AREA REDUCTION OF SYNDROME CALCULATOR FOR STRONG BOSE-CHAUDHURI-HOCQUENGHEM DECODER

By

KOAY KIM LEONG

A Dissertation submitted for partial fulfilment of the requirement for the degree of Master of Microelectronic Engineering

August 2016

ACKNOWLEDGEMENTS

First of all, I would like to say my deepest gratitude to my supervisor, AP Dr. Bakhtiar Affendi bin Rosdi for his guidance and patient capacity to improve the quality of this project. His advice and support lead me to the right path in completing this research project. I am pleased to be under his supervision. He guided me in conducting a proper research.

Next, I would like to thank my colleague PJ Tan for his extra time in ramping up me on operating the EDA tools for this research.

Last but not least, with my heartiest gratitude, I thank my wife for all her supports and extra time spent on our daughter in order to free me up for my academic life since the very beginning of this course. Without her, this thesis I would not be the same as presented here.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ARQ	Automatic Repeat Request
BCH	Bose–Chaudhuri–Hocquenghem
BM	Berlekamp-Massey
BMA	Berlekamp-Massey Algorithm
CS	Chien Search
ECC	Error Correction Code
EDA	Electronic Design Automation
FEC	Forward Error Correction
GF	Galois Fields
HDL	Hardware Description Language
LCM	Least Common Multiple
LDPC	Low-Density Parity-Check
MLC	Multi-Level Cell
RS	Reed-Solomon
RTL	Register Transfer Logic
SC	Syndrome Calculation or Syndrome Calculator
SLC	Single Level Cell
SSD	Solid-State Drives
SV	System Verilog
VCS	Verilog Compiler Simulator
WBAN	Wireless Body Area Network
XOR	Exclusive OR

PENGURANGAN KELUASAN KALKULATOR SINDROM UNTUK DEKODER BOSE-CHAUDHURI-HOCQUENGHEM YANG KUAT

ABSTRAK

Kod Bose–Chaudhuri–Hocquenghem (BCH) mempunyai penggunaan yang meluas untuk memberi perlindungan ralat untuk berbilang ralat rawak dalam kod binari. Ini merupakan faktor penting untuk menggunakan Kod BCH biasanya digunakan dalam pelbagai aplikasi seperti "solid-state drives" (SSDs) dan sistem komunikasi gentian optik berkelajuan tinggi, sistem komunikasi tanpa wayar. Operasi dalam dekoder BCH boleh dirumuskan kepada 3 langkah: 1) mengira sindrom daripada kod diterima; 2) pengiraan polinomial pengesanan ralat; 3) mengesan ralat daripada kod diterima. Projek penyelidikan ini mencadangkan blok kalkulator sindrom yang cekap untuk BCH (n = 255, k = 111, t = 18) dekoder dari segi penggunaan keluasan perkakasan. Dalam seni arkitek blok kalkulator sindrom sebelumnya, semua sindrom ganjil perlu dikira dengan pengiraan langsung yang memerlukan lebih keluasan. Dalam seni arkitek yang dicadangkan, ciri-ciri Galois field telah dieksploitasi untuk mengira sindrom ganjil dengan menggunakan kaedah operasi kuasa untuk menjimatkan penggunaan keluasan. Seni arkitek yang dicadangkan adalah lebih baik dari segi penggunaan keluasan berbanding dengan seni arkitek sebelumnya. Kesimpulannya, dengan mengira sindrom ganjil indeks dengan operasi kuasa, 8% penjimatan keluasan dicapai tanpa menjejaskan penggunaan kuasa dan frekuensi operasi.

AREA REDUCTION OF SYNDROME CALCULATOR FOR STRONG BOSE-CHAUDHURI-HOCQUENGHEM DECODER

ABSTRACT

Bose–Chaudhuri–Hocquenghem (BCH) codes have a widespread use to provide the error protection for multiple random errors in a binary code. BCH codes is commonly applied in various practical application such as advanced solid-state drives (SSDs), high-speed fiber optical communications system and wireless communication system. The operation in a BCH decoder can be summarized into 3 steps: 1) compute the syndromes from the received codeword; 2) computing the error locator polynomial; 3) locating the errors. This research project proposed an area efficient Syndrome Calculator block of the BCH (n=255, k=111, t=18) decoder. In the previous SC block architecture, all the odd-index syndromes need to be computed by direct calculation which consume more area. In the current proposed architecture, Galois field's property is exploited to compute the odd-index syndromes by using power operation in order to save the area consumption. This architecture is better in terms of area compared with previous architecture. In conclusion, by computing the odd-index syndromes with power operation, 8% area saving is achieved without compromising the power consumption and its operating frequency.

CHAPTER 1

INTRODUCTION

1.1 Background

Error-correction codes (ECC) are techniques that provide the delivery of digital data reliably over an unreliable communication channels. Many communication channels are subject to noise and interference. Errors may be introduced from the source to the receiver during transmission. Error detection techniques enable the detection of such errors, while error correction allow restoration of the original data in many cases. ECC have a widespread use in communication systems to recover errors caused by poor environment. There are many types of ECC such as Hamming codes, Bose–Chaudhuri–Hocquenghem (BCH) codes, Reed–Solomon (RS) codes, turbo codes and low-density parity-check codes (LDPC). Hamming codes is one of the earliest ECC [1] [2] [3]. BCH codes and RS codes are among the most popular codes due to their widespread use in current communication systems [1] [2] [3]. Turbo codes and LDPC codes are relatively new constructions that can provide almost optimal efficiency [1] [2].

