NON-BINARY CYCLIC CODES
AND ITS APPLICATIONS IN DECODING OF
HIGH DIMENSIONAL TRELLIS-CODED MODULATION

A Thesis Presented to
The Faculty of the
School of Electrical Engineering and Computer Science
Fritz J. and Dolores H. Russ
College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Biyun Zhou
March, 2000
First, I would like to express my deepest thanks to my advisor, Dr. Jeffrey C. Dill for his guidance and instruction on this research project. His creative thinking and insightful directions made me follow a correct way in research. I also appreciated his effort to arrange a cooperative group meeting to provide an encouraging research atmosphere.

I would also thank all the thesis committee members, Dr. Joseph Essman, Dr. Michael Braasch, and Dr. Sergio Lopez-Permouth for their helpful insights. I especially thank Dr. Sergio Lopez-Permouth for his instruction on the coding course, which guide me into this error control coding filed. His continues support on mathematics and patience in revising the initial version of this thesis is greatly appreciated.

I would like to thank other group members in this research project, Xiaoxiao Cui, Sally Song and Yun yan for their help and discussions on the research. I thank Yung-Cheng Lo, former Ph.D. student who initiates this HDTCM project for his introduction and help when I first joined the group.

I thank my family. Thanks to my father, a great telecommunications engineer for his guidance and encouragement in my academic advancement; and his always affection and support. I thank my mother for her sacrifice for the family and her love and anticipation. I thank my big brother and my sister-in-law for their tasty food each time I visit them. I especially thank my husband, Sen, who gave me love, support and encouragement all the time.
# TABLE OF CONTENTS

ACKNOWLEDGEMENT ................................................................. iii

TABLE OF CONTENTS ................................................................ iv

LIST OF FIGURES .................................................................. vi

LIST OF TABLES ...................................................................... vii

CHAPTER

1. INTRODUCTION .................................................................... 1
   1.1 History of Error-Control Coding ........................................... 1
   1.2 Contribution of this Research .............................................. 3
   1.3 Outline of the Thesis .......................................................... 4

2. BACKGROUND ......................................................................... 5
   2.1 Mathematical Background ................................................... 5
      2.1.1 Galois Fields ............................................................... 5
      2.1.2 Polynomials over Galois Field ...................................... 7
      2.1.3 Factoring $x^n - 1$ ....................................................... 11
   2.2 Communication System and Channel Coding ....................... 13
      2.2.1 Coding and Modulation .............................................. 14
      2.2.2 Error Correction Codes ............................................. 14
      2.2.3 Trellis-Coded Modulation .......................................... 15
   2.3 High Dimensional Trellis-Coded Modulation ....................... 18

3. LINEAR BLOCK CODES, CYCLIC CODES AND BCH CODES .... 20
   3.1 Linear Block Code ............................................................ 20
   3.2 Cyclic Codes .................................................................... 23
   3.3 BCH codes ..................................................................... 24
      3.3.1 Encoding of BCH codes ........................................... 24
      3.3.2 Erasures in BCH code .............................................. 28
   3.4 Reed-Solomon Codes ........................................................ 29
   3.5 Simulation Results ............................................................ 31

4. DECODING OF NON-BINARY BCH CODES ......................... 41
   4.1 Overview of Decoding Algorithm for Non-binary BCH code ..... 41
   4.2 Syndrome-Based Decoding and Berlekamp's Algorithm ........... 42
4.2.1 Standard Decoding Algorithm ................................................................. 43
4.2.2 Berlekamp-Massey Algorithm ................................................................. 46
4.2.3 Erasure Decoding ................................................................................... 48
4.2.4 Simulation Consideration ...................................................................... 50
4.3 Application of Non-Binary BCH Code in HDTCM .................................. 53

5. CONCLUSIONS ............................................................................................ 57
5.1 Summary .................................................................................................. 57
5.2 Future Research ....................................................................................... 59

References ....................................................................................................... 60

Appendix
A List of some lengths 4-ary BCH codes ..................................................... 62
B Generating generator polynomial for non-binary BCH codes ................. 68
C Decoding of BCH Codes .......................................................................... 73
D Main Testing Function ............................................................................. 78
LIST OF FIGURES

Figure 2.1: Block diagram for communication system ...........................................13
Figure 2.2: Rate ½ convolutional encoder with K=2 .............................................15
Figure 2.3 Comparison between conventional coding/modulation and TCM........17
Figure 3.1: Decoding spheres .............................................................................21
Figure 3.2: Sub-optimum BCH code selection algorithm ..............................28
Figure 3.3 Comparison of 4-ary, n=255 primitive BCH codes and n=255 Reed-Solomon codes .................................................................................40
Figure 4.1 Standard concatenated coding system .............................................55
Figure 4.2 Improved concatenated coding system .............................................56
LIST OF TABLES

Table 2.1 Construction of $GF(8)$ .................................................................9
Table 2.2 Conjugacy class of $GF(8)$ .............................................................10
Table 2.3: Cyclotomic cosets modulo 31 with respect to $GF(4)$ ......................11
Table 2.4 Factorization of $x^{31} - 1$ ................................................................13
Table 3.1 Cyclotomic cosets and minimal polynomials for 4-ary BCH code of length 21 ........................................................................................................26
Table 3.2 List of 21 length 4-ary BCH codes....................................................31
Table 3.3 Acceptable 4-ary BCH code of lengths 31, 33, 35 and 39 ...................32
Table 3.5 Comparison of variable length 4-ary BCH codes..............................35
Table 3.6 cyclotomic cosets and minimal polynomials for primitive 4-ary BCH code ....37
Table 3.7 cyclotomic cosets modulo 65 with respect to $GF(4)$ .......................38
Table 3.8 list of 4-ary length 63 BCH codes ....................................................39
CHAPTER 1
INTRODUCTION

Research on error-control coding in the past fifty years has lead to the renovation of modern communications systems. We start this thesis by outlining a brief history of error-control coding in digital communication systems. We will introduce several technologies such as trellis-coded modulation, high dimensional trellis-coded modulation, and BCH codes. The primary contributions of this thesis research will also be explained in this chapter. We conclude the chapter with an outline of the thesis.

1.1 History of Error-Control Coding

Concepts of error control coding can be traced back to Shannon’s theorem, which states that channel capacity is related to bandwidth as well as the signal to noise ratio via the equation

\[ C = B \log_2(1 + \frac{S}{N}) \]

From this equation, one can infer that theoretically there exist perfect channels with infinite capacity. However such channels do not exist in real life since they would require storing information before sending out. The practical challenge, then, is to improve the performance of available systems through strategies such as error control coding. Error-correcting codes are systematic ways to introduce redundancy into messages in order to allow the detection and or correction of transmission errors that may have occurred due to noisy channels. Error control coding has been deemed one of the major developments in
communications science. There exist two natural families of error-control codes; the block codes and the convolutional codes. Convolutional codes essentially handle the information as a stream with no interruption; they are implemented as specific connections of shift registers. On the other hand, block codes divide the source data into finite-length blocks, each of which are mapped uniquely to “encoded” blocks for transmission. The encoded messages include check symbols, which increase the block size and result in the expansion of the bandwidth required transmitting the signal if the original source data is at the constant rate.

Error control coding has been widely used in various applications such as computer memory design and audio/video coding. However, band-limited channels could not benefit from the coding gain until the introduction of trellis-coded modulation (TCM), proposed by Ungerboeck in 1982. TCM combines coding and modulation into a single step by integrating a multi-level or multi-phase signaling constellation [1][2][6]. By choosing a state-oriented encoder such as a convolutional encoder, the minimum distance between the legal code words becomes the distance between any two possible trellis paths, which is higher than the traditional Hamming distance in block codes if good symbol assignment rules are applied. The tradeoff for TCM is that the coding gain is achieved at the expense of decoding complexity. The design of high-speed telephone data modems is the most prominent application of TCM [6].

High dimensional trellis-coded modulation (HDTCM) was proposed as a way of using the high dimensional signaling constellation and the trellis structure on the block code. The direct goal of HDTCM is to achieve coding gain on a power-limited channel, such as spread-spectrum channels.
In order to encode the source information block by block on trellis structure, the trellis path must be forced to go into a certain known state at the ending of the block. This feature was employed in HDTCM by satisfying a so-called “state constraint.” The state constraint in HDTCM means that the starting state of a particular source data block must equal the ending state. This new circular trellis structure is one of the major contributions of HDTCM. Interested readers can refer to [3][10][11] for description and method to generate such structure.

To decode the HDTCM system, the concatenated decoding algorithm was proposed that appropriate block code must be used in combination with the Viterbi decoder. Since the HDTCM trellis normally employs a non-binary source alphabet size and various number of states, the job of finding the proper block code that can fit to HDTCM state structure becomes necessary.

1.2 Contribution of this Research

Following earlier research approach of using a “simplex” signaling constellation on HDTCM, we follow the source symbol alphabet size of four, and we aim to develop rules on finding the best performance non-binary BCH codes with emphasis on 4-ary BCH codes. A standard decoding algorithm is also implemented as a simulation and modified to accommodate the non-narrow-sense codes. Matlab functions for encoding and decoding of Non-binary BCH codes are included.

Various testing results on 4-ary BCH codes within certain length range are also presented. Comparison on the different length 4-ary BCH codes are included to show the existence of certain good codes having larger dimension than other codes with the same
decoding capability. In general, this thesis research present various 4-ary BCH codes with code dimension as well as error correction capability, which will aid the future work of selecting proper codes for concatenated decoding systems. Moreover, the simulation of encoding and decoding algorithms can be directly used in decoding system.

1.3 Outline of the Thesis

The structure of this thesis is outlined next:

Chapter 2 covers general communications background. The focus is placed on the Galois fields and polynomials over them. They are the fundamentals for block code. Trellis coded modulation (TCM) and HDTCM system introduction are presented last.

Chapter 3 explains how the BCH code was developed on the basis of Galois fields and polynomials. The encoding algorithm for BCH code is explained in detail and we tested various BCH codes of given lengths. Chapter 3 ends with an analysis of those experimented results.

Chapter 4 describes the decoding of non-binary BCH code. Modifications of the Berlekamp-Masey algorithm is used. Erasure decoding for BCH codes is explained with the modified decoding algorithm. The last part of this chapter is the application of non-binary BCH codes in the decoding scheme for HDTCM.

Chapter 5 consists of the conclusions and description of future work.

Appendices: Matlab source files for encoding and decoding of non-binary BCH codes. And the complete testing results of non-binary BCH codes; the code dimension, error correction capability and the generating polynomials for codes with various lengths are included.
CHAPTER 2
BACKGROUND

This chapter provides some mathematical background to help the reader to better understand the topics stated in the thesis. We start with the fundamentals of algebraic coding theory and digital communication systems. Conventional TCM and High Dimensional Trellis-Coded Modulation (HDTLM) and their applications are also surveyed.

2.1 Mathematical Background

The development of coding theory has been achieved with advancement in powerful mathematical concepts. The mathematical prerequisite to understand BCH codes include various mathematical concepts [5][13] such as rings, Galois fields, polynomials over Galois fields and vector spaces.

2.1.1 Galois Fields

Roughly speaking, a field is a set where one can add, subtract, multiply and divide (by on-zero element). When a field is finite, its cardinality $q$ is a power of a prime, denoted by $q = p^m$. There is essentially only one field with cardinality $q = p^m$; we refer it as the Galois field $GF(q)$. $GF(p^m)$ forms the extension field of $GF(p)$ with the equation of $GF(p) \subseteq GF(p^m)$. If we let $\beta$ be an element in this field and let 1 be the multiplicative identity, the following sequence of elements must also be in $GF(q)$ by closure under multiplication.
1, $\beta^1, \beta^2, \beta^3, \beta^4, \cdots$

The sequence begins to repeat at some stage and the first element to repeat must be a fortiori 1. The simplest example of a field is $GF(2) = \{0,1\}$. Here, some useful definitions are introduced below to aid further discussion.

Definition 2-1: Let $\beta$ be a non-zero element in $GF(q)$. The order of $\beta$ (written $\text{ord} (\beta)$) is defined as the smallest positive integer $m$ such that $\beta^m = 1$.

Definition 2-2: An element with order $(q-1)$ in $GF(q)$ is called a primitive element in $GF(q)$.

The importance of primitive elements is that their powers span all of $GF(q)^* = GF(q) - \{0\}$. That is, except 0, all elements in a Galois field can be represented as certain power of its primitive element.

Definition 2-3: The characteristic of a Galois field $GF(q)$ is the smallest positive integer $m$ such that summation of $m$ ones equals zero.

As mentioned before, there exists a prime $p$ such that $q = p^m$. When this is the case, $p$ is the characteristic of $GF(q)$. The following theorems are derived from the above definitions and are well known to describe the Galois fields.

Theorem 2-1: If $t = \text{ord}(\beta)$ for some $\beta \in GF(q)$, then $t$ must divide $q-1$ (written $t \mid (q-1)$).

Based on the above theorems and definitions, we can present an example of $GF(8)$. $\beta$ is defined as the primitive element of $GF(8)$ with order $2^3 - 1 = 7$. Then $GF(8) = \{0,1,\beta, \beta^2, \beta^3, \beta^4, \beta^5, \beta^6\}$. It can be seen that $\beta^7 = 1$ is the first element to
repeat. Also \( GF(2) = \{0,1\} \subseteq GF(8) \) since \( GF(8) \) is the extension field of \( GF(2) \). The construction of a Galois field will involve the polynomial manipulation, which will be explained later. The multiplication in Galois fields is then just the adding of exponent \( \mod(q - 1) \), whereas the addition operation would force a polynomial representation of the elements. On the other hand, Zech’s logarithms, also known as “add-one tables”, are frequently used as a memory efficient approach for addition in Galois fields.

2.1.2 Polynomials over Galois Field

The collection of all polynomials of arbitrary degree with coefficients from the finite field \( GF(q) \) is denoted by \( GF(q)[x] \). The addition and multiplication are performed in the usual way using the operations for the field from which the coefficients were taken. Analog to the concept of extension field in Galois fields, polynomials over the extension field follows the same rule. That is \( GF(p)[x] \subseteq GF(p^n)[x] \) if \( GF(p) \subseteq GF(p^n) \).

Definition 2-4: A polynomial \( p(x) \) is irreducible in \( GF(p) \) if it can not be factored into the product of polynomials of lower degree in \( GF(p)[x] \).

Definition 2-5: An irreducible polynomial \( p(x) \in GF(p)[x] \) of degree \( m \) is said to be primitive if the smallest positive integer \( n \) for which \( p(x) \) divides \( x^n - 1 \) is \( n = p^m - 1 \).

