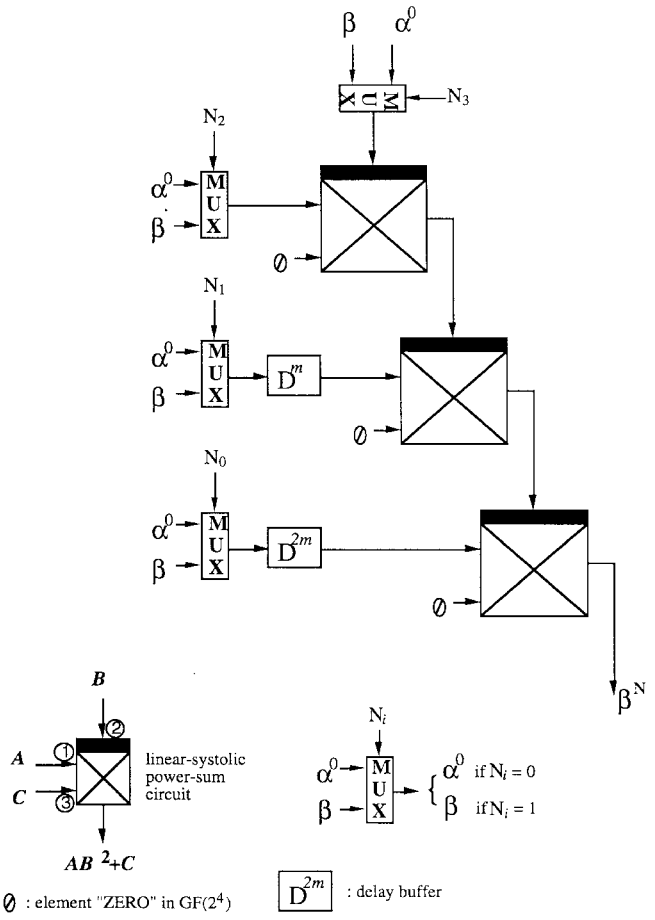

Figure 5.13: Exponentiation Using Power Sum Unit

Exponentiation Algorithm
$Input\ :\ Y = 1$
$for\ (i = m-1\,;i \geq 0\,;i--)$
$if(n_i = 1)\, Y = Y^2\alpha\ else\, Y = Y^2$

Power sum operation is the basic operation in exponentiation, and can be expressed as $P = AB^2 + C$. The complexity of the power sum operation is a critical matter, as it is multiplied by the field size m due to m repetitions of the operations. In general 2 approaches were proposed to perform power sum operation, polynomial basis power sum operation, and normal basis power sum operation.

5.2.1 Polynomial Basis Power Sum Operation

let A, B and C be polynomial basis elements of $GF(2^m)$. Thus $B^2 = [b_0 + b_1\alpha + b_2\alpha^2 ....b_{m-1}\alpha^{m-1}]^2$ is expressed as

$$B^2 = b_0 + b_1\alpha^2 + b_2\alpha^4 ...b_{m-2}\alpha^{2m-4} + b_{m-1}\alpha^{2m-2} \tag{5.13}$$

Thus the power sum product $P = p_0 + p_1\alpha + p_2\alpha^2 ...p_{m-1}\alpha^{m-1}$ is expressed as

$$\begin{aligned} P &= \sum_{k=0}^{m-1}(A.\alpha^{2k}).b_k + \sum_{n=0}^{m-1} c_n\alpha^n \\ &= \sum_{k=0}^{m-1} Q(k).b_k + \sum_{n=0}^{m-1} c_n\alpha^n \\ &= \sum_{k=0}^{m-1}(\sum_{n=0}^{m-1} q_n^{(k)}\alpha^n).b_k + \sum_{n=0}^{m-1} c_n\alpha^n \\ &= \sum_{n=0}^{m-1}(\sum_{k=0}^{m-1}(q_n^{(k)}b_k + c_n)\alpha^n) \end{aligned} \tag{5.14}$$

With initial values of $q_n^{(0)} = a_n$, Q can be represented as

$$\begin{aligned} Q &= (A.\alpha^{2k-2})\alpha^2 \\ &= q_{m-1}^{(k-1)}\alpha^{m+1} + q_{m-2}^{(k-1)}\alpha^m + \sum_{n=2}^{m-1} q_{n-2}^{k-1}\alpha^n \end{aligned} \tag{5.15}$$

Using modulo-polynomials

$$\begin{aligned} \alpha^m &= f_0 + f_1\alpha + f_2\alpha^2 ...f_{m-1}\alpha^{m-1} \\ \alpha^{m+1} &= f_0\alpha + f_1\alpha^2 + f_2\alpha^3 ...f_{m-1}\alpha^m \\ &= g_0 + g_1\alpha + g_2\alpha^2 ...g_{m-1}\alpha^{m-1} \end{aligned} \tag{5.16}$$

After substitution Q becomes as follows

$$q_n^k = q_{n-2}^{k-1} + q_{m-1}^{k-1}g_n + q_{m-2}^{k-1}f_n \tag{5.17}$$

The above algorithm can be realized according to [51] using the semi systolic

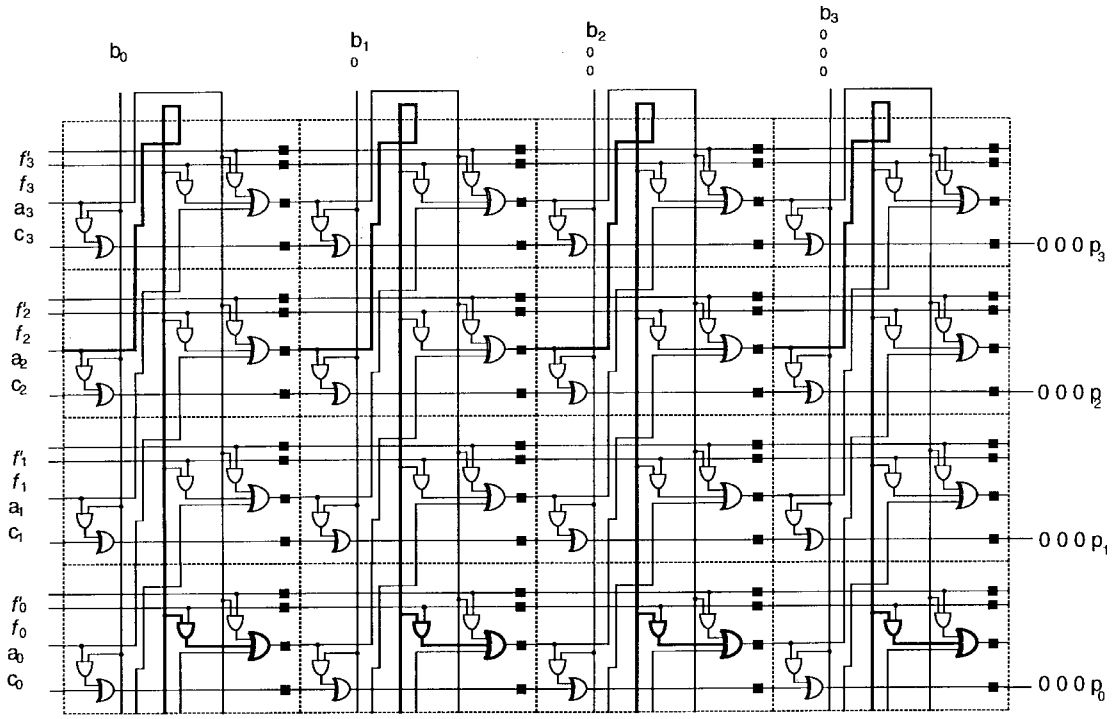

Figure 5.14: Power Sum Unit for GF(2^4)

implementation shown in figure 5.14, where each cell is as shown in figure 5.15.

5.2.2 Normal Basis Power Sum Operation

In case of formulating the power sum operation as two successive operations, a squaring operation followed by a multiplication operation the optimum basis function for such operation would be normal basis. As mentioned earlier squaring is merely a shift operation in normal basis. The block diagram of a normal basis power sum unit is as shown in figure 5.16, where the multiplier used could be any multiplier explained earlier in section 5.1.

5.2.3 Comparison and Discussion

As a comparison between exponentiation units based on polynomial basis power sum at [51] and exponentiation units based on normal basis power sum units, we will choose the normal basis multiplier at [48]. The table 5.2 compares both approaches, it is shown that the polynomial basis exponentiation unit is superior to the normal basis one in terms of area and latency.

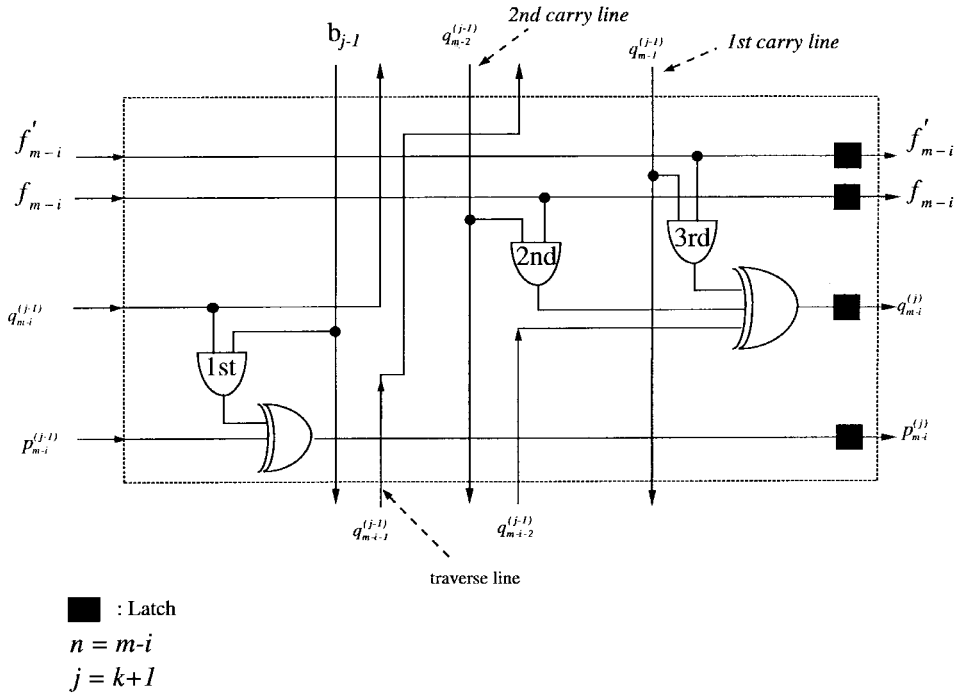

Figure 5.15: Power Sum Cell

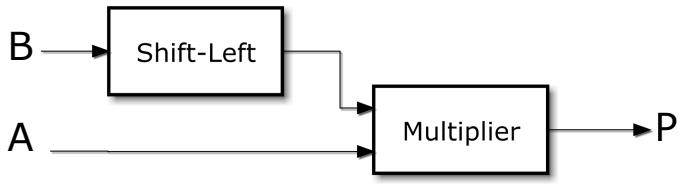

Figure 5.16: Power Sum Cell Using Normal Basis

Table 5.2: Comparison Between Exponentiation Units

Basis Representation	# of cells	cell component	critical path	latency
Polynomial basis [51]	$(m-1)m^2$	XOR : 2 AND : 3 $Latch$: 4	$T_{AND} + T_{XOR3}$	$m(m-1)$
Normal basis [48]	$(m-1)m^2$	XOR : 2 AND : 2 $Latch$: 8	$T_{AND} + T_{XOR}$	$3m(m-1)$

5.3 Summary

In this chapter a survey on Galois field arithmetic units was done. As will be shown later the proposed decoders requires addition, multiplication and power sum units. Accordingly, the most suitable basis representation for elements would be the polynomial basis. This returns to the efficiency of the multiplication using the unit proposed in [39] and the power sum unit proposed in [51].

Chapter 6

Proposed BCH Soft Decoding Algorithms

In this chapter, we propose two least reliable bit reprocessing algorithms. The choice of this class of algorithms is due to their low complexity and high performance gain for high-rate codes, with medium to large codeword length. The proposed algorithms are based on the algorithms in [32, 33] and use a variation proposed in [31]. These algorithms provide a trade off between complexity and performance, and are feasible to be implemented in programmable hardware. This rest of the chapter is organized as follows. The two proposed algorithms are shown in sect. 6.1. The simulation results are provided in section 6.2.