BCH codes is one of the most commonly applied error-correction code in many communication system. For instance, BCH codes applied to one of the standard that is most common choices in Digital TV broadcasting system [4] [5]. Besides, BCH codes is chosen to be implemented in Wireless Body Area Network (WBAN) for its low power consumption advantage [6]. Recent years, BCH codes is applied to

cryptographic hardware designs that need to store some high security information as well. This because the occurrence of malicious attack increases drastically due to the widespread usage of online activities globally [7].

Apart from that, recent applications of data storage system such as advanced solid-state drives (SSDs) are heavily rely on BCH code to correct the errors occurred in the memory cell [8] [9]. High demand for increased storage capacity has resulted in the introducing multi-level cell (MLC) from single level cell (SLC) to reduce the production cost. However, MLC is experiencing higher error rate as compared to previous SLC. BCH codes are added to detect and correct the error introduced in the storage devices. High speed BCH decoding performance and high error-correction capability are greatly demanded. Massive parallel BCH decoding is able to satisfy such a high-throughput and high error correction requirement by paying the additional cost to the area consumption. However, larger area resulted higher power consumption and lower die utilization of the storage devices. Therefore, a strong and high performance but yet small size of BCH decoder is required to overcome the issue. A BCH decoder is considered strong if it can correct *5* or more errors [31].

BCH codes is popular for its capability to correct multiple random error in a binary code. Also, BCH codes is known to be cost effective, reliable, flexibility and most importantly its simplicity in implementation [10]. BCH codes are cyclic codes which work under Galois Field (GF). The Galois fields or Finite fields' theory defines the properties of BCH codes. In general, development of a BCH decoder can be summarized into three steps: 1) syndromes calculation (SC) from the received codeword; 2) computing the error locator polynomial by using Berlekamp-Massey algorithm (BMA); 3) finding the error locations by applying Chien Search (CS).

In this project, syndrome calculation from the received codeword is carried out by the combination of direct computation and power operation in binary Galois fields. The direct computation unit is comprising of p-parallel syndrome calculation unit which process p-bit of codeword in an iteration. Power operation in binary Galois fields unit consist of a series of XOR logic gates. For computing the error locator polynomial, inversion-less BMA is chosen [11] to eliminate the complex calculation of inverses in Galois fields. Lastly, for the sake of area consideration, the conventional serial Chien Search [1] is selected to find the error locations.

1.2 Problem Statements

In order to increase the performance of the decoder, each sub-block of a BCH decoder can be implemented with a large parallel factor. Several optimization schemes have been developed for the Chien search to increase its performance as well as reducing the area consumption [12] [13]. On the other hand, there are several enhancement proposed by researchers to relax the complexity of a BMA design. For example, BMA architecture proposed in [14] reduces the area consumption, while BMA architecture proposed in [15] reduces the latency of the BMA block. In terms of SC block, performance of calculation was improved by implementing the parallel syndrome calculation unit in the SC block. This is to reduce the number of iterations required to calculate all the syndromes.

Error correcting capability of BCH decoder is also affecting the area consumption of the design. However, SC block is the one that mainly impacted because more parallel syndrome calculation units required to calculate all syndromes.

The SC block in [8] proposed to exploit Galois fields' property to compute the even-index syndromes from odd-index syndromes by power operation. Result shown signification improvement of more than 50% of area reduction. One year later, the same group of researchers proposed another innovation to further reduce the area consumption around *10% - 20%* by eliminating the duplicate calculation of the common sub-expression (CSE) of GF multiplication [16]. Even though both of the proposed architectures shrink the area consumption significantly, still the direct computation syndromes are required for all of the odd-index syndromes.

The specific focus area of the project is a continuous research to develop a new architecture to decrease the number of direct computation of the syndrome in the SC block. This is to further reduce the area of an SC block for a strong BCH decoder while not sacrificing its decoding performance.

1.3 Objectives

The objectives of the research project are as follows:

1. To propose a better architecture to reduce the area consumption of the SC block of a BCH decoder without sacrificing its performance.
2. To implement the proposed architecture into a RTL and synthesize the design to obtain the area report in order to justify the result.

1.4 Research Scope

The scope of this research project consists of:

1. Review of the previous state-of-the-art of the BCH SC block.
2. RTL implementation and simulation of the BCH decoder with the proposed architecture of new SC block by using System Verilog (SV) Hardware Description Language (HDL). The RTL design is verified by using Synopsys VCS simulator.
3. RTL logic synthesis of the design for comparison in between the proposed architecture and the previous BCH decoder. The logic synthesis process is carried out by using Synopsys Design Compiler tools.
4. The proposed architecture mainly focus on the area optimization of the SC block in a BCH decoder without compromising its performance and power consumption.

1.5 Thesis outline

This thesis consists of five main chapters.