Theorem: 2-4: The roots of a \( m \)th-degree primitive polynomial \( p(x) \in GF(p)[x] \) all have the order of \( p^m - 1 \).
Notice that the concept of irreducibility is dependent on the context field. Namely, given fields $F$ and $E$ with $F \subseteq E$, it is possible that a polynomial $f(x) \in F(x)$ is irreducible in $F(x)$ but, when $f(x)$ is considered as an element of $E(x)$, $f(x)$ is not irreducible over $E(x)$. And a primitive polynomial $p(x) \in GF(p)[x]$ is always irreducible in $GF(p)[x]$, but irreducible polynomials are not always primitive. Now we can see how Galois fields can be constructed using primitive polynomials. Let $p(x)$ be a primitive polynomial of degree $m$ in $GF(p)[x]$ and $\partial$ be a root of $p(x)$. Then $\partial$ must satisfy the expression:

$$\partial^m + a_{m-1}\partial^{m-1} + \ldots + a_1\partial + a_0 = 0$$

The individual powers of $\partial$ of degree greater than or equal to $m$ can therefore be expressed as polynomial expressions in $\partial$ of degree $(m-1)$ or less. Since $\partial$ has order $p^m - 1$, the distinct powers of $\partial$ must have $p^m - 1$ distinct nonzero polynomial representations of degree $m$ or less. And these $p^m - 1$ consecutive powers of $\partial$ with their polynomial representations consequently exchange the nonzero elements of the field $GF(p^m)$. As an illustration, when $q = p$ (i.e., when $m = 1$), $GF(q) = \mathbb{Z}/(p) = \{0, 1, \ldots, p-1\}$ with the usual modular arithmetic. For the case when $m > 1$, an analogous quotient construction gives $GF(q)$. Namely, substitute $Z$ with $GF(p)[x]$ (polynomials with coefficients in $\mathbb{Z}/(p)$) and “mod out” (multiplies of) $f(x)$, an irreducible polynomial in $GF(p)[x]$ with degree $m$. So, $GF(q) = \mathbb{Z}/(p)[x]/(f(x))$. If $f(x)$ is chosen carefully, the equivalent solution of $x$ in $GF(q)$ is a primitive element. Such
polynomials always exist and are referred to as “primitive polynomials”. (See definition 2-5). As an example, consider the construction of $GF(8)$ as in table 2.1. Suppose

$$GF(8) = \frac{\mathbb{Z}_2[x]}{x^3 + x + 1}$$

Notice that if $\partial = x \in GF(8)$, expression $\partial^3 = \partial + 1$ exist.

Table 2.1 Construction of $GF(8)$

<table>
<thead>
<tr>
<th>Exponential Representation</th>
<th>Polynomial Representation</th>
<th>Vector Space over $GF(2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\partial^{-\text{inf}}$</td>
<td>0</td>
<td>(0,0,0)</td>
</tr>
<tr>
<td>$\partial^0$</td>
<td>1</td>
<td>(1,0,0)</td>
</tr>
<tr>
<td>$\partial^1$</td>
<td>$\partial$</td>
<td>(0,1,0)</td>
</tr>
<tr>
<td>$\partial^2$</td>
<td>$\partial^2$</td>
<td>(0,0,1)</td>
</tr>
<tr>
<td>$\partial^3$</td>
<td>$\partial + 1$</td>
<td>(1,1,0)</td>
</tr>
<tr>
<td>$\partial^4$</td>
<td>$\partial^2 + \partial$</td>
<td>(0,1,1)</td>
</tr>
<tr>
<td>$\partial^5$</td>
<td>$\partial^4 \partial = \partial^3 + \partial^2 = \partial^2 + \partial + 1$</td>
<td>(1,1,1)</td>
</tr>
<tr>
<td>$\partial^6$</td>
<td>$\partial^3 + 1$</td>
<td>(1,0,1)</td>
</tr>
</tbody>
</table>

Since $\partial^7 = \partial^6 \partial = (\partial^2 + 1)\partial = \partial^3 + \partial = 1$, $\partial$ is indeed a primitive element. The polynomial representation of a finite field $GF(p^m)$ has coefficients from the sub-field $GF(p)$ and can be viewed as a construction over $GF(p)$. Notice that $GF(p^m)$ may contain sub-fields other than the base prime-order field $GF(p)$. As an example, $GF(64)$ has $GF(8)$, $GF(4)$, and $GF(2)$ as sub-fields. Let $\alpha$ be a primitive element in $GF(64)$, an element $\alpha^j$ in $GF(64)$ is also in the sub-field $GF(p)$ if and only if $j \cdot q \equiv j \mod 63$. So $GF(4) = \{0,1,\alpha^{21},\alpha^{42}\}$, $GF(2) = \{0,1\}$ and $GF(2) \subseteq GF(4)$. 
Definition 2-6: if $\mathfrak{d}$ is an element in the field $GF(p^m)$, the Minimal Polynomial $p(x)$ of $\mathfrak{d}$ over $GF(p)$ is the smallest degree nonzero mono polynomial with coefficient in $GF(p)$ that has $\mathfrak{d}$ as a root.

Minimal polynomials are unique. Also, if $f(\beta) = 0$, $f(x)$ must be a multiple of $p(x)$.

Other properties of minimal polynomial $p(x)$ include:

- The degree of $p(x)$ is less than or equal to $m$.
- $p(x)$ is irreducible in $GF(p)[x]$.

To better understand why the concept of a minimal polynomial is useful in code design, we introduce the following concept.

Definition 2-7: The Conjugates of a Field Elements $\beta$ in $GF(p^m)$ with respect to the sub-field $GF(p)$ are the elements $\beta, \beta^p, \beta^{p^2}, \ldots$.

It can be shown that the roots of minimal polynomial of $\beta$ with respect to $GF(p)$ are the conjugates of $\beta$. Continue the last example, the eight elements in $GF(8)$ are constructed by using the primitive polynomial $f(x) = x^3 + x + 1$ with its root $\beta$ can be arranged in conjugacy classes as showed in table 2.2:

<table>
<thead>
<tr>
<th>Conjugacy class</th>
<th>Associated Minimal Polynomial</th>
</tr>
</thead>
<tbody>
<tr>
<td>${0}$</td>
<td>$M_{\text{inf}}(x) = (x - 0) = x$</td>
</tr>
<tr>
<td>${ \beta^0 }$</td>
<td>$M_0(x) = (x - 1) = x + 1$</td>
</tr>
<tr>
<td>${ \beta, \beta^2, \beta^4 }$</td>
<td>$M_1(x) = (x - \beta)(x - \beta^2)(x - \beta^4) = x^3 + x + 1$</td>
</tr>
<tr>
<td>${ \beta^3, \beta^6, \beta^5 }$</td>
<td>$M_3(x) = (x - \beta^3)(x - \beta^6)(x - \beta^5) = x^3 + x^2 + 1$</td>
</tr>
</tbody>
</table>
Cyclotomic Cosets:

Cyclotomic cosets constitute a simple representation of the conjugacy class. The exponents of the $n$ distinct powers of a primitive $n$th root of unity with respect to $GF(q)$ are the elements for the cyclotomic cosets. We define the cyclotomic cosets $mod \ n$ with respect to $GF(p)$ as a partitioning of the set of integers $\{0, 1, K, n-1\}$ into sets of the form

$$\{\alpha, \alpha q, \alpha q^2, \alpha q^3, K, \alpha q^{d-1}\}$$

Cyclotomic cosets provide a simple form to describe and indeed calculate the complete factorization of polynomials of the form $x^n - 1$ into irreducible factors [5]. Table 2.3 lists out the cyclotomic cosets modulo 31 with respect to $GF(4)$.

Table 2.3: Cyclotomic cosets modulo 31 with respect to $GF(4)$

<table>
<thead>
<tr>
<th>Cosets</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0}</td>
</tr>
<tr>
<td>{1, 4, 16, 2, 8}</td>
</tr>
<tr>
<td>{3, 12, 17, 6, 24}</td>
</tr>
<tr>
<td>{5, 20, 18, 10, 9}</td>
</tr>
<tr>
<td>{7, 28, 19, 14, 25}</td>
</tr>
<tr>
<td>{11, 13, 21, 22, 26}</td>
</tr>
<tr>
<td>{15, 29, 23, 30, 27}</td>
</tr>
</tbody>
</table>

2.1.3 Factoring $x^n - 1$

Definition 2-8: Elements of order $n$ are called primitive $n$th root of unity.

It's obvious that the set of all nonzero elements in $GF(p^n)$ form the complete set of roots of the expression $x^{(p^n-1)} - 1 = 0$. Thus the minimal polynomials of the nonzero
elements in a given field provide a complete factorization of $x^{(p^m-1)} - 1$. A more general case is to factor polynomial of the form $x^n - 1$ (i.e., $n \neq p^m$). Assume the existence of an element $\beta$ with order in some field $GF(q^m)$. Elements $1, \beta, \beta^2, \ldots, \beta^{n-1}$ must be distinct and satisfy $x^n - 1 = 0$. Thus $\beta$ is a $n$th roots of unity and the $n$ roots of $x^n - 1$ are generated by computing $n$ consecutive powers of $\beta$. Next, we need to identify the field where we can find all these roots. A proven theorem states that $GF(q^m)$ is the smallest extension field of $GF(q)$ where the primitive $n$th roots of unity can be found if $m$ is the smallest integer such that $n$ divides $(q^m - 1)$. Thus the factorization of $x^n - 1$ needs to obtain the cyclotomic cosets modulo $n$ with respect to $GF(q)$ along with the minimal polynomials that have those conjugacy elements as roots. The following example shows the ideal of factoring $x^n - 1$.

*Example:* factoring $x^{21} - 1$ in $GF(4)[x]$

Now, with $q = 4$, the primitive 21st roots of unity can be found in $GF(4^3)$ because the smallest $m$ such that 4 divides $(4^m - 1)$ is just 3. The primitive element in $GF(64)$ is $\beta$, thus the primitive 21-th root of unity in $GF(64)$ is $\alpha = \beta^3$ since $\beta^3$ has order 21. The first step is to list the cyclotomic cosets modulo 21 with respect to $GF(4)$, then the minimal polynomials can be found as shown in table 2.4 which is the complete factorization of $x^{21} - 1$ in $GF(4)[x]$. 
Table 2.4 Factorization of $x^{21} - 1$

<table>
<thead>
<tr>
<th>$M_0(x)$</th>
<th>$M_4(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x + 1$</td>
<td>$x^3 + \alpha^{42}x^2 + 1$</td>
</tr>
<tr>
<td>$M_1(x) = x^3 + \alpha^{42}x + 1$</td>
<td>$M_5(x) = x + \alpha^{21}$</td>
</tr>
<tr>
<td>$M_2(x) = x^3 + \alpha^{21}x + 1$</td>
<td>$M_6(x) = x^3 + x + 1$</td>
</tr>
<tr>
<td>$M_3(x) = x^3 + x^2 + 1$</td>
<td>$M_{10}(x) = x^3 + \alpha^{21}x^2 + 1$</td>
</tr>
<tr>
<td>$M_{14}(x) = x + \alpha^{42}$</td>
<td></td>
</tr>
</tbody>
</table>

We will further explore the polynomials over Galois field after reviewing the background of communication system and error correction codes.

2.2 Communication System and Channel Coding

A communication system can be simplified as a model with just three parts namely transmitter, channel and receiver. The “channels” is an abstraction of various physical devices and media such as electromagnetic wave and optic fiber to transmit information. All the channels are noisy, that’s why a channel encoder is used to combat channel noise at the price of adding redundancy bits. The role of modulator is to match the encoder output, which is the baseband signal in digital communications, to the transmission channel [1]. The following figure 2.1 is the diagram for a simple communication system model with subdivided transmitter.

![Block diagram for communication system](image-url)

Figure 2.1: Block diagram for communication system
2.2.1 Coding and Modulation

Coding and modulation were implemented as two separate processes in conventional communication systems. There are various types of modulation schemes available that can deal with binary or $M$-ary encoded symbols and have different bit error performances. For a general case of $M$-ary signaling, the trade-off between error probability and bandwidth performance is that as $M$ is increased, we can either achieve improved bandwidth performance at the expense of error performance, or we can tolerate increased $Eb/N0$ (signal to noise ratio) without error performance degradation. Coding can achieve gain in $Eb/N0$ by adding the controlled redundancy, thus decreasing the bandwidth [1]. Considered together, trade-off between bandwidth and $Eb/N0$ with fixed error performance work for most modulation and coding schemes except for cases of combined modulation and coding, of which TCM (Trellis Coded Modulation) will be concentrated on.

2.2.2 Error Correction Codes

The code rate of error correction codes is defined as $k/n$, in which $k$ is the number of information symbol transmitted per code word of length $n$; thus the redundancy is $n-k$ symbols. The redundancy is called parity check symbols and is used to correct errors, improve the reliability of the demodulated data. Compared to an uncoded system, coded system can reduce the signal power needed to transmit the information for the fixed transfer rate. And this difference in power is called the coding gain. We usually categorize the error correction codes into two groups: block codes and convolutional codes.
Since BCH code is a major kind of block code, the discussion of block code will be covered later in detail. And convolutional codes are treated as part of TCM, which is the basis for the thesis topic.

2.2.3 Trellis-Coded Modulation

Convolutional codes is actually linear Trellis coding. It’s necessary to introduce it here before we talk about TCM. The basic structure of the convolutional encoder is shift registers with length $K$ and $k$ parallel input bits or $k$-tuple stages. It produces $n$ output bits for each of $k$ input bits; thus has a code rate of $k/n$. The encoder is essentially a finite-state machine with its output symbol determined by current inputs, as well as the state of the encoder [12]. For this reason, we can use state diagram, tree diagram, or trellis diagrams to analysis the convolutional encoder, of which the trellis diagram is the mostly used one.

![Trellis Diagram](image)

Figure 2.2: Rate $\frac{1}{2}$ convolutional encoder with $K=2$

Figure 2.2 shows a rate $\frac{1}{2}$ convolutional encoder with $K=2$. There are only four possible states for the length 2 shift register, which is 00, 01, 10, and 11. In trellis diagram, for a specific sequence of input bits, there is one to one mapping of trellis path.
available. Moreover, the number of paths for all possible input sequence is less than the number of all possible trellis paths on the diagram, which means illegal paths exist.

Trellis-coded modulation combines trellis coding and modulation as one entity to achieve coding gain without bandwidth expansion. It sounds impossible at first because conventional coding and modulation schemes always have trade-off between error performance and bandwidth. Actually, the difference between conventional coding & modulation scheme and TCM is that the distance between adjacent symbol points in TCM is Euclidean distance rather than Hamming distance in the other case. In TCM, for an increased signal set, bandwidth can be retained at certain level while the reduced distance between adjacent symbol points no longer determines error performance (or degrade error performance in conventional case). Instead, the minimum distance between trellis paths of the set of all allowable code symbol sequence determines the error performance. It is inferred [1] [4] that the assignment of signal points to the coded bits in a way to maximize Euclidean distance will optimize TCM.

Consider the following as a simple example to demonstrate the difference between TCM and conventional coding/modulation scheme (Figure 2.3). In the first case, we use rate ½ convolutional code and BPSK modulation; bandwidth will double the uncoded BPSK modulation transmission by the redundancy bits. If we combined rate ½ convolutional code with the QPSK modulation, one signal will still carry one information bit, which means the bandwidth stays the same with the uncoded BPSK scheme. Thus we can see that TCM achieves coding gain without bandwidth expansion.
Mapping between coded bits and signal must obey some rules to ensure the minimum distance between all possible trellis paths to be maximized. Undebock investigated the mapping through set partitioning, which becomes a big contribution [2] to the TCM.

Soft-decision decoding is selected to decode TCM, of which the Viterbi algorithm, a kind of maximum likelihood decoding has become a practical approach. For an optimum sequence decoder on AWGN channel, the decision rule for convolutional will depend on the free Euclidean distance. That means the maximum likelihood path is the one with minimum path metric (Euclidean distance). The basic concept of Viterbi algorithm is that it will only store the path with minimum metric among all the paths arriving at the same state at each node level.
It should be noted that TCM is not a free lunch, the coding gain achieved is at the expense of decoding complexity. The new trade-off for TCM is between bandwidth expansion and decoding complexity if signal power were kept constant.

