國 立 交 通 大 學

電子工程學系電子研究所

博 士 論 文

高面積效益軟性BCH及RS解碼器
Area-Efficient Soft BCH and RS Decoders

研 究 生 ：林義閔

指導教授 ：李鎮宜

張錫嘉

中華民國 100 年 六 月

高面積效益軟性 BCH 及 RS 解碼器

Area-Efficient Soft BCH and RS Decoders

研 究 生：	林義閔	Student：	Yi-Min Lin
指導教授：	李鎮宜　博士	Advisor：	Dr. Chen-Yi Lee
	張錫嘉　博士		Dr. Hsie-Chia Chang

國 立 交 通 大 學

電子工程學系電子研究所

博 士 論 文

A Dissertation

Submitted to Department of Electronics Engineering & Institute Electronics

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy

in

Electronics Engineering

June 2011

Hsinchu, Taiwan, Republic of China.

中華民國 100 年 六 月

高面積效益軟性 BCH 及 RS 解碼器

研究生：林義閔　　　　　　　　　　　　　指導教授：李鎮宜教授、張錫嘉教授

國立交通大學電子工程學系電子研究所

摘　要

本論文由演算法到架構設計與電路實現探討軟性 BCH 跟 RS 碼解碼器。依解碼方式不同可分成 EM 類型以及 Chase 類型解碼方式兩個主要部分做討論。

軟性錯誤更正碼解碼可以在維持相同碼率的前提下提升錯誤更正能力，而其中軟性 BCH 跟 RS 碼已經引起許多的研究風潮。和傳統的硬性解碼器相比，軟性解碼器雖能提供較好的更正能力，但必須付出高額的硬體代價。本論文首先提出 EM 類型解碼器來針對最不可靠的位置解碼以降低硬體複雜度。*低複雜度 EM 類型解碼*和傳統硬性解碼器相比，可以提供比較低的硬體複雜度，但是必須接受較高可靠度的資訊才能維持住錯誤更正能力。另一方面，*高效能 EM 類型解碼*使用和低複雜度 EM 類型解碼相似的概念，但是提供一個錯誤補償方式在非不可靠位置額外更正一個錯誤。因而，其軟性解碼器可以在提供和硬性解碼器相似的硬體複雜度前提下擁有較好的錯誤更正能力。

本論文提出了 Chase 類型解碼演算法來提供更通用的 BCH 跟 RS 碼軟性解碼方式。不像傳統 Chase 解碼器同時使用多個硬性解碼器來解多個候選數列產生其對應候選碼，所提出的簡易決定軟性解碼方式只需一個硬性解碼器即可執行 Chase 演算法。此外，透過漢明距離的運算，一個簡化的決定器被設計來決定最有可能的候選碼。而在所提出的限定決定解碼演算法中，可進一步地省略決定器的運算。透過限定由關鍵方程式運算器產生的錯誤位置方程式的次數，所提出的解碼方式只需完整解碼一組候選數列。

基於上述各項技術，我們實作了四個解碼器。一個 26.9 K 314.5 Mb/s 的軟性BCH (32400, 32208; 12) 解碼器晶片使用低複雜度EM類型解碼方式首先被設計應用在DVB-S2 系統上。實現在 90 奈米製程下，所提出的軟性BCH解碼器可以提供 314.5 Mb/s 的資料輸出量，且和一個擁有 99.3 Mb/s資料輸出量的傳統硬性BCH解碼器相比，其邏輯閘數量僅有硬性解碼器的一半。第二個解碼器則是將高效益EM類型解碼方式應用BCH (255, 239; 2) 跟BCH (255, 231; 3) 解碼器上。和傳統硬性解碼器相比，所設計的軟性BCH解碼器透過錯誤補償的方式最多可在 10^{-5} BER下得到 0.75 dB的更正效能增益，同時擁有 5%的硬體複雜度改善。第三個解碼器是應用在mmWave系統的軟性RS (224, 216; 4) 解碼器，其使用簡易決定軟性解碼方式用以達到30 K邏輯閘數和2.5 Gb/s的資料輸出量。和傳統的硬性解碼相比，可在 10^{-5} BER時擁有 0.5 dB的更正效能增益。最後一個解碼器則是設計於光通訊系統的軟性RS (255, 239; 8) 解碼器。其使用限定決定軟性解碼方式用以達到 45.3 K邏輯閘數和 2.56 Gb/s的資料輸出量。且在 10^{-4} CER下可比硬性解碼器多擁有 0.4 dB的更正效能。所有的實驗結果皆顯示我們所提的方式能得到如預期的成效。

Area-Efficient Soft BCH and RS Decoders

Student: Yi-Min Lin Adviser: Chen-Yi Lee and Hsie-Chia Chang

Department of Electronics Engineering & Institute Electronics

National Chiao Tung University

Abstract

This dissertation investigates the soft BCH and RS decoders from algorithms to architecture designs and circuit implementation. Two different decoding schemes are studied, including the *error magnitude (EM) type* and *Chase-type* soft decoding algorithms.

For higher error correcting performance with the same code rate, soft decoding algorithms of error control codes are the most popular methods and have aroused many research interests in BCH and RS decoding. As compared with traditional hard decoders, soft decoders provide better error correcting performance but much higher hardware complexity. In this dissertation, the EM-type soft decoding algorithms are firstly proposed for BCH codes to provide low hardware complexity by dealing with the least reliable bits. The *low complexity EM-type approach* can provide lower hardware complexity than the hard decoder but has to exploit higher reliable soft information for maintaining the error correcting performance. On the other hand, the *high performance EM-type approach* has similar concept as low complexity EM-type approach but compensates one extra error outside the least reliable set, leading to better performance while providing comparable hardware complexity.

The Chase-type soft decoding algorithms are discussed for providing more general low complexity decoding methods for both BCH and RS codes. Instead of utilizing various hard decoders to decode all candidate sequences simultaneously, a *decision-eased* soft decoding scheme is provided to process Chase algorithm with one hard decoder module. In addition, a simplified decision making unit is proposed to determine the most likely codeword with Hamming distance calculations. Moreover, the decision making unit can be eliminated with the proposed *decision-confined* soft decoding algorithm. By confining the degree of error location polynomial generated from the key equation solver, our proposal only needs to completely decode one candidate sequence.

Four implemented works are presented in this dissertation. A 26.9 K 314.5 Mb/s soft (32400, 32208; 12) BCH decoder chip is designed for DVB-S2 system based on low complexity EM-type approach. The proposed soft BCH decoder can achieve 314.5 Mb/s with 50.0% gate-count reduction in contrast to a 99.3 Mb/s traditional hard BCH decoder in CMOS 90 nm technology. The second designs are high performance EM-type soft BCH (255, 239; 2) and (255, 231; 3) decoders. Our proposed soft BCH decoders can achieve at most 0.75 dB coding gain at 10^{-5} BER with one extra error compensation and 5% less area than traditional hard BCH decoders. The third design is a 30 K 2.5 Gb/s decision-eased soft RS (224, 216; 4) decoder for millimeter-wave (mmWave) system, which has 0.5 dB coding gain at 10^{-5} BER as compared with the conventional hard decoder. The remaining design is a decision-confined soft RS (255, 239; 8) decoder chip for optical communications, which can provide 0.4 dB coding gain at 10^{-4} CER over hard decoders and achieve 2.56 Gb/s throughput with gate count of 45.3 K. All the implementation results reveal the positive consequence as expected.

誌　謝

在博士求學生涯中，承蒙許多師長提攜、朋友的協助以及家人的支持，讓我能一路順利地走到終點。非常感謝指導教授李鎮宜老師和張錫嘉老師提供了最好的研究環境，給予我嘗試各式各樣研究方向的機會。他們的諄諄教誨賦予我面對更多挑戰的信心，他們的啟蒙與指導使得我能在研究上有所發揮，能夠身處於這樣的研究環境，是做為學生的幸福。

接著我要謝謝 Si2 實驗室跟 Ocean 研究團隊的所有同仁。感謝每一位實驗室同仁相助，讓我得以在此學習成長，吸收許多寶貴經驗。在學業上，透過與大家的討論，使得研究更加完善充實；在生活上，也因為有了各位，在面對研究挑戰的路途上，多了許多歡笑和愉悅。

最後由衷地向家人獻上最真摯的感謝。你們無悔的付出，讓我無後顧之憂地完成博士論文。特別感謝我的太太立芯，這些日子多虧你的耐心等待，並教會我如何享受生活，讓我平淡的生活中多了許多美好的回憶。

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Thesis Background

While the digital information data are sent from a source to one or more destinations, several procedures will be involved to combat the noisy channel for efficient and reliable transmission. A traditional digital communication system can be depicted as Fig. 1.1, where the channel can be microwave links, wireline cables, or storage mediums. In general, the information data are binary symbol sequences and are compressed by the source encoder for transmitting a shorter bit sequence. The channel encoder then transforms these compressed data into a longer sequence called a *codeword* with some redundant symbols. These symbols, also known as parity check symbols, are exploited for allowing errors which are introduced into the channel to be corrected. Afterward, the modulator converts the channel coded symbols into analog signals appropriate for transmission over the channel. The channel noise, such as thermal noise, signal attenuation, distortion, and interference, will corrupt the transmitted data. In the receiver, the demodulator estimates the signal from the channel

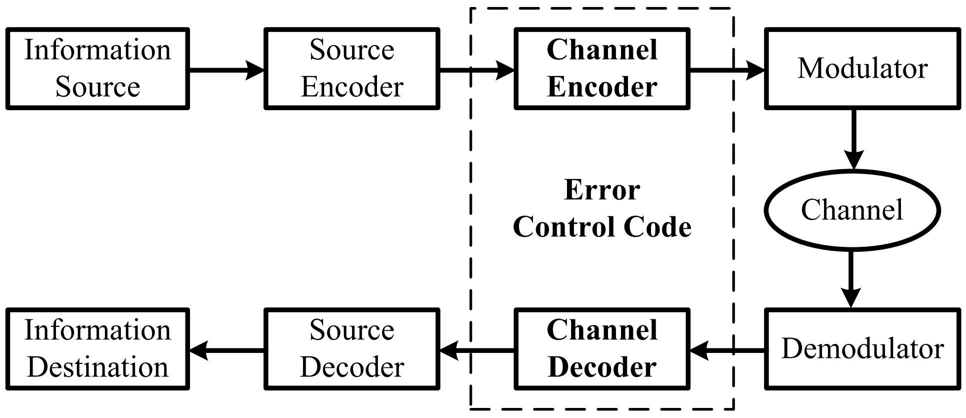


Figure 1.1: Block diagram of a digital communication system

outputs and converts it into a continuous or quantized symbols. Due to the influence of channel, the demodulated data might contain errors. If so, the channel decoder can utilize the parity check symbols to recover the transmitted information. An ideal channel decoder is able to remove all the errors and produce the data sequence identical with the compressed one. Finally, the source decoder decompresses such sequence to reproduce the information source.

The channel encoder and decoder are essential to the digital communication system for transmitting complete and accurate information through the noisy channel. The development of this study, referred to *channel coding* or *error control coding*, begins with the landmark paper by C. E. Shannon in 1948 [1, 2]. Shannon's channel coding theorem indicated that any asymptotically error-free transmission is achievable with appropriate coding as long as the information rate R in bits per channel is less than the *channel capacity* C for the channel. Based on this theorem, the theoretical performance limit can be calculated under different signaling schemes, code rates, and channels. After Shannon's proposal, a lot of coding researches are developed for devising efficient error control methods to resist noisy environment.

There are two major types of codes: *block codes* and *convolutional codes* [3, 4]. The convolutional code with the probabilistic decoding feature was first introduced by P. Elias in 1955 [5]. In 1961, J. M. Wozencraft and B. Reiffen proposed the efficient sequential decoding method [6], and A. J. Viterbi presented the Viterbi algorithm in 1967 [7] for simplifying the hardware implementation. Afterward, the convolution codes have been widely applied in wireless communications, such as wireless local area network (WLAN), ultra-wide band (UWB) and third generation (3G) mobile wireless communications [8–10]. On the other hand, the block codes have algebraic coding structures [11], and the first block codes were discovered by R. E. Hamming in 1950 called Hamming codes [12] which are designed for single error correction. For the multiple error correction block codes, Reed-Muller codes were first introduced by D. E. Muller in 1954 [13] and completed by I. S. Reed [14]. In 1960, Bose and Ray-Chaudhuri [15], and Hocquenghem [16] made significant headway by finding a large class of multiple error correcting codes, referred to BCH codes. In the meantime, I. S. Reed and G. Solomon proposed Reed-Solomon (RS) codes [17], which have been classified into non-binary BCH codes [18]. Subsequently, the block codes have been well accepted in storage devices, such as DVD and flash memories, and digital communications, either wireless or wireline systems [19–24].

Among these well-know error correcting codes, the BCH and the RS codes are the most commonly used cyclic block codes, where the cyclic feature has additional algebraic structure to make encoding and decoding more efficient. In the error control coding theory, the error correcting ability t of algebraic block codes is the largest integer satisfying $d^* \geq 2t+1$, and the *minimum distance* d^* is the total amount different places between the two most similar codewords. With each of the codewords as the the origin, nonintersecting spheres of radius t can be drawn as shown in Fig. 1.2. Then the received data is always in a sphere and can be decoded as the codeword at the center of that sphere if t or fewer errors occur. With block length of N symbols and information length of K symbols, an $(N, K; t)$ BCH code and an $(N, K; t)$ RS code operating under $GF(2^m)$ can correct up to t errors with $N - K \leq m \times t$ and $t = \left\lfloor \frac{N-K}{2} \right\rfloor$ respectively [3, 4].

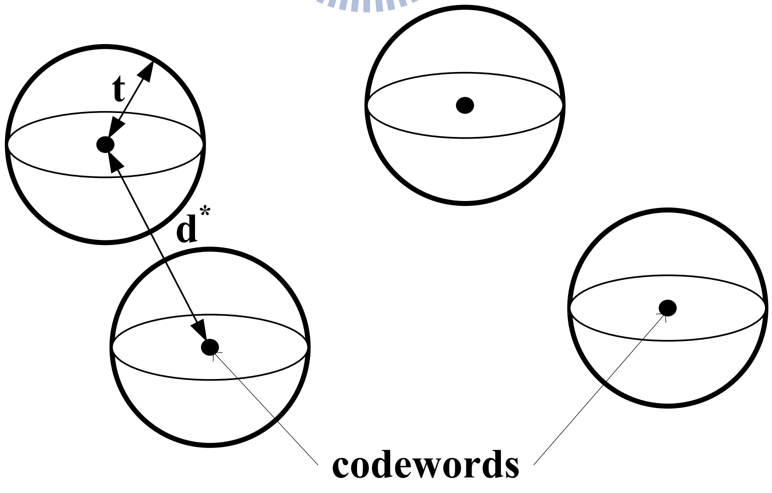


Figure 1.2: Decoding spheres

1.2 Thesis Motivation

For recent applications, the increasing data rate results in the higher uncertainty of received signals. To provide higher error correcting performance with the same code rate, soft decoding algorithms of error control codes exploit the reliability information from the channel to correct an error number larger than half the minimum distance for a code. Many research interests have been aroused in BCH and RS soft decoding methods [25–32]. In 1966, Forney developed the generalized-minimum-distance (GMD) [25], which uses soft information to generate test sequences for several hard decoders to form a list of candidate codewords. A decision making unit is applied to choose the most likely one from the candidate list. With the similar concept, Chase algorithm [26] and Chase-GMD methods [27] are also widely used to efficiently generate the candidate list and applied in many applications. In 2003, Koetter and Vardy introduced the algebraic soft-decision decoding algorithm [28] based on the list decoding [29] and the soft information are exploited to interpolate each symbol, leading to larger coding gain. In above methods, the soft informations are only used to generate test sequences. However, several algorithms based on probability propagation have been developed to employ the soft information for improving error correcting performance. A sub optimum maximum a posteriori (MAP) algorithm [30] with a Hamming SISO decoder was proposed by Therattil and Thangaraj in 2005 [31]. The adaptive belief propagation algorithm, often used in LDPC decoding, was firstly applied in soft BCH and RS decoding in 2006 by Jiang [32]. However, the complex computations of soft decoding algorithms still make soft decoders several times hardware complexity in contrast to hard decoders although the VLSI architectures of algebraic soft decoders have been developed in these years [33–35].

In this thesis, we will focus on the approaches for reducing the complexity of soft BCH and RS decoders. Two types of soft decoders are implemented, and both of them exploit soft informations to determine the least reliable bits. The first type evaluates the error magnitudes in these unreliable bits and uses the characteristic of BCH codes that the valid error magnitudes are binary values in the decoding procedure. The second type utilizes these unreliable bits to process Chase decoding but improves the complex calculations of the decision making procedure. The corresponding implementation results can prove the feasibility of the proposed designs.

1.3 Thesis Organization

This thesis consists of 6 chapters. Chapter 2 gives a general introduction of BCH and RS codes. It reviews the conventional code structure and decoding algorithm, including both hard-decision and soft-decision methods. In Chapter 3, the error magnitude (EM) type decoding algorithms are presented for BCH codes. According to the different reliability of input informations, low complexity and high performance EM-type approaches are provided respectively. For supporting both BCH and RS codes, the Chase-type decoding algorithms are proposed in Chapter 4. The complex calculation of determining the output codeword from a list of candidate codewords is the bottleneck of Chase-type soft decoder design. At the first place, we show that the complexity of decision making procedure can be eased with Hamming distance calculation. Afterward, the decision-confined approach is presented by confining the degree of error location polynomial generated by the key equation solver. Chapter 5 presents implementation results of four soft decoders and gives the state of the art. Finally, we conclude the dissertation in Chapter 6.

Chapter 2

Review of BCH and RS Codes

BCH codes were first constructed by Bose and Ray-Chaudhuri [15], and Hocquenghem [16] in individual research in 1960. At the same time, Reed-Solomon (RS) codes, which have been classified into non-binary BCH codes [18]. were proposed by I. S. Reed and G. Solomon [17]. As shown in Chapter 1, both the BCH and RS codes have become the most important algebraic block codes. The block lengths of an $(N, K; t)$ BCH code and an $(N, K; t)$ RS code are N symbols consisting of K message symbols in $GF(2)$ and $GF(2^m)$ respectively. With the minimum distance of $2t + 1$, the error correcting capability t are $N - K \leq m \times t$ and $t = \lfloor \frac{N-K}{2} \rfloor$ for BCH and RS codes respectively over $GF(2^m)$ [3, 4]. Because of the effective error correction, BCH and RS codes have enjoyed countless applications, including data storage systems and digital communications, either wireless or wireline systems [19–24].

The organization of this chapter is as follows. Section 2.1 introduces the basic concepts of finite field, which is the preliminary knowledge of mathematical background for BCH and RS codes. The overview of BCH codes and RS codes, including the encoding and decoding procedures, is described in Section 2.2 and Section 2.3 respectively. At the end of Chapter 2, Section 2.4 reviews five soft decoding algorithms for BCH and RS codes.

2.1 Finite Field

Before the description of finite fields, the definition of a group G is presented first. A group G is set of at least two elements on which we can do a binary (algebraic) operation "$*$" without leaving the set, where the binary operation "$*$" is defined to satisfy the following conditions:

a) The associative law holds under "$*$" operation for every $a, b, c \in G$, ie., $a * (b * c) = (a * b) * c \;\; \forall a, b, c \in G$.

b) There exists an element e in G such that $a * e = e * a = a$ for all a in G, where e is called the *identity* of G.

c) For any a of G, there exists a' in G such that $a * a' = a' * a = 1$, where a' is called the *inverse* of a.

Notice that a group G is commutative (Abelian) if $a * b = b * a$ for all a, b in G.

Based on the definition of group G, a field F is defined as a set of at least two elements with two algebraic operations, addition "+" and multiplication "$\cdot$". Addition, subtraction (or additive inverse), multiplication and division (or multiplicative inverse) can be operated in F and the conditions have to be satisfied under these operations as follows:

a) F is an Abelian group under "+" operation with identity element denoted by 0.

b) F is closed under "$\cdot$" opeartion and the set of nonzero elements in F is an Abelian group under "$\cdot$" opration with identity element denoted by 1.

c) The distributive law holds for every $a, b, c \in F$, ie., $a \cdot (b + c) = a \cdot b + a \cdot c \;\; \forall a, b, c \in F$.

The number of elements in a field, or the *order* of the field, can be infinite or finite. The set of all real numbers is one of the well-known example of infinite field. If F has finite elements, F is called a *finite field* or *Galois field* (GF). However, finite fields may not be constructed with arbitrary number of elements. As proved in [36], the number of elements in a finite field is a prime number p or is a power of a prime number. A finite field $GF(p)$, which has the elements $\{0, 1, \ldots, p - 1\}$, can be constructed according to modulo p arithmetic. Moreover, an *extension field* of $GF(p)$, $GF(p^m)$, can be built with p^m elements for any positive integer m and can be considered as a vector space of dimension m over $GF(p)$, where p is called the *characteristic* of $GF(p^m)$. Any element in $GF(p^m)$ can be represented by a corresponding set of m tuples, ie., $\hat{a} = (a_0, a_1, \ldots, a_{m-1})$, or can be denoted by a polynomial with coefficients over $GF(p)$ and degree less than or equal to $m - 1$, ie., $\hat{a} = a_0 + a_1 x + \cdots + a_{m-1} x^{m-1}$. In a finite field $GF(p)$, a m-degree polynomial $f(x)$ is said to be *irreducible* if $f(x)$ is not divisible by any non-zero polynomial over $GF(p)$ of degree less than m. The addition over $GF(p^m)$ is defined component-wise while the multiplication is defined based on the modulo of an operation irreducible $f(x)$. Therefore, the addition and multiplication operations between any $\hat{a}, \hat{b} \in GF(p^m)$ can be proceeded as follows.

$$\begin{aligned} \hat{a} + \hat{b} &= (a_0 + b_0) + (a_1 + b_1)x + \cdots + (a_{m-1} + b_{m-1})x^{m-1}. && (2.1) \\ \hat{a} \cdot \hat{b} &= (a_0 + a_1 x + \cdots + a_{m-1}x^{m-1})(b_0 + b_1 x + \cdots + b_{m-1}x^{m-1})] \mod f(x). && (2.2) \end{aligned}$$

The *minimum polynomial* of β, $m_\beta(x)$, is the least degree polynomial with coefficients in $GF(p)$ such that $m_\beta(\beta) = 0$ and it is an irreducible polynomial over $GF(p)$. Notice that although $m_\beta(x)$ is irreducible over $GF(p)$, it can be factored over $GF(p^m)$. Since β is a root of $m_\beta(x)$, its conjugates $\{\beta^p, \beta^{p^2}, \dots \beta^{p^r-1}\}$ are also roots of $m_\beta(x)$, where r is the smallest integer that results in $\beta^{q^r} = \beta$ and r must be a factor of m [36,37]. The minimal polynomial $m_\beta(x)$ with degree of r can be expressed as:

$$m_\beta(x) = (x - \beta)(x - \beta^p) \cdots (x - \beta^{p^{r-1}}). \tag{2.3}$$

According to definition of field, the power of a nonzero element β in $GF(p^m)$ is still in $GF(p^m)$. Since there are $p^m - 1$ nonzero elements in $GF(p^m)$, β^i has at most $p^m - 1$ distinct values for all integers i, implying there exists an integer s that leads $\beta^{i+s} = \beta^i$ or $\beta^s = 1$ with $1 \leq s \leq p^m - 1$. The smallest integer s which satisfies $\beta^s = 1$ is referred to the *order* of β. If a nonzero element α in $GF(p^m)$ has $p^m - 1$ order, it is called a *primitive element*. Then all the elements in $GF(p^m)$ can be represented with 0 and α:

$$GF(p^m) = \{0, \alpha, \alpha^2, \dots, \alpha^{p^m-1} = 1\}. \tag{2.4}$$

Besides, the minimum polynomial of α is called the *primitive polynomial p(x)* with degree of m. Notice that not all irreducible polynomials are primitive but all the primitive

Table 2.1: Representation of the elements in $GF(2^4)$

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

polynomials are irreducible. Hence, we can exploit the primitive polynomials to construct the finite fields. An example of $GF(2^4)$, an extension filed of $GF(2)$, is shown in Table 2.1 with the primitive polynomial $p(x) = x^4 + x + 1$ in standard basis. There are three representations for an element of $GF(p^m)$. Since the primitive element α is a root of $p(x)$, $p(\alpha) = \alpha^4 + \alpha + 1 = 0$. Any nonzero element of $GF(2^4)$ can be represented as power of α or can be expressed as a polynomial with degree < 4 according to the equation $\alpha^4 = \alpha + 1$. Another representation for elements of $GF(2^4)$ is the 4-tuple, in which the four components are the four coefficients in the polynomial representation. Because the addition over $GF(p^m)$ is component-wise and the multiplication is based on the modulo of the primitive polynomial $p(x)$, the polynomial representation is convenient for addition and the power representation is convenient for multiplication. It is worth to be noticed that the binary field $GF(2)$ and its extension field $GF(2^m)$ are more attractive for digital applications because of the binary arithmetic operations. In the the remainder of the thesis, all the discussed error control codes are based on operation over $GF(2^m)$.

2.2 BCH Code

BCH code is a kind of cyclic codes and can be view as a generalization of the Hamming codes for multiple errors correction. An $(N, K; t)$ BCH code has block length of N bits and information length of K bits. While operating under $GF(2^m)$, it has the error correcting capability t with $N - K \leq m \times t$. In the encoding procedure, a generator polynomial $g(x)$ is defined by the least common multiple (LCM) of the minimum polynomials over $GF(2)$ $M_1(x)$, $M_2(x)$, $\ldots$, and $M_{2t}(x)$, where $M_i(x)$ is the minimal polynomial for α^i for $i = 1 \sim 2t$, and α is primitive in $GF(2^m)$.

$$g(x) = \text{LCM}\{M_1(x), M_2(x), \cdots, M_{2t}(x)\}. \tag{2.5}$$

Notice that, not only α^i for $i = 1 \sim 2t$ but also their conjugates $(\alpha^i)^{2^j}$ are roots of $g(x)$, implying that $M_{i\times 2^j}(x) = M_i(x)$ for each positive integer j and $1 \leq i \times 2^j \leq 2t$. Therefore, the generator polynomial $g(x)$ can be reduced to

$$g(x) = \text{LCM}\{M_1(x), M_3(x), \cdots, M_{2t-1}(x)\}, \tag{2.6}$$

and the degree of each minimal polynomial is a factor of m.

There are two type methods for encoding a BCH codeword: non-systematic encoding and systematic encoding. According to the coding theory, a codeword polynomial $c(x)$ must be a multiple of $g(x)$, that is $g(x)|c(x)$. From this definition, the non-systematic encoding of a message polynomial $u(x) = u_0 + u_1x + u_2x^2 + \cdots + u_{k-1}x^{k-1}$ can be directly obtained by multiplying $u(x)$ with $g(x) = g_0 + g_1x + \cdots + g_rx^r$.

$$c(x) = u(x) \cdot g(x). \tag{2.7}$$

The matrix formulation of (2.7) can be expressed as

$$\begin{bmatrix} c_0 & c_1 & \cdots & c_{N-1} \end{bmatrix} = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{K-1} \end{bmatrix}^T \begin{bmatrix} g_0 & g_1 & \cdots & \cdots & g_r & 0 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & \cdots & g_r & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & g_0 & g_1 & \cdots & \cdots & g_r \end{bmatrix}, \tag{2.8}$$

where the $K \times N$ generator matrix G is defined as

$$G = \begin{bmatrix} g_0 & g_1 & \cdots & \cdots & g_r & 0 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & \cdots & \cdots & g_r & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & g_0 & g_1 & \cdots & \cdots & g_r \end{bmatrix}. \tag{2.9}$$

On the other hand, the systematic encoding generates the remainder polynomial $r(x)$ at first according to the modulo operation between $u(x)x^{N-K}$ and $g(x)$:

$$r(x) = u(x) \times x^{N-K} \mod g(x). \tag{2.10}$$

Then the systematic code $c(x)$ is obtained by adding $r(x)$ to $x^{N-K}u(x)$:

$$c(x) = u(x) \times x^{N-K} + r(x). \tag{2.11}$$

Since $x^{N-K}u(x) = q(x)g(x) + r(x)$ where $q(x)$ is the quotient polynomial, the systematic code $c(x)$ becomes

$$c(x) = x^{N-K}u(x) + r(x) = q(x)g(x), \tag{2.12}$$

which is a multiple of $g(x)$, and the message polynomial $u(x)$ is shown in the K successive positions of $c(x)$. The advantage of the systematic encoding is that we can get $u(x)$ directly from part of $c(x)$ without any computation. Thus, it is the most common way for encoding a BCH codeword.

Because α^i for $1 \leq i \leq 2t$ are roots of $g(x)$, they are also roots of $c(x)$, indicating

$$c(\alpha^i) = c_0 + c_1\alpha^i + c_2\alpha^{2i} + \cdots + c_{N-1}\alpha^{(N-1)i} = 0, \text{ for } 1 \leq i \leq 2t. \tag{2.13}$$

Hence, the matrix formulation of (2.13) is

$$\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{(N-1)} \\ 1 & \alpha^2 & (\alpha^2)^2 & \cdots & (\alpha^2)^{(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{2t} & (\alpha^{2t})^2 & \cdots & (\alpha^{2t})^{(N-1)} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{N-1} \end{bmatrix}. \tag{2.14}$$

Then the $(N-K) \times N$ parity check matrix H of an $(N, K; t)$ BCH code can be defined as

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \cdots & \alpha^{(N-1)} \\ 1 & \alpha^2 & (\alpha^2)^2 & \cdots & (\alpha^2)^{(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{2t} & (\alpha^{2t})^2 & \cdots & (\alpha^{2t})^{(N-1)} \end{bmatrix}. \tag{2.15}$$

In the receiver, the received data $R(x)$ includes the transmitted codeword $c(x)$ and noise signal:

$$R(x) = c(x) + e(x), \tag{2.16}$$

where $e(x)$ is the error pattern. The syndrome polynomial corresponding $R(x)$ is defined as $S(x) = S_1 + S_2x^1 + \cdots + S_{2t}x^{2t-1}$, in which

$$S_i = R(\alpha^i) = c(\alpha^i) + e(\alpha^i) = e(\alpha^i), \text{ for } 1 \leq i \leq 2t. \tag{2.17}$$

If $e(x)$ has v $(\leq t)$ errors in the positions $j_1, j_2, \cdots, j_v$ with $0 \leq j_1 < j_2 < \cdots < j_v \leq N-1$, $e(x)$ can be expressed as

$$e(x) = x^{j_1} + x^{j_2} + \cdots + x^{j_v}. \tag{2.18}$$

According to (2.17) and (2.18), the i-th coefficient of $S(x)$ will be

$$\begin{aligned} S_i &= (\alpha^{j_1})^i + (\alpha^{j_2})^i + \cdots + (\alpha^{j_v})^i \\ &= \beta_1^i + \beta_2^i + \cdots + \beta_v^i, \end{aligned} \tag{2.19}$$

where $\beta_x = \alpha^{j_x}$ for $x = 1, 2, \ldots, v$. Then the relation between $\{S_1, S_2, \cdots, S_{2t}\}$ and $\{\beta_1, \beta_2, \cdots, \beta_v\}$ can be derived as

$$\begin{aligned} S_1 &= \beta_1 + \beta_2 + \cdots + \beta_v \\ S_2 &= \beta_1^2 + \beta_2^2 + \cdots + \beta_v^2 \\ S_3 &= \beta_1^3 + \beta_2^3 + \cdots + \beta_v^3 \\ &\vdots \\ S_{2t} &= \beta_1^{2t} + \beta_2^{2t} + \cdots + \beta_v^{2t} \end{aligned} \tag{2.20}$$

To solve $e(x)$, an *error location polynomial* is defined as

$$\begin{aligned} \Lambda(x) &= (1-\beta_1 x)(1-\beta_2 x)\cdots(1-\beta_v x) \\ &= \Lambda_0 + \Lambda_1 x + \cdots + \Lambda_v x^v, \end{aligned} \tag{2.21}$$

whose roots are β_1^{-1}, β_2^{-1}, $\cdots$, and β_v^{-1}. After observation, we can see that the relation between $\{\Lambda_0, \Lambda_1, \cdots, \Lambda_v\}$ and $\{\beta_1, \beta_2, \cdots, \beta_v\}$ is as follows:

$$\begin{aligned} \Lambda_0 &= 1 \\ \Lambda_1 &= \beta_1 + \beta_2 + \cdots + \beta_v \\ \Lambda_2 &= \beta_1\beta_2 + \beta_2\beta_3 + \cdots + \beta_{v-1}\beta_v \\ &\vdots \\ \Lambda_v &= \beta_1\beta_2\cdots\beta_v \end{aligned} \tag{2.22}$$

Furthermore, from (2.20) and (2.22), Peterson proposed the first decoding algorithm for BCH codes in 1960 according to the following *Newton's identities* [38]:

$$\begin{aligned} S_i + \Lambda_1 S_{i-1} + \cdots + \Lambda_{i-1} S_1 + i\Lambda_i = 0, &\text{ for } 1 \le i \le v. \\ S_i + \Lambda_1 S_{i-1} + \cdots + \Lambda_{v-1} S_{i-v+1} + \Lambda_v S_{i-v} = 0, &\text{ for } i > v. \end{aligned} \tag{2.23}$$

That is,

$$\begin{aligned} i = 1 &\ : \ S_1 + \Lambda_1 = 0 \\ i = 2 &\ : \ S_2 + \Lambda_1 S_1 + 2\Lambda_2 = 0 \\ i = 3 &\ : \ S_3 + \Lambda_1 S_2 + \Lambda_2 S_1 + 3\Lambda_2 = 0 \\ &\ \vdots \\ i = v &\ : \ S_v + \Lambda_1 S_{v-1} + \Lambda_2 S_{v-2} + \cdots + v\Lambda_v = 0 \\ i = v+1 &\ : \ S_{v+1} + \Lambda_1 S_v + \Lambda_2 S_{v-1} + \cdots + S_1 \Lambda_v = 0 \\ &\ \vdots \\ i = 2t &\ : \ S_{2t} + \Lambda_1 S_{2t-1} + \Lambda_2 S_{2t-2} + \cdots + \Lambda_v S_{2t-v} = 0 \end{aligned} \tag{2.24}$$

That means $\Lambda(x)$ can be obtained once $S(x)$ is calculated. However, it is inefficient to directly solve (2.24) for large v.

In addition to Peterson's method, $\Lambda(x)$ can be solved by the definition of the *key equation* [39]:

$$S(x)\Lambda(x) = \Omega(x) \mod x^{2t}. \tag{2.25}$$

In(2.24), there is a linear feedback shift register relationship between $\{S_1, S_2, \cdots, S_{2t}\}$ and $\{\Lambda_0, \Lambda_1, \cdots, \Lambda_v\}$ for $i > v$:

$$S_i = \sum_{j=1}^{v} \Lambda_j S_{i-j}, \tag{2.26}$$

which can be expressed as a matrix form

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_{v+1} \\ S_2 & S_3 & \cdots & S_{v+2} \\ S_3 & S_4 & \cdots & S_{v+3} \\ \vdots & \vdots & & \vdots \\ S_{2t-v} & S_{2t-v+1} & \cdots & S_{2t} \end{bmatrix} \begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \vdots \\ \Lambda_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{2.27}$$

Berlekamp-Massey algorithm is an efficient iterative algorithm for solving key equation (2.25) which was developed by Berlekamp [39, 40] and independently by Massey [41, 42]. This algorithm iteratively generates $\Lambda(x)$ for satisfying (2.27). The detailed procedure is given as follows.

- Initial conditions:

$$\Lambda^{(0)}(x) = 1, \quad \tau^{(0)}(x) = 1,$$
$$\Delta_0 = S_1, \quad \delta = 1, \qquad L_0 = 0.$$

- Iteration from $i = 1$ to $2t$:

$$\Lambda^{(i)}(x) = \Lambda^{(i-1)}(x) + \frac{\Delta_{i-1}}{\delta}\tau^{(i-1)}(x) \cdot x. \tag{2.28}$$

$$\Delta_i = \sum_{j=0}^{t-1} \Lambda_j^{(i)} S_{i+1-j}. \tag{2.29}$$

If $\Delta_{i-1} = 0$ or $L_{i-1} \geq i - L_{i-1}$,

$$\tau^{(i)}(x) = \tau^{(i-1)}(x) \cdot x,$$
$$L_i = L_{i-1}.$$

Otherwise,

$$\tau^{(i)}(x) = \Lambda^{(i-1)}(x),$$
$$L_i = i - L_{i-1}, \qquad \delta = \Delta_{i-1}.$$

After $2t$ iterations, the error location polynomial is obtained from $\Lambda(x) = \Lambda^{2t}(x)$. In (2.28), the operation compensates the difference to make $\Lambda(x)^i$ satisfy the discrepancy $\Delta_{i-1} = 0$. And the operation in (2.29) evaluates a new discrepancy Δ_i to verify the correctness of $\Lambda(x)^i$. Besides, the control signal L indicates the degree of $\Lambda(x)$.

Another famous algorithm for solving the key equation is Euclidean algorithm [43, 44], which rewritten the key equation as

$$S(x)\Lambda(x) + Q(x)x^{2t} = \Omega(x). \tag{2.30}$$

Notice that $Q(x)$ is the quotient polynomial and $\Omega(x)$ is the remainder polynomial while $S(x)\Lambda(x)$ is divided by x^{2t}. According to Euclidean algorithm, $\Lambda(x)$ and $\Omega(x))$ can be obtained by deriving the greatest common divisor (GCD) algorithm between $S(x)$ and x^{2t} with a termination condition. The algorithm proceeds as follows.

- Initial conditions

$$\Omega^{(-1)} = x^{2t}, \quad \Omega^{(0)} = S(x)$$
$$\Lambda^{(-1)}(x) = 0, \quad \Lambda^{(0)}(x) = 1$$

- Iterations from $i = 1$:

$$\Omega^{(i)}(x) = Q^{(i)}(x)\Omega^{(i-1)}(x) + \Omega^{(i-2)}(x) \tag{2.31}$$
$$\Lambda^{(i)}(x) = Q^{(i)}(x)\Lambda^{(i-1)}(x) + \Lambda^{(i-2)}(x) \tag{2.32}$$

- Iteration terminated

$$deg(\Lambda(x)) > deg(\Omega(x))$$

The error location polynomial is acquired as long as the iteration is terminated and $\Lambda(x) = \Lambda^{(i)}(x)$. In the decoding procedure, the degree of $\Omega^{(i)}(x)$ decreases with i whereas $\Lambda^{(i)}(x)$ has an increasing degree. Once the degrees of $\Lambda(x)$ and $\Omega(x)$ satisfy the stopping criterion, $\Lambda(x))$ has the degree v and $\Omega(x)$ has the degree at most $v - 1$.

As a result, the decoding procedure contains three major steps: *syndrome calculator*, *key equation solver* and *Chien search* as shown in Fig. 2.1. If the syndromes $\{S_1, S_2, \cdots, S_{2t}\}$ are

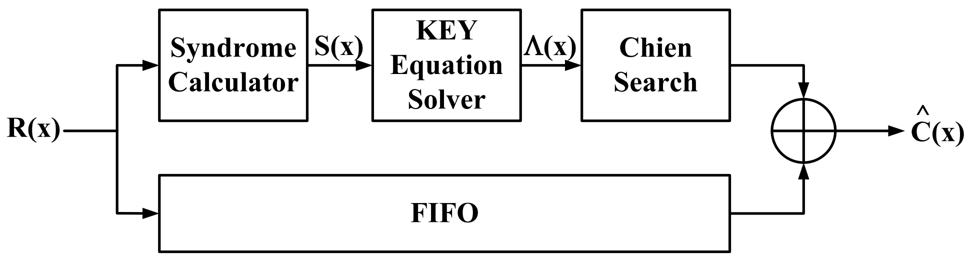


Figure 2.1: Binary BCH decoding process

calculated according to (2.19), the error location polynomial can be obtained by using Peterson's method or by solving the key equation (2.25) with either Berlekamp-Massey algorithm or Euclidean algorithm. The error locations can be found by calculating $\{\beta_1, \beta_2, \cdots, \beta_v\}$ through solving the roots of $\Lambda(x) = 0$. However, it is not easy to find a root of a polynomial by directly solving the equation. Since the number of elements in $GF(2^m)$ is finite, an efficient process known as *Chien search* [45] is exploited to substitute each nonzero element of $GF(2^m)$ into the equation. If α^{-i} is a root of $\Lambda(x)$, an error is assumed at the $i-th$ position. Finally, the estimated codeword polynomial $\hat{c}(x)$ is obtained by inverting those values at error locations in $R(x)$.

2.3 RS Code

RS code is a kind of cyclic codes and can be classified into non-binary BCH codes [18]. An $(N, K; t)$ RS code over $GF(2^m)$ has block length of N symbols and information length of K symbols, where a symbol consists of m-bit. It has the error correcting capability $t = \left\lfloor \frac{N-K}{2} \right\rfloor$ for error only decoding and can correct up t errors and ρ erasures with $t = \left\lfloor \frac{N-K-\rho}{2} \right\rfloor$ for error and erasure decoding. Notice that, an erasure is defined to be an error with a known error location. For the encoding procedure, there are two distinct constructions for RS codes. Although these two constructions initial appear to describe different codes, it is can be proved that the families of codes described are in fact equivalent according to Galois field Fourier transform techniques [4]. No matter which construction method is applied, the number of parity check symbols are $2t$ and the minimum distance is $2t + 1$.

The first RS code construction is proposed by I. S. Reed and G. Solomon in 1960 [17].