In chapter 2, an overview of the BCH codes and its properties is presented and Galois Fields will be discussed. Next, the general BCH encoder and decoder are discussed. Then, the conventional architecture of the SC block and several enhancements that have been proposed by other researchers on Syndrome Calculation are discussed here as well.

Chapter 3 discuss the methodology of this research project in detail. First of all, the proposed architecture of designing a small SC block is explained. Next, the details design flow of the proposed architecture is described. The design languages that used to implement the RTL and the tools that used to simulate and synthesis the design are presented as well. The RTL architecture of the proposed SC block is explained in detail and the comparison in between proposed method and the previous architecture are presented. Subsequently, the test bench that used to verify the functionality of the design is discussed. Lastly, the flow that used to justify the performance of the proposed architecture is explained.

In chapter 4, the simulation results of the RTL design are presented and discussed. Next, the logic synthesis results are analysed and discussed in various aspect such as area consumption, power consumption and maximum operating frequency. The simulation results and logic synthesis results are summarized in this chapter as well.

Last but not least, chapter 5 gives the conclusion regarding the overall research. Discussions and recommendations for future works on this project are highlighted as well.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter provides some basic concepts for better understanding of this research. First of all, it is important to identify and understand the goal of this research. Related researches on BCH code and current existing design architectures are described in this chapter. This chapter begin with the basic introduction of ECC. Next basic concept of Galois Fields that define the properties of BCH codes will be presented. Subsequently, the overview of the BCH codes and its properties will be discussed. In continuation with that, the conventional architecture of the SC block of a BCH decoder and several enhancements that have been proposed by other researchers on SC block are discussed as well.

2.2 Error correction codes (ECC)

Digital communication systems are very common in our daily lives. The most common examples include cell phones, digital television, and digital radio and internet connections [1]. Each of these examples are generally fits into a common digital communication system block diagram as shown in Figure 2-1. The block diagram

shows two types of encoders and decoders, there are source encoder and decoder together with channel encoder and decoder.

Source encoder converts the information source bit sequence into another bit sequence with a more efficient representation of the information. This operation is more often called compression. The source decoder is the encoder's counterpart which recovers the source sequence.

The function of the channel encoder is to protect the source sequence bits to be transmitted over a noisy channel. The encoder converts its input into an alternate sequence that provides immunity from the various channel impairments. On the other side, the role of the channel decoder is to retrieve compressed sequence bits that input to the channel encoder regardless of the presence of noise, distortion, and interference in the received word from the channel output.

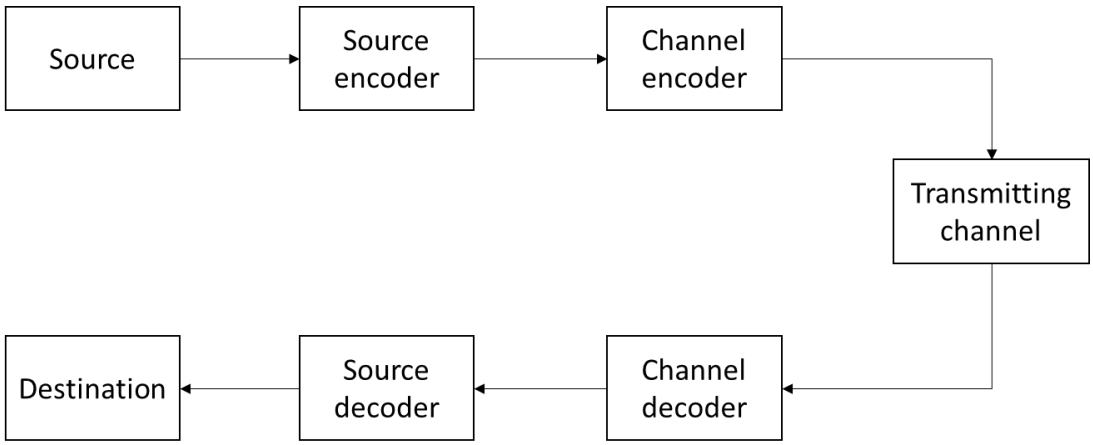

Figure 2-1 Basic digital communication system block diagram

There are huge number of channel coding techniques for the error prevention. There are two main basic techniques namely automatic request-for-repeat (ARQ) schemes and forward-error-correction (FEC) schemes [1]. In ARQ schemes, the

function of the code is simply to detect whether the received word contains any errors. A request will be generated for retransmission of the same word from the receiver back to the transmitter if a received word does contain one or more errors. This type of codes are said to be error-detection codes. In FEC schemes, the code is capable to correct the error detected through a decoding algorithm. The codes for this approach are said to be error-correction codes (ECC).

ECC mechanism is implemented in two inverse operations, encoding and decoding operation. The former operation is carried out by adding redundancy bits to the message or information bits to form a longer binary sequence called codeword. This operation is called encoding operation. The second operation is to retrieve the message bits by excluding the redundancy bits from the received codeword. The redundancy bits is often called parity check bits.

In block coding, an information sequence is segmented into message blocks of fixed length. Each message block consists of k message bits and there are 2^k unique messages. At channel encoder, each input message sequence of k message bits is encoded into an n-bits codeword with $n > k$. Each codeword are one to one mapped to each message. Since there are 2^k distinct messages, there are 2^k unique codewords as well.