6.1 Proposed Programmable BCH Soft Decoding Algorithm

Suppose an $(n\,, k\,, t\,)$ BCH code has an n-bits codeword, k-bits message and t correctable bits using algebraic decoders, operating under Galois field $GF(2^m)$, where $m = \log_2(n+1)$, and α is a primitive root over the primitive polynomial $f(x)$.

Consider a message $\underline{M} = [\, m_o \;\; m_1 \;\; m_2 \;\; \ldots \;\; m_{k-1}\,]$ encoded to a codeword $\underline{C} = [\, c_o \, c_1 \, c_2 \;\; \ldots \;\; c_{n-1}\,]$, and transmitted through a channel, the received signal at the decoder is $\underline{R_l} = [\, R_{l_0} \;\; R_{l_1} \;\; R_{l_2}; \ldots \;\; R_{l_{n-1}}\,]$, where the magnitude of R_{l_i} represents the reliability of the received bit i. The hard decision of the received sequence is $\underline{R} = [\, r_o \, r_1 \, r_2 \;\; \ldots \;\; r_{n-1}\,]$.

The Syndrome polynomial, $S\,(x) = S_1 + S_2X + S_3X^2 \cdots \;\; S_{2t}X^{2t-1}$, can be

expressed as

$$S_i = R\left(\alpha^i\right) = \sum_{j=1}^{v} \left(\alpha^i\right)^{e_j} = \sum_{j=1}^{v} \left(\beta_{e_j}\right)^i \tag{6.1}$$

for $i = 1, 2...t$, where e_i is the ith error location, and $\beta_{e_j} = \alpha^{e_j}$ indicates the corresponding error locator.

The input bits reliabilities are sorted into the set $\underline{L} = [\, l_1 \; l_2 \; \ldots \; l_{2t} \,]$, where the absolute value of the reliability of bit l_i is less than that of bit l_{i+1}. From the error location vector $\underline{L}$, the error locator vector $\underline{\beta} = [\, \beta_0 \;\; \beta_1 \;\; \ldots \;\; \beta_{2t-1} \,]$ is calculated as $\beta_i = \alpha^{l_i}$.

The relation between the error locator matrix $\underline{B} = [\, \underline{\beta} \;\; \underline{\beta}^2 \;\; \ldots \;\; \underline{\beta}^{2t} \,]$, the syndrome $\underline{S}$ and the discrepancy vector $\underline{\Delta} = [\, \delta_1 \; \delta_2 \; \ldots \; \delta_{2t} \,]$ is $\underline{\Delta} = \underline{B} \times \underline{\Gamma} + \underline{S}$, which is expressed as

$$\begin{bmatrix} \beta_{l_1} & \beta_{l_2} & \cdots & \beta_{l_{2t}} \\ \beta_{l_1}^2 & \beta_{l_2}^2 & \cdots & \beta_{l_3}^2 \\ \vdots & \vdots & \cdots & \vdots \\ \beta_{l_1}^{2t} & \beta_{l_2}^{2t} & \cdots & \beta_{l_{2t}}^{2t} \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_{2t} \end{bmatrix} + \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{2t} \end{bmatrix} = \begin{bmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \delta_{2t} \end{bmatrix} \tag{6.2}$$

where $\underline{\Gamma} = [\gamma_1 \; \gamma_2 \; \ldots \; \gamma_{2t} \,]$ is the error magnitude set corresponding to error bits at $\underline{L}$. In other words, γ_i represents whether an error exists at bit location l_i or not.

In [32], two decoding algorithms are proposed to solve Eq.(6.2). The first is by iterating on all error locations $\underline{\Gamma}$ and then test the resultant vector $\underline{\Delta}$, if $\underline{\Delta} = \underline{0}$, then $\underline{\Gamma}$ corresponds to error locations at $\underline{L}$, i.e if $\gamma_i = 1$ then there is an error at bit location i in $\underline{L}$. This technique is referred to as the heuristic error magnitude solver algorithm.

The second technique is to obtain the inverse of the $\underline{B}$ matrix; thus, $\underline{\Gamma}$ is calculated by solving $\underline{\Gamma} = \underline{B}^{-1}\underline{S}$. If $\underline{\Gamma}$ is a binary sequence, then $\underline{\Gamma}$ corresponds to error locations at $\underline{L}$. This technique is referred to as the Bjorck–Pereyra error magnitude solver algorithm.

These two proposed algorithms correct up to $2t$ errors in the received sequence. In our developed algorithm we increase the correction capability to p extra error locations outside the set $\underline{L}$ with the least reliabilities; thus, the total error correction capability is $2t + p$. The number of extra corrected bits p is a configurable number, chosen according to the required correcting capability

versus the decoder complexity.

6.1.1 Soft Decoding Algorithms

We define the incremental syndrome $\underline{\Delta S}_j$ as the difference in the syndrome due to flipping a bit at location il_j from the received polynomial $R(x)$, where $\underline{iL} = [\, il_1\, il_2\, \cdots\, il_p\,]$ is the incremental error location vector, and the values il_j are the locations with the least reliability outside the $2t$ least reliable locations represented in the set $\underline{L}$, these extra locations represent the increase in the code correcting capability. The incremental syndrome can be expressed by

$$\Delta S_{j_i} = R^{'}\,(\alpha^i\,) - R\,(\alpha^i\,) = \beta^i_{il_j} \tag{6.3}$$

where $R^{'}(x)$ is the received sequence after flipping il_j. Similarly define the extended error magnitude matrix $\underline{\Gamma_{ext}} = [\underline{\Gamma}\ \ \underline{\Gamma}_{l_1}\ \ \underline{\Gamma}_{l_2}\ \ \ldots\ \ \underline{\Gamma}_{l_p}\,]$, where $\underline{\Gamma_{l_j}} = [\gamma_{l_{j_1}}\, \gamma_{l_{j_2}}\, \ldots\, \gamma_{l_{j_{2t}}}\,]$ is the error magnitude set for the received sequence after flipping il_j. Eq.(6.2) will be reformulated to accommodate for the extra correcting capability in both heuristic search, and Bjorck–Pereyra error magnitude solver algorithms. The newly formulated decoding algorithms will be referred to as the extended heuristic search error magnitude solver algorithm (EHe-EMS) and the extended Bjorck–Pereyra error magnitude solver algorithm (EBP-EMS).

6.1.2 EHe-EMS Algorithm

In the extended heuristic error magnitude solver, Eq.(6.2) is extended to correct p extra bits by heuristically searching for all error combinations in the error location vectors $\underline{L}$ and $\underline{iL}$. This is performed by iterating on all error locations in $\underline{L}$ 2^p times. At each time we calculate the effective syndrome at each iteration, where the effective syndrome is the syndrome of the original pattern adding to it the incremental syndrome S_{j_i}:

$$\underline{S}_{eff} = \underline{S} + \sum_{i=1}^{p} a_i\, \underline{\Delta S}_{j_i}, \tag{6.4}$$

where a_i is the binary value representing the combination of incremental syndrome vectors, corresponding to extra errors in $\underline{iL}$, and $j = 0\,, 1 \cdots 2^p - 1$ represents all the possible combinations of error at $\underline{iL}$.

This is equivalent to searching for all error combinations in $\underline{L}$, for all combinations of extra errors in $\underline{iL}$.

The discrepancy vector can be formulated as

$$\underline{\Delta} = \underline{B} \times \underline{\Gamma} + \underline{S}_{eff} \tag{6.5}$$

A valid codeword is obtained when $\underline{\Delta} = \underline{0}$, representing error bits at locations $[\gamma_1\ \gamma_2\ \ldots\ \gamma_{2t}]$ in the error location vector $\underline{L}$, and error bits at locations $\underline{A} = [a_1\ a_2\ \ldots\ a_p]$ from the incremental error location vector $\underline{iL}$. For heuristic error magnitude solver, the correcting capability could be increased for an extra bit of correcting capability resulting in a correcting capability of $2t + p + 1$ if $\underline{\Delta}$ is checked for a geometric sequence. The presence of a geometric sequence $\underline{\Delta} = [\beta_x\ \beta_x^2\ \ldots\ \beta_x^{2t}]$ indicates an extra error at location x, as shown in [33].

This search for a valid error pattern may result in more than one allowed error pattern for the received codeword. The receiver chooses the codeword with the minimum error vector weight, calculated by summing the reliabilities of the error locations. It is known that $S_1^2 = S_2$, $S_2^2 = S_4$, $\ldots$,$S_t^2 = S_{2t}$; thus, a simplification can be employed to calculate only the odd syndromes and thus the simplified version of the Algorithm can be formulated into solving $\underline{\Delta_{odd}} = \underline{B_{odd}} \times \underline{\Gamma} + \underline{S_{odd}}_{eff}$ as shown in algorithm I.

Algorithm I - EHe-EMS Algorithm

Input : $\underline{B_{odd}}$, $\underline{S_{odd}}_{ext}$, $\underline{\Gamma} = \underline{0}$,
$\underline{S_{odd}}_{eff} = \underline{S_{odd}}, min\ weight = max$

1. $\underline{A} = \underline{0}$
2. $\underline{\Delta_{odd}} = \underline{B_{odd}} \times \underline{\Gamma} + \underline{S_{odd}}_{eff}$
3. $if\ \underline{\Delta_{odd}} = \underline{0}$
   $if\ error\ weight\ < min\ weight$
   $error\ bits\ at\ locations\ \underline{\Gamma}\ from\ \underline{L}$,
   $and\ \underline{A}\ from\ \underline{iL}$
4. $else\ if\ \underline{\Delta_{odd}} = [\,\beta_x\ \beta_x^3\ \ldots\ \beta_x^{2t-1}\,]$
   $if\ error\ weight\ < min\ weight$
   $error\ bits\ at\ locations\ \underline{\Gamma}\ from\ \underline{L}$, $\underline{A}$
   $from\ \underline{iL}$, $and\ extra\ error\ at\ x$
5. $\underline{\Gamma} = \underline{\Gamma} + 1$
6. $if\ \underline{\Gamma}\ < 2^{2t} - 1\ go\ to\ (2)\ else\ go\ to\ (7)$
7. $\underline{A} = \underline{A} + 1,\ \underline{S_{eff}} = \underline{S} + \sum_{i=0}^{p-1} a_i\ \underline{\Delta S_i}$
8. $if\ \underline{A}\ < 2^p - 1\ go\ to\ (1)\ else\ exit$

6.1.3 EBP-EMS Algorithm

Although the EHe-EMS algorithm provides a high performance gain by compensating an extra error location outside the $2t + p$ correcting capability, the complexity of the decoding algorithm grows exponentially with the error correcting capability t, and the number of extra compensated bits p. Also, the extra error compensation requires a look-up table. The size of this look-up table increases exponentially with the codeword size. In order to mitigate the complexity of EHe-EMS, EBP-EMS is proposed. The complexity of EBP-EMS grows linearly with t and p, and does not require an additional look-up table.

A matrix with geometric progression in each row or column is called a Vandermonde matrix; an $n \times n$ Vandermonde matrix has the form

$$\begin{bmatrix} 1 & X_1 & X_1^2 & \cdots & X_1^{n-1} \\ 1 & X_2 & X_2^2 & \cdots & X_2^{n-1} \\ 1 & \vdots & \vdots & \cdots & \vdots \\ 1 & X_{n-1} & X_{n-1}^2 & \cdots & X_{n-1}^{n-1} \end{bmatrix} \tag{6.6}$$

As the error locator matrix $\underline{B}$ is a Vandermonde matrix, an inverse can be obtained using the Bjorck–Pereyra matrix solution method [26, 52], where instead of $O(n^3)$ computations required to solve a system of n equations, the BP algorithm requires only $O(n^2)$ computations.

