# PERFORMANCE OF GENERALIZED BCH CODES OVER GF($Q^S$)

A Thesis

by

## KUNTAL DILIPSINH SAMPAT

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 1996

Major Subject: Electrical Engineering

# PERFORMANCE OF GENERALIZED BCH CODES OVER GF($Q^S$)

A Thesis
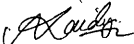
by

## KUNTAL DILIPSINH SAMPAT

Submitted to Texas A&M University
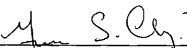in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE
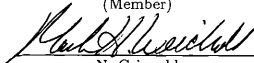
Approved as to style and content by:

Jay N. Livington
(Chair of Committee)

Costas Georghiades
(Member)

Gwan Seung Choi
(Member)

Nitin Vaidya
(Member)

N. Griswold
(Head of Department)

December 1996

Major Subject: Electrical Engineering

## ABSTRACT

Performance of BCH Codes over GF($q^s$). (December 1996)

Kuntal Dilipsinh Sampat, B.E., University of Bombay

Chair of Advisory Committee: Dr. Jay N. Livingston

The main work involved in this thesis is the implementation of a generalized encoder and decoder for BCH codes. The special feature of the implementation is that it works over all finite fields and their extensions as opposed to earlier implementations in literature involving only finite field extensions of the binary field. These generalized codes are attractive mainly because of their rate advantage. Interest in these codewords mapped into modified M-PSK constellations has been generated because of availability of a new method to make a code rotationally invariant. The purpose of this work is to measure the performance of these codes in an Additive White Gaussian Noise ( AWGN ) channel. The effect of Rayleigh fading on the performance has also been simulated for some codes. The new technique of making the M-PSK modulation scheme rotationally invariant is also presented. Chapter I gives an introduction to finite field algebra. Chapter II provides an introduction to the terms that are used in channel coding theory. Readers familiar with the basics in Chapters I and II may directly skip to Chapter III. This chapter deals with the encoding rules and the decoding algorithms for BCH codes from an implementation point of view. Chapter IV is the mainstay of the thesis. It explains rotational invariance, presents the new technique and discusses the system issues involved in making a modulation scheme rotationally invariant. The results of the simulation are shown in Chapter V. Appendix A shows an alternative way to make rotationally invariant block codes in a QAM scheme.

To my parents.

## ACKNOWLEDGMENTS

Nothing calls for being more politically correct than in writing an acknowledgment where only a few have been involved in the work but quite a few involved in deeming its suitability. It is without regard to the first sentence when I acknowledge the following persons for their help in my becoming a Master of Science even though I studied engineering all the time. Dr. J. Livingston for putting me on the path of my thesis from the first semester itself, for being a very patient advisor and tolerating my repeated bloopers. Dr. C. Georghiades without whose help I could not have typed this in my *own* office and on my *own* computer. Dr. V. Vaishampayan for simply being there when I needed to talk to him. The professor who has no office hours: Dr. G. Choi who convinced me that by the time I finished my graduate studies I would be an expert in my field, inspite of the conviction having worn off a bit. Dr. N. Vaidya for being supportive throughout all the stages - from forming the committee to scheduling the date of defense. Lastly, Dr. Sengupta of M.G.M.C.E.T., University of Bombay, without whom, perhaps, I would still have been doing my B.E.

vi

TABLE OF CONTENTS

CHAPTER                                                              Page

LIST OF TABLES

LIST OF FIGURES

## CHAPTER I

## INTRODUCTION TO ALGEBRA

This chapter serves to lay the background for the symbols and mathematics that are used in latter chapters. It provides an introduction to Galois Fields and lists some of their properties.

A. Introduction to Galois Fields

The notation closely follows that of [1].

*Group:*

A binary operation '$*$' is a rule that assigns to each pair of elements $a$ and $b$ a uniquely defined third element $c$.

A set $G$ on which a binary operation $*$ is defined is called a group if the following conditions are satisfied for all elements in $G$:

1. The binary operation is associative:

   $a * (b * c) = (a * b) * c$

2. There exists a unique element $e$ such that

   $a * e = e * a = a$.

   This element $e$ is called the identity element.

3. For any element $a$ there exists another element $a'$ such that $a * a' = a' * a = e$.

   The element $a'$ is called the inverse of $a$ and vice versa. where $a, b, c, a', e \in G$.

The journal model is *IEEE Transactions on Automatic Control.*

A group $G$ is called commutative if $\forall a, b \in G$,

$$a * b = b * a.$$

*Field:*

Let $F$ be a set of elements on which two binary operations called addition '+' and multiplication ' · ' are defined. The set $F$ is a field if the following conditions are satisfied:

1. $F$ is a commutative group under addition '+'. The identity element with respect to addition is called the *zero* element and is denoted by 0.

2. The set of non-zero elements in $F$ form a commutative group under multiplication ' · '. The identity element with respect to multiplication is called the *unit* element and is denoted by 1.

3. Multiplication is distributive over addition; i.e. for any three elements in $F$,
   $a \cdot (b + c) = a \cdot b + a \cdot c$

   In a field the *additive inverse* of an element $a$ is denoted by $-a$. Addition of the additive inverse of an element with itself results in the additive identity 0. The *multiplicative inverse* of a non-zero element $a$ is denoted by $a^{-1}$. Multiplication of $a^{-1}$ with $a$ results in the multiplicative identity 1.

   *Order:* The number of elements in a field is called the order of the field.

   If the number of elements in the field is finite then the field called as a Finite Field or a Galois Field, denoted by $\mathrm{GF}(q)$, where $q$ is the order of the field.

## 1.  Properties of Galois Fields

1. The order of any finite field is the positive power of a prime number. In general, a Galois Field may be expressed as $\mathrm{GF}(q^s)$ where $s$ is a positive integer and

$q^s = (p^m)^s$.

In our notation, in the case of GF($q^s$), $q = p^m$, $p$ is prime and $m$ and $s$ are positive integers, GF($q$) shall be called as the *base field*, GF($p$) shall be called the *prime field*, GF($p^m$) shall be called the *prime extension field* and GF($q^s$) as the [*base*]*extension field*. The base extension field is the highest in the hierarchical structure of fields. The base field elements form the codeword.

2. GF($p^r$) contains a *subfield* (isomorphic to) GF($p^s$), iff $s$ divides $r$.

   For example the subfields of GF($2^{12}$) are shown in Figure 1.
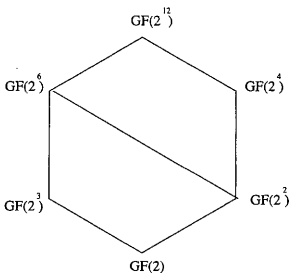


Fig. 1. Subfields of GF($2^{12}$).

3. The *characteristic* of GF($q^s$) is $p$.

4. The *order* of an element $\beta \in$ GF($q^s$) is the least positive integer $n$ such that $\beta^n = 1$.

5. If the order of an element of GF($q^s$) is $q^s - 1$ then that element is called the *primitive* element of that field. All GFs have a primitive element. A GF can

have more than one primitive element.

6. The 1, 2, ..., $q^s - 1$ powers of the primitive element generate all the non-zero elements of a finite field. The representation of the elements of the field as a power of the primitive element is called the *power* or the *exponential* representation of the field. Thus $F = \{0, \alpha^0, \alpha^1, \ldots, \alpha^{q^s-2}\}$. [1]

7. A polynomial is said to be *over* GF(q) if its coefficients are from GF(q).

8. A polynomial over a finite field is said to be *irreducible* if it is not divisible by any other polynomial over the same finite field.

9. An irreducible polynomial of degree $r$ is said to be *primitive* if the smallest positive integer $n$ for which it divides $X^n - 1$ is $n = p^r$. A polynomial that is primitive over GF($q^s$) is irreducible over GF($q$) and is of degree $s$.

Extensive tables of primitive polynomials, representation of field elements, etc. ( both binary as well as non-binary ) can be found in [2]. Methods for the systematic search of primitive polynomials can be found in [3].

By setting the primitive element $\alpha$ as the root of the primitive polynomial of degree $r$, each non-zero element of a field can be represented as a polynomial with a degree of $r - 1$ or less - called the *polynomial representation* - or equivalently as an *r-tuple* of the coefficients of the polynomial representation.

In all extension fields of GF($p$), arithmetic operations are defined modulo-$p$.

In a prime field: addition is modulo-$p$. Subtracting an element $b$ from $a$ is the same as adding the additive inverse of $b$ to $a$. Multiplication is modulo-$p$. Dividing $a$ by $b$ is the same as multiplying $a$ by the multiplicative inverse of $b$.

---

[1] $\alpha^{q^s-1} = \alpha^0 = 1$.

In an extension field: elements of an extension field are expressed as either powers of the primitive element $\alpha$ or as polynomials in $\alpha$ of degree less than the degree of the primitive polynomial that defines the extension field. Multiplication of two elements is easily carried out using the power representation of the elements. In order to multiply two elements, the powers of the two elements are added modulo-$n$, where $n = q^s - 1$. The multiplicative inverse of an element is found by subtracting the power of $\alpha$ which represents the element from $n$. Division is done by adding modulo-$n$ the powers of the divisor with the multiplicative inverse of the dividend. For addition and subtraction the polynomial representation is required. Addition: coefficients of like powers are added modulo-$p$. Subtraction $a - b$: First the additive inverse of $b$ is found; each coefficient is subtracted from $p$. The result is then added to $a$.