The codeword is more commonly represented in the form of *(n, k)* block code. There are $n - k$ parity check bits that are added to each input message sequence by the channel encoder. The purpose of adding parity check bits is to provide the codeword with the error detecting and error correcting capability. These parity check bits do not carry any new information. The ratio, $R = k/n$ is called the code rate, which is interpreted as the average number of information bits carried by each codeword bit.

By definition [3] a binary *(n, k)* block code of length n with 2^k codewords is known as a linear *(n, k)* block code if and only if the 2^k codewords form a k-dimensional subspace of the vector space, V_n of all the n-tuples over the field GF(2). In another word, it may be seen that in a binary linear code, the modulo-2 sum of any pair of code words generate another codeword.

There are many types of ECC such as Hamming codes, BCH codes, RS codes, turbo codes and LDPC codes. BCH codes is one of the most popular codes for current applications for its capability to correct multiple random error in a binary code, effective, reliable, flexibility and most importantly its simplicity in implementation [10]. BCH codes are cyclic codes which operate under Galois Field (GF). The Galois Field's theory defines the properties of BCH codes.

2.3 Galois Field (GF)

Galois field also known as finite field. It is the fields that contain finite numbers of elements. Galois field play an important role in the construction of error-correction codes that can be efficiently encoded and decoded. The set of integers, $\{0, 1, \dots, p-1\}$, forms a finite field GF(p) of order p under modulo-p addition and multiplication, where *0* and *1* are the zero and unit elements of the field.

2.3.1 Properties of Galois Fields

Some of the useful properties of a Galois field [1] are:

- All elements in GF are defined on two binary operations which are addition and multiplication.
- Both addition and multiplication operations are commutative, associative, and distributive.
- The result of the binary operation must be an element in the GF.
- The identity element of addition operation is called the "zero" element, such that $a + 0 = a$ for any element a in the field.
- The identity element of multiplication is called the unit element, such that $a * 1 = a$ for any element a in the field.
- For every element "a" in the GF, there is an inverse of addition element "b" such that a + b = 0.
- For every non-zero element "a" in the GF, there is an inverse of multiplication element "b" such that $ab = 1$.
- Subtraction can be defined as addition of the inverse whereas division can be defined as multiplication by the inverse.

2.3.2 Binary field GF(2)

The simplest Galois field is GF(*2*). Its elements are the set *{0, 1}* under modulo-*2* addition and multiplication. Addition and subtraction are the same. The addition and

multiplication operation of two elements, A and B in GF(*2*) are shown in Table 2-1 and Table 2-2 respectively.

Table 2-1 Modulo-*2* addition of two elements, A and B in GF(*2*)

A \ B	0	1
0	0	1
1	1	0

Table 2-2 Modulo-*2* multiplication of two elements, A and B in GF(*2*)

A \ B	0	1
0	0	0
1	0	1

2.3.3 Extended Binary Field GF(2^m)

The Galois field GF(2^m) contains GF(*2*) as a subfield and is an extension field of GF(2). Let us suppose $q = 2^m$, for any positive integer m, a Galois field GF(q) with q elements can be constructed based on the prime field GF(*2*) and the primitive element, α of the GF(q). The power of α are from α^0 to α^{q-2} and zero element from the GF(q). It is given that,

$$\alpha^{2^m-1} = \alpha^0 = 1 \tag{2.1}$$

Since addition and subtraction in GF(q) are the same, therefore,

$$\alpha^{2^m-1} + 1 = 0 \quad (2.2)$$

Construction of Galois field GF(q) elements is based on irreducible primitive polynomial denoted as p(X) with degree m, this polynomial need to be a factor of $X^{2^m-1} + 1$ [3]. For example, in GF(2^3) the factors of $X^7 + 1$ are:

$$X^7 + 1 = (X + 1)(X^3 + X^2 + 1)(X^3 + X + 1) \quad (2.3)$$

For both of the polynomials of degree *3* are primitive and irreducible that can be chosen. Let us choose the polynomial shown in equation (2.1).

$$p(X) = X^3 + X^2 + 1 \quad (2.4)$$

Let us suppose the primitive element α be the root of the primitive polynomial. By substituting α into equation (2.4),

$$p(\alpha) = \alpha^3 + \alpha^2 + 1 = 0 \quad (2.5)$$

Rearranging equation (2.5), the equation can be represented as equation (2.6),

$$\alpha^3 = \alpha^2 + 1 \quad (2.6)$$

The other non-zero elements of GF(2^3) can be computed as:

$$\alpha^4 = \alpha \times \alpha^3 = \alpha \times (\alpha^2 + 1) = \alpha^3 + \alpha = (\alpha^2 + 1) + \alpha = \alpha^2 + \alpha + 1 \quad (2.7)$$

$$\alpha^5 = \alpha \times \alpha^4 = \alpha \times (\alpha^2 + \alpha + 1) = \alpha^3 + \alpha^2 + \alpha = \alpha + 1 \quad (2.8)$$

$$\alpha^6 = \alpha \times \alpha^5 = \alpha \times (\alpha + 1) = \alpha^2 + \alpha \quad (2.9)$$

$$\alpha^7 = \alpha \times \alpha^6 = \alpha \times (\alpha^2 + \alpha) = \alpha^3 + \alpha^2 = 1 = \alpha^0 \quad (2.10)$$

All the eight elements in GF(2^3) can be computed by the primitive polynomial chosen from equation (2.4), are $\{0, \alpha^0, \alpha^1, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6\}$. All the elements starting from α^4 to α^6 are presented function of α^0, α^1 and α^2 which are called the basis of the Galois field.