The original definition of RS codes is based on evaluation map encoding method: a codeword $c(x)$ is formed by evaluating the message polynomial $u(x) = u_0 + u_1 x + u_2 x^2 + \cdots + u_{K-1} x^{K-1}$ at N elements of $GF(2^m)$.

$$\begin{aligned} c(x) &= c_0 + c_1 x + c_2 x^2 + \cdots + c_{N-1} x^{N-1} \\ &= u(1) + u(\alpha) x + u(\alpha^2) x^2 + \cdots + u(\alpha^{N-1}) x^{N-1}. \end{aligned} \tag{2.33}$$

It is obvious that this construction method is a non-systematic encoding. This evaluation map encoding method is useful because it provides insight leading to interpolation-based decoding algorithms, such as Guruswami-Sudan (GS) and Koetter-Vardy (KV) algorithms, and we will discuss these algorithms in Section 2.4.

However, the second RS code construction is more popular because of its relation with BCH codes and their associated decoding algorithms. According to the definition of generator polynomial in (2.5), the $g(x)$ of a RS code that corrects up to t errors is constructed with the minimum polynomials of $\alpha, \alpha^2, \alpha^3, \cdots$, and α^{2t}, where α is a primitive element in $GF(2^m)$. Notice that, the minimal polynomial $m_i(x)$ over $GF(2^m)$ for any element α^i is simply $m_i(x) = x - \alpha^i$. Therefore, the generator polynomial becomes

$$g(x) = (x - \alpha)(x - \alpha^2) \cdots (x - \alpha^{2t}). \tag{2.34}$$

The codeword can be generated either in the non-systematic form of (2.7) or in the systematic form of (2.11). Both non-systematic and systematic encoding schemes make the codeword $c(x)$ be a multiple of $g(x)$, indicating that $c(\alpha^i) = 0$ for $0 \leq i \leq 2t - 1$. In this thesis, most of the encoders and decoders are concerned with the second code construction.

The decoding of RS codes is quite similar to BCH code except for the requirement of error value calculation. In the receiver, the received data $R(x)$ is the transmitted codeword $c(x)$ corrupted by the error pattern $e(x)$:

$$R(x) = c(x) + e(x). \tag{2.35}$$

The syndrome polynomial corresponding $R(x)$ is defined as $S(x) = S_1 + S_2x^1 + \cdots + S_{2t}x^{2t-1}$, in which

$$S_i = R(\alpha^i) = c(\alpha^i) + e(\alpha^i) = e(\alpha^i), \text{ for } 1 \leq i \leq 2t. \tag{2.36}$$

If $e(x)$ has v $(\leq t)$ errors in the positions $j_1, j_2, \cdots, j_v$ with $0 \leq j_1 < j_2 < \cdots < j_v \leq N-1$, $e(x)$ can be expressed as

$$e(x) = e_1x^{j_1} + e_2x^{j_2} + \cdots + e_vx^{j_v}, \tag{2.37}$$

where $e_1, e_2, \cdots, e_v$ are error values corresponding to error locations $j_1, j_2, \cdots, j_v$. Therefore, the i-th coefficient of $S(x)$ will be

$$\begin{aligned} S_i &= e_1(\alpha^{j_1})^i + e_2(\alpha^{j_2})^i + \cdots + e_v(\alpha^{j_v})^i \\ &= e_1\beta_1^i + e_2\beta_2^i + \cdots + e_v\beta_v^i, \end{aligned} \tag{2.38}$$

where $\beta_x = \alpha^{j_x}$ for $x = 1, 2, \ldots, v$. The syndrome polynomial $S(x)$ becomes

$$\begin{aligned} S(x) &= \sum_{i=0}^{2t-1} S_ix^i \\ &= \sum_{i=0}^{2t-1}\sum_{j=1}^{v} e_j\beta_j^{i+1}x^i \\ &= \sum_{j=1}^{v}\sum_{i=0}^{2t-1} e_j\beta_j^{i+1}x^i \\ &= \sum_{j=1}^{v} \frac{e_j\beta_j(1-(\beta_jx)^{2t})}{1-\beta_jx}. \end{aligned} \tag{2.39}$$

The error location polynomial $\Lambda(x)$ is also defined as shown in (2.21). The relation of $S(x)$

and $\Lambda(x)$ can be formulated as

$$\begin{aligned} S(x)\Lambda(x) &= \sum_{j=1}^{v} \frac{e_j\beta_j(1-(\beta_j x)^{2t})}{1-\beta_j x} \times \prod_{i=1}^{v}(1-\beta_i x) \\ &= \sum_{j=1}^{v} e_j\beta_j \prod_{i=1,i\neq j}^{v}(1-\beta_i x) - \sum_{j=1}^{v} e_j\beta_j(\beta_j x)^{2t} \prod_{i=1,i\neq j}^{v}(1-\beta_i x). \end{aligned} \tag{2.40}$$

According to the definition of key equation in (2.25), the error evaluator polynomial $\Omega(x)$ becomes

$$\begin{aligned} \Omega(x) &= S(x)\Lambda(x) \mod x^{2t} \\ &= \sum_{j=1}^{v} e_j\beta_j \prod_{i=1,i\neq j}^{v}(1-\beta_i x). \end{aligned} \tag{2.41}$$

The error values can be calculated from $\Omega(x)$ based on Forney's algorithm [46]:

$$e_j = \frac{-\Omega(\beta_j^{-1})}{\Lambda'(\beta_j^{-1})}. \tag{2.42}$$

Notice that $\Lambda'(x)$ indicates the derivative of $\Lambda(x)$ and therefore

$$\Lambda'(x) = \frac{d\Lambda(x)}{dx} = \sum_{\jmath=1}^{v} -\beta_\jmath \prod_{i=1,i\neq\kappa}^{v}(1-\beta_i x). \tag{2.43}$$

From (2.41), the degree of $\Omega(x)$ is $v-1$, implying that, in $S(x)\Lambda(x)$, the coefficients of the terms with powers large than $v-1$ must be zero. Consequently, the following Newton's identity can be derived [39].

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_{v+1} \\ S_2 & S_3 & \cdots & S_{v+2} \\ S_3 & S_4 & \cdots & S_{v+3} \\ \vdots & \vdots & & \vdots \\ S_{2t-v} & S_{2t-v+1} & \cdots & S_{2t} \end{bmatrix} \begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \vdots \\ \Lambda_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{2.44}$$

As we have described in Section 2.2, it is inefficient to directly solve (2.44) for large v. The Berlekamp-Massey and Euclidean algorithms can be employed for solving the the key equation (2.41). In the error-only decoding, the Euclidean algorithm described in Section 2.2 computes $\Lambda(x)$ and $\Omega(x)$ simultaneously; therefore, it can be applied for RS code without modification. However, the Berlekamp-Massey algorithm described in Section 2.2 only calculates $\Lambda(x)$. The evaluation of $\Omega(x)$ can be derived according to the key equation (2.44) after $\Lambda(x)$ is obtained from Berlekamp-Massey algorithm:

$$\begin{aligned}\Omega(x) &= \sum_{i=0}^{v-1} \Omega_i x^i \\ &= \sum_{i=0}^{v-1} \sum_{j=0}^{i} \Lambda_j S_{i+1-j} x^i \end{aligned} \tag{2.45}$$

Nevertheless, the Berlekamp-Massey algorithm can be modified to evaluate $\Lambda(x)$ and $\Omega(x)$ simultaneously. The $\Omega(x)$ is iteratively updated with similar operations as $\Lambda(x)$ does. The detailed procedure is given as follows.

- Initial conditions:

$$\begin{aligned}&\Lambda^{(0)}(x) = 1, \quad \Omega^{(0)}(x) = \tfrac{1}{x}, \\ &\tau^{(0)}(x) = 1, \quad \gamma^{(0)}(x) = 1, \\ &\Delta_0 = S_1, \qquad \delta = 1, \qquad L_0 = 0.\end{aligned}$$

- Iteration from $i = 1$ to $2t$:

$$\Lambda^{(i)}(x) = \Lambda^{(i-1)}(x) + \frac{\Delta_{i-1}}{\delta}\tau^{(i-1)}(x) \cdot x. \tag{2.46}$$

$$\Omega^{(i)}(x) = \Omega^{(i-1)}(x) + \frac{\Delta_{i-1}}{\delta}\gamma^{(i-1)}(x) \cdot x. \tag{2.47}$$

$$\Delta_i = \sum_{j=0}^{t-1} \Lambda_j^{(i)} S_{i+1-j}. \tag{2.48}$$

If $\Delta_{i-1} = 0$ or $L_{i-1} \geq i - L_{i-1}$,

$$\begin{aligned} \tau^{(i)}(x) &= \tau^{(i-1)}(x) \cdot x, \\ \gamma^{(i)}(x) &= \gamma^{(i-1)}(x) \cdot x, \\ L_i &= L_{i-1}. \end{aligned}$$

Otherwise,

$$\begin{aligned} \tau^{(i)}(x) &= \Lambda^{(i-1)}(x), \\ \gamma^{(i)}(x) &= \Omega^{(i-1)}(x), \\ L_i &= i - L_{i-1}, \qquad \delta = \Delta_{i-1}. \end{aligned}$$

After $2t$ iterations, the error location polynomial is obtained from $\Lambda(x) = \Lambda^{2t}(x)$ and the error evaluator polynomial is obtained from $\Omega(x) = \Omega^{2t}(x)$.

In summary, the error only decoding procedure of RS codes contains four major steps: *syndrome calculator*, *key equation solver*, *Chien search*, and *error value evaluator* as shown in Fig. 2.2. The received polynomial $R(x)$ is fed into syndrome calculator to generate syndrome polynomial $S(x)$ according to (2.36). The key equation solver calculates the error location polynomial $\Lambda(x)$ and error evaluator polynomial $\Omega(x)$ according to the key equation with either Berlekamp-Massey algorithm or Euclidean algorithm. After the key equation solver, Chien search is applied to find the roots of $\Lambda(x)$ while error value evaluator is utilized to calculate error values based on Forney's algorithm. Finally, the estimated codeword polynomial $\hat{C}(x)$ is obtained by adding those error values at error locations in $R(x)$.

As consider to the error and erasure decoding, we will find that it is quite similar to the above mentioned error only decoding procedure. If the received data $R(x)$ contains v errors

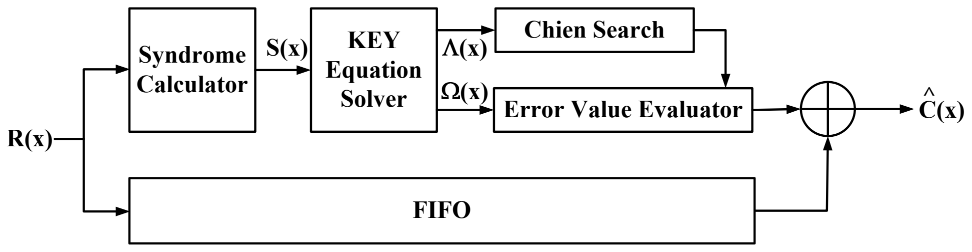


Figure 2.2: Error only RS decoding process

and u erasures with $2v+u \leq 2t$, the erasure symbols are replaced with arbitrary values, such as zeros, for the syndrome calculation in (2.36). A erasure location polynomial is defined as

$$\theta(x) = (1-\alpha^{l_1}x)(1-\alpha^{l_2}x)\cdots(1-\alpha^{l_u}x), \tag{2.49}$$

where $l_1, l_2, \ldots, l_u$ are erasure positions. The key equation should be modified by

$$S(x)\theta(x)\Lambda(x) = \Omega(x) \mod x^{2t}. \tag{2.50}$$

The errata location polynomial $\Phi(x) = \theta(x)\Lambda(x)$ identifies both error and erasure locations. The known polynomials $S(x)$ and $\theta(x)(x)$ can be combined as Forney syndrome polynomial [46]:

$$\begin{aligned} T(x) &\triangleq S(x)\theta(x) \mod x^{2t} \\ &= T_1 + T_2x + T_3x^2 + \cdots + T_{2t}x^{2t-1}. \end{aligned} \tag{2.51}$$

Consequently, the key equation (2.50) is written as:

$$T(x)\Lambda(x) = \Omega(x) \mod x^{2t}. \tag{2.52}$$

It is can be proved that the maximum degree of $\Omega(x)$ is $(v+u-1)$ while the degree of $\Lambda(x)\theta(x)$ is $(v+u)$. As a result, the similar equation as (2.44) is derived:

$$\begin{bmatrix} T_{u+1} & T_{u+2} & \cdots & T_{u+v+1} \\ T_{u+2} & T_{u+3} & \cdots & T_{u+v+2} \\ \vdots & \vdots & & \vdots \\ T_{2t-v} & T_{2t-v+1} & \cdots & T_{2t} \end{bmatrix} \begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \vdots \\ \Lambda_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.53}$$

The error location polynomial $\Lambda(x)$ can be iteratively computed with Berlekamp-Massey algorithm as follows [47]:

- Initial conditions:

$$\Lambda^{(u)}(x) = 1, \quad \Omega^{(u)}(x) = \tfrac{1}{x},$$
$$\tau^{(u)}(x) = 1, \quad \gamma^{(u)}(x) = 1,$$
$$\Delta_u = T_{u+1}, \quad \delta = 1, \qquad L_u = 0.$$

- Iteration from $i = (u+1)$ to $2t$:

$$\Lambda^{(i)}(x) = \Lambda^{(i-1)}(x) + \frac{\Delta_{i-1}}{\delta}\tau^{(i-1)}(x) \cdot x \tag{2.54}$$
$$\Omega^{(i)}(x) = \Omega^{(i-1)}(x) + \frac{\Delta_{i-1}}{\delta}\gamma^{(i-1)}(x) \cdot x \tag{2.55}$$
$$\Delta_i = \sum_{j=0}^{t-1} \Lambda_j^{(i)} T_{i+1-j} \tag{2.56}$$

If $\Delta_{i-1} = 0$ or $L_{i-1} \geq i - u - L_{i-1}$,

$$\tau^{(i)}(x) = \tau^{(i-1)}(x) \cdot x,$$
$$\gamma^{(i)}(x) = \gamma^{(i-1)}(x) \cdot x,$$
$$L_i = L_{i-1}.$$

Otherwise,

$$\tau^{(i)}(x) = \Lambda^{(i-1)}(x),$$
$$\gamma^{(i)}(x) = \Omega^{(i-1)}(x),$$
$$L_i = i - u - L_{i-1}, \quad \delta = \Delta_{i-1}$$

After $2t$ iterations, the error location polynomial is obtained from $\Lambda(x) = \Lambda^{2t}(x)$ and the error evaluator polynomial is obtained from $\Omega(x) = \Omega^{2t}(x)$.

On the other hand, Euclidean algorithm is also applicable for solving (2.52).

- Initial conditions

$$\Omega^{(-1)} = x^{2t}, \quad \Omega^{(0)} = T(x)$$
$$\Lambda^{(-1)}(x) = 0, \quad \Lambda^{(0)}(x) = 1$$

- Iterations from $i = 1$:

$$\Omega^{(i)}(x) = Q^{(i)}(x)\Omega^{(i-1)}(x) + \Omega^{(i-2)}(x) \tag{2.57}$$

$$\Lambda^{(i)}(x) = Q^{(i)}(x)\Lambda^{(i-1)}(x) + \Lambda^{(i-2)}(x) \tag{2.58}$$

- Iteration terminated

$$deg(\Lambda(x)) + u > deg(\Omega(x))$$

After the termination of Euclidean algorithm, $\Lambda(x) = \Lambda^{(i)}(x)$ and $\Omega(x) = \Omega^{(i)}(x)$, where $\Lambda(x))$ has the degree v and $\Omega(x)$ has the degree at most $u + v - 1$.

After the error locations are found with Chien search, the error values can be evaluated according to Forney's algorithm:

$$e_\kappa = \frac{-\Omega(\beta_\kappa^{-1})}{\Phi'(\beta_\kappa^{-1})} = \frac{-\Omega(\alpha^{-j_\kappa})}{\Phi'(\alpha^{-j_\kappa})}, \quad \text{for } 1 \le \kappa \le v, \tag{2.59}$$

and the erasure values are

$$\hat{e}_\rho = \frac{-\Omega(\alpha^{-l_\rho})}{\Phi'(\alpha^{-l_\rho})}, \quad \text{for } 1 \le \rho \le u, \tag{2.60}$$

where $\Phi'(x)$ indicates the derivative of $\Phi(x)$.

In summary, the major difference in error and erasure decoding algorithm is the Forney syndrome polynomial $T(x)$ in (2.51) and key equation (2.52). In addition, both Berlekamp-Massey and Euclidean algorithms are modified for the known erasure location polynomial $\theta(x)$ and $T(x)$. There are five major steps in the error and erasure decoding procedure of RS codes: *syndrome calculator*, *Forney syndrome calculator*, *key equation solver*, *Chien search* and *error value evaluator* as shown in Fig. 2.3. The syndrome calculator generates $2t$ syndromes from the received vector $R(x)$. If there are erasure symbols, $T(x)$ is evaluated by the Forney syndrome calculator. Then the key equation solver delivers $\Lambda(x)$ and $\Omega(x)$ with either Berlekamp-Massey or Euclidean algorithm. According to $S(x)$ or $T(x)$, $\Lambda(x)$ and

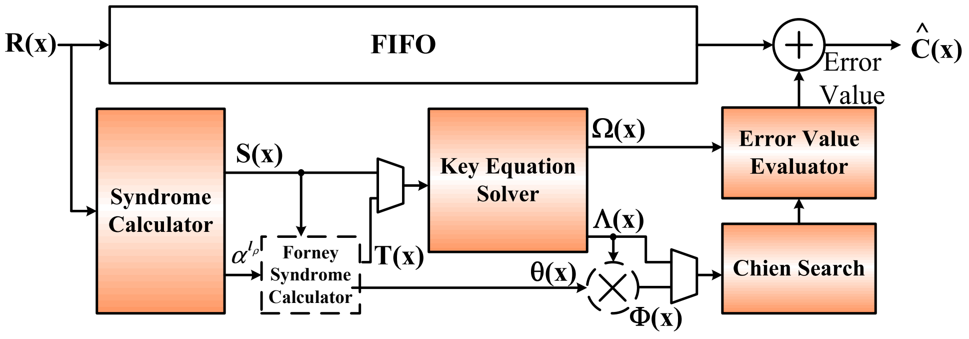


Figure 2.3: Error and erasure RS decoding process

$\Omega(x)$ are generated from the key equation solver with either Berlekamp-Massey or Euclidean algorithm. Additionally, the errata location polynomial $\Phi(x)$ is required when $\theta(x)$ exists. After Chien search finds the error locations, error value evaluator uses (2.42) for computing error values or uses (2.59) and (2.60) for computing both error and erasure values Finally, the estimated codeword polynomial $\hat{c}(x)$ is obtained by correcting the received data $R(x)$, which is stored in the first-in and first-out (FIFO).

2.4 Overview of Soft Decoding Algorithms

Soft decoding algorithms of error control codes exploit the reliability information from the channel to correct an error number larger than half the minimum distance for a code, As a general rule of thumb, soft decoding can provide as much as 3 dB of gain over hard decoding. Due to the higher transmission data rate, the recent applications require better decoding methods to resist growing uncertainty of received signals; therefore, soft decoding algorithms have aroused many research interests.

The soft decoding algorithms of BCH and RS codes discussed in this section are classified into soft-input hard-output decoding algorithms. That is, the soft input values are utilized to process the algebraic decoding and the decoded values are provided without any reliability information. On the other hand, the soft-input soft-output decoding algorithms apply probability propagation methods with soft input values and provide decoded values accompanied by a probability distribution for the decoded bits. In general, the soft decoding algorithms are much harder to implement than hard decoding algorithms and require more hardware complexity. For the practice issue, this section only discusses several soft-input hard-output algorithms based on list decoding approach, where a list decoder generally generates several candidate codewords and selects one from the list as the decoded codeword.

2.4.1 General Minimum Distance Algorithm

The generalized minimum distance (GMD) decoding algorithm was devised by Forney in 1966 [25]. Recall that a linear code with minimal distance d_{min} can correct up t errors and ρ erasures with $2t + \rho \leq d_{min} - 1$ based on the error and erasure decoding algorithm. The GMD algorithm exploits the soft input values to determine at most $d_{min} - 1$ least reliable positions (LRPs) and considers all possible case of up to $\rho \leq d_{min} - 1$ erasures in these positions. That is, this decoding algorithm will generate a list of at most $\lfloor \frac{d_{\min}+1}{2} \rfloor$ candidate codewords with error and erasure decoding algorithm. Then, a most likely codeword will be selected from the list according to the Euclidean distance calculations between the received input values and each candidate codeword. The GMD algorithm operates as follows:

GMD Algorithm

- Initiation:

  A hard decision received vector V is formed from the soft received sequence R and the reliability of each symbol in V is assigned.

- LRPs determination:

  The $d_{min} - 1$ least reliable positions are determined according to the reliabilities.

- Candidate codewords generation:

  A list of at most $\lfloor \frac{d_{\min}+1}{2} \rfloor$ sequence is generated by modified V with LRPs.

  If d_{min} is even:

  ```
      for $j = 1$ to $d_{min} - 1$ by 2:
  ```

```
        A modified vector $\hat{V}$ is formed by erasing $j$ LRPs in $V$.
else if $d_{min}$ is odd:
    for $j = 0$ to $d_{min} - 1$ by 2:
        A modified vector $\hat{V}$ is formed by erasing $j$ LRPs in $V$.
```

Then a list of candidate codewords is generated by decoding each sequence with the error and erasure decoding algorithm.

- Best codeword selection:

  The Euclidean distance of the soft received sequence R and each candidate codeword is calculated. Then the decoded codeword, which is with the smallest distance, is the selected from the list.

Notice that, since hard decision values are actually used in the error and erasure decoding algorithm, the algebraic decoding algorithm presented in Section 2.3 can be applied for GMD algorithm. In most case, fewer than $\lfloor \frac{d_{\min}+1}{2} \rfloor$ candidate codewords are generated because the error and erasure decoding may fail to generate a candidate codeword in some erasure number case.

To sum up, there are four major steps in the GMD decoding procedure: *LRPs evaluator*, *candidate sequence generator*, *error and erasure hard decoder*, and *decision making unit* as shown in Fig. 2.4. The soft values are fed into LRPs evaluator to determine $d_{min} - 1$ LRPs.

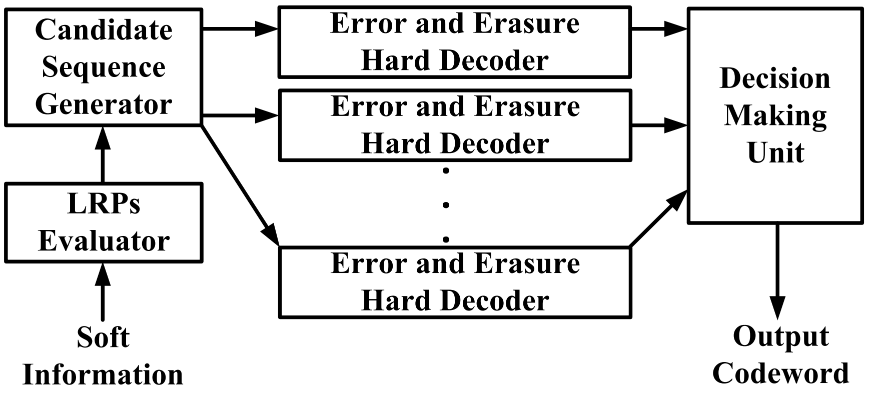

Candidate Sequence Generator
Error and Erasure Hard Decoder
Error and Erasure Hard Decoder
Error and Erasure Hard Decoder
Decision Making Unit
LRPs Evaluator
Soft Information
Output Codeword

Figure 2.4: GMD decoding process

With these positions, $\left\lfloor \frac{d_{\min}+1}{2} \right\rfloor$ candidate sequences are formed by the candidate sequence generator. Furthermore, totally $\left\lfloor \frac{d_{\min}+1}{2} \right\rfloor$ error and erasure hard decoders are applied to solve each candidate sequence. Finally, the decision making unit selects a best codeword from the outputs of all the error and erasure hard decoders.

2.4.2 Chase Algorithm

Based on the concept of GMD algorithm, Chase devised three algorithms, referred as Chase-1, Chase-2, and Chase-3, in 1973 [26]. Unlike GMD algorithm utilizes error and erasure decoding algorithm, Chase algorithm uses error only decoding algorithm to generate candidate codewords. In Chase-1 algorithm, all the possible combinations among $d_{min} - 1$ LRPs are complemented (i.e., changing a 1 to 0, or a 0 to 1), resulting in at most $C^{N}_{\left\lfloor \frac{d_{min}}{2} \right\rfloor}$ candidate codewords. Chase-3 algorithm, which is very similar to GMD algorithm, replaces the erasure position with the complementation operation at LRPs. A list of at most $d_{min} - 1$ candidate codewords will be generated. As considered both error correcting ability and computation complexity, Chase-2 algorithm is the most popular one.

In Chase-2 algorithm, there are four major steps in the decoding procedure: *LRPs evaluator*, *candidate sequence generator*, *error only hard decoder*, and *decision making unit* as shown in Fig. 2.5. The soft values are fed into LRPs evaluator to determine at most $\frac{d_{min}}{2}$ LRPs are determined according to the reliabilities. With these positions, a list of $2^{\frac{d_{min}}{2}}$ candidate sequences are formed by the candidate sequence generator. Furthermore, totally

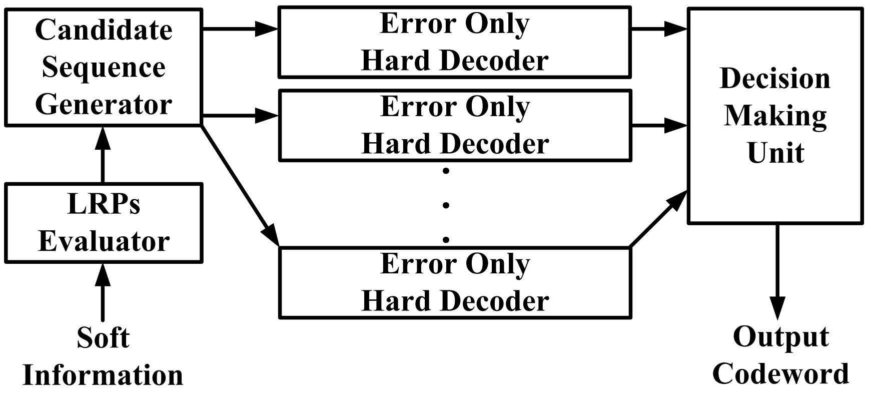


Figure 2.5: Chase decoding process

$\lfloor \frac{d_{min}+1}{2} \rfloor$ error only hard decoders solve each candidate sequence and generate a list of at most $2^{\frac{d_{min}}{2}}$ candidate codewords. Finally, a most likely codeword will be selected from the list by the decision making unit according to the Euclidean distance calculations between the received input values and each candidate codeword. The details of Chase-2 decoding process are listed as follows:

Chase Algorithm

- Initiation:

  A hard decision received vector V is formed from the soft received sequence R and the reliability of each symbol in V is assigned.

- LRPs determination:

  The $\frac{d_{min}}{2}$ least reliable positions are determined according to the reliabilities.

- Candidate codewords generation:

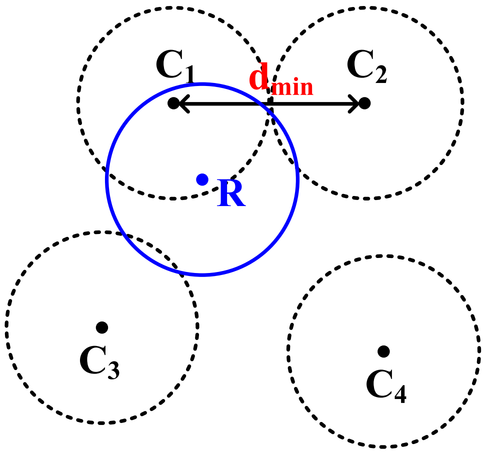
  A list of at most $2^{\frac{d_{min}}{2}}$ sequence is generated by modified V with LRPs.

  ```
      for $j = 1$ to $2^{\frac{d_{min}}{2}}$ by 1:
          A modified vector $\hat{V}$ is formed by complementing one combination of LRPs in $V$.
  ```

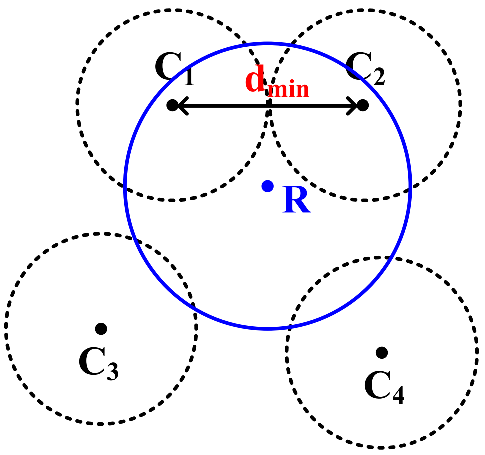
  Then a list of candidate codewords is generated by decoding each sequence with the error only decoding algorithm.

- Best codeword selection:

  The Euclidean distance of the soft received sequence R and each candidate codeword is calculated. Then the decoded codeword, which is with the smallest distance, is the selected from the list.



(a) Bounded distance decoding



(b) List decoding

Figure 2.6: The comparison of bounded distance decoding and the list decoding.

2.4.3 Guruswami-Sudan Algorithm

An alternative decoding algorithm of RS codes is Guruswami-Sudan (GS) algorithm, which was introduced by Madhu Sudan and Venkatesan Guruswami in 1999 [29]. GS algorithm is the first interpolation-based list decoding algorithm. It can stretch the error correction ability and find out all the probable codewords within its extended decoding sphere. Fig 2.6 shows the comparison of original bounded distance (BD) decoding and the list decoding with sphere ball representation. The BD decoding only returns at most one codeword will be in Hamming sphere $\frac{d_{min}-1}{2}$ with received data R as original. On the other hand, the list decoding returns all codewords within the extended Hamming distance, which is larger than $\frac{d_{min}-1}{2}$. Hence, the list decoding is able to solve some situations that would be uncorrectable for BD decoding.

The encoding of GS algorithm is based on the first construction (2.33) which has been discussed in Section 2.3. An evaluation mapping method constructs the codeword $C = (c_o, c_1, \ldots, c_{N-1}) = (u(1), u(\alpha), \ldots, u(\alpha^{N-1})$ from the message polynomial $u(x) = u_0 + u_1x + u_2x^2 + \cdots + u_{K-1}x^{K-1}$. The decoding of GS algorithm consists two major steps: *interpolation* and *factorization* as shown in Fig. 2.3. Since a set of data points (x_i, c_i), $i = 0, 1, \ldots, N-1$, are generated in the encoding procedure with $x_i = \alpha^i$, the codeword polynomial can be viewed as the interpolation result from each point of the set. However, the received data

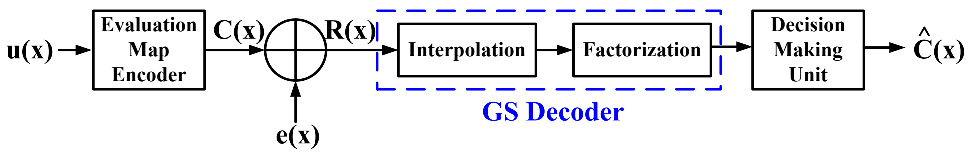


Figure 2.7: Guruswami-Sudan decoding process

are suffered from noise. To recover the points in error, polynomials that can match enough data points are searched based on interpolation procedure. The interpolating polynomial is constructed as a non-zero bivariate polynomials Q(x,y) by interpolating every received point (x_i, y_i) for $i = 0 \sim N-1$. In addition to simple interpolating, an interpolation parameter, multiplicity θ, is introduced to define the order of the interpolation at each point. The constructed polynomial is of the form

$$Q(x,y) = \sum_{j=0}^{C} a_j \phi_j(x,y) \tag{2.61}$$

$$C = n \cdot \frac{(\theta+1)!}{2!\,(\theta-1)!}, \tag{2.62}$$

where $\phi_j(x,y)$ is of the form $x^p y^q$. Notice that larger multiplicity θ increases the error correction ability of GS decoder but also raises the computation complexity of GS decoder. The time complexity of GS algorithm is $O(N^2\theta^4)$. However, the error correcting ability of GS algorithm has a upper bound. For an $(N, K; t)$ RS code applying GS algorithm, the maximum decoding sphere limit is $t_\infty = N - 1 - \left\lfloor \sqrt{N(K-1)} \right\rfloor$.

For efficient interpolating, Feng-Tzeng algorithm [48] and Koetter algorithm [49] were presented in 1991 and 2002 respectively. In general, Koetter algorithm is more popular and the detailed of Koetter algorithm is described as follows:

Koetter's Interpolation Algorithm

- Input:

  Points: (x_i, y_i), $i = 0, 1 \ldots, N-1$;

  Interpolation order θ_i;

  L = maximum list number.

- Initial conditions:

  $g_j = y^j$ for $j = 0 \sim L$.

- Iteration from $i = 0$ to $N - 1$:

```
for $(r, s) = (0, 0)$ to $(\theta_{i-1}, 0)$ by $(\theta_{i-1}, 1)$ lex order
  for $j = 0$ to $L$
    $\Delta_j = D_{r,s} g_j(x_i, y_i)$
  end (for $j$)
  $\mathbf{J} = \{j : \Delta_j \neq 0\}$
  if $(\mathbf{J} \neq NULL)$
    $j^* = argmin\{g_j : j \in \mathbf{J}\}$
    $f = g_{j^*}$
    $\Delta = \Delta_{j^*}$
    for $(j \in \mathbf{J})$
      if $(j \neq j^*)$
        $g_j = \Delta g_j + \Delta_j f$
      else if $(j = j^*)$
        $g_j = (x - x_i) f$
      end (if)
    end (for $j$)
  end (for $\mathbf{J}$)
end (for $(r, s)$)
```

- Output:

  $Q(x, y) = min_j\{g_j(x, y)\}$

After the interpolation procedure, $Q(x, y)$ will contain some factor polynomials of the form $y - p(x)$, where $p(x)$ is a polynomial of degree $K - 1$ or less, and is one of the decoded result of the candidates.

$$L = \{p_1(x), p_2(x), ..., p_\ell(x)\}. \tag{2.63}$$

Factorization is exploited to decompose these factors from $Q(x, y)$ to get the candidate codeword $p(x)$. Since we only interest in the factor of the form $y - p(x)$, it is not necessary to factorize all the factor of $Q(x, y)$. For the efficient computations, Roth and Ruckenstein algorithm [50] which was developed in 2000 is usually performed. A recursive computations are presented to construct a depth-first tree structure. The details of Roth and Ruckenstein algorithm is stated as follows:

Roth-Ruckenstein Algorithm

```
• Input:
  $Q(x, y)$;

• Initial conditions:
  $p(x) = 0$, $u = deg(p) = -1$,
  $D$ = maximum degree of $p(x)$, $v = 0$. Call Rothrucktree $(Q(x, y), u, p)$

• Function Rothrucktree $(Q(x, y), u, p)$:
  $v = v + 1$
  if $(Q(x, 0) = 0)$
    add $p(x)$ into output list
  else if $(u < D)$
    $R$ = list of roots of $Q(0, y)$
    for each $\alpha \in R$
      $Q_{new}(x, y) = Q(x, xy + \alpha)$
      $p_{u+1} = \alpha$
      Call rothrucktree $(Q_{new}(x, y), u + 1, p)$
    end (for)
  else
    output list = NULL
  end (if)
```

- Output:

  List of polynomials $p(x)$ of degree $\leq D$ such that $(y - p(x))|Q(x,y)$.

After the decoded list is generated, normally the result with the minimum Euclidean distance is chosen as the decoded codeword.

2.4.4 Koetter-Vardy Algorithm

Based on the GS algorithm, Kotter and Vardy proposed a soft decoding method, referred to Koetter-Vardy (KV) algorithm, that allowing each point on the interpolated curve to have its own multiplicity [28]. Unequal multiplicities are assign to points according to their relatively reliabilities. Fig. 2.8 shows the decoding process of KV algorithm.

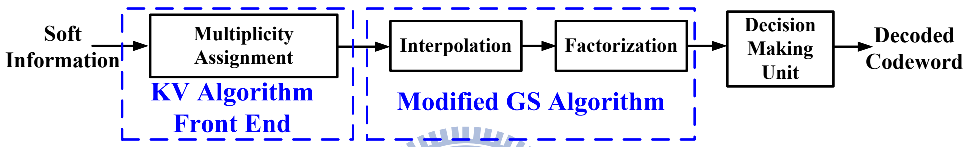


Figure 2.8: Kotter-Vardy decoding process

The major difference between KV and GS algorithms is that KV algorithm provides a multiplicity assignment to offer every point its own multiplicity according to the reliabilities of all possible transmitted/received symbol pairs. For an $(N, K; t)$ RS code over $GF(2^m)$, a $(2^m - 1) \times N$ *reliability matrix* Π is constructed for representing the *a posteriori probability* of each transmitted/received symbol pair. Then an multiplicity assignment is proposed to calculate Θ from Π subject to complexity constraints "s" [28]:

Multiplicity Assignment Algorithm

- Input:

  Reliability matrix Π ;

- Initial conditions:

  Choose a desired value for $s = \sum_{i=0}^{2^m-1} \sum_{j=0}^{N-1} \theta_{i,j}$.

  $\Pi^* = \Pi$, $\Theta = 0$.

- While $s > 0$:

  Find the position (i, j) of the largest entry $\pi_{i,j}$ in Π^*.

  $\pi^*_{i,j} = \frac{\pi_{i,j}}{\theta_{i,j}+2}$

  $\theta_{i,j} = \theta_{i,j} + 1$

  $s = s - 1$

- Output:

  Multiplicities Θ

Notice that, if s is large enough, it is possible to have more than N points having their own multiplicities. The interpolation procedure of KV algorithm deals with equal to or more than N points with their relatively reliabilities while that of GS algorithm handles exact N points with identical reliability. For example, for a (7, 5; 1) RS code over $GF(2^3)$, if the received reliability matrix is

$$\Pi = \begin{bmatrix} 0.959796 & 0.214170 & 0.005453 & 0.461070 & 0.001125 & 0.000505 & 0.691729 \\ 0.001749 & 0.005760 & 0.000000 & 0.525038 & 0.897551 & 0.025948 & 0.000209 \\ 0.028559 & 0.005205 & 0.000148 & 0.003293 & 0.000126 & 0.018571 & 0.020798 \\ 0.000052 & 0.000140 & 0.000000 & 0.003750 & 0.100855 & 0.954880 & 0.000006 \\ 0.009543 & 0.736533 & 0.968097 & 0.003180 & 0.000000 & 0.000000 & 0.278789 \\ 0.000017 & 0.019810 & 0.000006 & 0.003621 & 0.000307 & 0.000003 & 0.000084 \\ 0.000284 & 0.017900 & 0.026295 & 0.000023 & 0.000000 & 0.000002 & 0.008382 \\ 0.000001 & 0.000481 & 0.000000 & 0.000026 & 0.000035 & 0.000092 & 0.000003 \end{bmatrix},$$

after running multiplicity assignment with constraint $s = 12$, the multiplicity matrix becomes

$$\Theta = \begin{bmatrix} 2 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Then the interpolation procedure will run with the following input:

x	y	*Multiplicity*
1	0	2
α	α^3	1
α^2	α^3	2
α^3	0	1
α^3	1	1
α^4	1	2
α^5	α^2	2
α^6	0	1

In the KV algorithm, the interpolation step is the most time and hardware consuming component and the complexity reduction of the interpolation step is essential for the real-time applications. The re-encoding technique as shown in Fig. 2.9 is applied to reduce the number of interpolation points. After transmitting the codeword C through a noisy channel, the hard decision vector r can be extracted from the reliability matrix as $r = C + e$, where e is the error vector. The received symbols in r can be separated into two sets: $N - K$ symbols in U ("unreliable") and K symbols in R ("reliable"). In addition, the set of positions of the

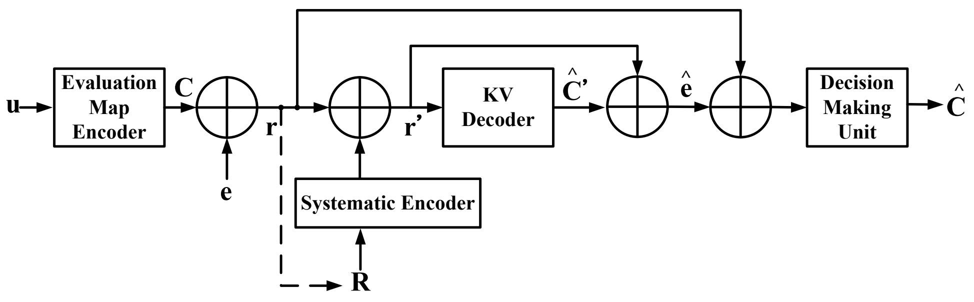


Figure 2.9: Kotter-Vardy decoding process with re-encoding technique

symbols in R is the set R_K. A *re-encoded codeword* ψ can be constructed by a systematic encoder with K most reliability symbol in R. Notice that, the systematic decoder is designed as an *erasure only decoder* for re-encoding with K arbitrary positions. The difference between r and ψ is

$$\begin{aligned} r' &= r - \psi \\ &= (C + e) - \psi \\ &= (C - \psi) + e \\ &= C' + e, \end{aligned} \tag{2.64}$$

indicating that r' can be viewed as a codeword suffered from the same error vector as r. Since the systematic encoder is applied for generating ψ from r, their difference r' will have K zeros symbols in the R_K locations, meaning that there are K interpolation points with a zero y-component:

$$V = \{(\alpha^i, 0)\}, \qquad i \in R_K. \tag{2.65}$$

Therefore, an interpolation polynomial for these K points in V with the same multiplicity θ is $v(x)^\theta$, where $v(x)$ can be calculated through a simple univariate interpolation:

$$v(x) = \prod_{i \in R_K} (x - \alpha^i). \tag{2.66}$$

Hence, Koetter's interpolation algorithm can be simplified for interpolating the remainder points with the initial polynomial set:

$$G = \{v(x)^{\theta}, v(x)^{\theta-1}y, \ldots, v(x)^{\theta-d_y}y^{d_y}\}, \tag{2.67}$$

where d_y is the maximum degree of the bivariate polynomial $Q(x, y)$. For the high code rate codes, most of the points are zero points after the re-encoding procedure and the calculation of Koetter's interpolation can be significantly reduced due to few number of the remainder points.

