PERFORMANCE COMPARISON OF NON-INTERLEAVED BCH CODES AND INTERLEAVED BCH CODES

By

NUR DIANA BINTI MOHD. NURI

FINAL REPORT

Submitted to the Electrical & Electronics Engineering Programme
in Partial Fulfillment of the Requirements
for the Degree
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Universiti Teknologi Petronas
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

PERFORMANCE COMPARISON OF NON-INTERLEAVED BCH CODES AND INTERLEAVED BCH CODES

by

Nur Diana binti Mohd. Nuri

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Approved:

[illegible]

Mr. Azizuddin bin Abdul Aziz
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK

June 2008

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

[illegible]

Nur Diana binti Mohd. Nuri

ABSTRACT

This project covers the research about the BCH error correcting codes and the performance of interleaved and non-interleaved BCH codes. Both long and short BCH codes for multimedia communication are examined in an AWGN channel. Algorithm for simulating the BCH codes was also being investigated, which includes generating the parity check matrix, generating the message code in Galois array matrix, encoding the message blocks, modulation and decoding the message blocks. Algorithm for interleaving that includes interleaving message, including burst errors and deinterleaving message is combined with the BCH codes algorithm for simulating the interleaved BCH codes. The performance and feasibility of the coding structure are tested. The performance comparison between interleaved and non-interleaved BCH codes is studied in terms of error performance, channel performance and effect of data rates on the bit error rate (BER). The Berlekamp-Massey Algorithm decoding scheme was implemented. Random integers are generated and encoded with BCH encoder. Burst errors are added before the message is interleaved, then enter modulation and channel simulation. Interleaved message is then compared with non-interleaved message and the error statistics are compared. Initially, certain amount of burst errors is used. It is found that the graph does not agree with the theoretical bit error rate (BER) versus signal-to-noise ratio (SNR). When compared between each BCH codeword (i.e. n = 31, n = 63 and n = 127), n = 31 shows the highest BER while n = 127 shows the lowest BER. This happened because of the occurrence of error bursts and also due to error frequency. A reduced size or errors from previous is used in the algorithm. A graph similar to the theoretical BER vs SNR is obtained for both interleaved and non-interleaved BCH codes. It is found that BER of non-interleaved is higher than interleaved BCH codes as SNR increases. These observations show that size of errors influence the effect of interleaving. Simulation time is also studied in terms of block length. It is found that interleaved BCH codes consume longer simulation time compared to non-interleaved BCH codes due to additional algorithm for the interleaved BCH codes.

ACKNOWLEDGEMENTS

Completion of this final year project would not have been possible without the assistance and guidance of certain individuals. Their contribution both technically and mentally is highly appreciated. I would like to thank the FYP committees of EE Department for their effort and commitment in ensuring smooth planning and scheduling of FYP lectures, seminars and presentations.

In particular, I would like to express my sincere and utmost appreciation to my supervisor, Mr. Azizuddin Abdul Aziz for his guidance, advice and commitment throughout the process of conducting the final year project. His concerns toward the project have helped me tremendously in achieving the objectives.

I would also like to express my gratitude towards Ms. Siti Hawa Tahir who is an FYP committee for her cooperation in guiding me on the project reports and stuff throughout the period of project's accomplishment.

Thanks are extended to my housemates-cum-coursemates, Nurul 'Atikah Mahzan, Najwa Ayub, Siti Anis Iylia Zainal, Ainol Hayati Mustafa and several others for their support and contribution in this project specifically regarding MATLAB programming.

Last but not least, I would like to thank both my parents En. Mohd. Nuri Bakar and Puan Puteri Nazirah Megat Hussain and several university mates, especially Mohammad Raziswady Salim for their moral supports and for being there whenever I need them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

AWGN	:	Additive White Gaussian Noise
BCH	:	Bose Chaudhuri Hocquenghem
BMA	:	Berlekamp – Massey algorithm
BPSK	:	Binary Phase-Shift Keying
BSC	:	Binary-symmetric Channel
DNS	:	Domain Name System
EA	:	Euclidean algorithm
ECC	:	Error Correction Code
FEC	:	Forward Error Correction
FSK	:	Frequency Shift Keying
IP	:	Internet Protocol
JPEG	:	Joint Photographic Experts Group
MPEG	:	Moving Picture Experts Group
RS	:	Reed-Solomon
TCP	:	Transfer Control Protocol
TFTP	:	Trivial File Transfer Protocol
UDP	:	User Datagram Protocol

CHAPTER 1
INTRODUCTION

1.1 BACKGROUND OF STUDY

The information revolution is vigorously proceeding over the last thirty or so years. Lots of web pages on computers are connected to the Internet [3]. More data are coming in and out of the networks, thus, the reliability of the systems is at risk. It made people wonder of a solution for good data to get through poor networks intact.