In order to extend this method to compensate for extra p error locations, Eq.(6.2) is solved $p+1$ times using the BP matrix solution method, at each time obtaining a resultant error magnitude vector $\underline{\Gamma_{l_i}}$, where $i = 0\ 1\ \ldots\ p$. For $i = 1\ \ldots\ p$, Eq.(6.2) is reformulated to

$$\underline{B} \times \underline{\Gamma_{l_i}} + \Delta S_i = \underline{0}, \tag{6.7}$$

where ΔS_i is the incremental syndrome, and $i = 0$ in the equation above is the case where $\underline{\Delta S_i} = \underline{S}$ the original syndrome of the received codeword. After the $p+1$ iterations the resultant error magnitude vectors for the extended error magnitude vector matrix $\underline{\Gamma_{ext}}$ are used to obtain the error locations as follows

$$\underline{\Delta} = \underline{\Gamma} + \sum_{i=1}^{2^p-1} b_i\, \underline{\Gamma_{l_i}} \tag{6.8}$$

where b_i is a binary value. If the discrepancy vector $\underline{\Delta} \in GF\,(2)$ then a solution is obtained with error bits at locations $[\gamma_1\, \gamma_2\, \ldots\, \gamma_{2t}\,]$ in the error location vector $\underline{L}$, and error bits at locations $\underline{B} = [\,b_1\ b_2\ \ldots\ b_p\,]$ from the incremental error location vector $\underline{iL}$. The receiver chooses the codeword with the minimum error vector weight,as shown in Algorithm II.

Algorithm II - EBP-EMS algorithm

$Input\ :\ \underline{\beta},\ \underline{S_{ext}},\ \underline{\Gamma_{ext}} = \underline{0},\ \underline{B} = \underline{0}$

$\underline{\Gamma}_{ext} = \underline{S_{ext}},\ min\ weight = max$

$a. for\ (j = 1\ ; j \leq p + 1\ ; j + +)$

1. $for\ (k = 1\ ; k < 2t\ ; k + +)$

$for\ (i = 2t\ ; i > k\ ; i - -)$

$\Gamma_{j,i} = \Gamma_{j,i} - \beta_k \Gamma_{j,i-1}$

2. $for\ (k = 2t - 1\ ; k > 0\ ; k - -)$

$for\ (i = k + 1\ ; i \leq 2t\ ; i + +)$

$\Gamma_{j,i} = \Gamma_{j,i}/(\beta_i - \beta_{i-k})$

$for\ (i = k\ ; i < 2t\ ; i + +)$

$\Gamma_{j,i} = \Gamma_{j,i} - \Gamma_{j,i+1}$

3. $for\ (k = 1\ ; k \leq 2t\ ; k + +)$

$\Gamma_{j,i} = \Gamma_{j,k}/\beta_k$

$b.\ \underline{\Delta} = \underline{\Gamma} + \sum_{i=0}^{p-1} b_i\, \underline{\Gamma}_{l_i}$

1. $if\ \underline{\Delta} = \underline{0}$

$if\ error\ weight\ < min\ weight$

$error\ bits\ at\ locations\ \underline{\Gamma}\ from\ \underline{L}\ ,$

$and\ \underline{B}\ from\ \underline{iL}$

2. $\underline{B} = \underline{B} + 1$

3. $if\ \underline{B}\ < 2^p - 1\ go\ to\ (b)\ else\ exit$

6.1.4 Geometric Interpretation of the Proposed Algorithms

Consider a unity n-dimensional cube. The set of vertices ψ for this cube defines all the possible binary combinations of an n-dimensional vector. A reduced set Φ, where $\Phi \subset \psi$, is a set that contains only the valid codewords for a given BCH code (n, k, t). The received codeword $\underline{R}$ is a point in the n-dimensional space. A decoding algorithm chooses a vertex from Φ satisfying a certain criterion depending on the decoding algorithm itself.

A maximum likelihood decoding algorithm calculates the distance between $\underline{R}$

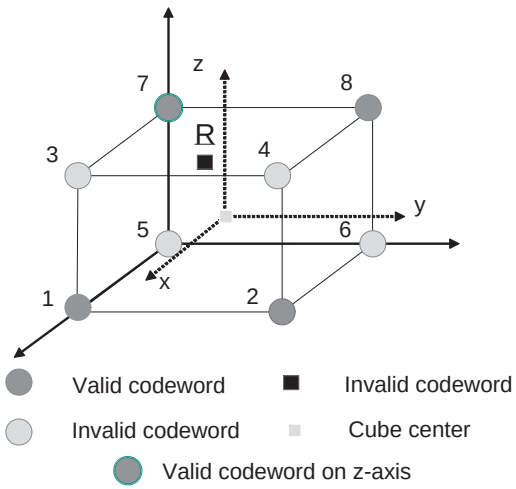

Figure 6.1: 3-D Representation of a Received Signal.

and all the possible vertices in Φ, and the correct codeword is the nearest vertex from $\underline{R}$. Our proposed algorithms reduce the computational effort required for maximum likelihood decoding, and consider only the vertices lying on the sides in the direction of the $2t + p$ dimensions corresponding to the bits of least reliability in $\underline{R}$. As p increases the complexity of the algorithm increases and the performance approaches more the theoretical maximum likelihood performance. A maximum likelihood decoding is obtained when $2t + p$ approaches n.

To better illustrate the idea, consider Fig. 6.1. It shows a hypothetical three-dimensional (3-D) space, with a set of valid and invalid vertices. The received signal $\underline{R}$ is a point in the 3-D space, thus the maximum likelihood decoding compares $\underline{R}$ to all the valid codewords in the 3-D space. For a *one*-dimensional decoding, i.e $2t+p$ hypothetically equals 1, our proposed algorithm will compare $\underline{R}$ only to the valid codewords that lie on sides in the direction of the dimensions where the point is nearer to the center of the cube. In our case the magnitude of $\underline{R}$ in the z-direction is the nearest to the cube center; thus, the chosen codeword is the valid codeword on the z-axis, in our case node 7.

6.2 Simulation And Comparison Results

Simulation and implementation results for the two proposed decoders are presented in this subsection; Fig. 6.2 and 6.3 show simulation results for BCH (255,239,2) for BPSK modulation under AWGN for p equal to 0,2,4 and 6. The

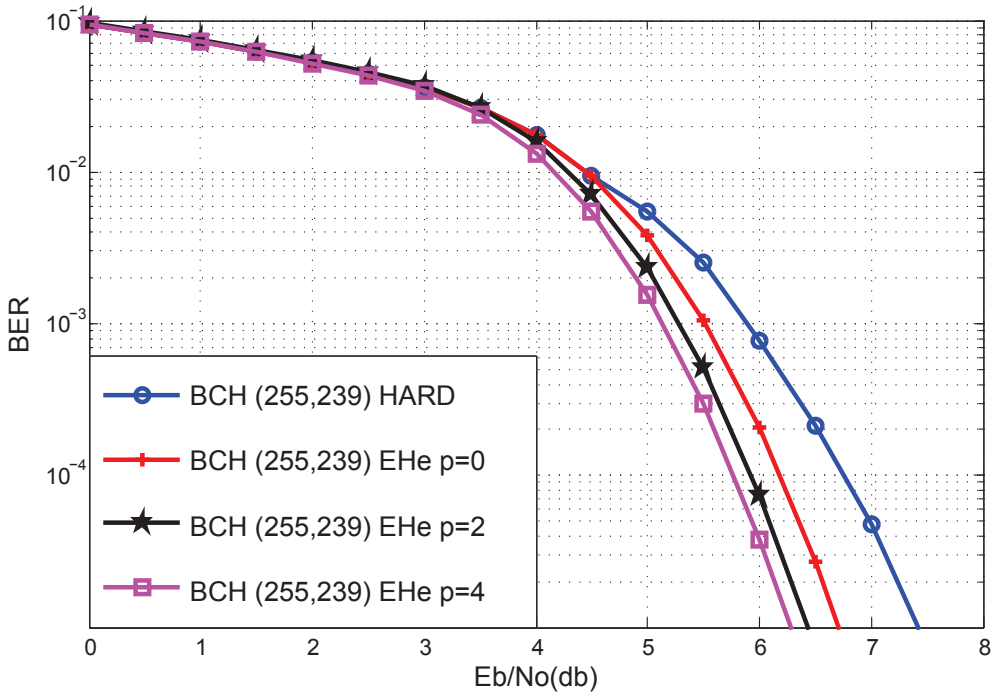

Figure 6.2: Simulation Results for BCH (255,239), EHe Decoder.

performance of EHe and EBP decoders are compared to the conventional algebraic decoder. Results show a gain that is proportional to the choice of p with a theoretical maximum likelihood decoding achieved at $p = n$. At bit error rate (BER)=10^{-5}, the EHe decoder has a coding gain of 0.75, 1, 1.2 dB for p=0, 2, 4, respectively, over hard decoding, and for EBP has a coding gain of -0.15, 0.05, 0.2, 0.35 dB for p=0, 2, 4, 6, respectively, over hard decoding. These results conform with our interpretation of the proposed algorithms.

Figure 6.4 compares the performance of algebraic decoder, EHe decoder and EBP decoder at $p = 2$ for same codeword length $n = 255$ for rate 0.97 at $k = 247$ and rate 0.9 at $k = 231$. The results show a 1 dB gain for EBP decoder and 1.3 dB gain for EHe decoder for rate 0.97 over algebraic decoding, while the results show an 0.3 dB gain for EBP decoder and 0.8 dB gain for EHe decoder at rate 0.9. These results show that with increasing the word length, the decoder still obtains a significant gain over algebraic decoding.

Figure 6.5 compares the performance of algebraic decoder, EHe decoder and EBP decoder at $p = 2$ for same codeword length $n = 511$ for rate 0.982 at $k = 502$ and rate 0.94 at $k = 484$. The results show a 0.9 dB gain for EBP decoder and 1.1 dB gain for EHe decoder for rate 0.982 over algebraic decoding,

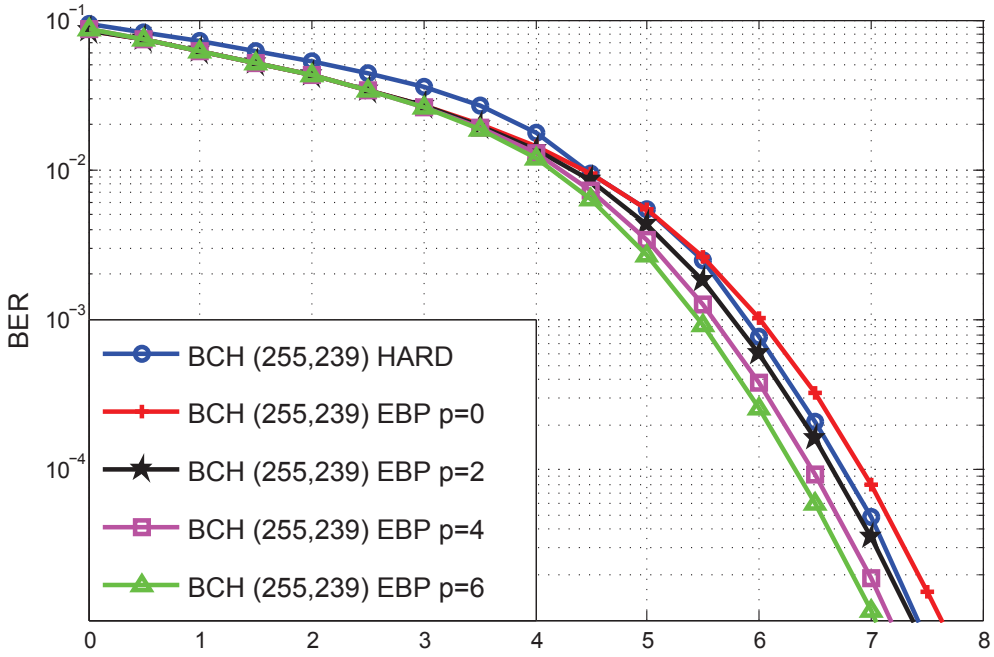

Figure 6.3: Simulation Results for BCH (255,239), EBP Decoder.