Another technique for improving the complexity is the coordinate transformation method, provided by Koetter and Vardy in 2003 [51,52]. If the re-encoding is used and the points in R have the maximum possible multiplicity $\theta = d_y$, the bivariate polynomial $Q(x, y)$ can be reduced to [49]

$$\begin{aligned} Q(x,y) &= \sum_{j=0}^{d_y} \omega_j(x) v(x)^{\theta-j} y^j \\ &= v(x)^{\theta} \sum_{j=0}^{d_y} \omega_j(x) \left(\frac{y}{v(x)}\right)^j, \end{aligned} \tag{2.68}$$

which indicates that a large amount of memories is required for storing the common term $v(x)$. Since we only interest in the factor of the form $y - p(x)$ from Q(x,y), the term $v(x)^{\theta}$ can be removed and a *reduced interpolation polynomial* can be defined as:

$$\widetilde{Q}(x, \widetilde{y}) = \sum_{j=0}^{d_y} \omega_j(x) \widetilde{y}^j, \tag{2.69}$$

where $\widetilde{y} = \frac{y}{v(x)}$. Hence, Koetter's interpolation algorithm can be further simplified for interpolating the remainder points with the initial polynomial set:

$$\widetilde{G} = \{1, \widetilde{y}, \ldots, \widetilde{y}^{d_y}\}. \tag{2.70}$$

With re-encoding method, if the decoding is successful, a message polynomial $u'(x)$ corresponding to the estimated transformed codeword C' will be a linear y-root of $Q(x,y)$, implying $\frac{u'(x)}{v(x)}$ is linear $\widetilde{y}$-root of $\widetilde{Q}(x,\widetilde{y})$:

$$\widetilde{Q}(x,\widetilde{y}) = \left(\widetilde{y} - \frac{u'(x)}{v(x)}\right) A(x,\widetilde{y}). \tag{2.71}$$

After applying Roth-Ruckenstein algorithm, we can obtain a *reduced factorization polynomial*:

$$s(x) = \frac{u'(x)}{v(x)}. \tag{2.72}$$

Obviously, the degree of $s(x)$ is much smaller than K, leading to the significantly computation improvement for Roth-Ruckenstein algorithm.

Although Roth-Ruckenstein algorithm is simplified, a complex procedure is required to construct $u'(x)$ from $s(x) \times v(x)$. For the further improvement, let's recall that the transformed received hard-decision word is $r' = C' + e$ and has K zeros in R_K. Therefore, in the K most reliable positions, the summation of encoded symbol $u'(\alpha^i$ and error value e_i is zero, i.e.,

$$u'(\alpha^i) = e_i, \qquad i \in R_K. \tag{2.73}$$

Notice that $(x - \alpha^i)$ is a root $u'(x)$ in all the error-free locations in R_K. Hence, $s(x)$ can be rewritten as:

$$\begin{aligned} s(x) &= \frac{u'(x)}{v(x)} \\ &= \frac{\prod\limits_{i\in R_k, e_i=0} (x-\alpha^i)\Omega(x)}{\prod\limits_{i\in R_k} (x-\alpha^i)} \\ &= \frac{\Omega(x)}{\prod\limits_{i\in R_k, e_i\neq 0} (x-\alpha^i)} && (2.74) \\ &= \frac{\Omega(x)}{\Lambda(x)}. && (2.75) \end{aligned}$$

The $\Lambda(x)$ can be viewed as an error location polynomial for the positions in R_K while $\Omega(x)$ can be viewed as a corresponding error evaluator polynomial. With the *syndrome polynomial*

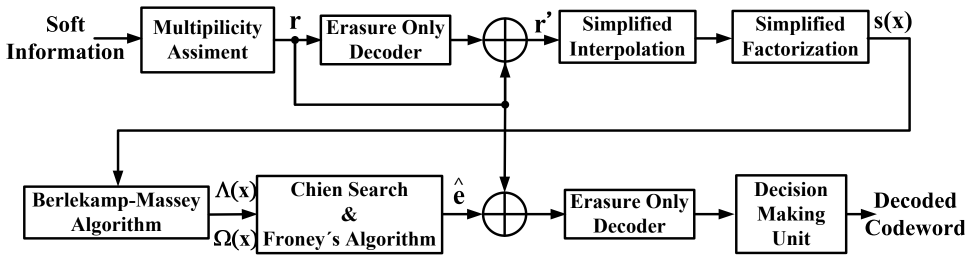


Figure 2.10: Simplified Kotter-Vardy decoding process

$s(x)$, the Berlekamp-Massey algorithm can be applied to solve $\Lambda(x)$ and $\Omega(x)$. Afterward, the roots of $\Lambda(x)$ are found by Chien search and the correspond error values can be evaluated by the Forney's algorithm:

$$\begin{aligned} e_i &= u'(\alpha^i) \\ &= \frac{\Omega(\alpha^i)v(\alpha^i)}{\Lambda(\alpha^i)} \\ &= \frac{\Omega(\alpha^i)v^{(1)}(\alpha^i)}{\Lambda^{(1)}(\alpha^i)}, \end{aligned} \tag{2.76}$$

where $v^{(1)}(x)$ and $\Lambda^{(1)}(x)$ are the formal derivatives of $v(x)$ and $\Lambda(x)$ respectively. Notice that, the estimated error vector $\hat{e}$ is found without subtract off r'. However, $\hat{e}$ only represents for the errors in R. An erasure only decoder is required to find the errors in U. Fig 2.10 shows the block diagram of the fully simplified KV algorithm.

2.4.5 Low Complexity Chase Algorithm

In Subsection 2.4.4, we introduce several techniques for reducing the hardware complexity of KV algorithm; however, the complexity of interpolation with high multiplicity θ is still too high for practical implementation. In 2006, a Chase-type decoding method exploiting KV algorithm with multiplicity $\theta = 1$ is introduced and is referred to *low complexity Chase (LCC)* algorithm [53]. According to the received reliability matrix Π, the most possible and second possible symbol pairs of each received point can be defined as the hard-decision (x_i, y_i^{HD}) and secondary hard-decision (x_i, y_i^{2HD}) respectively for $i = 0 \sim N-1$. The probability γ_i

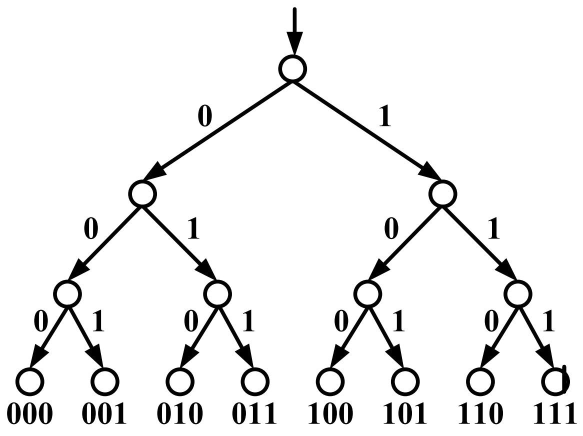


Figure 2.11: Binary tree interpolation procedure of the LCC decoding at $\eta = 3$

which gives a measure of the reliability of each point is given as

$$\gamma_i = \frac{p(c_i = y_i^{2HD}|\gamma_i)}{p(c_i = y_i^{HD}|\gamma_i)} \leq 1, \tag{2.77}$$

where the condition that the hard-decision is wrong and the secondary hard-decision is correct becomes more possible if γ_i is closer to 1.

In the LCC decoding, we determine totally η least reliability symbols which maximize γ_i for $i = 0 \sim N-1$ and form the unreliable position set: $\mathcal{I} = \{i_1, i_2, \ldots, i_\eta\}$. Then the N-point candidate sequence is defined as

$$(x_i, y_i) = \begin{cases} \{(x_i, y_i^{HD})\} & if \quad i \notin \mathcal{I} \\ \{(x_i, y_i^{HD}, (x_i, y_i^{2HD})\} & if \quad i \in \mathcal{I} \end{cases} \tag{2.78}$$

Since there are two possible points in each location at the unreliable position set $\mathcal{I}$, the total number of candidate sequences generated for the LCC decoding is 2^η. Notice that there are $N-\eta$ common points in these candidate sequences, implying that a common point interpolation can be processed for decoding these 2^η candidate sequences. For the remaining uncommon η points, a binary tree interpolation procedure can be applied after the common point interpolation for generating 2^η bivariate polynomials [53]. Fig. 2.11 is an example of the binary tree interpolation procedure of the LCC decoding at $\eta = 3$. In the tree, the '0' and '1' labels by the edges indicate (x_i, y_i^{HD}) and (x_i, y_i^{HD}) respectively. A breadth-first

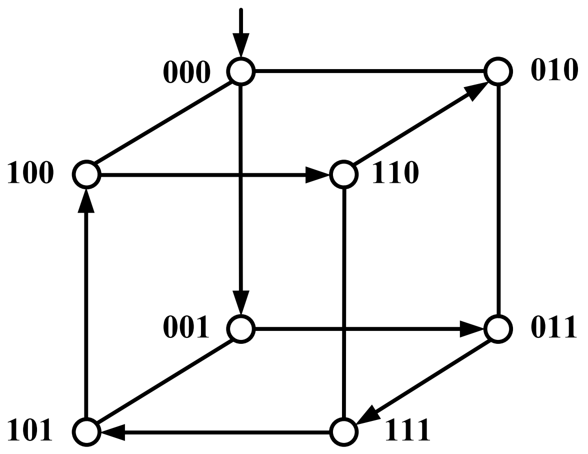


Figure 2.12: Hypercube interpolation procedure of the LCC decoding at $\eta = 3$

scheme scheme can be exploited to reuse $2^{\eta-1}$ intermediate interpolation results leading to large memory requirement.

In 2008, a η-dimension hypercube interpolation procedure for the LCC decoding was proposed [54]. Fig. 2.11 is an example of the hypercube interpolation procedure of the LCC decoding at $\eta = 3$. In hypercubes, there is only one bit different between the adjacent labels and we can travel through all the vertices with passing each vertex only once, where the path can be based on the flipping order of gray code. Therefore, the 2^{η} bivariate polynomials can be generated sequentially according to the travel path. After the first bivariate polynomial is constructed, the next bivariate polynomial can be derived from the former one by removing one point from the interpolation result and adding another point. Hence, only one intermediate interpolation result needs to be stored. The hypercube interpolation algorithm is described as follows:

Hypercube Interpolation Algorithm

- For each $(x_i, y_i^{HD}) \Rightarrow (x_i, y_i^{2HD})$ in the hypercube
- Input:

  (x_i, y_i^{2HD}),

  $g^0(x, y) = g_0^0(x) + g_1^0(x)y$,

  $g^1(x, y) = g_0^1(x) + g_1^1(x)y$.

- Backward interpolation:

```
Compute $g_0^0(x_i)$, $g_1^0(x_i)$, and $g^0(x_i, y_i^{2HD})$
Compute $g_0^1(x_i)$, $g_1^1(x_i)$, and $g^1(x_i, y_i^{2HD})$
$\delta = argmin_i(Wdeg(g^i(x,y))|g^i(x_i, y_i^{2HD}) \neq 0)$
$l = \{0,1\}\backslash\delta$
if $((g_0^l(x_i)! = 0)||(g_1^l(x_i)! = 0))$
    $g^l(x,y) = g^l(x_i, y_i^{2HD})g^\delta(x,y) + g^\delta(x_i, y_i^{2HD})g^l(x,y)$
divide $g^l(x,y)$ by $(x - x_i)$
```

- Forward interpolation:

```
Compute $g_0^0(x_i)$, $g_1^0(x_i)$, and $g^0(x_i, y_i^{2HD})$
Compute $g_0^1(x_i)$, $g_1^1(x_i)$, and $g^1(x_i, y_i^{2HD})$
$\delta = argmin_i(Wdeg(g^i(x,y))|g^i(x_i, y_i^{2HD}) \neq 0)$
$l = \{0,1\}\backslash\delta$
$g^l(x,y) = g^l(x_i, y_i^{2HD})g^\delta(x,y) + g^\delta(x_i, y_i^{2HD})g^l(x,y)$
$g^\delta(x,y) = g^\delta(x,y)(x - x_i)$
```

- Output:

```
$Q(x,y) = g^\kappa(x,y)$, where $\kappa = argmin_i(Wdeg(g^i(x,y))$
```

In 2009, Xinmiao Zhang indicated that the factorization and key equator solver procedure are not required for the LCC decoding [55]. According to the characteristic of the LCC decoding that the maximum y-degree of the interpolation output is one, $Q(x,y)$ can be expressed as

$$Q(x,y) = q_0(x) + q_1(x)y. \tag{2.79}$$

If the re-encoding and coordinate transformation techniques have been applied, the factorization procedure can generate the reduced factorization polynomial $s(x)$ of the form:

$$s(x) = \frac{q_0(x)}{q_1(x)}. \tag{2.80}$$

As compared to the (2.75), $q_0(x)$ and $q_1(x)$ can be derived as

$$\begin{cases} q_0(x)=h(x)\Omega(x) \\ q_1(x)=h(x)\Lambda(x) \end{cases}, \tag{2.81}$$

where $h(x)$ is the common factor between $q_0(x)$ and $q_1(x)$. From (2.75)) and (2.80), the estimated message polynomial becomes

$$u'(x) = s(x)v(x) = \frac{q_0(x)v(x)}{q_1(x)}. \tag{2.82}$$

In addition, it can be proved that $h(x)$ does not contain any factor $(x + \alpha^i)$ for $i \in R$ [34], implying that we can find error locations by finding the roots of $q_1(x)$. Consequently, the error value can be calculated as

$$e_i = u'(\alpha^i) = \frac{q_0(\alpha^i)v^{(1)}(\alpha^i)}{q_1^{(1)}(\alpha^i)}. \tag{2.83}$$

Fig. 2.13 is the LCC decoding process. As compared to the simplified KV decoding process in Fig. 2.10, the factorization and Berlekamp-Massey algorithm are eliminated.

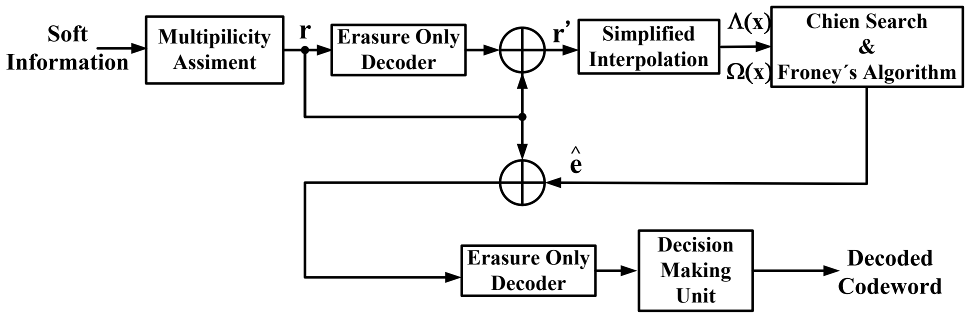


Figure 2.13: Low complexity Chase decoding process

Chapter 3

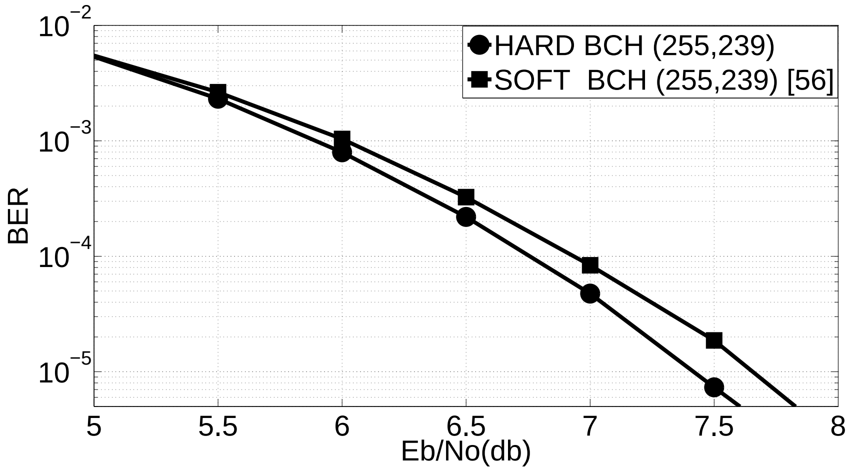
Error Magnitude Type Soft Decoders

In general, the hardware complexity and the storage requirement of a soft decoder are much higher than that of a hard decoder [25–32]. On the other hand, a soft decoding method which collects and deals with the least reliable bits instead of the entire codeword was developed to achieve lower complexity BCH decoders in 1997 [56]. However, this kind of soft decoder corrects the errors only when all actual error locations are collected in the limited possible locations. The decoder is unable to solve any error even though only one error occurred outside these locations. The hardware complexity is improved but the error correcting performance highly depends on the reliability of the input signals. As a result, the soft BCH decoders with existing algorithms provide either better error correcting performance or lower hardware complexity than traditional hard BCH decoders do.

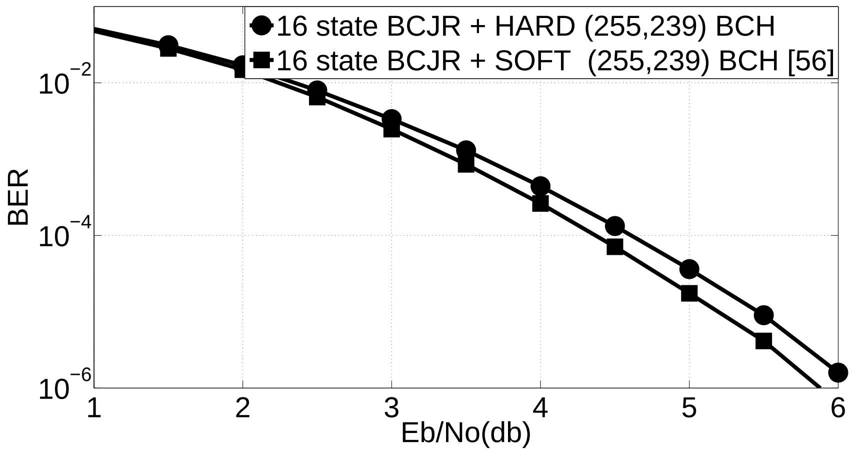
In this chapter, soft decoding algorithms based on the concept of [56] are presented. Section 3.1 introduces the low complexity error magnitude (EM) type approach for BCH codes, which provides low hardware complexity by dealing with the least reliable bits and maintaining the error correcting performance by exploiting soft information from another decoder. As considered to receive soft information from AWGN channel, the high performance approaches for BCH and BCH-Like codes are presented in Section 3.2 and Section 3.3 respectively. The proposed soft decoders perform better performance by compensating one extra error outside the least reliable set while providing comparable hardware complexity. At the end of Chapter 3, Section 3.4 describes the EM-type soft decoder designs.

3.1 Low Complexity EM-Type Approach

In [56], the soft decoder corrects the errors only when all the actual error locations are collected in the limited possible error locations; hence the error correcting performance highly depends on the reliability of the input signals. Fig. 3.1(a) shows that there is about 0.25dB performance loss at bit error rate (BER) = 10^{-5} in AWGN channel as compared to the hard (255,239) BCH decoder, indicating that soft information from AWGN channel is not sufficiently reliable. However, the soft BCH decoder based on [56] can provide 0.25dB performance gain with soft information from a 16-state Bahl-Cocke-Jelinek-Raviv (BCJR) decoder [57] as shown in Fig. 3.1(b). It is because that the BCJR decoder provides more



(a) Information from AWGN Channel



(b) Information from 16-State BCJR

Figure 3.1: Simulation results for BCH (255, 239; 2) under different soft information

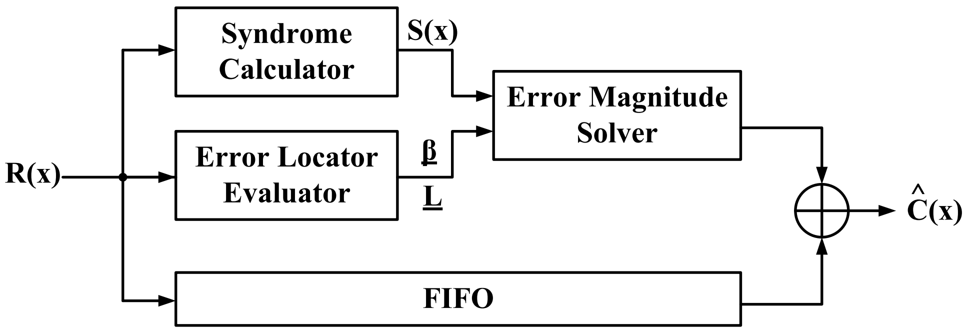


Figure 3.2: Low complexity EM-type soft BCH decoding block diagram

reliable soft information than AWGN channel. Accordingly, the soft BCH decoders with existing algorithms under AWGN channel provide either better error correcting performance or lower hardware complexity than a traditional hard BCH decoder. But, it is possible to provide a soft BCH decoder which has both correcting performance and hardware complexity advantages as long as sufficiently reliable soft information is provided.

Based on the concept of [56], Fig. 3.2 shows the proposed soft BCH decoder that includes three major steps: *syndrome calculator*, *error locator evaluator*, and *error magnitude solver* [58]. From the received polynomial $R(x)$, the syndrome polynomial $S(x) = S_1 + S_2x^1 + \cdots + S_{2t}x^{2t-1}$ is defined in (2.17). With the soft inputs, the decoder chooses $2t$ least reliable inputs and evaluates their corresponding error locators to form the error locator set $\underline{\mathcal{B}} = [\beta_{l_1}, \beta_{l_2}, \ldots, \beta_{l_{2t}}]^T$. Also, the error location set, $\underline{L} = [l_1, l_2, \ldots, l_{2t}]^T$, can be calculated with $\underline{\mathcal{B}}$ because β_{l_i} is the error locator of the l_i-th location and $\beta_{l_i} = \alpha^{l_i}$.

The relation between $\underline{\mathcal{B}}$ and the syndrome vector, $\underline{S} = [S_1, S_2, \ldots, S_{2t}]^T$, can be formulated as

$$\begin{bmatrix} \beta_{l_1} & \beta_{l_2} & \cdots & \beta_{l_{2t}} \\ \beta_{l_1}^2 & \beta_{l_2}^2 & \cdots & \beta_{l_{2t}}^2 \\ \vdots & \vdots & \cdots & \vdots \\ \beta_{l_1}^{2t} & \beta_{l_2}^{2t} & \cdots & \beta_{l_{2t}}^{2t} \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_{2t} \end{bmatrix} = \begin{bmatrix} \mathrm{S}_1 \\ \mathrm{S}_2 \\ \vdots \\ \mathrm{S}_{2t} \end{bmatrix}, \tag{3.1}$$

where γ_i is the error magnitude in the l_i-th location.

Notice that in BCH codes, the valid error magnitude set $\underline{\Gamma} = [\gamma_1, \gamma_2, \ldots, \gamma_{2t}]^T$ must be a binary vector. If the l_i-th location is the exact error location, γ_i is equal to 1; otherwise, γ_i is equal to 0. The $2t \times 2t$ matrix in (3.1) is defined as *error locator matrix* $\mathbf{B}$. The estimated codeword polynomial $\hat{C}(x)$ can be obtained by xoring γ_i with R_{l_i}. In this decoding algorithm, $2t$ locations are collected in $\underline{L}$ such that at most $2t$ errors can be corrected.

To represent the difference between $\underline{S}$ and multiplication result of $\mathbf{B}$ and $\underline{\Gamma}$, a discrepancy vector $\underline{\Delta} = [\delta_1, \delta_2, \ldots, \delta_{2t}]^T$ is defined as

$$\underline{\Delta} = \mathbf{B} \times \underline{\Gamma} + \underline{S}. \tag{3.2}$$

Notice that both the operations in (3.1) and (3.2) are under $GF(2^m)$. It is evident that if all the errors are occurred in the location set $\underline{L}$, the valid $\underline{\Gamma}$ can be determinated to make $\underline{\Delta}$ be a zero vector; otherwise, this decoding approach calculates $\underline{\Gamma}$ as a non-binary vector and fails to correct errors. For example, if there are three errors in 3rd, 7th and 9th locations for a BCH (255, 239; 2) decoder which can correct 2 errors, $\underline{S}$ is expressed as

$$\underline{S} = \begin{bmatrix} \beta_3 + \beta_7 + \beta_9 \\ \beta_3^2 + \beta_7^2 + \beta_9^2 \\ \beta_3^3 + \beta_7^3 + \beta_9^3 \\ \beta_3^4 + \beta_7^4 + \beta_9^4 \end{bmatrix}.$$

In hard BCH decoding, these three errors are unable to be corrected. However, in the case that the decoder chooses the least reliable bits and forms the $\underline{\mathcal{B}}$ as $[\beta_3, \beta_4, \beta_7, \beta_9]$, $\underline{\Delta}$ becomes

$$\underline{\Delta} = \begin{bmatrix} \beta_3 & \beta_4 & \beta_7 & \beta_9 \\ \beta_3^2 & \beta_4^2 & \beta_7^2 & \beta_9^2 \\ \beta_3^3 & \beta_4^3 & \beta_7^3 & \beta_9^3 \\ \beta_3^4 & \beta_4^4 & \beta_7^4 & \beta_9^4 \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{bmatrix} + \begin{bmatrix} \beta_3 + \beta_7 + \beta_9 \\ \beta_3^2 + \beta_7^2 + \beta_9^2 \\ \beta_3^3 + \beta_7^3 + \beta_9^3 \\ \beta_3^4 + \beta_7^4 + \beta_9^4 \end{bmatrix}.$$

After *Gauss Elimination* method, $\underline{\Gamma}$ is calculated as [1, 0, 1, 1] to make $\underline{\Delta}$ be a zero vector. According to $\underline{\mathcal{B}}$ and $\underline{\Gamma}$, the errors at 3rd, 7th and 9th locations can be corrected.

In the proposed decoder, the syndrome calculator is used to calculate $\underline{S}$. The error locator evaluator classifies the soft input to choose $2t$ least reliability and creates $\underline{\mathcal{B}}$ and $\underline{L}$. The error magnitudes solver (EMS) is used to solve (3.1) to get $\underline{\Gamma}$. The Gauss Elimination method is the most intuitive way to solve (3.1) but the complexity is $O(n^3)$. Therefore, the EMS dominates the hardware complexity in the proposed decoder. In this section, two alternative algorithms for improving decoding complexity are revealed. Heuristic Error Magnitudes Solver (H-EMS) uses the characteristic that the valid error magnitude in BCH codes is either 0 or 1, and Björck-Pereyra Error Magnitudes Solver (BP-EMS) employs the quick Vandermonde matrix solution.

In BCH codes, the valid error magnitude in $\underline{\Gamma}$ is a binary value. The problem can be formulated into checking all combinations of γ_i over $GF(2)$ instead of calculating exact error magnitudes. Thus, a $2t$-bit counter is used to execute a heuristic search for all binary combinations. Notice that $S_1^2 = S_2, S_2^2 = S_4, \ldots, S_t^2 = S_{2t}$ in BCH codes, the computation of the even part syndromes $(S_2, S_4, ..., S_{2t})$ can be eliminated. The odd syndrome vector $\underline{S_{odd}} = [S_1, S_3, \ldots, S_{2t-1}]^T$ and the error locator matrix with half rows, $\mathbf{B_{odd}}$, are applied to simplify (3.1) as :

$$\begin{bmatrix} \beta_{l_1} & \beta_{l_2} & \cdots & \beta_{l_{2t}} \\ \beta_{l_1}^3 & \beta_{l_2}^3 & \cdots & \beta_{l_{2t}}^3 \\ \vdots & \vdots & \cdots & \vdots \\ \beta_{l_1}^{2t-1} & \beta_{l_2}^{2t-1} & \cdots & \beta_{l_{2t}}^{2t-1} \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_{2t-1} \\ \gamma_{2t} \end{bmatrix} = \begin{bmatrix} \mathrm{S}_1 \\ \mathrm{S}_3 \\ \vdots \\ \mathrm{S}_{2t-1} \end{bmatrix}. \tag{3.3}$$

The discrepancy vector for (3.3) is modified as

$$\begin{aligned} \underline{\Delta_{odd}} &= [\delta_1, \delta_3, \ldots, \delta_{2t-1}]^T \\ &= \mathbf{B_{odd}} \times \underline{\Gamma} + \underline{S_{odd}}. \end{aligned} \tag{3.4}$$

Notice that only t rows in the $\mathbf{B_{odd}}$ and $\underline{S_{odd}}$ means only half computation in $\underline{\Delta_{odd}}$ calculation as compared with $\underline{\Delta}$ calculation. The complexity of $\underline{\Delta_{odd}}$ calculation is significantly reduced.

The detail of the low complexity implementation of H-EMS is illustrated as followed:

Heuristic EMS Algorithm

```
• Input:
  $\underline{\mathcal{B}}$ and $\underline{S_{odd}}$.

• Initial conditions:
  $\underline{\Gamma} = 0$.

• Step 1:
  Construct the $\mathbf{B_{odd}}$ with $\underline{\mathcal{B}}$.

• Step 2:
  $\underline{\Delta_{odd}} = \mathbf{B_{odd}} \times \underline{\Gamma} + \underline{S_{odd}}$.

• Step 3:
  if $\underline{\Delta_{odd}}$ is a zero vector
    Successful Decoding !!!
    Go to Output
  else
    if $\underline{\Gamma} == 2^{2t} - 1$
      Failed Decoding !!!
    else
      $\underline{\Gamma} = \underline{\Gamma} + 1$
      Go to Step 2

• Output:
  $\underline{\Gamma}$, where $\gamma_i$ is the error magnitude at $l_i$-th location
```

The heuristic search for all binary combinations is processed by iteratively counting $\underline{\Gamma}$ value, and the values of $\underline{\Delta_{odd}}$ are updated and verified whether $\underline{\Delta_{odd}}$ becomes a zero vector or not at each iteration. The decoding procedure successfully completes if certain $\underline{\Gamma}$ makes $\underline{\Delta_{odd}}$ become a zero vector; otherwise, this decoding procedure fails to correct the errors.

On the other hand, let us consider the matrix with the terms of a geometric progression in each row or column: a Vandermonde matrix, which of oder n is the form

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ X_1 & X_2 & \cdots & X_n \\ X_1^2 & X_2^2 & \cdots & X_n^2 \\ \vdots & \vdots & \cdots & \vdots \\ X_1^{n-1} & X_2^{n-1} & \cdots & X_n^{n-1} \end{bmatrix}.$$

To solve (3.1), the inverse matrix of **B** is calculated and it is very complicated to calculate the inverse matrix of an arbitrary matrix. However, the computation of (3.1) can be simplified because **B** is a Vandermonde matrix. The quick Vandermonde matrix computation is applied with Björck-Pereyra algorithm [59, 60] to evaluate the error magnitude efficiently. Instead of using the whole **B** to compute (3.1), the geometric relation in each column makes the proposed BP-EMS only need $\underline{\mathcal{B}}$ for the complexity reduction. Following shows the detail of the low complexity implementation of BP-EMS:

Björck-Pereyra EMS Algorithm

- Input:

  $\underline{\mathcal{B}}$ and $\underline{S}$.

- Step 1:

```
for $(k = 1; k < 2t, k = k + 1)$
   for $(i = 2t; i > k, i = i - 1)$
      $S_i = S_i - \beta_{l_k} S_{i-1}$
```

- Step 2:

```
for $(k = 2t - 1; k > 0, k = k - 1)$
```

```
    for $(i = k+1; i \leq 2t, i = i+1)$
        $S_i = S_i/(\beta_{l_i} - \beta_{l_{i-k}})$
    for $(i = k; i < 2t, i = i+1)$
        $S_i = S_i - S_{i+1}$
```

- Step 3:

```
for $(k = 1; k \leq 2t, k = k+1)$
    $S_i = S_k/\beta_{l_k}$
```

- Step 4:

```
if $\underline{S}$ is a binary sequence
    Successful Decoding !!!
    Go to Output
else
    Failed Decoding !!
```

- Output:

  $\underline{S}$, where S_i is the error magnitude at l_i-th location.

In the proposed BP-EMS algorithm, the variable S_i, which initially represents the i-th syndrome value, is updated iteratively. Each calculation of the syndrome represents a row operation in (3.1). After all computations, S_i indicates the l_i-th error magnitude. Although the valid error magnitude in BCH codes is either 0 or 1, a non-binary S_i may be generated because not all errors are in the $\underline{L}$. Therefore, a binary sequence check of $\underline{S}$ is applied in the final step to determine the decoding successful or not. The proposed BP-EMS is composed of division, multiplication and addition operations. The regular operations make BP-EMS suitable for low complexity hardware implementation.

In DVB-S2 system, (32400,32208) BCH over $GF(2^{16})$ is defined to be concatenated with (64800,32400) LDPC code. Fig. 3.3 presents the BER performance at 50 LDPC decoding iterations under QPSK modulation and AWGN channel. The proposed soft BCH decoder has similar performance as the hard BCH decoder in DVB-S2 system at BER $= 10^{-5}$.

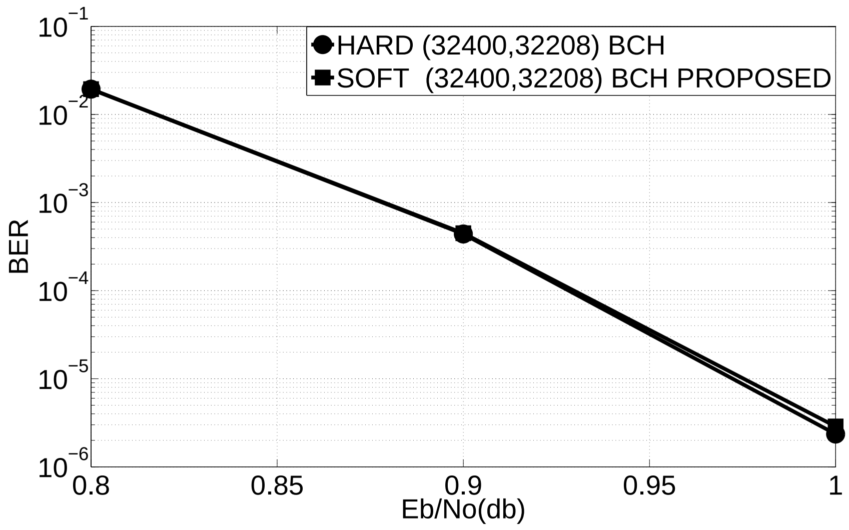


Figure 3.3: Simulation results for (32400, 32208; 12) BCH in DVB-S2 system.

3.2 High Performance EM-Type Approach

Although the soft BCH decoder introduced in Section 3.1 has significantly low hardware complexity, the error correcting performance highly depends on the reliability of the input signals. Without another decoder providing reliable soft information, it has worse error correcting ability than the hard BCH decoder. In this section, soft decoding algorithms having similar concept as [56] but with one extra error compensation are presented to enhance the correcting performance for BCH decoders. Consequently, in contrast to the conventional hard BCH decoders, the proposed soft BCH decoders perform better performance by compensating one extra error outside the least reliable set and provide comparable hardware complexity by dealing with the least reliable bits.

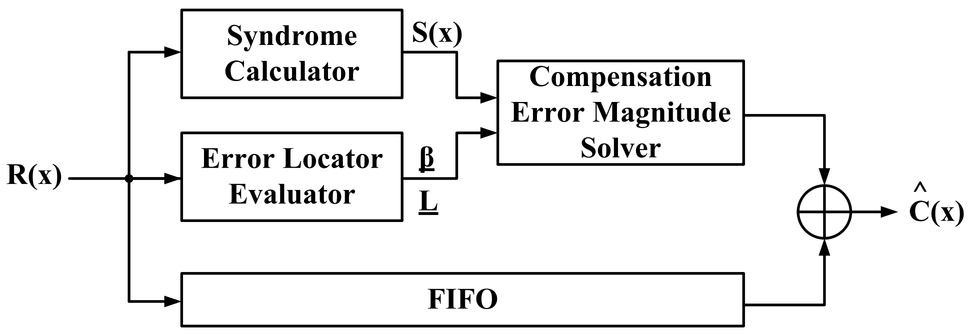


Figure 3.4: High performance EM-type soft BCH decoding block diagram

The proposed soft BCH decoder shown in Fig. 3.4 includes three major blocks: *syndrome calculator*, *error locator evaluator*, and *compensation error magnitude solver (CEMS)* The syndrome calculator is used to calculate the syndrome polynomial $S(x)$. The error locator evaluator classifies the soft input to choose $2t$ least reliabilities and creates the error locator set $\underline{\mathcal{B}}$ as well as the error location set $\underline{L}$. Based on these informations, the CEMS is applied to find the error locations. According to (3.1) and (3.2), it is evident that if all errors are located within the location set $\underline{L}$, the valid $\underline{\Gamma}$ can be determinated to make $\underline{\Delta}$ be a zero vector. Otherwise, $\underline{\Gamma}$ is calculated as a non-binary vector and this decoding approach fails to correct errors. If any error occurred outside $\underline{L}$, the decoder is unable to solve any error, resulting in the lower correcting performance. However, the error correcting ability can be enhanced by not only correcting errors located inside $\underline{L}$ but also correcting errors outside $\underline{L}$. Under the analysis of $\underline{\Delta}$, an error located at l_{miss} and outside $\underline{L}$ induces $\underline{\Delta} = [\beta_{l_{miss}}, \beta^2_{l_{miss}}, \ldots, \beta^{2t}_{l_{miss}}]$, where $\beta_{l_{miss}} = \alpha^{l_{miss}}$. Notice that a *geometrical progression* is a sequence of numbers where each term can be formulated by multiplying the previous one by a common ratio. In order to improve the error correcting ability, we can additionally check whether $\underline{\Delta}$ has the property of a geometrical progression and make a compensation for finding the missing location l_{miss} from $\underline{\Delta}$. For example, if there are four errors at the 1st, 3rd, 5th and 9th locations for a BCH (255, 239; 2) decoder, $\underline{S}$ can be expressed as

$$\underline{S} = \begin{bmatrix} \beta_1 + \beta_3 + \beta_5 + \beta_9 \\ \beta_1^2 + \beta_3^2 + \beta_5^2 + \beta_9^2 \\ \beta_1^3 + \beta_3^3 + \beta_5^3 + \beta_9^3 \\ \beta_1^4 + \beta_3^4 + \beta_5^4 + \beta_9^4 \end{bmatrix}.$$

The decoder which collects $\underline{\mathcal{B}} = [\beta_1, \beta_3, \beta_6, \beta_9]$ from the least reliable bits determined by the error locator evaluator is unable to correct errors without compensation procedure because of the missing β_5. However, once $\underline{\Gamma} = [1, 1, 0, 1]$, $\underline{\Delta}$ becomes

$$\begin{aligned} \underline{\Delta} &= \begin{bmatrix} \beta_1 & \beta_3 & \beta_6 & \beta_9 \\ \beta_1^2 & \beta_3^2 & \beta_6^2 & \beta_9^2 \\ \beta_1^3 & \beta_3^3 & \beta_6^3 & \beta_9^3 \\ \beta_1^4 & \beta_3^4 & \beta_6^4 & \beta_9^4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} \beta_1 + \beta_3 + \beta_5 + \beta_9 \\ \beta_1^2 + \beta_3^2 + \beta_5^2 + \beta_9^2 \\ \beta_1^3 + \beta_3^3 + \beta_5^3 + \beta_9^3 \\ \beta_1^4 + \beta_3^4 + \beta_5^4 + \beta_9^4 \end{bmatrix} \\ &= [\beta_5, \beta_5^2, \beta_5^3, \beta_5^4]. \end{aligned}$$

$\underline{\Delta}$ is a geometrical progression with β_5 as the common ratio. Therefore, the decoder can get the missing location by verifying $\underline{\Delta}$. Then not only errors at 1st, 3rd and 9th locations but also an error at 5th location can be corrected.

Moreover, if there are five errors at the 1st, 3rd, 5th, 6th and 9th locations, the decoder can collect at most four ($2t$) exact error locations in $\underline{L}$, such as $\underline{L} = [1, 3, 6, 9]$. While the decoder provides $\underline{\Gamma} = [1, 1, 1, 1]$, the $\underline{\Delta}$ becomes $[\beta_5, \beta_5^2, \beta_5^3, \beta_5^4]$. The missing location still can be found from the geometrical progression $\underline{\Delta}$. Accordingly, the proposed compensation soft BCH decoder can correct at most $2t + 1$ errors. Except for the l_{miss}, other error locations are l_i whose corresponding γ_i equals 1. The estimated codeword polynomial $\hat{C}(x)$ can be obtained by inversing values at these error locations in the received polynomial $R(x)$.

The CEMS is applied to calculate $\underline{\Gamma}$ and $\underline{\Delta}$ according to (3.1) and (3.2). Similar as in Section 3.1, the binary characteristic of BCH codes allows we solve (3.1) and (3.2) checking all combinations of γ_i over $GF(2)$ instead of calculating exact error magnitudes with Gauss Elimination method. A $2t$-bit counter is employed to execute a heuristic search for all binary combinations. Also, $S_1^2 = S_2, S_2^2 = S_4, \ldots, S_t^2 = S_{2t}$ in BCH codes, the computation of the even part syndromes $(S_2, S_4, ..., S_{2t})$ can be eliminated. The equation (3.1) and (3.2) can be modified as (3.3) and (3.4) respectively. Following steps illustrate the details of the proposed algorithm for CEMS.

