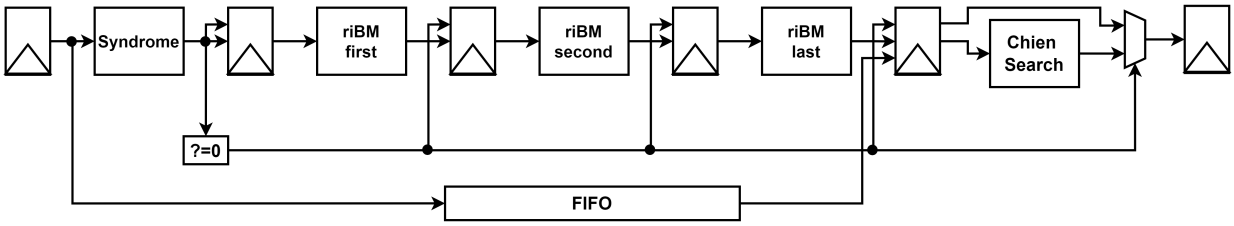
CHALMERS
UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG



High-throughput Low-power BCH Decoder for Short-reach Optical Communication System

Master's thesis in Embedded Electronic System Design

XU WANG

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

High-throughput Low-power BCH Decoder for Short-reach Optical Communication System

XU WANG

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

High-throughput Low-power BCH Decoder for Short-reach Optical Communication System
XU WANG

Chalmers advisor: Lars Svensson, Computer Science and Engineering
Company advisor: Christoffer Fougstedt, Ericsson
Examiner: Per Larsson-Edefors, Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Unrolled pipeline riBM BCH decoder

Typeset in LaTeX
Gothenburg, Sweden 2023

High-throughput Low-power BCH Decoder for Short-reach Optical Communication System
XU WANG
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Short-reach optical systems in data centers or high-performance computing systems are often subjected to harsh environmental conditions, especially high temperatures. These elevated temperatures can lead to various challenges, including high bit-error rates (BER) and lower data rates. In order to mitigate those problems, forward error corrections (FEC) can be employed. A suitable FEC can also help to reduce the total system power by relaxing the optical modulation amplitude (OMA) requirement at the receiver. FEC consists of two functional blocks, an encoder and a decoder. Compared with the encoder, the decoder significantly influences area usage and power consumption of the FEC. Among FEC alternatives, Bose–Chaudhuri–Hocquenghem (BCH) is favorable regarding the hardware complexity and error correction ability. In this work, we propose a BCH decoder circuit with error correction ability $t = 3$ and 4, and implement its corresponding encoder in order to evaluate the overall system power-dissipation reduction. We also propose an estimation model based on MATLAB to predict the performance of different system architectures and key equation solving algorithms. Based on the estimation result, a reformulated inversionless Berlekamp-Massey (riBM) algorithm is selected for the key equation solver and the unrolled pipeline (UP) system architecture is implemented to increase the throughput and reduce the power consumption. A system based on iterative parallel (IP) architecture is also implemented as a comparison architecture. Compared with the IP system, the UP system can save up to 20% power at an input BER of 10^{-3} in a 22-nm CMOS process technology. Including the power consumption of encoders and actual-case transmitters, the BCH FEC can significantly reduce the power by a maximum of 58% compared with the uncoded system in a 190-mW transmitter system.

Keywords: optical fiber system, forward error correction, lower power, high throughput, Bose–Chaudhuri–Hocquenghem (BCH), very large-scale integration (VLSI).

Acknowledgements

I am grateful to my parents for their unwavering support in enabling me to pursue my master's degree. I would also thanks to my Chalmers supervisor, Prof. Lars Svensson, who gives me this thesis opportunity and insight supervision. I would like to acknowledge the invaluable assistance and mentorship provided by Dr. Christoffer Fougstedt from Ericsson, who has been instrumental in helping me navigate and overcome numerous challenges and sharing his experiences which is crucial in adding depth and richness to the thesis work. I would like to extend my sincere appreciation to my examiner, Prof. Per Larsson-Edefors for his constructive suggestions and meticulous examination.

Xu Wang, Gothenburg, June 2023

Contents

1 Introduction **1**
- 1.1 Purposes and Goals . . . 2
- 1.2 Thesis Outline . . . 2

2 Technical Background **3**
- 2.1 Fiber Communication System . . . 3
- 2.2 Galois Field Arithmetic . . . 4
  - 2.2.1 Group . . . 4
  - 2.2.2 Field . . . 4
  - 2.2.3 Polynomial . . . 4
  - 2.2.4 Addition . . . 5
  - 2.2.5 Multiplication . . . 5
- 2.3 Forward Error Correction . . . 5
  - 2.3.1 Hamming Code . . . 6
  - 2.3.2 Reed-Solomon Code . . . 6
  - 2.3.3 BCH code . . . 6
- 2.4 Encoder . . . 7
- 2.5 Decoder . . . 8
  - 2.5.1 Syndrome-Table Decoding . . . 8
  - 2.5.2 Algebraic decoding . . . 9
    - 2.5.2.1 Berlekamp-Massey Algorithm . . . 11
    - 2.5.2.2 Peterson Algorithm . . . 11
  - 2.5.3 Chien Search . . . 13

3 Methods **15**
- 3.1 Bit-operation FEC Simulator . . . 15
- 3.2 Estimation Model . . . 16
- 3.3 VHDL Code Generator . . . 16
- 3.4 Auto-synthesis System . . . 16
- 3.5 BSC Simulator . . . 17

4 Design **19**
- 4.1 Power Evaluation System . . . 19
- 4.2 Galois Field Multiplier . . . 20
- 4.3 Berlekamp-Massey Decoder . . . 23
  - 4.3.1 Syndrome Calculation . . . 24

4.3.2 Key Equation Solver 24
4.3.3 riBM Structure 27
4.3.4 Parallel Chien Search 30
4.4 System Architecture 31
4.4.1 Direct Iteration 31
4.4.2 Parallel Iteration 33
4.4.3 Unrolled Pipeline 35
4.4.4 State of the Art 37

5 Results **39**
5.1 GF Multiplier 39
5.2 Syndrome Component 40
5.3 Key Equation Solver 41
5.3.1 $t = 3$ riBM Component 41
5.3.2 $t = 4$ riBM Component 42
5.4 Chien Search 43
5.5 BSC Simulation 44
5.6 Post-FEC Power Saving 45

6 Conclusion **47**

Bibliography **49**

Acronyms

BCH Bose-Chaudhuri-Hocquenghem.
BER bit error rate.
BM Berlekamp-Massey.
BSC Binary symmetric channel.

CMOS complementary metal–oxide–semiconductor.
CRT Chinese remainder theorem.
CS-RiBM compensated simplified RiBM.

DI Direct iteration.

FDSOI Fully depleted silicon on insulator.
FEC Forward error correction.
FIFO First-In-First-Out.

GF Galois Field.

HD Hard-Decision.

iBM Inversionless BM.

LCM Least common multiple.
LDPC Low-Density Parity-Check.
LFSR Linear feedback shift register.
LUT Look-Up table.

MUX Multiplexer.

OMA optical modulation amplitude.

PC Personal computer.
PE0 Processor engine zero.
PE1 Processor engine one.
PI Parallel iteration.

riBM Reformulated inversionless BM.
RS Reed-Solomon.
RTL Register-transfer level.

SD Soft-Decision.

TiBM Truncated iBM.

UP Unrolled pipeline.

VCD Value change dump.
VCSEL Vertical-cavity surface-emitting laser.
VHDL Very high-speed integrated circuit hardware description language.

XOR Exclusive OR.

1

Introduction

Short-reach interconnections are commonly used in network applications, such as data centers, and in connecting equipment that needs high-bandwidth links, such as front-haul links in communication networks. Vertical-cavity surface-emitting lasers (VCSELs) are a popular choice for such systems [1] due to their ability to be directly modulated at high speeds, which enables compact transceiver modules with small footprints and energy- and cost-efficient interconnects. Even with advanced cooling techniques, the operating environment for the VCSELs can reach a temperature above 70 °C [2] and even worse in an automotive system (maximum 125 °C). In addition, data centers and high-performance computers often necessitate supercomputing capabilities, which in turn rely on high bandwidth. Hence, those systems not only require a high throughput but also reliability at a high temperature.

Forward error correction (FEC) is a widely-used error correction technique family of detecting and correcting errors during transmission. The FEC adds redundant bits to the original data, which allows the receiver to detect and correct errors without requesting the transmitter to resend the data. FEC is commonly used in memory systems [3, 4, 5] to correct the random errors and burst errors that may occur during data reading and writing and improve data storage reliability. Moreover, in communication links, FEC can help to decrease the random errors added by the transmission medium, and push the data rate further without negatively impacting the bit error rate (BER) [6].

Adding FEC to the fiber-optic communication system may introduce hardware overheads, such as more area usage and power consumption; however, a suitable FEC system may reduce the overall system power, offering overall power saving for the total system [7]. Incorporating FEC can relax the optical modulation amplitude (OMA) requirement at the receiver while maintaining the same BER. Relaxation on the receiver reduces the power consumption at the transmitter, resulting in a power-efficient operation point. Szczerba et al. [7] developed an FEC based on Hamming code in a 65-nm complementary metal–oxide–semiconductor (CMOS) process, resulting in a 23% maximum power reduction. Fougstedt et al. [8] demonstrated a low-power Bose-Chaudhuri-Hocquenghem (BCH) FEC in a 28-nm CMOS process technology which offered a 25% total power saving with an error correction ability of two. In a harsh environment, such as high temperature, an error correction ability below two may be insufficient, thus an FEC with a higher error correction ability is required.

Regarding the latency, BCH code is a suitable candidate with moderate latency and it also offers a lower complexity compared with symbol based Reed-Solomon (RS), which can correct multiple errors but with higher latency. Hamming codes are indeed the low-latency option, but they can only correct one error which is insufficient for meeting the high error correction requirements. Therefore, BCH codes stand out as a probable option for applications where both low latency and high correction abilities are critical requirements.

1.1 Purposes and Goals

The purpose of this thesis work is to improve the reliability and throughput of short-reach optical communication systems operating in high-temperature environments by incorporating low-power FECs. By integrating low-power FECs, the systems can operate at a power-efficient point and mitigate transmission errors, ensuring reliability in high-temperature conditions.

The final goal is to implement a BCH decoder based on the Berlekamp-Massey (BM) algorithm with the error correction ability of three and four with low latency and a throughput of 56 Gbits/sec. The final goal can be divided into five sub-goals:

- Setting up a model comparing the uncoded BER (without FEC) and post-FEC BER (with FEC) to evaluate the power relaxation offered by the FEC.
- Implementing an efficient Galois Field (GF) multiplier component considering the area usage and timing, and evaluating the performance regarding different primitive polynomials.
- Implementing a suitable key equation solver algorithm with error correction ability of three and four on a 22-nm CMOS technology considering the timing performance, hardware usage, and power efficiency.
- Evaluating the performance considering the power relaxation for the total system, latency, and throughput.
- Appraising and comparing the performance of different error-correction abilities and methods.

1.2 Thesis Outline

In this thesis report, Chapter 2 provides technical backgrounds to the readers, including introductions to a simple fiber system, GF arithmetic, common FEC codes, and an FEC system based on binary BCH codes. In the following Chapter 3, the design methods are introduced from the Matlab models to very high-speed integrated circuit hardware description language (VHDL) codes. Then, Chapter 4 finalizes the design of this thesis project by comparing different GF multipliers, different key-solver typologies, and system architectures. Chapter 5 demonstrates the synthesis and implementation results from each individual component to different decoder structures. In the end, Chapter 6 summarizes the results.

2

Technical Background

This chapter introduces the technical background of the fiber communication system, the GF arithmetic, the common FEC codes, and an FEC system based on binary BCH codes.

2.1 Fiber Communication System

A fiber communication system transmits information from the transmitter to the receiver through an optical fiber. The message is carried by the infrared light generated from a laser, typically VCSEL, which offers high speed, improved power, and spectral proprieties at a low cost. A simple fiber system is shown in figure 2.1.

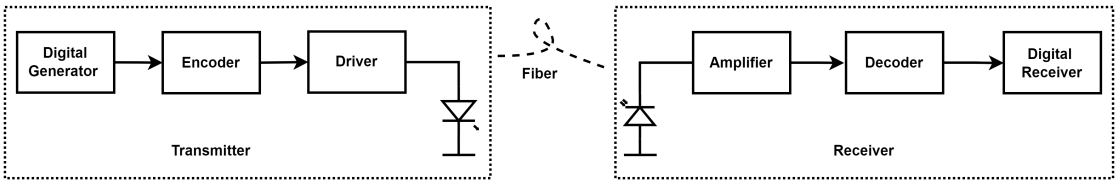


Figure 2.1: A simplified fiber system

The transmitter converts electrical signals into an optical format and transmits them over fiber optic cables. The specific components present in the transmitter are determined by the system's requirements. A typical transmitter includes a data generator, an FEC encoder, a driver, and a laser. The data generator produces the transmitted messages in an electrical format, while the driver converts the voltage to current to drive the laser. The laser then emits infrared light based on the current input. FEC is commonly utilized in long-haul transmissions but is gaining increasing interest in short-haul systems, given their capacity to enhance system reliability [9] and reduce power consumption [7]. A modulator is often added to the transmitter depending on the throughput requirement.

The receiver in turn converts the optical signals back to electrical signals. It may also need to correct errors introduced by the channel. Just like the transmitter, the complexity of the receiver can vary, but generally, it comprises a photo-diode, an amplifier, an FEC decoder, and a digital receiver. Based on the modulation format of the transmitted message, a demodulation component may also be required to demodulate the message. The photo-diode receives the optical signal and transforms

it into an electrical current, which is then amplified by the following amplifier. The FEC decoder then corrects any errors in the message, and the data from the decoder is sent to a receiving terminal, such as a personal computer (PC) interface.

2.2 Galois Field Arithmetic

A Galois field or finite field is a field that contains a finite number of elements [10]. A Galois field (GF) contains a prime number p or a prime power p^m number of elements, and the GF with p^m elements can also be referred to as $GF(p^m)$. The $GF(2)$ is the simplest finite field, which plays an important role in communication and storage systems. There are also addition, subtraction, multiplication, and division operations in this field but following different rules compared with the real field. To better understand the GF field, some important definitions and representations are provided first and then the addition and multiplication in this field are explained.

2.2.1 Group

A group is a set of elements G equipped with an operation $\cdot$ that satisfies the following properties:

- Closure: for any $a \in G, b \in G$, the result of $a \cdot b$ is also belong to G.
- Associative: for any $a, b, c \in G$, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.
- Identity: There is an identity element $e \in G$ where for all $a \in G$, $a \cdot e = e \cdot a = a$.
- Inverse: For any $a \in G$, there is an inverse $a^{-1} \in G$ such that $a \cdot a^{-1} = e$.

If the operation $\cdot$ is commutative, then the group is commutative or abelian.

2.2.2 Field

If the group is commutative and with two operations, additive operation $+$ and multiplicative operation $\cdot$, such that:

- Distributive law: for all $(a + b) \cdot c = (a \cdot c) + (b \cdot c)$.

then this group is a field F.

2.2.3 Polynomial

The elements in GF can be represented in two ways, power basis and standard basis (polynomial basis). The standard basis representation is used in this work and is constructed by a primitive polynomial. Below are definitions needed to understand the primitive polynomial:

- Primitive polynomial: a primitive polynomial is a minimal polynomial associated with a primitive element of GF.
- Minimal polynomial: a minimal polynomial is the lowest-degree irreducible polynomial that, when evaluated with the primitive element, results in zero.

- Irreducible polynomial: If a polynomial over GF cannot be factored into lower-degree polynomials, then it is irreducible.

2.2.4 Addition

Addition and subtraction are straightforward. These operations are performed by first adding or subtracting the two polynomials, if the element representation is polynomial, and then modulo the primitive number p. Assume the two addends are a and b with the polynomial $a(\alpha) = a_0 + a_1\alpha + \ldots + a_i\alpha^{m-1}$ and $b(\alpha) = b_0 + b_1\alpha + \ldots + b_i\alpha^{m-1}$, respectively. Then the addition can be performed by:

$$c(\alpha) = (a(\alpha) + b(\alpha))_{mod\ p}, \tag{2.1}$$

when p is two, the addition and subtraction can be simplified to exclusive OR (XOR) gates, and the formula 2.1 can be simplified to:

$$c(\alpha) = a(\alpha) \oplus b(\alpha), \tag{2.2}$$

where $\oplus$ denotes the XOR operation.

2.2.5 Multiplication

The multiplication in GF is relatively complex. Multiplication in a finite field is multiplication following a modulo operation with a primitive polynomial $p(x)$ used to define this field. Assume the same operators as the addition a and b, and the $p(x)$ with a polynomial of $p(x) = p_0 + p_1x + \ldots + p_ix^{i-1}$, where $0 \leq i \leq n$. The multiplication denotes as:

$$c(\alpha) = (a(\alpha) \cdot b(\alpha))_{mod\ p(x)} \tag{2.3}$$

which can be rewritten as:

$$c(\alpha) = b_0a(\alpha)_{mod\ p(x)} + b_1[a(\alpha)\alpha_{mod\ p(x)}] + \ldots + b_{m-1}[a(\alpha)\alpha^{m-1}_{mod\ p(x)}] \tag{2.4}$$

Many architectures [11, 12] have been proposed based on formula 2.4 to improve the hardware efficiency. Mastrovito multipliers were proposed in [13] to reformulate formula 2.4, aiming at improving the throughput, and different optimized Mastrovito multipliers were proposed [14, 15] to improve the performance.

2.3 Forward Error Correction

FEC is a type of error correction method and is commonly used in communication systems and memory systems to increase the reliability and efficiency of the systems. FEC typically comprises an encoder and a decoder. The encoder appends redundant bits calculated by mathematical algorithms, facilitating the decoder to decode the

message. According to the error patterns, the decoder detects and corrects the errors in the contaminated message. There are different types of FEC codes and their decoding algorithms can be divided into Hard-Decision (HD) and Soft-Decision (SD) [16]. HD is typically more power efficient and less complex compared with SD since the HD quantizes the message before decoding it [6]. However, HD offers lower coding gain, the improvement of signal-to-noise-ratio achieved by FEC over an uncoded system while maintaining the same BER, compared with its counterpart. Among the HD codes, Hamming, BCH, and RS are conventional codes. Although there are more advanced codes such as Low-Density Parity-Check (LDPC) codes and concatenated BCH code, which aim at higher coding gain and throughput, only the simple algebraic codes, Hamming, BCH, and RS are considered in this thesis.

2.3.1 Hamming Code

Hamming code is a type of linear error-correcting code, and it can only correct a signal bit error per block [17]. Hamming code is in fact a special-case BCH code with the correction ability equaling one, $t = 1$. Hamming code is the most lightweight code regarding complexity since a simple Look-Up table (LUT) can be used to decode the received codes. Hamming code can be denoted as (n, k), where n is the length of the codeword and k is the length of the message, and $n - k = m \cdot t$, where $n = 2^m - 1$ and t is the error correction ability. There are different types of Hamming codes regarding the codewords, a simple $(7, 4)$ code appends four bits message with three bits parity check bits to become a seven-bit codeword. While Hamming codes are the most lightweight, the error correction ability is limited to one. Therefore for the system with higher error correction ability requirements, BCH codes and RS codes are used.