**Example 1:** Consider GF($3^2$). The primitive polynomial is thus of degree 2 and from [2] is given by $x^2 + x + 2$. Let $\alpha$ be the primitive element of GF($3^2$). Thus $\alpha^8 = 1$. The non-zero elements of GF($3^2$) may be expressed by the $1^{st}, 2^{nd}, \ldots, 8^{th}$ powers of $\alpha$. To form the polynomial representation of elements, we let $\alpha$ be the root of the primitive polynomial: $\alpha^2 + \alpha + 2 = 0$, which may be expressed as:

$$\alpha^2 = -\alpha - 2 \tag{1.1}$$

Since addition and subtraction are modulo-3, the above equation can be written as

$$\alpha^2 = 2\alpha + 1 \tag{1.2}$$

This gives us the polynomial representation of the element whose power representation is $\alpha^2$. Other elements may be found in the following manner: multiply Equation 1.2 by $\alpha$

$$\alpha^3 = 2\alpha^2 + \alpha \tag{1.3}$$

Substituting the value of $\alpha^2$ from Equation 1.2 in Equation 1.3, we get

$$\begin{aligned} \alpha^3 &= 2(2\alpha + 1) + \alpha \\ &= 2\alpha + 2 \end{aligned} \qquad (1.4)$$

Continuing the same way, the nine elements of $GF(3^2)$ are as given in Table I.

Table I. Three representations of elements of $GF(3^2)$

| Power Representation | Polynomial Representation | 2-Tuple Representation |
|:---:|:---:|:---:|
| 0 | 0 | 0 0 |
| $\alpha^1$ | $\alpha$ | 0 1 |
| $\alpha^2$ | $1 + 2\alpha$ | 1 2 |
| $\alpha^3$ | $2 + 2\alpha$ | 2 2 |
| $\alpha^4$ | 2 | 2 0 |
| $\alpha^5$ | $2\alpha$ | 0 2 |
| $\alpha^6$ | $2 + \alpha$ | 2 1 |
| $\alpha^7$ | $1 + \alpha$ | 1 1 |
| $\alpha^8$ | 1 | 1 0 |

Multiplication: The multiplication of any element with the additive identity '0' results in '0'. To multiply other elements, the power representation of the elements is used. $\alpha^4 \times \alpha^5 = \alpha^{5+4} = \alpha^9 = \alpha^1$ since $\alpha^8 = 1$.

Addition: To add $\alpha^2$ with $\alpha^3$, consider their polynomial representations: $1 + 2\alpha$ and $2 + 2\alpha$ respectively. $1 + 2\alpha + 2 + 2\alpha = 1 + 2 + (2+2)\alpha = 0 + \alpha^1$. Thus $\alpha^2 + \alpha^3 = \alpha^1$ Table II shows the addition table for the elements of $GF(3^2)$.

Table II. Addition table for elements of GF($3^2$)

| | 0 | $\alpha^1$ | $\alpha^2$ | $\alpha^3$ | $\alpha^4$ | $\alpha^5$ | $\alpha^6$ | $\alpha^7$ | $\alpha^8$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | $\alpha^1$ | $\alpha^2$ | $\alpha^3$ | $\alpha^4$ | $\alpha^5$ | $\alpha^6$ | $\alpha^7$ | $\alpha^8$ |
| $\alpha^1$ | $\alpha^1$ | $\alpha^5$ | $\alpha^8$ | $\alpha^4$ | $\alpha^6$ | 0 | $\alpha^3$ | $\alpha^2$ | $\alpha^7$ |
| $\alpha^2$ | $\alpha^2$ | $\alpha^8$ | $\alpha^6$ | $\alpha^1$ | $\alpha^5$ | $\alpha^7$ | 0 | $\alpha^4$ | $\alpha^3$ |
| $\alpha^3$ | $\alpha^3$ | $\alpha^4$ | $\alpha^1$ | $\alpha^7$ | $\alpha^2$ | $\alpha^6$ | $\alpha^8$ | 0 | $\alpha^5$ |
| $\alpha^4$ | $\alpha^4$ | $\alpha^6$ | $\alpha^5$ | $\alpha^2$ | $\alpha^8$ | $\alpha^3$ | $\alpha^7$ | $\alpha^1$ | 0 |
| $\alpha^5$ | $\alpha^5$ | 0 | $\alpha^7$ | $\alpha^6$ | $\alpha^3$ | $\alpha^1$ | $\alpha^4$ | $\alpha^8$ | $\alpha^2$ |
| $\alpha^6$ | $\alpha^6$ | $\alpha^3$ | 0 | $\alpha^8$ | $\alpha^7$ | $\alpha^4$ | $\alpha^2$ | $\alpha^5$ | $\alpha^1$ |
| $\alpha^7$ | $\alpha^7$ | $\alpha^2$ | $\alpha^4$ | 0 | $\alpha^1$ | $\alpha^8$ | $\alpha^5$ | $\alpha^3$ | $\alpha^6$ |
| $\alpha^8$ | $\alpha^8$ | $\alpha^7$ | $\alpha^3$ | $\alpha^5$ | 0 | $\alpha^2$ | $\alpha^1$ | $\alpha^6$ | $\alpha^4$ |

*Cyclotomic Cosets:* The text and notation closely follow the excellent exposition given in [4]. The cyclotomic coset mod-$n$ over GF($q$) is given by

$$C_a = \{a, aq, aq^2, \ldots, aq^{m_a-1}\}$$

where, $a$ is the least integer in the set and is called the coset representative, $m_a$ is the least positive integer such that $aq^{m_a} = a \mod n$.

*Minimal Polynomial:* For an element $\beta \in$ GF($q^s$), if $\phi(X)$ is the polynomial of the smallest degree over GF($q$) such that $\phi(\beta) = 0$, then $\phi(X)$ is called the minimal polynomial of $\beta$.

*Constructing Minimal Polynomials:* The minimal polynomial of $\alpha^a$, where $\alpha \in$ GF($q^m$) is the primitive $n^{th}$ root of unity, over GF($q$) is

$$\phi^{(a)}(x) = \prod_{i \in C_a} (x - \alpha^i)$$

This is a monic polynomial (leading coefficient is unity) with coefficients from $GF(q)$.

For example consider the double error correcting (t=2) code of length n = 63 over $GF(8)$

$$C_0 = \{0\}$$

$$C_1 : 1.8 = 8, 1.8^2 = 64 = 1 \bmod 63 \Rightarrow C_1 = \{1, 8\}$$

$$C_2 : 2.8 = 16, 2.8^2 = 128 = 2 \bmod 63 \Rightarrow C_2 = \{2, 16\}$$

Likewise

$$C_3 = \{3, 24\}, C_4 = \{4, 32\}$$

Since $\alpha^{63} = 1$, it is the primitive element of $GF(64)$. Since $GF(64)$ is an extension field of $GF(8)$ it contains all the elements of $GF(8)$. The order of the primitive element of $GF(8)$ is 7. Therefore $\alpha^9$ is the primitive element of $GF(8)$.

$$\phi^{(1)}(x) = \prod_{i \in C_1} (x - \alpha^i) = (x + \alpha)(x + \alpha^8) = x^2 + \alpha^{45}x + \alpha^9.$$

$$\phi^{(2)}(x) = x^2 + \alpha^{27}x + \alpha^{18}$$

$$\phi^{(3)}(x) = x^2 + \alpha^{45}x + \alpha^{27}$$

$$\phi^{(4)}(x) = x^2 + \alpha^{54}x + \alpha^{36}.$$

CHAPTER II

## CODING THEORY

Noise is defined as any unwanted disturbance which affects the recovery of information that is stored in or transmitted over a medium. In a digital system, noise manifests itself by causing the bits that represent the information to be in error. To achieve reliable transmission/storage of information, error correcting or detecting means are required. A simple and an almost intuitive way to achieve this is by the use of a parity bit. A fixed rule is used to generate the parity bit. The application of a rule to generate additional bits from the information bits is called the encoding process. The information bits along with the generated parity bits form a codeword. The set of all possible codewords is called a code. If even parity is desired, then if an information word has an odd number of ones, a one is chosen as the parity bit. If the information word has an even number of ones then a zero is the parity bit. This parity bit is sent along with the information bits. While recovering the information, if the number of ones in a codeword is odd when an even parity encoding has been done, then an error is detected. This scheme can detect only an odd number of errors. If correction of the errors is also desired then additional parity bits must be generated. The addition of the redundant parity bits leads to a decrease in the rate of transmission of information. Thus, an increase in the reliability causes a decrease in the rate of transmission of information [5].

A. Definitions

*Source*: A source randomly generates elements chosen from a finite set of symbols. This set of symbols is known to the destination or the *receiver*. The order of generation of the symbols is not known to the receiver.

*Message*: The sequence of information symbols generated by the source is called the message. Generally the source is assumed to generate $k$ symbols in unit time. The information may then be regarded as a sequence of $k - tuples$ or as a polynomial of degree $k - 1$, called the message polynomial $(m(x))$, with the information symbols as coefficients.

*Encoder*: The device which assigns redundancy to information by the application of certain rules, is called the encoder. The mapping done by the encoder from the $k$ information symbols to $n, n > k$ symbols must be one-one onto to allow recovery of information. The mapping process may be viewed as a matrix multiplication of the $1 \times k$ message matrix with a $k \times n$ it generator matrix or as a polynomial multiplication of the message polynomialand the *generator polynomial* $(g(x))$. The formed codeword itself may be considered as a polynomial callled $c(x)$.

*Non-systematic encoding*: $c(x) = g(x) \times m(x)$.

*Systematic encoding*: $c(x) = x^{n-k} \times m(x) + b(x)$,

where $b(x)$ is the remainder obtained by dividing $x^{n-k} \times m(x)$ by $g(x)$. Here, as shown in Figure 2, a code word is divided into two parts, the unaltered message part of $k$ digits and the redundant checking part of $n - k$ parity-check digits.

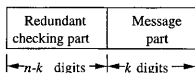| Redundant checking part | Message part |
|---|---|
| $\leftarrow n\text{-}k$  digits $\rightarrow$ | $\leftarrow k$ digits $\rightarrow$ |

Fig. 2. Systematic format of a codeword.