2.3 High Dimensional Trellis-Coded Modulation

HDTMC (High Dimensional Trellis Coded Modulation), a scheme that will be applied in spread spectrum communications is the backbone of this thesis research project. Unlike conventional TCM, HDTCM operates on a block by block range rather than continuously, which seems like a kind of block code with trellis structure. This is achieved though a so called “state constraint” technique [3] [10], in which a circular trellis structure can be found for the specific state number case that the starting state of the trellis path is equal to the ending state of the path. All the possible states can be the starting state for a certain legal trellis path, and one-to-one mapping between the information source block and a legal trellis path can be accomplished by a properly designed state table. Such a state table can be constructed by using Zech’s logarithm table and has been investigated by other researchers [3][11]. The next step is to map channel symbols to the legal trellis path, where the high dimensional signal constellation is incorporated. As for the TCM, the optimum transmission symbol is the one that can achieve the maximization of the minimum Euclidean distance. But the method of assigning the transmission symbol in HDTCM is quite different with TCM, such as the bi-orthogonal signaling. The truly high dimensional signal set will make HDTCM suitable for power limited channels in spread spectrum communications. The expanded constellation achieves coding gain while simultaneously providing a way of
accomplishing spreading which is the core part in spread spectrum communications. Thus the coding gain of HDTCM scheme is achieved by integrating coding, modulation and spreading together as one entity.

Decoding of HDTCM also uses Viterbi algorithm, but it must be modified since the initial state at each time is not available, thus some trials must be performed at first to determine which state has the greatest possibility as the starting state. As stated before, coding gain in TCM is at the price of decoding complexity, which is also true for HDTCM. The proposed decoding scheme for HDTCM applies the concatenated decoding structure where excellent error performance can be obtained. In this concatenated system, Viterbi decoder for circular trellis path is used as the inner decoder while the non-binary BCH code is used as the outer decoder. Still the computation complexity and the memory consumption of the Viterbi decoder and its combination with Block codes make the decoder slow in speed. However, with the fast development in the speed of microprocessors and other computer devices, the computation complexity will not be a constraint in the future.
CHAPTER 3
LINEAR BLOCK CODES, CYCLIC CODES AND BCH CODES

3.1 Linear Block Code

The linear block codes have a code rate of \( k/n \), in which \( k \) is the message length and \( n \) is the code length. Thus \((n-k)\) redundancy bits are added to the original message symbol of \( k \) bits. The concept of error correcting capability is closely related to the definition of minimum distance of block code. For block code, hamming distance is considered rather than the Euclidean distance which is useful in TCM.

Hamming Distance: Number of code bits in which two code words differ.

The minimum distance of block code \( C \) is the minimum Hamming distance between all distinct pairs of code words in \( C \). Since \( C \) forms a vector subspace over \( GF(q) \), the linear combination of any set of code words is also a code word, which infers that the all zero vector is always contained in linear codes. Then the code word with the lowest weight among all nonzero code words indicates the minimum distance of a linear code. The undetectable error patterns are those error words that can change a transmitted code word to another legal code word. The error correction capability is as follows:

A code with minimum distance of \( d_{\text{min}} \) can detect up to \((d-1)\) symbol errors while correcting a maximum of \( [(d-1)/2] \) symbol errors.

This decoding capability can be visualized by using the concept of the decoding sphere. As an example, here are two legal code words in code family \( C \) labeled as \( c_1, c_2, \)
and a received word \( v \), that is in the same vector space but not the code word as showing in figure 3.1. We assume that the minimum distance of code \( C \) is the Hamming distance between code words \( c_1 \) and \( c_2 \) which is 3 in the example. Now suppose code \( c_1 \) is sent but received as the word \( v_1 \). First, error can be detected, since the received word is not the legal code word. Then this error can be corrected by examining the Hamming distance of the received word to all legal code words. Obviously, \( c_1 \) is the code word that is nearest to \( v_1 \), if based on maximum likely decoding algorithm that is normally used, \( c_1 \) is the correct sending code word. Thus the error-correction sphere of a specific code word \( c \) is defined as a sphere centered at the code vector with the radius \( \left\lfloor \frac{d_{\min}}{2} - \frac{1}{2} \right\rfloor \). Any received word within or on the decoding sphere can be correctly decoded.

![Decoding spheres](image)

**Figure 3.1: Decoding spheres**

The dimension of a linear block code is the dimension of the corresponding vector space. Normally a linear code of length \( n \) and dimension \( k \) is called a \((n, k)\) code. If this \((n, k)\) code have its symbols from field \( GF(q) \) (namely an \( q \)-ary code), it will have a
total of $q^k$ code words each of length $n$. The minimum distance $d_{\text{min}}$ of a $(n,k)$ code is bounded by Singleton Bound, which is expressed as $d_{\text{min}} \leq n - k + 1$.

From the knowledge of Linear Algebra, we know every vector space $V$ has an independent basis from which all the vectors in $V$ can be constructed through linear combination. Thus with a given basis $\{g_i\}$, every code word of a linear block code can be uniquely constructed with linear combinations of the basis elements. Let $\{g_0, g_1, g_2, \ldots, g_{k-1}\}$ be a basis for the $(n,k)$ $q$-ary code $C$, the representation of the code word (linear combination of basis $\{g_i\}$) is as follows:

$$c = a_0 g_0 + a_1 g_1 + \Lambda + a_{k-1} g_{k-1}$$  \hspace{1cm} (3.1)

Matrix $G$ is defined as the generating matrix in the form of:

$$G = \begin{bmatrix} g_0 \\ g_1 \\ M \\ g_{k-1} \end{bmatrix} = \begin{bmatrix} g_{0,0} & g_{0,1} & \Lambda & g_{0,n-1} \\ g_{1,0} & g_{1,1} & \Lambda & g_{1,n-1} \\ M & M & O & M \\ g_{k-1,0} & g_{k-1,1} & \Lambda & g_{k-1,n-1} \end{bmatrix}$$  \hspace{1cm} (3.2)

Since it is the one-to-one mapping between the set of $k$-symbol blocks $(a_0, a_1, \Lambda, a_{k-1})$ over $\text{GF}(q)$ and the code words in $C$. The code $C$ can readily be used to encode information data if a $q$-ary message block $m = (m_0, m_1, \Lambda, m_{k-1})$ substitute $\{a_i\}$, the encoding process can be expressed in the form of matrix multiplication:

$$mG = (m_0, m_1, \Lambda, m_{k-1}) \begin{bmatrix} g_0 \\ g_1 \\ M \\ g_{k-1} \end{bmatrix} = m_0 g_0 + m_1 g_1 + \Lambda + m_{k-1} g_{k-1} = c$$  \hspace{1cm} (3.3)
3.2 Cyclic Codes

A linear cyclic code is an important group of linear block codes because of its easy implementation. It contains one of the most powerful codes named BCH codes. A linear \((n,k)\) block code \(C\) is said to be cyclic if the right cyclic shift of any code word \(c\) is also a code word. Since this cyclic feature is analog to multiplication operation of polynomials, a cyclic code can be readily represented by polynomials.

For every code word \(c = (c_0, c_1, \ldots, c_{n-1})\), there exists a polynomial associated with it in the form of \(c(x) = c_0 + c_1x + c_2x^2 + \ldots + c_{n-1}x^{n-1}\). The operation of \(j\) right cyclic shift is equivalent to the multiplication of \(x^j\) modulo \((x^n - 1)\), which can be stated as \(c^j(x) = x^j c(x) \mod (x^n - 1)\). For an arbitrary polynomial in \(GF(q)[x]/(x^n - 1)\) such as \(a(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1}\), the product \(a(x)c(x)\) is a linear combination of cyclic shifts of \(c\). In short, a cyclic code is said to be an ideal \([1]\) within \(GF(q)[x]/(x^n - 1)\) generally. Properties of the \(q\)-ary \((n,k)\) cyclic code \(C\) include:

Within the set of code polynomials in \(C\), there is a unique monic polynomial \(g(x)\) with minimal degree \(r < n\), which is called the generator polynomial of \(C\).

Every code polynomial \(c(x)\) can be expressed uniquely as \(c(x) = m(x)g(x)\), where \(m(x)\) is a polynomial of degree less than \((n - r)\) in \(GF(q)[x]\).

The generator polynomial \(g(x)\) of \(C\) is a factor of \(x^n - 1\) in \(GF(q)[x]\).

From the above theorem, the encoding of cyclic code can be formulated as \(c(x) = m(x)g(x)\) where \(m(x)\) is message polynomial, and \(g(x)\) is generator polynomial.
\[ c(x) = m(x)g(x) = (m_0 + m_1x + \Lambda + m_{n-r-1}x^{n-r-1})g(x) \]
\[ = m_0g(x) + m_1xg(x) + \Lambda + m_{n-r-1}x^{n-r-1}g(x) \]
\[ = \begin{bmatrix} 
    g(x) \\
    xg(x) \\
    M \\
    x^{n-r-1}g(x) 
\end{bmatrix} \tag{3.4} \]

From property 3 of cyclic codes, the determination of the generator polynomial for cyclic code of length \( n \) is to find the factors (irreducible polynomials) of \( (x^n - 1) \). That's why we need to investigate the factorization of \( (x^n - 1) \) in \( \text{GF}(q)[x] \) in the background discussion.

### 3.3 BCH codes

BCH and Reed-Solomon codes form the most powerful type of algebraic codes known. We will investigate its basic properties and explain why the non-binary BCH code is needed for the decoding of HDTCM scheme.

#### 3.3.1 Encoding of BCH codes

The beauty of BCH codes comes from its “design distance”, which means that a particular restraint on the generator polynomial will result in an assured minimum “design distance” code. This “design distance is also known as BCH bound. The following theorem defines the BCH bound [5] and gives the general code design procedure.

**BCH Bound:** For an \( q \)-ary \((n, k)\) cyclic code with generator \( g(x) \), if \( m \) is the multiplicative order of \( q \) modulo \( n \) (\( \text{GF} (q^n) \) is thus the smallest extension field of \( \text{GF}(q) \) that contains a primitive \( n \)th root of unity). Let \( \alpha \) be the primitive \( n \)th root of
unity. Select \( g(x) \) to be a minimal-degree polynomial in \( GF(q^m)[x] \) such that its roots include some consecutive power of \( \alpha \), namely 
\[
g(\alpha^b) = g(\alpha^{b+1}) = g(\alpha^{b+2}) = \Lambda = g(\alpha^{b+\delta - 2}) = 0
\]
for some integers \( b \geq 0 \) and \( \delta \geq 1 \).

The code defined by this \( g(x) \) has minimum distance \( d_{\text{min}} \geq \delta \).

Prove of the above theorem involves the Vandermonde feature of the square submatrices of a BCH parity-check matrix. Interested readers should refer to [5] for an in-depth explanation. Now the core part in BCH code design is to find the \( n \)th root of unity \( \alpha \) in the smallest extension field \( GF(p^m) \) of the giving field \( GF(p) \) and determine the conjugacy class. The minimal polynomial associated with each conjugacy class is now ready to form the generator polynomial. BCH bound requires arbitrarily choosing minimal polynomials that have the selected consecutive powers of \( \alpha \) as roots and uses the common multiple of those minimal polynomials as the generator polynomial.

Narrow-Sense and Primitive BCH Codes: BCH code is said to be narrow-sense if \( b = 1 \). And if \( n = q^m - 1 \) for some positive integer \( m \), it’s a primitive BCH code since \( n \)th root of unity is a primitive element in \( GF(q^m) \).

For example: if we are to design a 4-ary, length 21 BCH code with one-error-correcting capability.

1. Find \( 21^\text{th} \) root of unity \( \gamma \) in \( GF(4^m) \) where \( m \) is minimal.

The smallest \( m \) such that \( 21 | 4^m - 1 \) is 3 thus \( \gamma \) can be found in \( GF(64) \). Let \( \alpha \) be the primitive element in \( GF(64) \) and \( \gamma = \alpha^3 \). The 4 distinct elements in \( GF(4) \) can
be expressed by $\gamma$, which is $\{0, 1, \gamma^7, \gamma^{14}\}$ or equivalent $\{0, 1, \alpha^{21}, \alpha^{42}\}$ or simplified as $\{0, 1, \beta, \beta^2\}$ if $\beta = \alpha^{21}$.

2. Find cyclotomic cosets modulo 21 with respect to GF(4) and the associated minimal polynomials. (table 3.1)

Table 3.1 Cyclotomic cosets and minimal polynomials for 4-ary BCH code of length 21

<table>
<thead>
<tr>
<th>Cyclotomic cosets</th>
<th>Minimal Polynomials $M_i(x)$</th>
<th>Order of $M_i(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_0 = {0}$</td>
<td>$M_0(x) = x + 1$</td>
<td>$ord = 1$</td>
</tr>
<tr>
<td>$C_1 = {1, 4, 16}$</td>
<td>$M_1(x) = x^3 + \alpha^{42}x + 1$</td>
<td>$ord = 3$</td>
</tr>
<tr>
<td>$C_2 = {2, 8, 11}$</td>
<td>$M_2(x) = x^3 + \alpha^{21}x + 1$</td>
<td>$ord = 3$</td>
</tr>
<tr>
<td>$C_3 = {3, 12, 6}$</td>
<td>$M_3(x) = x^3 + x^2 + 1$</td>
<td>$ord = 3$</td>
</tr>
<tr>
<td>$C_4 = {5, 20, 17}$</td>
<td>$M_4(x) = x^3 + \alpha^{42}x^2 + 1$</td>
<td>$ord = 3$</td>
</tr>
<tr>
<td>$C_7 = {7}$</td>
<td>$M_7(x) = x + \alpha^{21}$</td>
<td>$ord = 1$</td>
</tr>
<tr>
<td>$C_9 = {9, 15, 18}$</td>
<td>$M_9(x) = x^3 + x + 1$</td>
<td>$ord = 3$</td>
</tr>
<tr>
<td>$C_{10} = {10, 19, 13}$</td>
<td>$M_{10}(x) = x^3 + \alpha^{21}x^2 + 1$</td>
<td>$ord = 3$</td>
</tr>
<tr>
<td>$C_{14} = {14}$</td>
<td>$M_{14}(x) = x + \alpha^{42}$</td>
<td>$ord = 1$</td>
</tr>
</tbody>
</table>

3. Find the generator with $2t$ consecutive powers of $\alpha$ as the roots.

For a narrow-sense BCH code, we can choose $\gamma, \gamma^2$ as the roots for $g(x)$. Thus

$$g(x) = LCM(M_1(x), M_4(x)) = M_1(x)M_4(x)$$

$$= (x^3 + \alpha^{42}x + 1)(x^3 + \alpha^{21}x + 1)$$

$$= x^6 + x^4 + x^2 + x + 1$$

The 6-degree $g(x)$ gives us a (21, 15), 4-ary one-error-correcting BCH code. It should be noted that in the above example, we arbitrarily set $b = 1$ to construct the
narrow-sense code. However this narrow-sense code is not the best performance code in length 21, 4-ary, one-error-correcting BCH code family. When constructing error correction codes, we want to achieve certain error correction capability while simultaneously minimizing the redundancy added to the data. In BCH code, it's equivalent to minimizing the number of "extraneous" zeros (roots of generator polynomial) while still ensuring the required number of consecutive zeros. By carefully selecting the starting root \( \varphi^b \) of generator polynomial, it's possible to find a code that has the same error correcting capability as a narrow-sense code but with larger dimension or less redundancy. As for the above case, if we set \( b = 6 \), the resulting generator polynomials will be \( g(x) = x^3 + \alpha^{42}x^2 + \alpha^{42}x + \alpha^{21} \) which gives a (21,17) one-error-correcting code that has a higher code rate than the previous example. The reason is that minimal polynomial \( M_{\gamma}(x) \) whose order is only one has been chosen as a factor of generating polynomial. However, this inequality of the BCH bound shows that the code constructed may exceed the design distance, which means different setting of \( b \) may result in codes of different correction capabilities. Thus one of the sub-optimal algorithm in BCH code design is to select a code with at least the required number of consecutive roots (meet design distance) while has the maximum dimension. A BCH code constructed in this way may have its true error correcting capability larger than the BCH bound. Also, for a specific design distance, using the above sub-optimum algorithm, we may not find any code to match that design distance and that design distance is non-existent for certain BCH codes.
Figure 3.2 is the flow chart of the sub-optimum algorithm for selecting minimal degree generator polynomial. Using this algorithm, it’s possible to produce the same code with different design distance, which means for a certain design distance, there doesn’t exists such a code with a good code rate that still matches the design distance.