2.3.2 Reed-Solomon Code

Reed-Solomon (RS) codes are also linear error-correcting codes and are commonly used in applications where high levels of error correction are required. Different from the BCH code and Hamming code, the RS code is based on symbols [18] rather than bits. RS code can be denoted as RS(n, k, t), where $2 \cdot t = n - k$ and $n = 2^m - 1$, and it can correct t symbol errors. To be more specific, an RS code with t error correction ability can correct at least t bit errors and at most mt bit errors. Because of the symbol operation, the RS codes are efficient at correcting burst errors, but the encoding and decoding of the RS code are more complex than the BCH and Hamming codes. RS codes are widely used in long-haul communication, and RS$(544, 514, 15)$ and RS$(528, 514, 7)$ are recommended in the current Ethernet standard [19].

2.3.3 BCH code

Bose-Chaudhuri-Hocquenghem (BCH) is a subclass of cyclic error-correcting codes and is constructed over a Galois field [10]. Binary BCH codes are commonly used in HD decoding because of their low complexity and moderate error correction ability. BCH code can be represented by BCH(n, k, t) with t error correction ability, the

relationship among these parameters is the same as the Hamming code except $t \geqslant 1$. The complexity of encoding with $t \geqslant 2$ is higher than the Hamming code due to the improved error correction ability, but is relatively lower compared with the RS codes. The complexity of decoding varies because of different decoding methods, such as Berlekamp-Massey (BM) [20, 21], and Peterson [22]. BCH codes are of interest in short-haul communication systems due to the low required error correction ability of those systems.

2.4 Encoder

The encoder is used to encode the original message based on the code type. In this section, a BCH encoding process is illustrated. The encoding begins by finding the minimum-degree generator polynomial $g(x)$ over GF(2). The $g(x)$ is the least common multiple (LCM) of the minimal polynomial $\phi_i(X)$ of α^i (see equation 2.5), where α is a primitive element in GF(2) and $1 \leq i \leq 2t$. Since any even integer number can be expressed as a product of an odd integer and a power of 2, this equation can be simplified to equation 2.6 [10].

$$g(x) = LCM\{\phi_1(X), \phi_2(X), ..., \phi_{2t}(X)\}, \tag{2.5}$$
$$g(x) = LCM\{\phi_1(X), \phi_3(X), ..., \phi_{2t-1}(X)\}. \tag{2.6}$$

There are two kinds of encoding methods: one is non-systematic encoding, where the parity-check bits are factors of the codeword, and the other one is systematic encoding, where the redundancy is a prefix of the message. The latter one is preferred since the message can be directly read out. Systematic encoding can be implemented by [23]:

$$c(x) = m(x) \cdot x^{mt} + Rem(m(x) \cdot x^{mt})_{g(x)}, \tag{2.7}$$

where $c(x)$ is the coefficients $(c_0, c_1, ..., c_n)$ of codeword, $m(x)$ is the coefficients $(m_0, m_1, ..., m_k)$ of message, and the $g(x)$ is the coefficients $(g_0, g_1, ..., g_{n-k})$ of the generator polynomial. From equation 2.7, the encoding is actually a remainder calculation process, which can be performed using a linear feedback shift register (LFSR) with low throughput and a fanout bottleneck.

Parallelism can help to increase the throughput by utilizing a generator matrix $\mathbf{G}$, which is derived from the generator polynomial $g(x)$ [24]. Using the generator matrix, the encoding becomes more straightforward:

$$c(x) = m(x) \cdot \mathbf{G}. \tag{2.8}$$

Since the multiplication of binary bits can be simplified to the connection or disconnection of wires and addition is XOR gate, the encoder is composed of XOR trees.

2.5 Decoder

Compared with the encoder, the decoder is more complex. The decoder detects and corrects the error according to the error patterns. There are two types of decoding methods that are commonly used: syndrome-table decoding and algebraic decoding. Both decoding methods are described below.

2.5.1 Syndrome-Table Decoding

The syndrome-table decoding comprises syndrome calculation and LUT addressing progress (see figure 2.2).

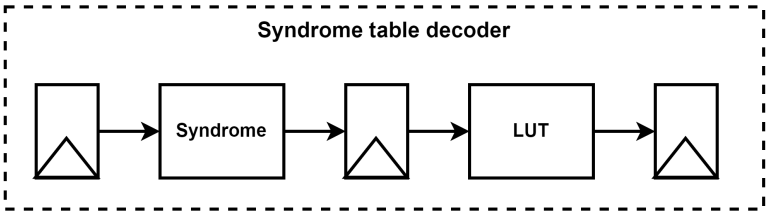


Figure 2.2: A typical syndrome table decoder

Assume a generator polynomial $g(x)$ has $[\alpha, \alpha^2, ..., \alpha^{2t}]$ and their conjugates as roots, thus $g(\alpha^i) = 0$ with $1 \leq i \leq 2t$. Since the $c(x)$ is divisible by $g(x)$, a root of $g(x)$ is also the root of $c(x)$ [10]. Therefore, a relationship can be found:

$$c(x) \cdot (1, \alpha^i, \alpha^{2i}, ..., \alpha^{(2^m-2)i})^T = 0. \tag{2.9}$$

Since $1 \leq i \leq 2t$, a $2t \times (2^m - 1)$ parity-check matrix can be derived by equation 2.9:

$$\mathbf{H} = \begin{bmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{2^m-2} \\ 1 & \alpha^2 & (\alpha^2)^2 & \dots & (\alpha^2)^{2^m-2} \\ 1 & \alpha^3 & (\alpha^3)^2 & \dots & (\alpha^3)^{2^m-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{2t} & (\alpha^{2t})^2 & \dots & (\alpha^{2t})^{2^m-2} \end{bmatrix}. \tag{2.10}$$

From equation 2.9 it can be concluded that for any $c(x)$, there is a relationship that:

$$c(x) \cdot \mathbf{H} = 0. \tag{2.11}$$

The transmission media may add interference $e(x)$ to the sent message, and the received message $r(x)$ will become:

$$r(x) = c(x) + e(x), \tag{2.12}$$

where $e(x) = e_0 + e_1 x + \ldots + e_{2^m-2} x^{2^m-2}$. Because of the interference, the relationship of equation 2.11 is disrupted, and the multiplication of $r(x)$ and $\mathbf{H}$ becomes the syndrome $\mathbf{S}$:

$$\mathbf{S} = (S_1, S_2, ..., S_{2t}) = r(x) \cdot \mathbf{H}^T, \tag{2.13}$$

where $1 \leq i \leq 2t$. From equations 2.12 and 2.11, the syndrome calculation can be rewritten to

$$r(x) \cdot \mathbf{H}^T = (c(x) + e(x)) \cdot \mathbf{H}^T = e(x) \cdot \mathbf{H}^T. \tag{2.14}$$

For the error correction ability $t = 1$, it is efficient to use syndrome-table decoding since there are not that many error patterns. The syndrome $\mathbf{S}$ can be directly mapped to the error patterns through a LUT [25]. However, this method is inefficient for the $t > 1$ given that the minimum-distance decoding of BCH codes requires $2^{n-k} - 1$ syndrome-table entries.

2.5.2 Algebraic decoding

Compared with syndrome-table decoding, algebraic decoding is more efficient when the error correction ability is more than one. The algebraic decoding also begins with a syndrome calculation to get the syndrome $\mathbf{S}$. The next step is to use the syndrome to determine the error-location polynomial $\Lambda(x)$ from the Newton identities. This process is also called key-equation solving (see figure 2.3).

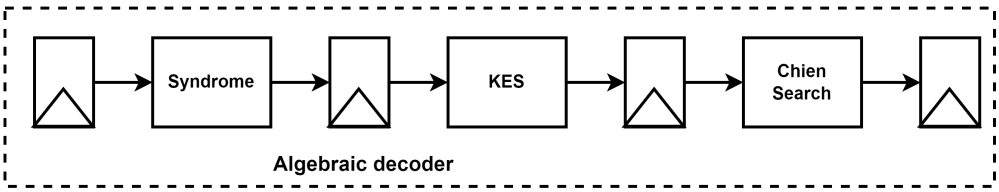


Figure 2.3: A typical algebraic decoder, KES is the key equation solver

There are different types of algebraic algorithms to solve the key equation, the commonly used ones are Berlekamp-Massey (BM) algorithm [20, 21] and Peterson's algorithm [22], which are illustrated below. After the error location polynomial is found, the Chien search [26] is applied to determine the roots and error pattern. To retrieve the original message, the error pattern is added to the received code.

According to equation 2.11 and 2.12, when there are errors added by the transmission media, equation 2.11 is not satisfied. Because $c(\alpha^i) = 0$, equation 2.13 can be simplified to:

$$S_i = e(\alpha^i). \tag{2.15}$$

Let us assume the errors happen at the locations $j_1, j_2, ..., j_z$, where $0 \leq j_1 < j_2 \ldots < j_z < n$. Then the error polynomial becomes:

$$e(x) = x^{j_1} + x^{j_2} + \ldots + x^{j_z}. \tag{2.16}$$

And the syndrome becomes:

$$\begin{aligned} S_1 &= \mathbf{e}(\alpha) = \alpha^{j_1} + \alpha^{j_2} + \cdots + \alpha^{j_z} \\ S_2 &= \mathbf{e}\left(\alpha^2\right) = \left(\alpha^{j_1}\right)^2 + \left(\alpha^{j_2}\right)^2 + \cdots + \left(\alpha^{j_z}\right)^2 \\ &\vdots \\ S_{2t} &= \mathbf{e}\left(\alpha^{2t}\right) = \left(\alpha^{j_1}\right)^{2t} + \left(\alpha^{j_2}\right)^{2t} + \cdots + \left(\alpha^{j_z}\right)^{2t}. \end{aligned} \tag{2.17}$$

Thus with the relationship between syndrome and error location, the problem becomes how to determine the error locations using the syndrome. Let us simplify the above set of equations by assuming:

$$\beta_l = \alpha^{j_l}. \tag{2.18}$$

So equation 2.17 can be optimized to:

$$\begin{aligned} S_1 &= \beta_1 + \beta_2 + \cdots + \beta_z \\ S_2 &= \beta_1^2 + \beta_2^2 + \cdots + \beta_z^2 \\ &\vdots \\ S_{2t} &= \beta_1^{2t} + \beta_2^{2t} + \cdots + \beta_z^{2t}. \end{aligned} \tag{2.19}$$

The above set of equations is nonlinear thus solving it directly will be very difficult. But we can solve it in an indirect way, first assuming the error-location polynomial:

$$\Lambda(x) = (1 + \beta_1 x)(1 + \beta_2 x) \ldots (1 + \beta_z x), \tag{2.20}$$

$$= \Lambda_0 + \Lambda_1 x + \ldots + \Lambda_z x^z, \tag{2.21}$$

which has $\beta_1^{-1}, \beta_2^{-1}, ..., \beta_z^{-1}$ as roots, where

$$\begin{aligned} \Lambda_1 &= \beta_1 + \beta_2 + \cdots + \beta_z \\ \Lambda_2 &= \beta_1\beta_2 + \beta_1\beta_3 + \cdots + \beta_{z-1}\beta_z, \\ \Lambda_3 &= \beta_1\beta_2\beta_3 + \beta_1\beta_2\beta_4 + \cdots + \beta_{z-1}\beta_{z-2}\beta_{z-3}, \\ &\vdots \\ \Lambda_z &= \beta_1\beta_2 \cdots \beta_z. \end{aligned} \tag{2.22}$$

Then the problem becomes determining the coefficients of the error location polynomial $\Lambda(x)$, then according to the error location polynomial, the roots can be founded. There are different methods to find the coefficients of the error location polynomial $\Lambda(x)$, such as BM and Peterson's algorithm, but the theory is the same. The roots can be determined by solving the Newton identities:

$$\begin{aligned} &S_1 + \Lambda_1 = 0 \\ &S_2 + \Lambda_1 S_1 + 2\Lambda_2 = 0 \\ &S_3 + \Lambda_1 S_2 + \Lambda_2 S_1 + 3\Lambda_3 = 0 \\ &\vdots \\ &S_z + \Lambda_1 S_{z-1} + \Lambda_2 S_{z-2} + \ldots + \Lambda_{z-1} S_1 + z\Lambda_z = 0. \end{aligned} \tag{2.23}$$

If we can solve the Newton identities, then we can find the coefficients of the error location polynomial and in the end the roots. In general, there may be more than one error pattern that satisfies the Newton identities [10], so the most-probable one should be found. In the binary symmetric channel (BSC), the problem becomes finding the coefficient which satisfies the Newton identities with a minimum degree.

2.5.2.1 Berlekamp-Massey Algorithm

Berlekamp-Massey (BM) algorithm is actually a process to iteratively find the minimum degree polynomial that satisfies the Newton identities. It takes $2t$ times to verify and adjust the minimum degree polynomial according to the value of discrepancy d_k. The calculation of d_k is actually a process to verify the first $k+1$ Newton identities 2.23:

$$d_k = S_{k+1} + \Lambda_1^{(k)} S_k + \Lambda_2^{(k)} S_{k-1} + \ldots + \Lambda_{l_k}^{(k)} S_{k+1-l_k}, \tag{2.24}$$

where l_k is the degree of current error correction polynomial, and $\Lambda^k(x)$ is the current minimum degree polynomial which satisfies the first k Newton identities:

$$\Lambda^{(k)}(x) = \Lambda_0^{(k)} + \Lambda_1^{(k)} x + \ldots + \Lambda_{l_k}^{(k)} x^{l_k}, \tag{2.25}$$

If $d_k = 0$ then it means the current error correction polynomial also satisfies the $(k+1)$th identity and there is no adjustment needed for the current error correction polynomial. However, if the discrepancy is nonzero, which means current $\Lambda^k(x)$ cannot satisfy the $(k+1)$th identity and a correction term should be added to $\Lambda^k(x)$ and the $\Lambda^{(k+1)}(x)$ becomes:

$$\Lambda^{(k+1)}(x) = \Lambda^{(k)}(x) + d_k d_i x^{k-i} \Lambda^{(i)}(x), \tag{2.26}$$

where i is a step prior to the k-th step with $d_i \neq 0$ and $i - l_k$ has the largest value. d_k and $\Lambda^{(i)}(x)$ is discrepancy and minimum error location polynomial for the i-th step. l_k increases one when the correction is applied to the $\Lambda^{(k)}(x)$. The final minimum error location polynomial becomes the $\Lambda^{(2t)}(x)$ at the $2t$-th iteration. The BM algorithm can be summed in algorithm 1.

It can be found that the most hardware and time-consuming parts of the BM algorithm are the calculation of discrepancy, and the inversion of d_i is unfriendly to the hardware, especially when m is larger. Determining the final $\Lambda(x)$ spends $2t$ cycles, leading to high latency. Many improved algorithms have been proposed to decrease the latency and hardware overhead, for example, simplified BM decreases the iteration time to $2t$ [10] by skipping the odd iterations because the discrepancy is always zero, and inversionless BM (iBM) removes the inversion of d_i and leads to a hardware efficient design [27].

2.5.2.2 Peterson Algorithm

Peterson algorithm [28] can remove the hardware-expensive inversion and reduce the latency for the binary BCH codes. From equation 2.23, the coefficients of the error

Algorithm 1: The original Berlekamp-Massey Algorithm for BCH code

input: S_i $(1 \leq i \leq 2t)$

initialization: $k = -1,\ \Lambda^{(-1)}(x) = 1,\ d_{-1} = 1,\ l_{-1} = 0,\ k - l_{-1} = -1$
$k = 0,\ \Lambda^{(0)}(x) = 1,\ d_0 = 1,\ l_0 = 0,\ k - l_0 = 0$

begin: for $k = 0, 1, 2, ..., 2t - 1$
$d_k = S_{k+1} + \Lambda_1^{(k)} S_k + \Lambda_2^{(k)} S_{k-1} + \ldots + \Lambda_{l_k}^{(k)} S_{k+1-l_k}$
if $d_k = 0$ then
$\Lambda^{(k+1)}(x) = \Lambda^{(k)}(x)$
$l_{k+1} = l_k$
else
$\Lambda^{(k+1)}(x) = \Lambda^{(k)}(x) + d_k d_i^{-1} x^{k-i} \Lambda^{(i)}(x)$
$l_{k+1} = l_k + 1$
end if
$k = k + 1$

output: $\Lambda(x) = \Lambda^{(2t)}(x)$ when $k = 2t$

location polynomials and syndromes satisfy the Newton identities. For the binary BCH code, $S_{2j} = S_j^2$, where $1 \leq j \leq 2t$ and

$$\begin{cases} a\Lambda_i = 0 & \text{if a is even integer,} \\ a\Lambda_i = \Lambda_i & \text{if a is odd integer,} \end{cases} \tag{2.27}$$

where $0 \leq i \leq t$. Then the even rows in the Newton identities can be skipped if its previous odd rows are satisfied. For example, if the first row is satisfied $S_1 + \Lambda_1 = 0$ then the second row is

$$\begin{aligned} & S_2 + \Lambda_1 S_1 + 2\Lambda_2 \\ & = S_1^2 + S_1 S_1 \\ & = 0. \end{aligned} \tag{2.28}$$

Therefore, the Newton identities can be rewritten and shortened to

$$A'\Lambda = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 & 0 \\ S_2 & S_1 & 1 & \ldots & 0 & 0 \\ S_4 & S_3 & S_2 & \ldots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ S_{2t-4} & S_{2t-5} & S_{2t-6} & \ldots & S_{t-2} & S_{t-3} \\ S_{2t-2} & S_{2t-3} & S_{2t-4} & \ldots & S_t & S_{t-1} \end{bmatrix} \begin{bmatrix} \Lambda_1 \\ \Lambda_2 \\ \Lambda_3 \\ \vdots \\ \Lambda_{t-1} \\ \Lambda_t \end{bmatrix} = \begin{bmatrix} S_1 \\ S_3 \\ S_5 \\ \vdots \\ S_{2t-3} \\ S_{2t-1} \end{bmatrix}, \tag{2.29}$$

where A' is the syndrome matrix. If A' is nonsingular, the set of equations in 2.29 have a unique solution. According to the equation, the error location polynomial can be directly calculated by the syndrome without iteration, leading to low overhead. For the small error correction ability, $t = 1, 2$, the set of equations is pretty simple. Nevertheless, for the error correction ability, $t \geq 3$, the complexity increases significantly for larger t, resulting in higher power consumption and a longer critical path.

2.5.3 Chien Search

From the error location polynomial $\Lambda(x)$, the roots, $\beta_1^{-1}, \beta_2^{-1}, ..., \beta_v^{-1}$, can be determined by the Chien search [26]. This procedure is based on the cyclic property of cyclic codes and substitutes the roots $\alpha, \alpha^2, ..., \alpha^n$ one by one to verify the satisfaction of $\Lambda(x)$. Assuming the received error location polynomial $\Lambda(x)$ is

$$\Lambda(x) = \Lambda_0 + \Lambda_1 x + \ldots + \Lambda_t x^t. \tag{2.30}$$

It has been proven in [26] that after τ_k shift, if equation 2.30 satisfies the relationship

$$\Lambda(\alpha^{\tau_k}) = \Lambda_0 + \Lambda_1 \cdot \alpha^{\tau_k} + \ldots + \Lambda_t \cdot (\alpha^{\tau_k})^t = 0, \tag{2.31}$$

then one error location is $\alpha^{n-\tau_k}$, the inversion of the root. After n successive transformations, t roots will be found. The transformation process can be implemented by GF multiplication and addition according to formula 2.31.