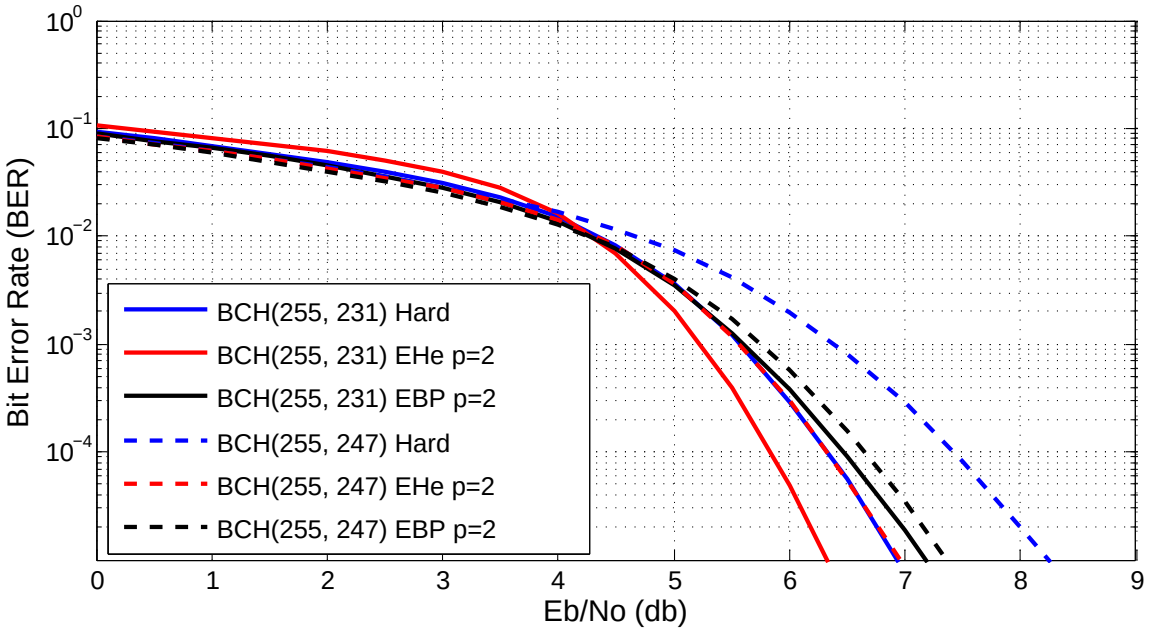

Figure 6.4: Simulation Results for $t = 1$ and 3 for $n = 255$

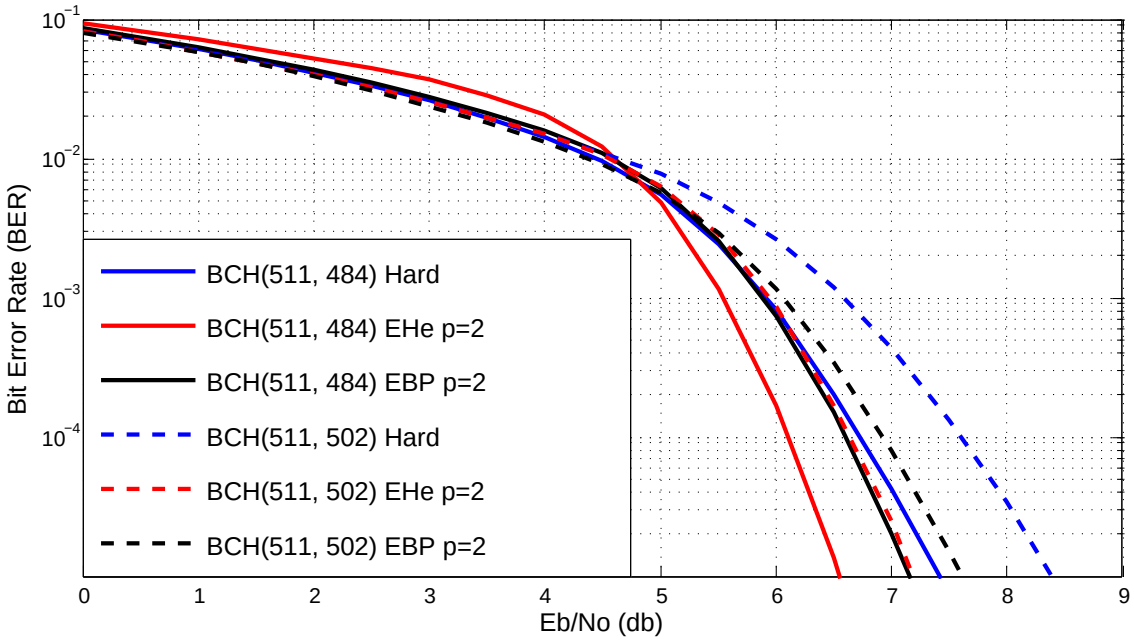

Figure 6.5: Simulation Results for $t = 1$ and 3 for $n = 511$

while the results show an 0.3 dB gain for EBP decoder and 0.8 dB gain for EHe decoder at rate 0.94. These results show an increase in the coding gain obtained as the code rate increases. These results indicate that the proposed decoders are more suitable for large rate codes.

6.3 Summary

In this chapter we proposed two decoding algorithms, the two decoding algorithms provide a programmable coding gain according to the number of extra compensated bits p with a theoretical maximum likelihood decoding as p approaches the codeword size n.

Chapter 7

Implementation of the EHE and EBP Soft Decoders

In this chapter we give a detailed explanation of the proposed architectures for the two soft decoding algorithms proposed. The objective of our design was to obtain a programmable architecture for the two proposed algorithms, with a focus on reducing the critical path of the overall system, in order to enable high operating frequency and a pipelined architecture.

In [32], [33] architectures of EHe and EBP decoding algorithms were proposed. Such architectures acquire a large critical path. The critical path in these architectures of the EHe algorithm is the delay of a square unit and a multiplier unit, while, in the EBP architecture, the critical path is the delay of a multiplier unit and an inversion unit. In our design, as will be shown later, the critical path of the overall system is around the critical path of the syndrome calculation unit, which is the delay of a constant multiplier, and an adder unit, where the critical path of the syndrome calculation unit is the least possible critical path for all BCH decoders. Also, the proposed soft decoders show hardware savings compared to the EHe and EBP architectures.

The architectures proposed can be divided into error locator evaluator, syndrome calculation unit, EHe error magnitude solver (EHe-EMS) in the case of EHe architecture and EBP error magnitude solver (EBP-EMS) in the case of EBP architecture.

This section will be divided as follows: in Subsect. 7.1, the used arithmetic units are discussed in detail. The error locator evaluator and syndrome calculation unit architectures are shown in Subsect. 7.2 7.3, respectively. The

architecture of the EHe-EMS for the EHe decoder is described in Subsect. 7.4, and the architecture of the EBP-EMS for the EBP decoder is described in Subsect. 7.5. In Subsect. 7.6, the complexity of the proposed architectures are shown, and compared against other BCH soft decoders.

7.1 Arithmetic Units

Several arithmetic units are used in the two proposed architectures. The choice of such units was based on finding a suitable unit with minimum complexity, short critical path and possibility of pipelining with different pipelining stages.

In the proposed architectures constant multiplier, adder, multiplier, power sum and inversion units were used, operating under the Galois field $GF\left(2^m\right)$. The Galois field constant multiplier is implemented using a maximum of $m^2 - \frac{m}{2}$ XOR gates, and a critical path of $log_2\left(m\right)$ the delay of an XOR gate. The adder unit requires m XOR gates with a critical path of the delay of one XOR gate. The semi-systolic multiplier unit used is based on the one proposed by [53], where a LSB first algorithm was used to compute the product $C = AB$ in $m + 1$ clock cycles, requiring m^2 cells. Each cell consist of two two-input AND gate, two two-input XOR gate, and three one-bit latches, with a critical path of $T_{AND} + T_{XOR}$, and a throughput of one operation per clock cycle. A less pipelined version of this architecture is used to reduce the latency and the number of required latches on the expense of increasing the critical path. Thus the number of cells required is $\frac{m^2}{q}$ and each unit consists of $2q$ two-input AND gate, $2q$ two-input XOR gate, and three one-bit latches, with a critical path of $q\left(T_{AND} + T_{XOR}\right)$, where q is the number of cells between each pipe-lining stage. The systolic power sum unit used is based on the one proposed by [51] . The calculation of $P = AB^2$ is performed in m clock cycles requiring m^2 cells. Each unit consist of three two-input AND gate, one two-input XOR gate, one three-input XOR gate, and four one-bit latches with a critical path of $T_{AND} + T_{3XOR}$, and a throughput of one operation per clock cycle. Also a less pipelined version of this architecture is used, thus the number of cells is reduced to $\frac{m^2}{q}$, and each cell consists of $3q$ two-input AND gate, q two-input XOR gate, q three-input XOR gate, and four one-bit latches, with a critical path of $q\left(T_{AND} + T_{3XOR}\right)$. The inversion unit used consists of $m - 1$ power sum units [51]. Table 7.1 shows the complexity and critical paths of the arithmetic units used.

Table 7.1: Arithmetic Units

	Constant multiplier	Adder	Multiplier	Power sum	Inversion
Cells	1	1	$\frac{m^2}{q}$	$\frac{m^2}{q}$	$\frac{m^2}{q}(m-1)$
Two-input AND gate	0	0	$2q$	$3q$	$3q$
Two-input XOR gate	$m^2 - \frac{m}{2}$	m	$2q$	q	q
Three-input XOR gate	0	0	0	q	q
One-bit latch	0	0	3	4	4
Latency (cycles)	1	1	$\frac{m+1}{q}$	$\frac{m}{q}$	$\frac{m}{q}$
Critical path	$log_2(m) * T_{XOR}$	T_{XOR}	$q(T_{AND} + T_{XOR})$	$q(T_{AND} + T_{3XOR})$	$q(T_{AND} + T_{3XOR})$

7.2 Error Locator Evaluator

The error locator evaluator section sorts the input bits according to their reliabilities, and for the 2t+p least reliable bits store their reliabilities $\underline{R_l} = [\, R_{l_1}\, R_{l_2}\, \ldots R_{l_{2t+p}}\,]$, where R_{l_i} is the ascending order sorted reliability number i, locations $\underline{l_e} = [\, \underline{L}\, \underline{iL}\,]$, and error locators $\underline{\beta_e} = [\, \beta_{l_1}\, \beta_{l_2}\, \beta_{l_3}\, \ldots\, \beta_{l_{2t+p}}\,]$.

This is obtained using a structure similar to the one proposed in [32] but the number of registers is extended to store the extra p values of reliabilities,locations, and error locators. The location of input bit i is obtained using a counter initialized with zero, where the output of the counter at each input bit is referred to as cL. The error locator of input bit i is obtained by multiplying a register by α^{-1} which is preloaded by α^m, where the output of the constant multiplier at each input bit is referred to as $c\beta$, and the reliability of each input bit is referred to as cR. A sorting algorithm is used where the input reliability is compared to the $2t+p$ stored reliabilities and the reliability, location, and error locator at stage i are updated as follows:

$$if\ \ input\ \ <\ R_{l_{i-1}} \qquad then\ \ R_{l_i}\ =\ R_{l_{i-1}}\ \&\ \beta_{l_i}\ =\ \beta_{l_{i-1}}\ \&\ l_{e_i}\ =\ l_{e_{i-1}} \quad (7.1)$$

$$if\ input\ > R_{l_{i-1}}\ \&\ input\ < R_{l_i} \qquad then\ R_{l_i} = cR\ \&\ \beta_{l_i} = c\beta \ \&\ l_{e_i} = cL \quad (7.2)$$

$$if\ \ input\ \ >\ R_{l_i} \qquad then\ \ R_{l_i}\ =\ R_{l_i}\ \&\ \beta_{l_i}\ =\ \beta_{l_i}\ \&\ l_{e_i}\ =\ l_{e_i} \quad (7.3)$$

This architecture for the error locator evaluator requires a total of $3\,(2t+p)$ registers, a counter circuit, and a constant multiplier $\left(\alpha^{-1}\right)$. The critical path of the unit is the delay of a constant multiplier, which is $log_2(m)T_{xor}$, requiring a total of n clock cycles.