**Figure 3.2: Sub-optimum BCH code selection algorithm**

3.3.2 Erasures in BCH code

It’s helpful to introduce the concept of erasure [13] here to aid further investigation of non-binary BCH code. As we have stated before, a hard-decision receiver forces the channel output signal to fit into a finite number of symbols, which destroys the information that can improve the system performance. Rather than forcing a decision that is likely to be incorrect, the receiver can claim the corresponding symbol values of certain received signals in doubt by indicating them as erasure. Suppose a received word
has a single erased coordinate or one code word has been erased. Over the unerased coordinates, all pairs of distinct code words are separated by a Hamming distance of \((d_{\text{min}} - 1)\). Thus the effective \(d_{\text{min}}\) of a received code word with \(f\) erasures becomes \((d_{\text{min}} - f)\) over the unerased coordinates and the resulting error correction capability changes to:

\[
e = \left\lfloor \frac{d_{\text{min}} - f - 1}{2} \right\rfloor
\]

In other words, with a given minimum distance of a code, we can guarantee it to correct up to \(e\) errors plus \(f\) erasures as long as \((2e + f) < d_{\text{min}}\). Since twice as many erasures as errors can be corrected for a fixed \(d_{\text{min}}\), a performance improving technique can be proposed as transforming errors to erasures in a code. This ideal will be addressed later in the next chapter with discussions of the decoding system.

### 3.4 Reed-Solomon Codes

Reed-Solomon codes can be considered as a special sub-group of BCH codes. The definition of Reed-Solomon codes is quite simple.

A Reed-Solomon code is an \(q^m\)-ary BCH code of length \(q^m - 1\). Let's consider a \(q^m\)-ary BCH code. If the length of the code is \(q^m - 1\), the primitive \(q^m\)-th root of unity will be found in \(GF(q^m)\). Then the construction of cyclotomic cosets modulo \((q^m - 1)\) over \(q^m\) becomes trivial since the code symbol directly comes from the extension field where the \(q^m\)-th root of unity resides. As we will see in the following example, the cyclotomic cosets are singleton sets of the form \(\{ s \}\) because \((s \cdot q^m) = s \mod(q^m - 1)\).
And the associated minimal polynomials are all of the form \((x - \alpha^r)\) and have a degree of 1.

For a \(t\)-error-correcting BCH code, \(2 \cdot t\) consecutive powers of \(\alpha\) are required as zeros of the generator polynomial \(g(x)\). Thus the product of the associated minimal polynomial gives \(g(x)\); there is no need to perform the least common multiplication operation as for general BCH codes. Two good features of Reed-Solomon codes are:

1. Minimum distance of Reed-Solomon code

Unlike general BCH codes, minimum distance in Reed-Solomon codes will never exceed the design distance because of the singleton set of cyclotomic cosets. A \((n, k)\) Reed-Solomon code has minimum distance of \((n - k + 1)\). We also call a \((n, k)\) code that satisfies the Singleton bound with equality the maximum-distance separable (MDS) code. Therefore Reed-Solomon codes is MDS codes.

2. Weight Distribution of Reed-Solomon codes

The weight distribution of error correction codes refers to the distribution of code words according to their weights, or the number of code words with a particular weight. Determining the weight distribution of non-binary BCH codes is a hot research area. Unlike BCH codes which have too many selections to affect the calculation of weight distribution, Reed-Solomon codes have standard weight distribution formulas because of the simple form of cyclotomic cosets.

Example: double-error-correcting 16-ary Reed-Solomon Code, length 15

Suppose \(\alpha\) is the primitive element in GF(16), and thus a primitive 15\(^{th}\) root of unity. The factorization of \((x^{15} - 1)\) results in the form of:
\[ x^{14} - 1 = (x - 1)(x - \alpha)(x - \alpha^2)(x - \alpha^4). \]

Because the cyclotomic cosets for 16-ary length 15 Reed-Solomon code is:

\{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}, \{7\}, \{8\}, \{9\}, \{10\}, \{11\}, \{12\}, \{13\}, \{14\}

For double-error-correcting, the generator polynomial \( g(x) \) must have \( 2 \cdot t = 4 \) consecutive powers of \( \alpha \) as zeros. An important note is that since Reed-Solomon codes are guaranteed to have the minimum distance equals design distance, there is no need to explore codes other than narrow-sense. Therefore, a narrow-sense generator polynomial can be constructed as:

\[ g(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4) = x^4 + \alpha^{13}x^3 + \alpha^6x^2 + \alpha^3x + \alpha^{10} \]

### 3.5 Simulation Results

We will go back to non-binary BCH codes for further analysis of code weight and decoding capability with the sub-optimum algorithm proposed. Tables 3.2 list all known 21 length, 4-ary BCH codes. The simulation result comes from a test program that constructs a code’s generating polynomial for each erasure correction capability set from 1 to \((n - 1)\). The output code is always the most efficient code with the largest dimension.

**Table 3.2 List of 21 length 4-ary BCH codes**

<table>
<thead>
<tr>
<th>Message length k</th>
<th>18</th>
<th>17</th>
<th>14</th>
<th>12</th>
<th>11</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>4</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Erasure correction capability ( t )</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>8</td>
<td>10</td>
<td>11</td>
<td>13</td>
<td>20</td>
</tr>
<tr>
<td>Starting power ( b ) of ( \alpha ) as zero of generating polynomial ( g(x) )</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Notice that there is no code of 7-erasure correction capability with acceptable dimension. A 7-erasure-correction BCH code on 4-ary with length 21 may or may not be found, because the number of message symbols it contains is too small. So utilizing the 8 erasure-correction code may be a better choice in selecting codes for a system.

Table 3.3 Acceptable 4-ary BCH code of lengths 31, 33, 35 and 39

<table>
<thead>
<tr>
<th>Code dimension k</th>
<th>30</th>
<th>21</th>
<th>16</th>
<th>11</th>
<th>6</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error correction capability $t_2$</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>10</td>
<td>14</td>
<td>29</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code dimension k</th>
<th>28</th>
<th>27</th>
<th>22</th>
<th>17</th>
<th>12</th>
<th>12</th>
<th>7</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error correction capability $t_2$</td>
<td>1</td>
<td>2</td>
<td>5</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>21</td>
<td>32</td>
</tr>
<tr>
<td>B</td>
<td>11</td>
<td>10</td>
<td>9</td>
<td>7</td>
<td>6</td>
<td>7</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code dimension k</th>
<th>29</th>
<th>30</th>
<th>27</th>
<th>24</th>
<th>20</th>
<th>18</th>
<th>18</th>
<th>15</th>
<th>9</th>
<th>7</th>
<th>6</th>
<th>4</th>
<th>3</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error correction capability $t_2$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>20</td>
<td>34</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>14</td>
<td>12</td>
<td>9</td>
<td>11</td>
<td>9</td>
<td>8</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code dimension k</th>
<th>33</th>
<th>32</th>
<th>26</th>
<th>21</th>
<th>20</th>
<th>14</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error correction capability $t_2$</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>25</td>
<td>38</td>
</tr>
<tr>
<td>B</td>
<td>13</td>
<td>12</td>
<td>11</td>
<td>1</td>
<td>8</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>13</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Tables 3.3 list the 4-ary BCH codes of lengths 31, 33, 35 and 39. Still, only the acceptable codes are listed, since there are two many selections in non-binary BCH codes, the obvious inefficient codes should be excluded. These code lengths are close to each
other, but their performances are quite different. 35-length codes provide the largest selections in code design with a number of available individual codes while 31-length codes have the smallest resolutions, which is due to the different cardinality of cyclotomic cosets of the different length codes. It’s interesting to note that when \( n = \frac{q^m}{\sqrt[4]{q}} - 1 \), where \( \sqrt[4]{q} \) is the primitive root of code symbol alphabet size, the performance of the code family seems to be the poorest. As an example of 31-length codes, \( 31 = \frac{4^3}{2} - 1 \), therefore only 6 possible selection of codes exist of which only (31,21) and (31,16) can be used in practical applications.

For a number of code lengths available for non-binary BCH codes, it’s desirable to test many 4-ary BCH codes of different lengths which is determined by factorizing \( (4^m - 1) \) with \( m \) be an integer larger than 1. Note that \( m \) should be the smallest one that gives the extension field where \( n \)-th roots of unity can be found since there is no need to compute the same length code in a larger Galois field which will incur increased computation complexity. After deleting the trivial case, code lengths of 4-ary BCH codes along with their extension Galois filed \( \text{GF}(4^m) \) are listed as in table 3.4. Integer \( m \) determines the Galois field where the \( n \)-th root of unity can be found and the computation complexity. As an example, \( m \) equals 8 means all the calculations are in \( \text{GF}(4^8) \), a huge field. It’s time-consuming as well as memory consuming to calculate in a large Galois field. That’s why my simulation program can only be applied to a small range of code lengths for 4-ary BCH codes.
To compare different length BCH codes with the same symbol alphabet size of 4, we use the table 3.5 to show the variable length, 4-ary BCH codes with the same design distance of 4, 5, 6, 7 and 8 respectively. In every slot, code dimension $k$ and achieved erasure correcting capability $t_2$ is presented as $(k, t_2)$ or is left blank indicating no such code exists for the specified design distance. These codes beyond doubt are the best codes available to meet the design distance using the proposed sub-optimum algorithm. It’s interesting to note that certain codes are inefficient when compared with other codes of the same error correction capability but larger dimension. For instance, in table 3.5, code $(35, 20)$ has 5 erasure-correction capability, however the smaller length code $(33, 22)$ has the same decoding capability and less redundancy. Thus we will always choose code $(33, 22)$ over $(35, 20)$ code since code $(33, 22)$ can carry more information symbols than
while providing the same error correction capability. If the code length is restricted to 35 in specific applications, code \((35,20)\) will be the only choice.

Table 3.5 Comparison of variable length 4-ary BCH codes

<table>
<thead>
<tr>
<th>Length</th>
<th>((k,t_2)) for Designed (d_{\text{min}} = 4)</th>
<th>((k,t_2)) for Designed (d_{\text{min}} = 5)</th>
<th>((k,t_2)) for Designed (d_{\text{min}} = 6)</th>
<th>((k,t_2)) for Designed (d_{\text{min}} = 7)</th>
<th>((k,t_2)) for Designed (d_{\text{min}} = 8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>(14,3)</td>
<td>(12,4)</td>
<td>(11,5)</td>
<td>(9,6)</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td>(21,4)</td>
<td></td>
<td>(16,6)</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
<td></td>
<td>(22,5)</td>
<td></td>
<td>(17,7)</td>
</tr>
<tr>
<td>35</td>
<td>(27,3)</td>
<td>(24,4)</td>
<td>(20,5)</td>
<td>(18,6)</td>
<td>(18,7)</td>
</tr>
<tr>
<td>39</td>
<td>(26,3)</td>
<td></td>
<td>(21,5)</td>
<td>(20,6)</td>
<td>(14,7)</td>
</tr>
<tr>
<td>45</td>
<td>(34,3)</td>
<td>(32,4)</td>
<td>(30,5)</td>
<td>(27,6)</td>
<td>(25,7)</td>
</tr>
<tr>
<td>51</td>
<td>(42,3)</td>
<td>(39,4)</td>
<td>(38,5)</td>
<td>(34,6)</td>
<td>(30,7)</td>
</tr>
<tr>
<td>63</td>
<td>(56,3)</td>
<td>(54,4)</td>
<td>(53,5)</td>
<td>(50,6)</td>
<td>(47,7)</td>
</tr>
<tr>
<td>65</td>
<td>(53,3)</td>
<td>(53,4)</td>
<td>(47,5)</td>
<td>(47,6)</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>(77,3)</td>
<td>(73,4)</td>
<td>(73,5)</td>
<td>(69,6)</td>
<td>(67,7)</td>
</tr>
<tr>
<td>91</td>
<td>(79,3)</td>
<td>(76,4)</td>
<td>(73,5)</td>
<td>(70,6)</td>
<td>(64,7)</td>
</tr>
<tr>
<td>105</td>
<td>(91,3)</td>
<td>(88,4)</td>
<td>(86,5)</td>
<td>(80,6)</td>
<td>(77,7)</td>
</tr>
<tr>
<td>127</td>
<td>(113,4)</td>
<td>(113,4)</td>
<td></td>
<td>(106,6)</td>
<td></td>
</tr>
</tbody>
</table>

It's interesting to note that \(p\)-ary BCH codes of length \(n = p^m - 1\), the primitive BCH codes, perform much better. Let's see the 4-ary 63-length BCH code, its code dimension is larger than the longer length codes (e.g. \(n = 65\)) at the same error correction capability levels. The reason can be traced back to the cyclotomic cosets of primitive BCH codes, of which two general observations [5] hold:
• For a fixed code-symbol alphabet filed $GF(q)$, the cardinality of cyclotomic cosets modulo $n$ is generally small for $n = q^m - 1$, thus providing more resolutions in code design and fewer extraneous zeros which might increase the code dimensions.

• For fixed code length $n$, the cardinality of cyclotomic cosets modulo $n$ grows smaller as the code symbol alphabet size increases.

With BCH code length $n = q^m - 1$ fixed, we can increase the code symbol alphabet size $q$ to decrease the cardinality of cyclotomic cosets until the maximum value reached is at $q^m$, which gives us the Reed-Solomon codes. Recall that the RS codes is the MDS codes in that the cyclotomic cosets each contains only one element, thus no extraneous zeros. Generally speaking, the smaller the cardinality of cyclotomic cosets, the better the code performance. We have discussed that codes with length $n = \frac{q^m}{\sqrt{q}} - 1$ have poor performance, which can also be attributed to the large cardinality of their cyclotomic cosets. To further explore this topic, 127-length codes are also tested since $127 = \frac{4^4}{2} - 1$.

These results can be found in the appendix of this thesis.

It’s helpful to take a close look on the 4-ary, primitive BCH code of length 63. The cyclotomic cosets modulo 63 over $GF(4)$ as well as the associated minimal polynomials are given in table 3.6. Here, \{21\} and \{42\} correspond to the 64th root of unity, which means there is only one element in the cyclotomic coset where the associated minimal polynomial has the order of 1.
Table 3.6 cyclotomic cosets and minimal polynomials for primitive 4-ary BCH code

Note: the symbol alphabet $\{0, 1, \gamma^{21}, \gamma^{42}\}$ where $\gamma$ is the primitive element in the extension field $GF(64)$ has been simplified as $\{0, 1, \beta, \beta^2\}$.