There are two approaches to implementing this formula in hardware: one prioritizes hardware efficiency, albeit with a longer delay, while the other emphasizes timing efficiency, albeit at the cost of larger area usage. For the hardware efficient one, only t GF multipliers and $t + 1$ GF adders are needed. The successive transformation is performed by iteratively inputting each element of this GF field in an orderly manner and subsequently evaluating whether the resulting output satisfies the formula 2.31 or not. If the resulting output satisfies the formula, then store the iteration index. After n iterations, the t roots are identified and found.

Unrolling can be employed to accelerate the Chien search process. The process starts by constructing an element array $\mathbf{E}$

$$\mathbf{E} = \begin{bmatrix} \alpha & \alpha^2 & \ldots & \alpha^t \\ (\alpha)^2 & (\alpha^2)^2 & \ldots & (\alpha^t)^2 \\ (\alpha)^3 & (\alpha^2)^3 & \ldots & (\alpha^t)^3 \\ \vdots & \vdots & \ddots & \vdots \\ (\alpha)^n & (\alpha^2)^n & \ldots & (\alpha^t)^n \end{bmatrix}, \tag{2.32}$$

which encompasses all the elements required for verification. Then the verification of 2.31 becomes

$$\Lambda_0 + [\Lambda_1, \Lambda_2, \ldots, \Lambda_t] \cdot \mathbf{E} = 0. \tag{2.33}$$

And the index of the root should be stored for the final error correction. By unrolling the iteration, the Chien search finishes in one clock cycle, but with the cost of nt multipliers and $n(t+1)$ adders. The final error correction is accomplished by adding the error pattern coefficients, determined by the error locations, to the received coefficients.

3

Methods

The development of this design commences with the implementation of a bit-operation FEC simulator using MATLAB and it is adjusted by an estimation model. The simulator provides features needed by the VHDL generator, and the VHDL codes are generated according to those features. VHDL codes are transferred to an auto-synthesis system to automatically synthesize and grab the results. In the end, the synthesized register-transfer level (RTL) netlists are delivered to the BSC simulator, and the results of the binary systematic channel (BSC) simulator and evaluation system are plotted in the plot generator. The overall development flow is shown in figure 3.1.

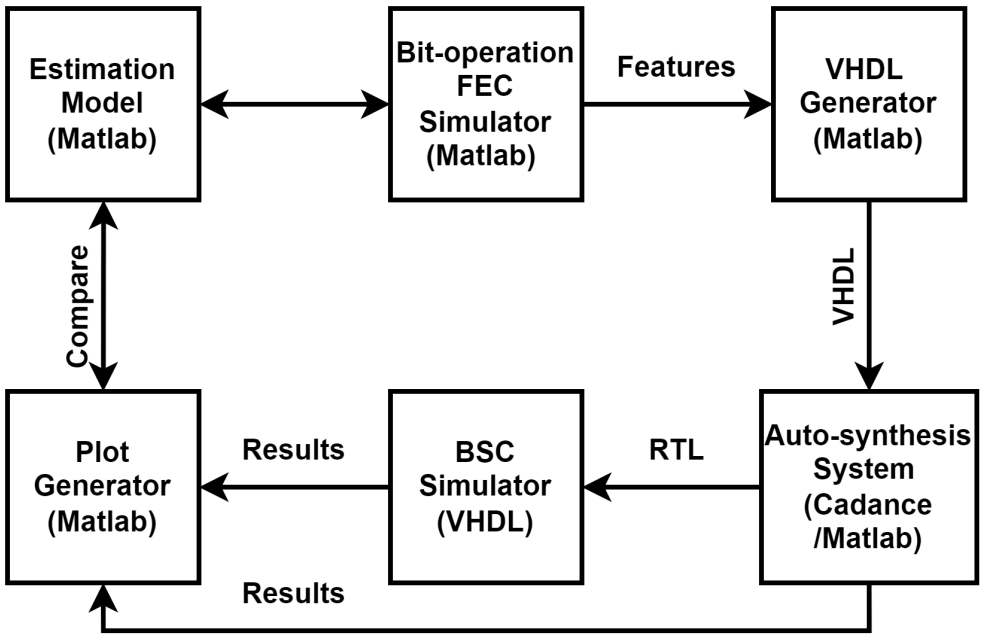


Figure 3.1: The overall thesis development flow

3.1 Bit-operation FEC Simulator

An FEC system comprising an encoder and decoder is established in MATLAB to validate the comprehension underlying those concepts. However, in order to gain a

deeper insight into the hardware perspective of GF arithmetic, all the components are implemented as bit-true operations, in other words, using logic gate operations. This approach avoids the reliance on GF embedded functions in MATLAB. At first, the model for each component is implemented in a bit-true operation style and then combined together to form a cohesive FEC system. To validate the functionality and accuracy, the results of the bit-true operation model are compared with an alternative model that utilizes embedded MATLAB functions. To facilitate the VHDL code generator, the bit-operation FEC simulator also generates the required feature array, such as **G** matrix and **H** matrix. The FEC simulator can be adjusted depending on the result of the estimation model.

3.2 Estimation Model

To compare the performance of different key equation solver algorithms and system architectures, an estimation model is implemented using MATLAB. The estimation model not only can predict the trends of timing and area usage when the error correction ability t and power m vary but also evaluates the timing and hardware complexity based on gate counts, using formulas derived from previous papers. Due to the challenge of obtaining precise performance of different logic gates during the early design phase, the estimation model can only give an estimation based on gate counts. The estimation results from the model serve as a basis for improving the FEC and making informed decisions. By leveraging the insights gained from the estimation model, the FEC can be refined aiming at low hardware and timing complexity, and a final decision can be reached regarding these goals.

3.3 VHDL Code Generator

Manually writing all RTL is tedious and cumbersome. Thanks to the regularity of the key solver algorithm and matrix-based syndrome and Chien search component, the codes can be produced by a VHDL code generator written in MATLAB. The VHDL code generator generates the VHDL codes based on the code length, error correction ability, and system architectures. Specifically, the generator connects and disconnects the input pins based on the generator matrix and parity-check matrix, and arranges the GF multipliers and adders according to the system architecture. To facilitate the functional test of different components and the whole system, testbench files are also generated by the code generator, using the test vectors produced by the bit-operation FEC simulator.

3.4 Auto-synthesis System

All the VHDL codes are synthesized to RTL netlist using a fully-depleted silicon-on-insulator (FDSOI) 22-nm CMOS design kit characterized at a typical 0.9-V corner. The VHDL codes are first compiled and evaluated for pre-synthesis functionality by Cadence Incisive. Then the codes are sent to the synthesis engine (Cadence Genus)

to generate the RTL design according to the design library. When the synthesis finishes, the synthesis engine also produces estimation reports of power consumption and area usage. Simulation is performed again to ensure the functionality of the post-synthesis netlist. To increase efficiency, an auto-synthesis system is used based on Bash. This script automates the evaluation process, saving time and labor by automatically running pre-simulation, synthesis, post-simulation, and collecting data from the reports.

3.5 BSC Simulator

A BSC simulator is proposed to evaluate the performance of the post-synthesis RTL netlists. The BSC simulator generates random bit errors according to the specific error rate. The errors are added to the encoded codewords and sent to the decoder, where the decoded codewords are compared with their encoded codewords, and uncorrected errors are counted to calculate the output BER. The power consumption is also estimated based on the value change dump (VCD) file generated by the pre-simulation.

4

Design

In this chapter, a power evaluation system is implemented to estimate the power reduction offered by FECs, and the final design is solidified with the help of the estimation model.

4.1 Power Evaluation System

In order to handle the high-temperature conditions, FECs are integrated to decrease the output BER. To prevent exacerbation of high-temperature environments, the additional power dissipation introduced by FECs should be lower than the total system power reduction offered by FECs. In order to demonstrate that, a realistic fiber-optic communication evaluation system needs to be built. The evaluation system should mimic the performance of the transmitter, such as the driver and VCSEL power, the influence of the transmission medium, and the performance of the receiver. An evaluation system is proposed in this design based on the transmitter power dissipation numbers from [29, 30, 31, 32] and the VCSEL performances from [33, 34]. The effect of the receiver is negligible due to its relatively minor impact compared to the transmitter.

Elevated temperatures can induce additional noise because of thermal fluctuations. The integration of FECs can help to protect the channel, enabling the system to operate reliably at a higher BER condition. Suitable FECs can also offer power relaxation to the whole system. By detecting and correcting the errors caused by the medium, the FEC can decrease the BER of the received signal, and hence decrease the OMA requirement at the receiver. Therefore the first thing is to find the OMA reduction when integrating FECs. An example is shown in figure 4.1, for the BER of 10^{-12}, the FEC with only $t = 3$ can result in around 3 dB OMA relaxations in a 55 Gbps throughput. The 3-dB relaxation at the transceiver will lead to the same OMA reduction at the transmitter.

Figure 4.2 reflects the relationship between throughput and OMA reduction, it can be concluded that when the system operates at higher throughput, the FEC can offer more power reduction from below 2.5 dB of 20 Gbps to above 3.5 dB of 57 Gbps. The OMA reduction finally relaxes the electrical power requirement at the transmitter, thus leading to a more power-efficient point. Figures 4.3 and 4.4 show the power reduction of $t = 3, 4$ BCH FECs at a 55-Gbps data rate. For the ultra-low (20-mW) transmitter system, the BCH FECs can offer around 10 mW power reduction, while for the 190-mW transmitter system, it can provide up to 113 mW

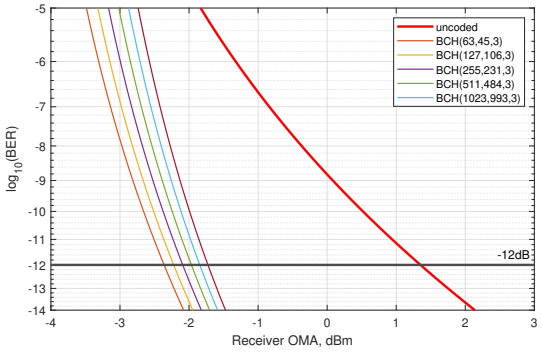


Figure 4.1: Post-FEC BER as a function of the receiver OMA at 55 Gbps

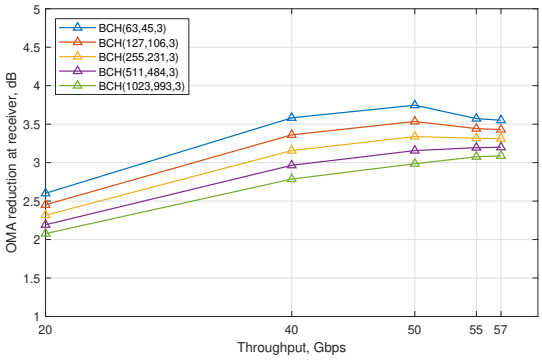


Figure 4.2: Receiver OMA reduction as a function of different throughput

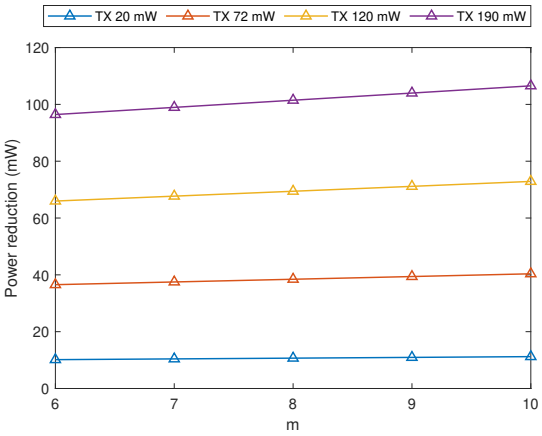


Figure 4.3: Power reduction as a function of m for $t = 3$ at 55 Gbps

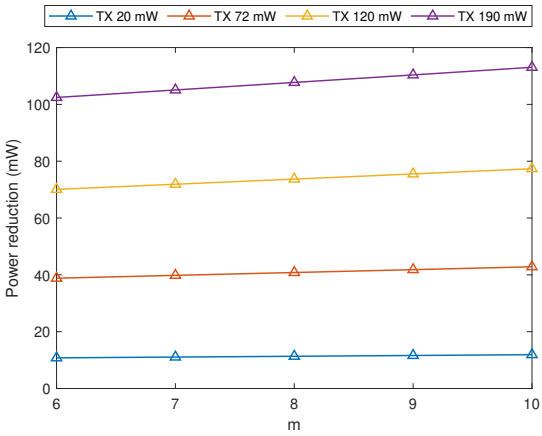


Figure 4.4: Power reduction as a function of m for $t = 4$ at 55 Gbps

power relaxation. If the additional power added by the FECs is less than those power reductions caused by FECs, then the harsh environment will not be exacerbated.

4.2 Galois Field Multiplier

The Galois field multiplier is a critical component for the decoder because the key equation solver and Chien search both need that component. There are different kinds of Galois field multipliers, such as Mastrovito multipliers [13], Karatsuba multipliers [35], and Chinese remainder theorem (CRT) multipliers [36]. The space and time complexities are generally used to evaluate the performance of GF multipliers. The space complexity is commonly represented by the total numbers of XOR gates and AND gates, and the time complexity is evaluated by the critical path, given the delay of XOR gates T_X and AND gates T_A.

From the above multipliers, Mastrovito and Karatsuba are the most commonly used [37]. Mastrovito multipliers are commonly used in implementations with a value of m that is small since the space complexity may be noncritical while using quadratic multipliers, i.e., Mastrovito, can obtain a high-speed design [37]. However, for the larger values of m, Karatsuba counterparts are more attractive in regard to space

complexity. In order to select a suitable multiplier, different implementations based on Karatsuba [35, 38, 39] and Mastrovito [40, 41] are analyzed, and a modular reduction multiplier [42] is also evaluated since a previous design [8] utilizes this method. Because the space complexity is not that critical for lower m, the analysis is performed in terms of time complexity. The comparison of mentioned multipliers is shown in figure 4.5.

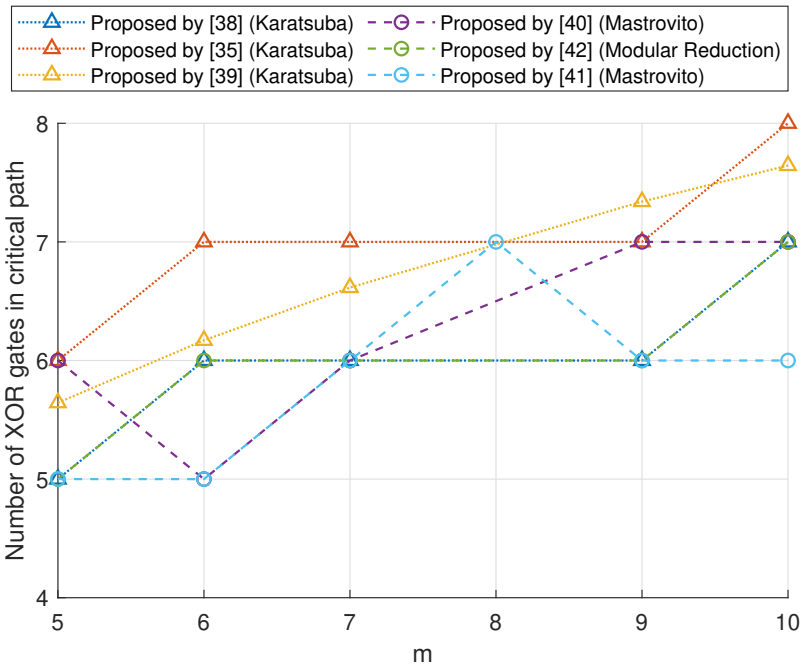


Figure 4.5: The timing comparison of different Galois field multipliers over order m

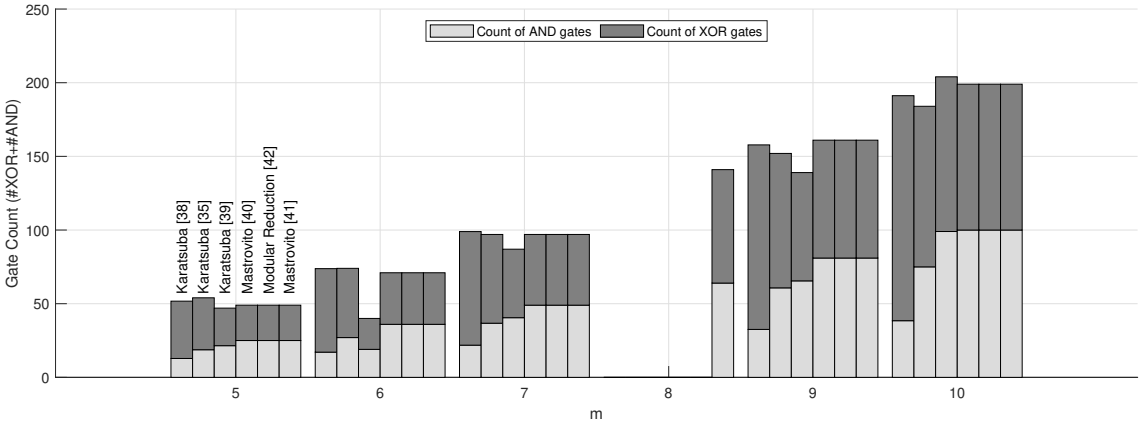


Figure 4.6: Gate count of different Galois-field multipliers over order m

The timing comparison in figure 4.5 only considers the delay of XOR gates, given the delay caused by the AND gates is constant through different methods. From figure 4.5, it can be concluded the Mastrovito multiplier in [41] is the fastest because of the elimination of useless XOR gates and optimization of XOR trees. Therefore, we implemented the Mastrovito multiplier from [41]. Figure 4.6 evaluates the hardware

complexity of different methods by counting the AND gates and XOR gates. Karatsuba multipliers from [39] offer the lowest hardware complexity but pay in terms of timing. The Mastrovito multipliers in [41] also provide comparable lower hardware complexities. It should be noted that for $m = 8$, there is an absence of implementations based on the pentanomial for Karatsuba multipliers [35, 38, 39, 40, 42], thus they are not included in figures 4.5 and 4.6.

The Mastrovito multiplier performs matrix multiplication. It optimizes the multiplication $C = A \cdot B$ to $\underline{c} = \mathbf{M} \cdot \underline{b}$, where $\underline{c}$ and $\underline{b}$ are the binary-coefficient vectors of A and B, and $\mathbf{M}$ is a multiplication matrix generated by some elements of $\underline{a}$. Then the Mastrovito multiplier can be sorted into two steps.