7.3 Syndrome Calculator

The $2t$ syndromes are calculated using the Horner rule. Thus $S_i = ((\ldots(\, r_{n-1}\alpha^i + r_{n-2})\alpha^i + r_{n-3})\alpha^i \ldots r_o)$. This is calculated using $2t$ constant multipliers and adders. The critical path of this unit is the delay of a constant multiplier, and an adder which is bounded by $(log_2(m) + 1)T_{xor}$. The number of clock cycles required to calculate the $2t$ syndromes is n clock cycles, executed at the beginning of the decoding procedure.

7.4 EHe-EMS Architecture

The EHe-EMS solver can be divided into four units:

- the $\underline{B_{odd}}$ matrix calculation unit,
- the effective syndrome calculation unit,
- the heuristic search unit, and
- the weight and error location unit.

7.4.1 B_{odd} Matrix Calculation Unit

$\underline{B_{odd}}$ is calculated from $\underline{\beta}$ by sequentially multiplying β_i by β_i^2 to obtain β_i^3, then multiplying β_i^3 by β_i^2 and $(\beta_i^3)^2 = \beta_i^6$ to obtain β_i^5 and β_i^9 and so on to obtain β_i^{2t-1}. This is performed using t power sum circuit and $2t\,(t-1)$ registers, with a critical path of $q\,(T_{and}+T_{3xor})$. Define $D(i, x, m, n)$ as the number of clock cycles required to calculate x exponents of a Galois field element, using a power sum circuit, up to the exponent power n, with step m between each two exponents, where $D(i, x, m, n) = \frac{m}{q} * i + x(\sum_{j=0}^{i-1} 3^j) - \frac{x}{m}(3^i - 2t + 1) - i$ and $i \;:\quad 3^{i-1} < n \leq 3^i$. Thus the number of clock cycles required by this unit is $D(i,\, 2,\, 2,\, 2t-1)$ clock cycles, to be executed once after the first n clock cycles.

7.4.2 The Effective Syndrome Calculation Unit

The effective syndrome at each iteration can be expressed by $\underline{S}_{effective} = \underline{S} + \sum_{i=1}^{p} a_i\, \underline{\Delta S}_i$ and a_i is the binary value representing the combination of incremental syndrome vectors. The effective syndrome $\underline{S}_{eff}$ at each iteration is calculated,

as in step 7 of Algorithm I in subsection 6.1.2. The incremental syndrome $\underline{\Delta S_i}$ is calculated using a power sum circuit similar to the approach used in the $\underline{B_{odd}}$ matrix calculation. Using a Grey code counter in the calculation of $\underline{S}_{eff}$, leads to a difference in only one position per iteration between $\underline{S}_{eff}^{(i)}$ and $\underline{S}_{eff}^{(i+1)}$. This technique reduces the number of adder units to one adder, and the critical path to the critical path of one adder, which is 1 XOR gate. Thus this unit requires one power sum circuit, one adder unit, and t registers with a critical path of this unit of $q\,(T_{and} + T_{3xor}) + T_{xor}$. The number of clock cycles required by this unit is $D(i,\ t,\ 2,\ 2t-1)$, to be executed after the first n clock cycles once every 2^{2t} cycles.

7.4.3 The Heuristic Search Unit

The heuristic search unit iterates on all possible error locations $\underline{\Gamma}$ solving the equation $\underline{\Delta_{odd}} = \underline{B_{odd}} \times \underline{\Gamma} + \underline{S_{odd}}_{effective}$, as in steps 2, 5, 6, 7, and 8 in Algorithm I in subsection 6.1.2. The output sequence $\underline{\Delta_{odd}}$ is checked whether it forms a geometric sequence or not. If so, an extra is located, unless this geometric sequence is a zero vector $\underline{0}$; then, no extra error exists.

Using a Grey code counter for $\underline{\Gamma}$ simplifies the calculations by reducing the number of AND gates and adders required to one adder circuit only, and the critical path to the delay of one XOR gate only. At the beginning of 2^{2t} clock cycles $\underline{\Delta_{odd}}$ is loaded with the value output from the effective syndrome calculation unit.

To check the output for a geometric sequence $t-2$ power sum circuit are used on the output of the adders circuit to calculate $\beta_l^3\ \beta_l^5\ \ldots\ \beta_l^{2t-1}$ from β_l and compare them with the output sequence $\underline{\Delta_{odd}}$. Additional register stages are added to accommodate for the $\frac{m}{q}$ delay cycles of the power sum circuit,requiring additional $(t-1)\left(\frac{m}{q}-1\right)$ registers.

A comparator unit is used to compare the output of the power sum circuit against $\underline{\Delta_{odd}}$, if the check succeeds a memory access is attempted and the extra error is calculated, if $\underline{\Delta_{odd}} = \underline{0}$ no additional error bits exist.

The heuristic search unit requires $t-2$ power sum circuit, $3t-1$ adders, a look-up table, and $(t-1)\left(\frac{m}{q}\right)+1$ registers, the critical path of this unit is $q\,(T_{and}+T_{3xor})+T_{xor}$, to be executed after $\underline{B_{odd}}$ matrix calculations continuously till the decoding stops, requiring $\frac{m}{q}+2^{2t}$ clock cycles.

7.4.4 The Weight and Error Location Unit

This unit calculates the error weight after each iteration, as in steps 3 and 4 in Algorithm I in subsection 6.1.2, and if a valid codeword is found, the error weight is compared to the minimum weight stored at this iteration. If the error weight is found less than the minimum weight, the minimum weight is updated with the new value, and the corresponding error locations are stored. In case an extra error is found the weight of that error location is assumed to be half the maximum reliability value, such assumption reduces the complexity required for storing the reliabilities of all codewords, and does not affect the performance significantly.

The same approach of a Grey code counter is used, where at each iteration the reliabilities are updated with only the changed location. Since a transition can either be the presence of an error at a location or the absence of an error from that location, an adder subtracter fixed point unit is used. The previous procedure is performed for the $2t$ least reliabilities each clock cycle, and for p extra bits each $2^{2t} + \frac{m}{q}$ clock cycles.

This unit requires two fixed point adder/subtracter units, four registers and one comparator. The critical path of this unit is the delay of a fixed point adder and comparator, which is $2(T_{and} + T_{xor})$. This unit requires one clock cycle, and it operates each clock cycle after B_{odd} calculation.

7.5 EBP-EMS Architecture

EBP-EMS can be divided into four main sections:

- the BP solver unit,
- the binary sequence check unit,
- the incremental syndrome unit, and
- the error calculation unit.

7.5.1 The BP Solver Unit

The BP algorithm is carried out sequentially performing steps a1, a2 and a3 from Algorithm II in subsection 6.1.3. The values in S are loaded from the syndrome

calculation unit each iteration. Since the inversion unit's input is β_i which is constant, no stalls are required for the inversion unit. In case of the multiplier, data dependency occurs between the output of the multiplier unit and its input, which leads to stalls; such stalls are $\frac{m}{2q}\left(\frac{m}{q}-1\right)$ for step a1, $2(\frac{m}{2q}\left(\frac{m}{q}-1\right)-1)$ for step a2 and none for step a3, resulting in a total of $\frac{3}{2}\left(\frac{m}{q}\right)^2-\frac{3}{2}\left(\frac{m}{q}\right)-2$ stall cycles to be added to the $6t^2-t$ cycles required to perform the algorithm. The critical path of this unit is $q\,(T_{and}+T_{3xor})+T_{xor}$. This unit operates iteratively $P+1$ times to obtain $\underline{\Gamma_{ext}}$ requiring three adder circuits, an inversion unit and multiplier units.

7.5.2 The Binary Sequence Check Unit

The binary sequence check unit receives $\underline{\Gamma_{ext}}$ from the BP solver unit, and iterates on all possible combinations $\underline{\Delta}=\underline{\Gamma}+\sum_{i=1}^{p}b_i\,\underline{\Gamma}_{l_i}$, as in steps b1, b3 and b4 in Algorithm II in subsection 6.1.3. The result is checked whether it forms a binary sequence or not. A Grey code counter is used to perform the previous iterations. The output of the adder unit is checked to be a binary sequence using a comparator. The previous operation requires 2^{p-1} extra delay cycles after the calculation of $\underline{\Gamma}_{l_p}$, with a critical path of the delay of an adder which is T_{xor}, performed after the first iteration of the BP solver unit.

7.5.3 The Incremental Syndrome Unit

The incremental syndrome unit uses a power sum circuit to calculate ΔS_i during each iteration from the BP solver unit, similar to the approach used to calculate B_{odd}. This unit requires $2t$ registers, and one power sum circuit ,with a critical path of $q\,(T_{and}+T_{3xor})$. The number of clock cycles required by this unit is $D(i,\,t,\,1,\,2t)$ clock cycles, to be executed after the first n clock cycles once during every $6t^2-t+\frac{3}{2}(\frac{m}{q})^2-\frac{3}{2}(\frac{m}{q})-2$ cycles.

7.5.4 The Error Calculation Unit

The error calculation unit calculates the error weight of each error sequence as in step b2 in Algorithm II in subsection 6.1.3. The unit operates in two stages; the first stage uses a Grey code counter to update the error weight due to the

incremental syndrome, similar to that in weight and the error locations unit, while the other stage depends on whether the output sequence is binary. In case of a binary sequence the error weight is calculated using a fixed point adder tree.

The error weight is compared to the minimum weight stored, and if found less the minimum weight is updated with the new weight, and the corresponding error pattern is stored. This circuit requires $m+1$ fixed point adders, and three registers. The critical path of this unit is the delay of a fixed point adders tree and a comparator, which is $(log_2(t)+1)(T_{and}+T_{xor})$. Pipelining may be required to reduce the critical path of the fixed point tree, in such case the critical path becomes $\frac{(log_2(t)+1)(T_{and}+T_{xor})}{St+1}$, where St is the number of pipelining stages. The number of clock cycles required is one, to be performed after each iteration of the binary sequence check unit.

7.6 Proposed Architecture Evaluation and Discussion

In this subsection the complexity of the proposed architectures are shown, then the complexity of the proposed architecture is compared to other BCH decoders for general BCH (n, k, t) code and for BCH (255, 239, 2) code.

7.6.1 Complexity of the Proposed Architectures

The combined architectures of the two proposed decoders are shown in Figs. 7.1, 7.2 and 7.3, each architecture was divided into main functional units, where for each unit in both algorithms the number of arithmetic units required, the critical path and the latency are stated. In this section we show the overall complexity of architectures proposed for the two algorithms.

Tables 2 and 3 show summaries of the complexities of the EHe and EBP decoders. It is shown that the number of power sum units for the EHe decoder depends on the number of correctable bits t, the latency of the decoding procedure is a function of 2^{2t}, also the size of the LUT depends on the codeword size n; thus, the EHe decoding algorithm is more suited to codes of medium size, rather than large codes.