<table>
<thead>
<tr>
<th>Cyclotomic cosets modulo 63</th>
<th>Minimal polynomials</th>
</tr>
</thead>
<tbody>
<tr>
<td>${0}$</td>
<td>$x + 1$</td>
</tr>
<tr>
<td>${1, 4, 16}$</td>
<td>$x^3 + x^2 + \beta^2 x + \beta$</td>
</tr>
<tr>
<td>${2, 8, 32}$</td>
<td>$x^3 + x^2 + \beta x + \beta^2$</td>
</tr>
<tr>
<td>${3, 12, 48}$</td>
<td>$x^3 + \beta^2 x + 1$</td>
</tr>
<tr>
<td>${5, 20, 17}$</td>
<td>$x^3 + \beta^2 x^2 + \beta x + \beta^2$</td>
</tr>
<tr>
<td>${6, 24, 33}$</td>
<td>$x^3 + \beta x + 1$</td>
</tr>
<tr>
<td>${7, 28, 49}$</td>
<td>$x^3 + \beta$</td>
</tr>
<tr>
<td>${9, 36, 18}$</td>
<td>$x^3 + x^2 + 1$</td>
</tr>
<tr>
<td>${10, 40, 34}$</td>
<td>$x^3 + \beta x^2 + \beta^2 x + \beta$</td>
</tr>
<tr>
<td>${11, 44, 50}$</td>
<td>$x^3 + \beta^2 x^2 + \beta^2 x + \beta^2$</td>
</tr>
<tr>
<td>${13, 52, 19}$</td>
<td>$x^3 + x^2 + x + \beta$</td>
</tr>
<tr>
<td>${14, 56, 35}$</td>
<td>$x^3 + \beta^2$</td>
</tr>
<tr>
<td>${15, 60, 51}$</td>
<td>$x^3 + \beta^2 x^2 + 1$</td>
</tr>
<tr>
<td>${21}$</td>
<td>$x + \beta$</td>
</tr>
<tr>
<td>${22, 35, 37}$</td>
<td>$x^3 + \beta x^2 + \beta x + \beta$</td>
</tr>
<tr>
<td>${23, 29, 53}$</td>
<td>$x^3 + \beta x^2 + x + \beta^2$</td>
</tr>
<tr>
<td>${26, 41, 38}$</td>
<td>$x^3 + x^2 + x + \beta^2$</td>
</tr>
<tr>
<td>${27, 45, 54}$</td>
<td>$x^3 + x + 1$</td>
</tr>
<tr>
<td>${30, 57, 39}$</td>
<td>$x^3 + \beta x^2 + 1$</td>
</tr>
<tr>
<td>${31, 61, 55}$</td>
<td>$x^3 + \beta^2 x^2 + \beta^2 x + \beta$</td>
</tr>
<tr>
<td>${42}$</td>
<td>$x + \beta^2$</td>
</tr>
<tr>
<td>${43, 46, 58}$</td>
<td>$x^3 + \beta^2 x^2 + x + \beta$</td>
</tr>
<tr>
<td>${47, 62, 59}$</td>
<td>$x^3 + \beta x^2 + \beta^2 x + \beta^2$</td>
</tr>
</tbody>
</table>
As a comparison, the cyclotomic cosets modulo 65 over $GF(4)$ are listed in table 3.7 to show the difference of its cardinality from the primitive BCH code.

Table 3.7 cyclotomic cosets modulo 65 with respect to $GF(4)$

<table>
<thead>
<tr>
<th>Cyclotomic coset modulo 65</th>
<th>Order of the minimal polynomial</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0}</td>
<td>$ord = 1$</td>
</tr>
<tr>
<td>{1, 4, 16, 64, 61, 49}</td>
<td>$ord = 6$</td>
</tr>
<tr>
<td>{2, 8, 32, 63, 57, 33}</td>
<td>$ord = 6$</td>
</tr>
<tr>
<td>{3, 12, 48, 62, 53, 17}</td>
<td>$ord = 6$</td>
</tr>
<tr>
<td>{5, 20, 15, 50, 45, 50}</td>
<td>$ord = 6$</td>
</tr>
<tr>
<td>{6, 24, 31, 59, 41, 34}</td>
<td>$ord = 6$</td>
</tr>
<tr>
<td>{7, 28, 47, 58, 37, 18}</td>
<td>$ord = 6$</td>
</tr>
<tr>
<td>{9, 36, 14, 56, 29, 51}</td>
<td>$ord = 6$</td>
</tr>
<tr>
<td>{10, 40, 30, 55, 25, 35}</td>
<td>$ord = 6$</td>
</tr>
<tr>
<td>{11, 44, 46, 54, 21, 19}</td>
<td>$ord = 6$</td>
</tr>
<tr>
<td>{13, 52}</td>
<td>$ord = 2$</td>
</tr>
<tr>
<td>{22, 23, 27, 43, 42, 38}</td>
<td>$ord = 6$</td>
</tr>
<tr>
<td>{26, 39}</td>
<td>$ord = 2$</td>
</tr>
</tbody>
</table>

As shown in the tables, with the symbol alphabet size be fixed at 4, the cardinality of cyclotomic cosets modulo 65 is much larger than that of the cyclotomic cosets modulo 63. Although $65 > (4^3 - 1) = 63$, it should be pointed out that the cardinality of cyclotomic cosets modulo 255, which gives the next primitive BCH code family, will only be 4. The constructed code families (refer to table 3.8) of length 65 and length 63 verify that the primitive codes has better performance. The generator polynomials for some 4-ary, variable length code families can not be listed here for the sake of spaces. They can be found at the appendix B of this thesis.
We compared the primitive BCH codes with the same length Reed-Solomon codes as shown in figure 3.3. The code curves with erasure correction capability $t_2$ versus the code dimension $k$ are plotted for both 4-ary, 255-length BCH code family and the same length RS code family. It’s easy to infer that the curve for the RS code family is a straight line because $k + t_2 = n$ holds for MDS codes. While the BCH code curve is inclined in the middle part with the two end points coincident with RS code curve. The straight line of RS codes shows that the MDS codes are the best possible codes we can find. No doubt that the curve for non-binary primitive BCH codes must be below that straight line. We

### Table 3.8 list of 4-ary length 63 BCH codes

Note: $k$ stands for the code dimension, $t_2$ is the erasure correcting capability

<table>
<thead>
<tr>
<th>4-ary 65-length primitive BCH codes</th>
<th>4-ary 63-length BCH codes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>K</strong></td>
<td><strong>t_2</strong></td>
</tr>
<tr>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td>59</td>
<td>2</td>
</tr>
<tr>
<td>53</td>
<td>3</td>
</tr>
<tr>
<td>53</td>
<td>4</td>
</tr>
<tr>
<td>47</td>
<td>5</td>
</tr>
<tr>
<td>47</td>
<td>6</td>
</tr>
<tr>
<td>41</td>
<td>8</td>
</tr>
<tr>
<td>35</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
can also infer the curve locations of other non-primitive BCH codes. Those curves will distribute between any two RS code curves of roughly the same shape as the primitive BCH code curves.

![Figure 3.3 Comparison of 4-ary, n=255 primitive BCH codes and n=255 Reed-Solomon codes](image)

Figure 3.3 Comparison of 4-ary, \(n=255\) primitive BCH codes and \(n=255\) Reed-Solomon codes

In short, among all 4-ary BCH codes, code lengths at 15, 63, 255, 1023 and 4095 are primitive codes with better performance than non-primitive codes at nearby lengths. Code families at length 15, 63, 255 have been tested to form the complete lists of error correction capabilities as well as the code dimensions (see Appendix).
CHAPTER 4
DECODING OF NON-BINARY BCH CODES

In this chapter, we will first discuss the popular decoding algorithms for non-binary BCH and Reed-Solomon codes. A detail description of the erasure decoding issue as well as the burst error decoding problem will be covered to propose the possible usage of non-binary BCH codes in combination with convolutional codes (specifically Viterbi decoding algorithm) to decode the HDTCM system.

4.1 Overview of Decoding Algorithm for Non-binary BCH code

There exist different decoding schemes [7] for non-binary BCH codes, namely syndrome-based decoding, remainder-based decoding, and transform decoding, all of which belong to hard-decision decoding (HDD). Some researchers have also investigated the maximum-likelihood soft-decision decoding (SDD) algorithms for BCH codes, especially for non-binary BCH codes and Reed-Solomon codes. Soft-decision decoding techniques can provide approximately 2 dB more coding gain than HDD in white Gaussian channel. The difference between SDD and HDD is that SDD accepts a vector of real samples (continues) of the noisy channel output and estimates the vector of channel input symbols that transmitted while HDD requires that its input be from the same alphabet as the channel input. Of the hard-decision decoding, Berlekamp-Massey algorithm [8] became a breakthrough in the searching of efficient decoding algorithm for non-binary BCH codes. The shift register approach makes this algorithm easy to
implement in VLSI. Using the standard decoding algorithm, we can investigate non-binary BCH codes of their polynomial structure, weight-distributions and decoding capabilities. The combination of non-binary BCH codes with the convolutional codes to provide a better decoding scheme for HDTDM is also included.

Syndrome-based algorithm is the most popular decoding algorithm for BCH codes because it can be implemented using shift registers, which makes commercial VLSI BCH codec available. My simulation work on non-binary BCH codes is based on the syndrome-based decoding algorithm.

Unlike syndrome calculation, remainder-based algorithm doesn't require syndrome value. Their algorithm instead relies on the remainder polynomial \( r(x) \) obtained from the division of the received polynomial \( v(x) \) by the generator polynomial \( g(x) \). Compared to the syndrome-based algorithm, it requires more registers in hardware implementation since more computation load has been transferred to the polynomial evaluation.

Transform decoding is based on the fact that BCH encoding can be deemed as Fourier transforms. It requires a Fourier transform at the beginning of the algebraic decoding algorithm if in frequency domain. The algebraic decoding part is the same as the syndrome-based algorithm. We need to evaluate the error location polynomials as well as the error magnitude polynomials.

### 4.2 Syndrome-Based Decoding and Berlekamp's Algorithm

Syndrome here is defined as the values that contain the error and erasure information. The successful manipulation of the syndrome values can lead to the solving of errors and erasures, which is the so called the syndrome based decoding algorithm.
The standard algorithm is introduced to show the mathematical derivation. Efficient algorithm will be covered later as the complement.

### 4.2.1 Standard Decoding Algorithm

Recall the construction of BCH codes discussed in the last chapter. For a legal code word \( c = (c_0, c_1, \Lambda, c_{n-1}) \), the code polynomial \( c(x) = c_0 + c_1 x + \Lambda + c_{n-1} x^{n-1} \) has \( 2t \) consecutive powers of \( \alpha \) as zeros because the code word is generated by the multiplication of generator polynomial \( g(x) \) with the message polynomial \( m(x) \). Now the received polynomial \( r(x) \) can be expressed as the sum of the transmitted code polynomial \( c(x) \) and the error polynomial \( e(x) = e_0 + e_1 x + \Lambda + e_{n-1} x^{n-1} \). A series of syndromes are calculated using the following equation, which actually is to evaluate the received polynomial at the \( 2t \) consecutive powers of \( \alpha \).

\[
S_j = r(\alpha^j) = c(\alpha^j) + e(\alpha^j) = e(\alpha^j) = \sum_{k=0}^{n-1} e_k (\alpha^j)^k, \quad j = b, b+1, b+2, \ldots, b+2t-1
\]  

(4.1)

The standard algorithm assumes narrow-sense codes for the purpose of minimizing the complexity of notation. However, since the encoding simulation under discussion is programmed to find the best possible codes (smallest redundancy) for giving error-correction constraint, we can not always assume a narrow-sense code.

The syndrome values will be all zeros if there is no corruption during transmission of the channel symbols since the received polynomial has the same \( 2t \) roots, which indicates an all zero error polynomial \( e(x) \). For the error case, syndrome values will exclusively depend on the error polynomial \( e(x) \) because the code polynomial \( c(x) \) that
has the same $2t$ zeros will never contribute to the resulting syndromes. Assuming the received word $r$ has $v$ errors at positions $i_1, i_2, \ldots, i_v$ with the error magnitude $e_{i_1}, e_{i_2}, \ldots, e_{i_v}$. The syndrome equations become:

$$S_j = e(\alpha^j) = \sum_{k=0}^{n-1} e_k (\alpha^j)^k = \sum_{i=1}^{v} e_{i} X_i^j$$  \hspace{1cm} (4.2)

We can express (4.2) in a series of $2t$ algebraic equations in $2v$ unknowns, which gives the equation 4.3.

$$S_{b+1} = e_{i_1} X_1^{b+1} + e_{i_2} X_2^{b+1} + \Lambda + e_{i_v} X_v^{b+1}$$

$$S_{b+2} = e_{i_1} X_1^{b+2} + e_{i_2} X_2^{b+2} + \Lambda + e_{i_v} X_v^{b+2}$$

$$M$$

$$S_{b+2t} = e_{i_1} X_1^{b+2t} + e_{i_2} X_2^{b+2t} + \Lambda + e_{i_v} X_v^{b+2t}$$  \hspace{1cm} (4.3)

The key point now is to reduce the above system to a set of linear functions. An error locator polynomial $\Lambda(x)$ (equation 4.4) is defined as a polynomial whose zeros are the error locators $\{X_i\}$.

$$\Lambda(x) = \prod_{i=1}^{v} (x + X_i) = \Lambda_0 + \Lambda_1 x + \Lambda_2 x^2 + \Lambda + \Lambda_{v-1} x^{v-1} + x^v$$  \hspace{1cm} (4.4)

For any $l$ within $1 \leq l \leq v$, we can multiply both sides of 6.4 by $e_{i_l} X_l^i$, then substitute $x = X_i$ to get $\Lambda(X_i) = 0$ since error location polynomial has zeros at $X_i$.

$$e_{i_l} (\Lambda_0 X_i^l + \Lambda_1 X_i^{l+1} + \Lambda + \Lambda_{v-1} X_i^{l+v-1} + X_i^{l+v}) = 0$$  \hspace{1cm} (4.5)

Now sum equation 4.5 over all indices $l$ to obtain an expression that will reduce the system equations.
\[ \sum_{i=1}^{v} e_i (\Lambda_0 X_i + \Lambda_1 X_i^{j+1} + \Lambda + \Lambda_{v-1} X_i^{j+v-1} + X_i^{j+v}) \]

\[ = \Lambda_0 \sum_{i=1}^{v} e_i X_i^j + \Lambda_1 \sum_{i=1}^{v} e_i X_i^{j+1} + \Lambda + \Lambda_{v-1} \sum_{i=1}^{v} e_i X_i^{j+v-1} + \sum_{i=1}^{v} X_i^{j+v} \]

\[ = \Lambda_0 S_j + \Lambda_1 S_{j+1} + \Lambda + \Lambda_{v-1} S_{j+v-1} + S_{j+v} = 0 \]

Which can be written as:

\[ \Lambda_0 S_j + \Lambda_1 S_{j+1} + \Lambda + \Lambda_{v-1} S_{j+v-1} = -S_{j+v} \]  \hspace{1cm} (4.7)

Since we know the values of \( S_{b+1}, S_{b+2}, K, S_{b+2v} \), we can substitute \( j = b + 1, K, b + v \) in turn to obtain \( v \) linear equations in the unknowns \( \Lambda_0, \Lambda_1, K, \Lambda_{v-1} \). Using the matrix form, with \( A \) to represent syndrome matrix, we can express the equations as:

\[
A \Lambda = \begin{bmatrix}
S_{b+1} & S_{b+2} & \Lambda & S_{b+v} \\
S_{b+2} & S_{b+3} & \Lambda & S_{b+v+1} \\
M & M & M & M \\
S_{b+v} & S_{b+v+1} & \Lambda & S_{b+2v-1}
\end{bmatrix}
\begin{bmatrix}
\Lambda_0 \\
\Lambda_1 \\
\Lambda \\
\Lambda_{v-1}
\end{bmatrix} = 
\begin{bmatrix}
-S_{b+v+1} \\
-S_{b+v+2} \\
M \\
-S_{b+2v}
\end{bmatrix} \]  \hspace{1cm} (4.8)

This linear system (Equation 4.8) shows that a maximum of \( t \) error locations can be found by this equation since the maximum number of syndrome values available is bounded by the decoding capability that is \( 2t \) as defined. If less than \( t \) errors occur, then matrix \( A \) will be singular, thus the right most column and the bottom row are removed and the determinant of the resulting matrix will be computed. Once the error locations are known, the original system becomes \( 2v \) equations in \( v \) unknowns of the error magnitudes. The system can then be reduced to form a linear system of \( v \) error magnitudes \( e_i \) as the unknowns (equation 4.9).
The decoding process is finally complete by solving the \( \{e_i\} \) in the equation (4.9).

\[
Be = \begin{bmatrix}
X_{b+1} & X_{b+2} & \cdots & X_{b+n}
\end{bmatrix}
\begin{bmatrix}
e_i
\end{bmatrix}
= \begin{bmatrix}
S_{b+1}
S_{b+2}
\vdots
S_{b+n}
\end{bmatrix}
\] (4.9)

The decoding process is finally complete by solving the \( \{e_i\} \) in the equation (4.9).