The first steps is to compute the matrix $\mathbf{M}$, assuming an irreducible polynomial $p(x)$ of GF(2^m) is:

$$p(x) = x^m + p_{m-1}x^{m-1} + \ldots + p_1 x + 1, \tag{4.1}$$

and the equation 4.1 has the roots $\underline{r} = [1, \alpha, \alpha^2, ..., \alpha^{m-1}]$. Given the polynomial representations of elements A and B:

$$A = a_0 + a_1\alpha + \ldots + a_{m-1}\alpha^{m-1} = \underline{r} \cdot \underline{a}^T, \tag{4.2}$$
$$B = b_0 + b_1\alpha + \ldots + b_{m-1}\alpha^{m-1} = \underline{r} \cdot \underline{b}^T, \tag{4.3}$$

then the calculation of C can be rewritten to:

$$\begin{aligned} C &= b_0 A + b_1 \cdot A \cdot \alpha + \ldots + b_{m-1} \cdot A \cdot \alpha^{m-1} \\ &= [A, A \cdot \alpha, \ldots, A \cdot \alpha^{m-1}] \cdot \underline{b}^T, \end{aligned} \tag{4.4}$$

where $\underline{b} = [b_0, b_1, ..., b_{m-1}]$. Introducing the $M_i = [M_{0,i}, M_{1,i}, \ldots, M_{m-1,i}]$ and reformulating equation 4.4 to:

$$C = \underline{r} \cdot [M_0^T, M_1^T, \ldots, M_{m-1}^T] \cdot \underline{b}^T, \tag{4.5}$$

gives the relationship:

$$A \cdot \alpha^i = \underline{r} \cdot M_i^T. \tag{4.6}$$

From equation 4.6, the first column of matrix M is equal to $\underline{a}$, and the other columns can be computed by this first column:

$$\underline{r} \cdot M_i^T = \underline{r} \cdot \alpha \cdot M_{i-1}^T. \tag{4.7}$$

Then the computation of the following column of matrix $\mathbf{M}$ becomes [41]:

$$\begin{aligned} &M_{0,i} = M_{m-1,i-1} \quad i = 1, \ldots, m-1 \\ &M_{j,i} = M_{j-1,i-1} + M_{m-1,i-1} \cdot p_j \\ &i, j \leq m-1 \text{ positive integer.} \end{aligned} \tag{4.8}$$

Since p_j is constant for each design, then the calculation of matrix $\mathbf{M}$ is the actual wire connection and XOR operations.

The second step is comparably simple:

$$c_j = M_{j,0} \cdot b_0 + M_{j,1} \cdot b_1 + \ldots + M_{j,m-1} \cdot b_{m-1}, \tag{4.9}$$

which can be implemented by m AND gates and $m-1$ XOR gates for binary multiplications and additions. A Mastrovito multiplier for GF(2^6) with $p(x) = x^6 + x^3 + 1$ is shown in figure 4.7.

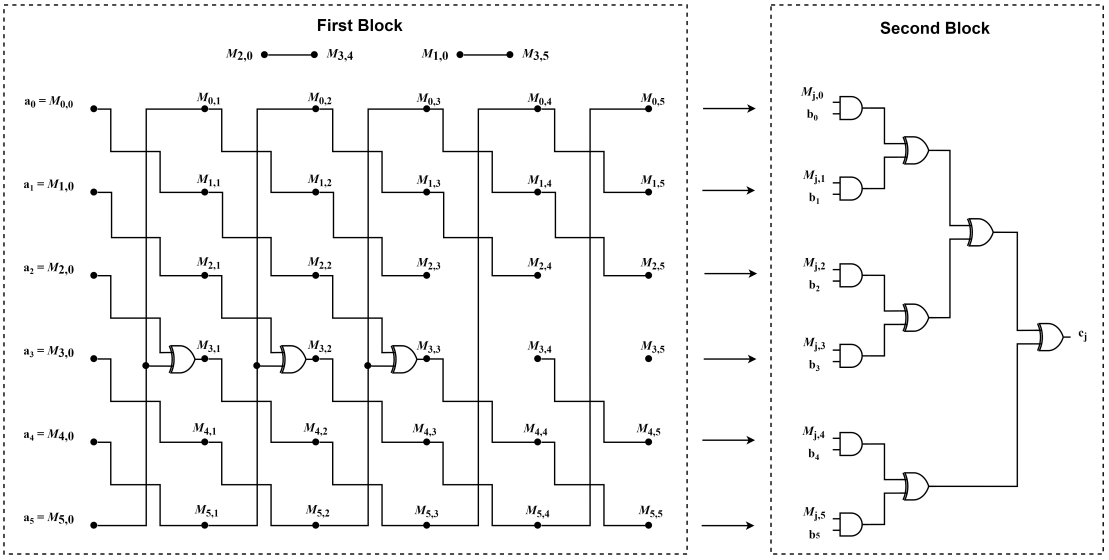


Figure 4.7: The Mastrovito multiplier structure for $GF(2^6)$ with $p(x) = x^6 + x^3 + 1$

The first block is used to calculate the matrix $\mathbf{M}$ and is composed of XOR gates and wire connections. The XOR gate is added depending on the value of p_j; when $p_j = 1$ then an XOR gate should be employed to calculate the element of the next column, otherwise, only a wire connection is needed. Hardware sharing can be exploited by tracking back the representation of the current result by the first column elements, for $M_{3,4}$ in the above example can be represented by $M_{2,0}$ XOR $M_{5,0}$ XOR $M_{5,0}$ that can be simplified to $M_{2,0}$ since XORing the same value will become zero and any value XORed with the zero is equal to itself. The second block is comprised of m AND gates and $m-1$ XOR gates and is used to compute $\underline{c} = \mathbf{M} \cdot \underline{b}$. Figure 4.7 only shows one computation unit of c_j, whereas, for the whole block, it contains m computation units.

4.3 Berlekamp-Massey Decoder

In this design, we implement a decoder based on a look-ahead reformulated inversionless BM (riBM) algorithm with $t = 3$ or 4.

4.3.1 Syndrome Calculation

The hardware structure of syndrome calculation is quite simple. From equation 2.13 and matrix 2.10, the syndrome calculation is composed of bit addition and multiplication. As mentioned before, the binary bit addition can be implemented by the XOR gate, and multiplication can be implemented by the AND gate. And if one element of AND operation is constant, the AND gate can be replaced by a wire connection or disconnection. Therefore, the syndrome calculation can be simplified to XOR trees generated by the parity-check matrix **H**.

In matrix 2.10, each GF element can be rewritten as an m binary vector. Hence the hardware complexity of the syndrome calculation can be represented by $2 \cdot tm$ XOR trees, and each tree contains at most $n-1$ XOR gates and average $\frac{n}{2}-1$ XOR gates, assuming each row is filled by half ones. Then the maximum and average total XOR gates become $2tm(n-1)$ and $2tm(\frac{n}{2}-1)$ XOR gates, respectively, assuming symmetric trees for lower logic depth (see figure 4.8).

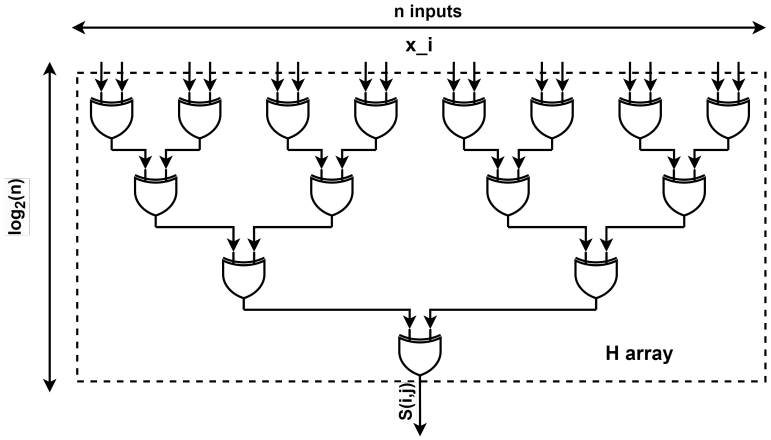


Figure 4.8: One syndrome-bit calculation structure assuming the worst case n inputs

From figure 4.8, the worst-case complexity of syndrome calculation is

$$T_{SD} = log_2(n) \cdot T_{xor}, \tag{4.10}$$
$$A_{SD} = 2tm(n-1) \cdot A_{xor}, \tag{4.11}$$

and the average complexity when the **H** is filling with half ones is

$$T_{SD} = log_2(\frac{n}{2}) \cdot T_{xor}, \tag{4.12}$$
$$A_{SD} = 2tm(\frac{n}{2}-1) \cdot A_{xor}. \tag{4.13}$$

4.3.2 Key Equation Solver

Since Berlekamp and Massey introduced the primitive BM algorithm, numerous enhanced algorithms have been developed building its foundation in efforts to decrease

the hardware complexity and improve the timing performance. A literature review is presented below regarding the hardware complexity and timing complexity at a component level. Specifically, the focus is evaluating the effects of the GF multipliers, GF adders, and multiplexers (MUXs). Regarding the hardware complexity, area usage, the effect of the register is also included. The timing complexities and hardware complexities of the GF multipliers are from section 4.2 with a dynamic range. For two m-bit inputs GF adders, the critical path is equal to one XOR gate, and the area usage is m XOR gates. The two m-bits inputs MUX has the same delay as the two 1-bit inputs MUX and has an area usage of m two 1-bit inputs MUX. The registers are modeled in the same way as the MUX. The complexities of different components are summarized in table 4.1 and 4.2, where A and T represent the area and delay of the basic logic gate, i.e. A_{xor} is the area usage of one XOR gate and T_{xor} is the delay of one XOR gate.

Hardware Complexity		
Multiplier	Trinomial $n = 1$	$(m^2 - 1) \cdot A_{xor} + m^2 \cdot A_{and}$
	Trinomial $2 \leq n \leq m/2$	$(m^2 - 1) \cdot A_{xor} + m^2 \cdot A_{and}$
	Pentanomial	$(m(m-1) + 3(m-1)) \cdot A_{xor} + m^2 \cdot A_{and}$
Adder	$m \cdot A_{xor}$	
MUX	$m \cdot A_{mux}$	
REG	$m \cdot A_{reg}$	

Table 4.1: The hardware complexity of each component

Timing Complexity		
Multiplier	Trinomial $n = 1$	$\lceil \log_2(2m-1) \rceil \cdot T_{xor} + T_{and}$
	Trinomial $2 \leq n \leq m/2$	$\lceil \log_2(2m+2n-3) \rceil \cdot T_{xor} + T_{and}$
	Pentanomial	$\lceil \log_2(4m+12n-21) \rceil \cdot T_{xor} + T_{and}$
Adder	T_{xor}	
MUX	T_{mux}	
REG	T_{reg}	

Table 4.2: The timing complexity of each component

The primitive BM algorithm [20, 21] has a longer critical path that depends on the error correction ability. As can see from algorithm 1, the delay in computing d_k is $T_{mult} + \lceil \log_2(l_k + 1) \rceil \cdot T_{add}$, where T_{mult} and T_{add} are the delays of two m-bits inputs GF multiplier and adder, respectively. Then, considering the worst case when the discrepancy is nonzero, the adjustment of $\Lambda^{k+1}(x)$ adds t GF multiplications and $t - 1$ additions to the circuit and increases the maximum delay to

$$T_{BM} = 2 \cdot T_{mult} + (\lceil \log_2(l_k + 1) \rceil + \lceil \log_2(t) \rceil) \cdot T_{add} + T_{MUX}, \tag{4.14}$$

where T_{MUX} is the delay of two m-bits input MUX. This delay only contains the influence of GF multiplier and adder, neglecting the effect of other circuitry such as LUT for inversion. Including these effects, the critical path will be larger. Regarding the hardware complexity, the calculation of d_k needs l_k GF multipliers and GF

adders, resulting in $l_k \cdot (A_{mult} + A_{add})$, where the A_{mult} and A_{add} are the usage of one GF multiplier and one adder. The computing of $\Lambda^{k+1}(x)$ require $t+1$ GF multipliers, $t+1$ adders, t MUXs and $t+1$ registers, leading to a total hardware complexity of

$$A_{BM} = (l_k + t + 1) \cdot A_{mult} + (l_k + t + 1) \cdot A_{add} + t \cdot A_{MUX} + (t + 1) \cdot A_{REG}. \tag{4.15}$$

For binary BCH codes, the even iterations can be eliminated because the result discrepancies are always zeros [10]. Hence the iteration count decreases to t, but the timing and hardware complexities are same

$$T_{BM} = 2 \cdot T_{mult} + (\lceil \log_2(l_k + 1) \rceil + \lceil \log_2(t) \rceil) \cdot T_{add}, \tag{4.16}$$

$$A_{BM} = (l_k + t + 1) \cdot A_{mult} + (l_k + t + 1) \cdot A_{add} + t \cdot A_{MUX} + (t + 1) \cdot A_{REG}. \tag{4.17}$$

In order to replace the hardware and timing consuming GF inversion in primitive BM algorithm, inversionless BMs (iBM) were proposed and implemented in [43, 44]. In those designs, the arduous task of division or LUT used to implement the inversion is replaced by multiplication, leading to a fast circuit. Based on an implementation from [27], the complexities of iBM change to

$$T_{iBM} = 2 \cdot T_{mult} + (1 + \lceil \log_2(t + 1) \rceil) \cdot T_{add}, \tag{4.18}$$

$$A_{iBM} = (5t + 3) \cdot A_{mult} + (3t + 1) \cdot A_{add} + (2t + 1) \cdot A_{MUX} + (4t + 2) \cdot A_{REG}. \tag{4.19}$$

Obviously, the increased hardware complexity pays for the decrease in timing complexity. To further reduce the critical path, reformulated inversionless algorithms (riBM) were introduced and implemented in [27]. Those algorithms compute the next discrepancy $\Delta^{k+1}(x)$ and the next error location polynomial $\Lambda^{k+1}(x)$ at the same time, reducing the longest delay to

$$T_{riBM} = T_{mult} + T_{add}, \tag{4.20}$$

by incurring the cost of only $t - 1$ additional multipliers,

$$A_{riBM} = (6t + 2) \cdot A_{mult} + (3t + 1) \cdot A_{add} + (3t + 1) \cdot A_{MUX} + (6t + 2) \cdot A_{REG}. \tag{4.21}$$

The RiBM algorithm is just a rearranged version of riBM, aiming to make the processing engine identical. Therefore, the complexity of RiBM is the same as the riBM. Subsequently, the truncated iBM (TiBM) [45] and compensated simplified RiBM (CS-RiBM) [46] were proposed to replace the unnecessary computations for evaluation polynomial in RiBM with simple compensation for the RS(255,239) code with 16 iteration times. For this design using BCH code, the iteration time decreases from $2t$ to t, and the target error correction abilities are $t = 3, 4$. Therefore, the iteration time is not that large so TiBM and CS-RiBM algorithm are not considered. In order to find a suitable algorithm, the hardware and timing complexities of BM, iBM, and riBM algorithms for the binary BCH code with $t = 3$ and 4 are compared in figures 4.9 to 4.12. The performance of both CS-RiBM and TiBM is also illustrated for the purpose of comparison.

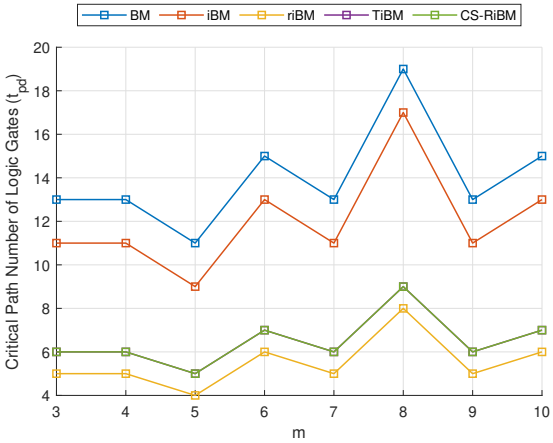


Figure 4.9: Number of logic gates in the critical path for $t = 3$

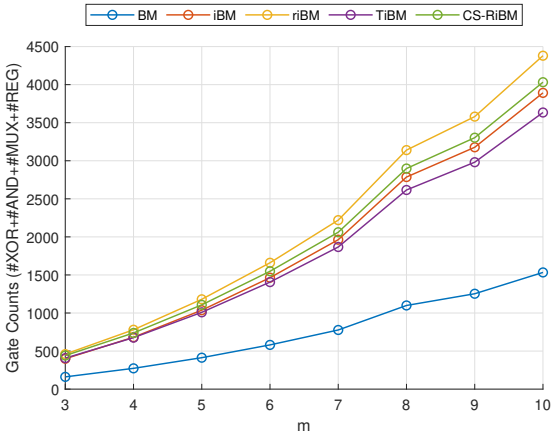


Figure 4.10: Gate count, including the XOR, AND, MUX, and REG gates, for $t = 3$

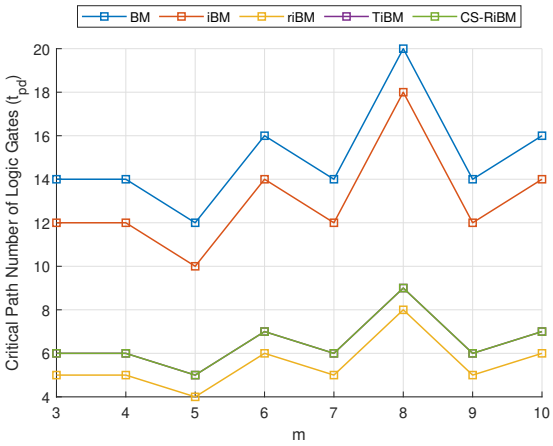


Figure 4.11: Number of logic gates in the critical path for $t = 4$

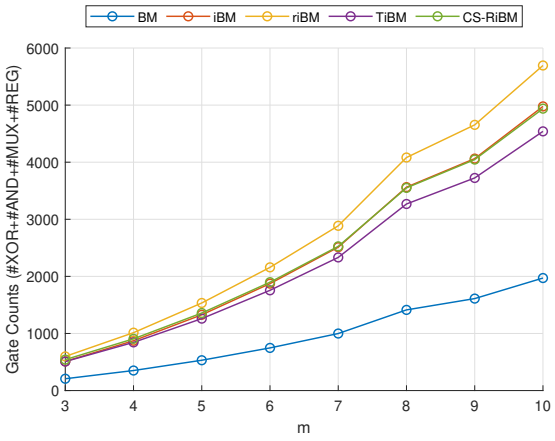


Figure 4.12: Gate count, including the XOR, AND, MUX, and REG gates, for $t = 4$

The timing of TiBM and CS-RiBM is overlapping in figures 4.9 and 4.11 since the timing of both are the same. From figures 4.9 to 4.12, riBM exhibits superior timing performance across different m, albeit at the expense of increased area usage. The performance of the key equation solver varies in accordance with the structure of the multiplier and has the longest path at $m = 8$, since no irreducible trinomial exits for $m = 8$ and a pentanomial is used. Consequently, it is advisable to avoid codes with $m = 8$ from a hardware perspective.

4.3.3 riBM Structure

This design utilizes the look-ahead reformulated inversionless Berlekamp-Massey algorithm (riBM) [27, 47] for the key equation solver. This algorithm calculates the next discrepancy and error location polynomial at the same time, eliminating the feedback loop at the primitive BM algorithm. Since the discrepancy used to control the MUX does not necessitate computation within the current cycle, the critical path

is correspondingly shortened. In addition, the hardware-expensive inversions in BM are replaced by multiplication, resulting in simple circuitry. The riBM algorithm is provided in the below pseudocode.