The BCH abbreviation stands for the inventors, Hocquenghem in 1959, then later by Bose and Chaudhuri in 1960 independently. The BCH codes form a large class of cyclic codes which is the generalization of the Hamming codes for multiple error correction [2]. BCH codes were generalized to code in p^m symbols by Gorenstein and Zierler in 1961 [4]. The first decoding algorithm for BCH codes were devised by Peterson in 1960 and was then generalized and refined by Gorenstein and Zierler, Chien, Forney, Berlekamp, Massey, Burton and others. Among all the decoding algorithms for BCH codes, Berlekamp's iterative algorithm and Chien's search algorithm are the most efficient ones.

The Noisy Channel Coding Theorem which was discovered by C. E. Shannon in 1948 claims that it is possible to communicate error-free digital data or information up to a given maximum rate through the channel regardless of how contaminated with noise interference a communication channel may be [4]. The theoretical maximum information transfer rate of the channel is with respect to Shannon limit.

Interleaving is a key component of many digital communication systems involving forward error correction (FEC) coding [11]. Burst errors overwrite a lot of bits in a row, but they seldom occur. Thus, interleaving the encoded symbols provides a form of time diversity to protect the transmission against these errors. All data is transmitted with some control bits (independently from the interleaving), such as error correction bits, that enable the channel decoder to correct a certain number of altered bits. The codeword cannot be correctly decoded if a burst error occurs, and more than this number of bits is altered. So the bits of a number of codeword are interleaved and then transmitted. Thus, a burst error affects only a correctable number of bits in each codeword, so the decoder can decode the codeword correctly [12].

Recently, interleavers have become an even more integral part of the code design itself. In the past, the interleaving strategy was weakly linked to selected FEC scheme with the exceptions to concatenated FEC schemes such as concatenated convolutional and RS codes. Parameters are carefully selected to match the error correcting capabilities of the codes involved [11]. As for error control code, block code and convolutional code are most widely used in a variety of applications [1].

For convolutional codes, error correcting capacity increases with the constraint length and the trellis dimension with the coding increase exponentially [1]. The time delay of decoding and deinterleaving is sometimes very large for interleaved convolutional codes. This is not permitted in time-sensitive applications. In block codes, algebraic decoding algorithm and regular structure reduce coding delay and complexity. Furthermore, since the data errors can be controlled to reasonable range, the complexity which also required cost and effort for error correction mechanism can be reduced by utilizing interleaving method.

The use of interleaved convolutional code for image transmission over fading channel has been observed in "Research on error-correcting scheme of image transmission" by D. F. Yuan and J. J. Luo. They found that the image quality with this error control scheme is not satisfactory. Furthermore, questions arise on the complexity and the time delay. In [32], the performance of interleaved BCH codes was estimated using the parameters of Binary-symmetric Channel (BSC). The simulation results show that it is very practical and efficient to estimate the performance of interleaved BCH codes applied to the mobile channel by using BSC when the degree of interleaving is large enough. The use of convolutional code with a novel interleaving scheme to improve image quality has been studied in [33] and it was proven that the scheme proposed is more suitable to image transmission in mobile fading channels compared to interleaved BCH codes.

1.2 PROBLEM STATEMENT

The International Standard Book Number (ISBN) system identifies every book with a ten-digit number, such as 0-226-53420-0. The first nine digits are the actual number but the tenth is added according to a mathematical formula based on the first nine. Any single change in the digit can be verified by a simple check. Some high-end computer memory chips, "ECC RAM," use extended nine-bit bytes. The ninth bit, or "check bit," is always set so that the total number of ones in the extended byte is even. This is called a "checksum" where an error is detected if the sum of the nine bits is not even.