2.3.4 Representation of Galois Field Elements

The elements in GF can be represented in three different forms namely power representation, polynomial representation and vector representation. Let α be the primitive element of GF(2^3) and the primitive polynomial is given by equation (2.4). The elements in GF(2^3) can be represented in three different forms as shown in Table 2-3.

Table 2-3 GF(2^3) generated by the primitive polynomial $p(X) = X^3 + X^2 + 1$ over GF(2)

Power representation	Polynomial representation	Vector representation $\alpha^2, \alpha^1, \alpha^0$
0	0	000
1	1	001
α^1	α^1	010
α^2	α^2	100
α^3	$1 + \alpha^2$	101
α^4	$1 + \alpha + \alpha^2$	111
α^5	$1 + \alpha$	011
α^6	$\alpha + \alpha^2$	110

2.4 BCH Code

In coding theory, the BCH codes form a class of cyclic codes that are able to correct multiple random errors. BCH codes were discovered by Hocquenghem back in

1959 [17], and independently discovered by Bose and Chaudhuri in 1960 [18]. BCH codes are specified in terms of the roots of their generator polynomials in finite fields. For any positive integer $m \geq 3$ and $t < 2^{m-1}$, there exists a binary BCH code with the following parameters:

- Block length: $n = 2^m - 1$
- Number of parity-check digits: $n - k \leq mt$
- Minimum distance: $d \geq 2t + 1$
- Error correcting capability t

This BCH code is capable of correcting t or fewer random errors over a span of $2^m - 1$ transmitted code bits. It is called a *t*-error-correcting BCH code. Figure 2-2 shows the typical BCH encoding and decoding operation. The encoded BCH codewords, $v(x)$ were sent to the receiver via a transmitting channel subject to noise and interference. The received BCH codewords, $r(x)$ with error at the receiver were stored in the buffer temporary. At the same time, the received codeword were fed into the BCH decoder to locate the error, $e(x)$ injected into the original encoded codeword. Finally, the error located is XOR'ed with the received codeword that stored in the buffer to retrieve the original encoded codeword.

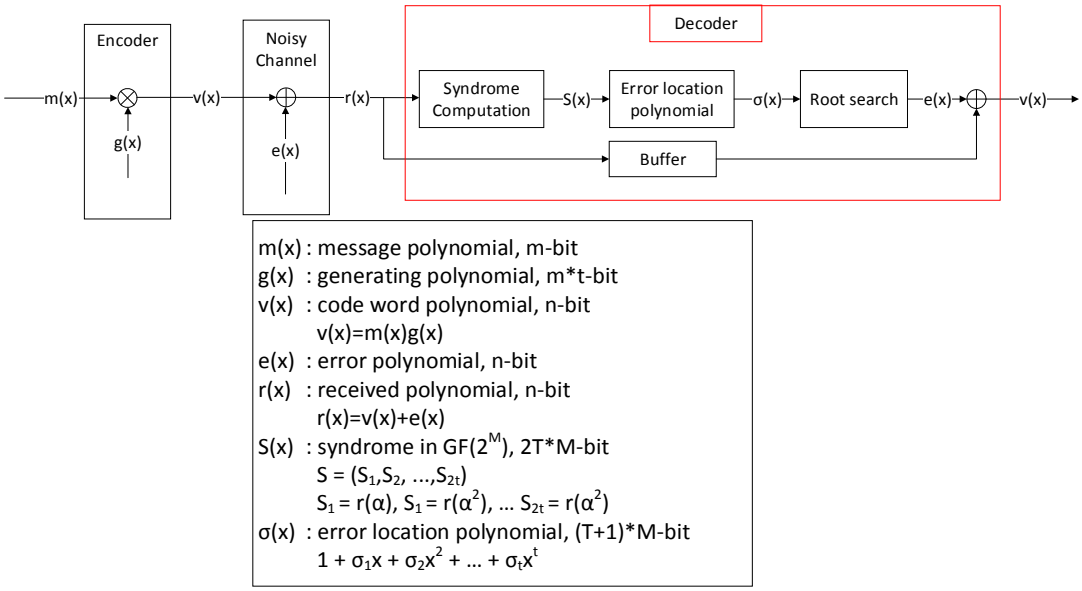

Figure 2-2 Typical BCH encoding and decoding operation

2.4.1 BCH Code Construction

Construction of a *t*-error-correcting BCH code begins with a Galois field $GF(2^m)$:

- Let α be a primitive element in $GF(2^m)$.
- The generator polynomial, $g(x)$ of the t-error-correcting binary BCH code of length $2^m - 1$ is the smallest-degree polynomial over $GF(2)$ that has the following *2t* consecutive powers of α as its roots.