Compensation EMS Algorithm

- Input:

$\underline{\mathcal{B}}$ and $\underline{S_{odd}}$.

- Initial conditions:

  $\underline{\Gamma} = 0$.

- Step 1:

  Construct the $\mathbf{B_{odd}}$ with $\underline{\mathcal{B}}$.

- Step 2:

  $\underline{\Delta_{odd}} = \mathbf{B_{odd}} \times \underline{\Gamma} + \underline{S_{odd}}$.

- Step 3:

```
if $\underline{\Delta_{odd}}$ is a zero vector
    $l_{miss}$ = NULL
    Go to Output
else if $\underline{\Delta_{odd}}$ is a geometrical progression
    Go to Step 4
else
    if $\underline{\Gamma} == 2^{2t} - 1$
        Failed Decoding !!!
    else
        $\underline{\Gamma} = \underline{\Gamma} + 1$
        Go to Step 2
```

- Step 4:

  Find l_{miss} according to the relation: $\delta_1 = \alpha^{l_{miss}}$

- Output:

  $\underline{\Gamma}$ and l_{miss}, where γ_i is the error magnitude at l_i-th location.

A heuristic search for all binary combinations is completed by iteratively counting $\underline{\Gamma}$ value from 0 to 2^{2t-1}. At each iteration, the solver verifies whether or not $\underline{\Delta_{odd}}$ becomes a

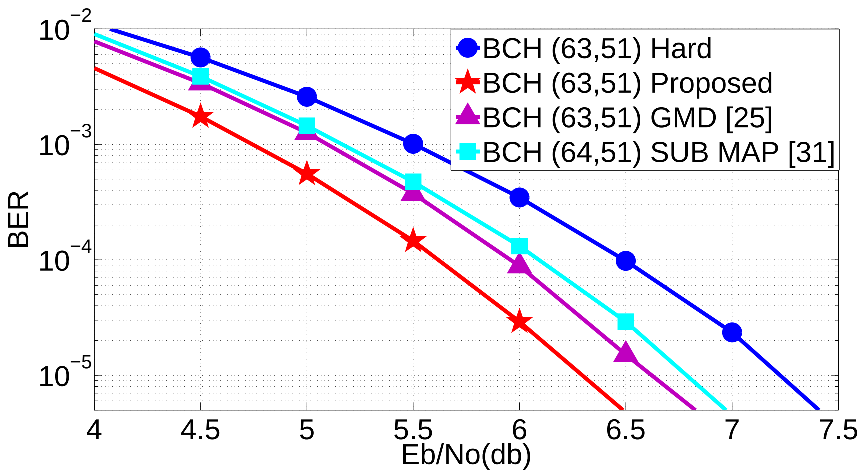


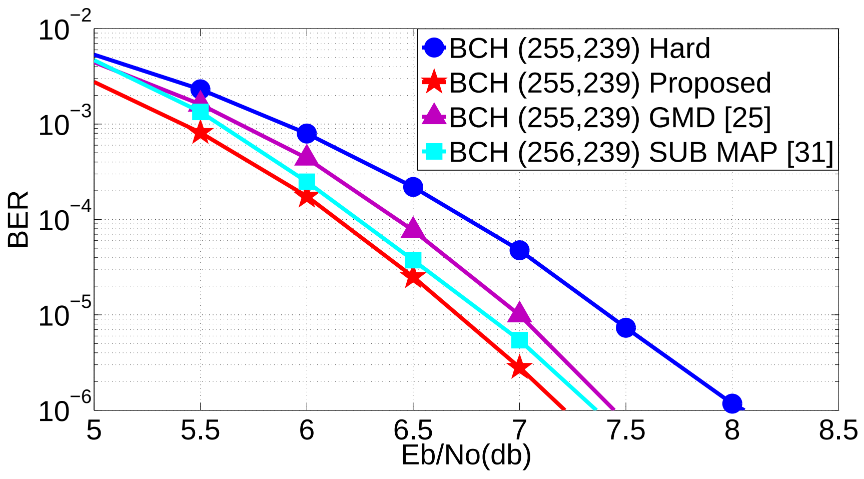
Figure 3.5: Simulation results for the proposed soft BCH (63, 51; 2) decoder.

geometrical progression. Once the geometrical progression check passes at certain $\underline{\Gamma}$ value, the corresponding error locations in $\underline{L}$ and l_{miss} can be found with $\underline{\Gamma}$ and $\underline{\Delta_{odd}}$.

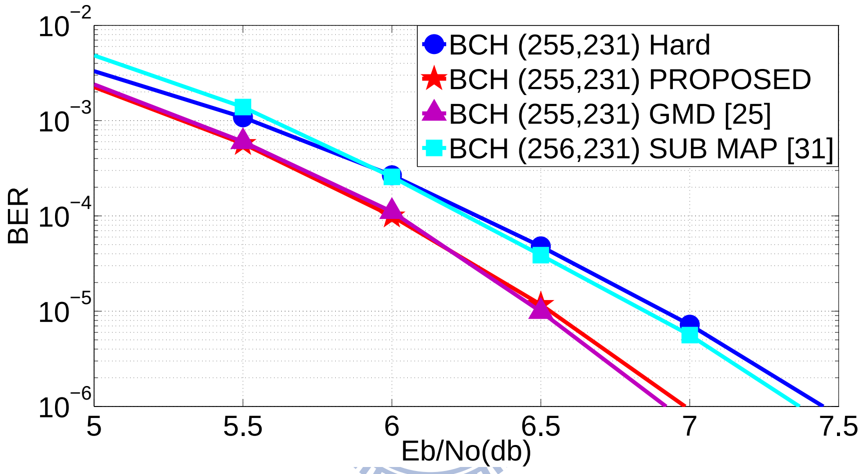
For the purpose of comparing with existing methods, our proposed designs are compared with traditional hard decision, GMD [25] and 2 iterations sub-optimum MAP [31] decoding algorithms. In all cases, BPSK modulation and AWGN channel are used and all the performances are compared at 10^{-5} BER.

For the 63-bit codeword length, the simulation results of 2-error-correcting BCH code is presented in Fig. 3.5. The achieved coding gain over the hard BCH (63, 51; 2) decoder is about 0.95 dB for the proposed soft BCH (63, 51; 2) decoder. Compared with other existing soft-decision methods, the proposed decoder can obtain 0.3 dB and 0.5 dB coding gains over GMD and sub-optimum MAP decoders respectively.

Fig. 3.6(a) and Fig. 3.6(b) provide the simulation results of 2-error-correcting and 3-error-correcting BCH codes with 255-bit codeword length. As compared to the hard BCH (255, 239; 2) decoder, the proposed soft BCH (255, 239; 2) decoder outperforms 0.75 dB. Our proposed decoder can outperform 0.35 dB and 0.13 dB coding gain as compared with GMD and sub-optimum MAP decoders respectively. The coding gain of our soft BCH (255, 231; 3) decoders is 0.33 dB and 0.4dB over sub-optimum MAP and conventional hard BCH (255, 231; 3) decoders. Also our proposed decoder is comparable with GMD decoder.



(a) BCH (255, 239; 2)



(b) BCH (255, 231; 3)

Figure 3.6: Simulation results for the proposed soft BCH-Like decoders with 255-bit codeword length.

3.3 High Performance EM-Type Approach for BCH-Like Codes

A new constructed code, referred as *BCH-Like* code, is proposed to enhance the code rate according to the original definition of the BCH codes. As compared to BCH codes with similar code rate, the BCH-Like codes with proposed decoding scheme can provide better error correcting performance.

For an $(N, K; t)$ t-error-correcting BCH code over $GF(2^m)$, the generator polynomial $g(x)$ is defined in (2.5) with minimal polynomials: $M_1(x), M_3(x), \cdots, M_{2t-1}(x)$. Notice that

$M_i(x)$ is the minimal polynomial for α^i, and the degree of the minimal polynomial is a factor of m. If m is even, there must exist a degree-2 minimal polynomial, such as $M_{21}(x)$ in $GF(2^6)$ or $M_{85}(x)$ in $GF(2^8)$. On the other hand, if m is odd, the existence of minimal polynomial with lower degree depends on whether m is a prime number. For instance, the $M_{73}(x)$ is the degree-3 minimal polynomial in $GF(2^9)$ whereas there is no minimal polynomial with degree less than 7 in $GF(2^7)$.

While the minimal polynomial $M_{2t-1}(x)$ in (2.5) is replaced with the minimal polynomial with less degree, a new generator polynomial used to construct higher code rate codes can be formulated as

$$g_{like}(x) = LCM\{M_1(x), M_3(x), \cdots, M_{2t-3}(x), M_\rho(x)\}, \tag{3.5}$$

where $deg(M_\rho(x)) < deg(M_{2t-1}(x))$. Consequently, an $(N, K'; t)$ BCH-Like code is constructed by

$$C(x) = m(x) \times x^{N-K'} + r(x), \tag{3.6}$$

and

$$r(x) = m(x) \times x^{N-K'} \bmod g_{like}(x), \tag{3.7}$$

where $m(x)$ represents the message polynomial and the information length $K' > K$ since $deg(g_{like}(x)) < deg(g(x))$. Notice that the item t in the BCH-Like codes means the number of minimal polynomials utilized to construct $g_{like}(x)$.

TABLE 3.1 lists the construction of BCH-Like codes over $GF(2^6) \sim GF(2^{10})$. For example, the BCH-Like (1023, 1001; 3) code is constructed from BCH (1023, 993; 3) code by replacing the degree-10 $M_5(x)$ with the degree-2 $M_{341}(x)$ in $GF(2^{10})$. Notice that an $(N, K'; t)$ BCH-Like code can be constructed with different $M_\rho(x)$ and the error correcting performance of BCH-Like codes will highly depend on which $M_\rho(x)$ is chosen. For instance, there are three minimal polynomials with degree 4 over $GF(2^8)$: $M_{17}(x)$, $M_{51}(x)$ and $M_{109}(x)$, and the BCH-Like (255, 243; 2) decoder can be constructed with anyone of them. However, the BCH-Like (255, 243; 2) decoder with $M_{51}(x)$ has the best error correcting

Table 3.1: List of BCH-Like codes over $GF(2^6) \sim GF(2^{10})$

	BCH-Like Code	Corresponding BCH Code	ρ	$\deg(M_\rho(x))$
$GF(2^6)$	(63, 54; 2)	(63, 51; 2)	27	3
	(63, 48; 3)	(63, 45; 3)	27	3
	(63, 49; 3)	(63, 45; 3)	21	2
$GF(2^8)$	(255, 243; 2)	(255, 239; 2)	51	4
	(255, 235; 3)	(255, 231; 3)	17	4
	(255, 237; 3)	(255, 231; 3)	85	2
$GF(2^9)$	(511, 490; 3)	(511, 484; 3)	219	3
$GF(2^{10})$	(1023, 1008; 2)	(1023, 1003; 2)	33	5
	(1023, 998; 3)	(1023, 993; 3)	99	5
	(1023, 1001; 3)	(1023, 993; 3)	341	2

performance according to our simulation results.

Based on the proposed BCH-Like codes construction method, there is only $2t-2$ consecutive roots for the codeword set. According to the BCH argument [3], the minimal distance of the BCH-Like codeword set is only guaranteed as $2t-1$ while that of the BCH codeword set is $2t+1$. As a result, the lower correcting performance is observed while the hard decision decoding is applied. However, the proposed soft decoding method is also suitable for BCH-Like codes and the maximum error correcting ability is $2t+1$ as well. Similar to BCH codes, only odd syndromes have to be calculated for the efficient computations in BCH-Like codes. According to (3.6) and (3.7), $\alpha^1, \alpha^3, \cdots, \alpha^{2t-3}$, and α^ρ are the roots of BCH-Like codewords, $C(x)$. The relation between $\underline{\mathcal{B}}$ and the modified odd syndrome vector $\underline{S'_{odd}} = [S_1, S_3, \ldots, S_{2t-3}, S_\rho]^T$ can be formulated as

$$\begin{bmatrix} \beta_{l_1} & \beta_{l_2} & \cdots & \beta_{l_{2t}} \\ \beta_{l_1}^3 & \beta_{l_2}^3 & \cdots & \beta_{l_{2t}}^3 \\ \vdots & \vdots & \cdots & \vdots \\ \beta_{l_1}^{2t-3} & \beta_{l_2}^{2t-3} & \cdots & \beta_{l_{2t}}^{2t-3} \\ \beta_{l_1}^{\rho} & \beta_{l_2}^{\rho} & \cdots & \beta_{l_{2t}}^{\rho} \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \vdots \\ \gamma_{2t-1} \\ \gamma_{2t} \end{bmatrix} = \begin{bmatrix} \mathrm{S}_1 \\ \mathrm{S}_3 \\ \vdots \\ \mathrm{S}_{2t-3} \\ \mathrm{S}_\rho \end{bmatrix}, \tag{3.8}$$

where the modified error locator matrix with half row is called as $\mathbf{B'_{odd}}$. The only difference between (3.3) and (3.8) is the bottom row of each matrix; therefore, the BCH-Like decoder has the similar decoding flow as shown in Fig. 3.4. The modifications are designed for dealing with S_ρ and $\beta_{l_i}^{\rho}$ in the syndrome calculator and the CEMS. The modified discrepancy vector $\underline{\Delta'_{odd}} = [\delta'_1, \delta'_3, \ldots, \delta'_{2t-3}, \delta'_\rho]^T$ is defined as

$$\underline{\Delta'_{odd}} = \mathbf{B'_{odd}} \times \underline{\Gamma} + \underline{S'_{odd}}. \tag{3.9}$$

If all errors are located within the location set $\underline{L}$, the valid $\underline{\Gamma}$ can be determined to make $\underline{\Delta'_{odd}}$ be a zero vector. In the case that there is an error location l_{miss} outside $\underline{L}$, $\underline{\Delta'_{odd}}$ becomes $[\beta_{l_{miss}}, \beta_{l_{miss}}^{3}, \ldots, \beta_{l_{miss}}^{2t-3}, \beta_{l_{miss}}^{\rho}]$, which is still a geometrical progression excluding the last term, $\beta_{l_{miss}}^{\rho}$. A modified CEMS (M-CMES) for BCH-Like codes is provided to additionally check δ'_ρ term for finding l_{miss}, and the details of the proposed algorithm for M-CEMS are illustrated as the following steps.

Modified Compensation EMS Algorithm

- Input:

  $\underline{\mathcal{B}}$ and $\underline{S'_{odd}}$.

- Initial conditions:

  $\underline{\Gamma} = 0$.

- Step 1:

  Construct $\mathbf{B'_{odd}}$ based on $\underline{\mathcal{B}}$.

- Step 2:

  $$\begin{aligned}\underline{\Delta'_{odd}} &= [\delta'_1, \delta'_3, \ldots, \delta'_{2t-3}, \delta'_\rho]^T \\ &= \mathbf{B'_{odd}} \times \underline{\Gamma} + \underline{S'_{odd}}\end{aligned}$$

- Step 3:

  if $\underline{\Delta_{odd}}$ is a zero vector

  $\quad l_{miss} = \text{NULL}$

```
    Go to Output
else if $\delta'_\rho == \delta'^{\rho}_1$ && $[\delta'_1, \delta'_3, \ldots, \delta'_{2t-3}]^T$ is a geometrical progression
    Go to Step 4
else
    if $\underline{\Gamma} == 2^{2t} - 1$
        Failed Decoding !!!
    else
        $\underline{\Gamma} = \underline{\Gamma} + 1$
        Go to Step 2
```

- Step 4:

  Find l_{miss} according to the relation: $\delta'_1 = \alpha^{l_{miss}}$

- Output:

  $\underline{\Gamma}$ and l_{miss}, where γ_i is the error magnitude at l_i-th location.

At Step 3, not only the geometrical progression check but also the relation between δ'_1 and δ'_ρ are evaluated for an extra error compensation for BCH-Like codes. After M-CEMS process, for those γ_i equal to 1, the corresponding l_i and l_{miss} are the estimated error locations.

For the purpose of comparing with existing methods, our proposed designs are simulated

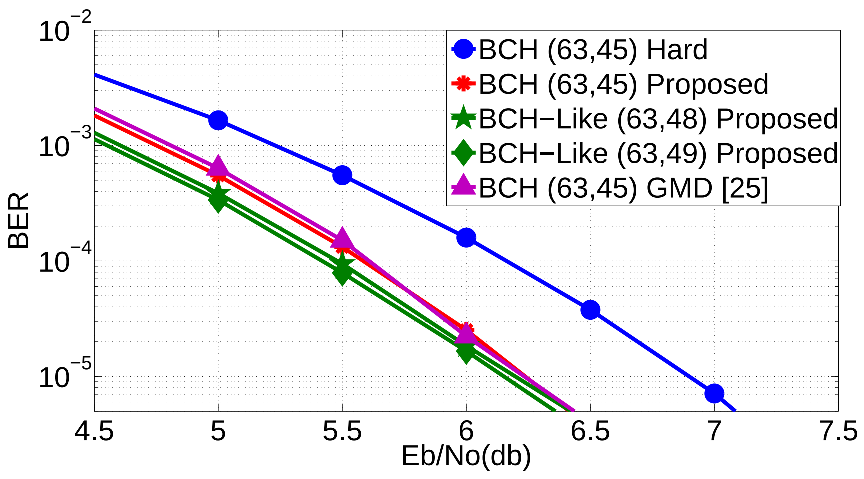
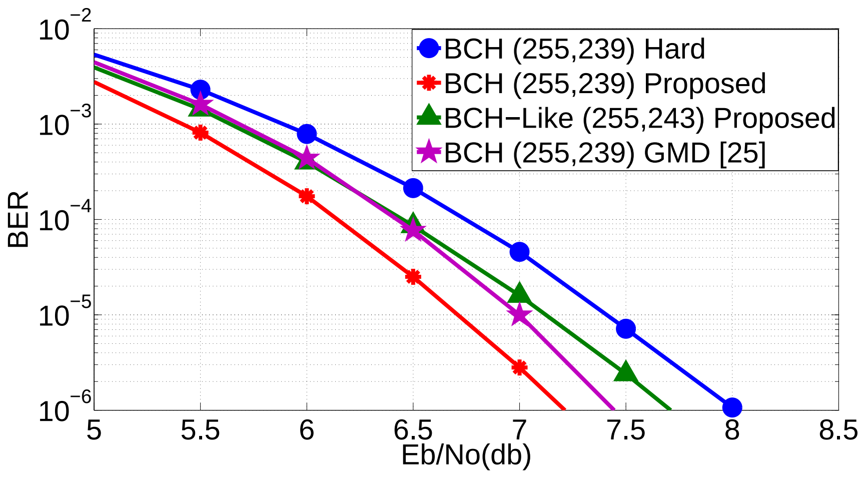
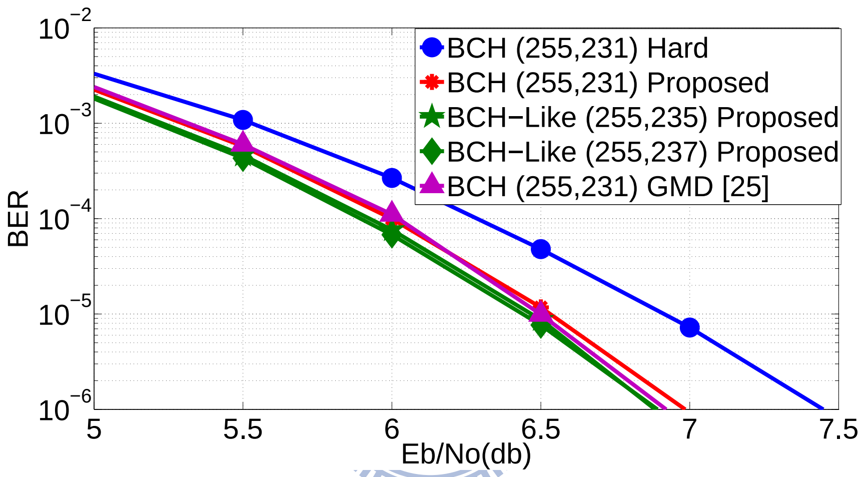


Figure 3.7: Simulation results for the proposed soft BCH-Like (63, 48; 3) and (63, 49; 3) decoders



(a) BCH-Like (255, 243; 2)



(b) BCH-Like (255, 235; 3) and (255, 237; 3)

Figure 3.8: Simulation results for the proposed soft BCH decoders with 255-bit codeword length.

with traditional hard decision and GMD [25] decoding algorithms. In all cases, BPSK modulation and AWGN channel are used and all performances are compared at 10^{-5} BER.

Fig. 3.7 shows the simulation results of 3-error-correcting BCH-Like code with 63-bit codeword length in $GF(2^6)$. The coding gains over the hard BCH (63, 45; 3) decoder are 0.7 dB and 0.75 dB respectively for our soft BCH-Like (63, 48; 3) and (63, 49; 3) decoders. Even though the minimal distance of BCH-Like (63, 48; 3) and (63, 49; 3) codeword sets is less than 7, the performances of our soft BCH-Like (63, 48; 3) and (63, 49; 3) decoders are comparable with that of the GMD decoder.

For the 255-bit codeword length in $GF(2^8)$, the simulation results of 2-error-correcting

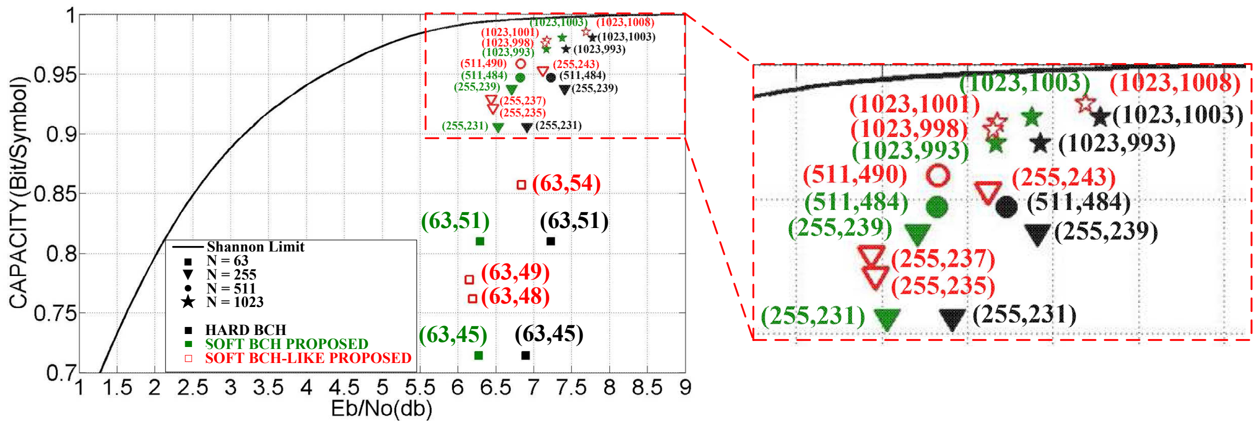


Figure 3.9: Simulation results of BCH and BCH-Like decoders at 10^{-5} BER under AWGN channel and BPSK modulation.

and 3-error-correcting BCH-Like codes are presented in Fig. 3.8(a) and Fig. 3.8(b) respectively. As compared to the hard BCH (255, 239; 2) decoder, our soft BCH-Like (255, 243; 2) decoder outperforms 0.3 dB. The corresponding coding gains over the hard BCH (255, 231; 3) decoder are around 0.5 dB for both soft BCH-Like (255, 235; 3) and (255, 237; 3) decoders. Moreover, our soft BCH-Like (255, 235; 3) and (255, 237; 3) decoders achieve similar coding gains in contrast to GMD decoder according to the higher code rate and one extra error compensation.

Although the proposed BCH-Like codes will decrease the error-correcting capability from t to $t-1$ if hard decision decoding is applied, our soft BCH-Like decoders with M-CEMS, like the proposed soft BCH decoders with CEMS, can utilize t syndrome values in the decoding procedures. As a result, both the code rate and the error correcting performance can be improved by our soft BCH-Like decoders. As shown in Fig. 3.7 and Fig. 3.8, our soft BCH-Like decoders outperform original BCH decoders with lower code rate, implying they can provide the better coding gain over BCH decoders with similar code rate. For example, the proposed soft BCH-Like (255, 237; 3) decoder outperforms 1.0 dB and 0.3 dB as compared to the hard and the proposed soft BCH (255, 239; 2) decoders respectively from Fig. 3.6(a) and Fig. 3.8(b).

Fig. 3.9 demonstrates the performances of BCH and BCH-Like decoders at 10^{-5} BER under AWGN channel and BPSK modulation. The capacity (symbol/bit) is equal to the

Table 3.2: Performance summary of BCH-Like codes over $GF(2^6) \sim GF(2^{10})$.

	BCH-Like Code	Corresponding BCH Code	Coding Gain [1](dB)	Coding Gain [2](dB)	Coding Gain [3](dB)
$GF(2^6)$	(63, 54; 2)	(63, 51; 2)	0.38	0.88	0
	(63, 48; 3)	(63, 45; 2)	0.7	1.1	0.44
	(63, 49; 3)	(63, 45; 2)	0.75	1.19	0.57
$GF(2^8)$	(255, 243; 2)	(255, 239; 2)	0.29	0.68	0
	(255, 235; 3)	(255, 231; 3)	0.45	0.75	0.36
	(255, 237; 3)	(255, 231; 3)	0.47	0.87	0.48
$GF(2^9)$	(511, 490; 3)	(511, 484; 3)	0.4	0.71	0.3
$GF(2^{10})$	(1023, 1008; 2)	(1023, 1003; 2)	0.19	0.49	0.82
	(1023, 998; 3)	(1023, 998; 3)	0.28	0.9	0
	(1023, 1001; 3)	(1023, 998; 3)	0.25	0.97	0.1

[1] The performance gain as compared with corresponding hard BCH decoder.
[2] The performance gain closer to corresponding Shannon limit in contrast to corresponding hard BCH decoder.
[3] The performance gain closer to corresponding Shannon limit in contrast to corresponding proposed soft BCH decoder.

code rate in the BPSK modulator [61]. Notice that codes with higher code rate have smaller gap code to the Shannon limit. Accordingly, the performances of our soft BCH-Like decoders are closer to Shannon limit as compared to the corresponding hard and soft BCH decoders. TABLE 3.2 summarizes the performance gains of BCH-Like codes over $GF(2^6) \sim GF(2^{10})$. Notice that these three performance gains achieved by the soft BCH-Like decoder are provided while the corresponding hard BCH decoders and proposed soft BCH decoders are compared respectively. In contrast to hard BCH decoders, the performances of our soft BCH-Like (63, 49; 3) and (255, 237; 3) decoders are 1.2 dB and 0.9 dB closer to Shannon limit respectively. Even compared with the proposed soft BCH decoders, our soft BCH-Like (63, 49; 3) and (255, 237; 3) decoders achieve 0.5 dB and 0.6 dB performance gain closer to Shannon limit.

3.4 Design of EM-Type Soft Decoders

As mentioned in Section 3.1 $\sim$ Section 3.3, the proposed soft decoders include three major steps. The syndrome calculator and error locator evaluator architectures are designed to deal with serial input. Following are the efficient architectures of different EMS. The architecture comparison between the hard BCH decoder and the proposed soft BCH decoders is discussed in the end of this section as well.

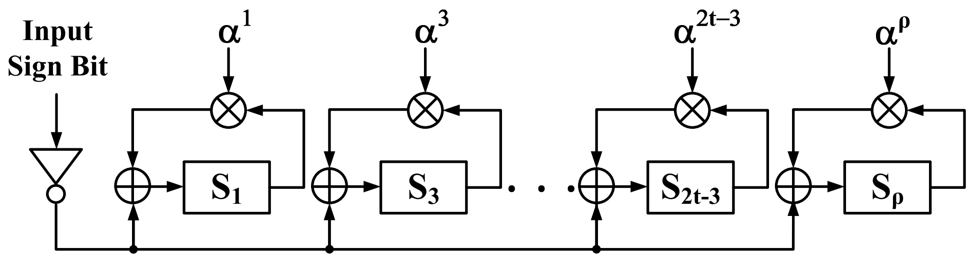


Figure 3.10: Syndrome calculator architecture.

3.4.1 Syndrome Calculator

The syndrome calculator generates the syndrome polynomial $S(x)$ from the received polynomial $R(x)$. Fig. 3.10 shows totally t syndrome cells composed of constant multipliers, adders and registers because only odd syndromes will be calculated for the efficient implementation. The difference between the syndrome calculator of the proposed soft BCH and BCH-Like decoders is the last syndrome cell, S_ρ. For soft BCH decoder, S_ρ will be replaced by S_{2t-1} with the constant multiplier α^{2t-1}. Receiving data from R_{N-1} to R_0, the i-th syndrome cell can be formulated as

$$\begin{aligned} S_i &= R(\alpha^i) \\ &= (((R_{N-1}\alpha^i + R_{N-2})\alpha^i + R_{N-3})\alpha^i + \cdots)\alpha^i + R_0 \end{aligned} \quad (3.10)$$

Totally N cycles are required to receive and calculate all data for a BCH codeword. At each cycle, the partial result is stored in the register as partial syndrome value, which is multiplied with α^i and then accumulated with the received data. Once all the data is received, the accumulated value is the i-th syndrome value. Notice that the input of the soft decoder is its reliability value, and R_{N-j} is the inverse of the j-th input sign bit.

3.4.2 Error Locator Evaluator

The proposed error locator evaluator shown in Fig. 3.11 classifies the soft inputs for choosing $2t$ least reliable inputs according to the candidate reliabilities, $R_{l_1}, R_{l_2}, \ldots, R_{l_{2t}}$. Their corresponding error locators β_{l_i} and error locations l_i are also calculated and stored in registers as

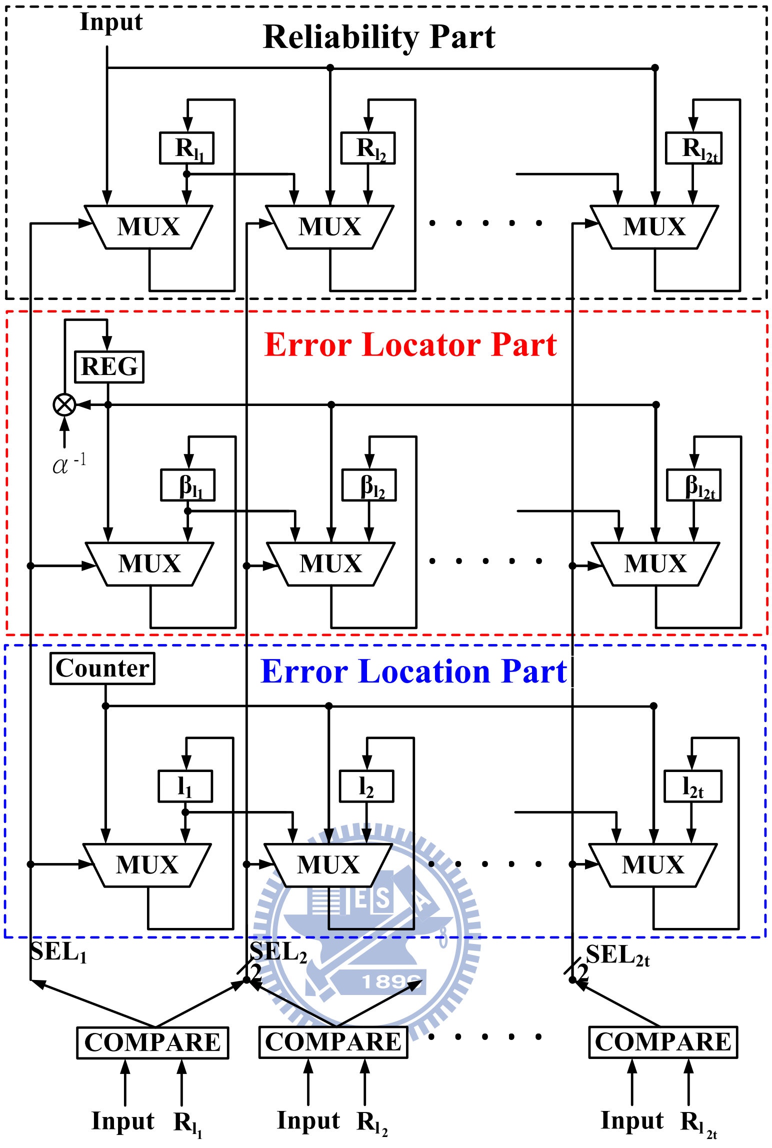


Figure 3.11: Error locator evaluator architecture.

soon as R_{l_i} is chosen. The proposed error locator evaluator architecture includes three major parts: the *reliability part*, the *error locator part* and the *error location part*. The reliability part stores $2t$ candidate reliabilities $R_{l_1} \sim R_{l_{2t}}$ from 1st stage to $2t$-th stage. The values stored in the reliability part are compared with soft inputs to generate the select signals SEL_i for controlling the multiplexers. In the i-th stage, if the input is smaller than $R_{l_{i-1}}$, the i-th stage value is updated with $(i-1)$-th stage value. If the input is greater than $R_{l_{i-1}}$ and smaller than R_{l_i}, the i-th stage value is updated with the input value. Otherwise, the i-th stage value holds its current value.

The error locator part constructs $\underline{\mathcal{B}}$. The input data is received from R_{N-1} to R_0 serially, and the error locator of the l_i-th location is α^{l_i}. Accordingly, the error locator can be computed by multiplying α^{-1} with register REG, which stores the error locator of previous location. The error locator part uses a constant multiplier to calculate the error locator of each input. Notice that α^{N-1} is the initial value of register REG. The error location part utilizes a counter to compute the corresponding error location l_i of each R_{l_i} to form the error location set $\underline{L}$. The least reliable bits, instead of the entire codeword, are handled by the proposed decoding method and the possible error locations are only searched in the $\underline{L}$. Hence, Chien search procedure is no longer required and lots of redundant decoding latencies can be eliminated.

3.4.3 Error Magnitude Solver

In Section 3.1 $\sim$ Section 3.3, we have proposed 4 type EMS: H-EMS, BP-EMS, CEMS and M-CEMS. The H-EMS and BP-EMS are proposed for low complexity EM-type soft BCH decoders, where H-EMS is more suitable for smaller t applications while BP-EMS is adequate for larger t applications.

Based on H-EMS algorithm, Fig. 3.12 illustrates the proposed H-EMS architecture to evaluate $\underline{\Delta_{odd}} = \mathbf{B_{odd}} \times \underline{\Gamma} + \underline{S_{odd}}$ while given $\underline{S_{odd}}$ and $\underline{\mathcal{B}}$. There are $2t^2$ registers used to store all entries in $\mathbf{B_{odd}}$ matrix. The register values of the i-th column forms a geometric progression with the common ratio $\beta_{l_i}^2$. In the i-th column, the initial values of the top registers is set as β_{l_i} so that the output of the squarer will always be $\beta_{l_i}^2$. Also, iteratively multiplied by $\beta_{l_i}^2$, the bottom register with initial values β_{l_i} as well generates $\beta_{l_i}^{2j+1}$ for j = 0 $\sim$ t-1. Each register, except for the top register, iteratively updates upper register with the generated value. Hence, $\mathbf{B_{odd}}$ matrix is calculated with totally only $2t$ multipliers and squarers in t-1 cycles and is stored in the registers. The registers will hold their values in matrix multiplication procedure.

Matrix multiplication is evaluated in the following 2^{2t} cycles. By counting $\underline{\Gamma}$ value, a heuristic search for all binary combinations can be completed. The $\underline{\Gamma}$ block in Fig. **??** is represented as a counter to generate a new $\underline{\Gamma}$ iteratively. At each iteration, each $\beta_{l_i}^{j}$ value will be calculated with γ_i, and the solver verifies whether $\underline{\Delta_{odd}}$ becomes a zero vector or not. A successful decoding is completed when $\underline{\Gamma}$ satisfies the verification.

On the other hand, the proposed BP-EMS has division, multiplication and addition operations from BP-EMS algorithm. To minimize the hardware complexity with only one divider, one multiplier and several adders simultaneously, the operations in BP-EMS algorithm are coped with sequentially, and take $4t^2 - 2t$, $2t^2 - t$, $2t$ cycles for Step 1 $\sim$ Step 3 respectively. The Step 4, binary sequence check of $\underline{S}$, is executed as soon as each S_i value has been fully updated. Notice that the multiplier can be shared if the divider can be decomposed into an inversion unit and a multiplier. Thus, as shown in Fig. 3.13, BP-EMS only contains 1 multiplier, 1 inversion unit, 3 adders and a control logic. The control logic determines the

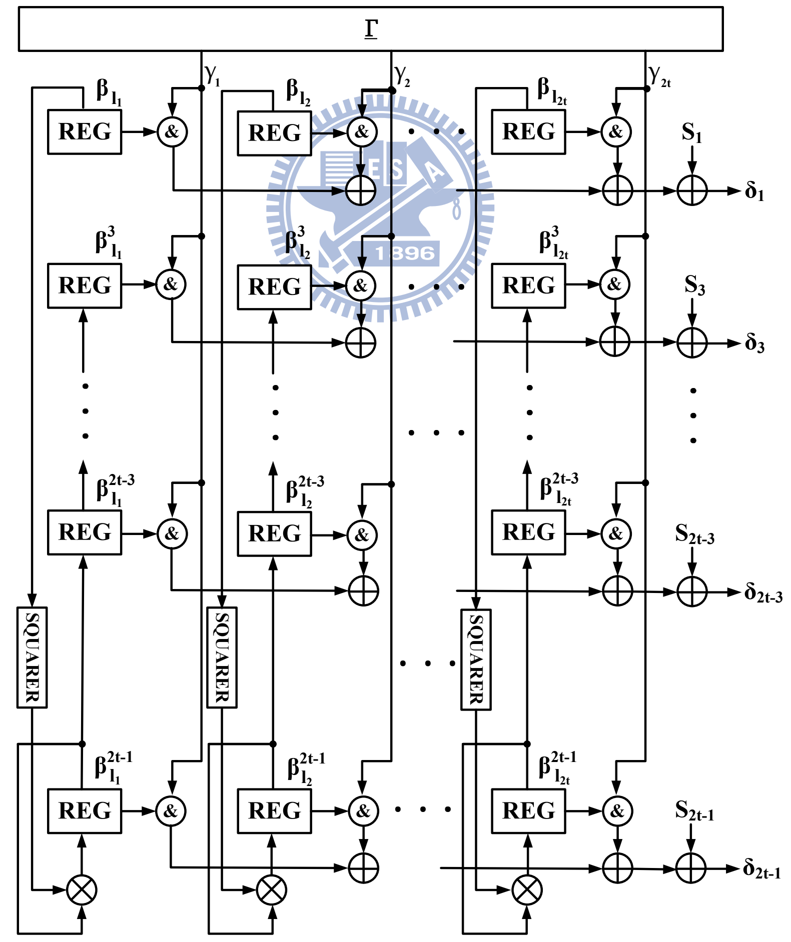


Figure 3.12: Heuristic error magnitude solver architecture

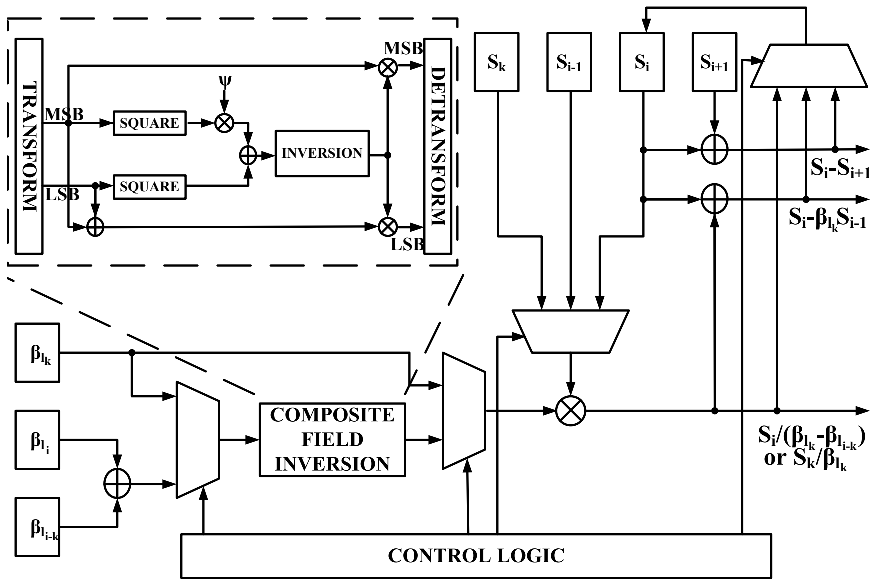


Figure 3.13: Björck-Pereyra error magnitude solver architecture.

computation order of the S_i and β_{l_i}, and the computation results will be used to update each S_i value. The inversion unit in the proposed architecture is carried out in composite field because the finite field inversion over $GF(2^m)$ is very complex with table-lookup implementation for large m.

Composite field [62] is viewed as an extension field of $GF(2^k)$ while given $m = kr$. The finite field $GF(2^m)$ can be constructed by coefficients from the subfield $GF(2^k)$. Operating in subfield leads to lower implementation complexity and better computation efficiency. For example, every element in $GF(2^{16})$ can be represented by $bx + c$ and inversion of $bx + c$ can be derived as (3.11) with the polynomial $x^2 + x + \psi$ [62]

$$\frac{1}{bx+c} = (b^2\psi + bc + c^2)^{-1}(bx + b + c). \tag{3.11}$$

The composite field inversion units over $GF(2^{14})$ and $GF(2^{16})$ are only 1.1K and 2.1K gate-count in CMOS 90nm technology respectively while the inversion units using Look Up Table method are about 41.5K and 186K gate-count.