Table 7.2: Summary of the EHe Decoder Complexity

	Power sum	Registers	Constant multiplier	Latency (cycles)	Critical path
Error locator evaluator	0	3(2t+p)	1	n	$log_2(m)T_{xor}$
Syndrome calculation	0	t	t	n	$(log_2(m)+1)T_{xor}$
B_{odd} matrix construction	t	$2t^2$-3t	0	$D(i,\ 2,\ 2\ ,\ 2t-1)$	$q\,(T_{and}+T_{3xor})$
Effective syndrome calculation	1	t	0	$D(i,\ t,\ 2\ ,\ 2t-1)$	$q\,(T_{and}+T_{3xor})+T_{xor}$
Heuristic search	t-2	$\frac{m}{q}\,(t-1)$ +1	0	2^{2t}+$\frac{m}{q}$	$q\,(T_{and}+T_{3xor})+T_{xor}$
Weight and error location	0	4	0	1	$2\,(T_{and}+T_{xor})$
Overall complexity	t	$2t^2$+ (5+$\frac{m}{q}$)t+3p-$\frac{m}{q}$+5	t+1	n+ $D(i,\ 2,\ 2,\ 2t\ -\ 1)$ +2^{2t+p}+ $\frac{m}{q}$+ 1	max(2 $(T_{and}\ +\ T_{xor})$, $q\,(T_{and}\ +\ T_{3xor})\ +\ T_{XOR}$, $(log_2(m)+1)T_{xor})$

Table 7.3: Summary of EBP Decoder Complexity

	Power sum	Registers	Constant multiplier	multiplier	Latency (cycles)	Critical path
Error locator evaluator	0	3(2t+p)	1	0	n	$log_2(m)T_{xor}$
Syndrome calculation	0	2t	2t	0	n	$(log_2(m)+1)T_{xor}$
BP solver unit	m-1	$2t^2$-3t	0	1	$6t^2$-t+ $\frac{3}{2}(\frac{m}{q})^2$- $\frac{3}{2}(\frac{m}{q})$-2	$q\,(T_{and}+T_{3xor})+T_{xor}$
Binary sequence check	0	2tp	0	0	2^p	T_{xor}
Incremental syndrome	1	2t	0		$D(i,t,1,2t)$	$q\,(T_{and}+T_{3xor})$
Error calculation	0	3	0	0	St+1	$\frac{(log_2(t)+1)}{St+1}$ * $(T_{AND}+T_{XOR})$
Overall complexity	m	10t+(2t +3)p+3	2t+1	1	n+$(p+1)(6t^2$- t+ $\frac{3}{2}(\frac{m}{q})^2$- $\frac{3}{2}(\frac{m}{q})$- 2)+2^{p-1} +st+1	max($\frac{(log_2(t)+1)}{St+1}$ * $(T_{AND}+T_{XOR})$, $q\,(T_{and}+T_{3xor})+T_{xor}$, $(log_2(m)+1)T_{xor}$)

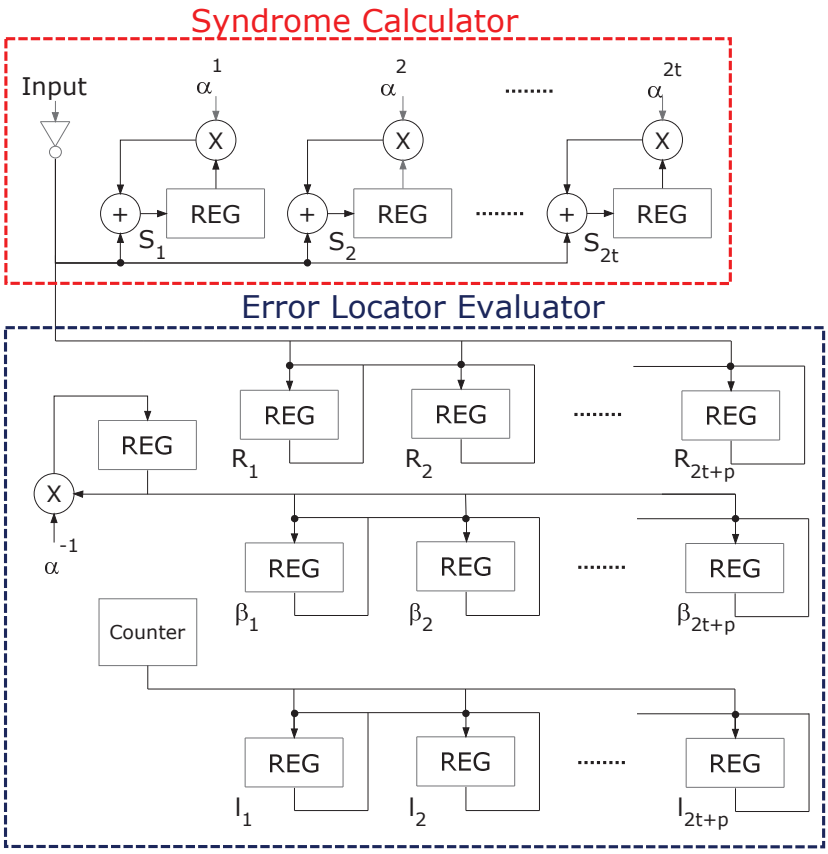

Figure 7.1: Syndrome Calculator and Error Locator Evaluator Units

In EBP algorithm the number of power sum units depends on the extension field size m, and the latency is a function of t^2, thus EBP is suitable for large BCH codes.

It is also worth noting that the major section of the latency in case of the EBP algorithm, which is the latency of the BP solver, is linearly proportional with the number of extra compensated bits p, while in the case of EHe algorithm the major section of the latency, which is the latency of the heuristic search unit, is exponentially proportional with p, allowing EBP algorithm to correct more extra bits (higher p) for the same latency compared to EHe algorithm, but the extra error compensation in EHe leads to a superior initial performance (at $p = 0$) compared to the EBP algorithm, as will be shown later.

Architecture Comparison

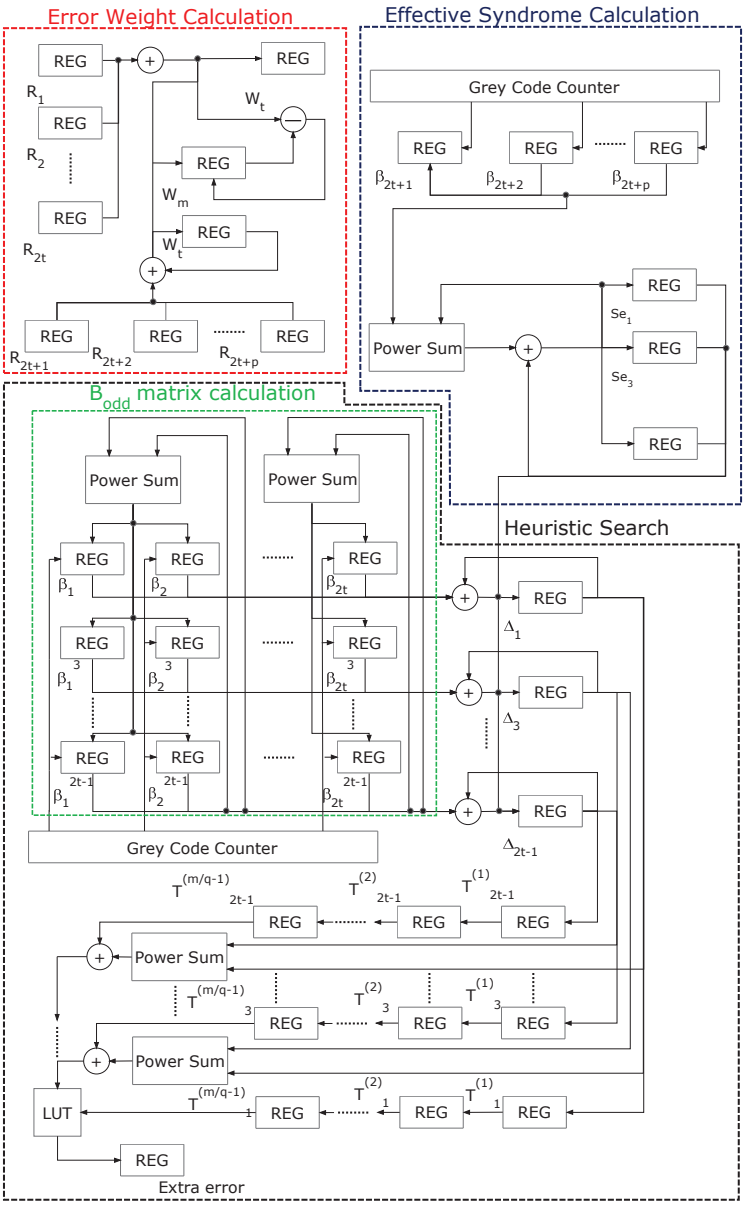

Figure 7.2: EHe-EMS Architecture

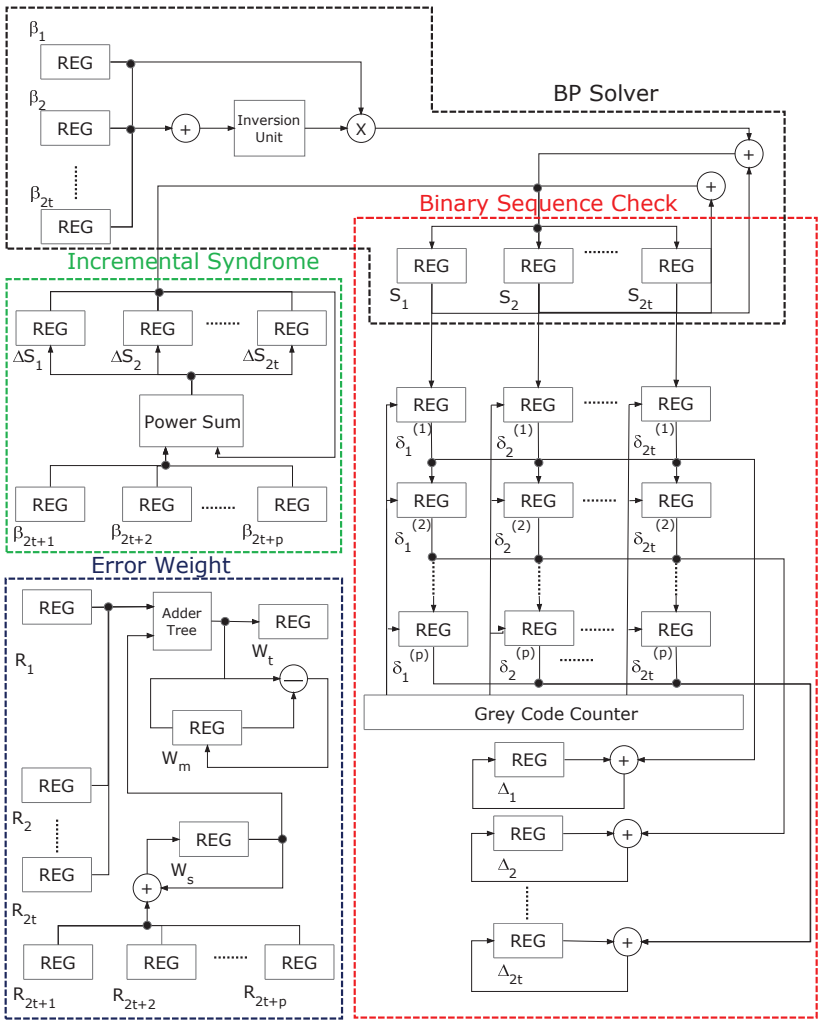

Figure 7.3: EBP-EMS Architecture

Table 7.4: Comparison for BCH (n, k, t) and (255, 239, 2)

BCH(n,k,t) / BCH(255,239,2)	iBM	He-EMS	BP-EMS	proposed EHe-EMS	proposed EBP-EMS
Register	5t+2	$2t^2$+6t	8t	$2t^2 + (6+\frac{m}{q})t + 3p-\frac{m}{q} + 5$	10t+(2t +3)p+3
	12	20	16	27+3p	23+7p
Multiplier	3t+3	3t-1	1	0	1
	9	5	1	0	1
Power Sum	0	0	0	t	m
	0	0	0	2	8
Constant Multiplier	3t	t+1	2t+1	2t+1	2t+1
	6	3	5	3	5
Squarer	0	2t+1	0	0	0
	0	5	0	0	0
Inversion Unit	0	0	1	0	0
	0	0	1	0	0
LUT	0	1	0	1	0
	0	1	0	1	0
Latency	2n+2t	n+2^{2t}+t-1	n+$6t^2$-t	n+$D(i, 2, 2, 2t - 1) + 2^{2t+p}+\frac{m}{q}+ 1$	n+($6t^2$-t+ $\frac{3}{2}(\frac{m}{q})^2$-$\frac{3}{2}(\frac{m}{q})$-2)$(p + 1)$ +2^p+St+1
	514	272	277	$265+ 16 * 2^p$	$294+38p +2^{p-1}$
Critical Path	$2T_{and}+(2m + 1)T_{xor}$	$2T_{and}+\frac{3}{2}mT_{xor}$	$2T_{and}+2mT_{xor})$	$\max(2(T_{and} + T_{xor}), q(T_{and} + T_{3xor}) + T_{XOR}, (log_2(m)+1)T_{xor})$	$\max(\frac{(log_2(t)+1)(T_{AND}+T_{XOR})}{St+1}, q(T+T_{3xor})+T_{XOR}, (log_2(m)+1)T_{xor})$
	$2T_{and}+17T_{xor}$	$2T_{and}+12T_{xor}$	$2T_{and}+16T_{xor}$	$2(T_{and} + T_{3xor})+T_{XOR}$	$2(T_{and} + T_{3xor})+T_{XOR}$

In this Subsection, a comparison between the proposed architecture and other BCH soft decoders for general BCH (n, k, t) and for BCH (255, 239, 2). As shown in Table 7.4 comparing the complexity of the proposed EHe-EMS to the complexity of He-EMS [33], the $3t-1$ multiplier circuits, and the $2t+1$ squarers in the He-EMS architecture are replaced with t power sum units. Extra registers are added in the proposed architecture to store the information corresponding to the extra p bits stored. Also, to compensate the latency of the systolic arithmetic units used. The latency of the proposed algorithm is 2^p clock cycles more than than of He-EMS, in order to iterate on the p extra extra error locations. Also some extra stall cycles appear due to the latency of the used systolic arithmetic units.