Algorithm 2: The look-ahead riBM Algorithm for binary BCH code

input: S_i $(1 \leq i \leq 2t)$

initialization: $\Lambda^{(-1)}(x) = 1,\ B^{(-1)}(x) = x^{-1},$
$k^{(-1)} = -1,\ \gamma^{(-1)} = 1,$
$\Delta^{(-1)}(x) = S_1 x + S_2 x^2 + \ldots + S_{2t} x^{(2t)},$
$\Theta^{(-1)}(x) = S_1 + S_2 x + \ldots + S_{2t} x^{(2t-1)}.$

begin: for $r = -1, 1, 3, ..., 2t-3$

$\Lambda^{(r+2)}(x) = \gamma^{(r)}\Lambda^{(r)}(x) + \Delta_1^{(r)} x^2 B^{(r)}(x)$

$\Delta^{(r+2)}(x) = \gamma^{(r)}\Delta^{(r)}(x)/x^2 + \Delta_1^{(r)}\Theta^{(r)}(x)$

if $\Delta_1^{(r)} \neq 0$ *and* $k^{(r)} \geq -1$ then

$B^{(r+2)}(x) = \Lambda^{(r)}(x)$

$\Theta^{(r+2)}(x) = \Delta^{(r)}(x)/x^2$

$\gamma^{(r+2)} = \Delta_1^{(r)}$

$k^{(r+2)} = -k^{(r)} - 2$

else

$B^{(r+2)}(x) = x^2 B^{(r)}(x)$

$\Theta^{(r+2)}(x) = \Theta^{(r)}(x)$

$\gamma^{(r+2)} = \gamma^{(r)}$

$k^{(r+2)} = k^{(r)} + 2$

end if

output: $\Lambda(x) = \Lambda^{(2t-1)}(x)$ when $r = 2t-1$

The input to the key equation solver is $2t$ syndrome from the syndrome calculation components. In the beginning, the syndrome is initialized to two register vectors, $\Delta^{(r)}(x)$ and $\Theta^{(-1)}(x)$. Then the error location polynomial $\Lambda^{(r+2)}(x)$ and the error evaluation polynomial $\Delta^{(r+2)}(x)$ are calculated according to the initialization values. The update of $\Lambda^{(r+2)}(x)$ requires $2t+2$ GF multipliers and $t+1$ adders, and the update of $\Delta^{(r+2)}$ needs $4t$ GF multiplier and $2t$ adders. Hence the total numbers of required multipliers and adders are $6t+2$ and $3t+1$, respectively. According to the current discrepancy $\Delta_1^{(r)}$ and current $k^{(r)}$, the MUXs are implemented to control the updating of the registers of $B^{(r+2)}$, $\Theta^{(r+2)}(x)$, $\gamma^{(r+2)}$, and $k^{(r+2)}$. The multiplication and division of x^2 can be achieved by signal indexing. An additional control counter is implemented to govern the operation of the above combinational logic.

Like the design in [27, 47], this implementation does follow an engine-by-engine structure. The process engine zero (PE0) for computing $\Delta^{(r+2)}(x)$ and $\Lambda^{(r+2)}(x)$ is almost the same except for the initial data. One computation unit of $\Lambda^{(r+2)}(x)$ is shown in figure 4.13. To maintain a clean figure, the control unit is omitted. The computation of one $\Lambda^{(r+2)}(x)$ needs two multipliers and one adder. At the first iteration, the registers load the initial data connecting by MUXs, and the multiplier and adder compute the $\Lambda^{(r+2)}(x)$, simultaneously. Meanwhile, the MUXs switch to the feedback loop, transmitting $\Lambda^{(r+2)}(x)$ for the subsequent computation.

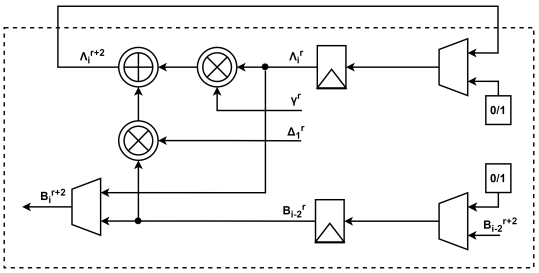


Figure 4.13: Process Engine (PE0) for $\Lambda^{(r+2)}(x)$ and $B^{(r+2)}$

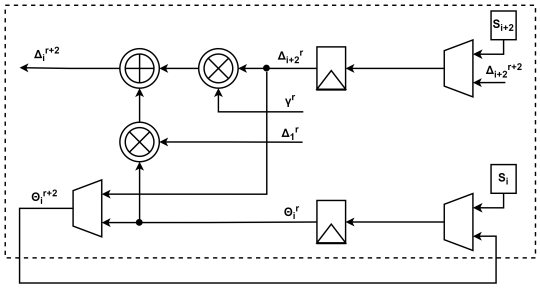


Figure 4.14: Process Engine (PE1) for $\Delta^{(r+2)}(x)$ and $\Theta^{(r+2)}(x)$

The value of $B^{(r+2)}$ is updated based on the value of $\Delta_1^{(r)}$ and $k^{(r)}$ and is then fed back to prepare for the next computation. Because the maximum length of $\Lambda^{(r+2)}$ is $t+1$, $t+1$ PE0 processors are needed.

The theoretical critical path of one PE0 is

$$T_{PE0} = T_{mult} + T_{add} + T_{MUX}, \tag{4.22}$$

which is one T_{MUX} higher than equation 4.20 due to the inclusion of input assignment MUXs. The process engine one (PE1) used to compute $\Delta^{(r+2)}(x)$ and $\Theta^{(r+2)}(x)$ is similar to the PE0, with the same critical path

$$T_{PE1} = T_{mult} + T_{add} + T_{MUX}. \tag{4.23}$$

However, since the maximum length of $\Delta^{(r+2)}(x)$ and $\Theta^{(r+2)}(x)$ is $2t$, $2t$ PE1 processors are needed. Finally, $t+1$ PE0 and $2t$ PE1 are connected according to the index (see figure 4.15). The control signals of the MUX have been excluded for the clarity of the figure.

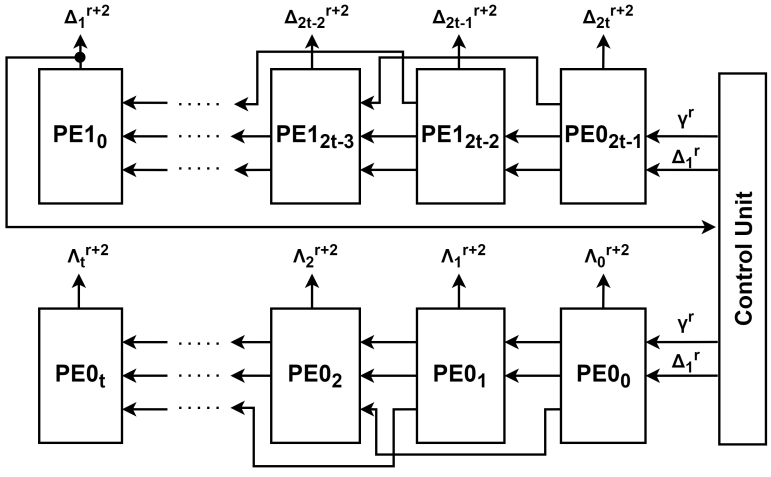


Figure 4.15: The connection of PE0 and PE1 processor based on the index

Thus the critical path is

$$T_{riBM\_imp} = T_{mult} + T_{add} + T_{MUX}. \tag{4.24}$$

Considering the input assignment MUXs, the hardware complexity is

$$A_{riBM\_imp} = (6t+2)\cdot A_{mult} + (3t+1)\cdot A_{add} + (9t+3)\cdot A_{MUX} + (6t+2)\cdot A_{REG}. \quad (4.25)$$

4.3.4 Parallel Chien Search

Due to the low latency requirement, the unrolled parallel Chien search structure is used in this design. First, the n-by-t element array **E** is generated by the FEC simulator based on the error correction ability and block length. In contrast to the multiplier utilized in the riBM processor, the multiplier employed in the Chien search performs multiplication with a constant element. This means that the multiplier in Chien search can be simplified to only XOR gates. The multiplier with a constant value has the critical path and area usage of

$$T_{const} = \lceil \log_2(m) \rceil \cdot T_{xor}, \quad (4.26)$$

$$A_{const} = (\frac{1}{2}m^2 - m) \cdot A_{xor}, \quad (4.27)$$

from [27, 48]. For the computation of each row, t constant multipliers and t adders are employed. The bit of error correction vector is initialized to zero at the beginning and is flipped to logic one according to the error location. The first row of a Chien search component is shown in figure 4.16, where x represents the received codeword, c is the correction pattern, and dc is the decoded codeword.

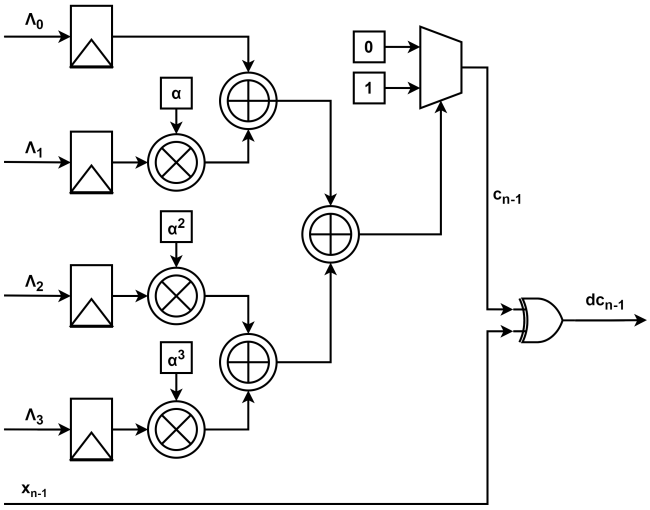


Figure 4.16: One example structure for the first row of Chien search component with $t = 3$

From figure 4.16, one row of Chien search computation and error correction involves t constant multipliers, t adders, one MUX, one XOR gate, and four m-bits registers. Then the complexity of the Chien search component is

$$T_{CS} = T_{const} + \lceil log_2(t) \rceil \cdot T_{add} + T_{mux} + T_{xor}, \tag{4.28}$$
$$A_{CS} = nt \cdot A_{const} + nt \cdot A_{add} + n \cdot A_{mux} + n \cdot A_{xor}, \tag{4.29}$$

4.4 System Architecture

To attain a high throughput and lower power design, it is crucial to establish a well-designed system architecture that prioritizes lower latency and lower power consumption, especially dynamic power. This section provides an analysis of three different system architectures, direct iteration (DI), parallel iteration (PI), and unrolled pipeline (UP). Only two architectures, PI and UP, are implemented and synthesized with clock-gating technology to decrease the dynamic power. First, to comprehensively understand the timing and hardware complexity of those three architectures, the complexities are thoroughly evaluated based on theoretical analysis down to the logic gate level, using the formula from tables 4.1 and 4.2 and the complexities of different components. It should be acknowledged that the evaluation presented below utilizes the average complexity (see equations (4.12) and (4.13)) of syndrome calculation.

4.4.1 Direct Iteration

The most hardware-efficient architecture involves utilizing a single riBM core iteratively processing the data. However, this architecture will result in a lower throughput due to requiring t iterations for one error location polynomial calculation in riBM component.

Figure 4.17: The direct iteration (DI) structure

Registers are employed at the input and output and are inserted between each component to allow a higher clock speed (see figure 4.17). This system can only process one received codeword per t clock cycles, significantly decreasing the throughput. To achieve a 56 Gbits/sec throughput, this architecture needs a clock frequency f_{clk} of

$$f_{clk} = t \cdot 56/k \; GHz \tag{4.30}$$

where k is the message length. The hardware complexities and timing complexities of different components are shown in figures 4.18 to 4.21.

The starting value of m is set at five due to the absence of a $t = 4$ BCH code below that threshold. From figures 4.18 and 4.20, the critical path is in the Chien search

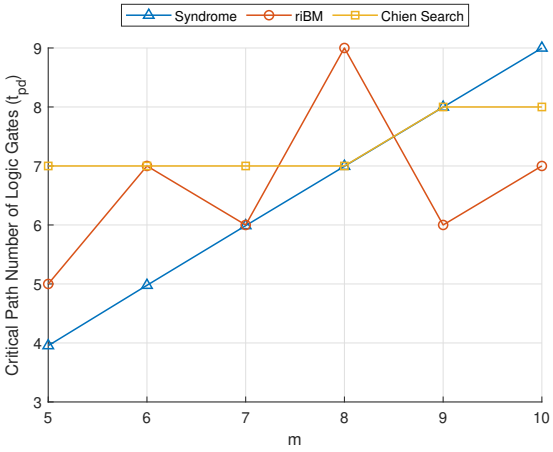


Figure 4.18: Number of logic gates in the critical path for a DI structure with $t = 3$

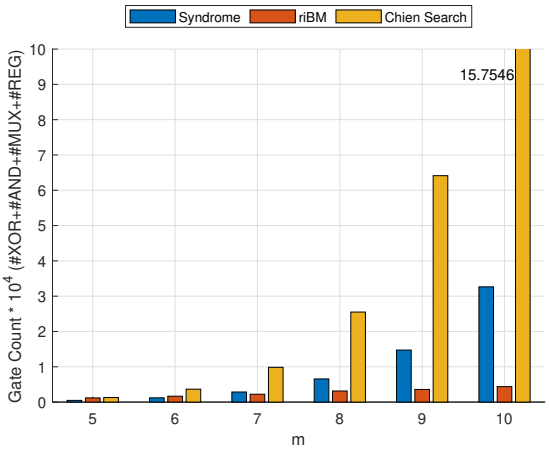


Figure 4.19: Gate count, including the XOR, AND, MUX, and REG gates, for a DI structure with $t = 3$

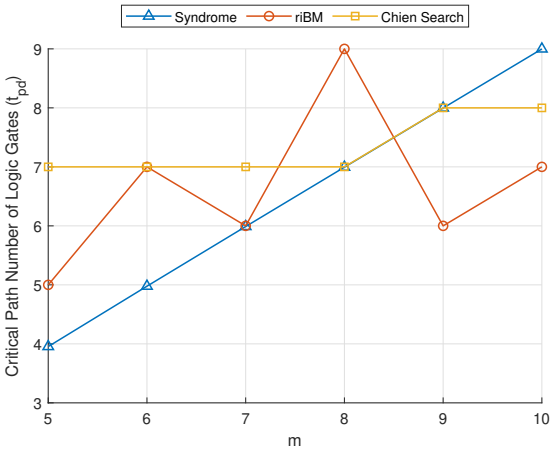


Figure 4.20: Number of logic gates in the critical path for a DI structure with $t = 4$

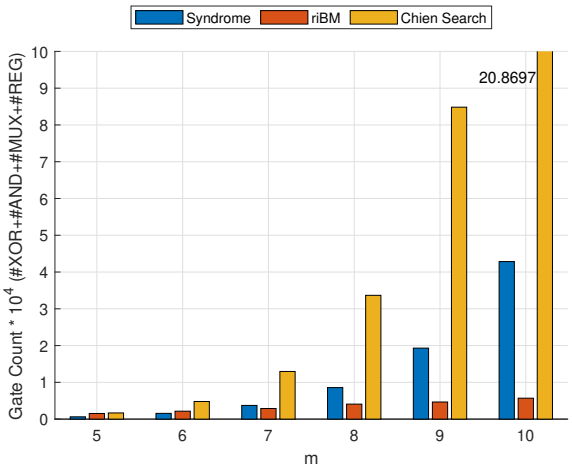


Figure 4.21: Gate count, including the XOR, AND, MUX, and REG gates, for a DI structure with $t = 4$

component when m is lower than eight, however, at $m = 8$, the critical path is in riBM core due to the complex pentanomial Mastrovito multiplier. In terms of hardware complexity, the Chien search is the most resource-intensive component, and its logic use tends to escalate exponentially. For $m = 5, 6$, the riBM component exhibits a larger area utilization compared to the syndrome component. However, as m surpasses six the area usage of the syndrome component is larger than riBM's. Since the total area usage exhibits an exponential upward trend with increasing m, a small m should be used to achieve a lower area usage.

Small m means short block length and message length, resulting in a high f_{clk} needed to maintain the high throughput requirement. For example, implementing a BCH(31,16,3) code with DI structure would need a 10.5 GHz clock to achieve a 56 Gbits/sec throughput based on equation 4.30. The circuit with that high-speed clock will consume a significant amount of power according to the dynamic power

formula

$$P_{dyn} = \alpha C_{chip} V_{DD}^2 f_{clk} \tag{4.31}$$

where α is the activity factor, C_{chip} is the switched capacitance and V_{DD} is the supply voltage. Therefore, despite the DI structure is hardware efficient, it suffers the problem of high latency and high power consumption.

4.4.2 Parallel Iteration

The iterative structure of DI implementation limits the achievable throughput. To reduce the latency, incorporating more riBM cores in parallel into the circuit can be an effective approach. Therefore, a parallel iteration (PI) architecture is proposed with three riBM cores in parallel (see figure 4.22). To enable core selection and ensure accurate output from the respective core, an additional de-multiplexer (DEMUX) and multiplexer (MUX) are integrated into the circuit. The state of DEMUX and MUX is controlled by the state controller, which stores the current positions and busy flags of those two selectors. In order to synchronize the arrival of the received codeword and its corresponding error location polynomial, a FIFO (First-In-First-Out) buffer is added. The received codeword is delayed by $t + 1$ cycles, based on the error correction ability of the design.