*Code*: The set of $n - tuples$ formed by the mapping by the encoder is called the code. This type of a code which assigns an $n$-tuple to each $k$-tuple is called a *block code*. Each $n$-tuple is called a *codeword*. The usual representation of a code is in the

form $(n, k)$. The number of information symbols $k$ is often called the dimension of the code. A codeword may be represented as:

$$(v_0, v_1, \ldots, v_{n-1})$$

*Linear Code*: If the addition over the finite field over which the code is defined of any two codewords produces another codeword then the code is a linear code.

*Weight*: The number of non-zero symbols in a codeword is called the weight of the codeword.

*Minimum Distance $d_{min}$*: The number of places at which two codewords differ is called the *distance* between the two. The least distance that exists between any two codewords of a code is called the minimum distance of the code. For a linear code the minimum distance is the weight of codeword with the least weight.

*Error Detecting Capacity*: The number of errors that can be detected by the code is $d_{min} - 1$.

*Error Correcting Capacity $t$*: The number of errors that can be corrected by the code. $\frac{(d_{min}-1)}{2} \leq t \leq \frac{(d_{min}-2)}{2}$.

*Rate $R$*: If the source is assumed to generate $k$ information symbols in unit time and after the addition of redundancy the number of symbols generated by the encoder is $n$, then the rate of the code is $R = k/n$.

*Cyclic Code*: If every cyclic shift of a codeword results in another codeword then the code is called a cyclic code, where a single cyclic shift of the $n$-tuple $(v_0, v_1, \ldots, v_{n-1})$ is $(v_1, \ldots, v_{n-1}, v_0)$.

CHAPTER III

BCH CODES

A. Definition

The Bose, Chaudhuri and Hocquenghem (BCH) codes form a large class of powerful random error-correcting cyclic codes. Binary BCH codes were discovered by Hocquenghem in 1959 and independently by Bose and Chaudhuri in 1960. The cyclic nature of these codes was proved by Peterson in 1960.

The decoding algorithms for BCH codes have been developed by Peterson, Gorenstein, Zierler, Chien, Forney, Berlekamp, Massey, Burton and others.

A BCH code over GF($q$) of length n and designed error correcting capability of $t$ errors is the code having

$$\alpha^{b+1}, \alpha^{b+2}, \alpha^{b+3}, \ldots, \alpha^{b+2t}$$

as the zeroes of its generator polynomial, which is of the smallest possible degree, where $\alpha \in$ GF($q^s$) is the primitive $n^{th}$ root of unity i.e. $\alpha^n = 1$, $b$ is a non-negative integer

Special cases:

$b = 0$ Narrow-sense BCH codes.

$n = q^s - 1$ Primitive BCH codes.

$s = 1, n = q - 1$ Reed Solomon ( RS ) codes.

Let $\phi^{(b+i)}$ be the minimal polynomial of $\alpha^{(b+i)}$. Then the generator polynomial is:

$$g(X) = LCM\{\phi^{(b+1)}, \phi^{(b+2)}, \ldots, \phi^{(b+2t)}\}$$

For Reed Solomon codes

$$g(X) = (X - \alpha^b)(X - \alpha^{b+2})(X - \alpha^{b+3}) \ldots (X - \alpha^{b+2t})$$

The degree of each minimal polynomial is $s$ or less. Therefore, the degree of the generator polynomial is at most $2st$, which is also the maximum number of parity check digits required.

For $p = 2$ we obtain the binary BCH codes. For $q = 2$ binary RS codes are obtained. This work concentrates on BCH codes over GF$(q)$, $q \neq 2$. For the purpose of implementation, the most important codes are those where $q$ is a power of 2 since all operations are done modulo-2.

Generalization of the binary BCH codes into $p^m$ symbols, $p$ prime, $m$ positive integer, was obtained by Gorenstein and Zierler in 1961. Such codes were referred to as generalized BCH codes. The advantages of using non-binary base fields is best illustrated by examples taken from the original paper [6] by Gorenstein and Zierler. The text has been slightly modified for the purpose of clarity.

There are two areas ( at least ) of application of codes in $q > 2$ symbols. First, data to be transmitted may appear in such a form and second, although the (binary) BCH codes tend to be highly efficient in correcting independent errors, still greater efficiency may be obtained with the general codes when the errors occur in bursts. A code $C$ of the general class is uniquely determined by the triple $(q, n, e)$ where $q$ is the number of symbols, $n$ is the block length ( number of symbols per block ) and $e$ is the error correction capacity of the code. The parameter $k$, the number of information symbols per block, is computed as a function of $n$ and $e$. Two examples of the second application are:

**Example 1.** Suppose we wish to transmit binary data in the presence of noise that occurs in bursts of length 5 ( binary symbols ) or less and that acceptable reception results when we are able to correct two bursts in a block length of about 60. A BCH code for the job has $n = 63, e = 10, k = 18$, and $q = 2$, and so gives a transmission rate $R = \frac{k}{n} = \frac{18}{63} = \frac{2}{7}$. On the other hand we can take the general code with $q = 2^4$ ( so a burst of length 5 causes errors in exactly two symbols, each symbol being encoded in a natural way as a binary 4-tuple ) $n = 15, e = 4$ and $k = 7$. Then the binary block length is $4 \times 15 = 60$ and the transmission rate is $\frac{7}{15}$.

**Example 2.** Suppose again that binary data are to be transmitted, that errors occur in bursts of length 9 or less and that we must be able to correct 4 such bursts in a binary block length of around 2050. An appropriate BCH code has $n = 2047, e = 36$ and $k$ very nearly 1670, so $R \approx \frac{4}{5}$. A suitable general code has $q = 2^8, n = 255, e = 8, k = 239$ and a binary block length of $8 \times 255 = 2040$; its rate is then $239/255 \approx 15/16$.

## B. Decoding BCH codes

Most of the material in this section has been derived from [7]. We know that the binary generator polynomial $g(x)$ is selected so that it has as zeros $2t$ consecutive powers of $\alpha$.

$$g(\alpha^{b+1}) = g(\alpha^{b+2}) = \ldots = g(\alpha^{b+2t}) = 0 \qquad (3.1)$$

A $q$-ary code vector $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ is a codeword if and only if its associated polynomial $c(x) = c_0 + c_1 x + \ldots + c_{n-1} x^{n-1}$ has as zeros these same $2t$ consecutive powers of $\alpha$. Now consider the received polynomial $r(x)$, which can be expressed as the sum of the transmitted code polynomial and the error polynomial

$e(x) = e_0 + e_1 x + \ldots + e_{n-1} x^{n-1}$. A series of syndromes is obtained by evaluating the received polynomial at the $2t$ zeros.

$$S_j = r(\alpha^{j+b}) = c(\alpha^{j+b}) + e(\alpha^{j+b}) = e(\alpha^{j+b}) = \sum_{k=0}^{n-1} e_k(\alpha^{j+b})^k, \quad j = 1, 2, \ldots, 2t \quad (3.2)$$

If $\nu$ errors have corrupted the transmitted word, at locations $i_1, i_2, \ldots, i_\nu$ then

$$S_j = \sum_{l=1}^{\nu} e_{i_l} X_l^j, \quad j = 1, 2, \ldots, 2t \quad (3.3)$$

where the $X_l$'s are the error locators for their values indicate the positions of the errors in the received word. Expanding the above equation we obtain a sequence of $2t$ algebraic *syndrome equations* in the $\nu$ unknown error locations.

$$\begin{aligned}
S_1 &= X_1 + X_2 + \ldots + X_\nu \\
S_2 &= X_1^2 + X_2^2 + \ldots + X_\nu^2 \\
&\vdots \\
S_{2t} &= X_1^{2t} + X_2^{2t} + \ldots + X_\nu^{2t}
\end{aligned} \quad (3.4)$$

Equations of this form are called *power-sum symmetric functions*. The above equations can be translated into a series of a linear equations. Let $\Lambda(x)$ be the *error locator polynomial* which has as its roots the inverses of the error locators $\{X_l\}$.

$$\Lambda(x) = \prod_{l=1}^{\nu}(1 - X_l x) = \Lambda_\nu x^\nu + \Lambda_{\nu-1} x^{\nu-1} + \ldots + \Lambda_1 x + \Lambda_0 \quad (3.5)$$

It follows that for some error locator $X_l$

$$\Lambda(X_L^{-1}) = \Lambda_\nu X_l^{-\nu} + \Lambda_{\nu-1} X_l^{-\nu+1} + \ldots + \Lambda_0 = 0 \quad (3.6)$$

Since the expression sums to zero, we can multiply by a constant without affecting the equality:

$$e_{i_l} X_l^j (\Lambda_\nu X_l^{-\nu} + \Lambda_{\nu-1} X_l^{-\nu+1} + \ldots + \Lambda_0)$$

$$= e_{i_l}(\Lambda_\nu X_l^{-\nu+j} + \Lambda_{\nu-1+j}X_l^{-\nu+1+j} + \ldots + \Lambda_0 X_l^j) = 0 \qquad (3.7)$$

Summing the above equation over all indices $l$,

$$\sum_{l=1}^\nu e_{i_l}(\Lambda_\nu X_l^{-\nu+j} + \Lambda_{\nu-1+j}X_l^{-\nu+1+j} + \ldots + \Lambda_0 X_l^j)$$

$$= \Lambda_\nu \sum_{l=1}^\nu e_{i_l}X_l^{j-\nu} + \Lambda_{\nu-1}\sum_{l=1}^\nu e_{i_l}X_l^{j-\nu+1} + \ldots + \Lambda_1 \sum_{l=1}^\nu e_{i_l}X_l^{j-1} + \Lambda_0 \sum_{l=1}^\nu e_{i_l}X_l^j$$

$$= \Lambda_\nu S_{j-\nu} + \Lambda_{\nu-1}S_{j-\nu+1} + \ldots + \Lambda_1 S_{j-1} + \Lambda_0 S_j = 0 \qquad (3.8)$$

