

國立交通大學

電子工程學系 電子研究所碩士班

碩 士 論 文

應用於光通訊的受限選擇軟性 RS 解碼器



**A Decision-Confined Soft Reed-Solomon Decoder for
Optical Communication Systems**

學生：許智翔

指導教授：李鎮宜教授

中華民國一〇〇年八月

應用於光通訊的受限選擇軟性 RS 解碼器

A Decision-Confined Soft Reed-Solomon Decoder for Optical Communication Systems

研究生：許智翔

Student : Chih-Hsiang Hsu

指導教授：李鎮宜教授

Advisor : Chen-Yi Lee



A Thesis

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

Master of Science

in

Electronics Engineering

September 2011

Hsinchu, Taiwan, Republic of China

中華民國一〇〇年八月

應用於光通訊的受限選擇軟性 RS 解碼器

學生：許智翔

指導教授：李鎮宜 教授

國立交通大學

電子工程學系 電子研究所碩士班

摘要

在光通訊系統中，因為更加快速的傳輸速度而增加的訊號不穩定性，錯誤更正裝置需要提供更加有力的錯誤更正能力。相較於傳統硬性RS解碼器，軟性解碼器可以提供更好的錯誤更正能力，但也需要高出許多的硬體複雜度。本論文提出了一種受限選擇軟性演算法不但可以增加錯誤更正能力並且保有面積效益。創新的重點在於，不是將許多可能的傳輸訊息解碼再從中挑選出最有可能的一組，而是將錯誤位置數學式 $\Lambda(x)$ 設下限制導致只有解碼其中一組。根據RS(255,239)的模擬結果，在 10^{-4} CER時可比硬性RS解碼器多達0.4 dB傳輸效益。實驗結果顯示我們提出的軟性解碼器在CMOS 90奈米製程下可達到2.56 Gb/s吞吐量並且只需花費與硬性解碼器相似的硬體複雜度。此軟性解碼器可以適用於利用16套解碼器的10-40 Gb/s光纖系統以及2.5 Gb/s GPON等應用。

A Decision-Confined Soft Reed-Solomon Decoder for Optical Communication Systems

Student : Chih-Hsiang Hsu

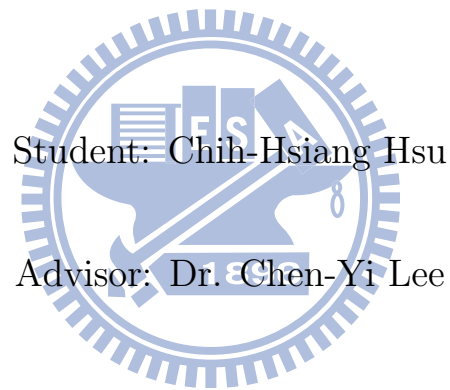
Advisor : Dr. Chen-Yi Lee

Department of Electronics Engineering
Institute of Electronics
National Chiao Tung University

ABSTRACT

Due to the increasing uncertainty of data for higher transmission rate, the Forward Error Correction (FEC) devices need to provide more powerful error correcting capability for optical communication systems. As compared with traditional hard RS decoders, the soft RS decoders can perform substantial coding gain but require much higher hardware complexity. In this thesis, a decision-confined algorithm is proposed to enhance the error correcting performance with an area-efficient architecture. The novelty is that, instead of decoding numerous possible transmitted codewords and choosing the most likely one, only one candidate sequence will be decoded after confining the degree of error-locator polynomial $\Lambda(x)$. For RS (255,239) codes, simulation results confirm that our approach provides 0.4 dB performance gain at 10^{-4} CER over the hard RS decoders. The experimental result reveals that our soft decoder can achieve 2.56 Gb/s throughput in standard CMOS 90 nm technology while having similar complexity as a hard decoder. It can fit well for 10-40 Gb/s with 16 RS decoders in optical fiber systems and 2.5 Gb/s GPON applications.

**A Decision-Confined Soft Reed-Solomon Decoder
for Optical Communication Systems**

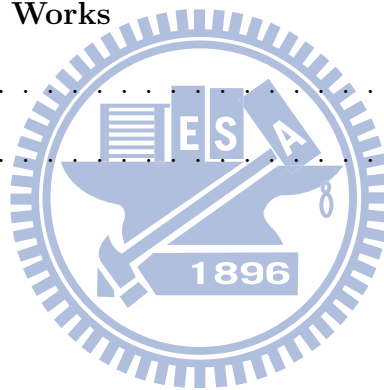


Department of Electronics Engineering and Institute of Electronics
National Chiao Tung University

Contents

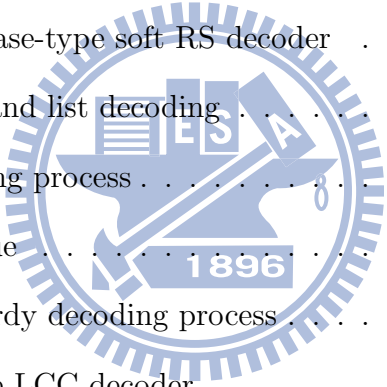
1	Introduction	1
1.1	Research Motivation	1
1.2	Thesis Organization	3
2	Principle of Reed-Solomon Code	4
2.1	Basic Concepts of Finite Fields	4
2.2	RS Code	5
2.2.1	Encoding of RS Code	5
2.2.2	Decoding of RS Code	6
2.3	Soft Decoding Algorithm for Reed-Solomon Code	10
2.3.1	Generalized-Minimum-Distance Algorithm	11
2.3.2	Chase Algorithm	12
2.3.3	Guruswami-Sudan Algorithm	13
2.3.4	Koetter-Vardy Algorithm	16
2.3.5	Low Complexity Chase Algorithm	20
3	Decision-Confined Soft Decoding Algorithm	23
3.1	Decision-Confined Soft Decoding Algorithm	23
3.2	Performance Analysis	27

4	Decision-Confined Soft Reed-Solomon (255,239) Decoder	29
4.1	Decoding Scheme	30
4.2	Proposed VLSI Architecture	32
4.2.1	Syndrome Calculator and Reliability Evaluator	32
4.2.2	Syndrome Updater	32
4.2.3	Half-iteration Key Equation Solver	34
4.2.4	Chien Search and Error Value Evaluator	41
4.3	Implementation Result	43
4.3.1	Hardware Analysis	43
4.3.2	Chip Specification	45
5	Conclusion and Future Works	48
5.1	Conclusion	48
5.2	Future Works	49



List of Figures

1.1	Block diagram of a digital communication system	1
1.2	The limited transport distance for different bit rate	2
2.1	The systemic RS encoder	6
2.2	The decoding flow of RS decoder	7
2.3	Block diagram of Chase-type soft RS decoder	13
2.4	Comparison for BD and list decoding	14
2.5	Kotter-Vardy decoding process	17
2.6	Re-encoding technique	18
2.7	Simplified Kotter-Vardy decoding process	21
2.8	The factorization-free LCC decoder	22
3.1	Simulation results of Chase and simplified Chase algorithm	24
3.2	Proposed soft decoding flow	25
3.3	Performance of the proposed soft decoding algorithm	27
4.1	Block diagram of a submarine system which use a FEC function	30
4.2	FEC decoder architecture of optical communication systems	30
4.3	Decoding scheme of the proposed soft RS decoder	31
4.4	Syndrome calculator	32
4.5	Reliability evaluator	33



4.6	Merge sorter	33
4.7	Gray code example	34
4.8	Syndrome updater	34
4.9	The updating criterion for half iteration BM algorithm	36
4.10	The processing element (H-PE) of half-iteration RiBM	38
4.11	The homogeneous architecture of half-iteration RiBM	39
4.12	The regular architecture of half-iteration RiBM without the calculation of $\Omega(x)$	41
4.13	Parallel-2 Chien search architecture	42
4.14	BP-based error value evaluator	44
4.15	Microphoto of soft RS (255,239) chip	46
4.16	Shmoo plot of proposed decoder chip	47



List of Tables

2.1	Representation of the elements in $GF(2^4)$	5
3.1	Percentage of degree of $\Lambda(x)$ equals to 8 (10^5 test patterns)	25
3.2	Comparison of Computational Complexity with Soft RS Decoder	28
4.1	Comparison of Time and Hardware Complexity	43
4.2	Comparison of Time and Hardware Complexity with Soft RS Decoder	45
4.3	Measurement Result of Decision-confined Soft RS (255,239) Decoder	46
4.4	Comparison of Time and Hardware Complexity with Hard Decision RS Decoder	47



Chapter 1

Introduction

1.1 Research Motivation

A communication system transmits an information source to a destination through an unknown channel. The typical block diagram of traditional digital communication system is shown in Fig. 1.1. Generally, the communication system can be divided into three component parts which consists of transmitter, receiver, and channel.

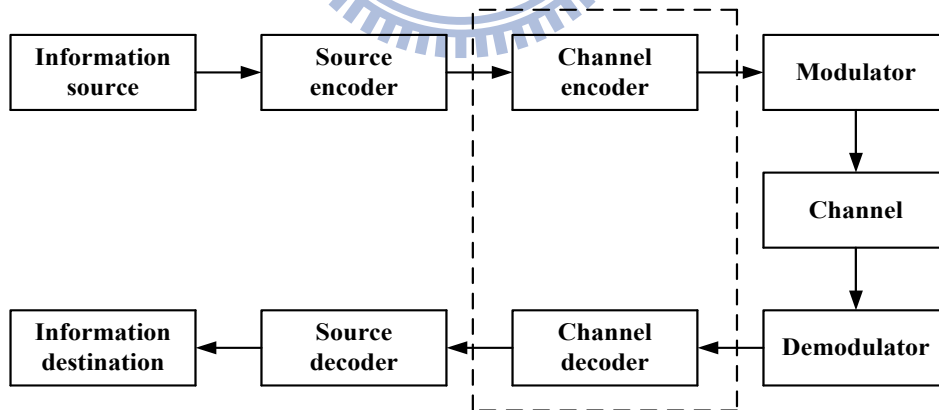


Figure 1.1: Block diagram of a digital communication system

The transmitter mainly includes source encoder, channel encoder, and modulator, which transforms the information into a form that can resist the interference of noise effectively.

Then the receiver will reverse the transformed signal by demodulator, channel decoder, and source decoder. Since the signal may be distorted by the effects such as noise, interference and distortion as it passes the channel, the channel encoder is incorporate to the system to overcome the transmission errors by adding certain redundancy message to the source code-word. These redundant messages can be used for detecting and correcting the errors. Thus, the channel coding resists the effects of noise and provides better performance compared with a simple uncoded communication system.

Reed-Solomon (RS) codes are widely used in various communications and digital data storage systems due to the advantage of overcoming the burst errors [1] [2]. According to International Telecommunication Union (ITU-T) recommendation, RS (255,239) is standardized in high speed optical fiber systems and gigabit passive optical network (GPON) applications [3] [4], which demand 2.5 Gb/s throughput for achieving 2.5, 10 and 40 Gb/s with 16 RS decoders (STM-16, STM-64 and STM-256) or satisfying the maximum up and

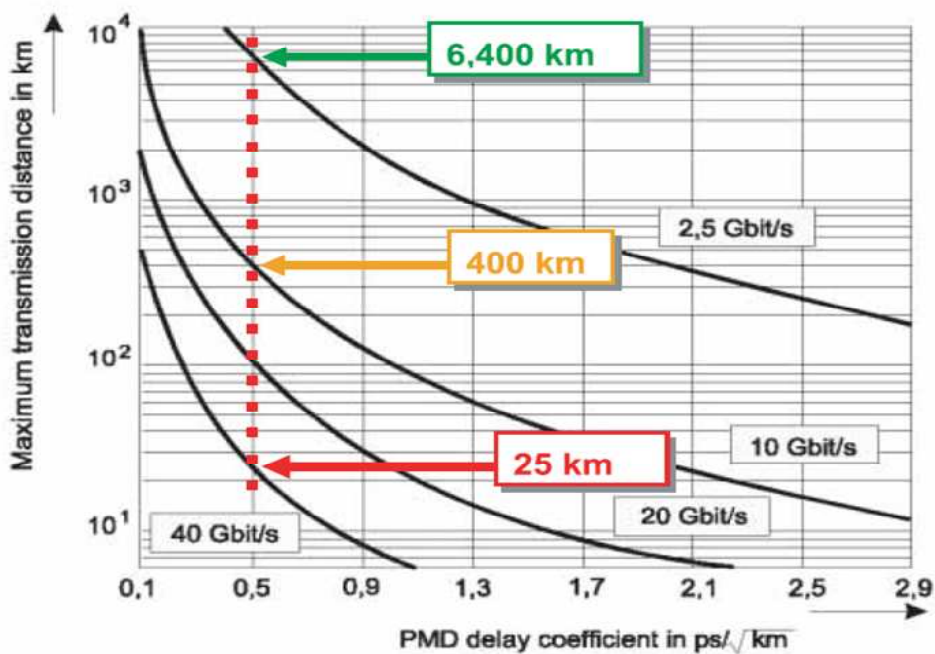


Figure 1.2: The limited transport distance for different bit rate

down link requirement. However, because of the increasing uncertainty of data, the transmission distance is restricted to the higher transmission rate. From [5], Fig 1.2 shows the limited transport distance for different data rate for optical communication systems. It's obvious that the limited transmission distance is decreased as the data rate is increased. As a result, the forward error correction (FEC) devices need to provide more powerful error correcting capability for optical communication systems to support longer transmission distance. As compared with traditional hard RS decoders, the soft RS decoders can perform substantial coding gain but require much higher hardware complexity. So how to enhance performance while maintaining area efficiency is our design goal. In this thesis, a novel decoding algorithm and its area-efficient architecture for soft RS codes are proposed.

1.2 Thesis Organization

This thesis consists of five chapters. In chapter 2, the fundamental knowledge of finite-field arithmetic for RS codes and the procedure of encoding and decoding are introduced. The concepts of several soft decoding algorithms of RS codes will also be discussed. Chapter 3 will characterize the proposed decision-confined algorithm. According to our proposed method, the low complexity VLSI architectures and the corresponding implementation results of the soft RS decoder will be illustrated in chapter 4. Finally, the conclusion is given in chapter 5.

Chapter 2

Principle of Reed-Solomon Code

Reed-Solomon (RS) codes were invented in 1960 by Irving S. Reed and Gustave Solomon [6] and have been widely employed in digital communication systems. The codewords of RS codes consist of non-binary symbol therefore RS codes are suitable for correcting burst errors. In this chapter, the encoding and decoding procedure of a conventional RS system will be introduced in detail following the basic concepts of finite fields. Then we will further talk about the *soft* decoding algorithms for RS codes in the later part of this chapter.