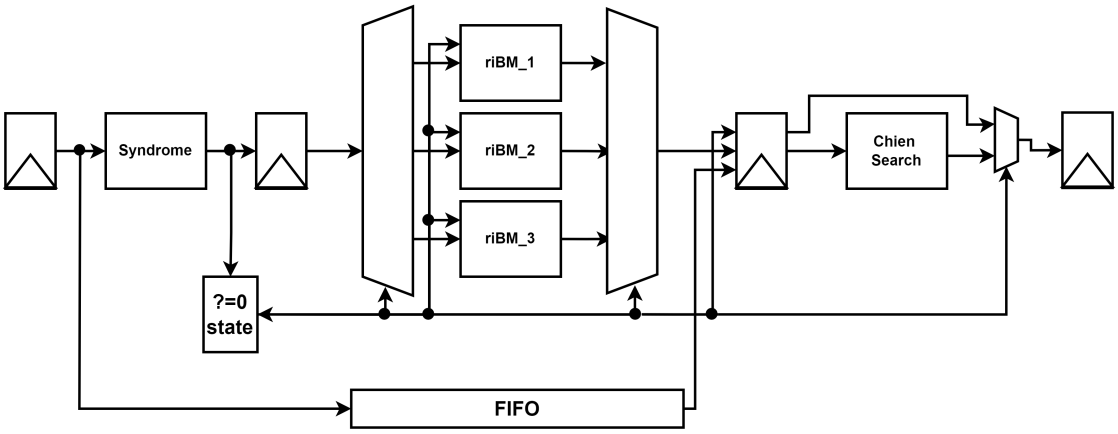


Figure 4.22: The structure of parallel iteration (PI) system with $t = 3$

By parallelizing three riBM cores, the delay caused by the core is reduced to only one clock cycle. Therefore the required clock frequency for 56 Gbits/sec throughout reduces to

$$f_{clk} = 56/k \; GHz. \tag{4.32}$$

Unfortunately, the integration of more riBM cores in this architecture leads to a more than threefold increment in area usage compared with DI because of the required control logic. The state controller controls the dispatch of each syndrome. In addition, clock gating is employed to reduce the power consumption. Enable signals are added to the registers for riBM cores and registers for Chien search. When there are no errors in the received codeword, the syndrome is zero, and the state controller disables the riBM cores and Chien search core according to the busy

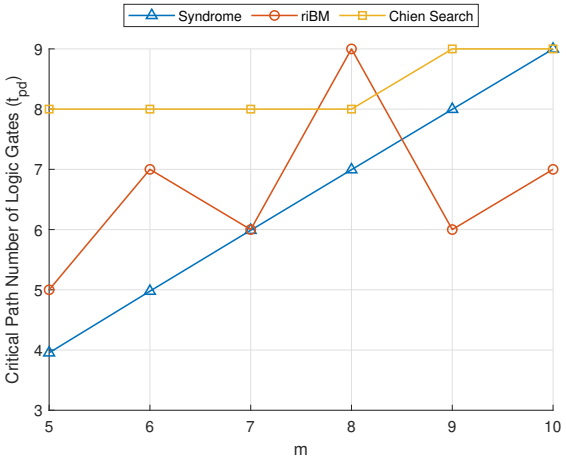


Figure 4.23: Number of logic gates in the critical path for a PI structure with $t = 3$

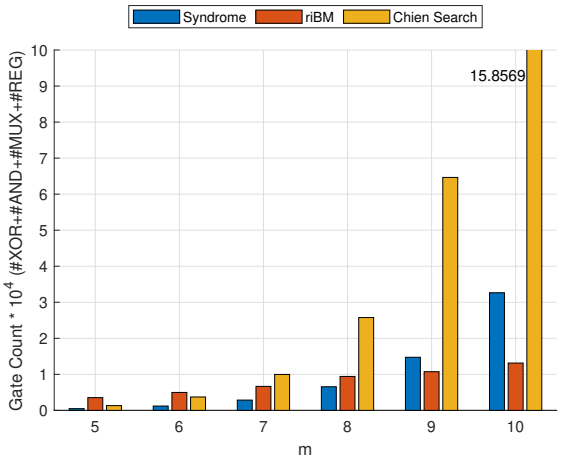


Figure 4.24: Gate count for a PI structure with $t = 3$

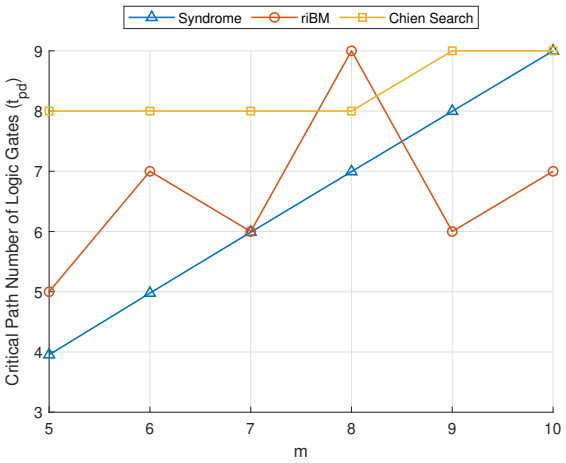


Figure 4.25: Number of logic gates in the critical path for a PI structure with $t = 4$

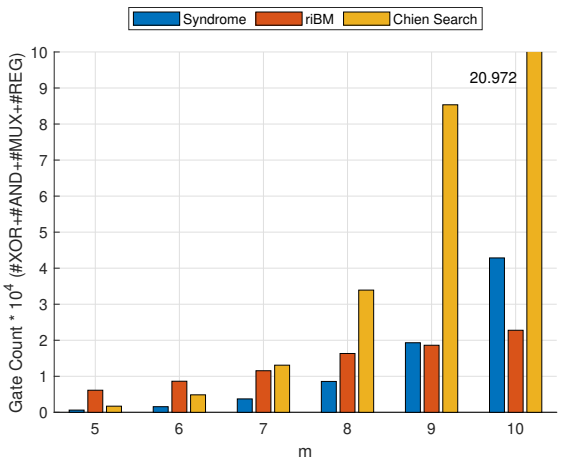


Figure 4.26: Gate count for a PI structure with $t = 4$

state of the riBM cores. If the syndrome result is not zero, which means errors are presented in the received codeword, the state controller activates one non-busy core and also enables the Chien search core to find the error locations and correct the errors. A MUX is added between the Chien search core and the final output, aiming at skipping the Chien search core when it is disabled. Hence the critical path of the modified Chien search core increases by one MUX. The complexity is shown in figures 4.23 to 4.25 excluding the FIFO and state controller.

The Chien search component has the longest critical path except $m = 8$ due to the one extra MUX. For $m = 8$, the riBM core dominates because of the pentanomial Mastrovito multiplier. Regarding hardware complexity, the riBM cores exhibit the largest area usage when $m = 5, 6$, however, for other values of m the Chien search demonstrates the largest area. A larger value of m results in larger area usage and may result in higher power consumption. Therefore, $m = 6$ BCH decoders are implemented in this design.

4.4.3 Unrolled Pipeline

Even though PI reduces the latency caused by the riBM core, integrating DEMUX and MUX, and parallelizing the cores increase the hardware complexity and may lead to higher power consumption which violates the lower power goal. To constrain power consumption, a power and hardware-efficient unrolled pipeline (UP) architecture is proposed. Unrolling is possible for the main iteration in algorithm 2 since it follows a regular pattern and does not involve any complex dependencies. Unrolling can also help to reduce hardware complexity compared with PI architecture. From the Algorithm 2, since the $\Lambda^{(-1)}(x) = 1, \gamma^{(-1)} = 1$ and $B^{(-1)}(x) = x^{-1}$, the first iteration can be simplified to algorithm 3.

Algorithm 3: The simplified first iteration of look-ahead riBM Algorithm

input: S_i $(1 \leq i \leq 2t)$

initialization: $\Lambda^{(-1)}(x) = 1,\ B^{(-1)}(x) = x^{-1},$
$k^{(-1)} = -1,\ \gamma^{(-1)} = 1,$
$\Delta^{(-1)}(x) = S_1 x + S_2 x^2 + \ldots + S_{2t} x^{(2t)},$
$\Theta^{(-1)}(x) = S_1 + S_2 x + \ldots + S_{2t} x^{(2t-1)}.$

begin: for $r = -1$
$\Lambda^{(r+2)}(x) = 1 + \Delta_1^{(r)} x$
$\Delta^{(r+2)}(x) = \Delta^{(r)}(x)/x^2 + \Delta_1^{(r)} \Theta^{(r)}(x)$
if $\Delta_1^{(r)} \neq 0$ *and* $k^{(r)} \geq -1$ then
$B^{(r+2)}(x) = 1$
$\Theta^{(r+2)}(x) = \Delta^{(r)}(x)/x^2$
$\gamma^{(r+2)} = \Delta_1^{(r)}$
$k^{(r+2)} = -1$
else
$B^{(r+2)}(x) = x$
$\Theta^{(r+2)}(x) = \Theta^{(r)}(x)$
$\gamma^{(r+2)} = 1$
$k^{(r+2)} = 1$
end if

output: $\Lambda^{r+2}(x), \Delta^{(r+2)}(x), B^{(r+2)}(x), \Theta^{(r+2)}(x), \gamma^{(r+2)}, k^{(r+2)}$

The GF multipliers and adders used to compute $\Lambda^{(r+2)}(x)$ can be replaced by direct assignments of constant one and $\Delta_1^{(r)}$, actually S_1. Moreover, $t+1$ GF multipliers are removed in the calculation circuit of $\Delta^{(r+2)}(x)$. The MUXs used to update $B^{(r+2)}(x)$ can be reduced to two due to only the first two elements of $B^{(r+2)}(x)$ are needed for the next iteration. With those optimizations, the hardware complexity of the first riBM core reduces to

$$A_{riBM\_first} = 2t \cdot A_{mult} + 2t \cdot A_{add} + (2t+3) \cdot A_{MUX} + (4t+2) \cdot A_{reg}. \quad (4.33)$$

Given that no additional logic is incorporated into the core, the timing complexity remains the same as equation 4.20. The last riBM core can also be optimized (see algorithm 4) due to only the error location polynomial being needed.

Algorithm 4: The simplified last iteration of look-ahead riBM Algorithm

input: $\Lambda^{(r)}(x),\ B^{(r)}(x),\ \gamma^{(r)}$
begin: for $r = 2t-3$
$\Lambda^{(r+2)}(x) = \gamma^{(r)}\Lambda^{(r)}(x) + \Delta_1^{(r)} x^2 B^{(r)}(x)$
output: $\Lambda(x) = \Lambda^{(2t-1)}(x)$ when $r = 2t-1$

Then the hardware complexity of the last riBM core reduces to

$$A_{riBM\_last} = 2t \cdot A_{mult} + 2t \cdot A_{add} + 2t \cdot A_{reg}. \tag{4.34}$$

Unfortunately, the cores located between the first and last positions can not be significantly simplified since they require calculation of all parameters. For the riBM cores in between, the complexity is

$$A_{riBM\_btn} = (6t+2)\cdot A_{mult} + (3t+1)\cdot A_{add} + (3t+1)\cdot A_{MUX} + (6t+2)\cdot A_{REG}. \tag{4.35}$$

By pipelining the riBM cores, a UP structure with $t = 3$ is shown below

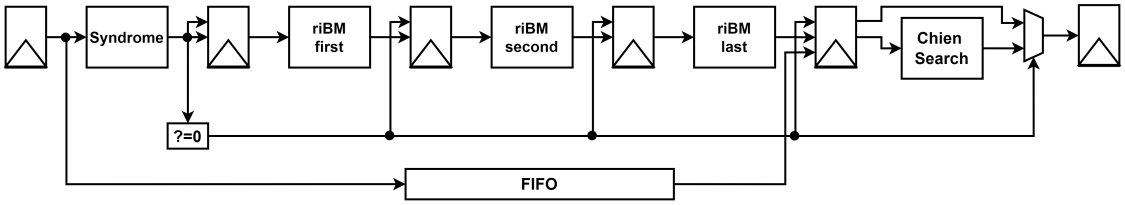


Figure 4.27: The structure of Unrolled Pipeline (UP) system with $t = 3$

The clock gating method is also implemented in this architecture aiming at reducing the total power consumption. Like the PI architecture, there is a controller to send the enable signal based on the resulting output of the syndrome. When there are no errors, the controller disables the riBM cores and the Chien search component, and the received codeword delayed by the FIFO directly connects to the output registers. It should be noted that due to the unrolling pipeline structure and each iteration is implemented by an individual core, the output MUXs in PE0 and PE1 are not needed and the timing complexity of riBM cores decreases to

$$T_{riBM\_UP} = T_{mult} + T_{add}. \tag{4.36}$$

Subsequently, the hardware and timing complexities are shown in figures 4.28 to 4.31. The Chien search component has the longest critical path from the range of $m = 5$ to $m = 10$ due to the one extra MUX. Regarding hardware complexity, the riBM cores exhibit the largest relative area usage when $m = 5, 6$, however, for other values of m, the Chien search dominates. In this design, the BCH code with $m = 6$ and $t = 3, 4$ are implemented.

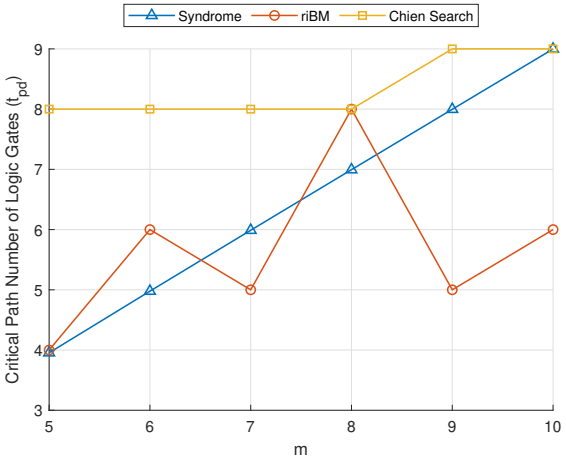


Figure 4.28: Number of logic gates in the critical path for a UP structure with $t = 3$

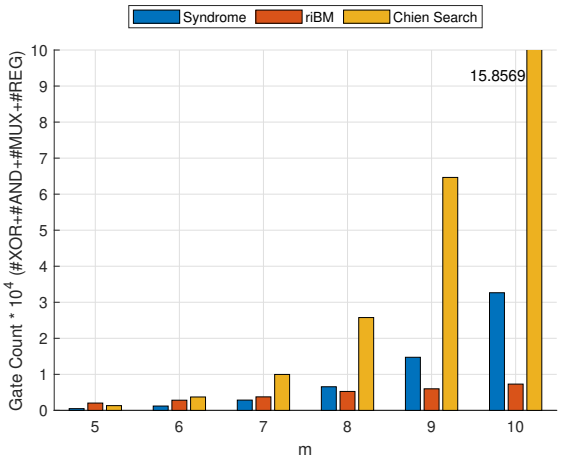


Figure 4.29: Gate count for a UP structure with $t = 3$

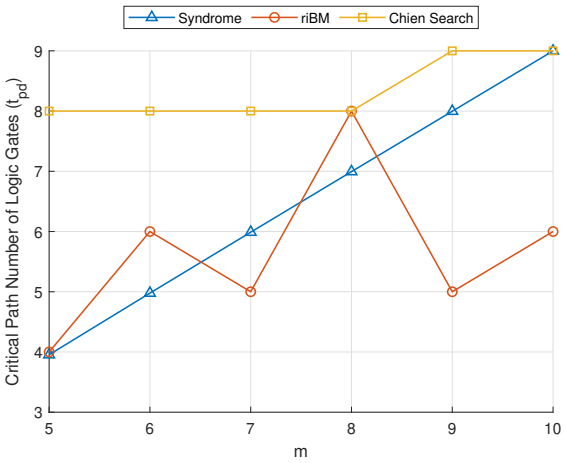


Figure 4.30: Number of logic gates in the critical path for a UP structure with $t = 4$

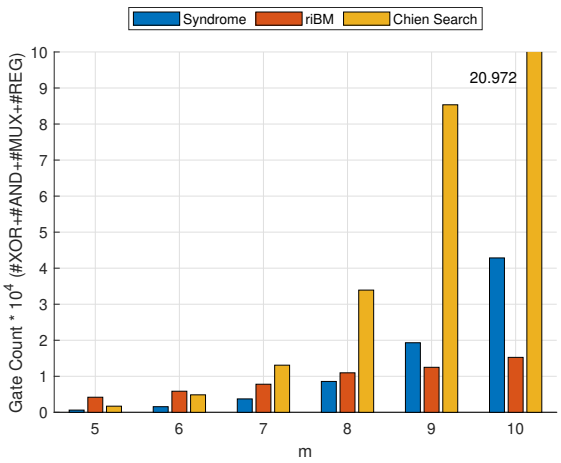


Figure 4.31: Gate count for a UP structure with $t = 4$

4.4.4 State of the Art

Figures 4.32 to 4.35 show a comparison of the complexities of different system architectures regarding the key equation solver components. The DI architecture has the shortest delay when $m = 5 - 7, 9$, and the UP has the shortest delay when $m = 8$, and for $m = 10$ three architectures have the same delay. Regarding the hardware complexity of riBM cores, the DI architecture has the lowest complexity due to only one core being used, but the UP has lower gate count compared with the PI architecture because of the optimization for the first and last riBM cores.

For $t = 3$, the gate count of UP in riBM cores is around half of the PI's, however, for $t = 4$ the gap is not that significant due to one more riBM core is added between the first core and last core. It should be noted that the area usage is inferred based on gate counts, only providing the relative relationship between various structures. In this design, the BCH decoders with $m = 6$ and $t = 3, 4$ are implemented.

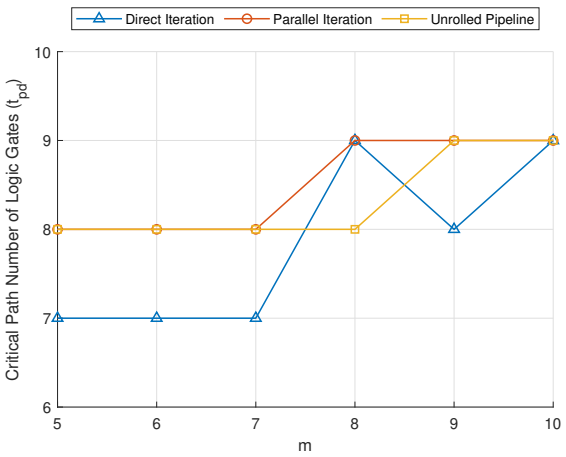


Figure 4.32: The comparison of logic gates count in the critical $t = 3$

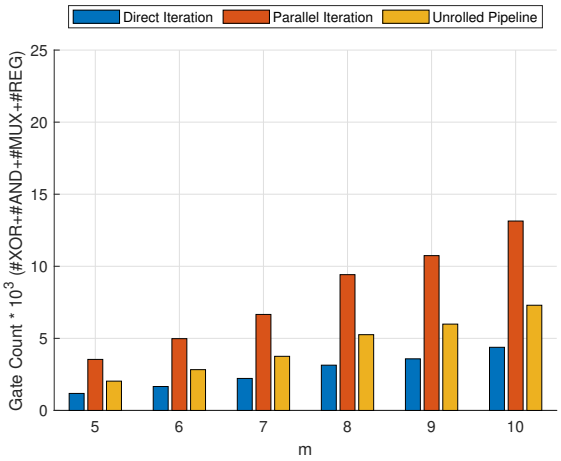


Figure 4.33: The comparison of gate counts of riBM cores for $t = 3$

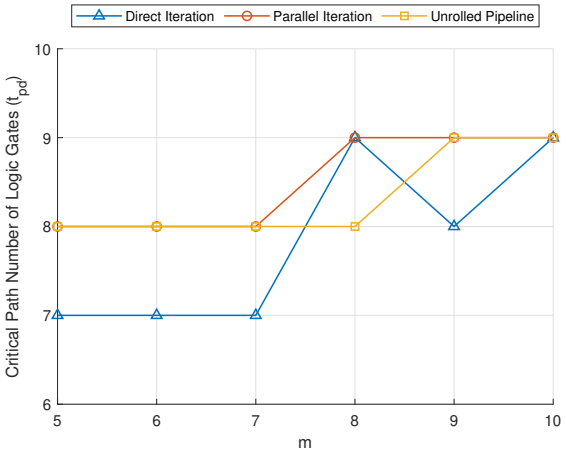


Figure 4.34: The comparison of logic gates count in the critical $t = 4$

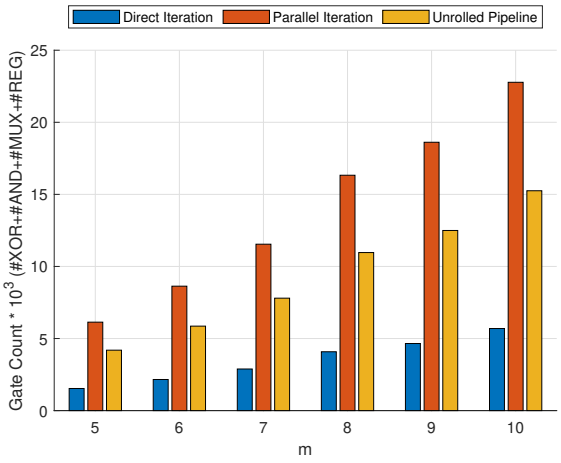


Figure 4.35: The comparison of gate counts of riBM cores for $t = 4$

5

Results

This chapter begins with the synthesis results of each individual component for $t = 3, 4$, and based on those components, the synthesis results of parallel iteration (PI) and unrolled pipeline (UP) are analyzed. Then a uniform random generator testbench is proposed to evaluate the performance of the post-synthesis circuit. In the end, the power is assessed including FEC, and a comparison is made with the system without FEC. All the results are synthesized in a 22-nm CMOS library using Cadence Genus.