From Equation 3.8 it is clear that $\Lambda_0$ is always one. Thus Equation 3.8 can be expressed as

$$\Lambda_\nu S_{j-\nu} + \Lambda_{\nu-1}S_{j-\nu+1} + \ldots + \Lambda_1 S_{j-1} = -S_j \qquad (3.9)$$

### 1. The *Peterson-Gorenstein-Zierler* algorithm

If we assume that $\nu = t$, we obtain the following matrix equation.

$$\mathbf{A}'\mathbf{\Lambda} = \begin{bmatrix} S_1 & S_2 & S_3 & \ldots & S_{t-1} & S_t \\ S_2 & S_3 & S_4 & \ldots & S_t & S_{t+1} \\ S_3 & S_4 & S_5 & \ldots & S_{t+1} & S_{t+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ S_{t-1} & S_t & S_{t+1} & \ldots & S_{2t-3} & S_{2t-2} \\ S_t & S_{t+1} & S_{t+2} & \ldots & S_{2t-2} & S_{2t-1} \end{bmatrix} \begin{bmatrix} \Lambda_t \\ \Lambda_{t-1} \\ \Lambda_{t-2} \\ \vdots \\ \Lambda_2 \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{t+1} \\ -S_{t+2} \\ -S_{t+3} \\ \vdots \\ -S_{2t-1} \\ -S_{2t} \end{bmatrix} \qquad (3.10)$$

It can be shown that $\mathbf{A}'$ is non-singular if the received word contains exactly $t$ errors. It can also be shown that $\mathbf{A}'$ is singular if fewer than $t$ errors have occurred. If $\mathbf{A}'$ is singular then the right-most column and the bottom row are removed and the determinant of the resulting matrix computed. This process is repeated till the resulting matrix is non-singular. The coefficients of the error locator polynomial are then found through the use of standard algebraic techniques with computations

performed in the appropriate finite field.

Once the $\nu$ errors are known, Equation 3.8 becomes a system of $2\nu$ equations in $\nu$ unknowns ( the error magnitudes). This system can be reduced to form the matrix relation below. Since the $\{X_l\}$ are non-zero and distinct, the matrix $\mathbf{B}$ [1] is Vandermonde and thus non-singular.

$$\mathbf{Be} = \begin{bmatrix} X_1^{1+b} & X_2^{1+b} & \ldots & X_\nu^{1+b} \\ X_1^{2+b} & X_2^{2+b} & \ldots & X_\nu^{2+b} \\ \vdots & \vdots & \ddots & \vdots \\ X_1^{\nu+b} & X_2^{\nu+b} & \ldots & X_\nu^{\nu+b} \end{bmatrix} \begin{bmatrix} e_{i_1} \\ e_{i_2} \\ \vdots \\ e_{i_\nu} \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_\nu \end{bmatrix} \qquad (3.11)$$

Decoding is completed by solving for the $\{e_{i_i}\}$

The essential steps may be recapitulated as:

1. Compute the syndromes for $r$: $\{S_j\} = \{r(\alpha^{l+b})\}, l = 1, 2, \ldots, 2t$.

2. Construct the syndrome matrix $A'$ in Equation 3.10.

3. Compute the determinant of the syndrome matrix. If the determinant is non-zero go to step 5.

4. Construct a new syndrome matrix by deleting the rightmost column and the bottom row of the old syndrome matrix. Shorten $\Lambda$ by one co-ordinate position by deleting $\Lambda_t$ for the highest remaining $t$. Go to 3.

5. Solve for $\Lambda$ and construct $\Lambda(x)$.

6. Find the roots of $\Lambda(x)$. If the roots are not distinct or not present in the desired field, go to step 10.

---

[1]This expression differs from the one given in [7].

7. Construct the matrix $B$ in Equation 3.11 and solve for the error magnitudes.

8. Subtract the error magnitudes from the values of the received word at the appropriate locations.

9. Output the corrected word and stop.

10. Declare a decoding failure and stop.

   *Limitations of the PGZ algorithm*: It is known [8] that decoding based on the PGZ algorithm may, under certain circumstances, produce an output which is not a codeword and the resulting failure is not announced. An obvious way to counteract this is to recompute the syndromes after the 'corrected' word has been formed. Arne Dur [9] provides a way to detect cases where the PGZ algorithm fails without the recomputation of syndromes by introducing the minimum number of checks during the error evaluation process. We have used a final comparison with the sent code word after decoding to check for legitimacy.

Matrix algebra is used to calculate error locations and magnitudes. Thus the increase in complexity is of the the order of the matrices is directly dependent on the error correcting capacity of the code. Thus as the error correction capacity increases linearly, the complexity of the computations increases by the order of $O(n^2)$. For example for a fifteen error correcting code the number of second order determinants required to be computed is $\frac{15!}{3!} \times C_2^{14} = 1.98 \times 10^{13}$. Thus the algorithm can only be used for codes with low error correcting capacity. A practical limit would be put at five.

## 2. The *Berlekamp-Massey* algorithm

Returning to Equation 3.9 it is seen that the syndrome $S_j$ can be expressed in a recursive form as a function of the coefficients of the error locator polynomial $\Lambda(x)$ and the earlier syndromes $S_{j-1}, \ldots, S_{j-\nu}$. Figure 3 shows that expressions of this form can be given a physical interpretation through the use of a linear feedback shift register (LFSR). The double lined elements denote storage of and operations on non-binary field elements.
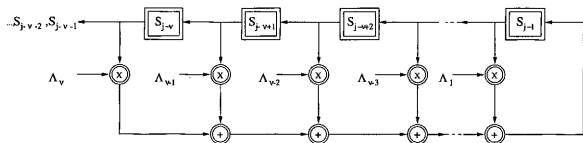


Fig. 3. LFSR interpretation of Equation 3.9.

The problem of decoding BCH and Reed-Solomon codes can thus be expressed as follows: find an LFSR of minimal length such that the first $2t$ elements in the LFSR output sequence are the syndromes $S_1, S_2, \ldots, S_{2t}$. The taps of this shift register provide the desired error locator polynomial $\Lambda(x)$.

Let $\Lambda^{(k)}(x) = \Lambda_k x^k + \Lambda_{k-1} x^{k-1} + \ldots + \Lambda_1 x + 1$ be the *connection polynomial* of length $k$ whose coefficients specify the taps of a length-$k$ LFSR. The Berlekamp-Massey algorithm starts by finding $\Lambda^{(1)}$ such that the first element output by the corresponding LFSR is the first syndrome $S_1$. The second output of this LFSR is then compared to the second syndrome. If the two are not equal then the *discrepancy* between the two is used to modify the connection polynomial. If there is no discrepancy,

then the same connection polynomial is used to generate a third sequence element, which is compared to the third syndrome. The process continues until a connection polynomial is obtained that specifies an LFSR capable of generating all $2t$ elements of the syndrome sequence [7]. The algorithm has five basic parameters: the connection polynomial $\Lambda^{(k)}(x)$, the correction polynomial $T(x)$, the discrepancy $\Delta^{(k)}$, the length $L$ of the shift register and the indexing variable $k$. The algorithm proceeds as follows:

1. Compute the syndrome sequences $S_1, \ldots, S_{2t}$ for the received word.

2. Initialize the algorithm variables as follows: $k = 0, \Lambda^{(0)}(x) = 1, L = 0$ and $T(x) = x$.

3. Set $k = k + 1$. Compute the discrepancy $\Delta^{(k)}$ by adding the $k$th output of the LFSR defined by $\Lambda^{(k-1)}(x)$ to the $k$th syndrome. [2]

$$\Delta^{(k)} = S_k + \sum_{i=1}^{L} \Lambda_i^{(k-1)} S_{k-i}$$

4. If $\Delta^{(k)} = 0$, then go to step 8.

5. Modify the connection polynomial: $\Lambda^{(k)}(x) = \Lambda^{(k-1)}(x) - \Delta^{(k)}(x)T(x)$.

6. If $2L \geq k$ then go to step 8.

7. Set $T(x) = x \cdot T(x)$

8. If $k \leq 2t$ go to step 3.

9. Determine the roots of $\Lambda(x) = \Lambda^{(2t)}(x)$. If the roots are distinct and lie in the right field then determine the error magnitudes, correct the corresponding locations in the received word and STOP.

---

[2]This expression differs from the one given in [7].

10. Declare a decoding failure and STOP.

It is seen that the Berlekamp-Massey algorithm avoids the calculation of determinants in forming the error locator polynomial $\Lambda(x)$ which is the main reason for its popularity.

C. Implementation of Galois Field arithmetic

This section may be skipped without losing understandability of the latter text. The method in which arithmetic computation in any Galois Field is accomplished is discussed. The prime purpose of this section is to explain the working of the program. This might well be used for an implementation in silicon.

For analyzing the performance of a code, the user specifies the following information:

1. **T** the error correcting capacity,

2. **N** the length,

3. **P** the prime number from which extension fields have been derived,

4. **M** the power of **P** which specifies the base field $Q = P^M$,

5. **S** to represent the base extension field $GF(Q^S)$ over which the code is defined,

6. **PRIM_POLY** the primitive polynomial over $GF(P)$ which defines the extension field $GF(Q^S)$,

7. **\*co_ordx and \*co_ordy** the constellation into which the symbols shall be modulated,

8. **R** the choice of a narrow-sense/non-narrow-sense code,

9. **SYS** the choice which decides whether systematic or non-systematic encoding is done,

10. **DIM** the number of modulation intervals over which a symbol from the base field is desired to be sent and

11. **PGZ** the choice between the Berlekamp-Massey and the Peterson-Gorenstein-Zierler decoding algorithms.