Fig. 3.14 illustrates a multi-mode BP-EMS architecture to support 21 modes in DVB-S2 system, including normal and short frames. Except an additional inversion unit for $GF(2^{14})$,

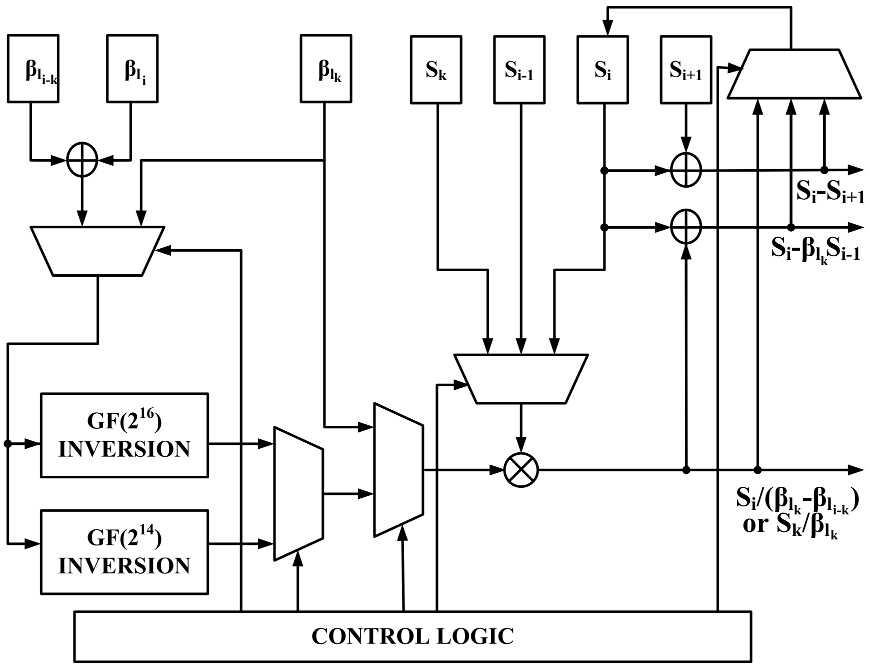


Figure 3.14: Multi-mode Björck-Pereyra error magnitude solver architecture for DVB-S2 system.

most components can be shared with a single mode design, leading to only few control logics requirement. Two composite field inversion units are provided for supporting $GF(2^{14})$ and $GF(2^{16})$ operations. Also, the partial product and modular operation in the multiplier design over $GF(2^{14})$ and $GF(2^{16})$ are shared such that one reconfigurable multiplier can be used in both normal and short frames.

For the high performance EM-type soft BCH decoders, CEMS is proposed. Based on CEMS algorithm, Fig. 3.15 illustrates the CEMS architecture to evaluate $\underline{\Delta_{odd}} = \mathbf{B_{odd}} \times \underline{\Gamma} + \underline{S_{odd}}$ with $\underline{S_{odd}}$ and $\underline{\mathcal{B}}$. The solid lines are data flow of the $\mathbf{B_{odd}}$ matrix construction procedure while the dash lines are the data flow of the geometrical progression check procedure. There are $2t^2$ registers for storing all entries in $\mathbf{B_{odd}}$ matrix. In the i-th column, the initial value of the first row register is set as β_{l_i} so that the output of the squarer will always be $\beta_{l_i}^2$. The t-th row register is also initially set as β_{l_i} and iteratively multiplied by $\beta_{l_i}^2$ to generate $\beta_{l_i}^{2j+1}$ for operating cycles $j = 1 \sim (t-1)$. Consequently, the register values of the i-th column form a geometric progression with the common ratio $\beta_{l_i}^2$ after $t-1$ cycles.

The $\mathbf{B_{odd}}$ matrix is calculated with totally only $2t$ multipliers and $2t$ squarers in $t-1$

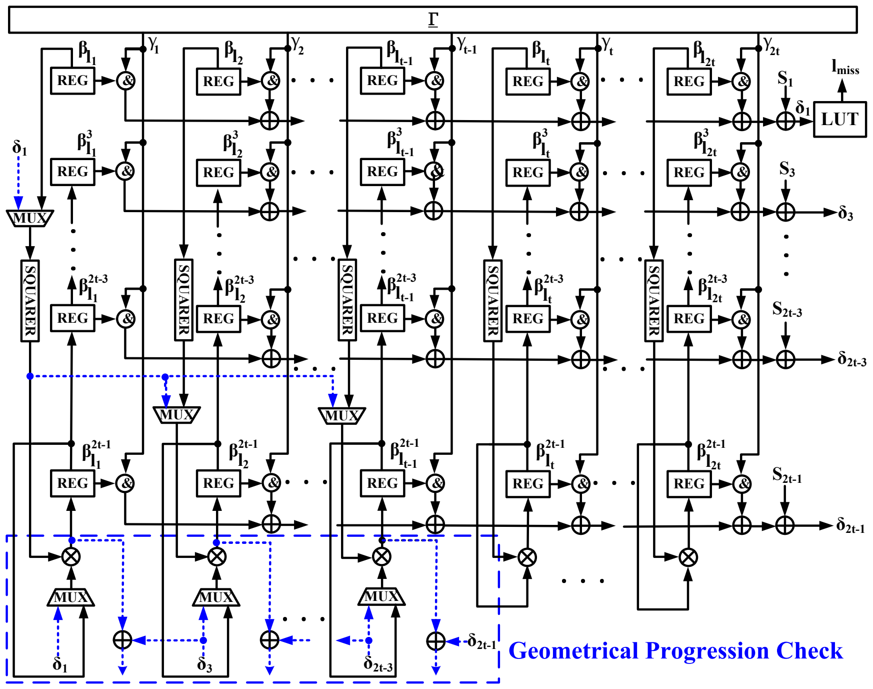


Figure 3.15: Compensation error magnitude solver.

cycles. These registers will hold their values in matrix multiplication procedure: $\underline{\Delta_{odd}} = \mathbf{B_{odd}} \times \underline{\Gamma} + \underline{S_{odd}}$.

Both matrix multiplication and geometrical progression check are evaluated simultaneously in the following 2^{2t} cycles. A heuristic search for all binary combinations is completed iteratively to count $\underline{\Gamma}$ value from 0 to 2^{2t-1}. At each iteration, the $\beta^{j}_{l_i}$ value stored in the register will be operated with γ_i to generate the modified discrepancy vector $\underline{\Delta_{odd}}$. Then the solver verifies whether $\underline{\Delta_{odd}}$ is a geometrical progression or not. In the geometrical progression check procedure, δ_1 passes through a squarer to generate δ_1^2, which is multiplied with each δ_i value for being compared with δ_{i+2}. If $\underline{\Delta_{odd}}$ is a geometrical progression, then $\delta_i \times \delta_1^2 = \delta_{i+2}$ for $i = 1, 3, \ldots, 2t-3$. The CEMS applies t-1 multipliers and 1 squarer to check this relation, and employs a look up table (LUT) to obtain l_{miss} according to $\delta_1 = \alpha^{l_{miss}}$. However, the geometrical progression check is processed after the $\mathbf{B_{odd}}$ matrix construction, the squarer and multipliers can be shared, leading to totally only $2t$ multipliers and $2t$ squarers in the proposed CEMS architecture. The critical path of the matrix multiplication procedure is $(T_{and} + 2t \times T_{xor})$ for generating $\underline{\Delta_{odd}}$ while that of the geometrical progression

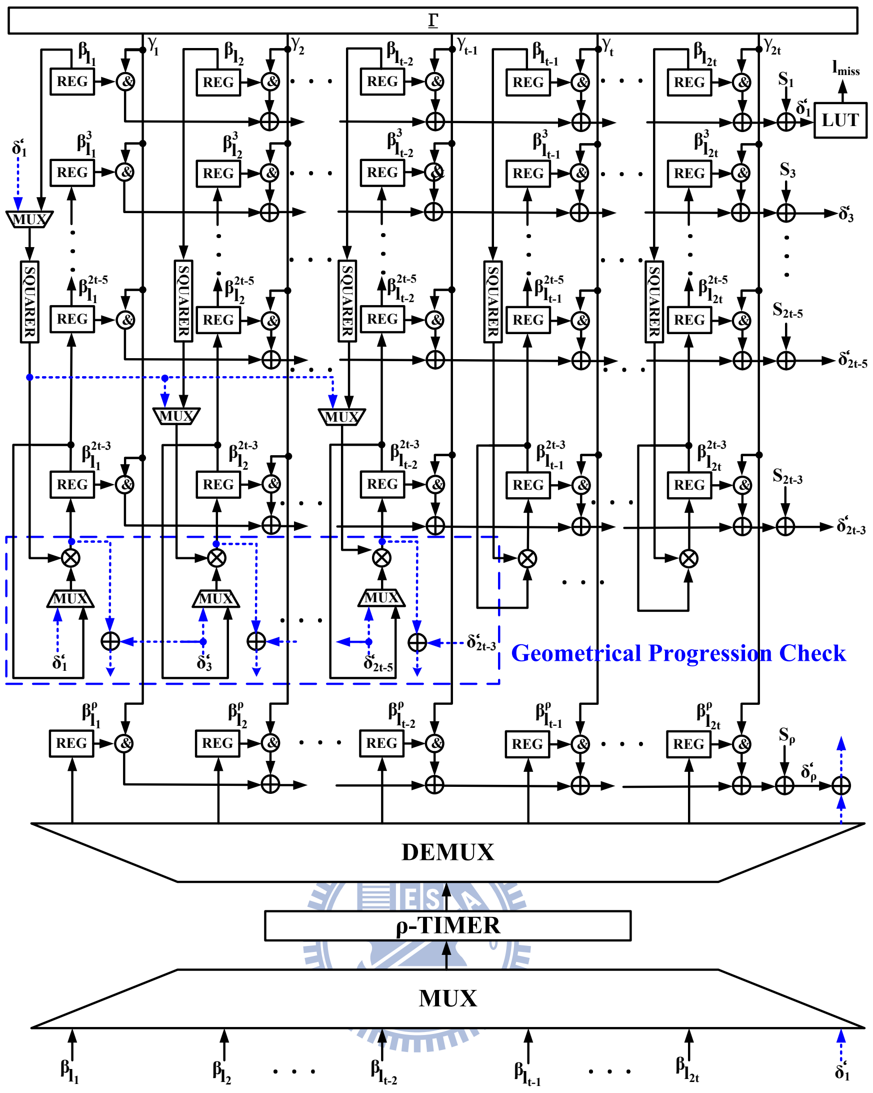


Figure 3.16: Modified compensation error magnitude solver.

check procedure is $(T_{xor}+2T_{mux}+T_{sq}+T_{mult})$ for using δ_1 to verify the relation $\delta_i \times \delta_1^2 = \delta_{i+2}$ with $i = 1, 3, \ldots, 2t-3$. Notice that $T_{and}, T_{xor}, T_{mux}, T_{sq}$, and T_{mult} represent the critical path of AND gate, XOR gate, multiplexer, squarer, and multiplier respectively. Consequently, the critical path of CEMS is $(T_{and} + (2t+1) \times T_{xor} + 2 \times T_{mux} + T_{sq} + T_{mult})$.

In the last place, M-CEMS is proposed or the high performance EM-type soft BCH-Like decoders. With $\underline{S'_{odd}}$ and $\underline{\mathcal{B}}$, the M-CEMS architecture shown in Fig. 3.16 is employed to evaluate $\underline{\Delta'_{odd}} = \mathbf{B'_{odd}} \times \underline{\Gamma} + \underline{S'_{odd}}$ based on algorithm B. The solid lines are data flow of the $\mathbf{B'_{odd}}$ matrix construction procedure while the dash lines are the data flow of the geometrical progression check procedure. The major difference between CEMS and M-CEMS is the requirement of the ρ-TIMER, which makes values become ρ times. The architecture design of a ρ-TIMER depends on the coefficient ρ. For a low complexity design, a ρ-TIMER can be decomposed into several smaller TIMERs with corresponding ρ as power of 2, which have similar hardware complexity as a constant multiplier under Galois field operations. For example, a 85-TIMER can be constructed with a 4-TIMER and a 16-TIMER according to 85 = (1+4)*(1+16). In the matrix construction procedure, the $(t-1)$-th row register in the i-th column replaces the t-th row register to generate $\beta_{l_i}^{2j+1}$ for operating cycles $j = 1 \sim (t-2)$. Nevertheless, the $2t$ register values in the t-th row are ρ times that in the 1st row and requires $2t$ cycles to be computed with only one ρ-TIMER. Hence, $\mathbf{B}'_{\mathbf{odd}}$ matrix, in total, is calculated with $2t$ multipliers, $2t$ squarers and a ρ-TIMER in $2t$ cycles.

The matrix multiplication and geometrical progression check procedures are evaluated in the following 2^{2t} cycles. A heuristic search of all binary combinations of $\underline{\Gamma}$ is completed. The geometrical progression check block verifies the relations that are $\delta'_i \times \delta'^2_1 = \delta'_{i+2}$ for $i = 1, 3, \ldots, 2t-5$ and $\delta'^{\rho}_1 = \delta'_\rho$. The hardware between $\mathbf{B}'_{\mathbf{odd}}$ matrix construction and the geometrical progression check can be shared since they operate separately. As a result, in total, the M-CMES utilize $2t$ multipliers, $2t$ squarers and a ρ-TIMER. The critical path of the matrix multiplication procedure is $(T_{and} + 2t \times T_{xor})$ for generating $\underline{\Delta'_{odd}}$ while that of the geometrical progression check procedure is $(T_{xor} + (2log_2(2t+1)) \times T_{mux} + T_{\rho-timer})$ for using δ'_1 to verify the relation $\delta'^{\rho}_1 = \delta'_\rho$. Notice that the critical path of one $(2t+1)$-to-1 multiplexer and one 1-to-$(2t+1)$ demultiplexer is assumed as $(2log_2(2t+1)) \times T_{mux}$ and the critical path of ρ-TIMER is represented as $T_{\rho-timer}$. Consequently, the critical path of M-CEMS is $(T_{and} + (2t+1) \times T_{xor} + (2log_2(2t+1)) \times T_{mux} + T_{\rho-timer})$.

3.4.4 Architecture Comparison

For the low complexity EM-type soft BCH decoders, the architectures of a hard BCH decoder and two proposed soft BCH decoders are compared in TABLE 3.3. The two proposed soft BCH decoders are designed with H-EMS and BP-EMS respectively while hard BCH decoder is designed with inversionless Berlekamp-Massey (iBM) algorithm [63]. As compared with

Table 3.3: Comparison Table for an (N, K, t) BCH Code

	Hard BCH (iBM)	Soft BCH * (H-EMS)	Soft BCH ** (BP-EMS)
Register	$5t + 2$	$2t^2 + 5t$	$8t$
Multiplier	$3t + 3$	$2t$	1
Constant Multiplier	$3t$	$t + 1$	$2t + 1$
Squarer	0	$2t$	0
Inversion Unit	0	0	1
Latency	$2N + 2t$	$N + 2^{2t} + t - 1$	$N + 6t^2 - t$

* In the special case : t = 1, the number of multipliers and squarers is 0.
If t is very small, like 1 or 2, we can check all combinations of γ_{ci} over $GF(2)$ at one cycle.

** Registers can be inserted into composite field inversion to reduce the critical path with the doubled latency in EMS step.

the soft BCH decoder with BP-EMS, only half syndromes are required for soft BCH decoder with H-EMS. In H-EMS, $2t$ multipliers and $2t$ squarers are used to construct the $\mathbf{B_{odd}}$. Notice that, if the error correcting capabilities is equal to 1, the number of multipliers and squarers is 0 because only S_1 will be computed. The first row registers in the H-EMS can be shared with registers of the error locator part in error locator evaluator so that totally $2t^2 - 2t$ registers are used in this part. In BP-EMS, only 1 multiplier and 1 inversion unit are employed to evaluate $\underline{\Gamma}$. Both the soft decoders take n clock cycles for syndrome calculator and error locator evaluator simultaneously and H-EMS and BP-EMS take $2^{2t} + t - 1$ and $6t^2 - t$ clock cycles respectively.

TABLE 3.4 illustrates the number of each component at $t = 1$, 2, and 12. Notice that the synthesis results in CMOS 90nm technology shows that the complexity ratio over $GF(2^{16})$ among 16-bit register, multiplier, constant multiplier, squarer and inversion unit is 1 : 10 : 3 : 1.5 : 25. With this normalized ratio, Fig. 3.17 shows the normalized complexity analysis among hard and soft decoders. In large finite field operations, a multiplier is much more complicated than a register. Due to fewer number of multipliers, the proposed soft BCH decoders with more registers have much lower hardware complexity as compared to the hard BCH decoder with iBM algorithm under different error correcting capabilities t. Because of non-linear increment of the number of registers, the complexity of soft BCH decoder with

Table 3.4: Comparison Table under Different Correct Ability

	Hard BCH (iBM) t = 1	Soft BCH (H-EMS) t = 1	Soft BCH (BP-EMS) t = 1	Hard BCH (iBM) t = 2	Soft BCH (H-EMS) t = 2	Soft BCH (BP-EMS) t = 2	Hard BCH (iBM) t = 12	Soft BCH (H-EMS) t = 12	Soft BCH (BP-EMS) t = 12
Register	7	7	8	12	18	16	62	348	96
Multiplier	6	0	1	9	4	1	39	24	1
Constant Multiplier	3	2	3	6	3	5	36	13	25
Squarer	0	0	0	0	4	0	0	24	0
Inversion Unit	0	0	1	0	0	1	0	0	1
Normalized Complexity * (number of register)	76	13	52	120	73	66	560	663	206
Latency	2N+2	N+4	N+5	2N+4	N+17	N+22	2N+24	$N + 2^{24} + 11$	N+852

* According to the synthesis results in CMOS 90nm technology, the complexity ratio over $GF(2^{16})$ among Register, Multiplier, Constant Multiplier, Squarer and Inversion Unit is 1 : 10 : 3 : 1.5 : 25

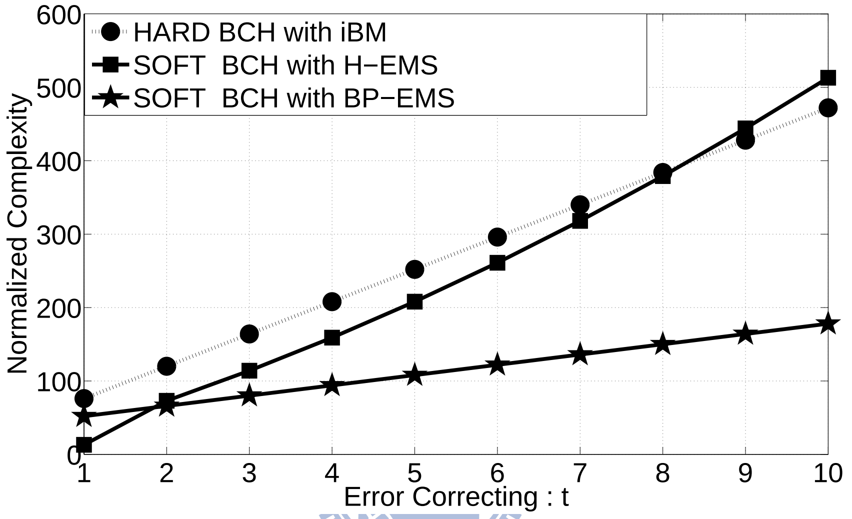


Figure 3.17: Normalized hardware complexity analysis of BCH decoders over $GF(2^{16})$.

H-EMS is less than that of hard BCH decoder only if t is smaller than 8 as shown in Fig. 3.17. H-EMS is more suitable to be applied in small t application, like BCH (762, 752, 1) decoder defined in DMB-T system. In addition, the soft BCH decoder with BP-EMS can always provide lower complexity than hard BCH decoder. In this paper, we applied BP-EMS in the proposed design according to the high error correcting capability requirement. The proposed soft decoders, searching error locations at error locator evaluator procedure, lead to a lot of latency saving. Consequentially, the proposed soft decoders provide both higher throughput and much lower hardware complexity.

For further improvement on latency, H-EMS could complete all the computations in one cycle with less hardware overhead for small t. In addition, BP-EMS could insert registers into composite field inversion for operation frequency improvement. Although the latency

Table 3.5: Comparison Table for an $(N, K; t)$ BCH Code

	$(N, K; t)$ Hard BCH with iBM	$(N, K; t)$ Hard BCH with SiBM	$(N, K; t)$ Soft BCH with CEMS	$(N, K; t)$ Soft BCH-Like with M-CEMS	(255, 239; 2) Hard BCH with iBM	(255, 239; 2) Hard BCH with SiBM	(255, 239; 2) Soft BCH with CEMS	(255, 243; 2) Soft BCH-Like with M-CEMS
Register	$5t+2$	$7t+2$	$2t^2+5t$	$2t^2+5t$	12	16	18	18
Mux	$t+1$	$2t$	$8t-2$	$10t-2$	3	4	14	18
Mult	$3t+3$	$4t$	$2t$	$2t$ *	9	8	4	0
Constant Mult	$3t$	$3t$	$t+1$	$t+1$	6	6	3	3
Squarer	0	0	$2t$	$2t$ *	0	0	4	0
ρ-Timer	0	0	0	1	0	0	0	1
LUT	0	0	1	1	0	0	1	1
RAM (bit)	N	N	N	N	255	255	255	255
Latency	$2N+2t$	$2N+t$	$N+2^{2t}+t-1$	$N+2^{2t}+2t$	514	512	272	275
Normalized ** Complexity	$54t+42$	$72t+5$	$5t^2+49t$ $+26.5$	$5t^2+51t$ $+56.5$*	150	149	144.5	133.5

* In the special case: $t = 2$, the number of multiplier and squarer is 0.

** The normalized complexity is in terms of number of 8-bit 2-to-1 multiplexer. According to the synthesis results in CMOS 90 nm technology, the complexity ratio over $GF(2^8)$ among a 8-bit 2-to-1 multiplexer, a squarer, a constant multiplier, a 8-bit register, a multiplier, a LUT and a 51-TIMER is 1 : 1.5 : 1.5 : 2.5 : 12 : 27 : 30.

in EMS step will be doubled, it is only few percentage of overall decoding procedure for long block length BCH decoders, resulting in that throughputs of the soft BCH decoder can be nearly doubled.

For the high performance EM-type soft BCH decoders, the architectures of hard BCH decoders and the proposed soft decoders are compared in TABLE 3.5. The proposed soft BCH and BCH-Like decoders are designed with the CEMS approach and the M-CEMS approach respectively whereas the hard BCH decoders are designed with inversionless Berlekamp-Massey (iBM) algorithm [63] and simplified iBM (SiBM) algorithm [64].

Both the soft BCH and BCH-Like decoders are designed with $2t$ multipliers, $2t$ squarers and 1 LUT. Nevertheless, an additional ρ-TIMER is applied in the M-CEMS. The registers in the first row of $\mathbf{B_{odd}}$ and $\mathbf{B'_{odd}}$ matrice in the CMES and the M-CMES are applied to store the error locator set $\underline{\mathcal{B}}$, which is also stored in the registers of the error locator evaluator. Therefore, these registers can be shared, resulting in totally $2t^2 - 2t$ registers used in the CEMS and the M-CEMS. Note that in the special case: $t = 2$, the M-CEMS can be constructed without any multiplier and squarer since there are only two syndrome values: S_1 and S_ρ. In addition, the syndrome calculator and the error locator evaluator take N clock cycles simultaneously in the decoding process and the CEMS and the M-CEMS take $2^{2t} + t - 1$ and $2^{2t} + 2t$ clock cycles respectively.

In finite field operations, a multiplier is more complex than a register and a multiplexer.

Due to fewer multipliers, the proposed soft BCH decoder with more registers and multiplexers as well as an additional LUT has similar hardware complexity while compared to the hard BCH decoders with iBM and SiBM algorithms. The proposed soft BCH-Like decoder has a little more hardware complexity due to an extra TIMER, which contains 2 or more multipliers dependent on the coefficient ρ. According to the synthesis results in CMOS 90 nm technology, the complexity ratio over $GF(2^8)$ among each 8-bit 2-to-1 multiplexer, squarer, constant multiplier, 8-bit register, multiplier, LUT and 51-TIMER is $1 : 1.5 : 1.5 : 2.5 : 12 : 27 : 30$. The normalized complexity of the soft BCH and BCH-Like decoders is around $(5t^2 + 49t + 26.5)$ and $(5t^2 + 51t + 56.5)$ 8-bit 2-to-1 multiplexers while that of the hard BCH decoder with iBM/SiBM algorithms is $(54t + 42)/(72t + 5)$ 8-bit 2-to-1 multiplexers respectively. For the high code rate BCH and BCH-Like codes, the error correcting ability t is small, implying that the proposed soft decoders can provide similar hardware complexity as hard decoders even though the complexity of hard and soft decoders is linear and quadratic to t respectively. For example, the normalized complexity of the soft BCH (255, 239; 2) and BCH-Like (255, 243; 2) decoders is around 144.5 and 133.5 8-bit 2-to-1 multiplexers while that of the hard BCH (255, 239; 2) decoder with iBM/SiBM algorithms is 150/149 8-bit 2-to-1 multiplexers respectively.Moreover, the proposed soft decoders have only 53% latency compared to that from the tranditional hard BCH decoders by searching for error locations at error locator evaluator procedure.

Chapter 4

Chase Type Soft Decoders

Since the EM-type methods only can provide better error correcting performance while applied in small error correcting codes or with reliable information, this chapter discusses the Chase-type soft decoding algorithms to provide more general low complexity decoding methods. Recall the Chased algorithm we have been introduced in Section 2.4.2. A list of 2^η candidate sequences are formed according to η LRPs. After the error only hard decoders solving each candidate sequence, a list of 2^η candidate codewords are generated. Finally, a most likely codeword will be selected from the list by the decision making unit according to the Euclidean distance calculations between the received input values and each candidate codeword. For the practical implementation, the decision making unit is too complex because evaluating the Euclidean distance between the received data and a candidate codeword requires N multiplication and 1 square root operations.

In this chapter, two Chase-type soft decoding algorithms are presented for both BCH and RS codes. Notice that, the major difference between BCH codes decoding and RS codes decoding is that the error value evaluation is required in the latter one. For the convenience of discussion, the following sections describe these algorithms with RS codes. In Section 4.1, instead of utilizing 2^η hard decoders to decode all candidate sequences simultaneously, a decision-eased soft decoding scheme is provided to process Chase algorithm with one hard decoder module, and to determine the most likely codeword with a simplified decision making unit. In order to eliminate the decision making unit, a decision-confined soft decoding algorithm is proposed in Section 4.2 by confining the degree of error location polynomial generated from the key equation solver. At the end of Chapter 4, Section 4.3 describes the Chase-type decoder designs.

4.1 Decision-Eased Approach

In the Chase algorithm, the final step is exploited a decision making unit to select the candidate codeword with the smallest Euclidean distance as the decoded codeword, where the Euclidean distance between the received data and i-th candidate codeword $y^{(i)}$ is calculated as

$$d^{(i)} = \sqrt{(y_0^{(i)} - r_0)^2 + (y_1^{(i)} - r_1)^2 + \cdots (y_{N-1}^{(i)} - r_{N-1})^2}. \tag{4.1}$$

For the long length codeword, however, the Euclidean distance calculation requires large number of multiplication and square root calculations, which are too complex for practical implementation. In this section, a decision-eased soft decoding algorithm is proposed to ease the computations of decision making procedure, leading to hardware reduction.

Figure 4.1 shows the proposed soft decision-eased decoder that includes seven major steps: *syndrome calculator, reliability evaluator, syndrome updater, key equation solver, Chien search, error value evaluator*, and *simplified decision making unit*. The received data is fed into the syndrome calculator for calculating the syndrome polynomial $S(x)$. In the meantime, the reliability evaluator determines η least reliable positions (LRPs) $\underline{L} = [l_1, l_2, \ldots, l_\eta]$ according to the received data. The corresponding error values: $\underline{E'} = [e'_1, e'_2, \ldots, e'_\eta]$ are also computed. Instead of evaluating of 2^η syndrome polynomials with (2.36) for 2^η candidate sequences, a syndrome updater is applied to obtain the i-th syndrome polynomial $S^{(i)}(x)$ from the syndrome polynomial $S(x)$. There are only η uncommon points for all the candidate sequences; therefore, the i-th candidate sequence can be constructed by adding the error pattern $e_f^{(i)}(x)$ induced by the bit flipping procedure to the received data. The $e_f^{(i)}(x)$ is of the form:

$$e_f^{(i)}(x) = a_1^{(i)} e_1' x^{l_1} + a_2^{(i)} e_2' x^{l_2} + \cdots + a_\eta^{(i)} e_\eta' x^{l_\eta}, \tag{4.2}$$

where $a_j^{(i)} \in GF(2)$ for $i = 0 \sim 2^\eta - 1$ and $j = 1 \sim \eta$. The i-th syndrome polynomial can be derived as

$$S^{(i)}(x) = \sum_{j=1}^{2t} (S_j + e_f^{(i)}(\alpha^j)) x^{j-1}. \tag{4.3}$$

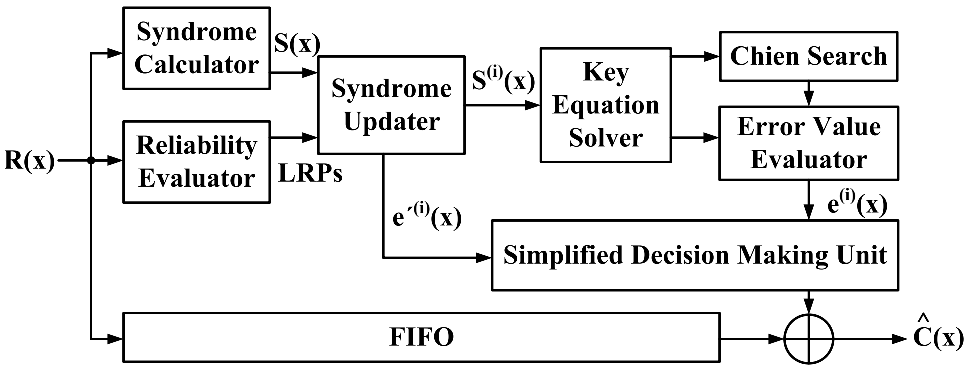


Figure 4.1: Decision-eased Chase-type soft decoding process.

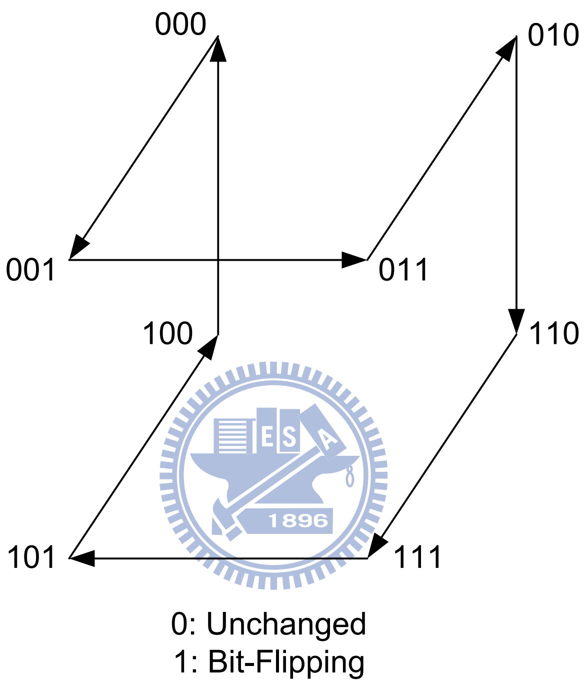


Figure 4.2: Gray code permutation.

To further improve the complexity, $S^{(i)}(x)$ can be obtained from the previous syndrome polynomial $S^{(i-1)}(x)$. With gray code based bit flipping order as shown in Fig. 4.2, there is only one different bit between successive two candidate sequences. For example, if the LRPs are at 3rd, 50th and 67th bits of the received data, then a sequence "011" in the Fig. 4.2 means to flip the 50th and 67th bits and maintain to the 3rd bit. In the general case, both the 50th and 67th bits will be flipped; however, only the 50th bit is flipped in this design since sequence "011" is permuted after sequence "001", indicating that the 67th bit has been

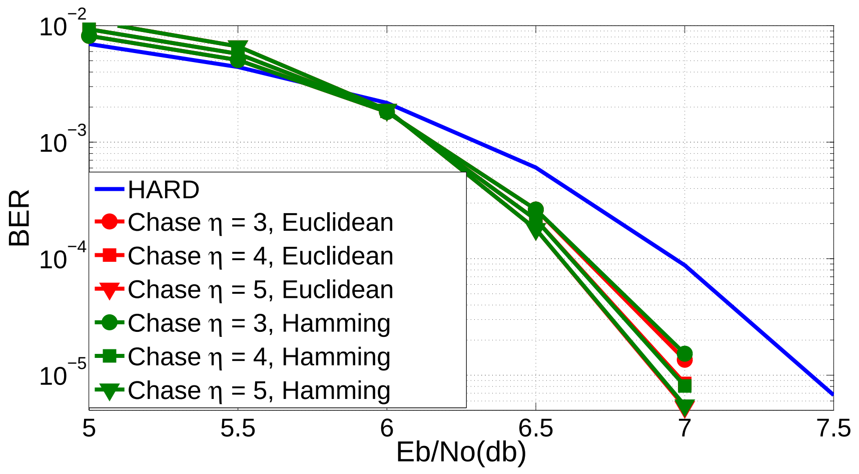


Figure 4.3: Simulation results of decision-eased algorithm for RS (224, 216; 4) codes.

flipped. Hence, $S^{(i)}(x)$ can be simplified as

$$S^{(i)}(x) = \sum_{j=1}^{2t} (S_j^{(i-1)} + e_f'^{(i)}(\alpha^j))x^{j-1}, \tag{4.4}$$

where $e_f'^{(i)}(x) = e_\kappa' x^{l_\kappa}$ is difference polynomial between $e_f^{(i-1)}$ and $e_f^{(i)}(x)$ and the only one different bit is at l_κ-th position. There is only one bit required to be flipped while the decoder generates one candidate codeword, leading to the reduction of the multipliers. Accordingly, the computations of the syndrome updater will be reduced.

After 2^η syndrome polynomials are generated from syndrome updater, the key equation solver, Chien search and error value evaluator are applied to form 2^η candidate codewords. Finally, the decision making unit selects the most possible one from the candidate list as decoded codeword. In the decision making unit, however, all the Euclidean distances are applied only for choosing the most likely codeword from the candidate list. Their exact values are not demanded as long as their relations are still held, implying that the square root calculation can be eliminated. Moreover, it can be found in Fig. 4.3 that the Chase algorithm with Hamming distance calculation almost has no performance loss as compared to the Chase algorithm with Euclidean distance calculation for RS (224, 216; 4) code with η LRPs, where $\eta = 3 \sim 5$. Hence, the Hamming distance calculation is utilized in the proposed decoder for replacing the Euclidean distance calculation due to its much simpler computations. In addition, the i-th candidate codeword $\hat{C}^{(i)}(x)$ in the Chase algorithm can be viewed as

$$\hat{C}^{(i)}(x) = R(x) + e^{(i)}(x) + e_f^{(i)}(x), \tag{4.5}$$

where $e^{(i)}(x)$ is the i-th estimated error pattern by decoding $R(x)+e_f^{(i)}(x)$. Notice that from (4.5),

$$\hat{C}^{(i)}(x) + R(x) = e^{(i)}(x) + e_f^{(i)}(x). \tag{4.6}$$

Therefore, we can computing the weight of $e^{(i)}(x) + e_f^{(i)}(x)$ instead of storing each candidate codeword and directly calculating the Hamming distance between $\hat{C}^{(i)}(x)$ and $R(x)$, leading to significantly reduced memory requirement. Also, in the decoding process, only the smallest weight error pattern $e^{(\phi)}(x)+e_f^{(\phi)}(x)$ will be added with the received data $R(x)$ for generating the decoded codeword, where

$$\phi = argmin\{weight(e^{(i)}(x) + e_f^{(i)}(x)) : i = 0 \sim 2^{\eta} - 1\} \tag{4.7}$$

This indicates that we do not have to store all the $e^{(i)}(x) + e_f^{(i)}(x)$ with $i = 0 \sim 2^{\eta} - 1$. The details of the proposed decision-eased soft decoding algorithm is illustrated as follows:

Decision-Eased Chase-Type Algorithm

- Input:

  The received sequence $R(x)$ and the reliability of each bit.

- Initial conditions:

  $i = 0$, $\gamma = 0$, $\gamma' = 0$.

  $e'(x) = 0$, $e_{min}(x) = \infty$

- Step 1:

  Evaluate η LRPs: $\underline{L} = [l_1, l_2, \ldots, l_{\eta}]$.

  Evaluate the corresponding error values: $\underline{E'} = [e'_1, e'_2, \ldots, e'_{\eta}]$.

  Calculate syndrome polynomial $S(x)$.

```
• Step 2:
  $\gamma' = \gamma$
  $\gamma = i \oplus (i >> 1)$
  Find the only 1 different bit between $\gamma$ and $\gamma'$: the $\kappa$-th bit
  Evaluate the flipped error pattern: $e'(x) = e'(x) \oplus e'_{\kappa} x^{l_{\kappa}}$;

• Step 3:
  Update syndrome polynomial $S^{(i)}(x) = \sum_{j=1}^{2t} (S_j^{(i-1)} + e'_{\kappa}(\alpha^{j \times l_{\kappa}}))x^{j-1}$.

• Step 4:
  Calculate $\Lambda^{i}(x)$ from key equal solver with $S^{(i)}(x)$
  Evaluate error location and error value to obtain $e^{(i)}(x)$

• Step 5:
  $e''(x) = e'(x) \oplus e^{(i)}(x)$
  if (weight of $e''(x)$ < weight of $e_{min}(x)$)
    $e_{min}(x) = e''(x)$
  else
    $e_{min}(x) = e_{min}(x)$
  if( $i = 2^{\eta} - 1$)
    Go to Output
  else
    $i = i + 1$
    Go to Step 2

• Output:
  The decoded codeword $\hat{C}(x) = R(x) \oplus e_{min}(x)$.
```

The η-bit γ and γ' is the gray code representation of the values i and $i-1$ respectively. At each time, there will be only 1 different bit between γ and γ': κ-th bit, implying that the flipped error pattern $e'(x)$ can be calculated by adding the previous flipped error pattern with the error patten caused by the κ-th bit: $e'_\kappa x^{l_\kappa}$. Then the estimated error pattern $e^{(i)}(x)$ can be obtained by solving the updated syndrome polynomial $S^{(i)}(x)$. According to (4.6), the Hamming distance between received data and candidate codeword is the same as the weight of the summation of flipped error pattern $e'(x)$ and estimated error pattern $e^{(i)}(x)$. Hence, Step 5 verifies whether the stored error pattern has smallest weight or not at each iteration. After 2^η iterations, the decoded codeword can be constructed with smallest weight error pattern: $\hat{C}(x) = R(x) \oplus e_{min}(x)$.

4.2 Decision-Confined Approach

Although the decision-eased method proposed in Section. 4.1 can decrease the complexity of decision making unit, it still have to decode 2^η the candidate sequence and select the most possible one from the candidate list. This section will discuss the soft decoding methods which can provide similar error correcting performance with fewer number of decoded candidate sequence and without a decision making unit for determining a most possible codeword.

To reduce the number of decoded candidate sequence, a decision-free soft decoding algorithm is proposed to decode without decision making unit. Instead of generating 2^η the candidate codewords and selecting one of them, the first successful decoded codeword is chosen as output codeword. Following steps illustrate the details of the proposed decision-free algorithm.

Decision-Free Chase-Type Algorithm

- Input:

  The received sequence $R(x)$ and the reliability of each bit.

- Initial conditions:

  $i = 0,\ \gamma = 0,\ \gamma' = 0.$

- Step 1:

  Evaluate η LRPs: $\underline{L} = [l_1, l_2, \ldots, l_\eta]$.

  Evaluate the corresponding error values: $\underline{E'} = [e'_1, e'_2, \ldots, e'_\eta]$.

  Calculate syndrome polynomial $S(x)$.

- Step 2:

  $\gamma' = \gamma$

  $\gamma = i \oplus (i >> 1)$

  Find the only 1 different bit between γ and γ': the κ-th bit

- Step 3:

  Update syndrome polynomial $S^{(i)}(x) = \sum_{j=1}^{2t} (S_j^{(i-1)} + e'_\kappa(\alpha^{j \times l_\kappa}))x^{j-1}$.

- Step 4:

  Calculate $\Lambda^{(i)}(x)$ from key equal solver with $S^{(i)}(x)$

- Step 5:

  Evaluate error location and error value to obtain $e^{(i)}(x)$

  ```
  If $(deg(\Lambda^{(i)}(x))$ = the number of roots found by the Chien search)
      Go to Output
  else if $(i = 2^\eta - 1)$
      Decoding Failed
  else
      $i = i + 1$
      Go to Step 2
  ```

- Output:

  The decoded codeword $\hat{C}(x) = R(x) \oplus e(i)(x)$.

Fig. 4.4 shows that the decision-free method almost has no performance loss with $\eta = 3$ and $\eta = 4$. for RS (255, 239; 8) codes. However, the Chien search and error value evaluator

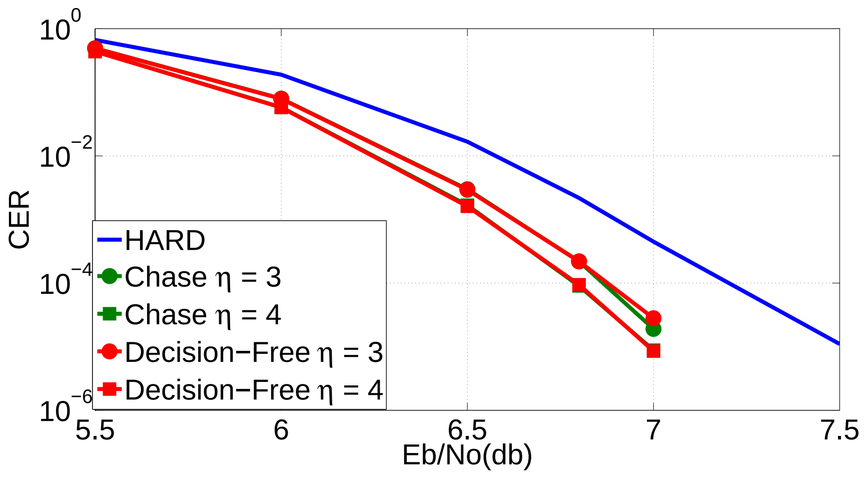


Figure 4.4: Simulation results of decision-free algorithm for RS (255, 239; 8) codes.