2.1 Basic Concepts of Finite Fields

A finite field, or *Galois* field (GF), is defined as a set of finite number of elements in which the arithmetic including addition, subtraction, multiplication, and division can be operated without leaving the set. A $GF(p)$ can be constructed based on the modulo p arithmetic and has the elements $\{0, 1, \dots, p - 1\}$, where p is usually a prime number. To extend the field $GF(p)$, $GF(p^m)$ for any positive integer m can also be built with p^m elements based on the modulo *primitive polynomial* $f(x)$ operation. Notice that $f(x)$ must be an irreducible m -th degree polynomial over $GF(p)$. In a finite field $GF(p^m)$, a nonzero element α is said to be primitive if the powers of α generate all the nonzero elements of $GF(p^m)$, and then the

p^m elements with m -tuple of the field can be constructed as $\{0, 1, \alpha, \dots, \alpha^{p^m-1}\}$. Since that only the binary arithmetic calculation is required when $p = 2$, the fields based on $GF(2^m)$ are attractive for applications. Table 2.1 shows an example of $GF(2^4)$ which is an extension field of $GF(2)$ and is built with the primitive polynomial $f(x) = x^4 + x + 1$. The primitive element α is a root of $f(x)$ and therefore $f(\alpha) = \alpha^4 + \alpha + 1 = 0$. In Table 2.1, we can find that the power representation is useful for multiplication, while the polynomial or 4-tuple format is more practical for addition.

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	$\alpha^3 + 1$	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

2.2 RS Code

2.2.1 Encoding of RS Code

An (n, k) RS code over $GF(2^m)$ with block length of n symbols and information length of k symbols, where each symbol consists of m bits, can correct up to $t = \lfloor \frac{n-k}{2} \rfloor$ errors. The

generator polynomial $G(x)$ of a RS code can be constructed by the minimum polynomials of $\alpha, \alpha^2, \dots, \alpha^{2t}$, where α is a primitive element. Let $(M_0, M_1, \dots, M_{k-1})$ denote k message symbols, the systematic encoding process is described in terms of the message polynomial $M(x) = M_0 + M_1x + \dots + M_{k-1}x^{k-1}$ being transformed into a codeword polynomial $C(x)$ with the message symbols followed by parity-check symbols $r(x)$.

$$G(x) = (x - \alpha)(x - \alpha^2) \cdots (x - \alpha^{2t}) \quad (2.1)$$

$$r(x) = M(x) \cdot x^{2t} \bmod G(x) \quad (2.2)$$

$$C(x) = M(x) \cdot x^{2t} + r(x) \quad (2.3)$$

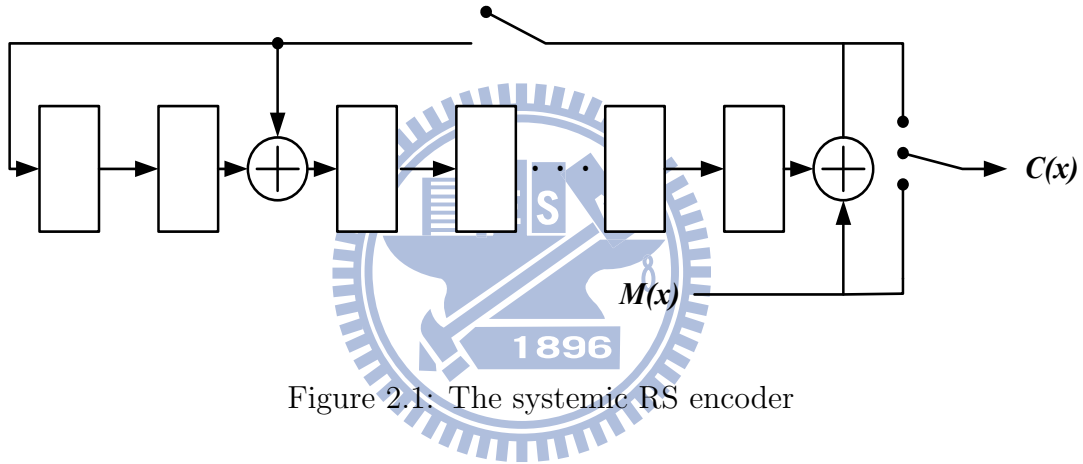


Figure 2.1: The systematic RS encoder

Fig 2.1 is the conventional systematic RS encoder with linear feedback shift register (LFSR) architecture. The combinational logic is depending on $G(x)$ and the feedback loop can be viewed as the *module* $G(x)$ operation in (2.2).

2.2.2 Decoding of RS Code

The conventional hard RS decoder, which is shown in Fig. 2.2, contains three major blocks : *syndrome calculator*, *key equation solver* and *Chien search & error value evaluator*. The received data $R(x)$ is firstly fed into syndrome calculator to generate syndrome polynomial $S(x)$. Then the key equation solver (KES) evaluates the error-locator polynomial $\Lambda(x)$ and

the error evaluator polynomial $\Omega(x)$. Finally, these polynomials are sent to Chien search and error value evaluator to find the error locations and calculate the corresponding error values.

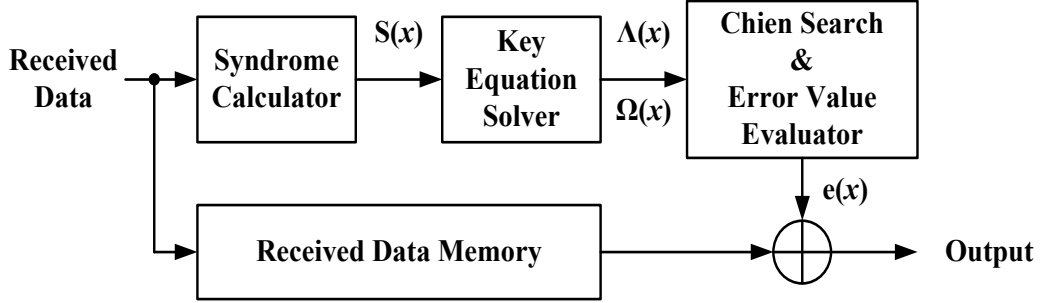


Figure 2.2: The decoding flow of RS decoder

Syndrome Calculator

Since $2t$ consecutive power of α are roots of generator polynomial $G(x)$ and codeword polynomial $C(x)$ is a multiple of the $G(x)$, it follows that

$$C(\alpha^i) = 0, \quad \text{for } 1 \leq i \leq 2t \quad (2.4)$$

for all codeword polynomials $C(x)$. Let $R(x)$ denote the received polynomial which can be viewed as the transmitted codeword polynomial $C(x)$ corrupted by error polynomial $E(x)$; that is $R(x) = C(x) + E(x)$ and $E(x)$ can be written as

$$E(x) = e_1x^{i_1} + e_2x^{i_2} + \cdots + e_vx^{i_v} \quad (2.5)$$

when v error values e_1, e_2, \dots, e_v have occurred at the error locators $X_1 = \alpha^{i_1}, X_2 = \alpha^{i_2}, \dots, X_v = \alpha^{i_v}$. The syndrome polynomial $S(x)$ of $R(x)$ then is defined to be

$$S(x) = S_1 + S_2x + \cdots + S_{2t}x^{2t-1} \quad (2.6)$$

where

$$S_i = R(\alpha^i) = C(\alpha^i) + E(\alpha^i) = \sum_{j=1}^v e_j X_j^i. \quad (2.7)$$

Key Equation Solver

After calculating the syndrome polynomial, the error-locator polynomial $\Lambda(x)$ and the error evaluator polynomial $\Omega(x)$ are determined by solving the *key equation*

$$\Omega(x) = S(x) \times \Lambda(x) \bmod x^{2t} \quad (2.8)$$

where $\Lambda(x)$ is defined as

$$\Lambda(x) = \prod_{j=1}^v (1 - X_j x) = \Lambda_0 + \Lambda_1 x + \cdots + \Lambda_v x^v. \quad (2.9)$$

According to the Newton's identities, the relation between syndrome and error-locator polynomial can be written as

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_v \\ S_2 & S_3 & \cdots & S_{v+1} \\ \vdots & \vdots & \ddots & \vdots \\ S_v & S_{v+1} & \cdots & S_{2v-1} \end{bmatrix} \begin{bmatrix} \Lambda_v \\ \Lambda_{v-1} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} S_{v+1} \\ S_{v+2} \\ \vdots \\ S_{2v} \end{bmatrix} \quad (2.10)$$

Solving the key equation to find the error locator and error evaluator polynomials is the most critical part in the design of RS decoders. Berlekamp-Massey (BM) algorithm [7] is one of the well-known methods by iteratively modifying the minimal polynomial $\Lambda(x)$ to fit the sequences of syndrome. The BM algorithm is described by the pseudocode shown in Algorithm 1. After the $2t$ cycles of computation, the minimal polynomial result will be the $\Lambda(x)$ that fulfills all the syndrome equations. Based on the BM algorithm, the inversionless BM (iBM) algorithm [8] is proposed to replace the divisions with multiplications, leading to smaller critical path delay. In 2001, the reformulated inversionless BM (RiBM) algorithm [9] provided an extremely efficient procedure to calculate $\Lambda(x)$ with a regular architecture and a half critical path compared with the iBM algorithm.

From another approach, the modified Euclidean algorithm [10], which takes the point of view that $\Omega(x)$ is the Greatest Common Divisor (GCD) of x^{2t} and $S(x)$, also provides an

Algorithm 1. : BM Algorithm**Initialization :**

$$L(0) = 0, \gamma(0) = 1, \Lambda_0(0) = B_0(0) = 1, \Lambda_i(0) = B_i(0) = 0, (i = 0, 1, \dots, t)$$

Input : $S_i, (i = 1, 2, \dots, 2t)$ **for** $k = 0$ **to** $2t - 1$ **do****begin**

Step 1. $\Delta_k = \Lambda_0 S_{k+1} + \Lambda_1 S_k + \dots + \Lambda_t S_{k-t+1}$
 $\Lambda_i(k+1) = \Lambda_i(k) - \frac{\Delta_k}{\gamma(k)} \cdot B_{i-1}(k), (i = 0, 1, \dots, t)$

Step 2. **if** $\Delta_k \neq 0$ **and** $2L(k) < (k+1)$ **begin**

$$B_i(k+1) = \Lambda_{i+1}(k), (i = 0, 1, \dots, t)$$

$$\gamma(k+1) = \Delta_k$$

$$L(k+1) = k+1 - L(k)$$

end**else****begin**

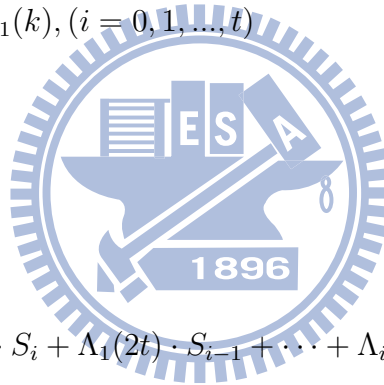
$$B_i(k+1) = B_{i-1}(k), (i = 0, 1, \dots, t)$$

$$\gamma(k+1) = \gamma(k)$$

$$L(k+1) = L(k)$$

end**end****for** $i = 0$ **to** $t - 1$ **do****begin**

Step 4. $\Omega_i(2t) = \Lambda_0(2t) \cdot S_i + \Lambda_1(2t) \cdot S_{i-1} + \dots + \Lambda_i(2t) \cdot S_0$

end**Output :** $\Lambda_i(2t), (i = 0, 1, \dots, t), \Omega_i(2t), (i = 0, 1, \dots, t - 1)$.

efficient way to solve the key equation. After the first appearance of ME algorithm, lots of methods are proposed to improve both the hardware cost and calculation time [11] [12]. The brief procedure of ME algorithm is shown as following description.

- Initial conditions:

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

$$\Lambda^{(-1)}(x) = 0, \Lambda^{(0)}(x) = 1$$

- Iterations from $i = 1$:

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

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

- Iteration terminated:

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

Chien Search

Once the error-locator polynomial $\Lambda(x)$ is known, the Chien search is applied to find the error locators X_i [13]. For instance, if α^{-i} , the inverse of *error locator* X_i , is a root of $\Lambda(x)$, one of the error locators is assumed at the i -th position. Note that in the decoding process, a successful decoding is defined that the number of roots found by Chien search is equal to the degree of $\Lambda(x)$. Otherwise, if the number of roots is less than the degree of $\Lambda(x)$ or $\Lambda(x)$ have repeated roots or illegal roots, the received codeword $R(x)$ is uncorrectable.

Error Value Evaluator

The final step of RS decoding is to calculate the error values for error correction. Conventionally, if X_i is one of the error locators, the decoder can calculate the error value e_i to be subtracted from i -th symbol of $R(x)$, R_i , via Forney's error value formula

$$e_i = \frac{\Omega(X_i^{-1})}{\Lambda^{(1)}(X_i^{-1})} \quad (2.11)$$

where $\Lambda^{(1)}(x) = \Lambda_1 + \Lambda_3x^2 + \dots$ denotes the formal derivative of $\Lambda(x)$.

2.3 Soft Decoding Algorithm for Reed-Solomon Code

Since the initial appearance of RS decoders, much research has been performed for extending the error correction capability. The soft decoding algorithms have been developed by

incorporating the reliability information from the channel. In the rest of this chapter, we will briefly introduce several soft decoding algorithms.

2.3.1 Generalized-Minimum-Distance Algorithm

The Generalized-Minimum-Distance (GMD) algorithm was devised by Forney in 1966 [14]. For RS decoders, a received symbol can be erased when the receiver declares it is interfered or ambiguous. With this method, it's possible to correct more error values since the most likely error locators have been already found, which can be viewed as translating error detecting capability into error correction capability. Accordingly, if ρ symbols are erased, the error-and-erasure decoder with almost the same algorithm can correct t errors in unerased locations in addition to ρ erasures on the condition that

$$2t + \rho < d_{min} \quad (2.12)$$

where d_{min} is the corresponding minimum distance. The GMD algorithm erases $2i$ least reliable symbols as i -th candidate sequence, where $i = 0, 1, \dots, t$, and applies $t + 1$ error-and-erasure decoders to generate all the candidate sequences. After decoding all the candidate sequences, the *decision making unit* (DMU) will determine the most probable one as the output codeword which is with the smallest Euclidean distance between the soft received sequence R . The brief decoding process of GMD algorithm is illustrated as following description.

- LRPs determination:

Determine $d_{min} - 1$ least reliable symbols.

- Candidate codewords generation:

for $i = 0$ to t by 1 : Erase $2i$ least reliable symbols and send it to error-and-erasure decoder.

- Output codeword selection:

Output the decoded codeword with the minimum Euclidean distance between soft received sequence R .

2.3.2 Chase Algorithm

Chase algorithm [15] can be viewed as the generalization of the GMD algorithm. There are three types of Chase algorithm, referred as Chase-1, Chase-2, and Chase-3. Chase-1 algorithm flips all $d_{min} - 1$ least reliable positions (LRPs) [16] and generates at most $C_{\lfloor \frac{d_{min}}{2} \rfloor}^N$ candidate codewords. Chase-3 algorithm does similar operations as the GMD algorithm, except the erasure operation in the GMD is being replaced by flipping the least reliable symbols. For binary codes, Chase-3 algorithm achieves the same error performance as GMD and require same computational complexity.