For the individual component, the power estimation is derived from vectorless simulation, a valuable technique for analyzing circuit behavior and performance during the early stages of circuit design. Unlike test-vector based simulation, vectorless simulation does not require specific input test vectors, simplifying the process and allowing for a quicker assessment of circuit performance, such as area usage and performance. However, the vectorless power estimation may not provide precise or realistic results for the whole system. Therefore, for the whole system evaluation, the power estimation is based on test-vector based simulation, which requires the user-provided input test vectors with specific input BER.

5.1 GF Multiplier

In order to evaluate the performance of Mastrovito multiplier from [41], $m = 5 - 10$ multipliers are synthesized with varied timing constraints, spanning from 500MHz to 5 GHz with a step of 100 ps. Since the synthesized area usage is almost the same regardless of the timing constraint, only the area usage at 1 GHz is presented. In addition to the current multiplier, the synthesis results of the modular reduction multiplier [42], used in a previous design [8], are also provided for comparison (see figure 5.1).

For the timing (which has not been presented here), the Mastrovito multipliers have almost the same time performance as the modular reduction ones. The reason is that the delay of one or two logic gates can not lead to considerably different timing results compared with the synthesis timing step of 100 ps. All multipliers can be successfully synthesized up to 5 GHz, however, aiming at a small area usage, the synthesis clock frequency should be lower than 5 GHz. Regarding area usage, the performance of Mastrovito multipliers is also similar to the modular reduction multipliers, except for the multipliers with $m = 8$. The reason for that is the $m = 8$ multipliers utilize a pentanomial irreducible polynomial and the modular reduction

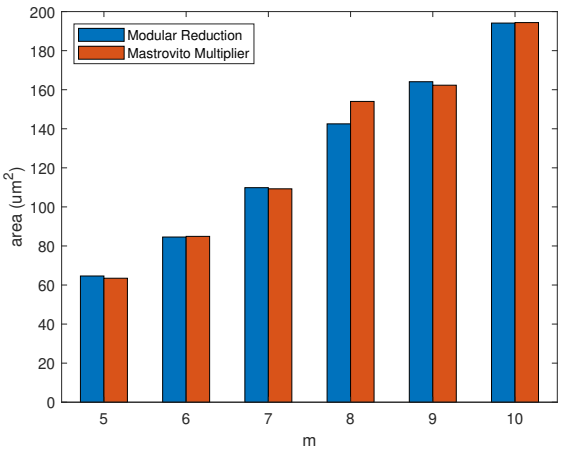


Figure 5.1: The area usage of the Mastrovito and modular reduction multiplier

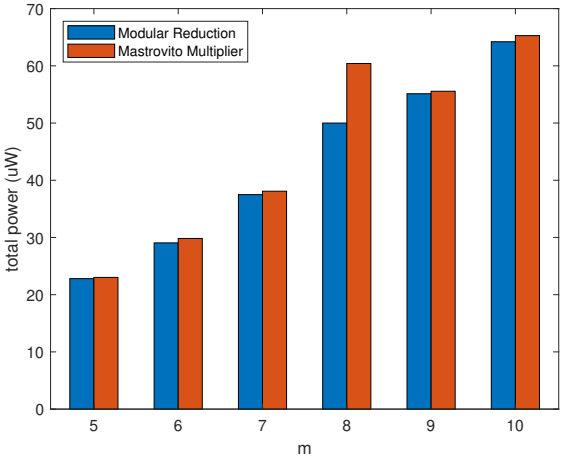


Figure 5.2: The power consumption of the Mastrovito and modular reduction multiplier

multipliers may perform better than the Mastrovito multiplier considering the area usage. The synthesis result is almost equivalent to the result of the evaluation model in figures 4.5 and 4.6, with the prediction that the Mastrovito multipliers have the same area usage as the modular reduction ones.

The power results are also presented in figure 5.2. Interestingly, the Mastrovito multiplier with $m = 8$ consumes more power than the one with $m = 9$ because of the pentanomial polynomial for $m = 8$. The power consumption increase follows m due to more XOR gates and input pins required.

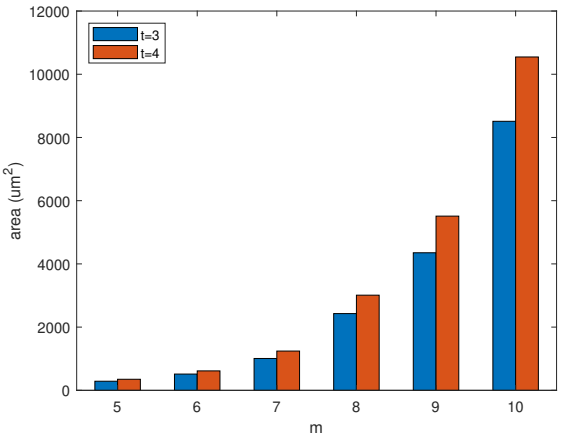


Figure 5.3: The area usage of $t = 3, 4$ syndrome components

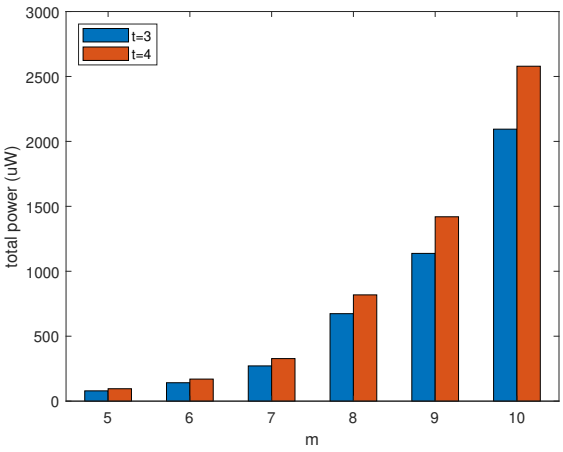


Figure 5.4: The power consumption of $t = 3, 4$ syndrome components

5.2 Syndrome Component

Like the synthesis of the GF multipliers, the syndrome components are also synthesized from 500 MHz to 5 GHz. The syndrome components with $t = 3, 4$ and

$m = 5 - 10$ are synthesized, and their area usage is provided in figure 5.3 for $t = 3, 4$. Similar to the result of the prediction model in chapter 4, the synthesis area utilization demonstrates an exponential increase as m increases. Although large m will lead to high area usage, a design based on large m provides high throughput. In addition, m has a more significant influence on the area usage compared with error correction ability t, especially when m is large since the pins will double when m increases by one.

The result of the power consumption follows the same trend as the result of the area usage (see figure 5.4). The power consumption increases as m increases. Subsequently, in order to reduce area usage and limit power consumption, the design with a small value of m should be implemented.

5.3 Key Equation Solver

The performance of the different key equation solvers based on the riBM algorithm is presented in this section. The synthesis frequency range is identical to the range of syndrome components and GF multipliers and those equation solvers are synthesized from $m = 5$ to $m = 10$. Due to the high latency of direct iteration (DI) architecture, the key equation solver with that architecture is not synthesized. Only unrolled pipeline (UP) and iteration parallel (IP) architectures are presented. It should be noted that for $m = 10$, the syndrome component cannot be successfully synthesized at 200 ps due to the long critical path, which is consistent with the result in chapter 4, the syndrome component dominates the critical path at $m = 10$.

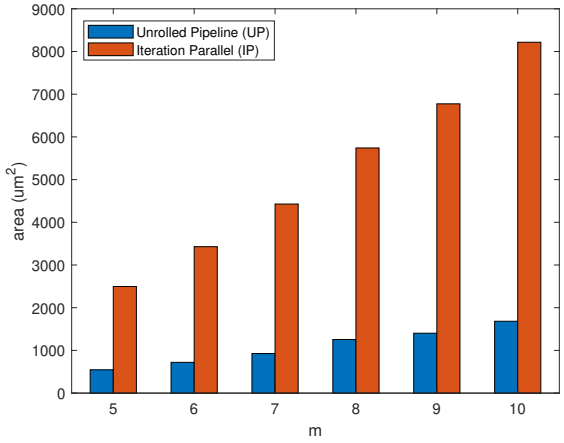


Figure 5.5: The area usage of the $t = 3$ riBM components

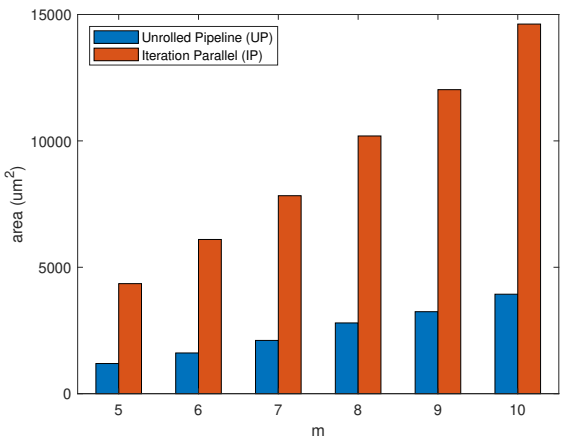


Figure 5.6: The area usage of the $t = 4$ riBM components

5.3.1 $t = 3$ riBM Component

The synthesis area usages of UP riBM and IP riBM are shown in figure 5.5. The area usage of IP riBM is around five times larger than the UP riBM, which indicated an increase of about two times compared with the result of the estimation model. The reasons are the estimation model does not include the input and output switching

MUXs in IP and neglects the influence of net connections. The estimation result is also based on the gate count, not including the actual area utilization for each gate. In addition, the area usage increases following the trends of GF multipliers because they are the main components for the riBM cores. For the IP riBM components, the circuit failed to be synthesized at 200 ps because of the longer critical path caused by pentanomial and the output MUXs.

The power estimation is based on vectorless simulation, which is not that precise and interesting (see figures 5.7 and 5.8). However, in general, power consumption follows the same trend as area usage, and when the clock frequency is reduced, the power consumption decreases.

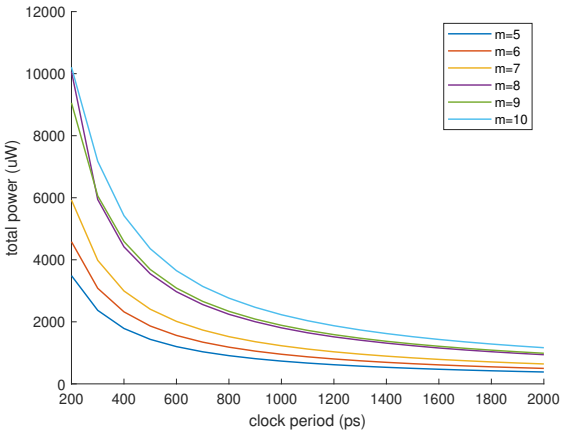


Figure 5.7: The power consumption of the $t = 3$ UP riBM component

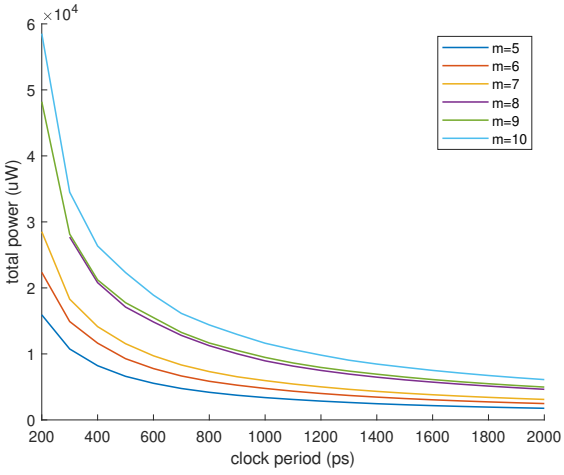


Figure 5.8: The power consumption of the $t = 3$ IP riBM component

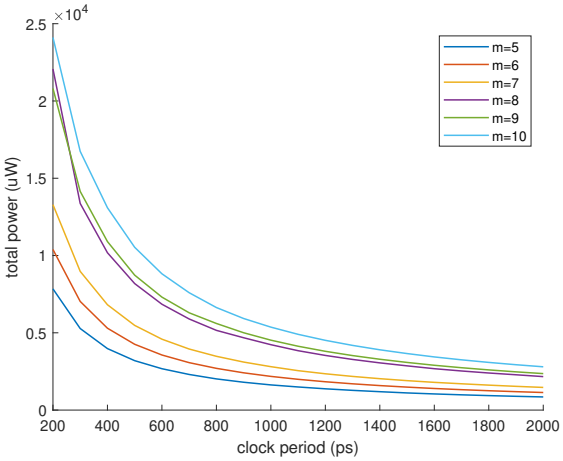


Figure 5.9: The power consumption of the $t = 4$ UP riBM component

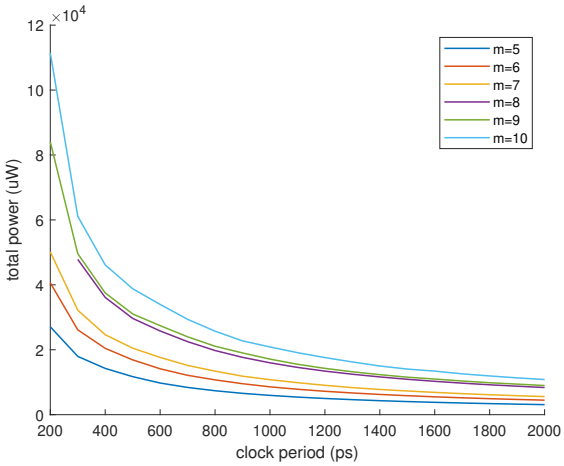


Figure 5.10: The power consumption of the $t = 4$ IP riBM component

5.3.2 $t = 4$ riBM Component

The hardware complexity of $t = 4$ riBM components is presented in figure 5.6. The ratio of IP over UP decreases to around 4 because of two inner cores added,

and this trend is the same as the result of the estimation model. The estimation model has not included the influence of input and output MUXs, the influence of net connection, and the actual area usage of different gates, which are the reason why the ratio of the estimation model is lower than the actual results. In addition, the area usage increases following the trends from the GF multipliers because it is the main component for the riBM cores. For the IP riBM components, the circuit failed to be synthesized at 200 ps because of the long critical path caused by pentanomial and the output MUXs.

Since the power estimation is based on vectorless simulation, it is not very precise but gives some indication (see figures 5.9 and 5.10). However, in general, power consumption follows the same trend as area usage, and when the clock frequency is reduced, the power consumption decreases.

5.4 Chien Search

In the estimation model, the Chien search components are the most hardware-consuming. In order to verify this result, the Chien search components are synthesized with $t = 3, 4$ from $m = 5$ to $m = 10$. The synthesis results are stated in figures 5.11 and 5.12.

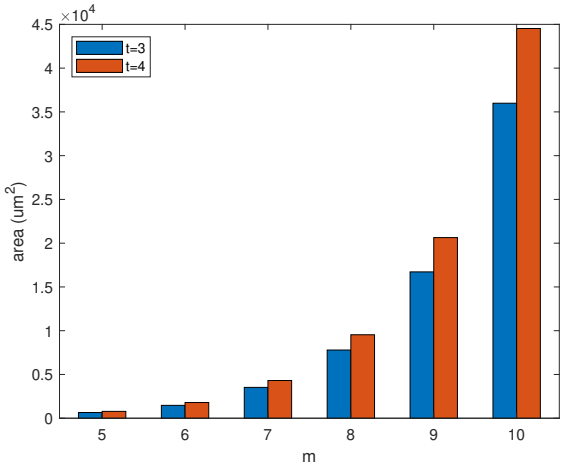


Figure 5.11: The area usage of Chien search components

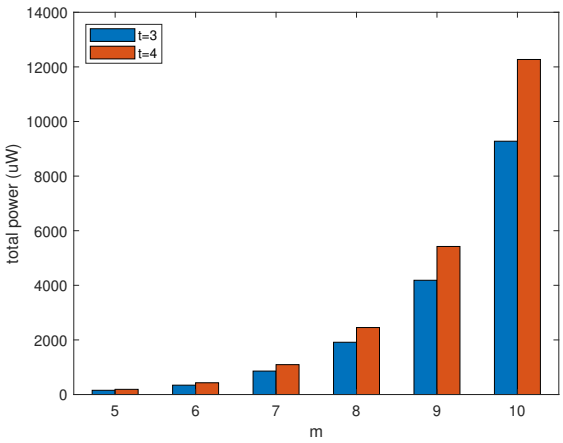


Figure 5.12: The power consumption of Chien search components

Based on the results in figure 5.11, it can be observed that the area usage exhibits an exponential increase as m increases. This trend is almost consistent with the prediction models. Furthermore, the synthesis results of Chien search support the argument that Chien search component dominates when $m \geq 6$ for $t = 3$ in terms of the hardware complexity. For $m \geq 9$, the component can not be synthesized at 200 ps because more XOR gates are required to calculate the roots. The power consumption shown in figure 5.12 also follows the trend of the area usage, and the Chien search component is the most power-expensive component for $m \geq 9$. In order to achieve the low-power consumption goal, the Chien search component should be disabled for received error-free codewords. Therefore clock gating technology is

required to enable and disable the Chien search component and to reduce power consumption.

5.5 BSC Simulation

In order to reduce the timing and hardware complexity, only $m = 6$ decoders are implemented based on the iteration parallel (IP) and unrolled pipeline (UP) system architecture. The decoders are simulated using a binary symmetric channel model to precisely evaluate the power consumption and output BER. The relationships of input and output BER are shown in figures 5.13 and 5.14. The prediction curves are generated by MATLAB with the theoretical formula for the BCH decoder, and the simulation curves are the resulting output of the post-synthesis netlists.

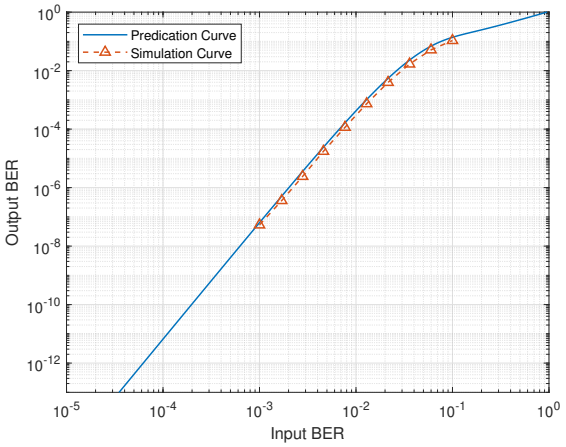


Figure 5.13: Predicated and simulated curve for $t = 3, m = 6$ decoder

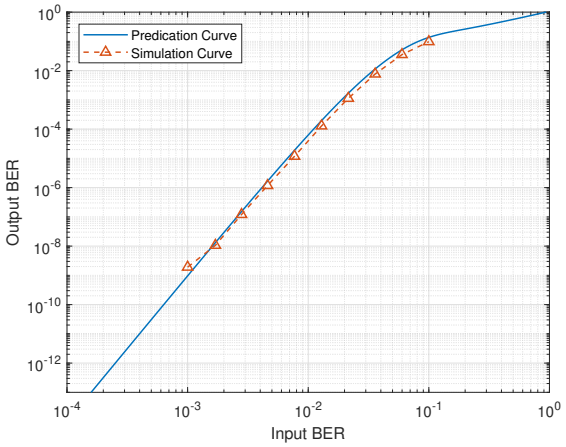


Figure 5.14: Predicated and simulated curve for $t = 4, m = 6$ decoder

It can be concluded that the simulation curves follow the prediction curves. The simulation result demonstrates a lower output BER compared to the predicted values. This improvement can be attributed to the ability of the decoder to detect more errors than it can correct. When it detects more errors than its correction ability, it avoids introducing additional errors to the noise code, resulting in a lower output BER compared to the prediction curve. The lift at the end of the simulated curve is caused by insufficient test errors. A low input BER means not many errors are presented in the received codewords and the decoder can correct most of them, resulting in a clean output. Thus depending on the throughput, it may take a long time, such as one week, to get enough errors in the output. The $t = 3, m = 6$ decoder possesses the capability to reduce the input of BER -3 dB by around 3 dB to -6 dB, relaxing the OMA requirement at the transmitter.