Table 4.1: Analysis of degree of $\Lambda(x)$ for RS (255, 239; 8) codes

Number of errors in $R(x)$	8	9	10	11	12	13	14
Percentage of $deg(\Lambda(x)) = 8$	99.5	99.5	99.5	99.5	99.5	99.5	99.5

have to been executed several times in decision-free method. The long decoding latency will still result in complexity architecture in a decoder design.

TABLE 4.1 shows the analysis of degree of $\Lambda(x)$ for RS (255, 239; 8) codes, where each value is generated under 10^5 codewords simulation. From our simulation results, there is over 99.5% probability for Chien search finding less than t roots when a received data has more than t errors. This indicates that $\Lambda(x)$ with degree t has more probability to result in failed decoding than $\Lambda(x)$ with other degree does. According to this characteristic, we propose a decision-confined algorithm to enhance the decoding efficiency by confining the degree of $\Lambda(x)$. The proposed algorithm only allows the $\Lambda(x)$ with degree less than t to be sent into Chien search and error value evaluator, which avoids that the decoder finishes the Chien search procedure but the decoding is failed. The details of the proposed decision-confined soft decoding algorithm is illustrated as follows:

Decision-Confined Chase-Type Algorithm

- Input:

  The received sequence $R(x)$ and the reliability of each bit.

```
• Initial conditions:
  $i = 1,\ \gamma = 0,\ \gamma' = 0.$

• Step 1:
  Evaluate $\eta$ LRPs: $\underline{L} = [l_1, l_2, \ldots, l_\eta]$.
  Evaluate the corresponding error values: $\underline{E'} = [e'_1, e'_2, \ldots, e'_\eta]$.
  Calculate syndrome polynomial $S(x)$.

• Step 2:
  $\gamma' = \gamma$
  $\gamma = i \oplus (i >> 1)$
  Find the only 1 different bit between $\gamma$ and $\gamma'$: the $\kappa$-th bit

• Step 3:
  Update syndrome polynomial $S^{(i)}(x) = \sum_{j=1}^{2t} (S_j^{(i-1)} + e'_\kappa(\alpha^{j \times l_\kappa}))x^{j-1}$.

• Step 4:
  Calculate $\Lambda^{(i)}(x)$ from key equal solver with $S^{(i)}(x)$
  If $(deg(\Lambda^{(i)}(x)) < t)$
     Go to Step 5
  else if $(i = 2^\eta - 1$ and $deg(\Lambda^{(i)}(x)) = t)$
     Go to Step 6
  else
     $i = i + 1$
     Go to Step 2

• Step 5:
  Calculate $\Lambda(x)$ from key equal solver with $S(x)$

• Step 6:
  Evaluate error location and error value to obtain $e(x)$
```

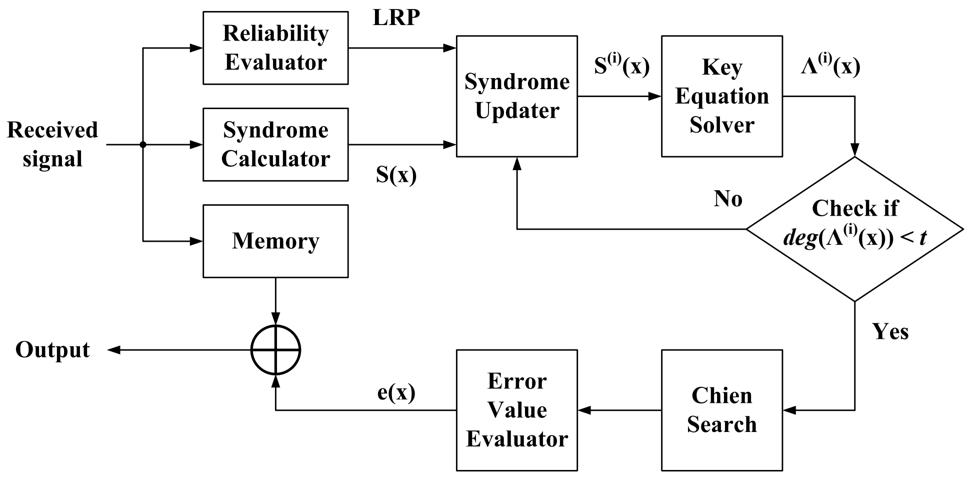


Figure 4.5: Decision-confined Chase-type soft decoding process

- Output:

  The decoded codeword $\hat{C}(x) = R(x) \oplus e(x)$.

In summary, the decoding procedure contains six major steps: *syndrome calculator, reliability evaluator, syndrome updater, key equation solver, Chien search* and *error value evaluator* as shown in Fig. 4.5. With the received soft information, the syndrome calculator calculates the syndrome polynomial $S(x)$. In the meantime, the reliability evaluator determines η LRPs $\underline{L} = [l_1, l_2, \ldots, l_\eta]$ and corresponding error values: $\underline{E'} = [e'_1, e'_2, \ldots, e'_\eta]$. The candidate sequences are generated according to Gray code based bit flipping method, leading to only one bit of these LRPs flipped between each two successive candidate. As a result, $S^{(i)}(x)$ for the i-th candidate sequence can be formed with the method in Step 2. Then the corresponding error location polynomial $\Lambda^{(i)}(x)$ is calculated by the key equation solver. Only if the degree of $\Lambda^{(i)}(x)$ is less than t, the Chien search and error value evaluator start to evaluate the error locations and error values. Otherwise, the decoder will generate a new syndrome polynomial with another bit flipping sequence. Notice that, if all the generated error location polynomials have degree of t, the proposed decoder will decode received data as a hard RS decoder to guarantee the error correction capability .

Fig. 4.6 shows the RS (255, 239; 8) simulation results for our proposed decision-confined algorithm with $\eta = 3 \sim 5$ under BPSK modulation and AWGN channel. Our proposal with $\eta = 5$ can provide 0.4 dB coding gain at 10^{-4} codeword error rate (CER) over the hard RS

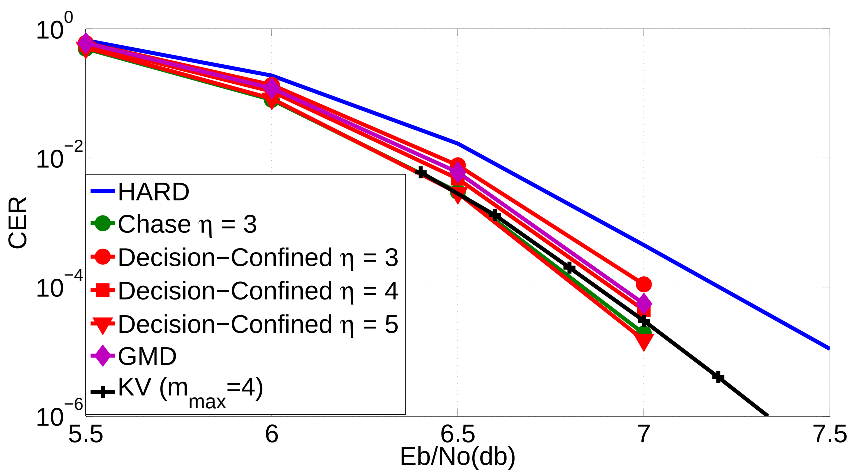


Figure 4.6: Simulation results of decision-confined algorithm for RS (255, 239; 8) codes.

Table 4.2: Average computation times for RS (255, 239; 8) codes

	Decision-Confined Algorithm $\eta = 5$			Chase Algorithm $\eta = 3$		
E_b/N_0	6.0	6.5	7.0	6.0	6.5	7.0
Syndrom Calculator	5.22	1.30	1.07	8	8	8
Key Equation Solver	5.22	1.30	1.07	8	8	8
Chien Search	1	1	1	8	8	8
Error Value Evaluator	1	1	1	8	8	8

decoder, and also outperforms than GMD and KV algorithms. As compared with Chase algorithm with $\eta = 3$, our proposed method can achieve similar error correcting ability with $\eta = 5$. Notice that, the average computation complexity of our proposal is much less than Chase algorithm although it requires more LRPs to achieve similar error correcting performance. As shown in TABLE 4.2, if our approach with $\eta = 5$ is applied at $E_b/N_0 = 7$, the average computation of syndrome calculator, key equation solver, Chien search and error value evaluator are 1.07, 1.07, 1 and 1 times respectively. However, the Chase algorithm with $\eta = 3$ executes 2^3 calculation for all the decoding blocks.

4.3 Design of Chase-Type Soft Decoders

4.3.1 Reliability Evaluator

For RS (224, 216; 4) and (255, 239; 8) codes over $GF(2^8)$, a symbol consisting of 8 bit is fed into the reliability evaluator each cycle for choosing η LRPs. Hence, the reliability evaluator has to compare $8+\eta$ values and determine η least reliability values. To reduce the computation time as well as the critical path, the *merge sort concept* introduced in [65] is exploited for designing the reliability evaluator with 3-stage pipeline architecture as shown in Fig. 4.7, where the architecture of merge sorter is shown in Fig. 4.8. The first and second

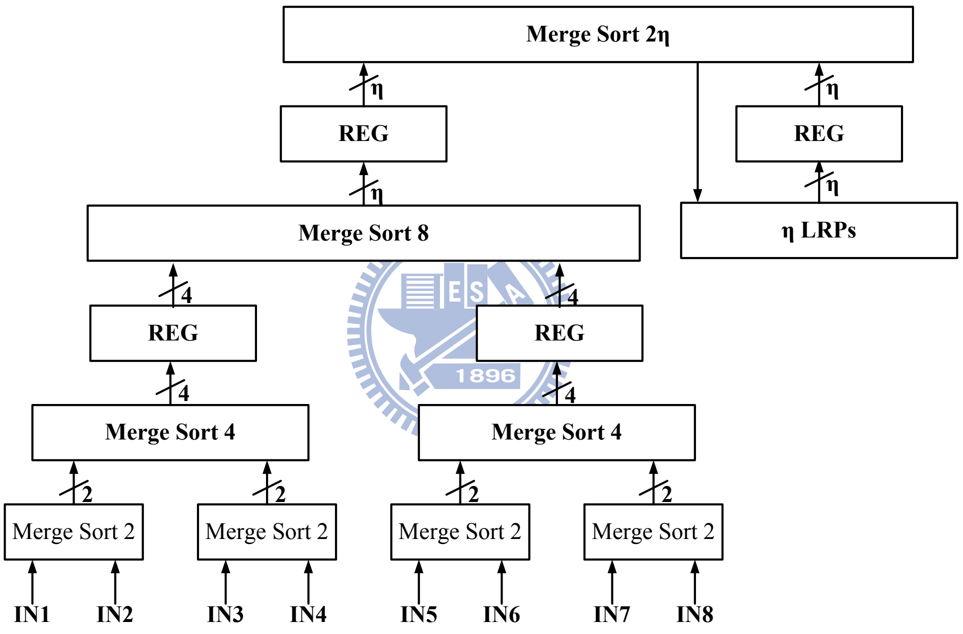


Figure 4.7: Reliability evaluator.

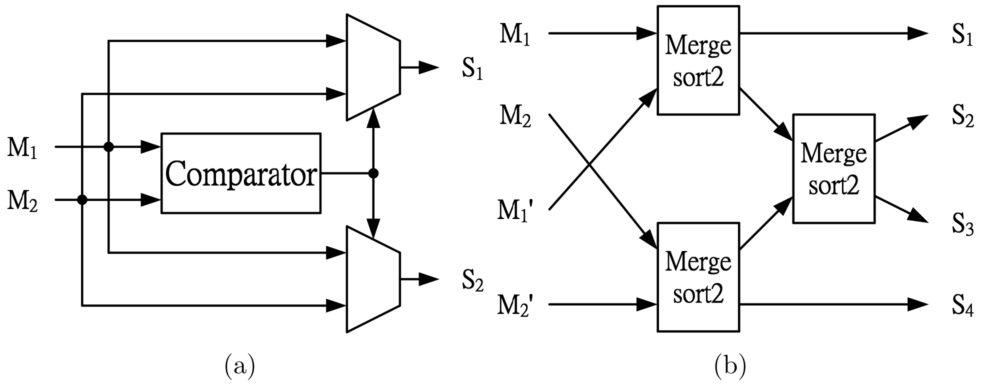


Figure 4.8: Merge Sorter. (a) Merge Sort2. (b) Merge Sort4.

stages calculate the η LRPs of the inputted 8 bits according to the *divide-and-conquer* concept. The third stage decides the new temporary LRPs among the output of the second stage and the temporary LRPs from previous calculation.

4.3.2 Syndrome Updater

Instead of recalculating the i-th syndrome polynomial $S^{(i)}(x)$ with syndrome calculator, the syndrome updater calculates it by updating the $(i-1)$-th syndrome polynomial $S^{(i-1)}(x)$, leading to further hardware reduction. According to (4.4), $S^{(i)}(x)$ can be obtained with $e'_\kappa \times x^{l_\kappa}$, where the only one different bit between successive two candidate sequences is at l_κ-th position. The proposed syndrome updater shown in Fig. 4.9 utilizes look up table 1 (LUT1) and LUT2 to find the e'_κ and α_{l_κ} based on the results of reliability evaluator. Note that the finite field multipliers (FFMs) and squarers for updating syndrome polynomials can be shared because there are N clock cycles for computing at most 2^η candidates sequences for each received data. For a RS (255, 239; 8) decoder, it only requires 4 FFMs and 2 squares for updating a syndrome polynomial.

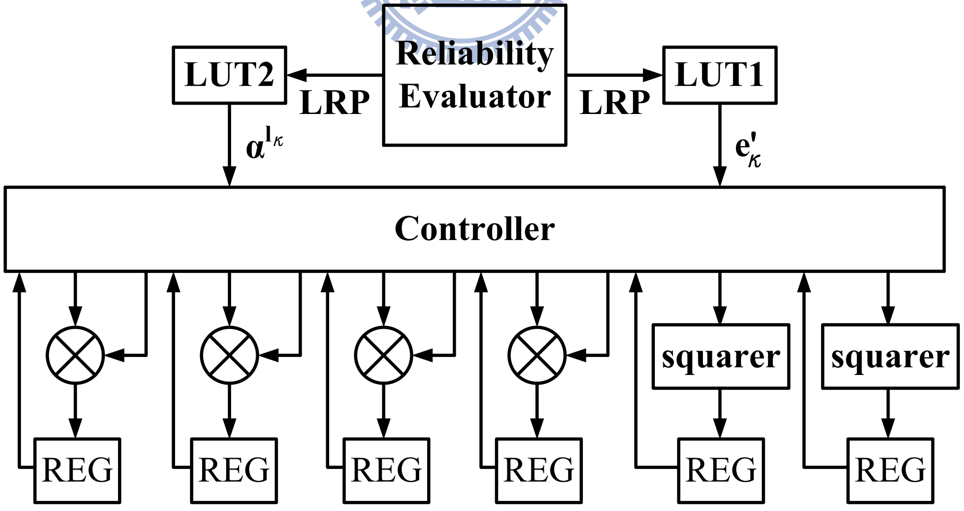


Figure 4.9: Syndrome updater.

4.3.3 Half Iteration Key Equation Solver

The Berlekamp-Massey (BM) algorithm is an iterative procedure to determine polynomials $\Lambda(x)$ and $\Omega(x)$. In 2000, Raghupathy and Liu found that the degree of $\Lambda(x)$, L, in the BM algorithm can be increased only once in any two successive iterations [66], implying that the computation cycle of key equation solver can be half-reduced. The discrepancy of the odd iteration Δ_{2k-1} and the discrepancy predicted for the even iteration ε_{2k} with $k = 1 \sim t$ are defined as:

$$\Delta_{2k-1} = \sum_{j=0}^{L_{k-1}} \Lambda_j^{(2k-2)} s_{2k-2-j} \tag{4.8}$$

$$\varepsilon_{2k} = \sum_{j=0}^{L_{k-1}} \Lambda_j^{(2k-2)} s_{2k-1-j} \tag{4.9}$$

The updating equation can be modified into five cases as shown in Fig. 4.10. Then $\Lambda(x)$ is calculated as

$$\Lambda_i^{(2k)} = \Lambda_i^{(2k-2)} + g_1 \Lambda_{i-1}^{(2k-2)} + g_2 B_{i-1}^{(2k-2)} + g_3 B_{i-2}^{(2k-2)} \tag{4.10}$$

where g_j $(j = 1 \sim 3)$ are the updating factors based on the different cases. As a result, $\Lambda(x)$ will be constructed within only t iterations.

In comparison with the RiBM algorithm [67], the architecture of [66] lacks for the irregularity and has a longer critical path delay. A half iteration RiBM (HI-RiBM) algorithm is proposed to improve the critical path as well as to providing regular construction. Notice

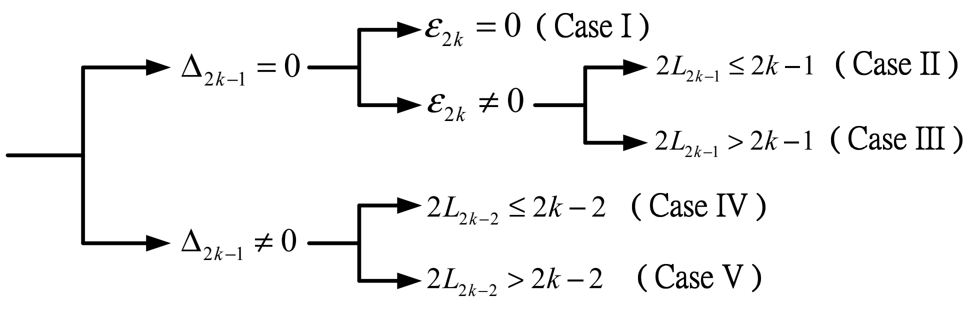

$\Delta_{2k-1} = 0$
$\varepsilon_{2k} = 0$ (Case I)
$\varepsilon_{2k} \neq 0$
$2L_{2k-1} \le 2k-1$ (Case II)
$2L_{2k-1} > 2k-1$ (Case III)
$\Delta_{2k-1} \neq 0$
$2L_{2k-2} \le 2k-2$ (Case IV)
$2L_{2k-2} > 2k-2$ (Case V)

Figure 4.10: The updating criterion for half iteration BM algorithm

that Δ_{2k-1} and ε_{2k} are exactly the coefficients of x^{2k-2} and x^{2k-1} in

$$\begin{aligned}\Delta(k,x) &= \Lambda(k,x)\cdot S(x)\\ &= \delta_0(k)+\delta_1(k)\cdot x+\cdots+\delta_k(k)\cdot x^k+\cdots\ .\end{aligned} \tag{4.11}$$

Therefore, we define $\hat{\delta}_i(k)=\delta_{i+2k-2}(k)$ and $\hat{\theta}_i(k)=\theta_{i+2k-2}(k)$ and the polynomial coefficients can be updated with the equation:

$$\begin{aligned}\hat{\delta}_i(k+1) &= \delta_{i+2k}(k+1)\\ &= g_0\cdot\delta_{i+2k}(k)+g_1\cdot\delta_{i+2k-1}(k)+g_2\cdot\theta_{i+2k-1}(k)+g_3\cdot\theta_{i+2k-2}(k)\\ &= g_0\cdot\hat{\delta}_{i+2}(k)+g_1\cdot\hat{\delta}_{i+1}(k)+g_2\cdot\hat{\theta}_{i+1}(k)+g_3\cdot\hat{\theta}_i(k).\end{aligned} \tag{4.12}$$

The details of the proposed half iteration RiBM algorithm is illustrated as follows:

Half iteration RiBM Algorithm

- Input:

  Syndrome polynomial: $S(x)$

- Initialization:

  $L_0=0,\ \alpha_0=S_0,\ c_0=1,\ \beta_0=S_1-S_0^2$

  $\hat{\delta}_i(0)=\hat{\theta}_i(0)=S_i,\ (i=1,...,2t-1)$

  $\hat{\delta}_{2t}(0)=\hat{\theta}_{2t}(0)=1$

- Iteration from $k=1$ to t:

  Step 1:

  Case 1: $\hat{\delta}_0(k-1)=0$ and $\hat{\delta}_1(k-1)=0$

  $g_0=c_{k-1},\ g_1=g_2=g_3=0$

  $\hat{\theta}_i(k)=\hat{\theta}_i(k-1),\ \hat{\theta}_{2t-2k}(k)=\hat{\theta}_{2t-2k+1}(k)=0$

  $L_k=L_{k-1},\ \alpha_k=\alpha_{k-1},\ c_k=c_{k-1}$

  Case 2: $\hat{\delta}_0(k-1)=0$ and $\hat{\delta}_1(k-1)\neq 0$ and $L_{k-1}>k-1$

$g_0 = c_{k-1},\ g_1 = g_2 = 0,\ g_3 = \hat{\delta}_1(k-1)$

$\hat{\theta}_i(k) = \hat{\theta}_i(k-1),\ \hat{\theta}_{2t-2k}(k) = \hat{\theta}_{2t-2k+1}(k) = 0$

$L_k = L_{k-1},\ \alpha_k = \alpha_{k-1},\ c_k = c_{k-1}$

Case 3: $\hat{\delta}_0(k-1) = 0$ and $\hat{\delta}_1(k-1) \neq 0$ and $L_{k-1} \leq k-1$

$g_0 = c_{k-1},\ g_1 = g_2 = 0,\ g_3 = \hat{\delta}_1(k-1)$

$\hat{\theta}_i(k) = \hat{\delta}_{i+2}(k-1)$

$L_k = 2k - L_{k-1},\ \alpha_k = \hat{\delta}_2(k-1),\ c_k = \hat{\delta}_1(k-1)$

Case 4: $\hat{\delta}_0(k-1) \neq 0$ and $L_{k-1} \leq k-1$

$g_0 = c_{k-1} \cdot \hat{\delta}_0(k-1),\ g_1 = \beta_{k-1},\ g_2 = \hat{\delta}_0^2(k-1),$

$g_3 = 0,\ \hat{\theta}_i(k) = \hat{\delta}_{i+1}(k-1),\ \hat{\theta}_{2t-2k}(k) = 0$

$L_k = 2k - 1 - L_{k-1},\ \alpha_k = \hat{\delta}_1(k-1),\ c_k = \hat{\delta}_0(k-1)$

Case 5: $\hat{\delta}_0(k-1) \neq 0$ and $L_{k-1} > k-1$

$g_0 = c_{k-1}^2,\ g_1 = 0,\ g_2 = c_{k-1} \cdot \hat{\delta}_0(k-1),\ g_3 = \beta_{k-1}$

$\hat{\theta}_i(k) = \hat{\theta}_i(k-1),\ \hat{\theta}_{2t-2k}(k) = \hat{\theta}_{2t-2k+1}(k) = 0$

$L_k = L_{k-1},\ \alpha_k = \alpha_{k-1},\ c_k = c_{k-1}$

Step 2:

$$\hat{\delta}_i(k) = g_0 \cdot \hat{\delta}_{i+2}(k-1) + g_1 \cdot \hat{\delta}_{i+1}(k-1) + g_2 \cdot \hat{\theta}_{i+1}(k-1) + g_3 \cdot \hat{\theta}_i(k-1)$$

$$\begin{aligned}\beta_k = {} & g_0(\alpha_k \cdot \hat{\delta}_2(k-1) + c_k \cdot \hat{\delta}_3(k-1)) + g_1(\alpha_k \cdot \hat{\delta}_1(k-1) + c_k \cdot \hat{\delta}_2(k-1)) \\ & + g_2(\alpha_k \cdot \hat{\theta}_1(k-1) + c_k \cdot \hat{\theta}_2(k-1)) + g_3(\alpha_k \cdot \hat{\theta}_0(k-1) + c_k \cdot \hat{\theta}_1(k-1))\end{aligned}$$

- Output:

$\Lambda_i(t+1) = \hat{\delta}_i(t+1), (i = 0, 1, ..., t).$

The structure of processing element in half iteration RiBM algorithm is depicted in Fig. 4.11 and Fig. 4.12 shows the homogeneous architecture of half iteration RiBM with $2t+1$ HI-PEs. At the beginning of the process, Λ_0 is initialized to 1 and stored into HI-PE$_{2t}$. In each iteration, Λ_0 is calculated and stored into HI-PE$_{2t-2k}$. Notice that the discrepancies Δ_{2k-1} and ε_{2k} are always in the first and second HI-PE ($\hat{\delta}_0(k)$ and $\hat{\delta}_1(k)$). The critical path passes through only two multipliers, two adders and one multiplexer because the updating factor β_k for the next iteration is required to be available at the beginning of the next clock cycle.

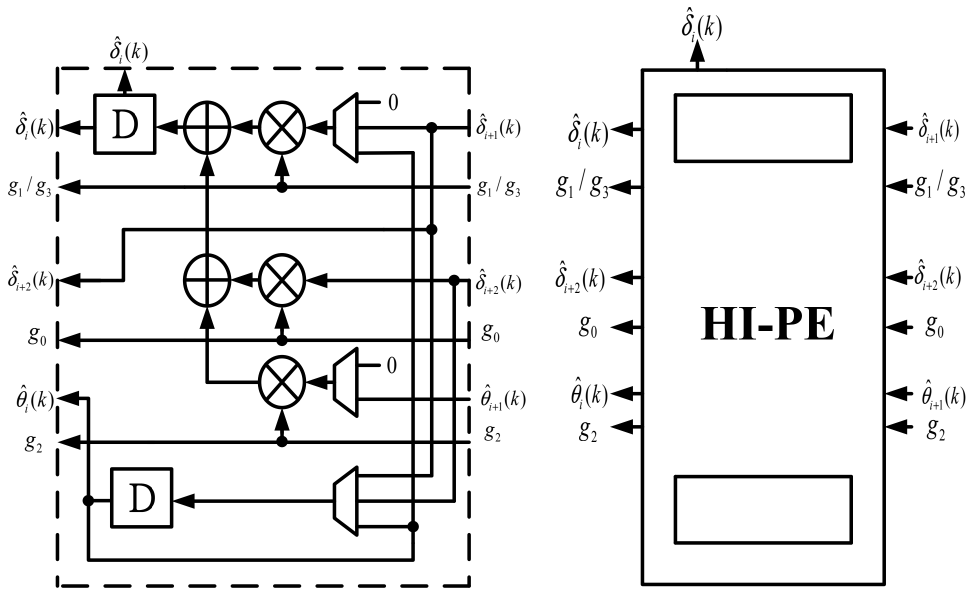


Figure 4.11: The processing element (HI-PE) of half iteration RiBM

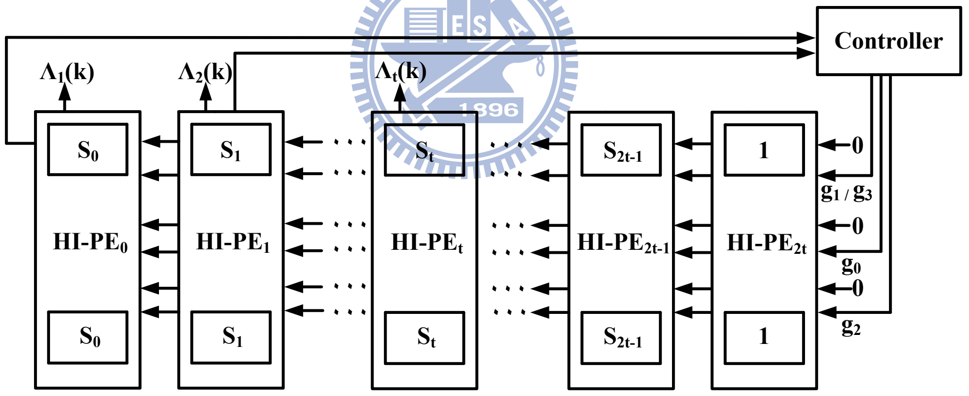


Figure 4.12: The homogeneous architecture of half iteration RiBM

4.3.4 Error Value Evaluator

The large parallelism requirement induces costly hardware complexity for error values calculation. The Björck-Pereyra (BP) [59] based method is applied to replace the Forney's algorithm for computing error values because syndrome values S_i and error locators β_{l_i} have following relation:

$$\begin{bmatrix} \beta_{l_1} & \beta_{l_2} & \cdots & \beta_{l_t} \\ \beta_{l_1}^2 & \beta_{l_2}^2 & \cdots & \beta_{l_t}^2 \\ \vdots & \vdots & \ddots & \vdots \\ \beta_{l_1}^t & \beta_{l_2}^t & \cdots & \beta_{l_t}^t \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_t \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_t \end{bmatrix}, \tag{4.13}$$

where l_i is the i-th error location. To decrease the operating latency for the BP algorithm introduced in Section 3.1, the S_i and S_{i-1} can be computed simultaneously in Step 2. Following shows the detail of the modified BP algorithm:

Modified Björck-Pereyra Algorithm

- Input:

  $\underline{\mathcal{B}}$ and $\underline{S}$.

- Step 1:

```
for $(k = 1; k < t, k = k + 1)$
    for $(i = t; i > k, i = i - 1)$
        $S_i = S_i - \beta_{l_k} S_{i-1}$
```

- Step 2:

```
for $(k = t - 1; k > 0, k = k - 1)$
    for $(i = k + 1; i \leq t, i = i + 1)$
        $S_i = S_i / (\beta_{l_i} - \beta_{l_{i-k}})$
        $S_{i-1} = S_{i-1} - S_i$
```

- Step 3:

```
for $(k = 1; k \leq t, k = k + 1)$
    $S_i = S_k / \beta_{l_k}$
```

- Output:

  $\underline{S}$, where S_i is the error value at l_i-th location.

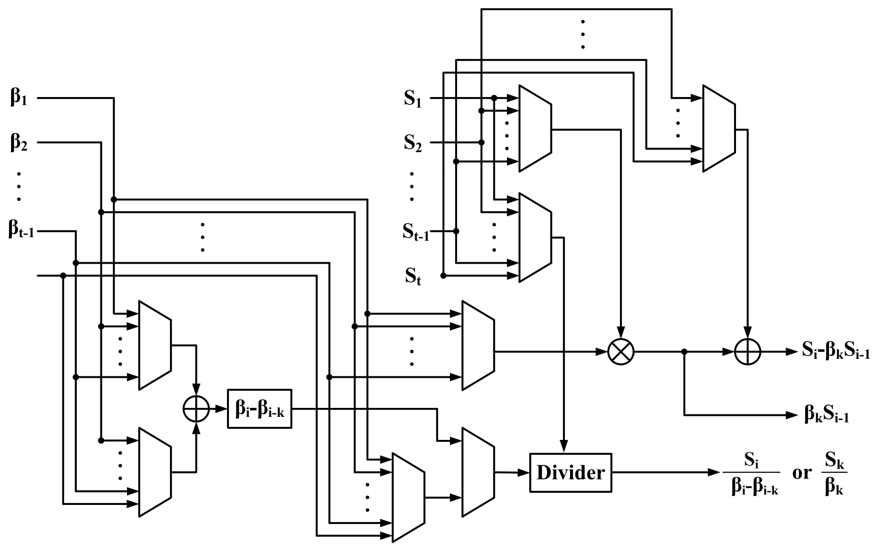


Figure 4.13: BP-based error value evaluator

The variable S_i, which initially represents the i-th syndrome value, is updated iteratively. Each calculation of the syndrome represents a row operation in (4.13). The control logic determines the computation order of the S_i and X_i, and the computation results will be used to update each S_i value. After all computations, S_i indicates the i-th error value. Although BP algorithm takes t^2 cycles time to work, in our design, it matches our timing schedule.

4.3.5 Simplified Decision Making Unit

The Hamming distance of each candidate codeword $\hat{C}^{(i)}(x)$ to the received data $R(x)$ is sufficient to choose the desired output. From (4.6), the weight of $e^{(i)}(x) + e_f^{(i)}(x)$ stands for the Hamming distance between $\hat{C}^{(i)}(x)$ and $R(x)$. That means at most $t + \eta$ symbols are different while counting the distance of candidates codewords. Fig. 4.14 illustrates the proposed simplified decision making unit. The estimated error pattern $e^{(i)}(x)$ and the flipped error pattern $e_f^{(i)}(x)$ are added at first and the corresponding weight is calculated. Then the minimum weight error pattern $e_{min}(x)$ is determined by comparing the current minimum weight with the weight of $e^{(i)}(x) + e_f^{(i)}(x)$. If the weight of $e^{(i)}(x) + e_f^{(i)}(x)$ is smaller than the stored value, $e^{(i)}(x) + e_f^{(i)}(x)$ becomes the minimum weight error pattern $e_{min}(x)$. While 2^η candidate codewords are all processed, the results in registers are the determined output.

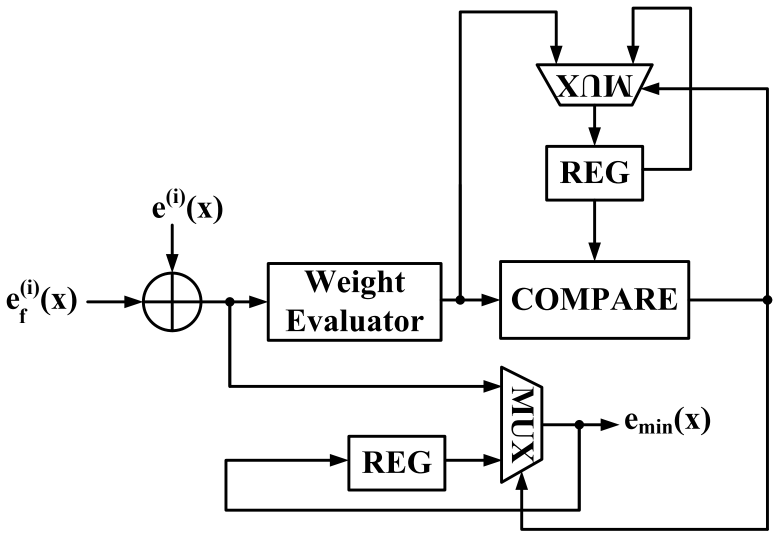


Figure 4.14: Simplified decision making unit.

4.3.6 Parallel Chien Search Architecture

In an $(N, K; t)$ BCH/RS decoder, once the error location polynomial $\Lambda(x)$ is obtained in the decoding process, a Chien search block shown in Fig. 4.15 can be used to exhaustively examine whether $\Lambda(\alpha^i) = 0$ for $i = 0 \sim N - 1$, where

$$\Lambda(\alpha^i) = \sum_{j=0}^{t} \Lambda_j \alpha^{(i)^j} = \sum_{j=1}^{t} \Lambda_j \alpha^{ij} + 1. \tag{4.14}$$

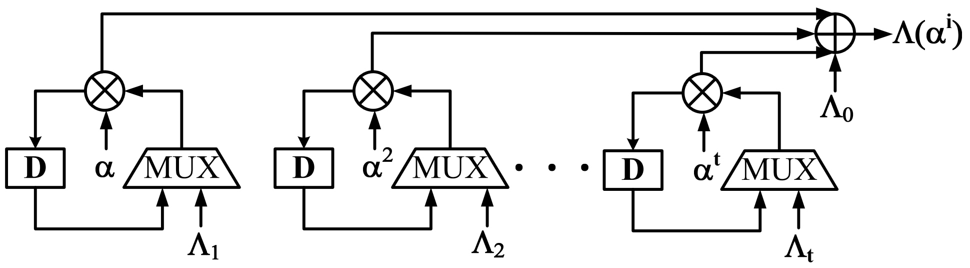


Figure 4.15: Conventional Chien Search Architecture.

Notice that an arbitrary element over $GF(2^m)$ is presented as $\sum_{i=0}^{m-1} a_i\alpha^i$ with the binary coordinate a_i due to the basis $\{\alpha^0, \alpha^1, \cdots, \alpha^{m-1}\}$. Therefore, $\Lambda_j\alpha^{ij}$ can be expressed as

$$\begin{aligned} P_{ij} &= \Lambda_j\alpha^{ij} \\ &= (\lambda_{j,0}\alpha^0 + \lambda_{j,1}\alpha^1 + \cdots + \lambda_{j,m-1}\alpha^{m-1})\alpha^{ij} \\ &= p_{ij,0}\alpha^0 + p_{ij,1}\alpha^1 + \cdots + p_{ij,m-1}\alpha^{m-1}, \end{aligned} \tag{4.15}$$

where $\{\lambda_{j,0}, \lambda_{j,1}, \cdots, \lambda_{j,m-1}\}$ and $\{p_{ij,0}, p_{ij,1}, \cdots, p_{ij,m-1}\}$ are the coordinates of Λ_j and P_{ij} respectively, and

$$\begin{bmatrix} p_{ij,0} \\ p_{ij,1} \\ \vdots \\ p_{ij,m-1} \end{bmatrix} = \begin{bmatrix} \alpha_0^{ij} & \alpha_0^{ij+1} & \cdots & \alpha_0^{ij+m-1} \\ \alpha_1^{ij} & \alpha_1^{ij+1} & \cdots & \alpha_1^{ij+m-1} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{m-1}^{ij} & \alpha_{m-1}^{ij+1} & \cdots & \alpha_{m-1}^{ij+m-1} \end{bmatrix} \begin{bmatrix} \lambda_{j,0} \\ \lambda_{j,1} \\ \vdots \\ \lambda_{j,m-1} \end{bmatrix}. \tag{4.16}$$

Notice that the binary element α_l^k stands for the l-th coordinate of α^k. We define ρ_{ij} as the *density* of the matrix constructed with coordinates of $\alpha^{ij} \sim \alpha^{ij+m-1}$ as shown in (4.16) (i.e., the ratio between the number of 1's and the number of all entries) [68]. Then the complexity of an α^{ij}-CFFM, a constant finite field multiplier with α^{ij} as the multiplicator, is around $m \times (m-1) \times \rho_{ij}$ XOR gates according to (4.16).

To improve the decoding efficiency for the long BCH/RS codes, multiple successive locations can be examined with parallel Chien search architectures. Fig. 4.16 depicts two conventional p-parallel Chien search architectures to shorten the operating cycles from N to $\left\lceil \frac{N}{p} \right\rceil$, where p is the parallel factor. Fig. 4.16(a) is the straight-forward version from Fig. 4.15 while Fig. 4.16(b) is the direct-unfolded version with the unfolded factor p [69, 70]. Both designs have $p \times t$ CFFMs and p $(t+1)$-input m-bit *finite field adders (FFAs)*, resulting in a linear dependence of p for the hardware complexity. The directed-unfolded architecture, which utilizes α^i-CFFM to replace α^{ij}-CFFM for $j = 2 \sim p$, provides lower hardware complexity because the density ρ_i is much smaller than ρ_{ij} for $i < m$. Nevertheless, the critical

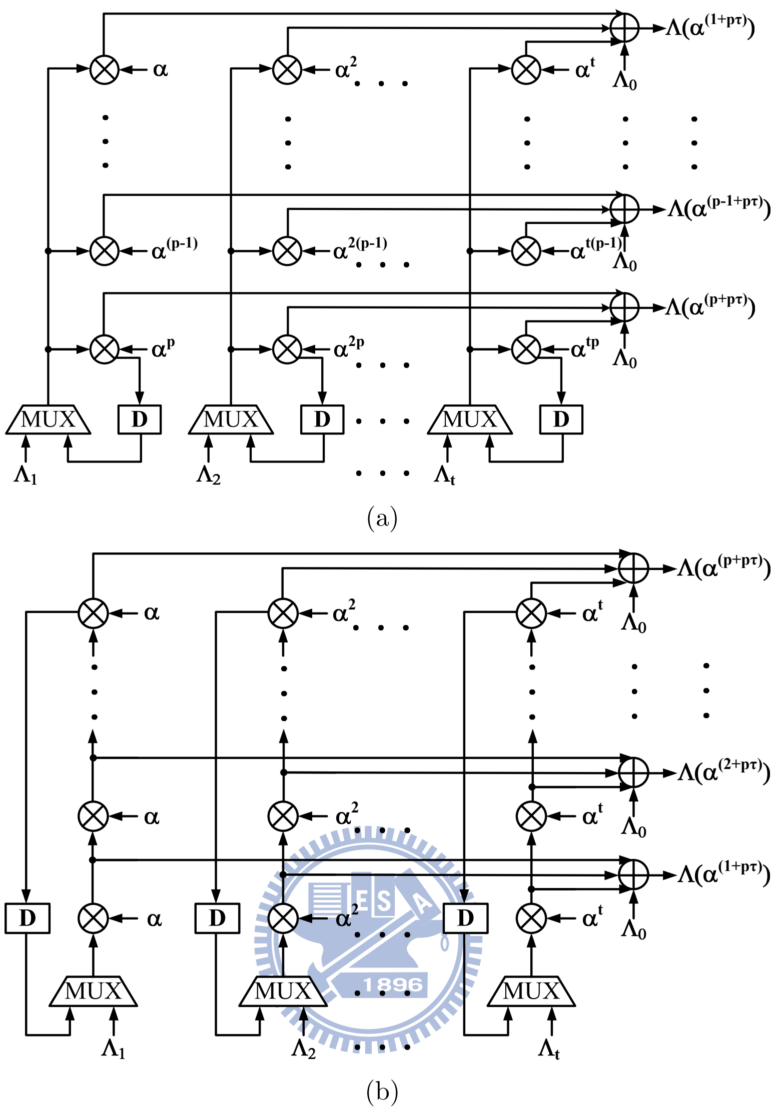


Figure 4.16: Conventional p-Parallel Chien Search Architectures. (a) Straight-Forward. (b) Direct-Unfolded.

path in Fig. 4.16(b) is $(T_{mux} + p \times T_m + T_a)$ while that in Fig. 4.16(a) is only $(T_{mux} + T_m + T_a)$, where T_{mux}, T_m, and T_a represent the critical path of multiplexer, CFFM, and FFA respectively. The direct-unfolded architecture will lead to p times longer critical path if CFFM dominates the delay path.