### 4.2.2 Berlekamp-Massey Algorithm

The above decoding process is called Perterson-Gorenstein-Zieler algorithm. It is a standard algorithm to describe the syndrome based decoding of BCH code, but not suitable for implementation. Another algorithm called Berlekamp-Massey algorithm offers a much more efficient alternative for the correction of large number of errors. The major contribution of Berlekamp-Massey algorithm is its demonstration of the fact that the solving of error locator polynomial is equivalent to finding the connection of a set of linear feedback shift registers, which can be iteratively solved quickly.

To explain the Berlekamp-Massey algorithm, we must first go back to equation 4.7 in which the syndrome is expressed in recursive form as a function of error locator polynomial \( \Lambda(x) \) with coefficients the earlier syndromes \( S_1, S_2, \ldots, S_{j+n-1} \). The physical interpretation of the equation 4.7 is a linear feedback shift register (LFSR) system with the storage units as the syndrome values and the connections to specify the coefficients of the error locator polynomial. Now the solving of error locator polynomial is to find a linear feedback shift register system of minimum length such that the first \( 2t \) elements output sequence of LFSR are the syndromes. The tabs of the LFSR thus present the correct error locator polynomial \( \Lambda(x) \).
Berlekamp-Massey algorithm starts by determining the first order connection polynomial $A^{(1)}(x)$ that can generate syndrome $S_1$ as the first element output of LFSR. The second output of current LFSR connection is compared to the second syndrome. The discrepancy $\Delta^{(k)}$ between the two is used to construct a modified connection polynomial in the form of $A^{(k)}(x) = A^{(k-1)}(x) - \Delta^{(k)}T(x)$. $T(x)$ is the correction polynomial originally set at $x$. It will be modified each time by the current connection of LFSR $A^{(k)}(x)$ and discrepancy $\Delta^{(k)}$ to be used in the next run of search. The beauty of the correction polynomial is that it can ensure the first $k - 1$ output elements of the next run will always be the $k - 1$ syndromes. If there is no discrepancy in any stage, the same connection polynomial will be used to generate the next output sequence. The process continues until a correct connection polynomial is found that the specified LFSR can generate all $2t$ syndromes. The detail proofs of the validity of the algorithm can be found in reference [8]. In the communication toolbox of Matlab, there is a build-in function to solve the error locator polynomial using Berlekamp-Massey algorithm of which I have used in the simulation programs.

With the error locator polynomial, the next step is to find the error magnitudes. The following process shows how to get the error magnitude values by making the use of syndrome polynomial. Notice that error magnitude calculation is not part of Berlekamp-Massey algorithm, but a standard linear system calculation.

The $2t$ syndromes can be expressed as an infinite-degree syndrome polynomial as:
With the error locator polynomial $A(x)$ defined above, there exists a key equation for BCH/RS decoding that relates the known syndrome values to the error locator and error magnitude polynomials:

$$A(x) = 1 \cdot S(x) = 1 + \sum_{j=1}^{\infty} (\sum_{i=1}^{v} e_i \cdot X_i^j) \cdot x^j$$

$$= 1 + \sum_{i=1}^{v} e_i \sum_{j=1}^{\infty} (X_i \cdot x)^j$$

$$= 1 + \sum_{i=1}^{v} e_i \left(\frac{X_i \cdot x}{1 - X_i \cdot x}\right)$$

With the error locator polynomial $\Lambda(x)$ defined above, there exists a key equation for BCH/RS decoding that relates the known syndrome values to the error locator and error magnitude polynomials:

$$\Lambda(x)[1 + S(x)] = \Omega(x) \mod x^{2^{t+1}}$$

The error magnitudes can finally be solved using the Forney algorithm, which is expressed as:

$$e_k = -\frac{X_k^{-1} \Omega(X_k)}{\Lambda'(X_k)}$$

### 4.2.3 Erasure Decoding

Erasure decoding for BCH codes poses only small modifications to the standard algorithm and does not increase the decoding complexity. For binary erasure decoding, we need only try the standard decoding process twice. That is to place zeros in all erased locations and decode normally, then repeat the decoding process by placing ones in those erased coordinates and decode again. Comparing the decoded words with the received word, the one with the minimum Hamming distance is chosen as the correctly decoded word. This binary erasure-decoding algorithm is based on the fact that there are only two
possibilities of the erased symbol magnitude, one or zero. So, one of them must be correct.

In non-binary erasure decoding, we define an erasure locator polynomial as:

\[ \Gamma(x) = \prod_{i=1}^{f} (Y - Y_i) \]  

(4.13)

Compared to the errors with unknown locations as well as magnitudes, erasures already provide the information about their locations. That information can then be used to modify the syndrome values in order to solve the true error location polynomial. After that, the error magnitudes \( e_k \) and erasure magnitudes \( f_{j_k} \) can be determined from the combined error and erasure locator polynomials and the known syndrome values. The key equations for error and erasure decoding now becomes:

\[ \Lambda(x) \Gamma(x) [1 + S(x)] \mod x^{2t+1} \]  

(4.14)

And the modified syndrome polynomial from equation 4.13 is:

\[ \Xi(x) = \Gamma(x) [1 + S(x)] \mod x^{2t+1} - 1 \]  

(4.15)

Note that erasure locator polynomial \( \Gamma(x) \) is constructed according to erasure locations at the beginning of decoding process. The erased symbols are then set to zeros to reduce the affect on syndrome calculation. With the modified syndrome polynomial \( \Xi(x) \) ready by using the equation 4.15, we can now use Berlekamp-Massey algorithm to solve the error location polynomial \( \Lambda(x) \). Applying Forney's algorithm, we can then solve error and erasure magnitudes. The procedure of erasure and error decoding algorithm for BCH codes are as follows:
1. Examine to see if there are any erasures in the received word. If the answer is yes, construct the erasure location polynomial \( \Gamma(x) \) and substitute zeros into those erased symbols. If no erasures exist, go to error only decoding part.

2. Calculate \( 2t \) syndromes \( \{S_0, K, S_{b+2t-1}\} \) by taking \( 2t \) consecutive powers of \( \alpha \) (starting from \( b \)) into the received word polynomial. (\( \alpha \) is the primitive element in extension field \( GF(q^m) \) where \( n \)-th root of unity can be found)

3. Using equation 4.15 to calculate the modified syndromes.

4. Apply Berlekamp-Massery algorithm with the modified syndromes as the input to obtain the error location polynomial \( \Lambda(x) \).

5. Solving the error location polynomial \( \Lambda(x) \) to get the error locations.

6. Apply Forney algorithm to solve the error and erasure magnitudes. Note that erasure magnitude are solved exactly the same as the error magnitude.

**4.2.4 Simulation Consideration**

In order to simulate non-binary BCH codes, we use the newest professional version of Matlab that contains the communication toolbox, where Galois filed calculation functions can be found. However, binary code is still Matlab’s major consideration, certain Galois field functions were written only for the binary case, that they have to be modified to work in non-binary situation. Another major work is to add erasure feature into the BCH decoding for binary as well as non-binary (includes Reed-Solomon code) cases.

It must be noted that the BCH decoder simulated only work on the \( q \)-ary codes such that \( q \) is certain power of 2, but not any other prime number. The reason comes from the
matlab function \texttt{gfconv} which performs the convolution operation only in $GF(2^n)$. It isn’t designed to work on $GF(q^n)$ where q is a prime other than 2, thus any q-ary code whose n-th root of unity reside in a extension field $GF(q^n)$ can not be decoded. The encoding part doesn’t involve the using of function \texttt{gfconv} so that any legal BCH codes can be generated.

As a hard-decision decoder, the simulation result proves the fact that the 4-ary BCH codes can correct any combination of errors and erasures as long as they meet the decoding constraint $2e + f < d_{\min}$ where minimum distance $d_{\min}$ is equivalent to the erasure correcting capability plus one. For error patterns that do not contain erasures, if $d_{\min} = 2t + 1$, the code is guaranteed to correct $\left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor = t$ errors. It should be pointed out that true minimum distance might exceed the design distance, thus the decoding capability of a specific code need to investigate its true $d_{\min}$.

Since the non-binary BCH codes can not always be narrow-sense, the starting power of $\alpha$ as the zero of generating polynomial must be transmitted as a code parameter from the encoder to the decoder. The decoder will then calculate syndrome value by taking $\alpha^b,K,\alpha^{b+2i-1}$ into the received word polynomial. Please note that $\alpha^b,K,\alpha^{b+2i-1}$ must be expressed as the elements in extension field $GF(q^n)$ where n-th root of unity exists since all calculations can only be performed in that field. Equation 4.16 relates the two Galois fields, namely $GF(n)$ who has $\alpha$ as the primitive element and $GF(q^n)$ has $\gamma$ as the primitive element.
\[ \alpha = \gamma^n : q \text{ Symbol field, } q\text{-ary} \]  

(4.16)

It’s better to show the simulation result in the plot of bit error probability versus the Signal to Noise Ratio (SNR) expressed as \( \frac{E_b}{N_0} \). However, this decoding curve needs to simulate the whole communication system with huge amount of input bits. And the purpose of investigating non-binary BCH code here is not to have it directly used in coding/modulation scheme but as a part of the decoding scheme for HDTCM where the decoding performance can benefit from the concatenated system which contains BCH codes as the outer code.

In summary, decoding a received BCH code word requires execution of three successive computational processes [9] with all computations performed over the extension field \( GF(q^n) \). These processes are syndrome calculation, solution of key equation using Berlekamp-Massey algorithm and error magnitude calculation for non-binary case. Since the computation burden is concentrated on the \( GF(q^n) \) calculation, especially the multiplication operation in \( GF(q^n) \), the manner of which multiplication in \( GF(q^n) \) are implemented is a very important part of the design optimization process. Although the current simulation doesn’t involve hardware implementation, calculation over huge extension field \( GF(q^n) \) is still impossible. Actually, any 4-ary BCH codes with \( n \)th root of unity residing in the extension field larger than \( GF(4^8) \) can not be implemented. There will be error information as “out of memory” and the program will be terminated when this fatal error occurs.
4.3 Application of Non-Binary BCH Code in HDTCM.

The reason of investigating the non-binary BCH codes in our research is to test and verify a combined decoding scheme that can improve the HDTCM system performance. First, the concatenated coding scheme will be introduced as a basis. We then present the description of proposed decoding scheme for HDTCM system.

Normally, a concatenated system combines Reed-Solomon codes and convolutional codes because the two codes have their own advantages and restraints, which can complement each other [7]. Reed-Solomon code is a special kind of non-binary BCH codes; thus many of its best features also belong to the non-binary BCH codes. Those advantages include the Berlekamp-Massey algorithm that can be implemented with shift registers, the efficient bounded distance decoding algorithm and the burst-error-correcting capability for Reed-Solomon codes. The only disadvantage of BCH/RS codes lies in the lack of efficient maximum likelihood soft-decision decoding algorithm, which can provide 2 dB more coding gain than the hard-decision decoding algorithm in white Gaussian channels.

For convolutional codes with Viterbi decoding, soft decisions are incorporated in a natural way. However there are also restraints with convolutional codes, such as its tendency to generate burst errors at the output for the high noise level. The combination of the two codes leads to the concatenated system where the convolutional codes (with soft-decision Viterbi decoding) are used before the Reed-Solomon codes in the decoding process. The reason is that convolutional codes can easily accept soft decisions and channel information from the real channels while Reed-Solomon codes can be used to clean up the errors left over. This concatenated system has been selected by the
Consultative Committee for Space Data Systems (CCSDS) of NASA for deep space mission because of its proven coding gain.

The system can be further improved by modifying the Viterbi decoder to generate erasures at the input of the Reed-Solomon decoder and have its result feedback to the inner Viterbi decoder for another round of Viterbi decoding. The probability of correct decoding for each individual bit at the output of Reed-Solomon decoders need to be defined and calculated as the soft information for RS codes. When the correct estimates from the outer decoder is feed back to the inner decoder; some burst errors can be correctly eliminated since the Viterbi decoder can then follow the more accurate decoding path. Another technique that can be applied comes from the erasure decoding of Reed-Solomon code. The bursty output from inner Viterbi decoder can be transformed into the erasures rather than the burst errors, that the error correction capability of Reed-Solomon codes can be increased since the number of erasures can be corrected is two times the number of errors. The problem is that where to place the erasures and the placing criteria require the symbol reliability information from the output of Viterbi decoder. Because the limited number of available RS codes restrict its combination with convolutional codes which has wide selections on block length and a number of states. Non-binary BCH codes is investigated to replace RS codes when necessary. The 4-ary BCH codes in this thesis can readily combine with 4-states Trellis with variable block length selections.

An interleaver is a device that rearranges the ordering of a sequence of symbols in a deterministic manner. Associated with an interleaver is a deinterleaver that applies the inverse permutation to restore the symbol sequence. The role of interleaver in
concatenated system is to spare out the burst error result from the Viterbi decoder. After interleaving, the burst errors from the inner decoder will be spared out as random errors for the Reed-Solomon decoder.

The block diagrams [7] for the standard concatenated system and the improved systems are shown in figure 4.1 and 4.2. Notice that Reed Solomon code is used in standard concatenated system for its potential capability to correct burst error bits in a symbol. And BCH codes are placed in the improved system to make it fit our special requirement on HDTCM system.

![Figure 4.1 Standard concatenated coding system](image)
Figure 4.2 Improved concatenated coding system
CHAPTER 5
CONCLUSIONS

5.1 Summary

This thesis research simulates the non-binary BCH codes with the erasure decoding capability and develops algorithms for searching the better codes within a code family. Certain encoding and decoding algorithms are implemented with modifications incurred by the non-binary case of source symbol alphabet size. The resulting BCH codec can be used directly in future research work to combine with the convolutional codes. Only the case of 4-ary BCH codes are tested and analyzed in this research, however, simulation on any other source alphabet size can be conducted readily since the programs are written for general purpose with ready to change variables.