Once the above data is supplied, the preamble.c function is called. This function assigns representations ( exponential and polynomial ) to each element of the Galois Field. Elements are arranged in the order of their power representations. The first element is the additive inverse '0', the second is the first power of $\alpha$, the third element is the second power of $\alpha$ and so on. Since the last element $\alpha^{Q^S-1} = 1$, it is represented in the program as $\alpha^0$. The polynomial form of each element is formed using the primitive polynomial and is kept in the two dimensional array named **elem**. In case the base field is a prime field then there elements are not expressed in their polynomial forms. Throughout the program except at certain places, an element is represented in its exponential form. This representation takes the form of an integer which as a power of $\alpha$ forms the exponential representation of that element. Thus the primitive element $\alpha$ is represented as 1. The multiplicative identity 1 is represented by 0. The additive identity zero is represented by $-1$. The power representation is stored in a one-dimensional array named **element**. Thus **element[0]** = -1, **element[1]** = 1, **element[2]**=2, ..., **element**$[Q^S - 1]$=0. The exceptions to exponential representation are those places where addition or subtraction needs to be done. Here the polynomial representation of each element is used. Polynomial addition/subtraction is done in the normal way except that it is done modulo-$p$. Multiplication of two elements is done by adding the integers

that represent them modulo-$n$. All arithmetic operations are done by the code in gf_arith.c. Polynomial multiplication and division are also done in the same file.

Following the formation of elements, an addition table is generated using the polynomial form of elements. In the case of the base field being a prime field, the addition table is simply formed by adding the elements modulo-$p$.

The next step is forming the generator polynomial. To do this minimal polynomials of $2t$ consecutive powers of $\alpha$ are formed. To form minimal polynomials, cyclotomic cosets of each of the $2t$ consecutive powers of $\alpha$ are formed. Repeat occurrences of elements are discarded. This acts as an algorithm to ensure that the LCM of the minimal polynomials forms the generator polynomial. The degree of the generator polynomial determines the dimension of the code or the number of input symbols $K$. This makes the set-up and initialization of the program complete.

The next step in simulation is generating the codeword. $K$ random symbols from the base field are generated. Encoding may be done in two forms: systematic and non-systematic depending on the set value of **SYS**.

Modulation is done by mapping the symbols in the codeword to a point in the two-dimensional constellation formed by **co_ordx[] and co_ordy[]**. Mapping is allowed to be done in a variety of ways. If there are 8 symbols in the base field, the constellation may have a point for each symbol so that one modulation interval is required to transmit one symbol or it may have 2 points so that 3 modulation intervals would be required to transmit one symbol. In this program the base field and the number of points in the constellation can only be powers of the same prime. This is the way it would be done in any practical system or else rate loss takes place. Splitting a symbol also has the consequence of making the code lose its rotational invariance property if it has been so designed.

Independent white Gaussian noise whose variance depends upon the SNR ( Signal

to Noise Ratio )[3] is added to both co-ordinates of the point into which the symbol has been mapped into. This simulates the AWGN channel. Recovery is done by inverse mapping the symbol-point which is nearest in Euclidean space to the received co-ordinates to the symbol.

Once the entire codeword has been recovered one of the decoding techniques is applied. It is observed that step 9 in the Berlekamp-Massey algorithm and step 6 in the PGZ algorithm are the same. After step 9 in the Berlekamp-Massey algorithm steps 7, 8, 9 and 10 of the PGZ algorithm are followed. In the implementation a common piece of code serves the purpose.

If the decoder has been unable to correct for the errors in the received codeword, an error counter $a$ is incremented. A count $b$ is also kept of the number of codewords transmitted. The fraction $a/b$ is the probability of block error. At each value of SNR, the simulation is kept running till 30 errors have occurred. At low values of SNR this results in a low accuracy of the error probability but in the region of interest it provides a value with high repeat accuracy. A plot of the probability of error versus the SNR gives the desired performance of the code.

---

[3]see Chapter IV.

CHAPTER IV

ROTATIONAL INVARIANCE

Quadrature Amplitude Modulation (QAM) and Phase Shift Keying (PSK) are types of modulation schemes that are susceptible to a catastrophic performance degradation if the receiver loses synchronization with the phase of the transmitted signal. These modulation schemes have certain inherent *rotational symmetries* such that rotation of the constellation into which symbols are mapped, by an angle which is a rotational symmetry, results in the same constellation. A receiver that has lost phase synchronization might interpret the rotated constellation as the original constellation, thus making incorrect decisions. Differential encoding is done to make the modulation transparent to signal element rotations.

Ungerboeck's seminal paper [10] on improving the Euclidean distance between transmitted codes by using expanded sets of multilevel/phase signals followed by soft decision decoding to achieve a gain of 3-4dB is called *joint coding and modulation*. A simpler way to achieve this gain was shown by Calderbank and Sloane [11]. In [12] rotational invariance is achieved using joint coding and modulation so that in making a code rotationally invariant also involves an increase in the Euclidean distance between symbols. It is easy to design 180° rotationally invariant codes without sacrificing this gain. Designing rotationally invariant codes immune to other phase ambiguities is more difficult and could involve a loss of the coding gain derived from expanded signal sets.

However joint coding and modulation followed by soft decision decoding to achieve rotational invariance necessarily implies the use of convolutional codes and QAM constellations. One would like to extend the rotational invariance property to PSK constellations as well. The techniques in [12] to make a code rotationally

invariant are quite involved when the constellation has phase symmetries other than 180°. Also if it were to make achieving rotational invariance simpler, one would like to explore the use of block codes for the purpose, even if it meant sacrificing the use of soft decision decoding. In [13], Wei refers to the problem as a difficult one, and resorts to the use of multidimensional constellation to achieve rotational invariance. In other ([14]- [16]) schemes too, multilevel codes/multidimensional constellations have been used. However even in these approaches, providing rotational invariance for anything higher than 4 PSK has proved difficult.

Points worth noting in the rotational invariance achieved in [12] are:

1. the basic property for making the modulation scheme for a code rotationally invariant is that when the modulated symbols are rotated, another valid codeword is formed.

2. for a constellation with $n$ elements, where $n = 2^m$ for some $m$, the convolutional coder produces at its output all possible binary sequences of length $m$. This is a special property of convolutional codes. Block codes of length $m$ typically form a subset of the set containing all possible sequences of lenght $m$.

3. Rotation of the states (current and next) should produce a codeword that corresponds to a point in the constellation which, when rotated by an equivalent amount in the opposite direction corresponds to the codeword formed by the unrotated states.

4. The code is formed simply by performing arithmetic operations on some, but not all of the input lines.

From the above considerations one may easily infer the following:

1. A QAM constellation cannot be used. This holds true if a *regular* QAM constellation is considered. If a constellation formed by equidistant points on concentric circles, the number of points on each circle being equal, is used then under certain circumstances detailed in Appendix A, the code can be made rotationally invariant. However for all practical purposes the error protection on such a code would remain unequal and hence this case does not have much importance.

2. If a QAM constellation as detailed in Appendix A is used then differential encoding has to be applied to all the inputs.

3. A natural alternative to QAM is PSK. Instead of having a point in the constellation represent an entire codeword, it may be made to represent only fixed parts of it. For instance the constellation used for a code over $GF(q)$ may be q PSK. Thus if the length of a codeword is $n$ then $n$ modulation intervals would be required to transmit the entire codeword.

The simplest way to achieve rotational invariance for MPSK is to differentially encode the symbols before modulation. The disadvantage is that a single error propagates and errors continue to occur until the next synchronization symbol is transmitted.

One would thus like to code the transmitted data such that a certain number of errors may be corrected so that the synchronization symbols are not required to be transmitted too frequently. The choice between increased redundancy introduced by coding and the redundancy due to synchronization symbols is an engineering problem with the decision likely to be made depending on the application.

Using stand-alone block codes for rotationally invariant MPSK constellations was first mentioned in [17]. This was followed up with [18]. This method for making

block codes rotationally invariant is presented in the following discussion. The main advantage of this new method is its simplicity.

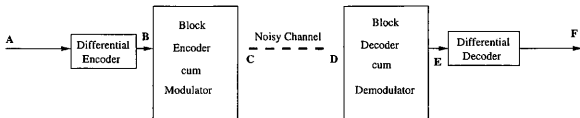Consider the basic scheme shown in Figure 4.



Fig. 4. Basic block diagram to achieve rotational invariance.

The data entering from point A is differentially encoded and forms the data at point B. The data at point B is taken $k$ symbols at a time and block coded into a length of $n$ symbols. The codeword symbols are then modulated into PSK constellations and enter the channel at point C. Data affected by noise and rotation is recovered from point D. After demodulation, this data is decoded, error correction, if any errors have occurred, is done and finally differentially decoding in a sense opposite to the one introduced in the encoding process is done. The data stream obtained at point F is the same as the stream entering at point A if the number of errors between points C and D is less than the error correcting capacity of the block code or if no errors have occurred. To take advantage of the error correcting properties of the code, it is necessary that after rotation between points C and D, the rotated symbols form another valid codeword. To achieve this property, one needs to choose a *proper* mapping scheme and a *matching* code. Suppose $(\alpha, \beta, \ldots, \gamma)$ are the symbols in a codeword and $(\delta, \epsilon, \ldots, \varepsilon)$ is the word formed when each symbol of the codeword is rotated by an angle equal to a rotational symmetry. If rotation of an element by a rotational symmetry $\theta$ can be expressed as an operation $\odot \theta$ then

$\delta = \alpha \odot \theta, \epsilon = \beta \odot \theta, \ldots, \varepsilon = \gamma \odot \theta$. Thus to achieve rotational invariance, $(\delta, \epsilon, \ldots, \varepsilon)$ must be a valid codeword. A different valid codeword must be formed when the original codeword is rotated by each rotational symmetry. This must be true for all codewords. Consider the mapping shown in Figure 5 for elements of GF(5). For
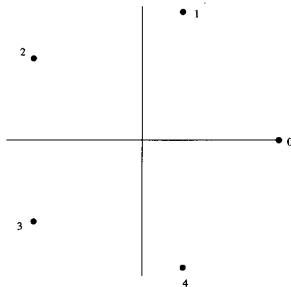