The VCD-based power simulation is employed in order to evaluate the practical performance of the decoder. The VCD files are generated according to specific input BER and contain the actual signal activities. Both the IP and UP system architectures are simulated, and the results are presented in figures 5.15 and 5.16.

The proposed simplified UP decoders offer a notable power saving of up to 30% compared with the IP decoder at an input BER of -1 dB. For the lower input BER,

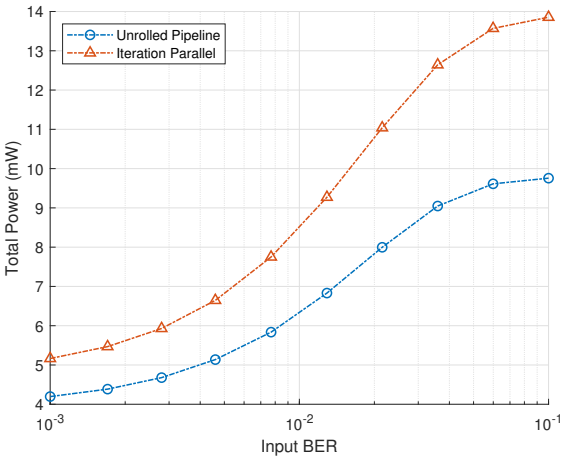


Figure 5.15: Power consumption for $t = 3, m = 6$ IP and UP decoders

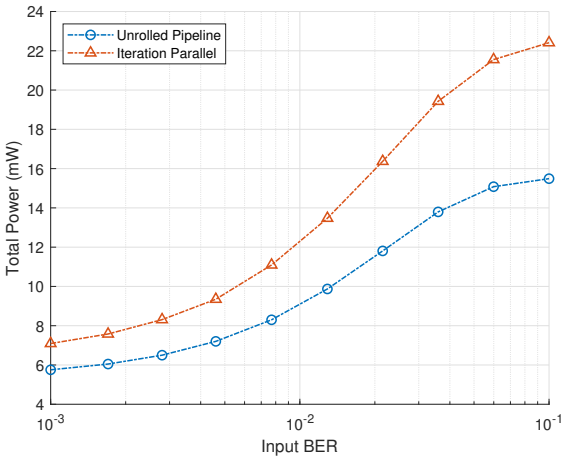


Figure 5.16: Power consumption for $t = 4, m = 6$ IP and UP decoders

such as -3 dB, the riBM cores are rarely used since they can be clock gated when an error-free block is received. Therefore, the power saving is not that significant, only 20% percent. Compared with the $t = 3$ decoder, the power consumption of $t = 4$ increases to around 1.5 times due to the more complex circuit.

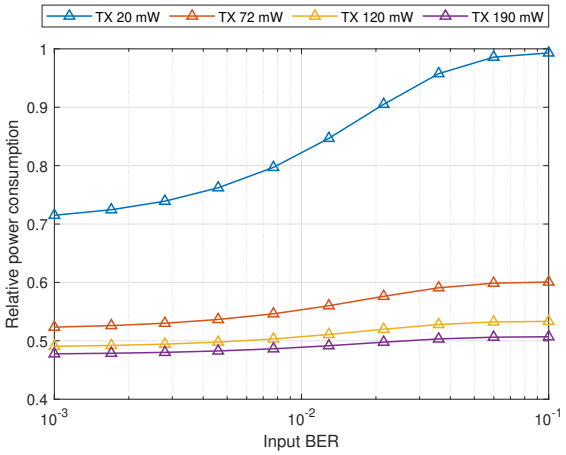


Figure 5.17: Relative system power consumption after adding FEC for $t = 3$ UP decoder

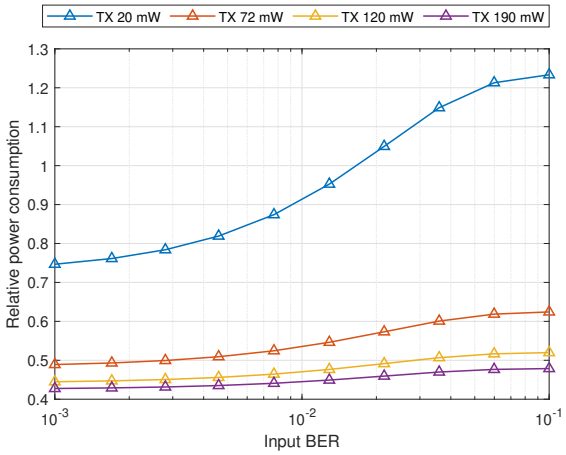


Figure 5.18: Relative system power consumption after adding FEC for $t = 4$ UP decoder

5.6 Post-FEC Power Saving

Introducing FEC may help to reduce power consumption in short-haul fiber-optic interconnects. In order to evaluate the total power consumption of the whole FEC, the $t = 3, 4$ encoders are also implemented, and switching activity is recorded and back-annotated to the netlist to perform more accurate power-dissipation estimation. Moreover, four transmitters [29, 30, 31, 32] are incorporated to simulate real-world transceiver systems. Figures 5.17 and 5.18 show the relative power consumption of

systems with FEC over systems without FEC under the 3.5-dB relaxation and 4-dB relaxation, respectively. For an ultra-low power transmitter system, the decoder can save a maximum of 29% power compared with the uncoded system. However, the power reduction is not significant and even the power consumption increases at a high input BER with a $t = 4$ FEC. Introducing FEC to a high power consumption transmitter system, such as the 72-mW, 120-mW, and 190-mW transmitter systems, results in more power saving. For example, incorporating a $t = 4$ FEC can relax the power dissipation up to 58% in a 190-mW transmitter system but with a low coding gain. A higher coding gain $t = 3$ FEC can still, however, offer 52% power dissipation reduction.

6

Conclusion

In this thesis, low-power BCH FECs are implemented to both reduce output BER and increase the data rate. Suitable FECs can also offer power relaxation for the whole fiber system. In an FEC, the decoder is typically more complex in comparison to the encoder and its complexity varies based on decoding algorithms. To identify appropriate decoders, an estimation model is proposed to assess the performance regarding timing and area usage based on gate count. It can correctly estimate the relationship between different key equation solver algorithms and different system architectures. With the estimation results, the riBM algorithm is employed for the key equation solver component, while the UP architecture is chosen for the overall system. By unrolling and optimizing the iterations, the unrolled (UP) decoder demonstrates a 20% reduction in power consumption compared to the parallel iteration (PI) decoder.

To further reduce the power consumption of decoders, clocking gating is employed to disable the key equation solver and the hardware-consuming Chien search component if error-free codewords are received. BCH decoders with $t = 3, 4$ and $m = 6$ are finally implemented on a 22-nm FDSOI CMOS technology in order to provide sufficient error correction ability and to limit area usage. A binary symmetric channel (BSC) simulator is implemented to validate that the output BER of the RTL netlist follows the trend of the predicted curve.

Four realistic transmitters are considered to evaluate the power relaxation and the corresponding encoders are also implemented to estimate the overall power consumption of BCH FEC. The best implementation (UP $m = 6$ and $t = 3$) can offer a maximum 58% power saving compared with the uncoded system at an input BER of 10^{-3} in a 190-mW transmitter system. Even for the ultra-low (20-mW) power consumption transmitter, the best implementation can still provide 29% power reduction. By integrating suitable low-power BCH decoders, short-reach optical systems can operate at a power-efficient point, thereby mitigating some of the problems of high-temperature environments. Additionally, the inclusion of FECs enhances the resilience of systems by providing error correction abilities.

Bibliography

[1] J. A. Tatum, D. Gazula, L. A. Graham, J. K. Guenter, R. H. Johnson, J. King, C. Kocot, G. D. Landry, I. Lyubomirsky, A. N. MacInnes, E. M. Shaw, K. Balemarthy, R. Shubochkin, D. Vaidya, M. Yan, and F. Tang, "VCSEL-based interconnects for current and future data centers," *Journal of Lightwave Technology*, vol. 33, no. 4, pp. 727–732, 2015.

[2] D. M. Kuchta, A. V. Rylyakov, C. L. Schow, J. E. Proesel, C. W. Baks, P. Westbergh, J. S. Gustavsson, and A. Larsson, "A 50 Gb/s NRZ modulated 850 nm VCSEL transmitter operating error free to 90 °C," *Journal of Lightwave Technology*, vol. 33, no. 4, pp. 802–810, 2015.

[3] Y. Lee, H. Yoo, I. Yoo, and I.-C. Park, "6.4Gb/s multi-threaded BCH encoder and decoder for multi-channel SSD controllers," in *2012 IEEE International Solid-State Circuits Conference*, 2012, pp. 426–428.

[4] W. Xueqiang, P. Liyang, W. Dong, H. Chaohong, and Z. Runde, "A high-speed two-cell BCH decoder for error correcting in MLC NOR flash memories," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 56, no. 11, pp. 865–869, 2009.

[5] K. Lee, S. Lim, and J. Kim, "Low-cost, low-power and high-throughput BCH decoder for NAND flash memory," in *2012 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2012, pp. 413–415.

[6] G. Tzimpragos, C. Kachris, I. B. Djordjevic, M. Cvijetic, D. Soudris, and I. Tomkos, "A survey on FEC codes for 100 G and beyond optical networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 209–221, 2016.

[7] K. Szczerba, C. Fougstedt, P. Larsson-Edefors, P. Westbergh, A. Graell i Amat, L. Svensson, M. Karlsson, A. Larsson, and P. A. Andrekson, "Impact of forward error correction on energy consumption of VCSEL-based transmitters," in *2015 European Conference on Optical Communication (ECOC)*, 2015, pp. 1–3.

[8] C. Fougstedt, K. Szczerba, and P. Larsson-Edefors, "Low-power low-latency BCH decoders for energy-efficient optical interconnects," *Journal of Lightwave Technology*, vol. 35, no. 23, pp. 5201–5207, 2017.

[9] E. Agrell, M. Karlsson, A. R. Chraplyvy, D. J. Richardson, P. M. Krummrich, P. Winzer, K. Roberts, J. K. Fischer, S. J. Savory, B. J. Eggleton, M. Secondini, F. R. Kschischang, A. Lord, J. Prat, I. Tomkos, J. E. Bowers, S. Srinivasan, M. Brandt-Pearce, and N. Gisin, "Roadmap of optical

communications," *Journal of Optics*, vol. 18, no. 6, p. 063002, may 2016. [Online]. Available: https://dx.doi.org/10.1088/2040-8978/18/6/063002

[10] W. Ryan and S. Lin, *Channel Codes: Classical and Modern.* Cambridge University Press, 2009.

[11] S. Jain, L. Song, and K. Parhi, "Efficient semisystolic architectures for finite-field arithmetic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 1, pp. 101–113, 1998.

[12] W. Kim, S. Kim, H. Cho, and K.-Y. Lee, "A fast-serial finite field multiplier without increasing the number of registers," in *2003 IEEE International Symposium on Circuits and Systems (ISCAS)*, vol. 5, 2003, pp. V–V.

[13] E. Mastrovito, "On fast Galois-field multiplication," in *Proceedings. 1991 IEEE International Symposium on Information Theory*, 1991, pp. 348–348.

[14] Y. Li, X. Ma, Y. Zhang, and C. Qi, "Mastrovito form of non-recursive Karatsuba multiplier for all trinomials," *IEEE Transactions on Computers*, vol. 66, pp. 1573–1584, 2017.

[15] N. Petra, D. D. Caro, and A. G. M. Strollo, "A novel architecture for Galois Fields $GF(2^m)$ multipliers based on Mastrovito scheme," *IEEE Transactions on Computers*, vol. 56, 2007.

[16] P. Larsson-Edefors, C. Fougstedt, and K. Cushon, "Implementation challenges for energy-efficient error correction in optical communication systems," in *Signal Processing in Photonic Communications, SPPCom*, 2018.

[17] R. W. Hamming, "Error detecting and error correcting codes," *Bell System Technical Journal*, vol. 29, pp. 147–160, 1950.

[18] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[19] "IEEE standard for Ethernet," *IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018)*, pp. 1–7025, 2022.

[20] E. Berlekamp, "Nonbinary BCH decoding," *IEEE Transactions on Information Theory*, vol. 14, no. 2, pp. 242–242, 1968.

[21] J. Massey, "Shift-register synthesis and BCH decoding," *IEEE Transactions on Information Theory*, vol. 15, no. 1, pp. 122–127, 1969.

[22] D. Gorenstein, W. W. Peterson, and N. Zierler, "Two-error correcting Bose-Chaudhuri codes are quasi-perfect," *Information and Control*, vol. 3, no. 3, pp. 291–294, 1960. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0019995860908779

[23] Z. Jun, W. Zhi-Gong, H. Qing-Sheng, and X. Jie, "Optimized design for high-speed parallel BCH encoder," in *Proceedings of 2005 IEEE International Workshop on VLSI Design and Video Technology, 2005.*, 2005, pp. 97–100.

[24] S. Lin and D. J. Costello, *Error control coding: fundamentals and applications.* Upper Saddle River, NJ: Pearson/Prentice Hall, 2004.

[25] G. C. Clark and J. B. Cain, *Error-Correction Coding for Digital Communications*. Perseus Publishing, 1981.

[26] R. Chien, "Cyclic decoding procedures for Bose-Chaudhuri-Hocquenghem codes," *IEEE Transactions on Information Theory*, vol. 10, no. 4, pp. 357–363, 1964.

[27] D. Sarwate and N. Shanbhag, "High-speed architectures for Reed-Solomon decoders," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 5, pp. 641–655, 2001.

[28] W. Peterson, "Encoding and error-correction procedures for the Bose-Chaudhuri codes," *IRE Transactions on Information Theory*, vol. 6, no. 4, pp. 459–470, 1960.

[29] A. Sharif-Bakhtiar, M. G. Lee, and A. C. Carusone, "A 40-Gbps 0.5-pJ/bit VCSEL driver in 28nm CMOS with complex zero equalizer," in *2017 IEEE Custom Integrated Circuits Conference (CICC)*, 2017, pp. 1–4.

[30] M. Khafaji, J. Pliva, R. Henker, and F. Ellinger, "A 42-Gb/s VCSEL driver suitable for burst mode operation in 14-nm Bulk CMOS," *IEEE Photonics Technology Letters*, vol. 30, no. 1, pp. 23–26, 2018.

[31] B. Sedighi *et al.*, "40 Gb/s VCSEL driver IC with a new output current and pre-emphasis," in *2012 IEEE/MTT-S Int. Microw. Symp.*, 2012.

[32] A. Malacarne, C. Neumeyr, W. Soenen, F. Falconi, C. Porzi, T. Aalto, J. Rosskopf, J. Bauwelinck, and A. Bogoni, "Optical transmitter based on a $1.3 - \mu m$ VCSEL and a SiGe driver circuit for short-reach applications and beyond," *Journal of Lightwave Technology*, vol. 36, no. 9, pp. 1527–1536, 2018.

[33] P. Westbergh, E. Haglund, E. Haglund, R. Safaisini, J. Gustavsson, and A. Larsson, "High-speed 850nm VCSELs operating error free up to 57Gbit/s," *Electronics Letters*, vol. 49, no. 16, pp. 1021–1023, 2013. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/el.2013.2042

[34] P. Westbergh, J. Gustavsson, B. Kögel, A. Haglund, A. Larsson, A. Mutig, A. Nadtochiy, and D. Bimberg, "850 nm VCSEL operating error-free at 40 Gbit/s," in *22nd IEEE International Semiconductor Laser Conference*, 2010, pp. 154–155.

[35] M. Elia, M. Leone, and C. Visentin, "Low complexity bit-parallel multipliers for $GF(2^m)$ with generator polynomial $x^m + x^k + 1$," *Electronics Letters*, vol. 35, pp. 551–552, 1999.

[36] H. Fan, "A chinese remainder theorem approach to bit-parallel $GF(2^n)$ polynomial basis multipliers for irreducible trinomials," *IEEE Transactions on Computers*, vol. 65, pp. 343–352, 2016.

[37] H. Fan and M. A. Hasan, "A survey of some recent bit-parallel $GF(2^n)$ multipliers," *Finite Fields Their Appl.*, vol. 32, pp. 5–43, 2015.

[38] Y. Li, G. Chen, and J. Li, "Speedup of bit-parallel Karatsuba multiplier in $GF(2^m)$ generated by trinomials," *Inf. Process. Lett.*, vol. 111, pp. 390–394, 2011.

[39] H. Fan, J. Sun, M. Gu, and K.-Y. Lam, "Overlap-free Karatsuba-Ofman polynomial multiplication algorithms," *Information Security, IET*, vol. 4, pp. 8 – 14, 04 2010.

[40] A. Halbutogullari and C. Koc, "Mastrovito multiplier for general irreducible polynomials," *IEEE Transactions on Computers*, vol. 49, no. 5, pp. 503–518, 2000.

[41] N. Petra, D. D. Caro, and A. G. M. Strollo, "A novel architecture for Galois Fields $GF(2^m)$ multipliers based on Mastrovito scheme," *IEEE Transactions on Computers*, vol. 56, 2007.

[42] A. Reyhani-Masoleh and M. A. Hasan, "Low complexity bit parallel architectures for polynomial basis multiplication over $GF(2^m)$," *IEEE Transactions on Computers*, vol. 53, pp. 945–959, 2004.

[43] H. Burton, "Inversionless decoding of binary BCH codes," *IEEE Transactions on Information Theory*, vol. 17, no. 4, pp. 464–466, 1971.

[44] T.-K. Truong, Y. W. Chang, and J. Jeng, "VLSI design of inverse-free Berlekamp-Massey algorithm for Reed-Solomon code," in *SPIE Optics + Photonics*, 2001.

[45] J.-I. Park and H. Lee, "Area-efficient truncated Berlekamp-Massey architecture for Reed-Solomon decoder," *Electronics Letters*, vol. 47, pp. 241–243, 2011.

[46] Z. Liang and W. Zhang, "Efficient Berlekamp-Massey algorithm and architecture for Reed-Solomon decoder," *Journal of Signal Processing Systems*, vol. 86, pp. 51–65, 2017.

[47] X. Zhang, *VLSI Architectures for Modern Error-Correcting Codes*. CRC Press, 2015.

[48] E. Mastrovito, "VLSI architectures for computation in Galois fields," *PhD Thesis (Linköping University)*, 1995.