Comparing the complexity of the proposed EBP-EMS to the complexity of BP-EMS [32], the inversion unit is replaced with m power sum units, note that the inversion in our proposed architecture is carried out using $m-1$ power sum units, and the extra unit is used to evaluate the incremental syndrome. Extra registers are added in the proposed architecture to store the information corresponding to the extra p bits stored. The latency of the proposed algorithm is about p times more than that of EBP-EMS, in order to iterate on the p extra extra error locations. Also some extra stall cycles appear in order to calculate the error locations.

As for the critical path, the two proposed architectures acquire a configurable critical path according to the choice of the pipelining stages q, thus allowing for flexible choice of the critical path. Also, in case of EBP-EMS the choice of the number of pipelining stages in the fixed point adder tree St allows a configurable critical path. In the case of He-EMS the critical path is the delay of a multiplier and a squarer unit, while in the case of EBP-EMS the critical path is delay of a multiplier and an inversion unit, and in the case in the inversion-less Berlekamp-Massey algebraic decoder (iBM) [8] the critical path is the delay 2 multipliers and $(log_2(\frac{t-1}{2})+2)$ adder units. Although the critical path of the multiplier, inversion, squarer units depends on the choice of the arithmetic unit itself, but the critical path of a bit parallel multiplier unit is around $T_{AND}+mT_{XOR}$ [48], while the critical path of an inversion unit is around $(m-1)(T_{AND}+mT_{XOR})$, and the critical path of the squarer is around $T_{AND}+\frac{m}{2}T_{XOR}$. For BCH (255,239, 2), the critical path for the EHe-EMS decoder becomes $\max(2\,(T_{and}+T_{xor}),$

$q\,(T_{and} + T_{3xor}) + T_{XOR}$, $4T_{xor}$), and for EBP-EMS becomes max($\frac{2(T_{AND}+T_{XOR})}{St+1}$, $q\,(T + T_{3xor}) + T_{xor}$, $4T_{xor}$), thus an optimum critical path is obtained by choosing $q = 2$ and $St = 0$, i=1, thus $D(1, 2, 2, 3)$=5, the critical path of EHe-EMS and EBP-EMS becomes $2\,(T_{and} + T_{3xor}) + T_{XOR}$.

The increase in required registers in our design compared to [32], [33], and [8] is a linear factor of p the extra compensated bits, while the gain from using the pipelined architecture is a decrease in the critical path of around $m/2$ where m is the Galois field size.

7.7 Implementation Results

Table 7.4 shows the BCH (255,239,2) complexity for the two proposed decoders and compares them to the algebraic decoder, and the soft decoders presented in [32, 33]. Results show a significant complexity reduction compared to the algebraic decoder [8], comparing the complexities of the EBP decoder to [32] a complexity reduction is obtained by using two power sum circuits instead of five squarers, and multipliers, and comparing the complexity of the EHe decoder to [33] only an increase in the required registers occurs which is insignificant to the overall complexity. The number of clock cycles for the decoding procedure is reduced compared to the algebraic decoder due to the elimination of the Chien search procedure; the number of clock cycles required for EBP and EHe decoders depends on the number of extra compensated bits p; thus, an increase in the required number of clock cycles occurs for high values of p, but with a critical path of about four times less than that of the algebraic decoder, or to the decoders at [32,33]. The overall latency of the decoding procedure is reduced as the dominating factor in the overall latency is due to the codeword length.

The register transfer level (RTL) implementation of the EBP and EHe decoders were written using VHDL hardware description language(HDL) for a generic number of extra corrected bits p, the block diagram of the decoders is shown in figures 7.4 and 7.5, and the pin description is shown in table 7.5.

Table 7.5: EBP and EHe Decoders Pin Description

Pin Name	Width	Pin Description
clock	1	Input clock
reset	1	Active high reset
data in	1	k input bits
op ready	1	Output ready flag
sorted rel	$(2t+p)*m$	Locations of the $2t+p$ least reliable bits
error locations	$2t+p$	Locations in the $2t+p$ sorted reliabilities containing error bits
extra error	m	In EHe decoder, the extra error location

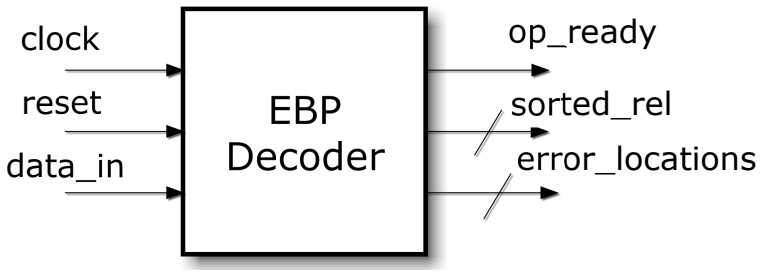

Figure 7.4: EBP Block Diagram

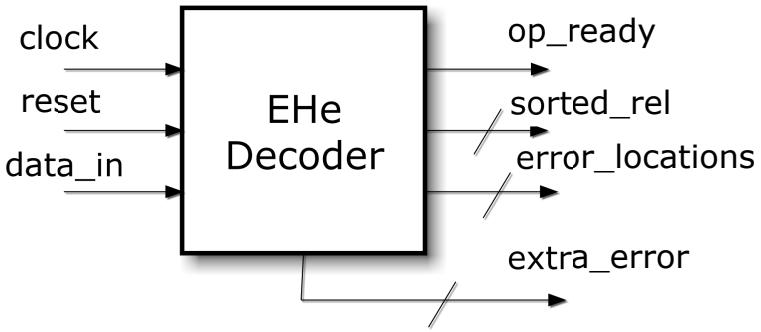

Figure 7.5: EHe Block Diagram

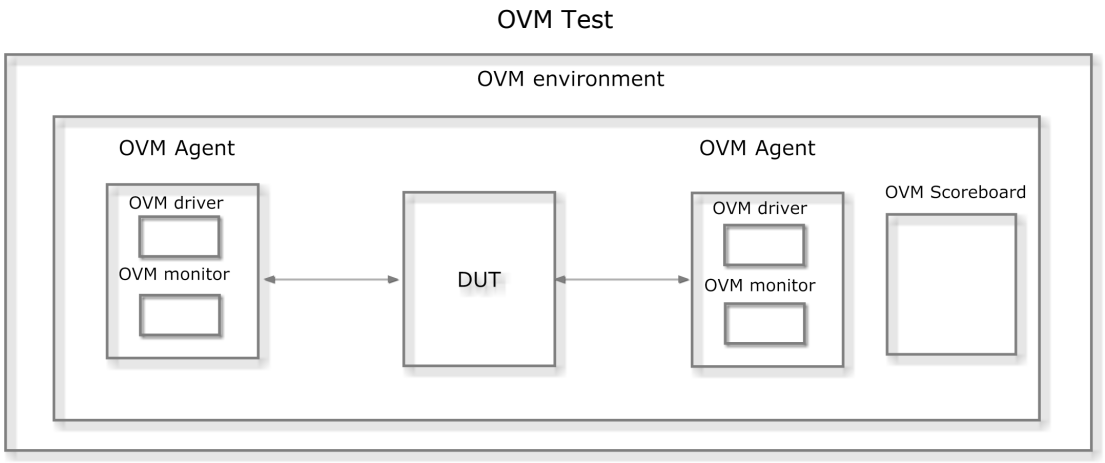

Figure 7.6: OVM Verification Environment

7.7.1 Functional Verification

To verify the functionality of the RTL designs open verification methodology(OVM) was employed. Two OVM agents were used the first agent generates the stimulus and sends the generated stimulus to the OVM scoreboard, and the other reads the stimulus and sends it to the OVM scoreboard. This structure is shown in figure 7.6

The first OVM agent's driver generates stimulus as follows:

- Random bits are generated with length equal to the message length k which is 239 in our case.
- The k random bits are encoded using the generator polynomial $1 + x^2 + x^3 + x^5 + x^7 + x^8 + x^{10} + x^{11} + x^{15} + x^{16}$.
- A configurable number of errors are inserted at the coded message of length n to test the correcting capability

An OVM monitor of the second agent then waits for the decoding to be finished, then reads the error locations and sends these error locations to the OVM scoreboard. The OVM scoreboard then compares the error locations generated at the first agent, to the results captured by the second agent. Then if the design under test fails to correct a number of error bits less than or equal to its correcting capability, an error message is produced to indicate a bug in the RTL design of the decoders.

Table 7.6: Implementation Results for BCH(255,239) EBP and EHe Decoders

BCH Decoder		Cell Area $(\mu m)^2$	Operating Frequency (MHz)	Latency (Clock Cycles)	Throughput $(Mbits/sec)$
EBP Decoder	$p = 2$	14727	1000	$n + 132$	617.5
	$p = 4$	18510		$n + 214$	509
	$p = 6$	22365		$n + 314$	420
EHe Decoder	$p = 2$	14320	1000	$n + 74$	726.44
	$p = 4$	19542		$n + 266$	458.7
	$p = 6$	28461		$n + 1034$	185.4
Algebraic Decoder		14400	360	$2n + 4$	167.4
He Decoder		13225	360	$n + 17$	316.3

7.7.2 Implementation Results

The designs were synthesized on Taiwan Semiconductor Manufacturing Company (TSMC) general purpose (G) 90 nm technology [54], and the target clock frequency was set to 1 GHz, and Physical Layout Estimation (PLE) interconnect model. With the inverter delay at standard voltage being 26 picoseconds; thus, a maximum of 38 inverter delays is required to be able to close the circuit timing at 1 nanosecond. The implementation results for BCH(255,239) EBP and EHe decoders for $p = 2,\ 4,\ 6$ is shown below in table 7.6 and compared to algebraic decoder, and decoders at [32, 33].

The number of clock cycles for EBP decoder $p = 2,\ 4\, and\, 6$ is $n+132$, $n+214$ and $n + 314$ clock cycles, at 1 Ghz operating clock this results in a throughput of 617.5 Mbits/sec, 509 Mbits/sec and 420 Mbits/sec. While for EHe decoder $p = 2,\ 4\, and\, 6$ is $n + 74$, $n + 266$ and $n + 1034$ clock cycles, at 1 Ghz operating clock this results in a throughput of 726.4 Mbits/sec, 458.7 Mbits/sec and 185.4 Mbits/sec. Thus, the throughput in our proposed decoders exceeds that of algebraic decoder of throughput 167.4 Mbits/sec and decoders proposed at [33] and [32] of throughput 316.3 Mbits/sec.

The cell area for EBP decoder for EBP decoder $p = 2,\ 4\, and\, 6$ is 14727, 18510 and 22365 μm^2. While for EHe decoder $p = 2,\ 4\, and\, 6$ is 14320, 19542 and 28461 μm^2. Thus, the area in our proposed decoders is comparable to that of algebraic decoder of area 14400 μm^2 and decoders proposed at [33] and [32] of area 13225 μm^2.

7.8 Summary

In this chapter the implementation is EHe and EBP decoders is detailed. Where EHe decoder's complexity increases exponentially with the number of extra compensated bits p and the number of power sum units is a function of the number of extra compensated bits t, while EBP decoder's complexity increases linearly with p and the number of power sum units is a function of the field size m. This makes EHe decoder more suitable for medium sized BCH codes with a small number of extra compensated bits, while EBP decoder is more suitable for large sized BCH codes with large number of extra compensated bits.