Chase-2 algorithm can be viewed as an improvement of Chase-3 algorithm, which flips η LRPs to generate 2^η candidate sequences and sends them to the error-only hard RS decoders, where η can be up to d_{min} . Hence totally 2^η error-only hard RS decoders are employed to solve each candidate sequence. After decoding all the candidate sequences, the output codeword will be chosen by the DMU. As considered both error correcting ability and computation complexity, Chase-2 algorithm is the most popular one. Based on the GMD and Chase algorithm, Chase-GMD [17] and successive error-and-ersure decoding (SED) [18] have been proposed to provide additional trade-off between error performance and decoding complexity. Fig. 2.3 is the block diagram of Chase-2 soft RS decoder. The brief decoding procedure of Chase-2 algorithm is illustrated as following description.

- LRPs determination:

Determine η least reliable positions.

- Candidate codewords generation:

for $j = 1$ to 2^n by 1 : Modify the LRPs by complementing one combination of LRPs and send it to hard RS decoder.

- Output codeword selection:

Output the decoded codeword with the minimum Euclidean distance between soft received sequence R .

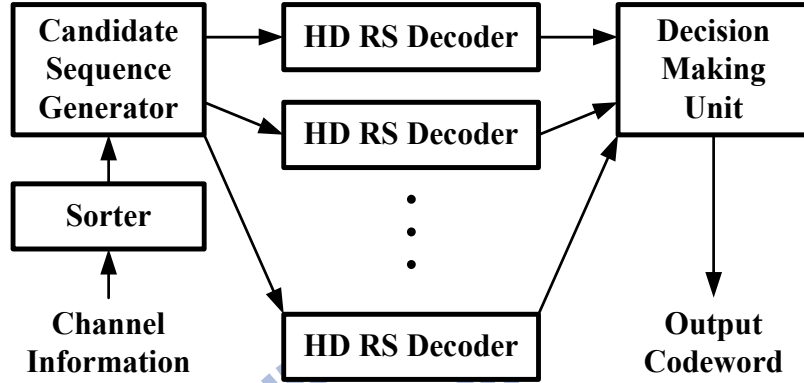


Figure 2.3: Block diagram of Chase-type soft RS decoder

2.3.3 Guruswami-Sudan Algorithm

Unlike GMD and Chase algorithms, Guruswami-Sudan (GS) algorithm [19] provides list decoding method to enhance the performance by stretching the error correction ability and finding out all the probable codewords within its extended decoding sphere. From Fig. 2.4, compared with the original bounded distance (BD) decoding, the list decoding can correct up to $\lceil n - \sqrt{nk} - 1 \rceil$ errors, which is able to solve some received messages that would be uncorrectable for BD decoding. For GS algorithm, the encoder is different from the typical RS encoder which constructs the codeword C by the evaluation mapping method; that is $C = (C_0, C_1, \dots, C_{n-1}) = (M(1), M(\alpha), \dots, M(\alpha^{n-1}))$, where $M(x)$ is the message polynomial. In addition, an adjustable integer parameter, multiplicity m' , must be given before decoding to build the corresponding designed decoding radius $t_{m'}$. Thus the GS(m')

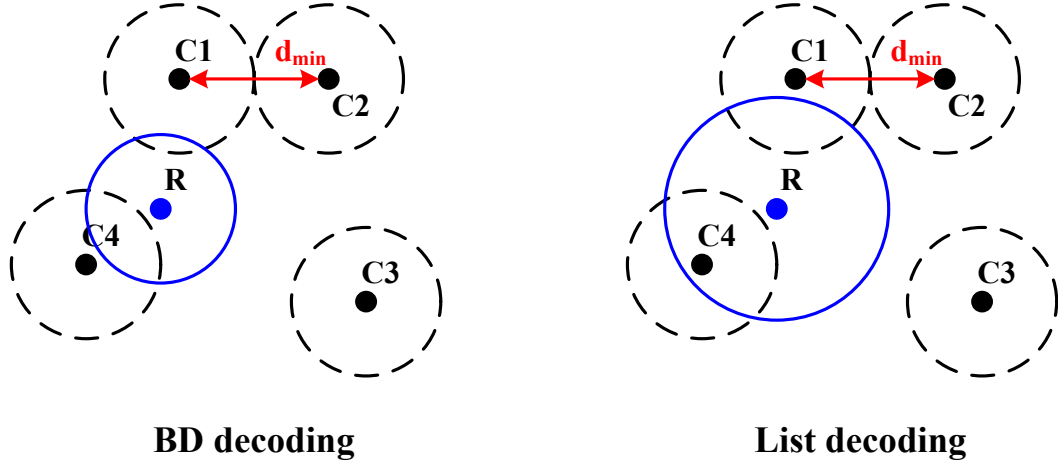


Figure 2.4: Comparison for BD and list decoding

decoder will return a list which includes all codewords within Hamming distance $t_{m'}$ from the received vector R . The GS decoding procedure can be mainly separated into two steps: interpolation and factorization. The following paragraphs will briefly introduce these two steps.

Interpolation

An *interpolation-based* decoding takes the point of view that the codeword polynomial is the interpolation result from each point of the set (x_i, C_i) , $i = 0, 1, \dots, n-1$, where $x_i = \alpha^i$. Because the received data may be interfered with noise, the decoder tries to use polynomial interpolation to reconstruct the transmitted codeword polynomial. Several non-zero bivariate candidate polynomials are constructed by interpolating each point (x_i, y_i) , $i = 0, 1, \dots, n-1$, and the minimum-degree polynomial is chosen as $Q(x, y)$ with the following form

$$Q(x, y) = \sum_{j=0}^C a_j \phi_j(x, y) \quad (2.13)$$

$$C = n \cdot \frac{(m' + 1)!}{2!(m' - 1)!} \quad (2.14)$$

where $\phi_j(x, y)$ is of the form $x^p y^q$. Larger multiplicity m' will increase the error correction ability. However, the computation complexity would be highly raised, which has a time

complexity of $O(n^2m^4)$. In 2002, Koetter's algorithm [20] provided an efficient interpolation method, which is described in Algorithm 2.

Algorithm 2. : Koetter's Interpolation Algorithm

Input :

Points : $(x_i, y_i), i = 0, 1 \dots, n - 1$;

Interpolation order m'_i ;

$L = \text{maximum list number.}$

Initialization :

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

for $i = 0$ **to** $n - 1$

for $(r, s) = (0, 0)$ **to** $(m'_{i-1}, 0)$ *by* $(m'_{i-1}, 1)$ *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 $(\mathbf{J} \neq \text{NULL})$

$j^* = \text{argmin}\{g_j : j \in \mathbf{J}\}$

$f = g_{j^*}$

$\Delta = \Delta_{j^*}$

for $(j \in \mathbf{J})$

if $(j \neq j^*)$

$g_j = \Delta g_j + \Delta_j f$

else if $(j = j^*)$

$g_j = (x - x_i)f$

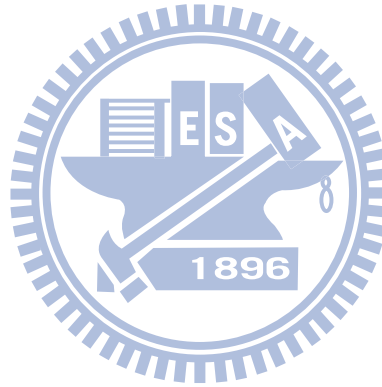
end (if)

end (for j)

end (for J)

end (for (r, s))

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



Factorization

After the interpolation procedure, $Q(x, y)$ will contain some factors polynomial of the form $y - p(x)$ with degree of $p(x)$, $\text{deg}(p(x))$, less than k , and $p(x)$ is one of the decoded result in the list. Thus factorization is exploited to identify all the factors of $p(x)$ and the output of

the algorithm is a list of the codewords that correspond to these factors. The factorization step based on Roth-Ruckenstein algorithm [21] is illustrated in Algorithm 3.

Algorithm 3. : Roth-Ruckenstein Algorithm

Initialization :

$p(x) = 0$, $u = \deg(p) = -1$, $D = \text{maximum degree of } p(x)$, $v = 0$.
 Call *Rothrucktree* ($Q(x, y), u, p$)

Input :

$Q(x, y)$.

Function *Rothrucktree* ($Q(x, y), u, p$) :

$v = v + 1$

if ($Q(x, 0) = 0$)

add $p(x)$ *into output list*

else if ($u < D$)

$R = \text{list of roots of } Q(0, y)$

for each $\alpha \in R$

begin

$Q_{\text{new}}(x, y) = Q(x, xy + \alpha)$

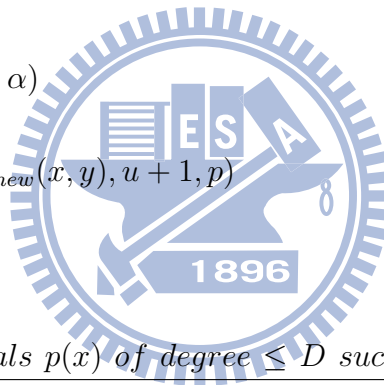
$p_{u+1} = \alpha$

 Call *Rothrucktree* ($Q_{\text{new}}(x, y), u + 1, p$)

end

else

output list = NULL



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

2.3.4 Koetter-Vardy Algorithm

Based on the GS algorithm, Kottter and Vardy presented an algebraic soft decoding algorithm, referred to Koetter-Vardy (KV) algorithm, by extending the GS algorithm to include a method for translating soft information into algebraic conditions [22].

By applying those soft information, KV algorithm provides a multiplicity assignment to offer every point its own multiplicity according to the relatively reliabilities of all possible transmitted/received symbol pairs and hence a variable number of constraints for each point. For an (n, k) RS code over $GF(2^m)$, a $(2^m - 1) \times n$ reliability matrix Π can be built as the a

posteriori probability (APB) with complexity constraints s while each entry $\pi_{i,j}$ represents the reliability of a point (x_j, y_i) . Therefore the multiplicity matrix M can be calculated from Π . Fig. 2.5 is the decoding process of the KV algorithm.

Algorithm 4. : Multiplicity Assignment Algorithm

Initialization :

Choose a desired value for $\sum_{i=0}^{2^m-1} \sum_{j=0}^{n-1} m'_{i,j}$

$\Pi^* = \Pi, M = 0$

While $s > 0$ **do**

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

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

$$m'_{i,j} = m'_{i,j} + 1$$

$$s = s - 1$$

Output :

Multiplicities M

In the KV algorithm, the most computationally demanding step is bivariate polynomial interpolation. The re-encoder technique [23], which is shown in Fig. 2.6, can be applied to reduce the number of interpolation points which can be viewed as an erasure only decoder. The received message is the summation of codeword and error polynomial after transmitting the codeword through a noisy channel, that is $R = C + E$. The received symbols in R can be separated into two sets: $n - k$ symbols in I_U (“unreliable”) and k symbols in I_R (“reliable”) based on the reliability matrix where I_R includes k most reliable symbols and the rest of them are in I_U . Thus, a *re-encoded codeword* ψ can be constructed by a systematic encoder

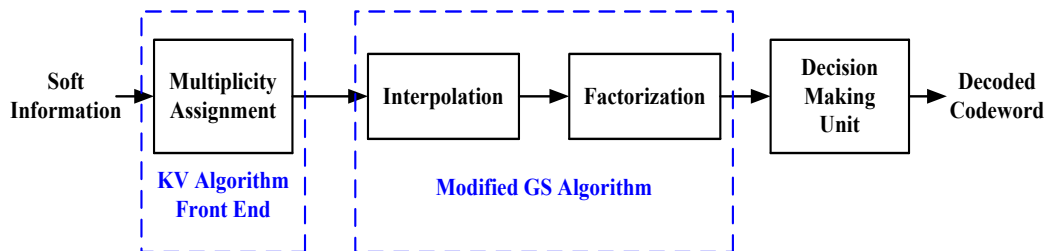


Figure 2.5: Kotter-Vardy decoding process

with k most reliability symbol in I_R . The difference between R and ψ is

$$\begin{aligned}
 R' &= R - \psi \\
 &= (C + E) - \psi \\
 &= (C - \psi) + E \\
 &= C' + E.
 \end{aligned} \tag{2.15}$$

Since both C and ψ are codewords, C' will also be a legal codeword and R' can be regarded as a codeword influenced by the same error vector as R . Due to the property of the systematic encoder, R' will have k zero symbols in the I_R locations, which means that there are k interpolation points with a zero y-component:

$$V = \{(\alpha^i, 0)\}, \quad i \in I_R. \tag{2.16}$$

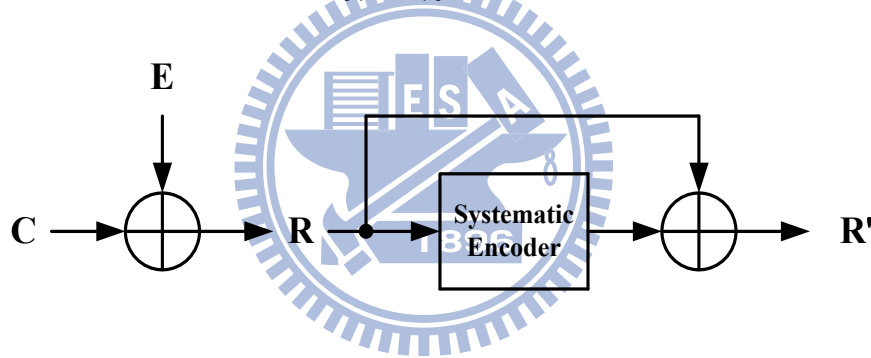


Figure 2.6: Re-encoding technique

As a result, an interpolation polynomial for these k points in V with the same multiplicity m is $v(x)^m$, where $v(x)$ can be calculated through a simple univariate interpolation:

$$v(x) = \prod_{i \in I_R} (x - \alpha^i). \tag{2.17}$$

Since most of the points are zero points after the re-encoding process, the computation of Koetter's interpolation can be considerably reduced due to only few number of the remainder points.