Chapter 2 provides the necessary mathematical background on Galois fields with emphasis on the polynomials over such fields and the concepts of error control coding. TCM and HDTCM are introduced as the applications of this research.

Cyclic codes are introduced in chapter 3 as the derivation from the special case of polynomials over Galois fields. The encoding of BCH codes is explained in detail on the basis of cyclic code characteristics. Concepts such as minimal polynomials, root of unity and BCH design distance constitute the fundamentals of BCH codes. Because of the inequality of BCH bound, algorithm for obtaining better codes within the same code family (same code length) is developed and the resulting generator polynomials will not always belong to the narrow-sense case. With the generating polynomials, the encoder
simulation is carried out using the standard matrix multiplication of the source data matrix and the generating matrix. Simulation results for 4-ary BCH codes are presented and analyzed to show the existence of certain better performance codes such as the 4-ary, 64-length primitive BCH codes. The comparisons of this primitive 4-ary, 64-length BCH codes with other lengths 4-ary BCH codes and the 64-length Reed-Solomon codes are also included.

Decoding of BCH codes is explained in chapter 4 where the popular Berlekamp-Massey algorithm is a focus. Simulation of the decoder calculates the syndromes from the received polynomial at first, then applies the Berlekamp-Massey algorithm to determine the error location polynomial. Erasure correction capability is incorporated with the output location polynomial that combines errors and erasures. Error and erasure magnitudes are finally determined by using the information from the syndrome polynomial as well as the error locator polynomial, which forms a key equation for BCH decoding. The non-narrow-sense nature of the simulated BCH codes result in transferring one more parameter from encoder to decoder than the standard algorithm. Programming consideration is presented where certain restraint is explained. The proposed application of non-binary BCH codes in the decoding of HDTCM is to combine with the Viterbi decoder. The hard-decision nature of BCH codes ensures the decoding performance but results in a lack of soft-decision information, which can be a potential match with the Viterbi decoder for HDTCM scheme.
5.2 Future Research

*Burst error correction:* For the purpose of using BCH codes as the outer decoder, the inner Viterbi decoder will benefit much more if BCH can dealing with the burst errors generated by Viterbi decoder. Possible solutions include finding a way of transferring errors to erasures thus error correction capability can be increased.

*Combination with inner Viterbi decoder:* The immediate next step is to combine the BCH codes with the convolutional codes that use the Viterbi decoder. More research work needs to be conducted on the interleaver design, soft information from BCH decoder to control more runs of decoding process and feedback structure of the concatenated system.

*Test of other alphabet size of BCH code:* The trellis structure of HDTCM can have higher source data alphabet size. That’s why the test result of other symbol alphabet size is needed to analysis the code performance.
References


Appendix A

List of some lengths 4-ary BCH codes

The tested code lengths are 15, 17, 21, 31, 33, 39, 43, 51, 63, 65, 85, 127, and 255.

For code lengths less than 65, their associated generator polynomials are also listed in the form of 3 digits of 4-ary in which the 4-ary elements come from the extension Galois field and mapped to the ring of integer modulo 4 as \{0, 1, 2, 3\}. For example: the generator polynomial for \((21,20)\) 4-ary BCH code is \(x + \beta^{21} = x + 2\), which expressed here as \{24\} that equals to \{120\}. The first digit corresponds to the highest degree of the polynomial, and the zeros from the lowest degrees should be truncated.

<table>
<thead>
<tr>
<th>(n)</th>
<th>(k)</th>
<th>(r2)</th>
<th>Generator polynomials (g(x))</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>14</td>
<td>1</td>
<td>{24}</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>2</td>
<td>{28,48}</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>3</td>
<td>{19,11}</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>4</td>
<td>{29,26,16}</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>5</td>
<td>{20,49,40}</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>6</td>
<td>{19,55,49}</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>7</td>
<td>{21,15,40,24}</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>9</td>
<td>{20,47,29,7}</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>10</td>
<td>{27,41,51,52,32}</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>14</td>
<td>{21,21,21,21,21}</td>
</tr>
<tr>
<td>17</td>
<td>13</td>
<td>1</td>
<td>{29,52}</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>2</td>
<td>{23,20}</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>3</td>
<td>{24,42,9}</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>6</td>
<td>{28,63,13}</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>8</td>
<td>{25,58,26,54,16}</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>15</td>
<td>{21,21,21,21,21}</td>
</tr>
<tr>
<td>21</td>
<td>20</td>
<td>1</td>
<td>{24}</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>2</td>
<td>{30,24}</td>
</tr>
<tr>
<td>14</td>
<td>3</td>
<td>{16,27,24}</td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>{21,5,1,16}</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>5</td>
<td>{19,8,24,8}</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>{26,35,52,45,16}</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>{17,31,34,32,56}</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>{27,12,29,12,57}</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>{22,53,31,35,60,9}</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>{24,24,24,24,24,24,24}</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>20</td>
<td>{21,21,21,21,21,21,21}</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>31</th>
<th>26</th>
<th>2</th>
<th>{21,17}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>21</td>
<td>4</td>
<td>{16,20,0,20}</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>6</td>
<td>{17,20,17,5,17,16}</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>10</td>
<td>{20,1,20,5,5,0,17}</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>14</td>
<td>{20,20,5,20,16,17,5,16,4}</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>29</td>
<td>{21,21,21,21,21,21,21,21,16}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>32</th>
<th>32</th>
<th>1</th>
<th>{24}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>27</td>
<td>2</td>
<td>{28,33}</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>5</td>
<td>{19,3,16,19}</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>7</td>
<td>{28,63,12,51,60,56}</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>8</td>
<td>{20,26,39,22,46,21,34,32}</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>9</td>
<td>{24,3,0,27,16,8,3,16}</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>11</td>
<td>{18,55,63,61,50,26,42,38,49}</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>21</td>
<td>{24,24,24,24,24,24,24,24,24}</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>32</td>
<td>{21,21,21,21,21,21,21,21,21}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>35</th>
<th>33</th>
<th>1</th>
<th>{29}</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30</td>
<td>2</td>
<td>{31,9}</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>3</td>
<td>{27,19,31}</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>4</td>
<td>{29,60,51,5}</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>5</td>
<td>{26,27,24,14,6,16}</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>6</td>
<td>{16,9,46,47,12,49}</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>7</td>
<td>{21,27,11,13,43,53}</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>9</td>
<td>{20,27,60,42,16,49,33}</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>11</td>
<td>{26,37,11,7,41,8,21,27,1}</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>12</td>
<td>{22,24,46,8,18,29,59,51,13,52}</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>13</td>
<td>{27,34,56,51,22,35,23,22,62,13}</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>14</td>
<td>{17,16,4,20,1,5,0,17,16,4,20}</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>{25,1,36,6,16,25,1,36,6,16,25}</td>
<td></td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>---------------------------------</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>34</td>
<td>{21,21,21,21,21,21,21,21,21,21,20}</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>1</td>
<td>{24}</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td>{21,47,8}</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>3</td>
<td>{16,8,8,3,24}</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>5</td>
<td>{29,1,4,5,1,5,10,16}</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>6</td>
<td>{20,40,22,55,23,41,40}</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>14</td>
<td>{27,16,11,24,27,24,3,27,8}</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>12</td>
<td>{27,32,28,28,17,56,18,18,16,37,16}</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>13</td>
<td>{16,60,21,21,25,31,26,42,40,43,56}</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td>{24,50,16,0,6,12,36,0,1,35,9}</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>25</td>
<td>{24,24,24,24,24,24,24,24,24,24,24}</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>38</td>
<td>{21,21,21,21,21,21,21,21,21,21,21}</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>1</td>
<td>{18,23,4}</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>3</td>
<td>{31,58,36}</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>4</td>
<td>{26,18,0,49,61}</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>5</td>
<td>{29,52,50,6,25}</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>22</td>
<td>{24,51,43,48,43,56,35,16}</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>9</td>
<td>{29,47,55,46,4,19,46,26,45,36}</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>11</td>
<td>{17,36,10,46,41,63,47,48,29,4}</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>13</td>
<td>{17,36,10,46,41,63,47,48,29,4}</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>42</td>
<td>{21,21,21,21,21,21,21,21,21,21,16}</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>44</td>
<td>{24}</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>2</td>
<td>{23,41}</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>3</td>
<td>{22,61,0,61}</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>4</td>
<td>{16,36,60,48,60}</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>5</td>
<td>{28,44,48,0,28,48}</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>6</td>
<td>{28,20,40,32,28,8,32}</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>7</td>
<td>{19,42,63,6,19,57,6}</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>8</td>
<td>{16,20,16,52,56,4,52,24,60}</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>9</td>
<td>{19,57,39,44,63,39,13,33,13}</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>10</td>
<td>{20,13,35,46,18,55,6,20,18,60}</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>11</td>
<td>{20,37,37,31,37,20,31,31,58,31,20}</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>{20,25,6,11,13,18,18,11,13,6,31,20}</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>13</td>
<td>{26,59,33,50,19,59,9,50,19,33,40,40}</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>14</td>
<td>{29,26,16,0,0,29,26,16,0,0,29,26,16}</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>17</td>
<td>{20,49,40,0,0,20,49,40,0,0,20,49,40}</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>{19,55,49,0,0,19,55,49,0,0,19,55,49}</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>23</td>
<td>{21,15,40,24,0,21,15,40,24,0,21,15,40,24}</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>29</td>
<td>{20,47,29,7,0,20,47,29,7,0,20,47,29,7}</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>32</td>
<td>{27,41,51,52,32,27,41,51,52,32,27,41,51,52,32}</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>44</td>
<td>{21,21,21,21,21,21,21,21,21,21,21,21,21,21,21}</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>1</td>
<td>{24}</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>2</td>
<td>{20,53}</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>3</td>
<td>{19,8,0,16}</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>4</td>
<td>{27,4,63,42,16}</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>5</td>
<td>{20,8,7,23,56}</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>6</td>
<td>{26,32,34,42,5,58}</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>7</td>
<td>{20,5,43,5,2,5,60,48}</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>9</td>
<td>{29,37,23,0,10,30,45,16}</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>10</td>
<td>{18,24,59,23,59,46,31,41,36}</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>22</td>
<td>{26,4,21,28,33,22,38,20,29,35}</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>16</td>
<td>{27,40,40,14,22,53,22,28,11,11,42}</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>17</td>
<td>{16,55,39,15,44,3,60,37,8,24,53,48}</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>{24,61,21,52,26,2,2,36,29,21,60,36}</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>19</td>
<td>{20,1,40,50,9,45,8,43,49,12,59,23,51}</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>21</td>
<td>{27,43,58,27,33,59,20,3,35,4,45,1,0,26}</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>22</td>
<td>{19,48,56,61,25,38,52,8,25,26,59,33,8,59}</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>33</td>
<td>{18,56,43,18,33,34,39,48,58,36,44,36,27,35,52,16}</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>34</td>
<td>{30,33,46,62,53,22,48,24,51,51,42,35,22,7,31,28}</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td>{21,21,21,21,21,21,21,21,21,21,21,21,21,21,21}</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>62</td>
<td>{24}</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>2</td>
<td>{23,4}</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>3</td>
<td>{19,0,24}</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>4</td>
<td>{19,17,30,16}</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>5</td>
<td>{23,35,30,24}</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>6</td>
<td>{17,11,19,62,60}</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>7</td>
<td>{29,43,52,7,24,60}</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>8</td>
<td>{31,41,46,54,32,7,16}</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>9</td>
<td>{29,16,34,43,22,8,40}</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>10</td>
<td>{30,34,51,12,38,42,49,56}</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>11</td>
<td>{19,42,60,4,8,23,42,44,52}</td>
<td></td>
</tr>
</tbody>
</table>
The following codes of length larger than 63 have their code dimension $k$ and erasure correction capability $t_2$ listed out only.

<table>
<thead>
<tr>
<th>n</th>
<th>Code dimension $k$</th>
<th>Erasure correction capability $t_2$</th>
<th>Code dimension $k$</th>
<th>Erasure correction capability $t_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>12</td>
<td>{31,11,39,9,20,12,15,55,16,16}</td>
<td>29</td>
<td>12</td>
</tr>
<tr>
<td>35</td>
<td>13</td>
<td>{24,29,32,50,51,2,37,59,36,40}</td>
<td>30</td>
<td>12</td>
</tr>
<tr>
<td>34</td>
<td>14</td>
<td>{28,28,24,47,51,20,2,19,12,29,36}</td>
<td>43</td>
<td>12</td>
</tr>
<tr>
<td>29</td>
<td>15</td>
<td>{28,56,60,3,6,5,42,16,45,25,2,56}</td>
<td>19</td>
<td>12</td>
</tr>
<tr>
<td>27</td>
<td>20</td>
<td>{20,49,28,35,26,21,43,29,20,34,24,51,16}</td>
<td>20</td>
<td>12</td>
</tr>
<tr>
<td>26</td>
<td>21</td>
<td>{30,53,53,47,1,47,4,4,62,46,35,55,8}</td>
<td>25</td>
<td>12</td>
</tr>
<tr>
<td>25</td>
<td>32</td>
<td>{31,28,44,56,51,56,59,11,11,14,13,61}</td>
<td>22</td>
<td>12</td>
</tr>
<tr>
<td>22</td>
<td>23</td>
<td>{19,30,1,56,46,54,39,56,40,12,30,10,21,3}</td>
<td>20</td>
<td>12</td>
</tr>
<tr>
<td>20</td>
<td>25</td>
<td>{30,54,13,36,9,40,8,3,10,24,56,60,33,15,24}</td>
<td>19</td>
<td>12</td>
</tr>
<tr>
<td>19</td>
<td>26</td>
<td>{22,5,62,35,27,23,15,32,28,19,44,55,22,45,61}</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>16</td>
<td>27</td>
<td>{16,59,21,57,23,20,45,52,40,16,63,44,1,24,58,42}</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>29</td>
<td>{25,32,39,55,7,9,34,35,14,61,57,18,20,52,63,40,4}</td>
<td>13</td>
<td>16</td>
</tr>
<tr>
<td>13</td>
<td>30</td>
<td>{28,34,1,21,17,44,63,34,25,52,37,53,45,36,43,5,58}</td>
<td>10</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>31</td>
<td>{31,31,39,27,59,48,49,52,60,51,32,46,37,36,62,54,1,46}</td>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>41</td>
<td>{28,39,58,3,18,62,10,59,9,1,19,22,18,38,11,2,23,60,40}</td>
<td>7</td>
<td>22</td>
</tr>
<tr>
<td>7</td>
<td>42</td>
<td>{18,32,19,19,62,36,27,34,40,49,47,7,14,49,26,34,8,22,45}</td>
<td>4</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>46</td>
<td>{31,29,46,19,58,30,63,5,4,38,13,26,6,21,48,51,34,52,10,50}</td>
<td>1</td>
<td>26</td>
</tr>
<tr>
<td>1</td>
<td>62</td>
<td>{21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21,21}</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>73</td>
<td>5</td>
<td>29</td>
<td>22</td>
</tr>
<tr>
<td>---</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td>69</td>
<td>6</td>
<td>27</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>7</td>
<td>23</td>
<td>29</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>9</td>
<td>23</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>10</td>
<td>19</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>11</td>
<td>19</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>12</td>
<td>17</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>13</td>
<td>15</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>14</td>
<td>11</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>15</td>
<td>7</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>16</td>
<td>5</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>84</td>
</tr>
<tr>
<td>120</td>
<td>2</td>
<td>57</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>4</td>
<td>50</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>6</td>
<td>43</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>8</td>
<td>36</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>10</td>
<td>29</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>12</td>
<td>22</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>14</td>
<td>15</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>18</td>
<td>8</td>
<td>62</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>20</td>
<td>1</td>
<td>125</td>
<td></td>
</tr>
</tbody>
</table>
Appendix B
Generating generator polynomial for non-binary BCH codes

function [pg, k, pm, cs, t2, b] = bchpoly_m1(n,t2,ary)
%BCHPOLY Produces BCH codeword length, message length, and
%   error-correction capability for non-binary BCH codes.
%   pg: Generating polynomial
%   k: Message length
%   pm: Minimal polynomials
%   cs: Cyclotomic cosets modulo n over GF(ary)
%   t2: Erasure correcting capability. T2 as input is designed decoding
%       Capability while t2 as output is the true decoding capability.
%   b: Starting power of nth root of unity as the roots for pg
% [Pg, k, pm, cs, t2, b] = BCHPOLY_M1(N, T2, ARY) outputs code
% parameters for non-binary BCH codes.
% [Pg, k, Pm, cs, t2, b] = BCHPOLY_M1(N, T2) outputs codes in GF(2^M).
% Modified 5.24.99 by Biyun Zhou