Fig. 5. Sequential mapping of elements of GF(5).

this constellation there are five rotational symmetries viz. $72^\circ, 144^\circ, 216^\circ, 288^0$ and $360^\circ$. A rotation of a symbol in this constellation by $m \times 72^\circ$ may be interpreted as an addition of $m$ mod 5 to the symbol. Consider a simple repeat code with four codewords each of the same but arbitrary length $n$ formed by repeating a symbol$\in$ GF($q$), $n$ times. This is also a cyclic code. This type of a code clearly forms a rotationally invariant scheme if mapped into the arrangement shown in Figure 5. Since Reed-Solomon codes are also cyclic, have repeat codes as thier subsets and are linear, they can be used in such constellations where elements from a Galois Field are labelled consequtively on an M PSK constellation. For short codeword lengths this

scheme is quite satisfactory. For the purpose of getting better rate performance longer codewords are required. BCH codes are the best suited for this because their length is dependent[1] on the base extension field while their modulation symbols are from the base field. However when the base field is not prime, the simple arrangement of modulation symbols shown in Figure 5 does not work because elements can be expressed only as powers of, or in terms of a polynomial in, the primitive element $\alpha$. Addition of power representation of elements is non-linear. For example consider the base extension field GF(64), with the base field of GF(8). Since these are the field extensions of the prime field GF(2), the addition of any element with itself produces the additive identity 0. $\alpha^9 + \alpha^{18} = \alpha^{36}, \alpha^9 + \alpha^{27} = \alpha^{63}$. The alternative is to place the elements in the order of increasing power of $\alpha$ and to interpret rotation of symbols as multiplication. A sequence of symbols from a given Galois Field of the appropriate length forms a valid codeword if, when expressed as a polynomial, it has as its roots all the roots of the generator polynomial. If $g(x)$ is the generator polynomial, then - as has been shown in Chapter II - a codeword may be expressed as $g(x) \cdot m(x)$, where $m(x)$ is the message polynomial. By definition $\alpha \cdot g(x) \cdot m(x)$ is also a valid codeword since its roots remain the same.

If elements of GF(5) are arranged as shown in Figure 6-a, then by multiplying each symbol by $\alpha$, a rotation is *almost* achieved as shown in Figure 6-b. The additive identity '0' is not rotated when it is multiplied by $\alpha$. To ensure that the sequence of symbols after rotation also forms a valid codeword, it is proposed to move the '0' element to the center of the constellation [17]. This new constellation for M PSK has been called (M-1)+1 constellation.
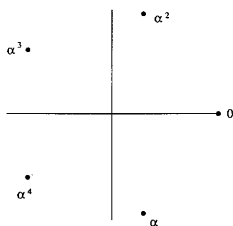
The new 5 PSK constellation in Figure 7-a is called 4+1 PSK constellation.

---

[1]If the code is a primitive BCH code.
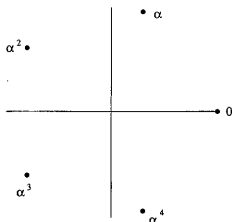
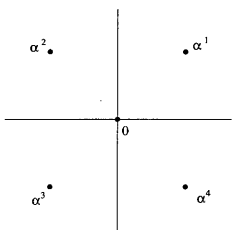$\alpha = 2$ o r $3$, the primitive elements of GF(5)

(a)

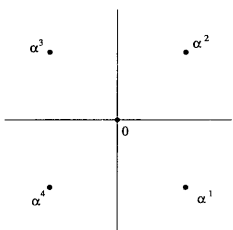Multiplication of each element by $\alpha$

(b)

Fig. 6. 5 PSK constellation.



4+1 PSK modified constellation

(a)

Rotated 4+1 PSK constellation.

(b)

Fig. 7. 4+1 PSK constellation.

The result of multiplication of each symbol of the constellation with $\alpha$ is shown in Figure 7-b. It is seen that complete rotation is achieved. This constellation is more power efficient than the original MPSK constellation. The improvement is .97 dB for the 5 PSK constellation. For the much more common 8 PSK constellation the improvement is .58 dB. Additionally, this constellation has a larger minimum distance between points $(d = 2\sin(\pi/(M - 1)))$ for an additional improvement of $10\log_{10}\frac{\sin^2(\pi/(M-1))}{\sin^2(\pi/M)}$, which for 5 PSK is 1.6dB and for 8 PSK is 1.1dB. Thus for 5 PSK this constellation offers an improvement of 2.57dB compared to the conventional 5 PSK constellation while achieving rotational invariance at the same time. The corresponding figure for 8 PSK is 1.7dB. Thus in our scheme, Figure 4 would be modified into Figure 8.
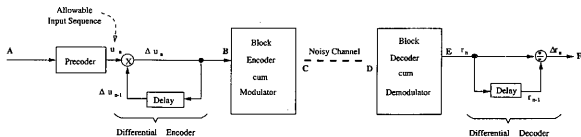


Fig. 8. New rotationally invariant scheme.

## A. System issues

The (M-1)+1 PSK constellation has advantages and disadvantages. Its advantages are better power efficiency and larger minimum Euclidean distance. However, the point at the origin has a drawback for maintaining synchronism. Repeated transmission of the all-zero codeword would probably cause the system to lose symbol timing. Thus,

there needs to be a mechanism for limiting the number of zeros transmitted in a row. A second problem is the requirement of a precoder. The differential encoder creates the input element to the encoder by recursively forming:

$$\Delta u_n = \Delta u_{n-1} \times u_n \tag{4.1}$$

where $\Delta u_n$ is the current and $\Delta u_{n-1}$ is the previous output of the differential encoder, and $u_n$ is the current output from the precoder. Since multiplication is performed in the encoding process, the presence of a '0' in the input stream **u** will make all subsequent outputs of the decoder zero.

Since rotation is seen to be the same as multiplying the sequence by a non-zero element from $GF(q)$, then to undo the rotation, we must resort to division. The differential decoder in Figure 8 then performs division to undo the differential encoding and any rotation:

$$\Delta r_n = r_n \div r_{n-1} \tag{4.2}$$

where $r_n$ and $r_{n-1}$ are the current and previous outputs from the block decoder and $\Delta r_n$ is the current output of the differential decoder. Since division is performed in the decoding process, the sequence **r** cannot have zeros in it.

A precoder is thus required which avoids mapping the input symbols at point **A** in Figure 8 into the symbol '0'. However, the rate loss accompanying this is more than made up for by the power efficiency increase of the new constellation. For example, using a (7,5) Reed-Solomon code, the 5 input symbols allow a total of $8^5 = 2^{15}$ code sequences. Therefore 15 bits could be used as an input to generate 5 symbols from $GF(8)$. However since 0 is not an allowable symbol in our scheme, we have $7^5$ valid code sequences. This corresponds to 14.037 bits. Thus we can create a mapping of

14 input bits to an allowable input sequence with no zeros in it for a rate loss of only 1bit/codeword. This reduces the overall rate to 2/3 from 5/7, or a loss of 0.3dB. The net gain of using the 7+1 PSK constellation is thus 1.7-0.3 = 1.4dB.

A matter of concern that arises is whether at point E in Figure 8, the symbol zero occurs due to rotation of codewords at point D. In the case where non-systematic encoding has been done, rotation may be represented as $\alpha \times c(x) = \alpha \times (g(x) \times m(x)) = g(x) \times (\alpha \times m(x))$. Therefore, the rotated codeword is formed when the original sequence $m(x)$ is multiplied by $\alpha$. Since there are no zeros in $m(x)$ by design, $\alpha \times m(x)$ also cannot have any zeros, which also implies that systematic encoding also would not produce any zeros at point E.

The issue of synchronization loss, as a result of transmission of a zero in the new constellation can be mitigated in part by the use of a systematic encoder. Since the input to the encoder does not contain any zeros, it is guaranteed that at least $k$ of the code symbols of a $(n, k)$ codeword will not be zeros.

Every uncorrectable error that occurs also causes the next decoded word to be in error. In general if the number of consecutive non-correctable errors is $b$ then $b + 1$ words are wrongly decoded.

CHAPTER V

## SIMULATION RESULTS AND CONCLUSION

In this chapter we present results of the various simulations carried out. In all examples the block error probability ($P_b$) has been plotted against the signal to noise ratio ( SNR ) expressed in energy per bit. A block error is said to have occurred when a) the error has not been detected or b) the errors in the received codeword cannot be corrected. To make the decision of whether a block error has occurred, a comparison is made between the 'corrected' codeword and the transmitted codeword. The SNR in energy per bit is calculated as follows: If $E_s$ is the energy per symbol, $n$ is the number of symbols in one codeword or the length of the code, $k$ is the number of information symbols per codeword, and $GF(q)$ is the Galois Field over which the code is defined, then the energy per bit,

$$E_b = \frac{nE_s}{k \log_2 q}$$

and

$$SNR = 10 \log_{10} \frac{E_b}{N_0}$$

where $N_0$ is the single sided noise spectral density.

Random variables with Gaussian distribution having zero mean and unit variance i.e. $N(0,1)$ are generated in the following manner:

First two uniform ( in [0,1) ) random variables $u$ and $v$ are generated. The function drand48 , available in stdlib.h, which returns double precision floating point values, is used for this purpose. The function srand48 - also available in stdlib.h - may be used to initialize the seed of drand48 to a different value each time srand48 is called. Then,

$$y_1 = \sqrt{-2ln(u)}cos(2\pi v)$$

and

$$y_2 = \sqrt{-2ln(u)}sin(2\pi v)$$

are formed which are $N(0,1)$ random variables.

Depending on the SNR, a scalar noise multiplier $m$ is generated, which satisfies the equation

$$SNR = 10\log_{10}\frac{E_b}{2m^2}$$

$m$ is multiplied to $y_1$ and $y_2$, which are then added to the X and Y co-ordinates of the transmitted symbol respectively. Thus $m^2$ acts as the variance of the additive noise.