To further reduce the computation complexity, the coordinate transformation based technique was provided by Koetter and Vardy in [24]. After choosing the maximum possible

multiplicity $m = d_y$ and using the re-encoding method, the bivariate polynomial $Q(x, y)$ can be reduced to

$$\begin{aligned} Q(x, y) &= \sum_{j=0}^{d_y} \omega_j(x) v(x)^{m-j} y^j \\ &= v(x)^m \sum_{j=0}^{d_y} \omega_j(x) \left(\frac{y}{v(x)} \right)^j, \end{aligned} \quad (2.18)$$

which reveals that a large requirement of memories is applying to store the common term $v(x)$. Because the purpose of factorization is only to find the factor of the form $y - p(x)$ from $Q(x, y)$, the term $v(x)^m$ can be removed and a *reduced interpolation polynomial* can be defined as:

$$\tilde{Q}(x, \tilde{y}) = \sum_{j=0}^{d_y} \omega_j(x) \tilde{y}^j, \quad (2.19)$$

where $\tilde{y} = \frac{y}{v(x)}$. In addition, a message polynomial $u(x)$ corresponding to the estimated transformed codeword C' will be a linear y -root of $Q(x, y)$ for a successful decoding with re-encoding method. Therefore $\frac{u(x)}{v(x)}$ is linear \tilde{y} -root of $\tilde{Q}(x, \tilde{y})$:

$$\tilde{Q}(x, \tilde{y}) = \left(\tilde{y} - \frac{u(x)}{v(x)} \right) A(x, \tilde{y}). \quad (2.20)$$

It's obvious that the degree of $\frac{u(x)}{v(x)}$ is much smaller than k , which leads to significant simplification of calculation for Roth-Ruckenstein algorithm. Moreover, as mentioned before, R' has k zero symbols in the I_R locations, which means the error values e_i occurred in the k most reliable positions can be evaluated with the following method

$$u(\alpha^i) = e_i, \quad i \in I_R \quad (2.21)$$

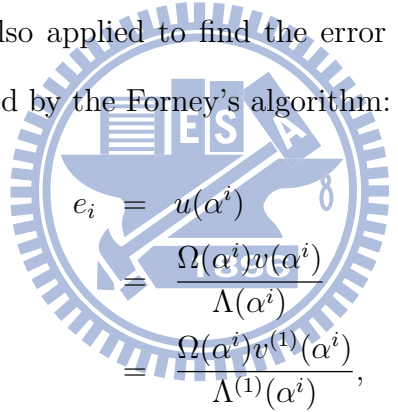
and $(x - \alpha^i)$ will be a root $u(x)$ in all the error-free locations in I_R . Therefore, a *reduced*

factorization polynomial $s(x)$ can be constructed with the following form

$$\begin{aligned}
 s(x) &= \frac{u(x)}{v(x)} \\
 &= \frac{\prod_{i \in I_R, e_i=0} (x - \alpha^i) \Omega(x)}{\prod_{i \in I_R} (x - \alpha^i)} \\
 &= \frac{\Omega(x)}{\prod_{i \in I_R, e_i \neq 0} (x - \alpha^i)} \tag{2.22}
 \end{aligned}$$

$$= \frac{\Omega(x)}{\Lambda(x)}. \tag{2.23}$$

The $\Lambda(x)$ and $\Omega(x)$ can be regarded as an error locator polynomial for the positions in I_R and the corresponding error evaluator polynomial respectively. Like traditional RS decoder, $\Lambda(x)$ and $\Omega(x)$ can be figured out with *syndrome polynomial* $s(x)$ based on the Berlekamp-Massey algorithm. Chien search is also applied to find the error locations and the corresponding error values will be determined by the Forney's algorithm:



$$\begin{aligned}
 e_i &= u(\alpha^i) \\
 &= \frac{\Omega(\alpha^i)v(\alpha^i)}{\Lambda(\alpha^i)} \\
 &= \frac{\Omega(\alpha^i)v^{(1)}(\alpha^i)}{\Lambda^{(1)}(\alpha^i)}, \tag{2.24}
 \end{aligned}$$

where $v^{(1)}(x)$ and $\Lambda^{(1)}(x)$ are the formal derivatives of $v(x)$ and $\Lambda(x)$ respectively. Accordingly, Fig 2.7 is the decoding process of the simplified KV algorithm. The latest information about KV algorithm can be found in [25].

2.3.5 Low Complexity Chase Algorithm

By translating soft information from channel, KV algorithm enables the back-end GS decoder to perform better correction ability with less complexity. However, the calculation complexity of interpolation with high multiplicity m' is still too high for practical implementation. In 2006, Low complexity Chase (LCC) algorithm [26] is presented, which is a

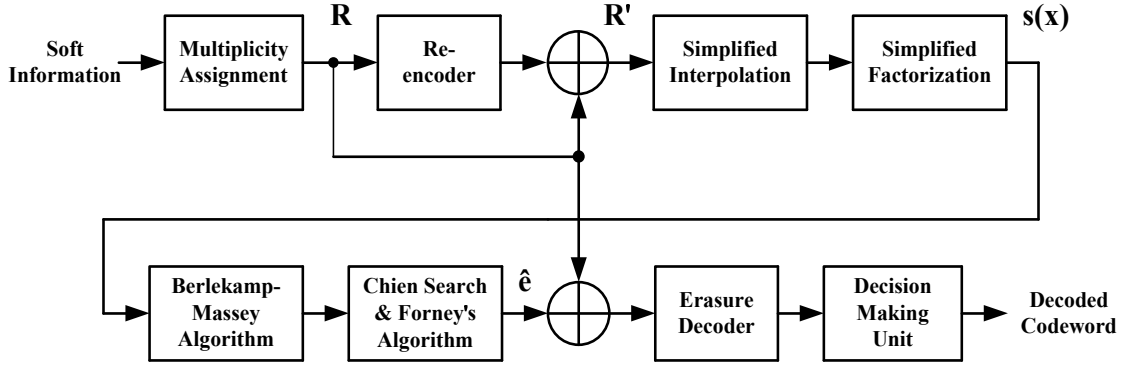


Figure 2.7: Simplified Kotter-Vardy decoding process

Chase-type decoding method cooperating with KV algorithm with multiplicity $m' = 1$. According to the received reliability matrix Π , the most and second probable symbol pairs of each received point will be defined as hard-decision (x_i, y_i^{HD}) and secondary hard-decision (x_i, y_i^{2HD}) respectively for $i = 0 \sim n - 1$. Then the probability γ_i which represents the reliability of each point is given as

$$\gamma_i = \frac{p(c_i = y_i^{2HD} | \gamma_i)}{p(c_i = y_i^{HD} | \gamma_i)} \leq 1. \quad (2.25)$$

If γ_i is close to 1, it's highly possible that the hard-decision is wrong and the secondary hard-decision is correct.

For LCC decoding, η least reliability symbols with maximum γ_i will be found out and form the unreliable position set $\mathcal{I} = \{i_1, i_2, \dots, i_\eta\}$. Then the n -point candidate sequence is defined as

$$(x_i, y_i) = \begin{cases} \{(x_i, y_i^{HD})\} & \text{if } i \notin \mathcal{I} \\ \{(x_i, y_i^{HD}, (x_i, y_i^{2HD}))\} & \text{if } i \in \mathcal{I} \end{cases} \quad (2.26)$$

Note that there are two possible symbols for each position in \mathcal{I} , the total number of candidate sequences is 2^η . Since there are $n - \eta$ common points in these candidate sequences, interpolation results of these points will be the same for 2^η candidate sequences by taking advantage of that each point is interpolated individually. Hence only η data has to be interpolated repeatedly. In 2009, further improvement about LCC for area efficiency is proposed in [27]. Since the multiplicity $m' = 1$, the maximum y -degree of the interpolation output is

one. Thus $Q(x, y)$ can be expressed as

$$Q(x, y) = q_0(x) + q_1(x)y. \quad (2.27)$$

Hence the factorization step can be removed. After using re-encoding and coordinate transformation methods, the reduced factorization polynomial $s(x)$ can be created as

$$s(x) = \frac{q_0(x)}{q_1(x)}. \quad (2.28)$$

As mentioned in the procedure of KV algorithm, the error values can be evaluated with the following process.

$$\begin{aligned} e_i &= u(x)|_{x=\alpha^i} \\ &= s(x)v(x)|_{x=\alpha^i} \\ &= \frac{q_0(\alpha^i)v(\alpha^i)}{q_1(\alpha^i)} \\ &= \frac{q_0(\alpha^i)v^{(1)}(\alpha^i)}{q_1^{(1)}(\alpha^i)}. \end{aligned} \quad (2.29)$$

Based on the above technique, the factorization-free LCC decoding process is shown in Fig 2.8.

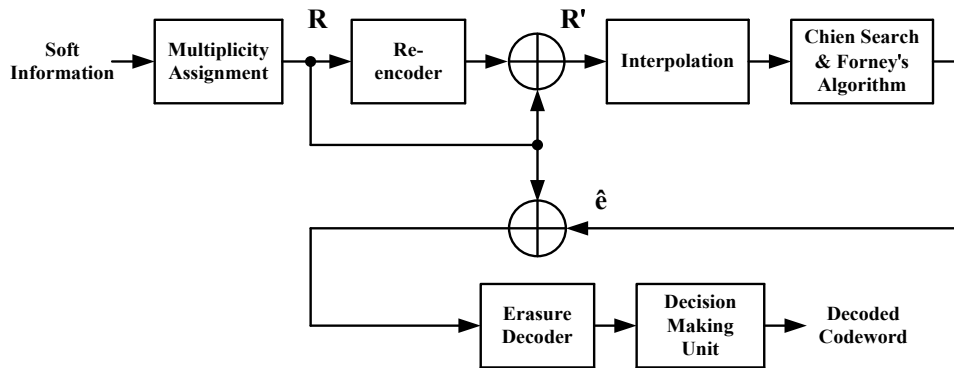


Figure 2.8: The factorization-free LCC decoder

Chapter 3

Decision-Confined Soft Decoding

Algorithm

In the previous section, we have introduced some well-developed soft decoding algorithms. However, because of high computational complexity, these methods are still unsuitable for practical implementation. In order to make use of soft information from channel to enhance the correction capability and maintain area efficiency, a decision-confined soft decoding algorithm is presented in this chapter.

In this chapter, we will present the detail description of our proposed method. Then some simulation results and the comparison between other different design will be shown.

3.1 Decision-Confined Soft Decoding Algorithm

In the Chase-2 (Chase) algorithm, the DMU consumes high hardware cost due to the complex calculation of the Euclidean distance. However, the effect of the DMU on the error performance is not clear. Fig. 3.1 shows the simulation results of the conventional Chase and the simplified Chase algorithm. From Fig. 3.1, the performance loss without the DMU is very slight. As a result, the hardware complexity can be significantly reduced by removing

the DMU without leading to obvious loss of error correction capability. The decoding flow of simplified Chase algorithm is illustrated as the following description.

- LRP's determination:

Determine η least reliable positions.

- Candidate codewords generation:

for $j = 1$ to 2^η by 1 : Modify the LRPs by complementing one combination of LRPs and send it to hard RS decoder.

- Output codeword:

If the number of error of decoded sequence is no more than t , output the decoded codeword and terminate the decoding procedure.

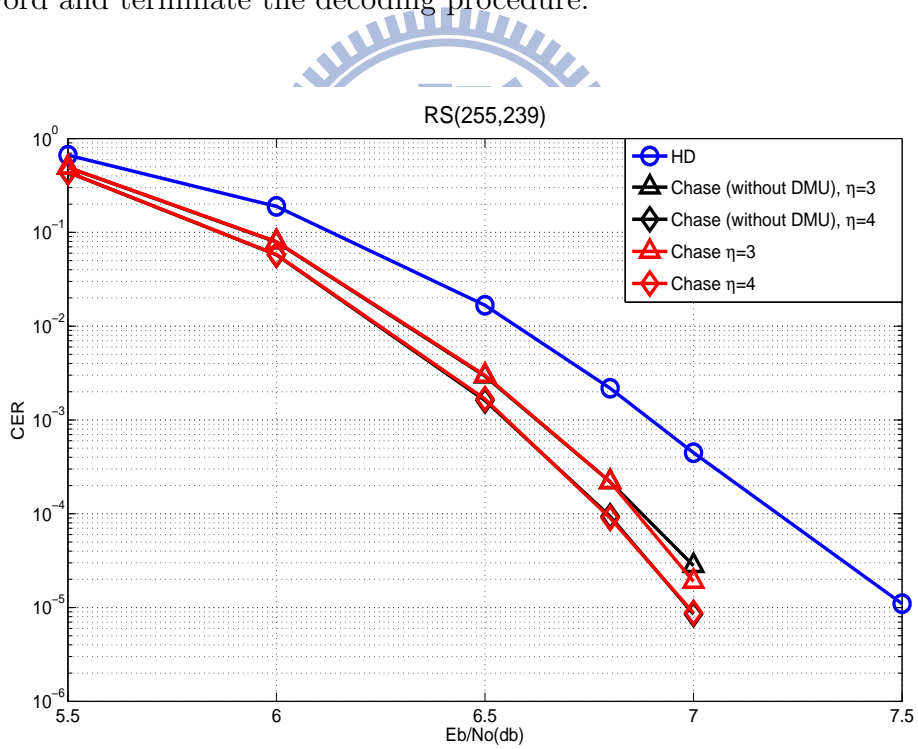


Figure 3.1: Simulation results of Chase and simplified Chase algorithm

As mentioned in Section 2.2.2, a successful RS decoding is defined that the number of roots found by Chien search is equal to the degree of error-locator polynomial $\Lambda(x)$. However,

from our simulation results, when a received data has more than t errors, it's highly possible for Chien search to find less than t roots for a degree- t $\Lambda(x)$, which leads to an unsuccessful decoding and might introduce some extra errors.

Table 3.1: Percentage of degree of $\Lambda(x)$ equals to 8 (10^5 test patterns)

Error number	9	10	11	12	13	14
Percentage %	99.5	99.5	99.6	99.6	99.5	99.5

Table 3.1 is the simulation result of RS (255,239), $t = 8$, and there is over 99.5% probability for Chien search to find a degree-8 $\Lambda(x)$ when the number of errors exceeds 8. According to this characteristic, we propose a decision-confined algorithm to enhance the decoding efficiency by confining the degree of $\Lambda(x)$, leading to only one candidate will be sent to Chien search.

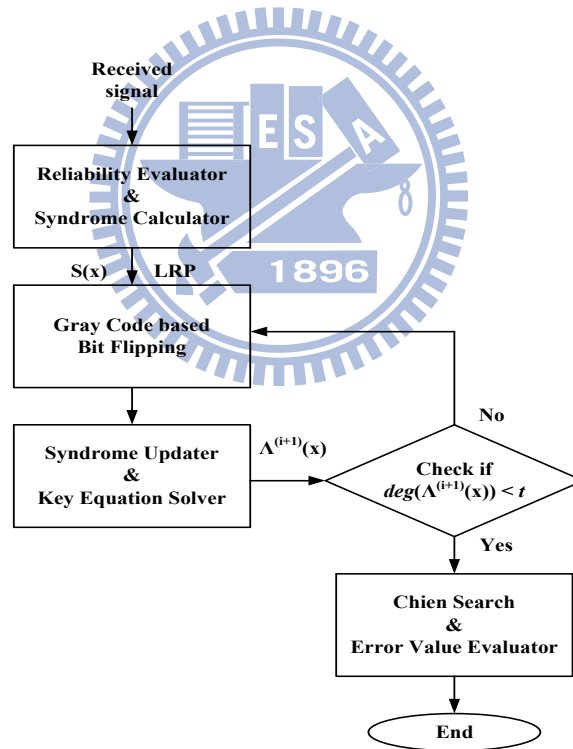


Figure 3.2: Proposed soft decoding flow

Fig. 3.2 shows our proposed soft decoding flow. First of all, based on the received soft information, η least reliable positions (LRPs), $[l_0, l_1, \dots, l_{\eta-1}]$, are defined and $S(x)$ is calcu-

lated simultaneously. The candidate sequences are generated according to Gray code based bit-flipping method, leading to only one bit of these LRPs flipped between each successive candidate. As a result, $S^{(i+1)}(x)$ for the $(i + 1)$ -th candidate can be updated with the following method.

$$S_j^{(i+1)} = S_j^{(i)} + e'_k \times \alpha^{l_k \times j}, \quad 1 \leq j \leq 2t. \quad (3.1)$$

$S_j^{(i+1)}$ is the j -th coefficient of $S^{(i+1)}(x)$ and $e'_k \times \alpha^{l_k \times j}$ is the *compensation value*, which can be viewed as the error pattern induced by the bit-flipping operation of k -th LRP. For example, if the flipped LRP is the 4th bit in 25th symbol of the received data, the values

Algorithm 5. : Decision-Confined Algorithm

Input : $R(x)$ and η -bit integers $\gamma = \gamma' = 0$.

step 1.