$$\alpha, \alpha^2, \alpha^3, \dots, \alpha^{2t}$$

- $g(x)$ has $\alpha, \alpha^2, \alpha^3, \dots, \alpha^{2t}$ and their conjugates as all of its roots.

$$g(\alpha^i) = 0 \text{ for } 1 \leq i \leq 2t \qquad (2.11)$$

- For $1 \leq i \leq 2t$, let $\varphi_i(x)$ be the minimal polynomial of α^i. Then $g(x)$ is given by the least common multiple (LCM) of $\varphi_1(x), \varphi_2(x), \ldots, \varphi_{2t}(x)$, that is:

  $$g(x) = LCM\{\varphi_1(x), \varphi_2(x), \ldots, \varphi_{2t}(x)\} \quad (2.12)$$

- If *i* is an even integer, *j* is an odd integer and $k > 1$, and *i* can be expressed as $i = j2^k$. Then $\alpha^i = \left(\alpha^j\right)^{2^l}$ is a conjugate of α^j. Therefore,

  $$\varphi_i(x) = \varphi_j(x) \quad (2.13)$$

- Generator polynomial can be simplified as equation given by:

  $$g(x) = LCM\{\varphi_1(x), \varphi_3(x), \ldots, \varphi_{2t-1}(x)\} \quad (2.14)$$

The degree of $g(x)$ is at most *mt*, That is, the number of parity-checks digit, $n - k$, of the code is at most equal to mt.

2.4.2 Syndrome Calculation (SC)

Suppose a code polynomial $v(x)$ of a t-error-correcting BCH code, $r(x)$ be the corresponding received polynomial and $e(x)$ be the error pattern:

- The received polynomial is $r(x)$ given by:

  $$r(x) = v(x) + e(x) \quad (2.15)$$

- The syndrome of $r(x)$ which consists of *2t* syndrome components is given by:

  $$S = (S_1, S_2, \cdots, S_{2t}) = r \cdot H^T \quad (2.16)$$

- For $1 \leq i \leq 2t$, the i^{th} syndrome component is given by:

  $$S_i = r\left(\alpha^i\right) = r_0 + r_1\alpha^i + r_2\alpha^{2i}, + \cdots + r_{2^m-2}\alpha^{(2^m-2)i} \quad (2.17)$$

- Since $\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{2t}$ are roots of each code word polynomial, $v(\alpha^i) = 0$ for $1 \leq i \leq 2t$. From equation (2.15) and equation (2.17), the i^{th} syndrome component can be expressed as equation (2.18)

  $S_i = e(\alpha^i)$ for $1 \leq i \leq 2t$ (2.18)

If $r(x)$ is divided by the minimum polynomial $\varphi_i(x)$ of α^i:

- Since $\varphi_i(\alpha^i) = 0$, then $S_i = r(\alpha^i) = b_i(\alpha^i)$ for $1 \leq i \leq 2t$. We have:

  $r(x) = a_i(x)\varphi_i(x) + b_i(x)$ (2.19)

  where $b_i(x)$ is the remainder with degree less than that of $\varphi_i(x)$.

2.4.3 Computation of Error Locator Polynomial

Suppose the error pattern, $e(x)$ contains v errors at the locations $j_1, j_2, \cdots, j_v$, where $0 \leq j_1 < j_2 < \cdots < j_v < n$:

- Then the error polynomial, $e(x)$ is given by:

  $e(x) = x^{j_1} + x^{j_2} + \cdots + x^{j_v}$ (2.20)

- $2t$ syndrome components, $S_1, S_2, \cdots, S_{2t}$:

  $S_1 = e(\alpha) = \alpha^{j_1} + \alpha^{j_2} + \cdots + \alpha^{j_v}$,

  $S_2 = e(\alpha^2) = (\alpha^{j_1})^2 + (\alpha^{j_2})^2 + \cdots + (\alpha^{j_v})^2$,

  $\vdots$

  $S_{2t} = e(\alpha^{2t}) = (\alpha^{j_1})^{2t} + (\alpha^{j_2})^{2t} + \cdots + (\alpha^{j_v})^{2t}$ (2.21)

- For $1 \leq l \leq v$, define $\beta_l = \alpha^{j_l}$. $2t$ syndrome components can be expressed in the similar form below:

  $S_1 = \beta_1 + \beta_2 + \cdots + \beta_v$,

$$S_2 = {\beta_1}^2 + {\beta_2}^2 + \cdots + {\beta_v}^2,$$

$$\vdots$$

$$S_{2t} = {\beta_1}^{2t} + {\beta_2}^{2t} + \cdots + {\beta_v}^{2t} \quad (2.22)$$

- Define the error-location polynomial, $\sigma(x)$ of degree *v* over $GF(2^m)$ that has ${\beta_1}^{-1}, {\beta_2}^{-1}, \cdots, {\beta_v}^{-1}$ (the inverses of the location numbers $\beta_1, \beta_2, \cdots, \beta_v$) as roots:

  $\sigma(x) = (1 + \beta_1 x)(1 + \beta_2 x) \cdots (1 + \beta_v x) = \sigma_0 + \sigma_1 x + \cdots + \sigma_v x^v$ (2.23)

  where,

  $$\sigma_0 = 1,$$

  $$\sigma_1 = \beta_1 + \beta_2 + \cdots + \beta_v,$$

  $$\sigma_2 = \beta_1\beta_2 + \beta_1\beta_3 + \cdots + \beta_{v-1}\beta_v,$$

  $$\sigma_3 = \beta_1\beta_2\beta_3 + \beta_1\beta_2\beta_4 + \cdots + \beta_{v-2}\beta_{v-1}\beta_v$$

  $$\vdots$$

  $$\sigma_v = \beta_1\beta_2\beta_3 \cdots \beta_{v-2}\beta_{v-1}\beta_v.$$

- The inverses of the roots of error-location polynomial, $\sigma(x)$ give the error-location numbers.