Fading has been simulated in the following way: A uniform random variable $w$ ( in $[0,1)$ ) is generated. The fading magnitude is given by $\sqrt{-\log(w)}$. This value multiplies the amplitude of the modulated signal. White noise is then added.

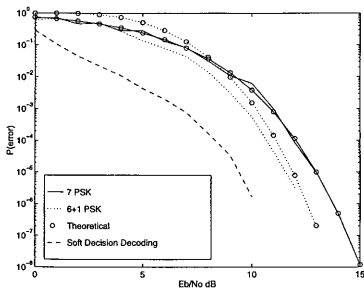Figure 9 is an example of a non-binary RS code.



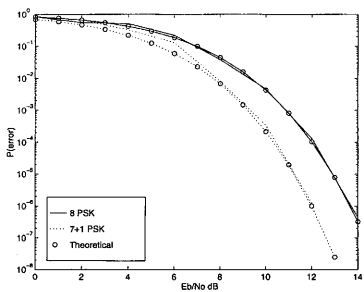Fig. 9. Comparison for soft and hard decision decoding of a (6,2,2) code over GF(7).

Fig. 10. (7,3,2) code over GF(8). Performance compared for 8 PSK modulation and 7+1 PSK modulation.
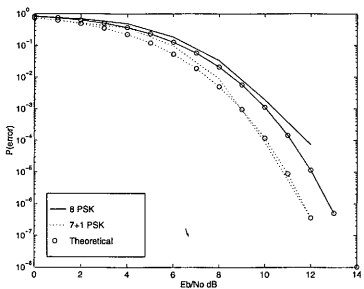


Fig. 11. (7,5,1) code over GF(8). Performance compared for 8 PSK modulation and 7+1 PSK modulation.

Figures 10, 11 and 12 are examples of generalized binary BCH codes.
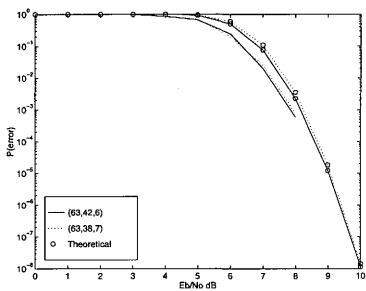


Fig. 12. Performance of rate=2/3 BCH codes over GF(8) using 7+1-PSK.
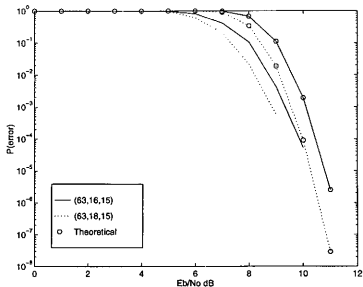
Fig. 13. 15 error correcting BCH codes over GF(8) of length 63 using 7+1 PSK.

Figure 13 shows comparison between a narrow-sense and a non-narrow-sense code. We have chosen R=3 for the code which is non-narrow-sense. When R=0, we find that the dimension(K) of the 15 error correcting code of length 63 is 18. When R=3, K=16.
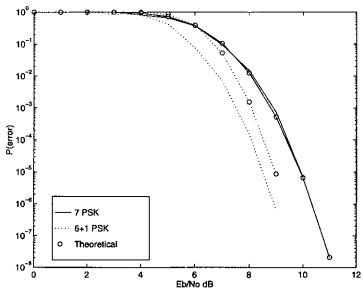
Fig. 14. (48,27,6) code over GF(7). Comparison of performance between 7 PSK and 6+1 PSK modulation.
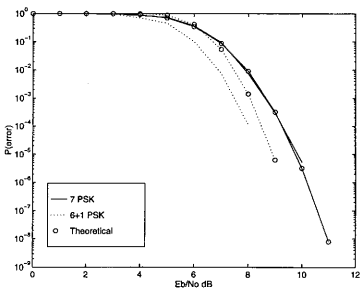


Fig. 15. (48,25,7) code over GF(7). Comparison of performance between 7 PSK and 6+1 PSK modulation.

Figures 14 and 15 are examples of generalized non-binary BCH codes.
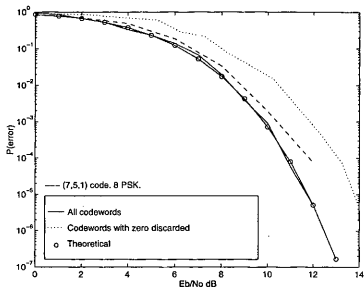


Fig. 16. (8,6,1) code over GF(9). 8+1 PSK modulation.

In the case where loss of synchronization caused by the center zero is undesirable, one might choose to avoid transmitting all codewords that have a zero. Figure 16 shows the performance of such a scheme. For the sake of comparison (7,5,1) code over GF(8) with 8 PSK modulation has been included in the figure as well.

Figure 17 shows the performance of the codes mentioned in Example 1 of Gorenstein's and Zierler's paper [6] under fading in the presence of white noise.
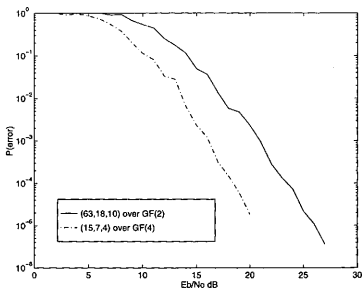
Fig. 17. Advantage of using generalized BCH codes under fading conditions.
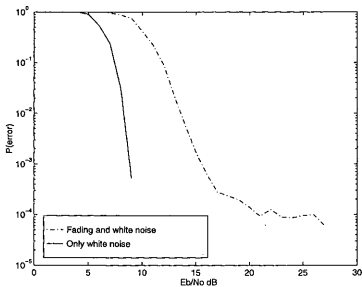


Fig. 18. Comparison between fading and non-fading performance of (63,18,15) code over GF(8) with 7+1 PSK modulation.

Figure 18 plots the performance of one of the codes of Figure 13 under fading conditions.

## A. Conclusion

The rate advantage and superior burst error correcting capability of generalized BCH codes has been known since 1961 [6]. Performance of these codes in AWGN was not available in literature. This work led to the development of a generalized BCH codec whose performance is measured in AWGN. To illustrate the better burst error correction capacity of the generalized codes, simulation in the presence of fading has also been done. This work also presents a new way of making M-PSK schemes rotationally invariant. Using BCH codes and modified PSK constellations, the task of making rotationally invariant has been reduced to a trivial exercise. Performance curves for these modified PSK constellations have been plotted. Since the process of decreasing the size of the channel-symbol alphabet as compared to the codeword alphabet makes the code lose its rotational invariance, the two advantages of these generalized codes viz. higher rate and ease of being rotationally invariant are mutually exclusive.

REFERENCES

[1] Shu Lin and Daniel J. Costello, Jr., *Error Control Coding Fundamentals and Applications.* Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1983.

[2] Rudolf Lidl, *Encyclopedia of Mathematics and its Applications*, vol. 20: Finite Fields. Reading, Massachusetts: Addison-Wesley Pub. Co., Advanced Book Program/World Science Division, 1983.

[3] E. Berlekamp, *Algebraic Coding Theory.* New York, New York: McGraw-Hill Book Co., 1968.

[4] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error Correcting Codes*, Part I. Amsterdam, New York: North-Holland Pub. Co., 1977.

[5] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Technical Journal*, vol. 27, pp. 379–423 and pp. 623–656, 1948.

[6] D. Gorenstein and N. Zierler, "A Class of Cyclic Linear Error-Correcting Codes in $p^m$ Symbols," *J. Soc. Ind. Appl. Math.*, vol. 9, pp. 207–214, June 1961.

[7] Stephen Wicker, *Error Control Systems for Digital Communications and Storage.* Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1995.

[8] D. V. Sarwate and R. D. Morrison, "Decoder Malfunction in BCH Decoders," *IEEE Trans. Inf. Th.*, vol. 36, pp. 884–889, July 1990.

[9] A. Dur, "Avoiding Decoder Malfunction in the Peterson-Gorenstein-Zierler Decoder," *IEEE Trans. Inf. Th.*, vol. 39, no. 2, pp. 640–643, March 1993.

[10] Gottfried Ungerboeck, "Channel Coding with Multilevel/Phase Signals," *IEEE Trans. Inf. Th.*, vol. IT-28, no. 1, pp. 55–67, January 1982.

[11] A. R. Calderbank and N. J. A. Sloane, "New Trellis Codes Based on Lattices and Cosets," *IEEE Trans. Inf. Th.*, vol. IT-33, no. 2, pp. 177–195, March 1987.

[12] Lee-Fang Wei, "Rotationally Invariant Convolutional Coding with Expanded Signal Space - Parts I and II," *IEEE Journal on Selected Areas in Communications*, vol. SAC-2, no. 5, pp. 672–686, September 1984.

[13] ——, "Rotationally Invariant Trellis-Coded Modulations with Multidimensional M-PSK," *IEEE Journal on Selected Areas in Communications*, vol. SAC-7, no. 9, pp. 1281–1295, December 1989.

[14] T. Kasami, T. Takata, T. Fujiwara and S. Lin, "On Linear Structure and Phase Rotation Invariant Properties of Block M-PSK Modulation Codes," *IEEE Trans. Inf. Th.*, vol. 37, no. 1, pp. 164–167, January 1991.

[15] Jay N. Livingston, C.-C. Tung and A. Kulandaivelu, *Rotationally Invariant Multilevel Codes*, Conference Record, 1993 ICC, Geneva, Switzerland, May 1993.

[16] S. S. Pietrobon, G. Ungerboeck, L. C. Perez and D. J. Costello, Jr., "Rotationally Invariant Non-linear Trellis Codes for Two-Dimensional Modulation," *IEEE Trans. Inf. Th.*, vol. 40, pp. 1773–1791, November 1994.

[17] Jay N. Livingston, private communication, October 1995.