Chapter 8

Conclusion

In this thesis, two soft BCH decoders are proposed for high-rate BCH codes, where the EHe decoder is suitable for codes with medium code word size, and the EBP decoder is suitable for codes with large code word size. This work was published in [9] and [10].

The proposed algorithms obtain a programmable coding gain that varies according to the number of extra compensated bits p and reaches a theoretical maximum likelihood bound as p approaches the codeword length n. The coding gain of the proposed algorithms increase compared to algebraic decoding algorithms as the code rate n/k increase. As shown in Table 7.4, the increase in the number of compensated bits results in an insignificant increase in complexity of the decoder, and an increase in the required number of clock cycles is compensated by the reduced critical path of the design. the shorter critical path allows the system to operate at the maximum operating frequency which is set by the syndrome calculation unit. Compared to the legacy Berlekamp-Massey decoder, the proposed soft decision decoding algorithms not only outperform the hard-decision based algorithm but also eliminate the Chien search procedure resulting in a reduction in the hardware complexity, latency and an increase in the overall system throughput.

Synthesizing the two proposed decoders on TSMC 90 nm G technology for BCH (255, 239), our proposed EBP decoder obtains a throughput of 617.5 Mbits/sec, 509 Mbits/sec and 420 Mbits/sec at an area of 14727, 18510 and 22365 μm^2 at $p = 2, 4 and 6$ obtaining a gain of 0.05, 0.2, 0.35 dB over algebraic decoding. While EHe decoder obtains a throughput of 726.4 Mbits/sec, 458.7 Mbits/sec and 185.4 Mbits/sec at an area of 14320, 19542 and 28461 μm^2 at

$p = 2,\ 4\ and\ 6$ obtaining a gain of 0.75, 1, 1.2 over algebraic decoding.

As the results indicate the EHe decoder obtain a higher coding gain over EBP decoder, but with a non-linear increase in the complexity with the increase in the number of extra corrected bits p. This makes EHe decoder more suitable for medium length codes with high rates and less number of extra correctable bits, while, EBP decoder is more suitable for large codes with high rates with high number of extra correctable bits.

8.1 Future Work

The future work could be to integrate the proposed decoders into DVB second generation systems and test the following:

1. The two algorithms provided here can be integrated into , to test the efficiency of an iterative loop between the LDPC decoder and the BCH decoder.
2. The two decoders can be implemented, and the reduction in power consumption could be measured, or the possibility of changing the number of extra compensated bits p which changes the coding gain according to system power mode.

The two proposed decoders can be implemented for a larger size

Bibliography

[1] Cohen. *Codes correcteurs d'erreurs*. December 1997.

[2] G. Dagnino. On a new class of binary group codes. *CALCOLO*, 5(2):277–294, 1968.

[3] I. S. Reed and G. Solomon. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[4] Faisal Rasheed Lone, Arjun Puri, and Sudesh Kumar. Performance comparison of reed solomon code and bch code over rayleigh fading channel. *CoRR*, abs/1307.6930, 2013.

[5] ETSI Standard. *Digital Video Broadcasting (DVB); Frame Structure Channel Coding and Modulation for a Second Generation Digital Terrestrial Television Broadcasting System (DVB-T2)*, Sep. 2009.

[6] ETSI Standard. *Digital Video Broadcasting (DVB); Frame Structure Channel Coding and Modulation for a Second Generation Digital Terrestrial Television Broadcasting System (DVB-C2)*, Feb. 2011.

[7] G. Atwood, A Fazio, D Mills, and B Reavesi. Intel StrataFlash memory technology overview. In *Intel Tech. Journal*, pages 1–8, 1997.

[8] I.S. Reed and M.T. Shih. Vlsi design of inverse-free berlekamp-massey algorithm. *Computers and Digital Techniques, IEE Proceedings E*, 138(5):295–298, Sep 1991.

[9] M.T.A. Osman, H.A.H. Fahmy, Y.A.H. Fahmy, and M.A. Elsabrouty. Two programmable bch soft decoders for high rate codes with large word length. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pages 1556–1559, 2013.

[10] MohamedT.A. Osman, HossamA.H. Fahmy, YasmineA.H. Fahmy, MahaM. Elsabrouty, and Ahmed Shalash. Two extended programmable bch soft decoders using least reliable bits reprocessing. *Circuits, Systems, and Signal Processing*, pages 1–23, 2014.

[11] R E Blahut. *Theory and practice of error-control codes*. Addison-Wesley, Reading, MA, 1983.

[12] Shu Lin and Daniel J Castello. *Error control coding, Fundamentals and applications*. Prentice hall, New Jersey, 1983.

[13] I Stewart. *Galois theory*. Chapman and Hall, London, 1973.

[14] F J Mac Williams and N J A Sloane. *The theory of error correcting codes*. North Holand, 1977.

[15] E R Berlekamp. Bit-serial Reed-Solomon encoders. In *IEEE Transaction on Information Theory*, volume 28, pages 120–126, Nov 1982.

[16] E Mastrovito. *VLSI Architectures for Computation in Galois Fields*. PhD thesis, Linkoping Univ., Dept. of Electrical Eng, 1991.

[17] Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8:300–304, 1960.

[18] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. North-holland Publishing Company, 2nd edition, 1978.

[19] Shu Lin and Daniel J. Costello Jr. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, 1983.

[20] Richard E. Blahut. *Theory and practice of error control codes*. Addison-Wesley Pub. Co. c1983, Reading, MA, 1983.

[21] Yasuo Sugiyama, Masao Kasahara, Shigeichi Hirasawa, and Toshihiko Namekawa. A method for solving key equation for decoding goppa codes. *Information and Control*, 27(1):87–99, January 1975.

[22] E. R. Berlekamp. *Algebraic coding theory*. Aegean Park Press, Laguna Hills, CA, USA, 1984.

[23] R. Chien. Cyclic decoding procedures for bose- chaudhuri-hocquenghem codes. *IEEE Transaction on Information Theory*, 10(4):357–363, September 2006.

[24] W Liu, J Rho, and W Sung. Low-power high-throughput BCH error correction VLSI design for multi-level cell NAND flash memories. In *Proc. SiPS*, pages 303–308, Banff, Alberta, Canada, Oct. 2006.

[25] E R Berlekamp. *Algebraic Coding Theory*. Aegean Park Press, New York: McGraw-Hill, 1968.

[26] J Hong and M Vetterli. Simple algorithms for BCH decoding. In *IEEE Transaction on Communications*, volume 43, pages 2324–2333, Aug. 1995.

[27] Jr. Forney, G.D. and A. Vardy. Generalized minimum-distance decoding of euclidean-space codes and lattices. *Information Theory, IEEE Transactions on*, 42(6):1992–2026, Nov 1996.

[28] M Lalam, K Amis, D Lerous, D Feng, and J Yuan. An improved iterative decoding algorithm for block turbo codes. In *Proc. IEEE International Symposium on Information Theory*, pages 2403–2407, Jul. 2006.

[29] D Chase. A class of algorithms for decoding block codes with channel measurement information. In *IEEE Transaction on Information Theory*, volume IT-18, pages 170–182, 1972.

[30] Xinmiao Zhang, Jiangli Zhu, and Yingquan Wu. Efficient one-pass Chase soft-decision BCH decoder for multi-level cell NAND flash memory. In *Proc. MWSCAS*, pages 1–4, 2011.

[31] W J Reid, L L Joiner, and J J Komo. Soft decision decoding of BCH codes using error magnitudes. In *IEEE International Symposium on Information Theory*, page 303, June 1997.

[32] Y M Lin, C Chen, H Chang, and C Lee. A 26.9 K 314.5 Mb/s soft (32400,32208) BCH decoder chip for DVB-S2 system. *IEEE Journal of solid-state circuits*, 45(11), Nov. 2010.

[33] Y M Lin, H Chang, and C Lee. An improved soft BCH decoder with one extra error compensation. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pages 3941–3944, Paris, France, 2011.

[34] M Fossorier and S Lin. Soft-decision decoding of linear block codes based on ordered statistics. In *IEEE Transaction on Information Theory*, volume 41, pages 1379–1396, Sep. 1995.

[35] A Valembois and M Fossorier. Box and match techniques applied to soft decision decoding. In *Information Theory and Applications Workshop*, page 143, 2002.

[36] W Jin and M Fossorier. Efficient box and match algorithm for reliability-based soft-decision decoding of linear block codes. In *Information Theory and applications workshop*, pages 160–169, Feb. 2007.

[37] J S Yedidia, J Chen, and M Fossorier. Generating code representations suitable for belief propagation decoding. In *Proc. Allerton*, Oct. 2002.

[38] J Jiang and K R Narayanan. Iterative soft-input-soft-output decoding of Reed-Solomon codes by adapting the parity check matrix. In *IEEE Transaction on Information Theory*, volume 52, 2006.

[39] S K Jain, L Song, and K K Parhi. Effecient Semisystolic Architectures for Finite-Field Arithmetic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pages 101–113, 1998.

[40] C L Wang and J L Lin. Systolic array implementation of multipliers for finite fields GF(2^m). In *IEEE Transaction on Circuits Systems*, volume 38, pages 796–800, 1991.

[41] C S Yeh, I S Reed, and T K Troung. Systolic multipliers for finite fields GF(2^m). In *IEEE Transaction on Computer*, 1984.

[42] C Y Lee. Low-complexity bit-parallel systolic multipliers over GF(2^m). Integrat. In *IEEE Journal on VLSI*, 2008.

[43] B A Laws and C K Rushforth. A Cellular-Array Multiplier for GF (2^m). In *IEEE Transaction on computers*, volume 100, pages 1573–1578, March 1971.

[44] Arash Hariri. *Arithmetic Units for the Elliptic Curve Cryptography with Concurrent Error Detection Capability*. PhD thesis.

[45] IEEE Standard Specifications for Public-Key Cryptography, Jan. 2000.

[46] J L Massey and J K Omura. Computational Method and Apparatus for Finite Field Arithmetic. Technical report, US Patent No. 4,587,627, 1986.

[47] R Lidl and H Niederreiter. *Introduction to finite fields and their applications.* Cambridge University Press, New York, 1994.

[48] S T J Fenn, M Benaissa, and D Taylor. GF(2^m) Multiplication and division over the dual field. In *IEEE Transaction on computers*, volume 45, pages 319–327, March 1996.

[49] T Beth and D Gollman. Algorithm Engineering for Public Key Algorithms. In *IEEE Journal Selected Areas in Communication*, volume 7, pages 458–465, May 1989.

[50] S Kwon. A low complexity and a low latency bit parallel systolic multiplier over GF(2^m) using an optimal normal basis of Type II. In *16th IEEE Symposium Computer Arithmetic*, pages 196–202, June 2003.

[51] Shyue-Win Wei. VLSI architectures for computing exponentiations, multiplicative inverses, and divisions in GF(2^m). In *IEEE Transaction on Circuits and Systems-II*, volume 44, pages 847–855, Oct. 1997.

[52] A Bjorck and V Pereyra. Solution of Vandermonde systems of equations. In *Math. Comput.*, volume 24, pages 893–903, Oct. 1970.

[53] Surendra K Jain, Leilei Song, and Keshab K Parhi. Efficient semisystolic architectures for finite-field arithmetic. In *IEEE Transaction on VLSI systems*, volume 6, March 1998.

[54] C.C. Wu, Y.K. Leung, C.S. Chang, M.-H. Tsai, H.T. Huang, D.W. Lin, Y. M Sheu, C.-H. Hsieh, W.J. Liang, L. K. Han, W.M. Chen, S. Z. Chang, S.Y. Wu, S.S. Lin, H.C. Lin, C.H. Wang, P.W. Wang, T.-L. Lee, C.Y. Fu, C.W. Chang, S-C Chen, S.M. Jang, S. L. Shue, H.T. Lin, Y.C. See, Y.J. Mii, C.H. Diaz, B.J. Lin, M.-S. Liang, and Y.C. Sun. In *Electron Devices Meeting, 2002. IEDM '02. International, title=A 90-nm CMOS device technology with high-speed, general-purpose, and low-leakage transistors for system on chip applications*, pages 65–68, Dec 2002.