- From equation (2.21) and equation (2.22), *2t* syndrome components, $S_1, S_2, \cdots, S_{2t}$ can be expressed in terms of the coefficients of the error-location polynomial, $\sigma_0, \sigma_1, \cdots, \sigma_v$:

  $$S_1 + \sigma_1 = 0,$$

  $$S_2 + \sigma_1 S_1 + 2\sigma_2 = 0,$$

  $$S_3 + \sigma_1 S_2 + \sigma_2 S_1 + 3\sigma_3 = 0,$$

  $$\vdots$$

  $$S_v + \sigma_1 S_{v-1} + \sigma_2 S_{v-2} + \cdots \sigma_{v-1} S_1 + v\sigma_v = 0,$$

$$S_{v+1} + \sigma_1 S_v + \sigma_2 S_{v-1} + \cdots \sigma_{v-1} S_2 + \sigma_v S_1 = 0, \quad (2.24)$$

- Above identities is called Newton identities.

In general there will be more than one error pattern for which the coefficients of its error-location polynomial satisfy the Newton identities. To minimize the probability of a decoding error, the most probable error pattern for error correction need to be found. Finding the most probable error pattern means determining the error-location polynomial of minimum degree whose coefficients satisfy the Newton identities. This can be achieved iteratively by Berlekamp–Massey (BM) algorithm.

2.4.4 Berlekamp-Massey Algorithm (BMA)

Berlekamp-Massey algorithm [19] [20] is an algorithm that will be used in BCH decoder to find the error-location polynomial, $\sigma(x)$ iteratively in *2t* steps:

- For $1 \leq k \leq 2t$, the algorithm at the k-th step gives an error-location polynomial of minimum degree as below:

  $$\sigma^{(k)}(x) = {\sigma_0}^{(k)} + {\sigma_1}^{(k)} x + \cdots + {\sigma_{l_k}}^{(k)} x^{l_k} \quad (2.25)$$

  where coefficients satisfy the first k Newton identities.

- *(k+1)*th step error-location polynomial, $\sigma^{(k+1)}(x)$ is given by:

  $$\sigma^{(k+1)}(x) = \sigma^{(k)}(x) + d_k {d_i}^{-1} x^{k-i} \sigma^{(i)}(x) \quad (2.26)$$

  where

  $d_k {d_i}^{-1} x^{k-i} \sigma^{(i)}(x)$ is the correction term

  d_k is the *k*th discrepancy $d_k = S_{k+1} + {\sigma_1}^{(k)} S_k + {\sigma_2}^{(k)} S_{k-1} + \cdots + {\sigma_{l_k}}^{(k)} S_{k+1-l_k}$

d_i^{-1} is inverse of ith discrepancy

i is the step prior to k which is $\sigma^{(i)}(x)$ such that the ith discrepancy, $d_i \neq 0$ and $i - l_i$ has the largest value.

l_i is the degree of $\sigma^{(i)}(x)$

- Steps of using BM algorithm for finding the Error-Location Polynomial of a BCH Code:
  - Initialization:
    - For $k = -1$, set $\sigma^{(-1)}(X) = 1, d_{-1} = 1, l_{-1} = 0$ and $-1 - l_{-1} = -1$.
    - For $k = 0$, set $\sigma^{(0)}(X) = 1, d_0 = S_1, l_0 = 0$ and $0 - l_0 = 0$.
  - Step 1: If $k = 2t$, output $\sigma^{(k)}(X)$ as the error-location polynomial $\sigma(x)$; otherwise go to Step 2.
  - Step 2: Compute d_k and go to Step 3.
  - Step 3: If $d_k = 0$, set $\sigma^{(k+1)}(X) = \sigma^{(k)}(X)$; otherwise, set $\sigma^{(k+1)}(X) = \sigma^{(k)}(X) + d_k d_i^{-1} x^{k-i} \sigma^{(i)}(X)$. Go to Step 4.
  - Step 4: $k \leftarrow k + 1$. Go to Step 1.
- The BM algorithm can be executed by setting up and filling in the following table 2.4:

Table 2-4 BMA execution table

Step k	Partial solution $\sigma^{(k)}(X)$	Discrepancy d_k	Degree l_k	Step/degree difference $k - l_k$
-1	1	1	0	– 1
0	1	S_1	0	0
1	$\sigma^{(1)}(X)$	d_1	l_1	$1 - l_1$
2	$\sigma^{(2)}(X)$	d_2	l_2	$2 - l_2$
⋮				
$2t$	$\sigma^{(2t)}(X)$	-----	-----	-----