Calculate syndrome $S(x)$.

Evaluate η LRPs, $L = [l_0, l_1, \dots, l_{\eta-1}]$, and

corresponding error values, $E' = [e'_1, e'_2, \dots, e'_\eta]$.

step 2.

for $i = 0, i < 2^n - 1, i = i + 1$:

$\gamma' = \gamma, \gamma = i \oplus (i \gg 1)$ (Gray code)

Find the bit different between γ and γ' , k -th bit

Update syndrome $S^{(i+1)}(x)$:

$$S_j^{(i+1)} = S_j^{(i)} + e'_k \times \alpha^{l_k \times j}, \quad 1 \leq j \leq 2t.$$

Calculate $\Lambda^{(i+1)}(x)$ from KES with $S^{(i+1)}(x)$.

if $(\deg(\Lambda^{(i+1)}(x)) < t)$

go to **step 4.**

else if $(i = 2^n - 2 \text{ and } \deg(\Lambda^{(i+1)}(x)) = t)$

go to **step 3.**

end for

step 3.

Calculate $\Lambda(x)$ with $S(x)$.

step 4.

Find error locations and

evaluate error values to obtain $e(x)$.

Output : $\hat{C}(x) = R(x) \oplus e(x)$.

End

of e'_k and α^{l_k} will be α^4 and α^{25} respectively. After updating the syndrome $S^{(i+1)}(x)$ and calculating the corresponding $\Lambda^{(i+1)}(x)$, we set a condition that only the $\Lambda^{(i+1)}(x)$ with degree less than t will be sent to Chien search to find the error locations because it's highly possible for the $\Lambda^{(i+1)}(x)$ to be in the limit of correction capability. If the condition is met, the candidate sequence will be decoded as the output message and the decoding procedure will be terminated. Otherwise, next candidate will be generated to repeat above-mentioned steps. If no one meets the condition among all $2^\eta - 1$ candidates, the received signal will be decoded without the condition, and the error correction capability as hard RS decoders is guaranteed. Algorithm 5. is the detail illustration of our proposed method.

3.2 Performance Analysis

Fig. 3.3 shows the RS (255,239) simulation results for our proposed algorithm with different η under BPSK modulation and AWGN channel. The performance gain at 10^{-4} CER is 0.4 dB with $\eta = 5$ over the hard decoding. From Fig. 3.3, our proposed method with $\eta = 5$ can

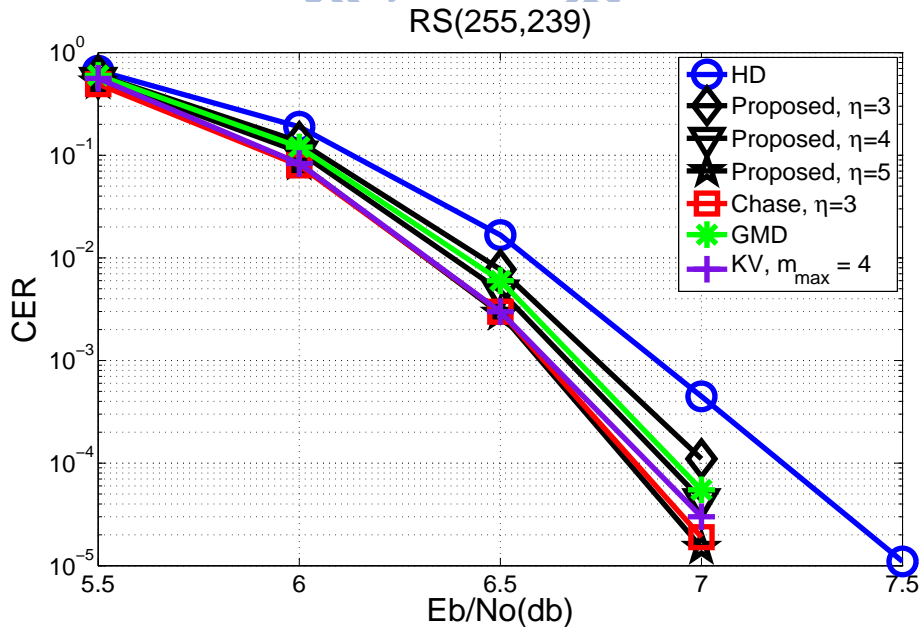


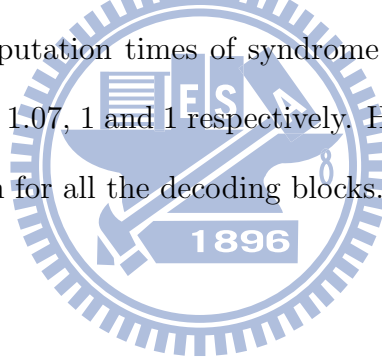
Figure 3.3: Performance of the proposed soft decoding algorithm

provide better error performance than GMD and KV algorithm with $m_{max} = 4$, and achieve competitive coding gain as Chase algorithm with $\eta = 3$.

Table 3.2: Comparison of Computational Complexity with Soft RS Decoder

	Proposed Algorithm			Chase Algorithm		
	$\eta = 5$			$\eta = 3$		
E_b/N_0	6.0	6.5	7.0	6.0	6.5	7.0
Syndrome Calculator	5.22	1.30	1.07	8	8	8
KES	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

Although it needs to flip more LRPs, the average computational complexity of our proposal is much less than Chase algorithm. Table 3.2 is the comparison of computational complexity with Chase algorithm. For instance, at $E_b/N_0 = 7$, according to our approach with $\eta = 5$, the average computation times of syndrome updater, KES, Chien search and error value evaluator are 1.07, 1.07, 1 and 1 respectively. However, the Chase algorithm with $\eta = 3$ consumes 2^3 calculation for all the decoding blocks.



Chapter 4

Decision-Confined Soft Reed-Solomon (255,239) Decoder

For the optical fiber systems and GPON applications [3] [4], RS (255,239) is standardized with demand for 2.5 Gb/s throughput to achieve 2.5, 10 and 40 Gb/s (STM-16, STM-64 and STM-256) with 16 RS decoders or satisfy the maximum up and down link requirement. Fig 4.1 illustrates the block diagram of a submarine system which use a FEC function to benefit the overall system. Fig 4.2 shows the frame structure of the FEC decoder for optical communication systems.

Due to the parallel decoder architecture in optical fiber systems and large amount of users for GPON, the hardware cost is also an important issue. According to our proposed algorithm, an area-efficient decision-confined soft Reed-Solomon (255,239) decoder is presented to support longer transmission distance for high data rate optical communication systems. In the beginning of this chapter, we will introduce the overall decoding scheme. Then the detail VLSI architecture of each component will be shown. In the end of this chapter, we will reveal the implementation result of our soft decoder chip.

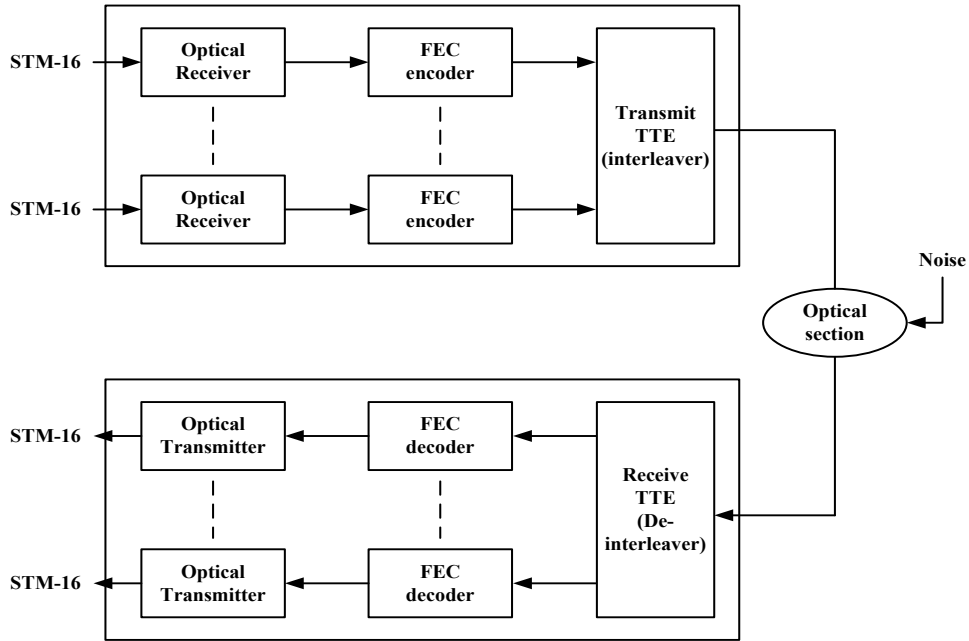


Figure 4.1: Block diagram of a submarine system which use a FEC function

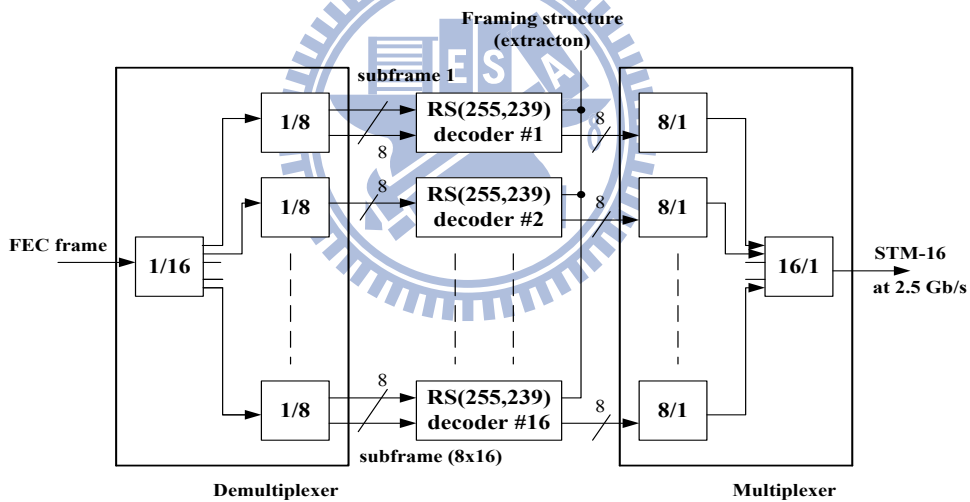


Figure 4.2: FEC decoder architecture of optical communication systems

4.1 Decoding Scheme

For the 2.5 Gb/s requirement of the optical communication systems, a soft RS (255,239) decoder with three pipelined stages based on our decision-confined decoding algorithm is presented and the decoding scheme is shown in Fig. 4.3. At the first stage, the reliabil-

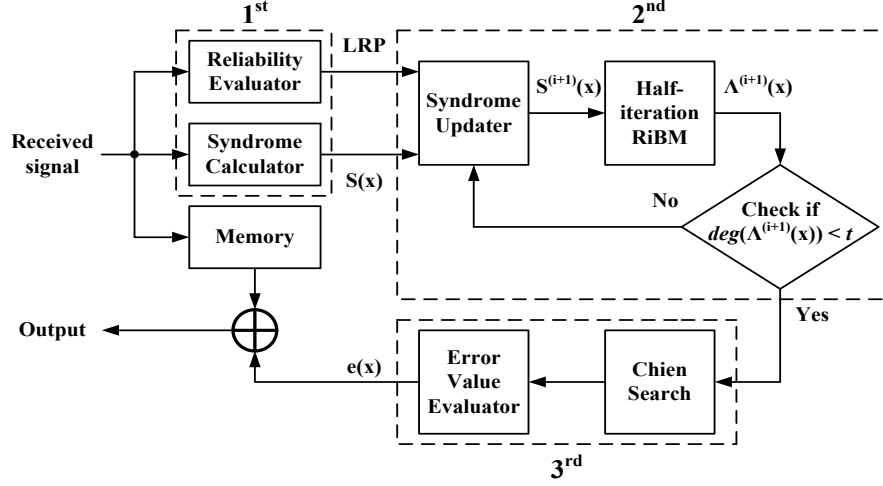


Figure 4.3: Decoding scheme of the proposed soft RS decoder

ity evaluator computes 5 LRPs and the syndrome calculator counts the syndrome $S(x)$ in the meanwhile. The syndrome updater then iteratively modifies the syndromes $S^{(i+1)}(x)$ according to the LRPs with Gray code based bit-flipping method and the KES solves the corresponding $\Lambda^{(i+1)}(x)$ at the second stage as illustrated in step 2. of Algorithm 4. The candidate sequence with the degree of $\Lambda^{(i+1)}(x)$ less than t will be sent to a parallel-2 Chien search to find the error locations and calculate the error values with Björck-Pereyra based method [28] at the last stage. Since the received message for the decoders is serial input and there are totally 255 symbols for a RS (255,239) codeword, the computation cycle for each pipelined stage is usually 255. However, according to our timing schedule, each pipelined stage has been slightly enlarged from 255 to 259 clock cycles due to the 3-pipeline-staged reliability evaluator, which will be described more specifically in the following paragraph. The rest of the sections will show the VLSI architecture of each component of our proposal in detail.

4.2 Proposed VLSI Architecture

4.2.1 Syndrome Calculator and Reliability Evaluator

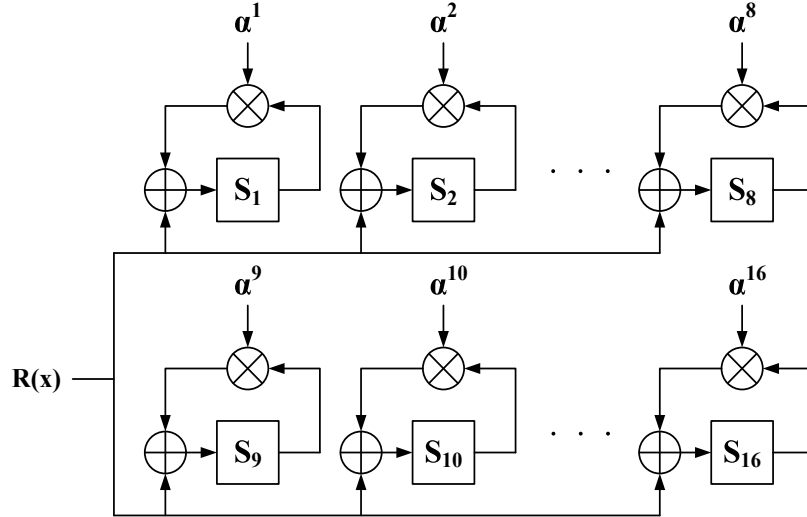


Figure 4.4: Syndrome calculator

At the first stage, the syndrome polynomial is calculated based on (2.7) with the architecture shown in Fig. 4.4. In addition, with the advantage of *divide-and-conquer* leading to fewer comparing times, we choose the *merge sort* concept [29] as the reliability evaluator to find 5 LRPs to reduce critical path. The reliability evaluator is formed with 3-stage pipeline design which is shown in Fig. 4.5. The first and second stages calculate the 5 LRPs of the serial input while the third stage decides the new temporary candidate LRPs among the output of last stage and the last temporary candidate. Fig. 4.6 shows the small components of the reliability evaluator. Due to the delay of pipelined design of the reliability evaluator, the latency of each pipelined stage has been enlarged from 255 to 259 clock cycles.