[18] Jay N. Livingston, Kuntal Sampat and Chien-Cheng Tung, "Rotationally Invariant Reed-Solomon Coding for PSK Constellations," *Conf. Proc. ISITA '96*, Soc. for Inf. Th. and its Apps., Victoria, British Columbia, September 1996.

# APPENDIX A

## USING A QAM CONSTELLATION WITH ROTATIONALLY INVARIANT BLOCK CODES

Consider the double error correcting $(6,2)$ code over $GF(7)$. The code consists of the following codewords:

| Codeword | | | | | | Input | | Codeword | | | | | | Input | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 5 | 4 | 6 | 2 | 3 | 3 | 4 |
| 2 | 5 | 6 | 4 | 1 | 0 | 0 | 1 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 5 |
| 4 | 3 | 5 | 1 | 2 | 0 | 0 | 2 | 5 | 1 | 2 | 0 | 4 | 3 | 3 | 6 |
| 6 | 1 | 4 | 5 | 3 | 0 | 0 | 3 | 0 | 1 | 6 | 3 | 2 | 4 | 4 | 0 |
| 1 | 6 | 3 | 2 | 4 | 0 | 0 | 4 | 2 | 6 | 5 | 0 | 3 | 4 | 4 | 1 |
| 3 | 4 | 2 | 6 | 5 | 0 | 0 | 5 | 4 | 4 | 4 | 4 | 4 | 4 | 4 | 2 |
| 5 | 2 | 1 | 3 | 6 | 0 | 0 | 6 | 6 | 2 | 3 | 1 | 5 | 4 | 4 | 3 |
| 0 | 2 | 5 | 6 | 4 | 1 | 1 | 0 | 1 | 0 | 2 | 5 | 6 | 4 | 4 | 4 |
| 2 | 0 | 4 | 3 | 5 | 1 | 1 | 1 | 3 | 5 | 1 | 2 | 0 | 4 | 4 | 5 |
| 4 | 5 | 3 | 0 | 6 | 1 | 1 | 2 | 5 | 3 | 0 | 6 | 1 | 4 | 4 | 6 |
| 6 | 3 | 2 | 4 | 0 | 1 | 1 | 3 | 0 | 3 | 4 | 2 | 6 | 5 | 5 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 4 | 2 | 1 | 3 | 6 | 0 | 5 | 5 | 1 |
| 3 | 6 | 0 | 5 | 2 | 1 | 1 | 5 | 6 | 4 | 1 | 0 | 2 | 5 | 5 | 2 |
| 5 | 4 | 6 | 2 | 3 | 1 | 1 | 6 | 1 | 2 | 0 | 4 | 3 | 5 | 5 | 3 |
| 0 | 4 | 3 | 5 | 1 | 2 | 2 | 0 | 3 | 0 | 6 | 1 | 4 | 5 | 5 | 4 |
| 2 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 5 | 5 | 5 | 5 | 5 | 5 | 5 | 5 |
| 4 | 0 | 1 | 6 | 3 | 2 | 2 | 2 | 4 | 6 | 2 | 3 | 1 | 5 | 5 | 6 |
| 6 | 5 | 0 | 3 | 4 | 2 | 2 | 3 | 2 | 3 | 1 | 5 | 4 | 6 | 6 | 0 |
| 1 | 3 | 6 | 0 | 5 | 2 | 2 | 4 | 4 | 1 | 0 | 2 | 5 | 6 | 6 | 1 |
| 3 | 1 | 5 | 4 | 6 | 2 | 2 | 5 | 6 | 6 | 6 | 6 | 6 | 6 | 6 | 2 |
| 5 | 6 | 4 | 1 | 0 | 2 | 2 | 6 | 1 | 4 | 5 | 3 | 0 | 6 | 6 | 3 |
| 0 | 6 | 1 | 4 | 5 | 3 | 3 | 0 | 3 | 2 | 4 | 0 | 1 | 6 | 6 | 4 |
| 2 | 4 | 0 | 1 | 6 | 3 | 3 | 1 | 5 | 0 | 3 | 4 | 2 | 6 | 6 | 5 |
| 4 | 2 | 6 | 5 | 0 | 3 | 3 | 2 | 0 | 5 | 2 | 1 | 3 | 6 | 6 | 6 |
| 6 | 0 | 5 | 2 | 1 | 3 | 3 | 3 | | | | | | | | |

Let us form *partitions* of the code using the following rule:

1. Form closed sets of codewords such that rotating a codeword ( assumed to

be equivalent to adding 1-mod7 to each element of the codeword ) forms another codeword in the same set. Thus we have the following seven sets:

|   | A |   |   |   |   |   |   | B |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |   | 2 | 5 | 6 | 4 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |   | 3 | 6 | 0 | 5 | 2 | 1 |
| 2 | 2 | 2 | 2 | 2 | 2 |   | 4 | 0 | 1 | 6 | 3 | 2 |
| 3 | 3 | 3 | 3 | 3 | 3 |   | 5 | 1 | 2 | 0 | 4 | 3 |
| 4 | 4 | 4 | 4 | 4 | 4 |   | 6 | 2 | 3 | 1 | 5 | 4 |
| 5 | 5 | 5 | 5 | 5 | 5 |   | 0 | 3 | 4 | 2 | 6 | 5 |
| 6 | 6 | 6 | 6 | 6 | 6 |   | 1 | 4 | 5 | 3 | 0 | 6 |

|   | C |   |   |   |   |   |   | D |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 3 | 5 | 1 | 2 | 0 |   | 6 | 1 | 4 | 5 | 3 | 0 |
| 5 | 4 | 6 | 2 | 3 | 1 |   | 0 | 2 | 5 | 6 | 4 | 1 |
| 6 | 5 | 0 | 3 | 4 | 2 |   | 1 | 3 | 6 | 0 | 5 | 2 |
| 0 | 6 | 1 | 4 | 5 | 3 |   | 2 | 4 | 0 | 1 | 6 | 3 |
| 1 | 0 | 2 | 5 | 6 | 4 |   | 3 | 5 | 1 | 2 | 0 | 4 |
| 2 | 1 | 3 | 6 | 0 | 5 |   | 4 | 6 | 2 | 3 | 1 | 5 |
| 3 | 2 | 4 | 0 | 1 | 6 |   | 5 | 0 | 3 | 4 | 2 | 6 |

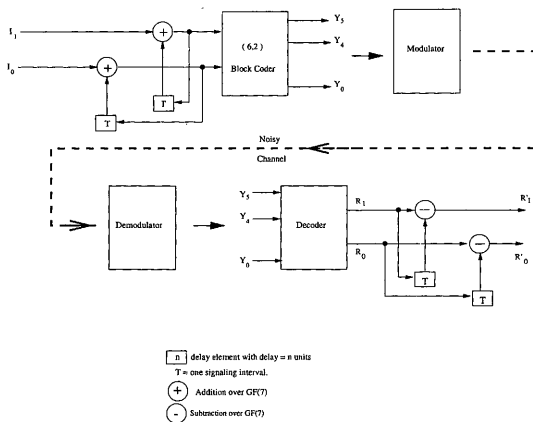|   | E |   |   |   |   |   |   | F |   |   |   |   |   |   | G |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 6 | 3 | 2 | 4 | 0 |   | 3 | 4 | 2 | 6 | 5 | 0 |   | 5 | 2 | 1 | 3 | 6 | 0 |
| 2 | 0 | 4 | 3 | 5 | 1 |   | 4 | 5 | 3 | 0 | 6 | 1 |   | 6 | 3 | 2 | 4 | 0 | 1 |
| 3 | 1 | 5 | 4 | 6 | 2 |   | 5 | 6 | 4 | 1 | 0 | 2 |   | 0 | 4 | 3 | 5 | 1 | 2 |
| 4 | 2 | 6 | 5 | 0 | 3 |   | 6 | 0 | 5 | 2 | 1 | 3 |   | 1 | 5 | 4 | 6 | 2 | 3 |
| 5 | 3 | 0 | 6 | 1 | 4 |   | 0 | 1 | 6 | 3 | 2 | 4 |   | 2 | 6 | 5 | 0 | 3 | 4 |
| 6 | 4 | 1 | 0 | 2 | 5 |   | 1 | 2 | 0 | 4 | 3 | 5 |   | 3 | 0 | 6 | 1 | 4 | 5 |
| 0 | 5 | 2 | 1 | 3 | 6 |   | 2 | 3 | 1 | 5 | 4 | 6 |   | 4 | 1 | 0 | 2 | 5 | 6 |

Fig. 19. The complete en/de-coding scheme for making the (6,2) code over GF(7) rotationally invariant.
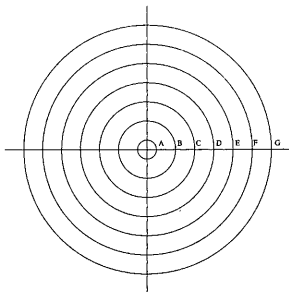
Fig. 20. Mapping the sets.

Both the inputs are differentially encoded by the inputs in the previous mod-ulation interval. Once a codeword of the differentially encoded input is formed it is sent as a single symbol of the QAM constellation shown in Figure 20. Elements of each set are distributed uniformly along the circumference of the circle alongside the label of the set. At the receiving side the differential decoding is done in the opposite sense to that of the transmitter. It can be easily verified that the scheme shown in Figures 19 and 20 is rotationally invariant. The main disadvantage of such a scheme is the unequal error protection.

VITA

Kuntal Dilipsinh Sampat was born in Mumbai, India on 10th April 1973. He completed his B.E. First Class with Honours in Electronics and Telecommunications from the University of Bombay in 1994.

The programs that were used for the simulation can be found at the URL http://ee.tamu.edu/~sampat.

The permanent address of the author is Flat no. 2, Vaishali Co.-op. Hsg. Soc., RDP-1/3, Opp. Ekvira School, Charkop, Kandivli(W), Mumbai, 400 067, India.

The typist for this thesis was Kuntal Dilipsinh Sampat.