The hardware complexity of the high parallel Chien search architecture is dominated by numerous CFFMs. This section will reformulate the Chien search equation with minimal polynomials and utilize *minimal polynomial combinational networks (MPCNs)* for replacing the CFFMs in Chien search architecture. In addition, the proposed MPCN-based Chien search architecture can merge the syndrome calculator with small overhead, leading to significant hardware complexity reduction.

To calculate $\Lambda_j\alpha^{ij}$ with minimal polynomials, the proposed new Chien search scheme defines a $m-1$ degree polynomial $T_j(x) = t_{j,0} + t_{j,1}x^1 + \cdots + t_{j,m-1}x^{m-1}$, and the relation between Λ_j and $T_j(x)$ is defined as

$$\begin{aligned}\Lambda_j &= T_j(x)\,|_{x=\alpha^j} \\ &= t_{j,0}\alpha^0 + t_{j,1}\alpha^j + \cdots + t_{j,m-1}\alpha^{(m-1)j} \\ &= \lambda_{j,0}\alpha^0 + \lambda_{j,1}\alpha^1 + \cdots + \lambda_{j,m-1}\alpha^{m-1}. \end{aligned} \tag{4.17}$$

From (4.17), $\{\lambda_{j,0}, \lambda_{j,1}, \cdots, \lambda_{j,m-1}\}$ and $\{t_{j,0}, t_{j,1}, \cdots, t_{j,m-1}\}$ are viewed as coordinates with the basis $\{\alpha^0, \alpha^1, \cdots, \alpha^{m-1}\}$ and $\{\alpha^0, \alpha^j, \cdots, \alpha^{(m-1)j}\}$ respectively, where

$$\begin{bmatrix} \lambda_{j,0} \\ \lambda_{j,1} \\ \vdots \\ \lambda_{j,m-1} \end{bmatrix} = \begin{bmatrix} \alpha_0^0 & \alpha_0^j & \cdots & \alpha_0^{(m-1)j} \\ \alpha_1^0 & \alpha_1^j & \cdots & \alpha_1^{(m-1)j} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{m-1}^0 & \alpha_{m-1}^j & \cdots & \alpha_{m-1}^{(m-1)j} \end{bmatrix} \begin{bmatrix} t_{j,0} \\ t_{j,1} \\ \vdots \\ t_{j,m-1} \end{bmatrix}. \tag{4.18}$$

After the binary matrix operation in (4.18), Λ_j is represented with the basis from $\{\alpha^0, \alpha^j, \cdots, \alpha^{(m-1)j}\}$ to $\{\alpha^0, \alpha^1, \cdots, \alpha^{m-1}\}$; therefore, this operation is called as *j-th basis transformer (BT_j)*. As a result, the coefficients of $T_j(x)$ can be determined as

$$\begin{bmatrix} t_{j,0} \\ t_{j,1} \\ \vdots \\ t_{j,m-1} \end{bmatrix} = \begin{bmatrix} \alpha_0^0 & \alpha_0^j & \cdots & \alpha_0^{(m-1)j} \\ \alpha_1^0 & \alpha_1^j & \cdots & \alpha_1^{(m-1)j} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_{m-1}^0 & \alpha_{m-1}^j & \cdots & \alpha_{m-1}^{(m-1)j} \end{bmatrix}^{-1} \begin{bmatrix} \lambda_{j,0} \\ \lambda_{j,1} \\ \vdots \\ \lambda_{j,m-1} \end{bmatrix}, \tag{4.19}$$

where the operation of the inverse matrix in (4.19) is called as *j-th inverse basis transformer (IBT_j)*.

Based on our definitions, Λ_j can be represented in terms of $T_j(x)$ and (4.15) becomes

$$\begin{aligned} P_{ij} &= \Lambda_j \alpha^{ij} \\ &= T_j(x)\,\big|_{x=\alpha^j} \times (\alpha^j)^i \\ &= x^i T_j(x)\,|_{x=\alpha^j} \\ &= M_j(x) \times W_j(x) + D_j(x)\,|_{x=\alpha^j}\ , \end{aligned} \tag{4.20}$$

where $M_j(x)$ is the minimal polynomial of α^j and $D_j(x)$ is the remainder polynomial resulting from dividing $x^i T_j(x)$ by $M_j(x)$. Since α^j is a root of $M_j(x)$, $D_j(\alpha^j)$ is the only non-zero term in (4.20). Then the Chien search equation can be reformulated as

$$\Lambda(\alpha^i) = \sum_{j=1}^{t} P_{ij} + 1 = \sum_{j=1}^{t} D_j(\alpha^j) + 1. \tag{4.21}$$

As shown in (4.21), the Chien search can be simply realized by summing up all the evaluation results of 1st $\sim$ t-th BTs. Instead of executing summation after the basis transformations, the addition operation can be moved before the transformation, leading to fewer transformation operations.

Hence, (4.22) can be reformulated with *group basis transformer (GBT)* as

$$\begin{aligned} \Lambda(\alpha^i) &= \sum_{j=1}^{t} \sum_{k=0}^{m-1} d_{j,k} \alpha^{jk} + 1 \\ &= \sum_{v=0}^{mt} \left(\sum_{\forall jk=v} d_{j,k} \right) \alpha^v + 1, \end{aligned} \tag{4.22}$$

where $d_{j,k}$ is the k-th coefficient of $D_j(x)$.

Fig. 4.17 shows the architectures of three basis components, including j-th MPCN, j-th BT and GBT. The j-th MPCN ($MPCN_j$) shown in Fig. 4.17(a) executes modulo operation with the divisor $M_j(x)$. It is constructed by the combinational circuit of the linear feedback shift register with the connection polynomial $M_j(x)$. Each binary element m_j^k in Fig. 4.17(a) is the k-th coefficient of $M_j(x)$, indicating the wire connection. In the j-th BT shown in

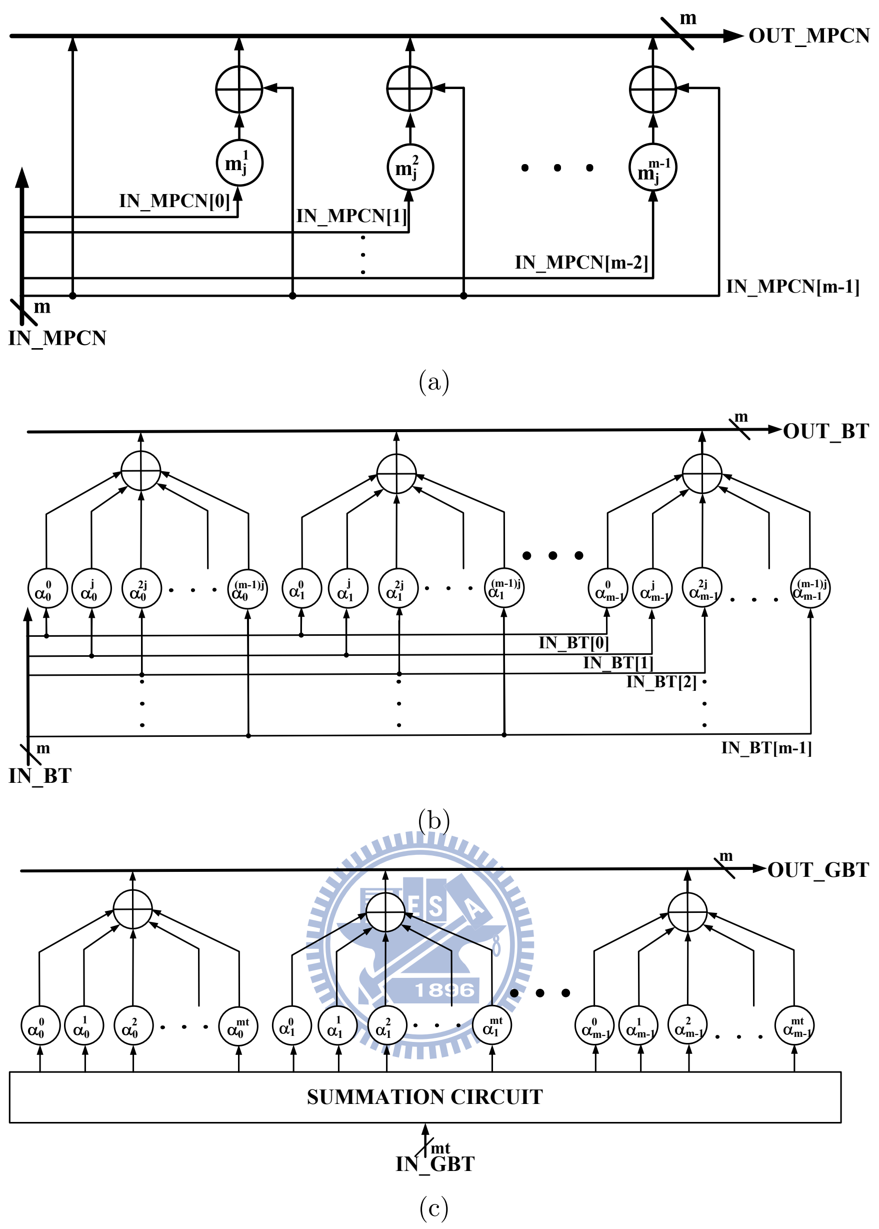


Figure 4.17: Basic components in proposed Chien search architecture. (a) j-th minimal polynomial combinational network ($MPCN_j$). (b) j-th basis transformer (BT_j). (c) Group basis transformer (GBT).

Fig. 4.17(b), each α_l^k is a binary element as in (4.18) and can be represented whether the wire is connected or not. Fig. 4.17(c) illustrates the block diagram of the GBT. The additions are executed firstly with all the coefficients of $D_j(x)$ for $j = 1 \sim t$ (total mt bits), and the similar operations as a BT are applied with the basis $\alpha^0 \sim \alpha^{mt}$.

In the proposed MPCN-based parallel-p Chien search architecture shown in Fig. 4.18, the coefficients of $\Lambda(x)$ are applied to the IBTs for transforming the operating basis. According to

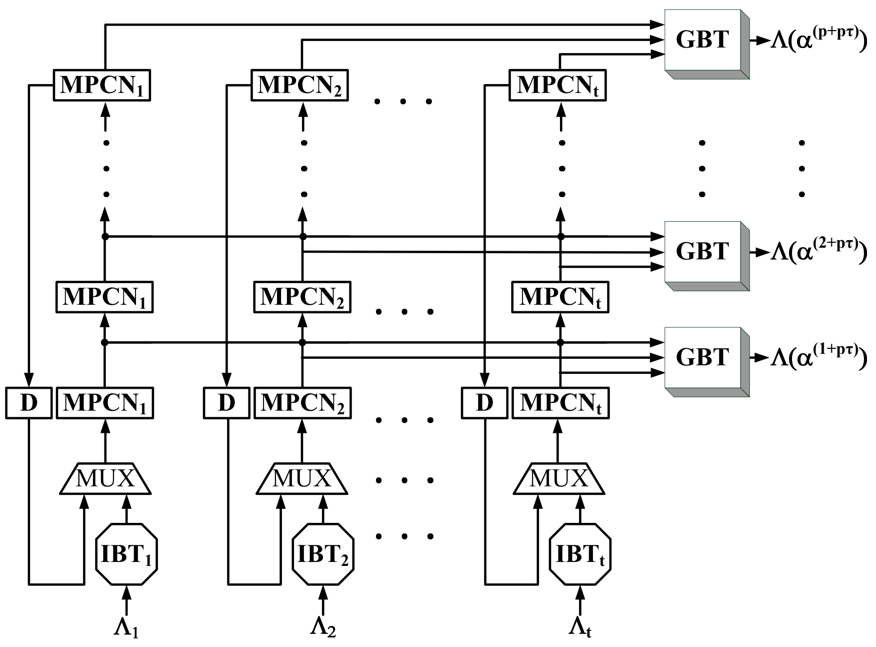

GBT
$\Lambda(\alpha^{(p+p\tau)})$
$\Lambda(\alpha^{(2+p\tau)})$
$\Lambda(\alpha^{(1+p\tau)})$
$MPCN_1$
$MPCN_2$
$MPCN_t$
D
MUX
IBT_1
IBT_2
IBT_t
Λ_1
Λ_2
Λ_t

Figure 4.18: MPCN-Based parallel-p Chien search architecture.

(4.20) $\sim$ (4.22), the transformed values are evaluated with minimal polynomials for obtaining the Chien search results. All the multiplexers select the outputs of IBTs in the first cycle, and then select the register data afterward. Searching from $(N-1)$-th to 0-th location, the proposed design checks p locations at each cycle. In each row, mt bits data are fed into a GBT to examine the error locations. An error is found at $(N+r-p(\tau+1)-1)$-th location if the output of the r-th row GBT equals zero at τ-th cycle. In contrast to Fig. 4.16, our proposed Chien search architecture utilizes $p \times t$ MPCNs to replace $p \times t$ CFFMs. Notice that the XOR gate count requirement of one MPCN is at most $m-1$, which is much smaller than that of one CFFM. Therefore, it is area-efficient to apply the MPCNs, especially in the large parallelism conditions.

The proposed MPCN-based architecture can merge the syndrome calculator and Chien search in the same hardware with small overhead. In the BCH/RS decoding process, the received polynomial $R(x)$ is fed into the syndrome calculator to generate syndrome polynomial

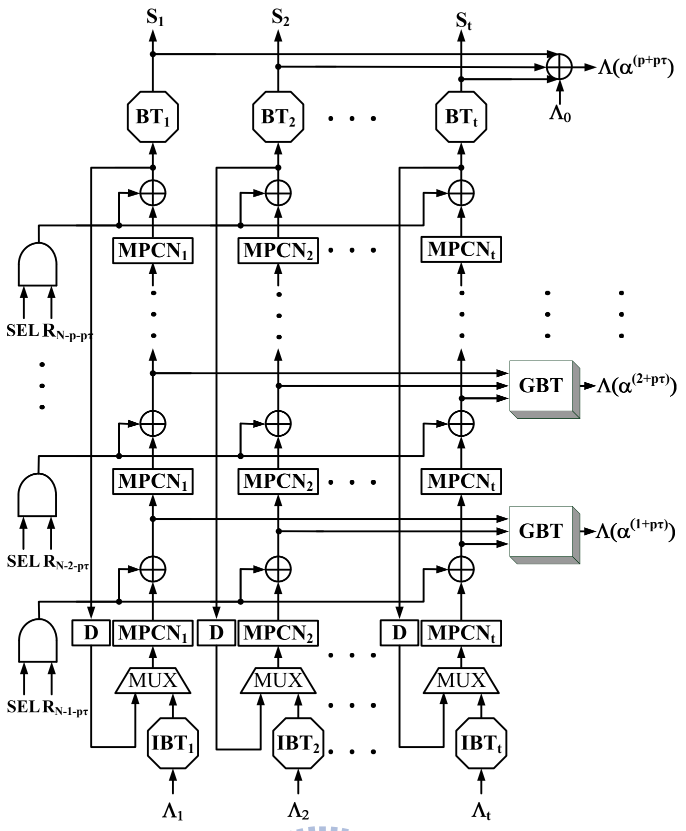


Figure 4.19: Parallel-p joint syndrome calculator and Chien search with MPCN-based architecture.

$S(x) = S_1 + S_2x^1 + \cdots + S_{2t}x^{2t-1}$, which is expressed as [3]

$$\begin{aligned} S_j &= R(x)\,|_{x=\alpha^j} \\ &= M_j(x) \times Q_j(x) + B_j(x)\,|_{x=\alpha^j} \\ &= B_j(\alpha^j), \end{aligned} \tag{4.23}$$

where $B_j(x)$ is the remainder polynomial resulting from dividing $R(x)$ by $M_j(x)$. Consequently, the j-th syndrome value can be calculated with $M_j(x)$.

Fig. 4.19 illustrates our parallel-p joint syndrome calculator and Chien search with MPCN-based architecture. The syndrome calculator phase and Chien search phase are determined by the *SEL* signal. When the *SEL* signal is high, the j-th syndrome value is formulated as

$$\begin{aligned} S_j &= (((R_{N-1}x^{p-1}+\cdots+R_{N-p-1}) \bmod M_j(x))x^p \\ &+ (R_{N-p-2}x^{p-1}+\cdots+R_{N-2p-1})) \bmod M_j(x))x^p \\ &+ \cdots)x^p+R_{p-1}x^{p-1}+\cdots+R_0) \bmod M_j(x)\,|_{x=\alpha^j} \end{aligned} \tag{4.24}$$

The partial remainder stored in the register is multiplied by x^p and accumulated with the received symbols. After all the received symbols are processed, the BT_j transforms the accumulated result to j-th syndrome value. In contrast to Fig. 4.18, t BTs are applied instead of one GBT in the first row to evaluate individual syndrome value. Note that the FFA in Fig. 4.19 is only a 1-bit operation because each coefficient of $R(x)$ is binary value. Therefore, except for the difference between the BT and GBT, the overhead of supporting syndrome calculation is only p NAND and $p \times t$ XOR gates.

4.3.7 Architecture Comparison

For RS (224, 216; 4) code, there was no soft RS (224, 216; 4) decoder has been published according the best knowledge of the author. Thus, a LCC soft RS (255, 239; 8) decoder [34] is shortened to RS (224, 216; 4) decoder for comparison. The proposed decision-eased soft RS decoder with 2-stage pipeline architecture is compared with the LCC soft RS decoder with 4-stage pipeline architecture as shown in TABLE 4.3. Both soft decoders evaluates 3 LRPs for generating candidate sequences.

The LCC decoder has 4-stage pipeline architecture and has to storage every candidate codeword, resulting in large amount of storage elements. Notice that, the complexity ratio over $GF(2^8)$ among XOR, CFFM, FFM, FFA, MUX, Register, ROM (byte) and RAM (byte) is 1 : 20 : 100 : 8 : 3: 1 : 8 : 8. While the complexity of these designs is normalized to XOR gate, the proposed decision-eased soft RS decoder is around 13,248 XOR gates and the LCC soft RS decoder is about 32,991 XOR gates. Due to fewer number of FFMs and storage elements, our proposed decoder can save around 59.8% complexity as compared to

LCC decoder, even though the LCC decoder excludes the decision making unit.

Table 4.3: Comparison Table for Soft RS (224, 216; 4) Decoder

Architecture	$GF(2^8)$ CFFM	$GF(2^8)$ FFM	$GF(2^8)$ FFA	2-to1 MUX (Bit)	Register (Bit)	ROM (Byte)	RAM (Byte)	Latency (Cycle)
Decision-Eased with $\eta = 3$								
Syndrome Calculator	8	0	8	0	64	0	0	224
Reliability Evaluator	0	0	0	192	0	0	0	224
Syndrome Updater	0	1	1	128	64	232	0	16
Key Equation Solver	0	9	8	72	136	0	0	16
Parallel-16 Chien Search	64	0	64	128	224	0	0	16
BP-Based Error Value Evaluator	0	2	2	192	64	224	0	16
Simplified Decision Making Unit	0	0	3	80	80	0	0	16
Total	**72**	**12**	**86**	**792**	**632**	**456**	**0+224×2**	**224**
LCC with $\eta = 3$ [34]								
Re-encoder	0	13	23	392	344	448	0	464
Interpolation	0	14	12	87	166	0	68	461
Polynomial Select	0	8	8	139	264	0	0	23
Chien Search	4	0	4	0	64	0	0	216
Forney's Algorithm	0	2	2	136	24	224	0	76
Erasure Decoder	0	13	23	243	168	224	0	464
Total	**4**	**50**	**72**	**997**	**1430**	**896**	**68+224 × 8**	**464**

* The complexity ratio over $GF(2^8)$ among XOR, CFFM, FFM, FFA, MUX, Register, ROM (byte) and RAM (byte) is 1 : 20 : 100 : 8 : 3: 1 : 8 : 8.

For RS (255, 239; 8) code, the proposed decision-confined soft RS decoder with 3-stage pipeline architecture is compared with the LCC soft RS decoder with 4-stage pipeline architecture as shown in TABLE 4.4. The proposed design evaluates 5 LRPs for generating candidate sequences while the LCC decoder evaluates 3 LRPs.

In LCC decoder, each candidate codeword to be stored; therefore, the LCC decoder with 4-stage pipeline architecture requires large number of storage elements. While the complexity of these designs is normalized to XOR gate, the proposed decision-eased soft RS decoder is around 22,534 XOR gates and the LCC soft RS decoder is about 38,671 XOR gates. Even though the complexity of LCC decoder excludes the complexity of the decision making unit, our proposed decoder can save around 42% complexity as compared to LCC decoder because of fewer number of storage elements.

Table 4.4: Comparison Table for Soft RS (255, 239; 8) Decoder

Architecture	$GF(2^8)$ CFFM	$GF(2^8)$ FFM	$GF(2^8)$ FFA	2-to1 MUX (Bit)	Register (Bit)	ROM (Byte)	RAM (Byte)	Latency (Cycle)
Decision-Confined with $\eta = 5$								
Syndrome Calculator	16	0	16	0	128	0	0	256
Reliability Evaluator	0	0	0	40	90	0	0	259
Syndrome Updater	0	4	16	288	128	264	0	8
Half Iteration Key Equation Solver	0	62	37	464	296	0	0	8
Parallel-2 Chien Search	16	0	16	64	64	0	0	128
BP-Based Error Value Evaluator	0	2	2	448	128	256	0	92
Total	**32**	**68**	**87**	**1592**	**834**	**520**	**0+256×3**	**259**
LCC with $\eta = 3$ [34]								
Re-encoder	0	21	39	448	600	512	0	528
Interpolation	0	14	12	87	166	0	68	525
Polynomial Select	0	8	8	139	264	0	0	23
Chien Search	8	0	8	0	128	0	0	239
Forney's Algorithm	0	2	2	136	24	256	0	152
Erasure Decoder	0	21	39	299	424	256	0	528
Total	**8**	**66**	**108**	**1109**	**1606**	**1024**	**68+256 × 8**	**528**

* The complexity ratio over $GF(2^8)$ among XOR, CFFM, FFM, FFA, MUX, Register, ROM (byte) and RAM (byte) is 1 : 20 : 100 : 8 : 1 : 3 : 8 : 8.

Chapter 5

Implementation Results

5.1 EM-Type Soft BCH (32400, 32208; 12) Decoder for DVB-S2 System

DVB-S2 system [22] shown in Fig. 5.1 is the second generation standard of Digital Video Broadcasting via satellites, which is developed by European Telecommunications Standard Institute (ETSI). It provides higher order modulations and a powerful FEC system based on serial concatenation of BCH codes and low density parity check (LDPC) codes, leading to 30% channel capacity gain over exsiting DVB-S standard [21]. For the high speed and long distance data transmission, the BCH codes with long block length are specified to suppress the error floor due to iterative LDPC decoding. The BCH codes have different 21 code rates, where there are $GF(2^{16})$ for DVB-S2 *normal frame* and $GF(2^{14})$ for DVB-S2 *short frame*. The detailed specifications of 21 kinds BCH codes are listed in TABLE. 5.1. The long

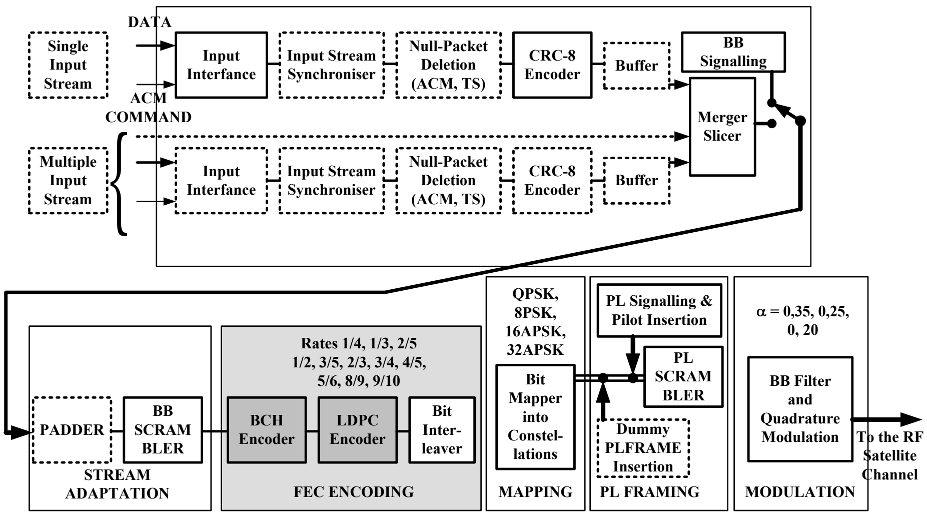


Figure 5.1: Block diagram of DVB-S2 transmitter.

Table 5.1: DVB-S2 Specification (N : Codeword Length, K : Message Length)

	Normal Frame N_LDPC : 64800			Short Frame N_LDPC : 16200		
LDPC Code Rate	N_BCH	K_BCH	t	N_BCH	K_BCH	t
1/4	16200	16008	12	3240	3072	12
1/3	21600	21408	12	5400	5232	12
2/5	25920	25728	12	6480	6312	12
1/2	32400	32208	12	7200	7032	12
3/5	38880	38688	12	9720	9552	12
2/3	43200	43040	10	10800	10632	12
3/4	48600	48408	12	11880	11712	12
4/5	51840	51648	12	12600	12432	12
5/6	54000	53840	10	13320	13152	12
8/9	57600	57472	8	14400	14232	12
9/10	58320	58192	8	NA	NA	NA

block length, which has 58320 bits at most, demands considerably complex arithmetic over large finite field, resulting in higher circuit complexity. Moreover, the storage requirement becomes costly due to the large codeword should be buffered for error correction, where a memory bank is about 100 K gate-count for a BCH codeword in DVB-S2 system.

Over 50 decoding iterations in the LDPC decoder result in a long decoding latency and a short period of data output time. The BCH decoder for the DVB-S2 system is required to achieve at most 250 Mb/s throughput that accommodates the LDPC decoder output. For long block length BCH decoders, the decoding latency is dominated by the syndrome calculator and Chien search. Pipeline architecture can improve the throughput but more memory banks are required to store each stage codeword. Parallelism Chien search is another approach to enhance the throughput [68, 71] but it causes more hardware complexity. Therefore, a decoder with single stage pipeline and serial architecture forms a low complexity design.

In this section, a low complexity and high throughput BCH decoder is designed for DVB-S2 system. To achieve the throughput requirement with low complexity, the soft information from the LDPC decoder can be employed by BCH decoders in DVB-S2 system. Soft information can help the decoder to choose the least reliable bits and then the decoder

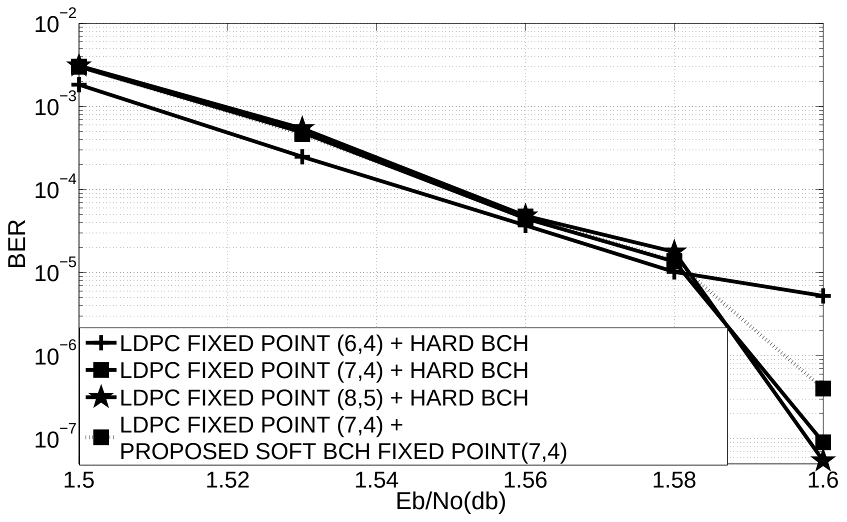


Figure 5.2: Fixed point simulation results for BCH (32400, 32208; 12))in DVB-S2 system

can find certain errors from those bits. Based on this concept, an error locator evaluator is proposed to eliminate Chien search for higher throughput. Furthermore, the decoder can maintain similar performance because the soft information from LDPC decoder provides sufficient reliability.

In DVB-S2 system, BCH (32400, 32208; 12) over $GF(2^{16})$ is defined to be concatenated with LDPC (64800, 32400) code. Fig. 5.2 analyzes the required bit-width for the LDPC and BCH codes in DVB-S2 system. The simulation parameters *fixed point (A,B)* represents A-bit input quantization with B-bit decimal fraction. The BER performance for the fixed point LDPC code concatenated with the hard BCH code is demonstrated with the solid lines. The BER curves indicate that the fixed point (6,4) LDPC code has performance loss and the fixed point (7,4) LDPC code is sufficient to achieve similar performance to the fixed point (8,5) LDPC code. Since the more input quantization in LDPC code results in linear increment for longer critical path delay and message storage, the fixed point (7,4) LDPC code is adequate in our approach. In addition, the proposed soft BCH code with 7-bit input quantization is also simulated in Fig. 5.2 with the dotted line while it receives the soft information from the the fixed point (7,4) LDPC code. It can achieve less than 0.01 dB performance loss in contrast to the hard BCH code at BER $= 10^{-6}$, indicating that 7-bit input quantization is sufficient for our approach. Notice that the input quantization of the proposed soft BCH

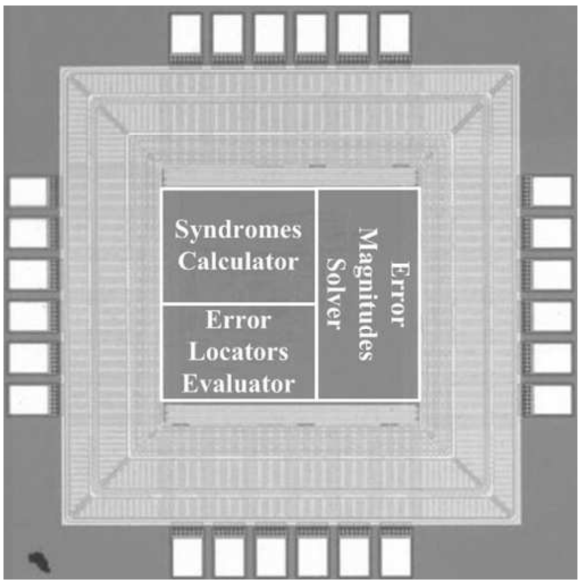


Figure 5.3: Microphoto of Soft BCH (32400, 32208; 12) Chip

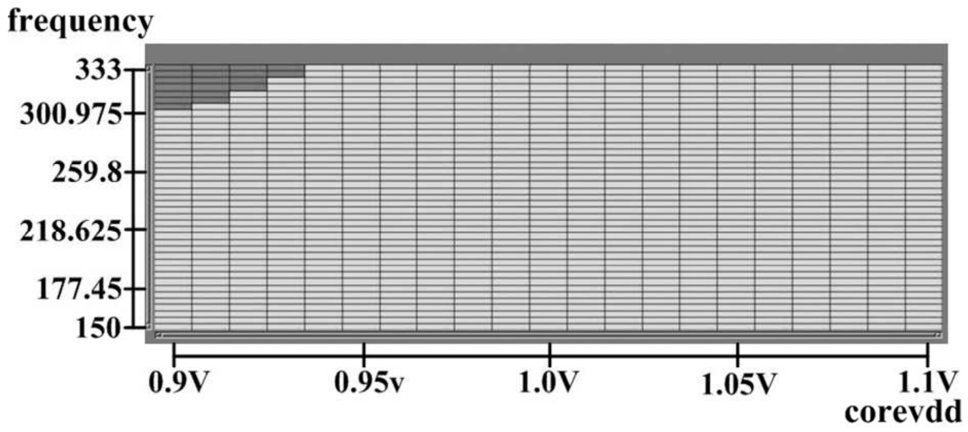


Figure 5.4: Shmoo Plot of Soft BCH (32400, 32208; 12) Chip

code only affects the size of the comparators and components of reliability part in error locator evaluator. Consequently, it has little influence on hardware complexity.

Fig. 5.3 shows the proposed soft BCH (32400, 32208; 12) decoder die photo, which is implemented with the cell-based design flow and fabricated in 90nm 1P9M CMOS process. The chip is verified by Agilent 93000 SOC test system and the Shmoo plot shown in Fig. 5.4 indicates that our design can achieve 333 MHz operation frequency at 0.94 V supply. TABLE 5.2 illustrates the chip summary as well as a hard BCH decoder for comparison. To minimize the storage requirement, all the designs in TABLE 5.2 are constructed with single stage pipeline architecture. In the hard BCH decoder, the iBM algorithm is utilized to

Table 5.2: Summary of Implementation Results

	Hard (32400,32208) BCH t = 12	Hard DVB-S2 BCH [72] Normal + Short Frame	Soft (32400,32208) BCH t = 12	Soft DVB-S2 BCH Normal Frame	Soft DVB-S2 BCH Normal + Short Frame
Technology	90nm	0.13 μm	90nm	90nm	90nm
Architecture	iBM w/ Chien Search	Reversed iBM w/ Parallel-4 Chien Search	BP-EMS w/o Chien Search	BP-EMS w/o Chien Search	BP-EMS w/o Chien Search
Pipeline Stage	1	1	1	1	1
Number of Mode	1	21	1	11	21
Operation Frequency	200MHz (Post Layout)	190 MHz (Post Layout)	333MHz (Measurement)	300MHz (Post Layout)	300MHz (Post Layout)
Core Area	190497μm^2	336000μm^2	102400μm^2	105625μm^2	119025μm^2
Gate Count	54.0K	44K	26.9K	28.2K	32.4K
Normalized Gate Count	2.06	1.64	1	1.05	1.2
Maximum Throughput	99.3Mb/s	380 Mb/s *	314.5Mb/s	295.5Mb/s *	295.5Mb/s *
Normalized Throughput	1	3.8	3.17	2.98	2.98
Maximum Latency	64824	29208*	34104	59072 *	59072 *
Power	6.32mW @200MHz	26mW @ 190MHz	8.43mW @333MHz	9.45mW @300MHz	11.9mW @300MHz
Energy Efficiency	63.6 pJ/bit	68.4 pJ/bit	26.8 pJ/bit	32 pJ/bit	40.2 pJ/bit

* The maximum latency and throughput are provided by (58320,58192) BCH code.

solve key equation as well as Chien search is applied to get error locations. By inserting registers into composite field inversion unit, the operation frequency of our proposed soft BCH decoder can be enhanced from 166 MHz to 333 MHz with only 2.5% latency overhead. Moreover, our soft BCH decoder that computing error locations without Chien search has almost half latencies of the hard BCH decoder. The measurement results reveal that our proposed soft BCH decoder performs three times throughput with 50.0% gate-count saving as compared with the hard BCH decoder. The measured power consumption of the proposed soft BCH decoder is 8.43 mW with 1.0 V supply at 314.5 Mb/s, and the energy efficiency is 26.8 pJ/bit while that of the hard BCH decoder is 63.6 pJ/bit.

In addition, the proposed architectures are favorable to multi-mode designs because most components can be shared and only a few control logics are required. While extended to process DVB-S2 normal frame which consists of 11 modes, the proposed design can achieve 300 MHz operation frequency with only 5% gate-count increment as compared to the proposed soft BCH (32400, 32208; 12) decoder. To support those 21 modes defined in DVB-S2 system, including normal and short frames, our approach can achieve 300 MHz operation frequency with gate-count of 32.4 K, which is 20% more than the original single mode design. The post-layout simulation results of these two multi-modes design are also illustrated in TABLE 5.2.

5.2 EM-Type Soft BCH and BCH-Like Decoders with 255-Bit Codeword Length

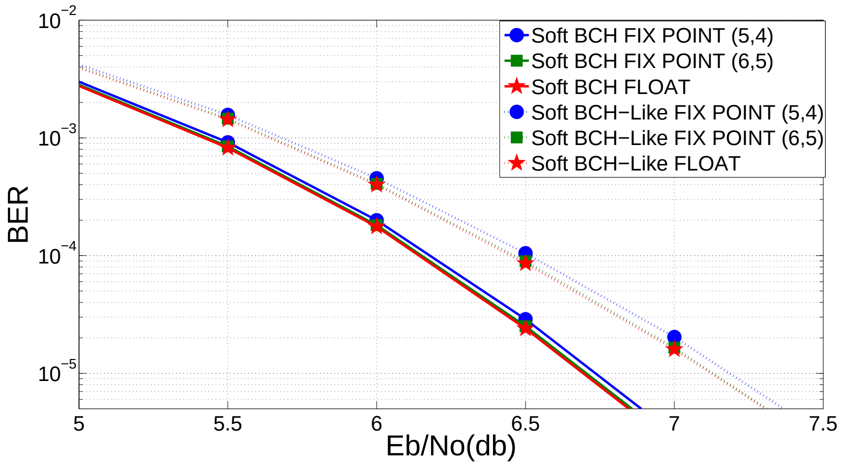
Fig. 5.5 analyzes the required bit-width for the proposed soft BCH and BCH-Like decoders with 255-bit codeword length, where Fig. 5.5(a) is for 2-error-correcting codes and Fig. 5.5(b) is for 3-error-correcting codes respectively. The BER performance for the soft BCH decoders is demonstrated with the solid lines while that of the soft BCH-Like decoders is demonstrated with the dash lines. The BER curves indicate that the proposed soft BCH/BCH-Like codes with 6-bit input quantization are sufficient to achieve similar performance to the floating point BCH/BCH-Like codes. Notice that the input quantization of the proposed soft BCH code only affects the size of the comparators and components of reliability part in error locator evaluator. Consequently, it has little influence on hardware complexity.

In TABLE 5.3, the BCH decoders are implemented with hard and soft decision methods to demonstrate BCH (255, 239; 2) and BCH (255, 231; 3) codes. From TABLE 3.5, the BCH decoder with iBM algorithm has similar hardware complexity as that with SiBM algorithm; therefore, we design the hard BCH decoders which solve key equation with iBM algorithm and evaluate error locations with Chien search. The soft BCH and BCH-Like decoders are designed with CEMS and M-CEMS respectively. The BCH-Like (255, 243; 2) decoder is designed from BCH (255, 239; 2) with $\rho = 51$ whereas the BCH-Like (255, 237; 3) decoder is constructed from BCH (255, 231; 3) with $\rho = 85$.

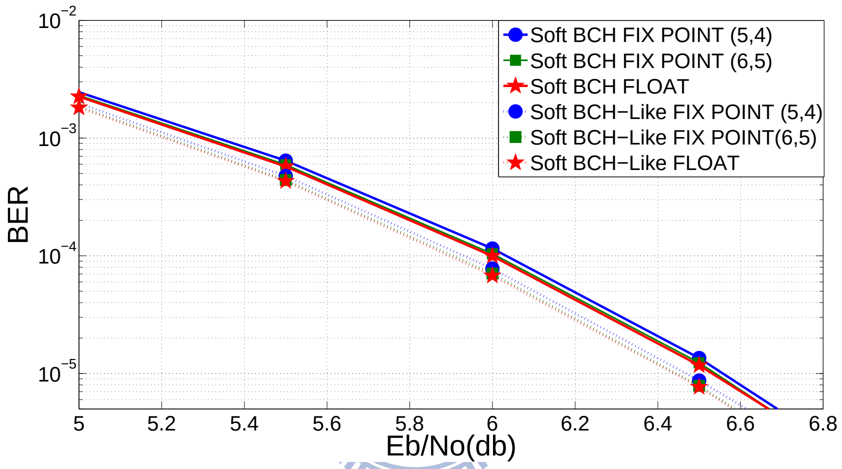
The implementation results reveal that the proposed soft BCH (255, 239; 2) and (255, 231; 3) decoders can reach 4.2 K and 6.7 K gate count with 400 MHz and 360 MHz opera-

Table 5.3: Summary of Implementation Results

	BCH (255, 239) HARD	BCH (255, 239) SOFT	BCH-Like (255, 243) SOFT	BCH (255, 231) HARD	BCH (255, 231) SOFT	BCH-Like (255, 237) SOFT
Technology	90 nm	90 nm	90 nm	90 nm	90 nm	90 nm
Architecture	iBM + Chien Search	CEMS	M-CEMS	iBM + Chien Search	CEMS	M-CEMS
Code Rate	0.937	0.937	0.952	0.905	0.905	0.929
Operation Frequency	400 MHz (Post Layout)	400 MHz (Post Layout)	320 MHz (Post Layout)	360 MHz (Post Layout)	360 MHz (Post Layout)	315 MHz (Post Layout)
Core Area	$14,400\mu m^2$	$13,225\mu m^2$	$13,225\mu m^2$	$21,025\mu m^2$	$21,025\mu m^2$	$24,025\mu m^2$
Gate Count	4.4 K	4.2 K	4.2 K	6.8 K	6.7 K	7.5 K
Latency	514	272	275	516	321	325
Throughput	186 Mb/s	351.5 Mb/s	282.8 Mb/s	161.2 Mb/s	259.1 Mb/s	229.7 Mb/s



(a) BCH (255, 239; 2) and BCH-Like (255, 243; 2)



(b) BCH (255, 231; 3) and BCH-Like (255, 237; 3)

Figure 5.5: Fixed point simulation results for the proposed soft decoders with 255-bit codeword length.

tion frequency respectively in standard CMOS 90 nm technology, which are similar to that provided by the hard BCH decoders. The BCH-Like (255, 243; 2) decoder provides 4.2 K gate count because there is no multiplier in $t = 2$ case, while the BCH-Like (255, 237; 3) decoder has 7.5 K gate count due to an extra ρ-TIMER. The ρ-TIMER results in longer critical path so that the BCH-Like (255, 243; 2) and (255, 237; 3) decoders have slower operation frequency than BCH (255, 239; 2) and (255, 231; 3) decoders do. Nevertheless, the proposed soft BCH and BCH-Like decoders computing error locations without Chien search achieves 1.4 $\sim$ 1.9 times throughput enhancement over the hard BCH decoders as shown in TABLE 5.3.