4.2.2 Syndrome Updater

Since the Gray code based bit-flipping method is applied, leading to only one bit of these LRPs flipped between each successive candidate. Fig. 4.7 is an example of Gray code based

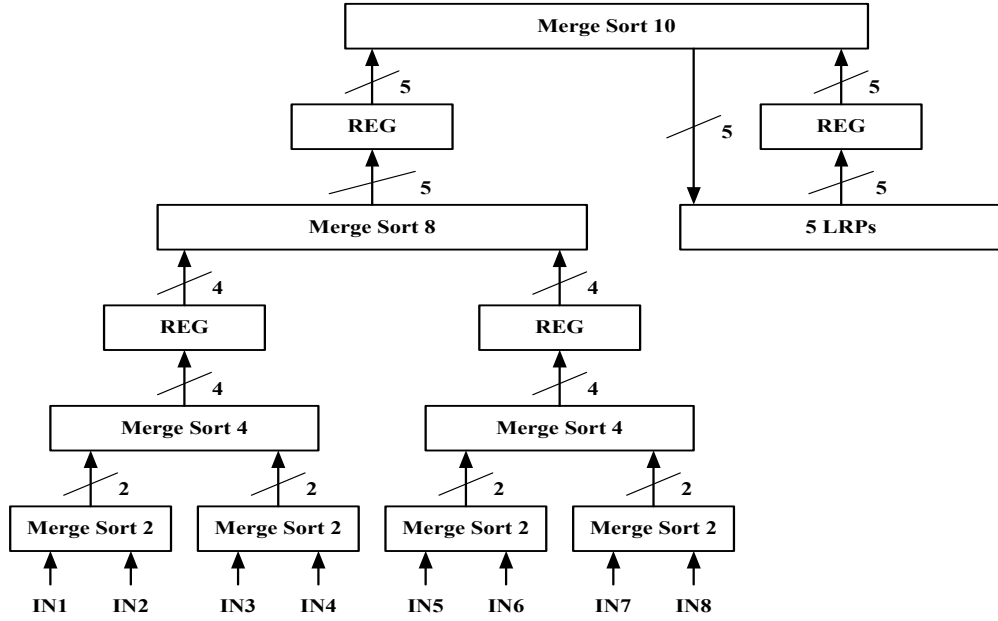


Figure 4.5: Reliability evaluator

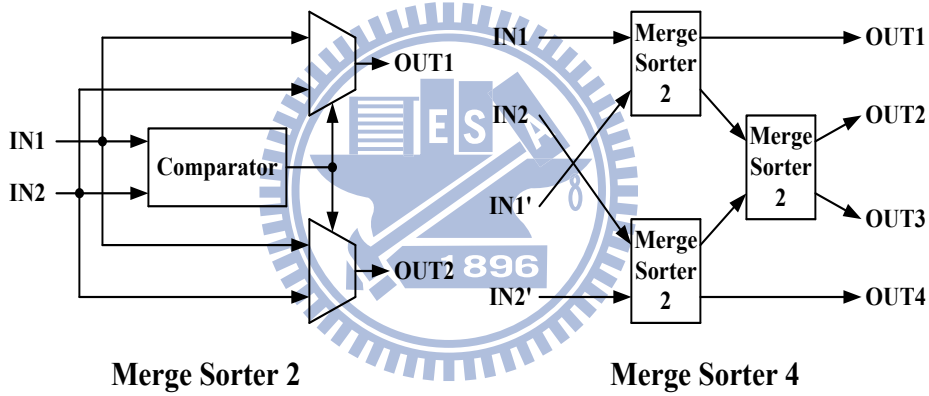


Figure 4.6: Merge sorter

bit-flipping method. Thus the candidate syndrome $S^{(i+1)}(x)$ can be updated from $S^{(i)}(x)$ according to (3.1) by utilizing a look-up table (LUT) instead of recalculating it with syndrome calculator as (2.7) for further cost efficiency.

Note that there are at most 2^5 candidates for each received message and 259 computational cycles for each pipelined stage. Thus it has 8 computational cycles for every $S^{(i+1)}(x)$ and $\Lambda^{(i+1)}(x)$. As a result, the finite field multipliers (FFMs) and the squares can be shared to compute 16 compensation values for further hardware reduction. In our design, it only

costs 4 FFMs and 2 squares for the calculation of all compensation values as shown in Fig. 4.8.

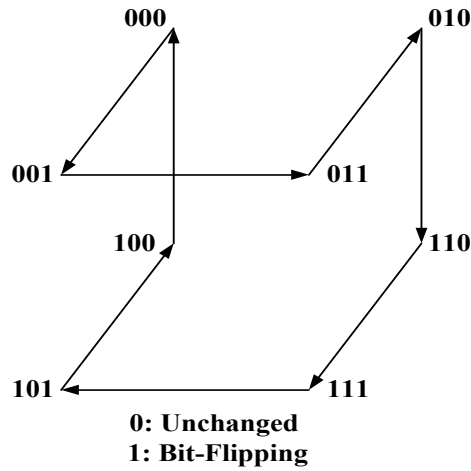


Figure 4.7: Gray code example

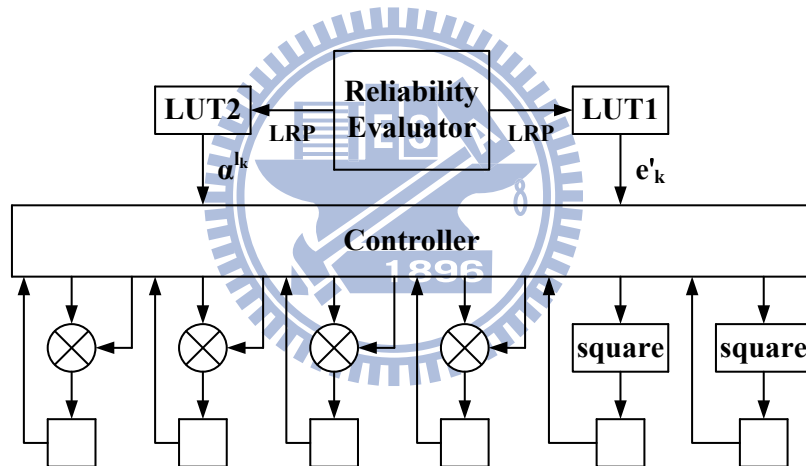


Figure 4.8: Syndrome updater

4.2.3 Half-iteration Key Equation Solver

The conventional key equation solver (KES) needs $2t$ iterations to solve the *key equation* : $\Omega(x) = S(x) \times \Lambda(x) \bmod x^{2t}$. For RS (255,239), it will cost 16 cycles to calculate $\Lambda(x)$. Instead of using two KES to meet 8 cycles timing constraint, which results in high complexity and difficult signal controlling, we propose a half-iteration RiBM algorithm on the basis

of [9] and [30] to shorten the latency of KES. Combining the advantages of homogeneous architecture and half computation latency, half-iteration RiBM can fully match our desire for KES.

Based on the method in step 1. of Algorithm 1, the discrepancy $\Delta(k, x)$ is exactly the coefficients of x^k in the polynomial product of syndrome and error-locator polynomial

$$\begin{aligned}\Delta(k, x) &= \Lambda(k, x) \cdot S(x) \\ &= \delta_0(k) + \delta_1(k) \cdot x + \cdots + \delta_k(k) \cdot x^k + \cdots\end{aligned}\quad (4.1)$$

Notice that the discrepancy $\delta_{i+k}(i+k)$ can not be changed by $\delta_i(k)$ and $\theta_i(k)$ for any $i < k$, so the RiBM algorithm utilizes the following updating equation to calculate $\Lambda(x)$ and $\Omega(x)$ in $2t$ iterations [9].

$$\hat{\delta}_i(k+1) = \gamma(k) \cdot \hat{\delta}_{i+1}(k) - \hat{\delta}_0(k) \cdot \hat{\theta}_i(k)\quad (4.2)$$

To halve the computation iteration of KES, Raghupathy and Liu found that the control signal L , which is normally considered to be the degree of $\Lambda(x)$ in the BM algorithm, can be increased only once in any two successive iterations, the updating iterations of key equation can be separated into odd and even iterations [30]. Based on the discrepancy of the odd iteration Δ_{2k-1} and the discrepancy predicted for the even iteration ε_{2k} , where

$$\Delta_{2k-1} = \sum_{j=0}^{L_{k-1}} \Lambda_j^{(2k-2)} S_{2k-1-j}\quad (4.3)$$

$$\varepsilon_{2k} = \sum_{j=0}^{L_{k-1}} \Lambda_j^{(2k-2)} S_{2k-j}\quad (4.4)$$

the updating equation can be modified into five cases as shown in Fig. 4.9. $\Lambda(x)$ then is calculated as

$$\Lambda_i^{(2k)} = \Lambda_i^{(2k-2)} + g_1 \Lambda_{i-1}^{(2k-2)} + g_2 B_{i-1}^{(2k-2)} + g_3 B_{i-2}^{(2k-2)}\quad (4.5)$$

where g_j ($j = 1 \sim 3$) are the updating factors based on the different cases. $\Lambda(x)$ will be then completed in only t iterations.

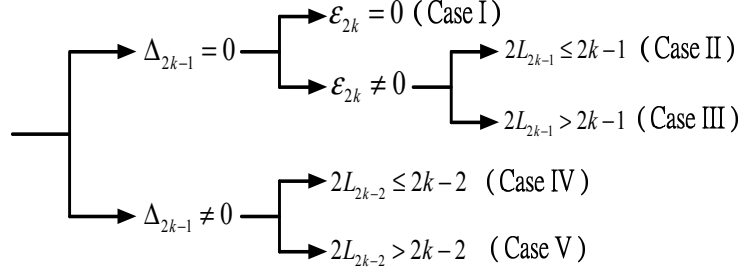


Figure 4.9: The updating criterion for half iteration BM algorithm

In comparison with the RiBM algorithm, the method of [30] can halve the computation iteration. However, due to the adder tree discrepancy calculation, it requires a critical path delay longer than $3 \cdot (T_{mult} + T_{add})$, where T_{mult} and T_{add} are the delay time of finite field multiplier (FFM) and finite field adder (FFA) respectively. Thus, we try to improve the critical path and provide the advantage of regularity by making use of the property of RiBM algorithm. Notice that Δ_{2k-1} and ε_{2k} are exactly the coefficients of x^{2k-2} and x^{2k-1} in (4.1). 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 (4.6).

$$\begin{aligned}
\hat{\delta}_i(k+1) &= \delta_{i+2k}(k+1) \\
&= g_0 \cdot \delta_{i+2k}(k) + g_1 \cdot \delta_{i+2k-1}(k) \\
&\quad + g_2 \cdot \theta_{i+2k-1}(k) + g_3 \cdot \theta_{i+2k-2}(k) \\
&= g_0 \cdot \hat{\delta}_{i+2}(k) + g_1 \cdot \hat{\delta}_{i+1}(k) \\
&\quad + g_2 \cdot \hat{\theta}_{i+1}(k) + g_3 \cdot \hat{\theta}_i(k). \tag{4.6}
\end{aligned}$$

To reduce the critical path, β_{k+1} can be updated with the similar manner of $\hat{\delta}_i(k+1)$

Algorithm 6. : Half-iteration RiBM Algorithm**Initialization :** $L_0 = 0, \alpha_0 = S_1, c_0 = 1$

$$\hat{\delta}_i(0) = \hat{\theta}_i(0) = 0, (i = 0, \dots, 3t - 1)$$

$$\hat{\delta}_{3t}(0) = \hat{\theta}_{3t}(0) = 1$$

Input : $\hat{\delta}_i(0) = \hat{\theta}_i(0) = S_{i+1}, (i = 0, \dots, 2t - 1)$

$$\beta_0 = S_2 - S_1^2$$

for $k = 1$ **to** t **do****begin****Step 1.****Case 1:** $\hat{\delta}_0(k - 1) = 0$ **and** $\hat{\delta}_1(k - 1) = 0$

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

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

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

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

$$g_3 = \hat{\delta}_1(k - 1), \hat{\theta}_i(k) = \hat{\theta}_i(k - 1)$$

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

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** $L_{k-1} \leq k - 1$

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

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

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

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

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

$$g_3 = \beta_{k-1}, \hat{\theta}_i(k) = \hat{\theta}_i(k - 1)$$

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

Step 2.