Based on the above BM algorithm, an interesting pattern *k*-th step solution will be observed. The solution $\sigma^{(2k-1)}(x)$ at the *(2k–1)*th step of the BMA is also the solution $\sigma^{(2k)}(x)$ at the *2k*-th step of the BMA:

$$\sigma^{(2k)}(x) = \sigma^{(2k-1)}(x), \text{ for } 1 \leq k \leq t \tag{2.27}$$

Consequently, for decoding a binary BCH code, the BM algorithm can be simplified as follows:

- Steps of using Simplified BM algorithm for finding the Error-Location Polynomial of a BCH Code:
  - Initialization:
    - For $k = -1/2$, set $\sigma^{(-1/2)}(X) = 1, d_{-1/2} = 1,\ l_{-1/2} = 0$ and $-2(1/2) - l_{-1/2} = -1$.
    - For $k = 0$, set $\sigma^{(0)}(X) = 1, d_0 = S_1,\ l_0 = 0$ and $0 - l_0 = 0$.
  - Step 1: If $k = t$, output $\sigma^{(k)}(X)$ as the error-location polynomial $\sigma(x)$; otherwise go to Step 2.
  - Step 2: Compute $d_k = S_{2k+1} + {\sigma_1}^{(k)} S_{2k} + {\sigma_2}^{(k)} S_{2k-1} + \cdots + {\sigma_{l_k}}^{(k)} S_{2k+1-l_k}$ and go to Step 3.

- Step 3: If $d_k = 0$, set $\sigma^{(k+1)}(X) = \sigma^{(k)}(X)$; otherwise, set $\sigma^{(k+1)}(X) = \sigma^{(k)}(X) + d_k d_i^{-1} X^{2(k-i)} \sigma^{(i)}(X)$. Go to Step 4.
- Step 4: $k \leftarrow k + 1$. Go to Step 1.

- The simplified BM algorithm can be executed by setting up and filling in the following table 2-5:

Table 2-5 Simplified BMA execution table

Step k	Partial solution $\sigma^{(k)}(X)$	Discrepancy d_k	Degree l_k	Step/degree difference $2k - l_k$
– ½	1	1	0	– 1
0	1	S_1	0	0
1	$\sigma^{(1)}(X)$	d_1	l_1	$2 - l_1$
2	$\sigma^{(2)}(X)$	d_2	l_2	$4 - l_2$
⋮				
t	$\sigma^{(t)}(X)$	-----	-----	-----

It can be noticed that from either the conventional or simplified BMA, the evaluation of the correction term in each iteration required GF inverter. However, designing a GF inverter and running it at each iteration consume extra logic and impose additional delay in the calculation. Therefore, the inversion-less BMA [21] was introduced and several improvements [11] [14] [15] were proposed by researchers to eliminate the GF inverter that relax the complexity of the BMA design.

2.4.5 Chien Search (CS)

After the error location polynomial is obtained, the error locations are found by finding the all the roots from the error location polynomial. The error location is the power of alpha from each of the roots found.

Consider error location polynomial, $\sigma(x)$ in equation (2.28).

$$\sigma(x) = \sigma_0 + \sigma_1 x + \sigma_2 x^2 + \cdots + \sigma_v x^v \tag{2.28}$$

One method to find the roots is evaluating $\sigma(x)$ with each non-zero element in $GF(2^m)$, $(1, \alpha, \alpha^2, \alpha^3, \ldots, \alpha^{2^m-2})$. However, this will require a lot of variable multiplication and addition.

The Chien Search algorithm observed that:

- Let $\lambda_{j,i} = \sigma_j(\alpha^i)^j$, then

$$\sigma(\alpha^i) = \sigma_0 + \sigma_1\alpha^i + \sigma_2(\alpha^i)^2 + \cdots + \sigma_v(\alpha^i)^t \tag{2.29}$$

$$\sigma(\alpha^i) = \lambda_{0,i} + \lambda_{1,i} + \lambda_{2,i} + \cdots + \lambda_{t,i} \tag{2.30}$$

$$\sigma(\alpha^{i+1}) = \sigma_0 + \sigma_1\alpha^{i+1} + \sigma_2(\alpha^{i+1})^2 + \cdots + \sigma_v(\alpha^{i+1})^t \tag{2.31}$$

$$\sigma(\alpha^{i+1}) = \sigma_0 + \sigma_1(\alpha^i)\alpha^1 + \sigma_2(\alpha^i)^2\alpha^2 + \cdots + \sigma_v(\alpha^i)^t\alpha^t \tag{2.32}$$

$$\sigma(\alpha^{i+1}) = \lambda_{0,i} + \lambda_{1,i}\alpha^1 + \lambda_{2,i}\alpha^2 + \cdots + \lambda_{t,i}\alpha^t \tag{2.33}$$

- From equation (2.29), (2.30), (2.31), (2.32) and (2.33), it can be observed that:

$$\lambda_{j,i+1} = \lambda_{j,i}\alpha^j \tag{2.34}$$

If $\sum_{j=0}^{t} \lambda_{j,i} = 0$, then α^i is a root. For each of the α^i, i will be the bit location of the received codeword that contains error.