% give a list of all binary BCH code
% code word length / message length / error-correction capability

n_lst = [3 7 1; 4 15 3; 5 31 5; 6 63 11; 7 127 17; 8 255 33; 9 511 57; 10 1023 105];

% known error-correction capability
t_lst = [1:3,...
   [1:3],5 7,...
   [1:7],10 11 13 15,...
   [1:7], 9,10 11 13 14 15 21 23 27 31,...
   [1:15], 18 19 [21:23],[25:27],[29:31],42 43 45 47 55 59 63,...
   85 87 91 93 95 109 111 119 121,...
if nargin < 1
   error('not enough input parameters');
end;

%digits % of Galois field
if nargin < 3
   p = 2;
   ary = 2;
end;
i = 1;
while rem(ary^i-1,n)~=0
   if i>20
      error('not valid code length when p = 2');
   end;
i = i+1;
end;
if nargin < 3
   dim = i;
   dim_fld = 2^dim-1;
   if nargin < 2
      disp('use default primitive polynomial');
      k = n - dim;
   end;
else
   fac = factor(ary);
   if max(fac) ~= min(fac)
      error('GF(ary) is not a exist valid Galois field');
   end;
   p = fac(1);
pow = i;
dim_fld = ary^pow-1;
dim = log2(dim_fld+1)./log2(p);
tp = gftuple([-1:dim_fld-1],dim,p);
end;
m = dim_fld./n;
k_init = n - t2;
if (dim<1) | (k_init<1) | (floor(dim) ~= dim) | (floor(k_init) ~= k_init)
   error('Input variable for BCHPOLY is not valid');
end

prime_flag = 0;
numerr_flag = 0;
% compute the minimum polynomial
if k_init == n - dim & ary == p & m == 1
  % It is a trivial case, simply assign the primitive polynomial
  % as BCH polynomial.
  mp = gfprimdf(dim,p);
  pm = mp;
  pg = mp;
  t = 1;
  if (nargout > 2)
    cs = gfcoset_m(dim,ary,n);
  end;
  prime_flag = 1;
  k = k_init;
else
  % list cosets and taking off the first row.
  if ary == p
    cs = gfcoset_m(dim,p,n);
    [n_cs, m_cs] = size(cs);
    pl = gfminpol_m1(cs(1:n_cs,1),dim,p,tp,n);
  else
    cs = gfcoset_m(pow,ary,n);
    [n_cs, m_cs] = size(cs);
    pl = gfminpol_m1(cs(1:n_cs,1),pow,ary,tp,n);
  end;
  prim_flag = 1;
  index = [];
  n_terms = [];
  n_i = 0;
  n_end = [];
  pm = [];
  rs = [];
for j = 1:k_init
  n_terms(j) = 0;
  index(j,1) = 0;
  ff = 1;
  rs(j) = 0;
for n_i = j : j+n-k_init-1
  count = rem(find(cs == n_i),n_cs);
  if count == 0
    count = n_cs;
  end;
  rs(j) = rs(j) + 1;
  if isempty(find(count == index(j,:)))
    n_terms(j) = n_terms(j) + sum(~isnan(cs(count,:)));
index(j,ff) = count;
ff = ff+1;
end;
if rs(j) >= t2
break;
end;
end;
match = find(rs == t2);
if ~isempty(match)
    [y, id] = min(n_terms(match));
    n_th = id;
    numerr_flag = 1;
else
    [y, id] = min(n_terms);
    n_th = id;
end;
len = length(find(index(n_th,:)>0));

kk = 1:len;
pm(kk,:) = pl(index(n_th,kk,:));

if isprime(ary)
    pg = gftrunc(pm(1,:));
else
    pg = pm(1,:);
end;
[n_pm, m_pm] = size(pm);
if n_pm > 1
    if ~isprime(ary)
        for i = 2 : n_pm
            pg = gfconv(pg, pm(i,:),tp);
        end;
    else
        for i = 2 : n_pm
            pg = gfconv(pg, gftrunc(pm(i,:)),p);
        end;
    end;
end;
loc = find(pg == -inf);
pg(loc) = -1;
rr = length(pg)-1;
k = n-rr;
str = sprintf('We generated a (%d,%d) BCH code on %d-ary',n,k,ary);
disp(str);

% detect error-correction capability.
if nargout > 4

if prime_flag & (dim <= 10) & ary == 2 & m == 1
    % look up the data base
    ind = find(n_lst(:,1) < dim);
    if ~isempty(ind)
        base_i = sum(n_lst(ind, 3));
    else
        base_i = 0;
    end;
    t = t_lst(base_i + n_i);
else
    if numerr_flag
        t = t2;
    else
        t = 0;
        delta = n_th;
        conj_matrix = [];
        tt = length(find(index(n_th,:)) > 0));
        for i = 1 : tt
            conj_matrix = [conj_matrix cs(index(n_th,i,:))];
        end;
        conjs = length(conj_matrix);
        loops = 1;
        while loops < conjs
            if ~isempty(find(conj_matrix==delta))
                t = t+1;
                delta = delta + 1;
            end;
            loops = loops + 1;
        end;
        tt = tt + 1;
    end;\n    end;

b = n_th;

end;

s = sprintf('which has %d erasure correcting capability',floor(t2));
disp(s);
Appendix C  
Decoding of BCH Codes

function [msg, err, ccode] = bchcore_m(code, pow_dim, dim, k, t2, tp, b);
%BCHCORE The core part of the BCH decode.
% [MSG, ERR, CCODE] = BCHCORE(CODE, POW_DIM, DIM, K, T2, TP, B)
% decode a BCH code, in which CODE is a code word row vector, with its column
% size being POW_DIM. POW_DIM equals 2^DIM - 1. K is the message length,
% T is the error correction capability. TP is a complete list of the
% elements in GF(2^DIM). B is the starting power of nth root of unity
% as the zeros of generating polynomial.
%
% This function can share information between a SIMULINK file and
% MATLAB functions. It is not designed to be called directly. There is
% no error checking in order to eliminate overhead. This function is
% modified from the original version of err1op.m which is a Matlab’s
% function in Communication Toolbox. The major addition is erasure
% decoding process and non-narrow-sense decoding process.
%
% Modified: 5.26.99 by Biyun Zhou

code=code(:)';
is_eras = 0;
good_f = 0;
x = factor(pow_dim+1);
if length(x) ~= dim
    error('code dimention is wrong');
else
    base = x(1);
end;
t = floor((t2-1)/2);
 tp_num = tp * base.^[0:dim-1]';
 tp_inv(tp_num+1) = 0:pow_dim;

err = 0;
% (1) find syndrome
n_code = length(code);
m = pow_dim/n_code;
% Handle with erasure case
erasure = find(code >= pow_dim);
%disp(erasure);
code(erasure) = -1;
erasure = m*erasure - m;
era_num = length(erasure);
if era_num > t2
    error('error number exceed the bch code error correcting capability');
end;

if era_num > 0
    era_locp = [erasure(1) 0];
    for l = 1:era_num -1;
        temp = [erasure(l+1) 0];
        era_locp = gfconv(era_locp, temp, tp);
    end;
    is_erasure = 1;
    sy_i = m*b:m:m*(b+t2-1);
    for loc = 1:t2
        % The ith element of syndrome equals the result of dividing code(X) by alpha^i+X
        [tmp, syndrome(loc)] = gfdeconv(code, [sy_i(loc), 0], tp);
    end;
    good_syn = find(syndrome ~= -inf);
    if isempty(good_syn)
        ccode = code;
        good_f = 1;
    else
        ss = -ones(1,b+t2);
        ss(b+1:b+t2) = (syndrome(1:t2));
        ss(1) = 0;
        if era_num < t2
            era_syn1 = ones(1,t2 - era_num).*(-1);
            for i = 1 : t2-era_num
                era_syn = gfmul(era_locp, syndrome(i:i+era_num), tp);
                for j=1 : length(era_syn)
                    era_syn1(i) = gfadd(era_syn1(i), era_syn(j), tp);
                end;
            end;
        end;
        [era_sigma, err] = errlocp(era_syn1, t, tp, pow_dim, err, 1);
        era_locp = flipr(era_locp);
        sigma = gfconv(era_sigma,era_locp,tp);
    else
        sigma = flipr(era_locp);
    end
end

%err = err + era_num;
% (2) find error location polynomial
% For non-binary case, using Berlekamp's iterative method to do the computation.
else
    sy_i = m*b:m*(b+t2-1);
    for loc = 1:t2
% The ith element of syndrome equals the result of dividing code(X) by alpha^i+X
        [tmp, syndrome(loc)] = gfdeconv(code, [sy_i(loc), 0], tp);
    end;
    ss = -ones(1,b+t2);
    ss(b+1:b+t2) = syndrome(1:t2);
    ss(1) = 0;
    [sigma, err] = errlocp_m(syndrome, t, tp, pow_dim, err, 1,is_eras,0);
end;

% (3) find solution for the polynomial
loc_err = zeros(1, n_code) - Inf;
num_err = length(sigma) - 1;
if is_eras == 0
    if num_err > t
% in the case of more errors than possibly correction.
        err = 1;
    end;
else
    if 2*(num_err - era_num)+era_num > t2;
        err = 1;
    end;
end;
if (~err) & (num_err > 0&(good_f == 0)
cnt_err = 0;
pos_err = [];
er_i = 0;
while (cnt_err < num_err) & (er_i < pow_dim * dim)
    test_flag = sigma(1);
    for er_j = 1 : num_err
        if sigma(er_j + 1) >= 0
% The following 6 lines is equivalent to
            % tmp = gfmul(er_i * er_j, sigma(er_j+1), tp);
            tmp = er_i * er_j;
            if (tmp < 0) | (sigma(er_j+1) < 0)
                tmp = -1;
            else
                tmp = rem(tmp + sigma(er_j + 1), pow_dim);
            end;
        end;
    end;
end;
test_flag = gfplus(test_flag, tmp, tp_num, tp_inv);

end;
end;
if test_flag < 0
  cnt_err = cnt_err + 1;
  pos_err = [pos_err, rem(pow_dim-er_i, pow_dim)];
end;
er_i = er_i + 1;
end;

pos_err = rem(pow_dim+pos_err, pow_dim);
pos_err = sort(pos_err);
%pos_err = pow_dim - pos_err;
pos_err = pos_err + m; % shift one location because power zero is one.
%loc_err(pos_err) = ones(1, cnt_err);
disp(pos_err);
err = num_err;
else
  if err
    err = -1;
  end;
end;

% (4) find the amplitude of the error

if (err > 0)&(good_f == 0)
  % Construct Z(X) in (6.34)
  Z_full = gfconv(ss, sigma, tp);
  if length(Z_full) >= b+num_err
    Z = Z_full(1:b+num_err);
  else
    Z = Z_full;
  end;
  len_z = length(Z);
pos_err_1 = pos_err-m; % considering position starting from zero.
er_loc = [1];
for am_i = 1 : length(pos_err_1)
  num = 0;
  den = 0;
pos_err_inv = rem((pow_dim - pos_err_1(am_i)), pow_dim);
for am_j = 1:len_z-1
  %num = gfadd(num, gfmul(Z(am_j + 1), pos_err_inv * am_j, tp), tp);
tmp = pos_err_inv * am_j;
if (tmp < 0) | (Z((am_j+1)) < 0)
  tmp = -1;
else
tmp = rem(tmp + Z(am_j+1), pow_dim);
end;
num = gfplus(num, tmp, tp_num, tp_inv);
if am_i == am_j & am_j <= num_err
    den = gfmul(den, gfadd(0, gfmul(pos_err_1(am_j), pos_err_inv, tp), tp), tp);
    if (den < 0)
        den = -1;
    else
        if (pos_err_1(am_j) < 0) | (pos_err_inv < 0)
            tmp = -1;
        else
            tmp = rem((pos_err_1(am_j) + pos_err_inv), pow_dim);
        end;
        tmp = gfplus(0, tmp, tp_num, tp_inv);
        if (tmp < 0)
            den = -1;
        else
            den = rem(den + tmp, pow_dim);
        end;
    end;
end;
% er_loc(am_i) = gfmul(num, pow_dim-den, tp);
tmp = pow_dim - den;
if (tmp < 0) | (num < 0)
    er_loc(am_i) = -1;
else
    er_loc(am_i) = rem(tmp + num, pow_dim);
end;
loc_err(pos_err./m) = er_loc;
end;
ccode = gfplus(loc_err, code, tp_num, tp_inv);
msg = ccode(n_code-k+1 : n_code);
Appendix D

Main Testing Function

% Main function to test the BCH encoder and decoder

clear;
N0 = 14;
N = 21;
t2 = 3;
mag_ary = 4;
[pg K pm cs t2 b] = bchpoly_m1(N,t2,mag_ary)
j = 1;
while rem(mag_ary^j-1, N)~=0
    j=j+1;
end;

if isprime(mag_ary)
    msg = randint(No,K,[0,mag_ary-1]);
    NOI = randbit(No,N,[1]);
else
    beta= (mag_ary^j-1)/(mag_ary-1);
    msg = beta.*randint(No,K,[1,mag_ary-2]);
    NOI = beta.*randbit(No,N,[1]/2);
end;
CODE = bchenco_m(msg, N, K, mag_ary, pg);

if isprime(mag_ary)
    CODE_NOI = rem(CODE+NOI,mag_ary);
    dim = log2(mag_ary^k)/log2(mag_ary);
    tp = gftuple([-1:mag_ary^j-2],dim,mag_ary);
else
    x = factor(mag_ary);
    base = x(1);
    dim = log2(mag_ary^j)/log2(base);
    tuple = gftuple([-1:mag_ary^j-2],dim, base);
    CODE_NOI = gfadd(CODE, NOI, tuple);
endif;
ERA = (mag_ary^j).*randbit(No,N,[1]/2);
ind = find(CODE == -inf);
CODE(ind) = -1;
CODE_ERA = CODE + ERA;
[G, ERR, DEC] = bchdeco(CODE_ERA, K, t2,mag_ary,b);
v = find(msg == -21);
msg(v) = -1;
diff = eq(msg, G);
disp(diff);