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

$$+ g_2 \cdot \hat{\theta}_{i+1}(k - 1) + g_3 \cdot \hat{\theta}_i(k - 1)$$

$$\beta_k = g_0(\alpha_k \cdot \hat{\delta}_2(k - 1) + c_k \cdot \hat{\delta}_3(k - 1))$$

$$+ g_1(\alpha_k \cdot \hat{\delta}_1(k - 1) + c_k \cdot \hat{\delta}_2(k - 1))$$

$$+ g_2(\alpha_k \cdot \hat{\theta}_1(k - 1) + c_k \cdot \hat{\theta}_2(k - 1))$$

$$+ g_3(\alpha_k \cdot \hat{\theta}_0(k - 1) + c_k \cdot \hat{\theta}_1(k - 1))$$

end**Output:** $\Omega_i(t + 1) = \hat{\delta}_i(t + 1), (i = 0, 1, \dots, t - 1).$

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

with $\hat{\delta}_0(k)$ and $\hat{\delta}_1(k)$

$$\begin{aligned}
\beta_k &= \hat{\delta}_1(k) \cdot c_k - \hat{\delta}_0(k) \cdot \alpha_k \\
&= g_0(\alpha_k \cdot \hat{\delta}_2(k-1) + c_k \cdot \hat{\delta}_3(k-1)) \\
&+ g_1(\alpha_k \cdot \hat{\delta}_1(k-1) + c_k \cdot \hat{\delta}_2(k-1)) \\
&+ g_2(\alpha_k \cdot \hat{\theta}_1(k-1) + c_k \cdot \hat{\theta}_2(k-1)) \\
&+ g_3(\alpha_k \cdot \hat{\theta}_0(k-1) + c_k \cdot \hat{\theta}_1(k-1)).
\end{aligned} \tag{4.7}$$

The structure of H-PE is depicted in Fig. 4.10 and the half-iteration RiBM algorithm can be implemented with $3t + 1$ H-PEs as illustrated in Fig. 4.11. At the beginning of the process, Λ_0 is initialized to 1 and stored into H-PE $_{3t}$. As the computation cycle k grows, Λ_0 is calculated and stored into H-PE $_{3t-2k}$. After t cycles of computation, the iterative algorithm will be completed. The coefficients of $\Omega(x)$ and $\Lambda(x)$ are stored in the H-PE $_i$ ($i = 0 \sim t - 1$) and H-PE $_{t+i}$ ($i = 0 \sim t$) respectively. Notice that the discrepancies Δ_{2k-1} and ε_{2k} will be always in the first and second H-PE ($\hat{\delta}_0(k)$ and $\hat{\delta}_1(k)$). The critical path passes through only two FFMs, two FFAs and one mux because the updating factor β_k for the next iteration is already available at the beginning of the next clock cycle. Algorithm 6. is the clear description of the half-iteration RiBM algorithm. From Algorithm 6, there is only one control signal among g_1 and g_3 not zero at each updating iteration, we can share

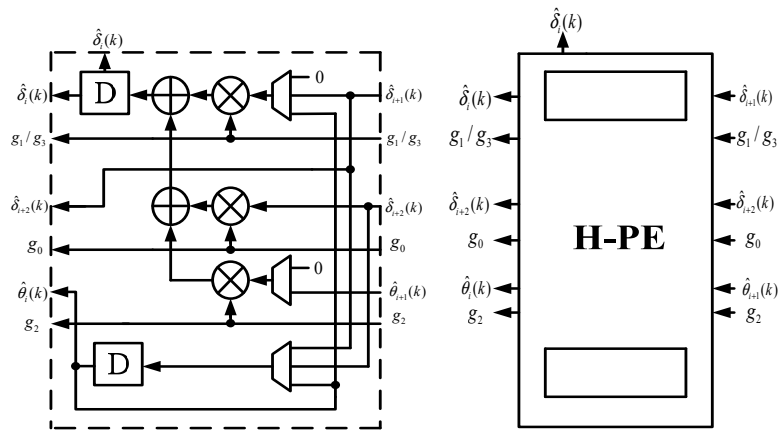


Figure 4.10: The processing element (H-PE) of half-iteration RiBM

Algorithm 7. : Half iteration RiBM Algorithm (without $\Omega(x)$)**Initialization :** $L_0 = 0, \alpha_0 = S_1, c_0 = 1, \beta_0 = S_2 - S_1^2$

$$\hat{\delta}_i(0) = \hat{\theta}_i(0) = S_{i+1}, (i = 0, \dots, 2t - 1)$$

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

for $k = 1$ **to** t **do****begin****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.

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

$$+ g_2 \cdot \hat{\theta}_{i+1}(k-1) + g_3 \cdot \hat{\theta}_i(k-1)$$

$$\beta_k = g_0(\alpha_k \cdot \hat{\delta}_2(k-1) + c_k \cdot \hat{\delta}_3(k-1))$$

$$+ g_1(\alpha_k \cdot \hat{\delta}_1(k-1) + c_k \cdot \hat{\delta}_2(k-1))$$

$$+ g_2(\alpha_k \cdot \hat{\theta}_1(k-1) + c_k \cdot \hat{\theta}_2(k-1))$$

$$+ g_3(\alpha_k \cdot \hat{\theta}_0(k-1) + c_k \cdot \hat{\theta}_1(k-1))$$

end**Output :** $\Lambda_i(t+1) = \hat{\delta}_i(t+1), (i = 0, 1, \dots, t)$.

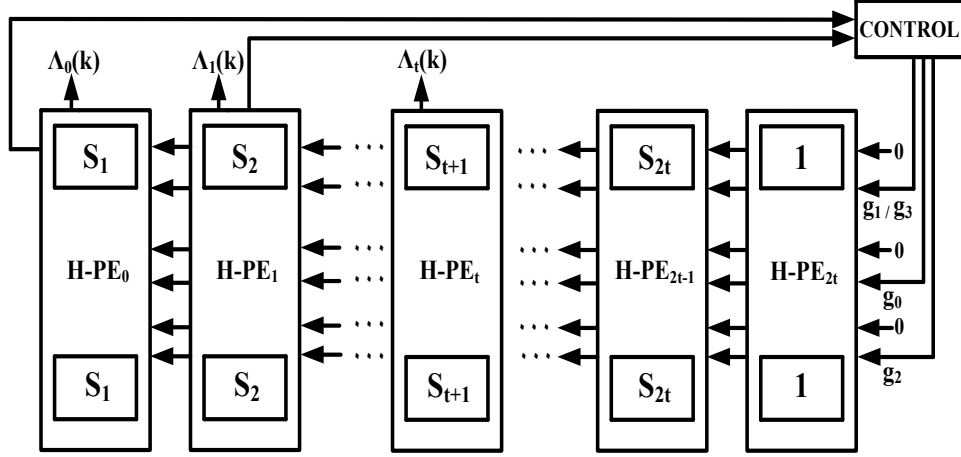


Figure 4.12: The regular architecture of half-iteration RiBM without the calculation of $\Omega(x)$

4.2.4 Chien Search and Error Value Evaluator

Once KES calculates $\Lambda(x)$, Chien search will be applied to find the error locators X_i . Conventionally, after Chien search evaluates the error locators, the corresponding error values e_i 's can be calculated with $\Lambda(x)$ and $\Omega(x)$ based on the Forney's algorithm (2.11).

From another approach, the Björck-Pereyra (BP) based method [28] can compute the error values by solving the Vandermonde relation between the syndrome S_i 's and error locators X_i 's as (4.8).

$$\begin{bmatrix} X_1 & X_2 & \cdots & X_8 \\ X_1^2 & X_2^2 & \cdots & X_8^2 \\ \vdots & \vdots & \ddots & \vdots \\ X_1^8 & X_2^8 & \cdots & X_8^8 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_8 \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_8 \end{bmatrix} \quad (4.8)$$

Algorithm 8. is the illustration of BP-based method. Each calculation of the syndrome represents a row operation in (4.8). To update each S_i value, the control logic settles the calculation order of the S_i and X_i . After computation, S_i will be the i -th error value.

Table (4.1) is the comparison of time and hardware complexity between different methods including KES, Chien search and error value evaluator. In the lower rows of the table for total hardware requirement, the estimate is given for RS (255, 239) over $GF(2^8)$. The

Algorithm 8. : Björck-Pereyra Algorithm
Input : S_i and $X_i, i = 1 \sim t$
step 1.
for $k = 1$ **to** $t - 1, k = k + 1 :$
for $i = t$ **to** $k + 1, i = i - 1 :$

$$S_i = S_i - X_k S_{i-1}$$

step 2.
for $k = t - 1$ **to** $1, k = k - 1 :$
for $i = k + 1$ **to** $t, i = i + 1 :$

$$S_i = \frac{S_i}{X_i - X_{i-k}}$$

$$S_{i-1} = S_{i-1} - S_i$$

step 3.
for $k = 1$ **to** $t, k = k + 1 :$

$$S_k = \frac{S_k}{X_k}$$

Output : $Y_i = S_i, i = 1 \sim t$

estimate for the latency is for each pipelined stage, where Chien search and error value evaluator are in the same stage. The difference between Forney's algorithm (1) and (2) is that method (1) calculates error values after the operation of Chien search; however, method (2) applies Chien search and evaluates error values in the meantime. From Table (4.1), since the Forney's algorithm and BP-based method consume nearly the same hardware costs, the hardware cost can be reduced distinctly by removing the calculation of $\Omega(x)$.

As a result, parallel-2 Chien search architecture, which is shown in Fig. 4.13, is employed for the timing constraint of our three pipeline design. After determining the error locators,

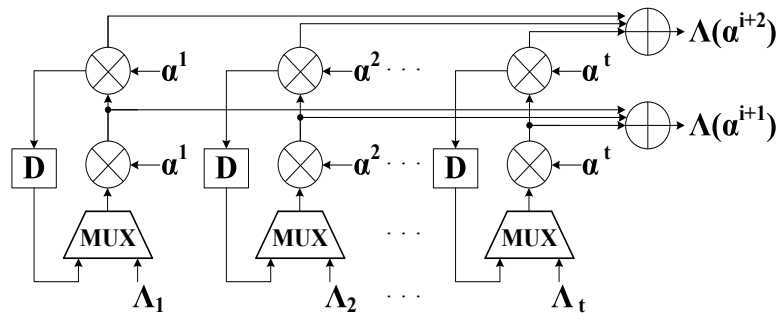
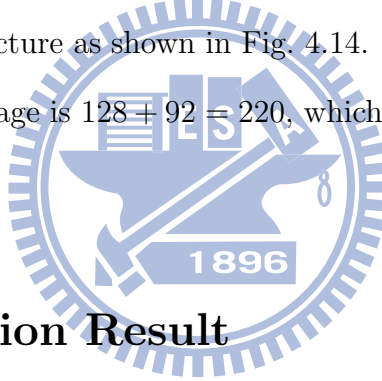


Figure 4.13: Parallel-2 Chien search architecture

Table 4.1: Comparison of Time and Hardware Complexity

	Constant FFM	Variable FFM	FFA	Reg. (bytes)	Mux (bytes)	ROM (bytes)	Latency
Method 1.							
Half-iteration RiBM (with $\Omega(x)$)	0	$9t + 11$	$6t + 5$	$6t + 5$	$9t + 10$	0	t
Forney's Algorithm (1)	0	2	2	t	2	n	$(t + 1)^2$
Chien Search parallel-2	$2t$	0	$2t$	t	t	0	$n/2$
Total	$2t$	$9t + 13$	$8t + 7$	$8t + 5$	$10t + 12$	n	$(t + 1)^2 + n/2$
	16	85	71	69	92	255	209
Method 2.							
Half-iteration RiBM (with $\Omega(x)$)	0	$9t + 11$	$6t + 5$	$6t + 5$	$9t + 10$	0	t
Forney's Algorithm (2)	t	0	t	t	t	n	n
Chien Search	t	0	t	t	t	0	n
Total	$2t$	$9t + 11$	$8t + 5$	$8t + 5$	$11t + 10$	n	n
	16	83	69	69	98	255	255
Method 3.							
Half-iteration RiBM (without $\Omega(x)$)	0	$6t + 11$	$4t + 5$	$4t + 5$	$6t + 10$	0	t
BP Algorithm	0	2	2	t	$8t$	n	$\frac{3}{2}t^2 - \frac{t}{2}$
Parallel-2 Chien Search	$2t$	0	$2t$	t	t	0	$n/2$
Total	$2t$	$6t + 13$	$6t + 7$	$6t + 5$	$15t + 10$	n	$\frac{3}{2}t^2 - \frac{t}{2} + n/2$
	16	61	55	53	130	255	220

the error value evaluator based on the BP method will solve the error values which can be implemented with the architecture as shown in Fig. 4.14. The number of total computation cycle of the third pipelined stage is $128 + 92 = 220$, which meets our timing schedule of 259 cycles for each stage.



4.3 Implementation Result

Based on the architecture described above, we proposed a 2.56 Gb/s soft RS (255,239) decoder chip for optical communication systems. Following paragraphs will illustrate the implementation and measurement results of our chip. The comparison between other soft and hard RS decoders will also be shown in this section.

4.3.1 Hardware Analysis

Table (4.2) shows the detail hardware requirement of our three pipelined decision-confined decoder and the four pipelined LCC-based decoder [27], which both methods can provide 0.4 dB coding gain at 10^{-4} CER. The first stage pipelined of our design includes syndrome

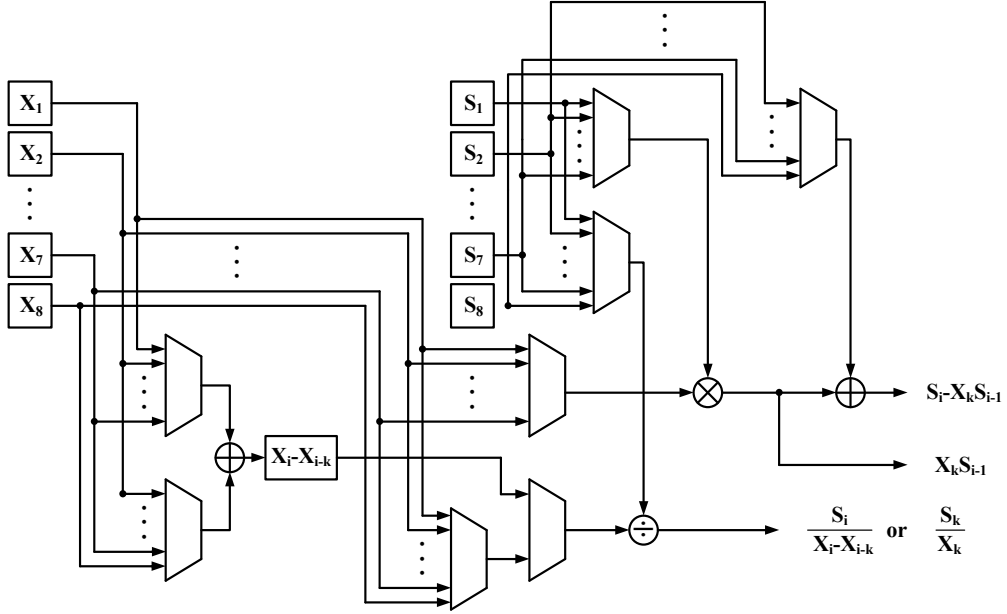


Figure 4.14: BP-based error value evaluator

calculator and reliability evaluator. The syndrome updater and KES based on half-iteration RiBM algorithm compose the second stage. The final stage consists of Chien Search and error value evaluator. In LCC-based decoder, re-encoder and interpolation are the first and second pipelined stages respectively. The third stage includes polynomial selection, Chien search and Forney's algorithm. The erasure decoder then composes the final stage of the decoder.

To normalize the area cost with composite field arithmetic, each $GF(2^8)$ constant multiplier consists 20 XOR gates and each $GF(2^8)$ variable multiplier requires 100 XOR gates. Each $GF(2^8)$ adder occupies 8 XOR gates, each Mux or memory cell has the same area as an XOR, and each register needs about three times of the area of an XOR. Accordingly, with only half latency for each pipeline and less pipelined stages, our design consumes around 22534 XOR gates while the LCC-based decoder occupies about 38671 XOR gates, which achieves more than 40% area reduction. Note that the assumption is even not including the cost of decision making unit consumed in [27].