5.3 Decision-Eased Soft RS (224, 216; 4) Decoder for mmWave System

The millimeter-wave (mmWave) system can promise throughput in the order of multi-Gpbs because of the huge bandwidth ranging from 57-64 GHz. Hence it has aroused much interest, such as Ecma International TC32-TG20 Task Group [73] and IEEE 802.15.3c Task Group [74]. To support the uncompressed high-definition (HD) video transmission, which can enhance picture quality, the mmWave system is required to provide 3.0 Gb/s throughput if the color depth of a single uncompressed HD (1080p) stream is 8-bit. In the future, the throughput is demanded to achieve 4.5 Gb/s due to the 12-bit color depth in the future applications.

Fig. 5.6 shows a block diagram of mmWave system for supporting uncompressed video streaming. Two RS (224, 216; 4) decoders are applied to execute error correction for the 4 MSB and 4 LSB bitstreams respectively, implying a RS (224, 216; 4) decoder is demanded to provide 2.25 Gb/s throughput for future video applications. However, the high throughput

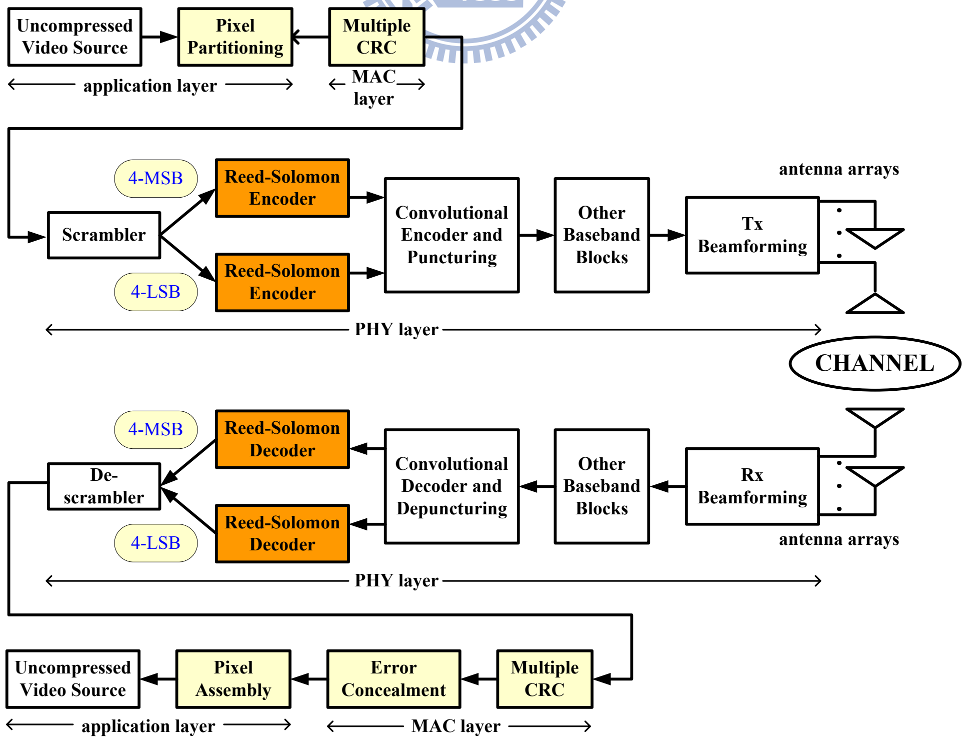


Figure 5.6: Block diagram of the transmitter and the receiver of the mmWave system

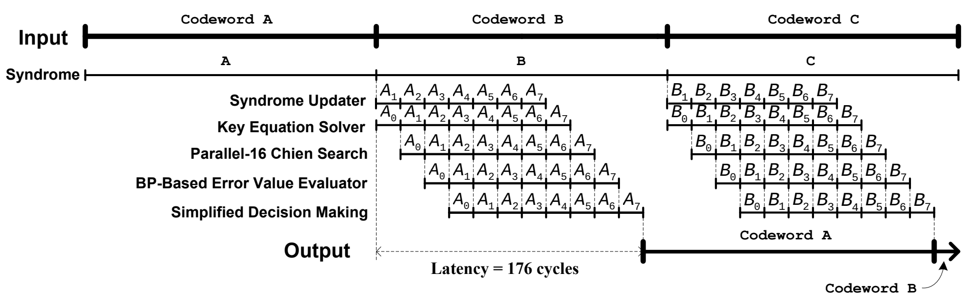


Figure 5.7: Decoding scheme of the decision-eased soft RS decoder

will damage the reliability of the transmitted data. In this section, a low complexity and high throughput soft RS decoder is designed for mmWave system.

For the 2.25 Gb/s requirement of the mmWave systems, a soft RS (224, 216; 4) decoder with 2-stage pipeline architecture is presented in Fig. 5.7 based on our decision-eased decoding algorithm. The syndrome polynomial $S(x)$ is reckoned at data input and the first syndrome pattern is available while the first input pattern ends. Then, the key equation solver starts working. Note that the syndrome updater computes the updated syndrome polynomial at the same time, which ensures that the next syndrome polynomial $S^{(i+1)}(x)$ is ready while key equation solver finishes its previous work. After a series of calculation of Chien search and error value evaluation, the simplified decision making unit computes the Hamming distance and updates the estimated codeword $C'(x)$ when each candidate codeword arrives. Total 8 candidate codewords will be generated since η equals 3 in our proposed design. Therefore, the final estimated codeword is determined and outputted after the 8th candidate codeword arriving. In our design, each block is well-arranged and takes only 16 cycles, leading to 176 cycles decoding latency except for the syndrome calculation procedure.

Fig. 5.8 analyzes the required bit-width for the proposed decision-eased soft RS (224, 216; 4) decoder with 3 LRPs. The simulation parameters *fixed point (A,B)* represents A-bit input quantization with B-bit decimal fraction. The BER curves indicate that the proposed soft RS decoder with 6-bit input quantization can provide around 0.5 dB coding gain at 10^{-5} BER over the hard RS decoder as the floating point soft RS decoder does. Notice that the input quantization of the proposed soft RS decoder only affects the size of the comparators and registers in reliability evaluator. Consequently, it has little influence on

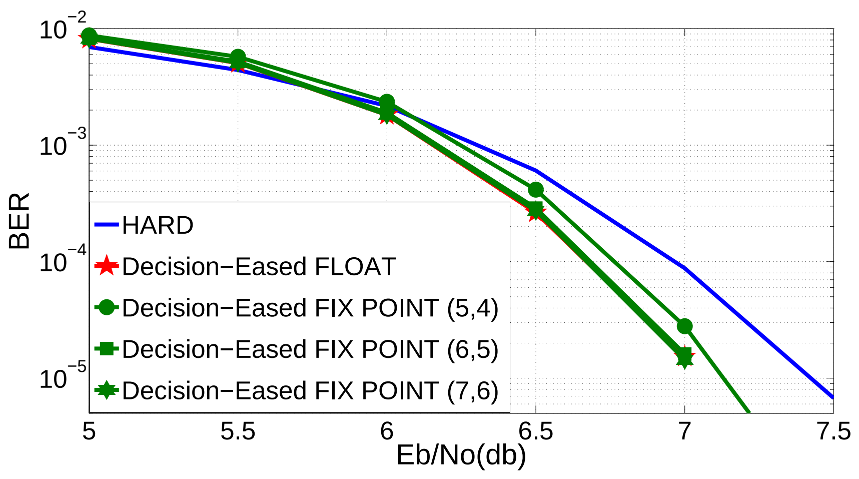


Figure 5.8: Fixed point simulation results for Soft RS (224, 216; 4) with $\eta = 3$

Table 5.4: Implementation Results and Comparision

	Decision-Eased	pRiBM [75]	pDCME [76]	DCME [77]
Code Type	Soft RS (224, 216)	Hard RS (255, 239)	Hard RS (255, 239)	Hard RS (255, 239)
Technology	90 nm	90 nm	0.13 μm	0.25 μm
Operation Frequency	312.5 MHz (Post Layout)	690 MHz (Synthesis)	660 MHz (Synthesis)	200 MHz (Synthesis)
Core Area	221,030 μm^2	-	-	-
Gate Count * (Normalized)	30.0 K (1.42)	43.6 K (1.03)	53.2 K (1.26)	42.2 K (1)
Throughput	2.5 Gb/s	5.52 Gb/s	5.28 Gb/s	1.6 Gb/s
Coding Gain	0.5 dB @ 10^{-5} BER	-	-	-
Power	23.8 mW	-	-	-

* The normalized gate count is calculate as: $\frac{(gate\ count)\times(\frac{8}{t})}{(gate\ count\ of\ DCME)}$.

hardware complexity.

TABLE 5.4 illustrates the implementation result of our soft RS decoder as well as three hard RS decoders for comparison. From the post-layout simulation, our design can achieve 2.5 Gb/s throughput with gate count of 30 K in 90nm 1P9M CMOS process. Moreover, it can fit well for 2.25 Gb/s throughput requirement in mmWave system with 0.5 dB coding gain over hard decoders at 10^{-5} BER. Note that, as considered the difference of the error correcting ability among these designs, a normalized area is defined as $\frac{(gate\ count)\times(\frac{8}{t})}{(gate\ count\ of\ DCME)}$ for a fair comparison. From [34], the LCC based soft RS decoder has about three times area in contrast to the hard decoder. However, our proposed decision-eased soft RS decoder is only 1.38 and 1.42 times as compared to pRiBM and DCME based hard decoders respectively.

5.4 Decision-Confined Soft RS (255, 239; 8) Decoder for Optical Communications

According to International Telecommunication Union (ITU-T) recommendation, RS (255, 239; 8) is standardized in the high speed optical fiber system [78], which provides 2.5 Gb/s throughput with 16 RS decoders as shown in Fig. 5.9 and Fig. 5.10. However, in the near future, the optical fiber system will be demanded to achieve 10 $\sim$ 40 Gb/s throughput, implying that a RS decoder is demanded to provide 2.5 Gb/s throughput. In addition, the 2.5 Gb/s throughput can also satisfy the maximum up and down link requirement in Gigabit Passive Optical Network (GPON) [79]. However, the high throughput will damage the reliability of the transmitted data. In this section, a low complexity and high throughput soft RS decoder is designed for optical communications.

For the 2.5 Gb/s requirement of the optical communication systems, a soft RS (255, 239; 8) decoder with 3-stage pipeline scheme is presented in Fig. 5.11 based on our decision-confined decoding algorithm. At the first stage, the reliability evaluator computes 5 LRPs and the syndrome calculator counts the syndrome $S(x)$ in the meantime. Then the syndrome

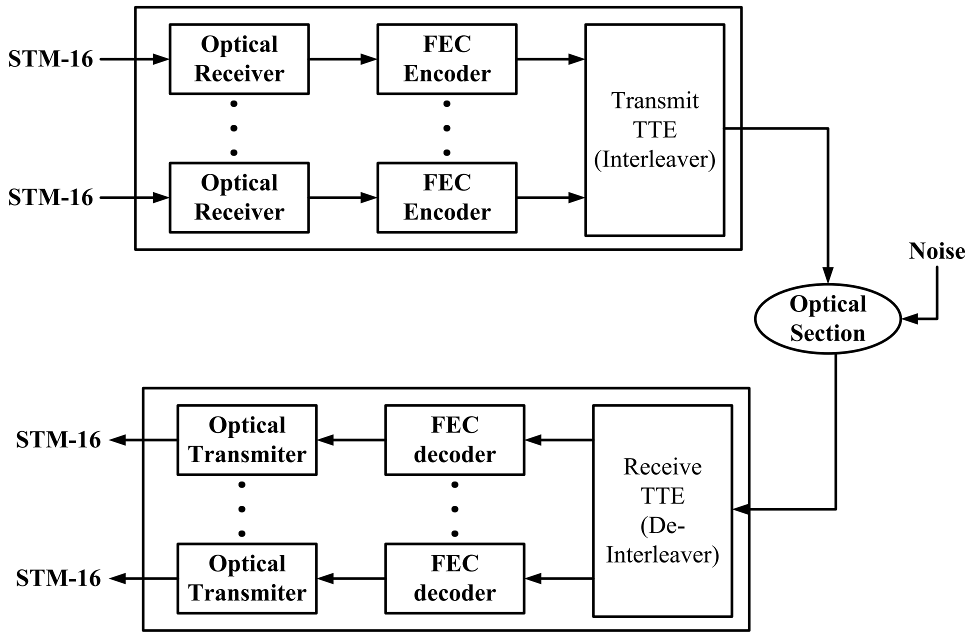


Figure 5.9: Block diagram of G.975

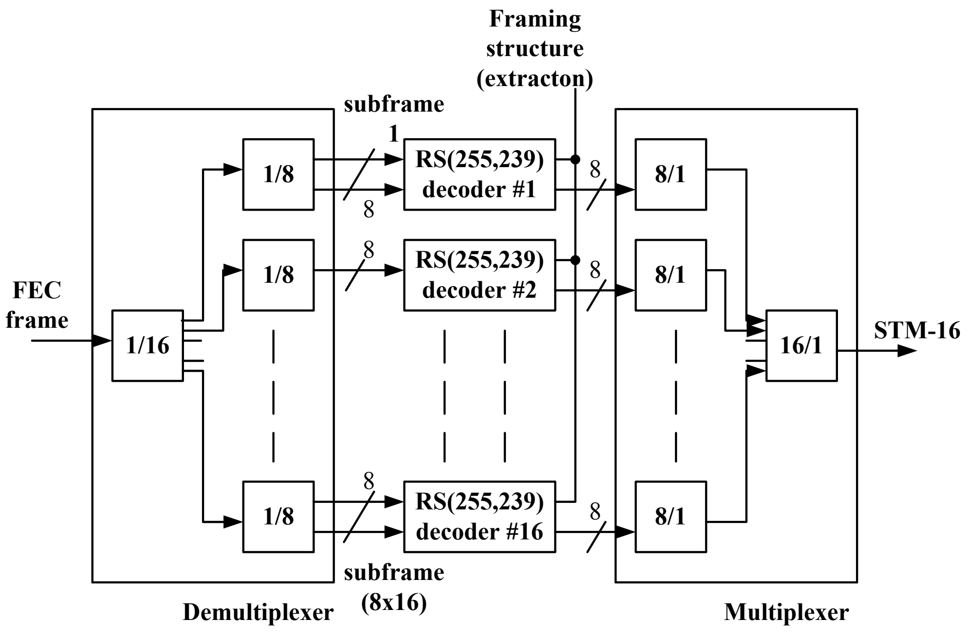


Figure 5.10: Block diagram of G.975 - forward error correction architecture

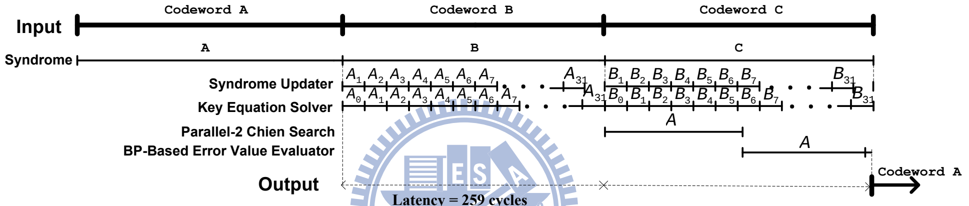


Figure 5.11: Decoding scheme of the decision-confined soft RS decoder

updater iteratively modifies the syndromes $S^{(i)}(x)$ according to the LRPs with Gray code based bit-flipping method. The half iteration key equation solver solves the corresponding $\Lambda^{(i)}(x)$ within 8 clock cycles. As long as the degree of $\Lambda^{(i)}(x)$ is less than t, the parallel-2 Chien search and the BP-based error value evaluator will be applied to find the error locations as well as error values at the last stage. According to our timing schedule, each pipeline stage has been slightly enlarged from 255 to 259 clock cycles because of the 3-stage reliability evaluation.

Fig. 5.12 analyzes the required bit-width for the proposed decision-confined soft RS (255, 239; 8) decoder with 5 LRPs. The CER curves indicate that the proposed soft RS decoder with 6-bit input quantization is sufficient to achieve similar performance to the floating point soft RS decoder. Notice that the input quantization of the proposed soft RS decoder only affects the size of the comparators and registers in reliability evaluator. Consequently, it has

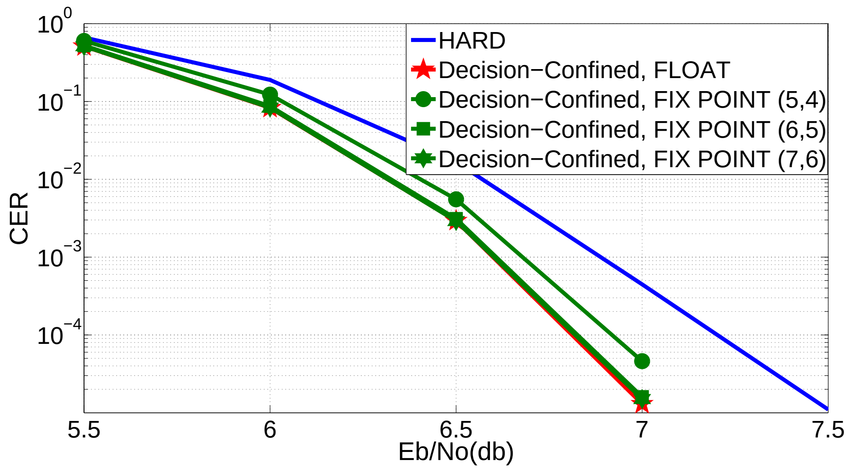


Figure 5.12: Fixed point simulation results for soft RS (255, 239; 8) with $\eta = 5$

Table 5.5: Implementation Results and Comparision

	Decision-Confined	pRiBM [75]	pDCME [76]	DCME [77]
Code Type	Soft RS (255, 239)	Hard RS (255, 239)	Hard RS (255, 239)	Hard RS (255, 239)
Technology	90 nm	90 nm	0.13 μm	0.25 μm
Operation Frequency	320MHz (Measurement)	690MHz (Synthesis)	660MHz (Synthesis)	200MHz (Synthesis)
Core Area	216,225 μm^2	-	-	-
Gate Count	45.3 K	43.6 K	53.2 K	42.2 K
Throughput	2.56 Gb/s	5.52 Gb/s	5.28 Gb/s	1.6 Gb/s
Coding Gain	0.4 dB @ 10^{-4} CER	-	-	-
Power	19.6 mW	-	-	-

little influence on hardware complexity.

Fig. 5.13 shows the decision-confined soft RS (255, 239; 8) decoder die photo, which is implemented with the cell-based design flow and fabricated in 90nm 1P9M CMOS process. The chip is verified by Agilent 93000 SOC test system and the Shmoo plot shown in Fig. 5.14 indicates that our design can achieve 320 MHz operation frequency at 0.98 V supply.

In our understanding, our decoder chip is the first soft RS decoder chip. Hence, TABLE 5.4 illustrates the implementation results of our soft RS decoder with other hard RS decoders. Our chip with 45.3K gates is comparable with a conventional hard decoder. Moreover, it can fit well for 10-40 Gb/s with 16 RS decoders in optical fiber systems and 2.5 Gb/s GPON applications with 0.4 dB coding gain over hard decoders at 10^{-4} CER.

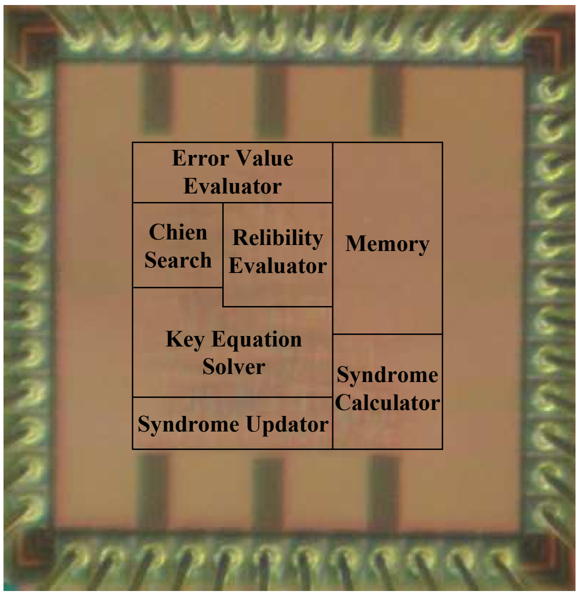


Figure 5.13: Microphoto of decision-confined soft RS (255, 239; 8) chip

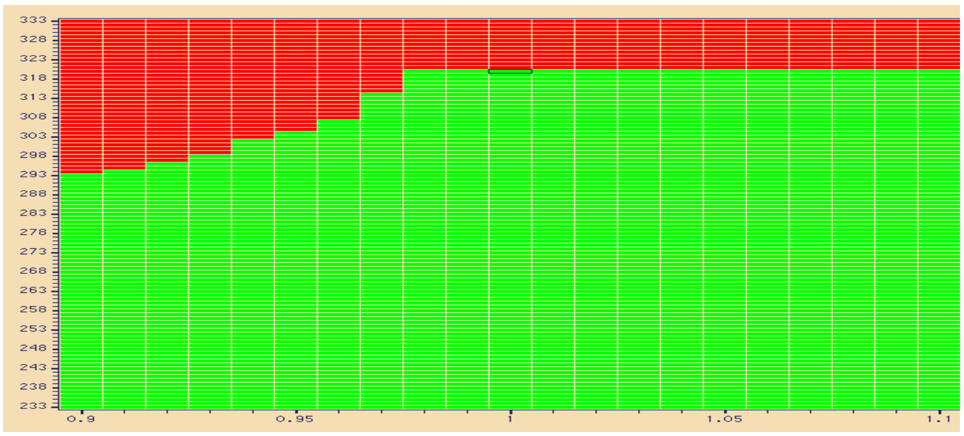


Figure 5.14: Shmoo Plot of decision-confined soft RS (255, 239; 8) chip

Chapter 6

Conclusion

The research on the area-efficient soft BCH and RS decoder design and implementation is reported in this dissertation. We investigate the error magnitude (EM) type decoders for BCH codes and the Chase-Type decoders for both BCH and RS codes.

6.1 Summary

A 26.9 K 314.5 Mb/s soft BCH (32400, 32208; 12) decoder chip is designed for DVB-S2 system. With the EM-type approach, the proposed decoder architecture not only deals with the least reliable bits to reduce complexity but also utilizes the single stage pipeline to minimize the memory bank usage. The proposed error locator evaluator eliminates Chien search to ensure sufficient throughput without parallelism. As compared with the conventional hard BCH decoder, our BCH decoder with soft information from LDPC decoder provides similar system performance. From the measurement results, the proposed soft BCH decoder can achieve 314.5 Mb/s with 50.0% gate-count reduction in contrast to a 99.3 Mb/s traditional hard BCH decoder in CMOS 90 nm technology. While extended to fully support 21 modes in DVB-S2 system, the proposed design can operate 300 MHz frequency with 32.4 K gate-count.

To enhance the error correcting performance, improved EM-type soft decoders with one extra error compensation are proposed. Based on the reformulation of generator polynomial, the BCH-Like codes are also defined to enhance code rate, resulting in higher channel usage efficiency. Compared to the conventional hard BCH decoder, the proposed soft BCH and BCH-Like decoders not only achieve better error correcting performance but also provide competitive hardware complexity. The decoders with soft information can reduce hardware complexity by focusing on the least reliable bits. Meanwhile, the error correcting ability is improved with one extra error compensation. From the experimental results, the proposed soft BCH (255, 239; 2) and BCH-Like (255, 237; 3) decoders can obtain 0.75 dB and 0.47 dB coding gain over the hard BCH (255, 239; 2) and (255, 231; 3) decoders at 10^{-5} BER respectively. The soft BCH-Like (63, 49; 3) decoder with higher code rate can also improve the coding gain to be 1.2 dB closer to Shannon limit as compared with the hard BCH (63, 45; 3) decoder. According to the post-layout simulation in 90 nm CMOS technology, the proposed soft decoders can achieve at most 1.9 times throughput enhancement and 5% gate count reduction as compared with the traditional hard BCH decoders.

Since the EM-type methods only can provide better error correcting performance for BCH codes while applied in small error correcting codes or with reliable information. To provide more general low complexity decoding methods for both BCH and RS codes, the Chase-type soft decoders are proposed. From the simulation, the proposed soft (224, 216; 4) decoder has 0.5 dB coding gain at 10^{-5} BER as compared with the conventional hard decoder. In addition, a decision-eased soft RS (224, 216; 4) decoder is designed in CMOS 90 nm technology for the mmWave system. The rescheduled decoding scheme allows for generating 8 candidate codewords with only one suit hardware, leading to significantly hardware complexity reduction. Moreover, determining the output codeword with the Hamming distance calculation avoids the complex computation of Euclidean distance calculation. According to the post-layout simulation, the proposed soft RS decoder can achieve 2.5 Gb/s throughput with gate count of 30 K, which is only 1.38 times the normalized area of a pRiBM based hard RS decoder. As a result, it is very suitable for practical applications.

In addition, the decision-confined decoder is proposed for soft BCH and RS codes. By confining the degree of error location polynomial, our approach only needs to completely decode one candidate sequence unlike conventional Chase-type methods using several hard RS decoders and determining the most possible candidate. From the simulation, the proposed soft (255, 239; 8) decoder can achieve 0.4 dB coding gain at 10^{-4} CER over hard decoders. A decision-confined soft RS (255, 239; 8) decoder is implemented in CMOS 90 nm technology for optical communications. According to the measurement results, the proposed soft RS decoder can operate at 320 MHz frequency to achieve 2.56 Gb/s throughput with 45.3 K gates, where the hardware complexity is comparable with the hard RS decoders. As a result, our proposal can provide more powerful error correcting ability with a high-speed and area efficient solution for optical communications.

6.2 Future Work

In this dissertation, we have discussed two EM-types soft decoders for BCH codes and two Chase-type soft decoders for both BCH and RS codes. Although the proposed decision-confined approach can support both BCH and RS codes with arbitrary error correcting ability and lead the soft decoder to have competitive hardware complexity as compared to the hard decoder, there are still two design challenges demanded to be improved. The first one is to improve the throughput. Due to the doubled operations in the key equation solver, which leads to half-reduced decoding iterations, the critical path of the decision-confined soft decoder is nearly twice as long as the hard decoders, resulting the throughput reduction. If we could modify the decoding criterion for making the soft decoder provide similar performance with fewer number of LRPs, the key equation solver will not need to execute doubled operations and the critical path will be half-reduced. The second challenge is the performance gain provided by our approach for larger t, like $t = 24$, is not as obvious as for codes with smaller t, like $t = 8$. This is because the Chase-type decoding algorithms enhance the performance by flipping η LRPs to generate candidate codewords and the maximum number of errors can be corrected is $t + \eta$. If the ratio between η and t is decreasing due to the increasing t, the performance enhancement will be smaller. To overcome this situation, we would like to investigate the soft-input soft-output decoding algorithms, like belief propagation algorithm which is often used in LDPC decoding.

References

[1] C. E. Shannon, “A mathematical theory of communication,” *Bell Syst. Tech. J.*, vol. 27, pp. 379–428(Part I), Jul. 1948.

[2] ——, “A mathematical theory of communication,” *Bell Syst. Tech. J.*, vol. 27, pp. 623–656(Part II), July 1948.

[3] S. Lin and D. J. Costello, Jr., *Error control coding: fundamentals and applications*, 2nd ed. Englewood Cliffs, NJ: Pearson-Hall, 2004.

[4] R. J. McEliece, *The theory of information and coding*, 2nd ed. Cambridge, UK: Cambridge University Press, 2004.

[5] P. Elias, “Coding for noisy channels,” *IRE Conv. Rec.*, vol. pt.4, pp. 37–47, 1955.

[6] J. M. Wozencraft and B. Reiffen, *Sequential decoding*. Cambridge: MIT Press and John Wiley, 1961.

[7] A. J. Viterbi, “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm,” *IEEE Trans. Inform. Theory*, vol. IT-13, no. 2, pp. 260–269, Apr. 1967.

[8] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*, IEEE Std. 802.11a, 1999.

[9] *Physical Layer Standard for cdma2000 Spread Spectrum Systems*, 3GPP2 Std. C.S0002-D, Rev. 1.0, 2004.

[10] *High Rate Ultra Wideband PHY and MAC Standard*, ECMA Std. ECMA-368, 2005.

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

[12] R. W. Hamming, “Error detecting and error correcting codes,” *Bell Syst. Tech. J.*, vol. 29, pp. 147–160, Apr. 1950.

[13] D. E. Muller, “Applications of boolean algebra to switching circuits design and to error detection,” *IRE Trans.*, vol. EC-3, pp. 6–12, Sept. 1954.

[14] I. S. Reed, “A class of multile-error-correcting codes and the decoding scheme,” *IRE Trans.*, vol. IT-4, pp. 38–49, Sept. 1954.

[15] R. C. Bose and D. K. Ray-Chaudhuri, “On a class of error correcting binary group codes,” *Inform. Control*, vol. 2, pp. 68–69, Mar. 1960.

[16] A. Hocquenghem, “Codes corecteurs d’erreurs,” *Chiffres*, vol. 2, pp. 147–156, 1959.

[17] I. S. Reed and G. Solomon, “Polynomial codes over certain finite fields,” *J. Soc. Ind. Appl. Math.*, vol. 8, pp. 300–304, June 1960.

[18] D. C. Gorenstein and N. Zierler, “A class of cyclic linear error-correcting codes in p^m symbols,” *J. Soc. Indust. Appl. Math.*, vol. 9, pp. 207–214, June 1961.

[19] *Forward error correction for submarine systems*, ITU-T Std. G.975, 2000.

[20] *Forward Error Correction for high bit-rate DWDM Submarine System*, ITU-T Std. G.975.1, 2004.

[21] *Digital Video Broadcasting (DVB): Framing Structure, Channel Coding and Modulation for 11/12 GHz Satellite Services*, ETSI Std. EN 300 421 v1.1.2, 1997.

[22] *Digital Video Bracasting (DVB) Second Generation System for Broadcasting, Interactive Services, News Gathering and Other Broadband Satellite Applications*, ETSI Std. En 302 307, 2005.

[23] *Framing Structure, Channel Coding and Modulation for Digital Television Terrestrial Broadcasting System*, NSPRC Std. GB 20 600-2006, 2007.

[24] *Digital Video Broadcasting (DVB); Frame structure channel coding and modulation for a second generation digital terrestrial television broadcasting system (DVB-T2)*, ETSI Std. EN 302 755 V1.2.0b, 2008.

[25] G. D. Forney, “Generalized minimum distance decoding,” *IEEE Trans. Inform. Theory*, vol. 12, p. 125V131, Apr. 1966.

[26] D. Chase, “A class of algorithms for decoding block codes with channel measurement information,” *IEEE Trans. Inform. Theory*, vol. IT-18, p. 170V182, Jan. 1972.

[27] H. Tang, Y. Liu, M. Fossorier, and S. Lin, “On combining Chase-2 and GMD decoding algorithms for nonbinary block codes,” *IEEE Commun. Lett.*, vol. 5, no. 5, pp. 209 –211, May 2001.

[28] R. Koetter and A. Vardy, “Algebraic soft-decision decoding of $reed - solomon$ codes,” *IEEE Trans. Inform. Theory*, vol. 49, no. 11, pp. 2809–2825, Nov. 2003.

[29] V. Guruswami and M. Sudan, “Improved decoding of Reed-Solomon and algebraic-geometry codes,” *IEEE Trans. Inform. Theory*, vol. 45, no. 6, pp. 1757–1767, Sept. 1999.

[30] C. Hartmann and L. Rudolph, “An optimum symbol-by-symbol decoding rule for linear codes,” *IEEE Trans. Inform. Theory*, vol. 22, no. 5, pp. 514–517, Sept. 1976.

[31] F. Therattil and A. Thangaraj, "A low-complexity soft-decision decoder for extended BCH and RS-Like codes," in *IEEE Int. Symp. on Info. Theory. (ISIT)*, Sept. 2005, pp. 1320–1324.

[32] J. Jiang and K. Narayanan, "Iterative soft-input soft-output decoding of Reed-Solomon codes by adapting the parity-check matrix," *IEEE Trans. Inform. Theory*, vol. 52, no. 8, pp. 3746–3756, Aug. 2006.

[33] W. J. Gross, F. R. Kschischang, R. Koetter, and P. G. Gulak, "Towards a VLSI architecture for interpolation-based soft-decision Reed-Solomon decoders," *J. VLSI Signal Process*, vol. 39, pp. 93–111, 2005.

[34] X. Zhang, "High-speed VLSI architecture for low-complexity Chase soft-decision Reed-Solomon decoding," in *IEEE Info. Theory and Application Workshop. (ITA)*, Feb. 2009, pp. 422–430.

[35] A. Ahmed, R. Koetter, and N. Shanbhag, "VLSI architectures for soft-decision decoding of Reed-Solomon codes," *IEEE Trans. Inform. Theory*, vol. 57, no. 2, pp. 648 –667, Feb. 2011.

[36] R. J. McEliece, *Finite field for computer scientists and engineers*. Boston: Kluwer Academic, 1987.

[37] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes*. Amsterdam: North-Holland, 1977.

[38] W. W. Peterson, "Encoding and error-correction procedures for the Bose-Chaudhuri codes," *IRE Trans. Inform. Theory*, vol. IT-6, pp. 459–470, Sept. 1960.

[39] E. R. Berlekamp, *Algrbraic coding theory*. New York: McGraw-Hill, 1968.

[40] ——, "On decoding binary Bose-Chaudhuri-Hocquenghem codes," *IEEE Trans. Inform. Theory*, vol. IT-11, pp. 577–579, Oct. 1965.

[41] J. Massey, "Step-by-step decoding of the Bose-Chaudhuri-Hocquenghem codes," *IEEE Trans. Inform. Theory*, vol. IT-11, pp. 580–585, Oct. 1965.

[42] ——, "Shift-register synthesis and bch decoding," *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 122–127, Jan. 1969.

[43] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, "A method for solving key equation for decoding Goppa codes," *Inform. Contr.*, vol. 27, pp. 87–99, Jan. 1975.

[44] L. Welch and R. Scholtz, "Continued fractions and Berlekamp's algorithm," *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 19–27, Jan. 1979.

[45] R. Chien, "Cyclic decoding procedure for the Bose-Chaudhuri-Hocquenghem codes," *IEEE Trans. Inform. Theory*, vol. IT-10, pp. 357–363, Oct. 1964.

[46] G. D. Forney, Jr., "On decoding BCH codes," *IEEE Trans. Inform. Theory*, vol. IT-11, pp. 549–557, Oct. 1965.

[47] H. C. Chang, "Research on Reed-Solomon decoder-design and implementation," Ph.D. dissertation, National Chiao Tung Univ., Hsinchu, Taiwan, 2002.

[48] G. Feng and K. Tzeng, "A generalization of the Berlekamp-Massey algorithm for multisequence shift-register synthesis with applications to decoding cyclic codes," *IEEE Trans. Inform. Theory*, vol. 37, pp. 1274–1287, Sept. 1991.

[49] W. Gross, F. Kschischang, R. Koetter, and M. Sudan, "A VLSI architecture for interpolation in soft-decision list decoding of Reed-Solomon code," in *IEEE Workshop Signal Process Syst. (SiPS)*, Oct. 2002, pp. 39–44.

[50] R. Roth and G. Ruckenstein, "Efficient decoding of Reed-Solomon codes beyond half the minimum distance," *IEEE Trans. Inform. Theory*, vol. 46, no. 1, pp. 246 –257, Jan. 2000.

[51] R. Koetter, J. Ma, A. Vardy, and A. Ahmed, "Efficient interpolation and factorization in algebraic soft-decision decoding of Reed-Solomon codes," in *IEEE Int. Symp. on Info. Theory. (ISIT)*, 2003.

[52] R. Koetter and A. Vardy, "A complexity reducing transformation in algebraic list decoding of Reed-Solomon codes," in *IEEE Info. Theory Workshop. (ITW)*, March 2003, pp. 10 – 13.

[53] J. Bellorado and A. Kavcic, "A low-complexity method for Chase-Type decoding of Reed-Solomon codes," in *IEEE Int. Symp. on Info. Theory. (ISIT)*, 2006, pp. 2037 –2041.

[54] J. Zhu, X. Zhang, and Z. Wang, "Novel interpolation architecture for low-complexity Chase soft-decision decoding of Reed-Solomon codes," in *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, May 2008, pp. 3078 –3081.

[55] J. Zhu and X. Zhang, "Factorization-free low-complexity Chase soft-decision decoding of Reed-Solomon codes," in *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, May 2009, pp. 2677 –2680.

[56] W. J. ReidIII, L. L. Joiner, and J. J. Komo, "Soft decision decoding of BCH codes using error magnitudes," in *IEEE Int. Symp. on Info. Theory. (ISIT)*, June 1997, p. 303.

[57] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. 20, pp. 284–287, Mar. 1974.

[58] Y.-M. Lin, C.-L. Chen, H.-C. Chang, and C.-Y. Lee, "A 26.9 K 314.5 Mb/s soft (32400, 32208) BCH decoder chip for DVB-S2 system," *IEEE J. Solid-State Circuits*, vol. 45, pp. 2330–2340, Nov. 2010.

[59] A. Björck and V. Pereyra, "Solution of Vandermonde systems of equations," *Math. Computation*, vol. 24, pp. 893–903, Oct. 1970.

[60] J. Hong and M. Vetterli, "Simple algorithms for BCH decoding," *IEEE Trans. Commun.*, vol. 43, pp. 2324–2333, Aug. 1995.

[61] P. E. McIllree, “Channel capacity calculations for M-ary N-dimensional signal sets,” Master thesis, University of South Australia, 1995.

[62] C. Parr, “Efficient VLSI architectures for bit-parallel computation in Galois fields,” Ph.D. dissertation, Inst. for Experimental Mathematics of Univ. of Essen Germany, 1994.

[63] I. S. Reed, M. T. Shih, and T. K. Truong, “VLSI design of inverse-free Berlekamp-Massey algorithm,” in *IEEE Proc. Inst. Elect. Eng*, vol. 138, Sept. 1991, pp. 295–298.

[64] W. Liu, J. Rho, and W. Sung, “Low-power high-throughput BCH error correction VLSI design for multi-level cell NAND flash memories,” in *IEEE Workshop Signal Process Syst. (SiPS)*, 2006, pp. 303 –308.

[65] D. Knuth, *The Art of Computer Programming, Vol. 3, Sorting and Searching*, 2nd ed. Addision-Wesley, 1998.

[66] A. Raghupathy and K. J. R. Liu, “Algorithm-based low-power/high-speed Reed-Solomon decoder design,” *IEEE Trans. Circuits Syst. II*, vol. 47, no. 11, pp. 1254–1270, Nov. 2000.

[67] D. Sarwate and N. Shanbhag, “High-speed architectures for Reed-Solomon decoders,” *IEEE Trans. VLSI Syst.*, vol. 9, no. 5, pp. 641–655, Oct. 2001.

[68] J. Cho and W. Sung, “Strength-reduced parallel Chien search architecture for strong BCH codes,” *IEEE Trans. Circuits Syst. II*, vol. 55, no. 5, pp. 427–431, May 2008.

[69] H.-C. Chang, C.-C. Lin, and C.-Y. Lee, “A low-power Reed-Solomon decoder for STM-16 optical communications,” in *IEEE Asia-Pacific Conf. on ASIC (APASIC)*, 2002, pp. 351 – 354.

[70] L. Song, M.-L. Yu, and M. Shaffer, “10- and 40-Gb/s forward error correction devices for optical communications,” *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1565 – 1573, Nov. 2002.

[71] Y. Chen and K. Parhi, “Small area parallel Chien search architectures for long BCH codes,” *IEEE Trans. VLSI Syst.*, vol. 12, no. 5, pp. 545–549, May 2004.

[72] Y.-M. Lin, J.-Y. Wu, C.-C. Lin, and H.-C. Chang, “A long block length BCH decoder for DVB-S2 application,” in *IEEE Int. Symp. on Integrated Circuits (ISIC)*, Dec. 2009, pp. 171 –174.

[73] *High Rate Short Range Wireless Communication*, ECMA TC32-TG20 Std., [online]. Available:http://www.ecma-international.org/memento/TC-32-TG20-M.htm/.

[74] *IEEE 802.15 WPAN Millimeter-wave Alternative PHY Task Group 3c (TG3c).*, IEEE Std. Draft Document, 2009.

[75] J.-I. Park, K. Lee, C.-S. Choi, and H. Lee, “High-speed low-complexity Reed-Solomon decoder using pipelined Berlekamp-Massey algorithm,” in *IEEE Int. SoC Design Conference (ISOCC)*, 2009, pp. 452 –455.

[76] S. Lee, H. Lee, J. Shin, and J.-S. Ko, “A high-speed pipelined degree-computationless modified Euclidean algorithm architecture for Reed-Solomon decoders,” in *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, May 2007, pp. 901 –904.

[77] J. Baek and M. Sunwoo, “New degree computationless modified Euclid algorithm and architecture for Reed-Solomon decoder,” *IEEE Trans. VLSI Syst.*, vol. 14, no. 8, pp. 915 –920, Aug. 2006.

[78] *Forward Error Correction for Submarine Systems*, ITU-T Std. G.975, 1996.

[79] *Gigabit-capable Passive Optical Networks (G-PON): Transmission convergence layer specification*, ITU-T Std. G.984.3, 2008.