Table 4.2: Comparison of Time and Hardware Complexity with Soft RS Decoder

	$GF(2^8)$ Constant Multiplier	$GF(2^8)$ Variable Multiplier	$GF(2^8)$ Adder	Mux (bits)	ROM (bytes)	RAM (bytes)	Reg. (bits)	Latency (each stage) (# of clock cycle)
Decision-confined ($\eta = 5$)								
Syndrome Calculator	16	0	16	128	0	0	128	256
Reliability Evaluator	0	0	0	200	0	0	90	259
Syndrome Updater	0	4	16	288	8+256	0	128	8×32
Half-iteration RiBM	0	62	37	464	0	0	296	8×32
Chien Search	16	0	16	64	0	0	64	128
Error Value Evaluator	0	2	2	448	256	0	128	92
Pipelined Memory	0	0	0	0	0	256×3	0	-
Total	32	68	87	1592	520	256×3	834	259
LCC [27] ($\eta = 3$)								
Re-encoder	0	21	39	448	512	0	600	528
Interpolation	0	14	12	87	0	68	166	525
Polynomial Selection	0	8	8	139	0	0	264	23
Chien Search	8	0	8	0	0	0	128	239
Forney's Algorithm	0	2	2	136	256	0	24	152
Erasurer Decoder	0	21	39	299	256	0	424	528
Total	8	66	108	1109	1024	68+256×8	1606	528

4.3.2 Chip Specification

Fig. 4.15 shows our decoder chip and Table (4.3) is the description of the measurement result. With 90-nm standard CMOS process, the total gate count is 45.3 K excluding the FIFO memory, which is used as a buffer for soft input signal. The core size is $465 \times 465 \mu\text{m}^2$. Fig. 4.16 is the shmoo plot of our chip. The operating frequency can achieve up to 320 MHz and the maximum throughput is 2.56 Gb/s which can fit well for 10-40 Gb/s with 16 RS decoders in optical fiber systems and 2.5 Gb/s GPON applications. Furthermore, the average chip power consumption operated at 320 MHz is 19.6 mW with 1.0 V supply voltage.

Since our proposal, in our understanding, is the first soft RS decoder chip, Table (4.4) illustrates the implementation results of our soft RS decoder with other hard RS decoders. Implemented in 90nm CMOS process, our chip with 45.3K gates is comparable with a conventional hard decoder. Moreover, it can meet the throughput requirement of optical communication systems and provide 0.4 dB coding gain over hard decoders at 10^{-4} CER.

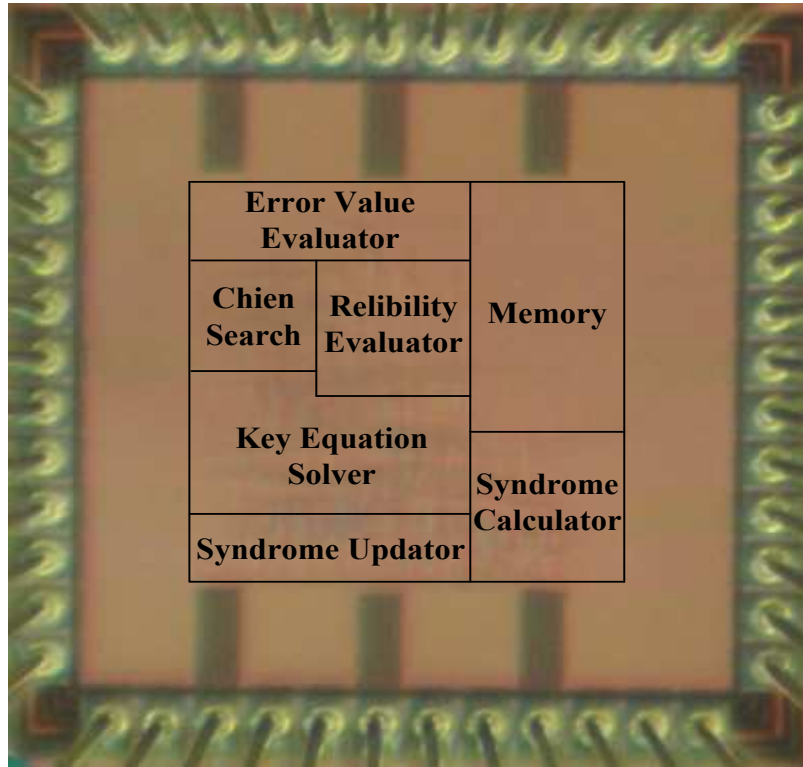


Figure 4.15: Microphoto of soft RS (255,239) chip

Table 4.3: Measurement Result of Decision-confined Soft RS (255,239) Decoder

Technology	90 nm
Total # of Gate Count	45.3 K
Core Size (μm^2)	465 \times 465
Clock Rate (MHz)	320
Throughput (Gb/s)	2.56
Power (mW)	19.6

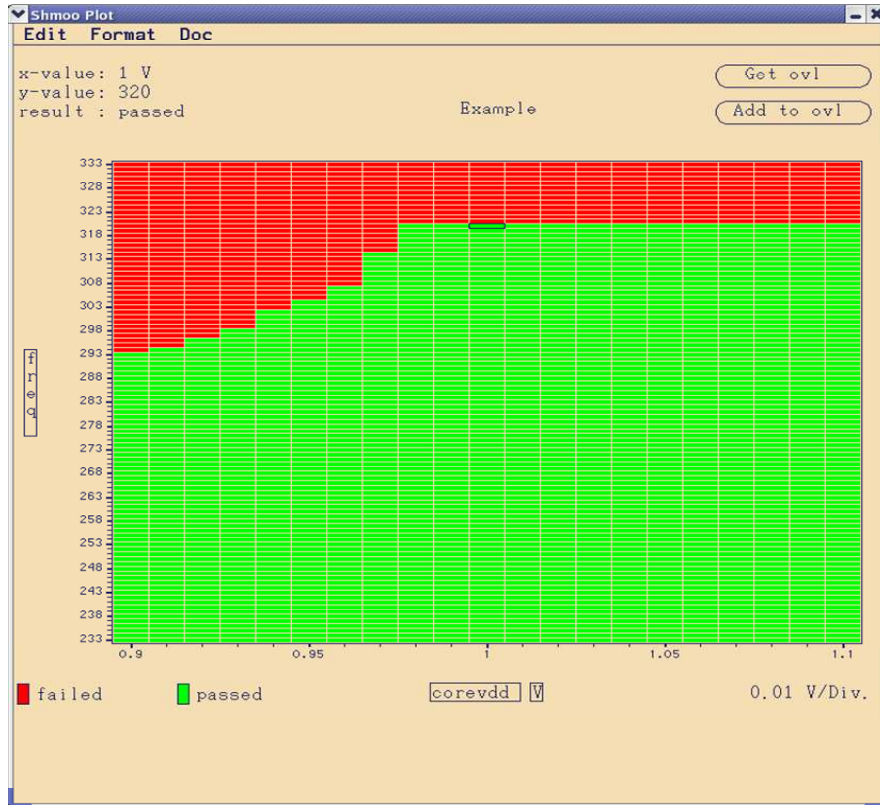


Figure 4.16: Shmoo plot of proposed decoder chip

Table 4.4: Comparison of Time and Hardware Complexity with Hard Decision RS Decoder

	Proposed	pRiBM [31]	pDCME [12]	DCME [11]
Code Type	Soft RS (255, 239)	Hard RS (255, 239)	Hard RS (255, 239)	Hard RS (255, 239)
Technology	90nm	90nm	0.13 μ m	0.25 μ m
Operation Frequency	320MHz (Measurement)	690MHz (Synthesis)	660MHz (Synthesis)	200MHz (Synthesis)
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	-	-	-

Chapter 5

Conclusion and Future Works

5.1 Conclusion

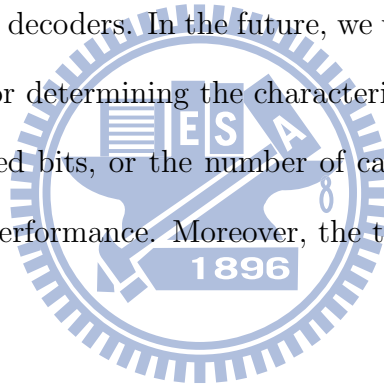
To provide an area efficient and more powerful error correcting capability for optical communication systems, this thesis proposes a novel decision-confined decoding algorithm and its area-efficient architecture for soft RS codes. By confining the degree of error-locator polynomial, our approach determines the more likely candidate sequence which leads to only one candidate sequence being decoded. From our simulation, our method, for RS (255,239) codes, can achieve 0.4 dB coding gain at 10^4 CER over hard decoders.

Unlike Chase-type methods using several hard RS decoders and determining the most probable candidate, our proposal only demands one, leading to significant hardware complexity reduction. By using Gray code based bit-flipping method, which leads to only one bit of these LRPs flipped between each successive candidate, the syndrome for the next candidate can be updated with much more efficient method without recalculating it. In order to meet our timing schedule, we combine the advantage of half-iteration BM and RiBM algorithm and proposed a half-iteration RiBM algorithm and its homogeneous architecture. Moreover, by removing the calculation of error evaluator polynomial and applying BP-based method to compute the error values, the hardware cost can be further reduced. According to the

measurement results, the proposed soft RS decoder can achieve 2.56 Gb/s throughput with 45.3 K gate. As a result, our proposal can fully meet the criterion of optical communications applications and provide more powerful correcting ability with a high-speed and area efficient solution to support longer transmission distance.

5.2 Future Works

Although our proposal can provide an area-efficient RS decoder with better performance gain over traditional hard RS decoders, we still have some design challenge for improvement. Compared with Chase-type methods, our design needs to flip more LRPs to achieve the competitive coding gain, leading to double operations of KES. Therefore the critical path will also be doubled over hard decoders. In the future, we will investigate new approaches to find more efficient methods for determining the characteristic of out of correction. If it can be done, the number of flipped bits, or the number of candidate sequence will be reduced while maintaining the error performance. Moreover, the throughput and the hardware cost can also be enhanced.



Bibliography

- [1] H. C. Chang, C. B. Shung, and C. Y. Lee, “A Reed-Solomon Product-code (RS-PC) Decoder Chip for DVD Applications,” *IEEE J. Solid-State Circuits*, vol. 36, no. 2, pp. 229–238, Feb. 2001.
- [2] S. B. Wicker and V. K. Bhargava, “Reed-Solomon Codes and Their Applications,” *New York: IEEE Press*, 1994.
- [3] *Forward Error Correction for Submarine Systems*, ITU-T Std. G.975, 1996.
- [4] *Gigabit-capable Passive Optical Networks (G-PON): Transmission convergence layer specification*, ITU-T Std. G.984.3, 2008.
- [5] Q. Mao, “Development Progress of 40 Gb/s (STM-256) SDH Optical System in China,” *China Communications, Feature Articles*, Dec. 2005.
- [6] I. Reed and G. Solomon, “Polynomial codes over certain finite fields,” *J. Soc. Indust. and Appl. Math*, vol. 8, no. 2, pp. 300–304, June 1960.
- [7] E. R. Berlekamp, “Algebraic coding theory. new york: Mcgraw-hill,” 1968.
- [8] I. S. Reed, M. T. Shih, and T. K. Truong, “VLSI Design of Inverse-Free Berlekamp-Massey Algorithm,” *Proc. Inst. Elect. Eng*, vol. 138, pp. 295–298, Sept. 1991.
- [9] 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.

- [10] H. Lee, "High-speed VLSI Architecture for Parallel Reed-Solomon Decoder," *IEEE Trans. VLSI Syst.*, vol. 11, no. 2, pp. 288–294, 2003.
- [11] 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, 2006.
- [12] S. Lee, H. Lee, J. Shin, and J.-S. Ko, "A High-Speed Pipelined Degree-Computationless Modified Euclidean Algorithm Architecture for Reed-Solomon Decoders," *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, pp. 901–904, May 2007.
- [13] R. Chien, "Cyclic decoding procedure for the Bose-Chaudhuri-Hocquenghem codes," *IEEE Trans. Inform. Theory*, vol. IT-10, pp. 357–363, Oct. 1964.
- [14] G. D. Forney, "Generalized Minimum Distance Decoding," *IEEE Trans. Inform. Theory*, vol. 12, pp. 125–131, Apr. 1966.
- [15] D. Chase, "A Class of algorithms for decoding block codes with channel measurement information," *IEEE Trans. Inform. Theory*, vol. 18, no. 1, pp. 170–182, Jan. 1972.
- [16] M. Lalam, K. Amis, and D. Leroux, "On the use of Reed-Solomon codes in space-time coding," *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, pp. 31–35, Sept. 2005.
- [17] H. Tang, Y. Liu, M. Fossorier, and S. Lin, "On combining chase-2 and GMD decoding algorithms for nonbinary block codes," *IEEE Communications Letters*, vol. 5, no. 5, pp. 209–211, May 2001.
- [18] S. W. Lee and B. V. K. V. Kumar, "Soft-Decision Decoding of Reed-Solomon Codes Using Successive Error-and-Erasure Decoding," *IEEE GLOBECOM 2008*, pp. 1–5, 2008.

- [19] V. Guruswami and M. Sudan, “Improved decoding of Reed-Solomon and algebraic-geometry codes,” *IEEE Trans. Inform. Theory*, vol. 45, pp. 1757–1767, 1999.
- [20] W. Gross, F. Kschischang, R. Koetter, and R. Gulak, “A VLSI architecture for interpolation in soft-decision list decoding of Reed-Solomon codes,” *IEEE Workshop on Signal Processing Systems (SIPS)*, pp. 39–44, Oct. 2002.
- [21] 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. 264–257, Jan. 2000.
- [22] R. Koetter and A. Vardy, “Algebraic Soft-Decision Decoding of Reed-Solomon Codes,” *IEEE Trans. Inform. Theory*, vol. 49, no. 11, pp. 2809–2825, 2003.
- [23] J. Zhu and X. Zhang, “High-speed re-encoder design for algebraic soft-decision Reed-Solomon decoding,” *Proc. IEEE International Symposium on Circuits and Systems*, May 2010.
- [24] A. V. R. Koetter, J. Ma and A. Ahmed, “Efficient interpolation and factorization in algebraic soft-decision decoding of Reed-Solomon codes,” *IEEE Int. Symp. Inform. Theory*, July 2003.
- [25] 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.
- [26] J. Bellorado and A. Kavcic, “A low-complexity method for Chase-type decoding of Reed-Solomon codes,” *Proc. ISIT*, pp. 2037–2041, Jul. 2003.

- [27] X. Zhang, “High-speed VLSI architecture for low-complexity Chase soft-decision Reed-Solomon decoding,” *IEEE Inform. Theory and Application Workshop*, pp. 422–430, Feb 2009.
- [28] A. Björck and V. Pereyra, “Solution of Vandermonde Systems of Equations,” *Math. Computation*, vol. 24, pp. 893–903, Oct. 1970.
- [29] D. Knuth, “The Art of Computer Programming, Vol. 3 - Sorting and Searching.” 1973.
- [30] A. Raghupathy and K. J. R. Liu, “Algorithm-Based Low-Power/High-Speed Reed-Solomon Decoder Design,” *IEEE Trans. on Circuits and Systems-II: Analog and Digital Signal Processing*, vol. 47, no. 11, pp. 1254–1270, Nov. 2000.
- [31] J.-I. Park, K. Lee, C.-S. Choi, and H. Lee, “High-speed low-complexity Reed-solomon decoder using pipelined berlekamp-massey algorithm,” *IEEE Int. SoC Design Conference (ISOCC)*, pp. 452–455, 2009.