All these processes can *detect* a single error in short notice but they cannot *correct* any error that is detected. Moreover, combinations of two or more errors occurring within the message will not be sensed.

BCH codes are originally designed to fit random-error-correction, and not fit for fading channels. In order to reuse BCH codes, we must first disperse burst errors [1]. Error control coding is combined with interleaving technique which is simple and effective to combat long burst errors. We study two encoding techniques using BCH codes; non-interleaved BCH codes and interleaved BCH codes. The comparison study is important in order to implement proper applications for error correction codes based on the projects' constraints such as time for decoding and codes' complexity.

1.2.1 Importance of error correction codes

The need for consistent and efficient digital data communication systems has been gradually increasing in recent years. Among the various reasons that have brought this need are the enhancement in automatic data processing equipment and the increased need for long range communication. Thus, the BCH codes were developed. The significant applications that require the error correction codes are Internet, deep space communications, and satellite broadcasting.

1.2.2 Applications of error correction codes

Internet

Error detection is performed at multiple levels in a typical TCP/IP stack. Each Ethernet frame carries a CRC-32 checksum. The receiver discards frames with unmatch checksums. Ethernet is a frame-based computer networking technology for local area networks (LANs). A checksum is a form of redundancy check, which is extra data added to a message for the purpose of error detection and error correction. A redundancy check is a very simple measure for protecting the reliability of data by detecting errors in data that is sent through space (telecommunications) or time (storage). [10]

User Datagram Protocol (UDP) has an optional checksum. Packets found to have incorrect checksums are thrown out. [4, 10] Among common network applications are the Domain Name System (DNS), for example, http://elearning.utp.edu.my, streaming media applications, Voice over IP, Trivial File Transfer Protocol (TFTP), and online games.

Transfer Control Protocol (TCP) has a checksum of the payload, TCP header and IP header source and destination addresses. Packets with wrong checksums are discarded and eventually get retransmitted when the sender receives a triple-ack or time-out occurs. [4, 10] By using TCP, networked hosts can swap information or packets, thus, create connections to one another. The protocol ensures that delivery from sender to receiver is reliable and in sequence. TCP also distinguishes data for multiple, concurrent applications such as Web server and email server that were conducted by the same host. TCP supports many internet's application protocols and resulting applications, for instance, World Wide Web, email and Secure Shell.

Deep Space Telecommunication

NASA has used many different error correcting codes. For missions between 1969 and 1977 the Mariner spacecraft used a Reed-Muller code. The noise these spacecraft were subject to was well approximated by a "bell-curve" (normal distribution), so the Reed-Muller codes were well suited to the situation. [4]

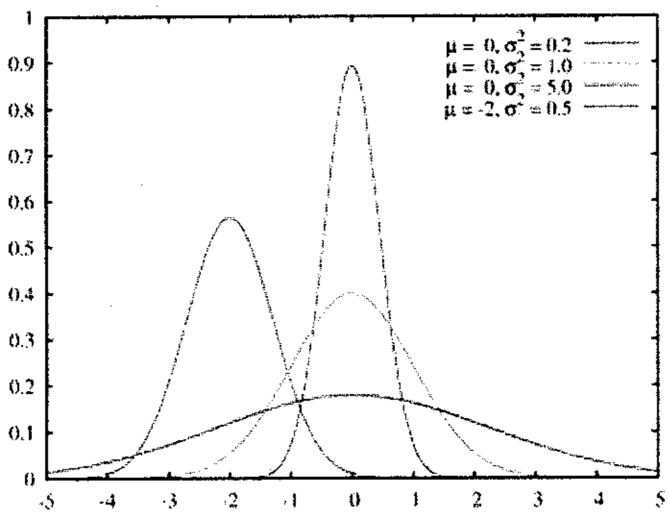


Figure 1: Bell curve [13]

The standard normal distribution is the normal distribution with a mean of zero and a standard deviation of one. It is often called the bell curve because the graph of its probability density resembles a bell.

<u>*Satellite Broadcasting*</u>

The demand for satellite transponder bandwidth continues to grow, fueled by the desire to deliver television, including new channels and High Definition TV and IP data. An automatic device that receives, amplifies, and retransmits a signal on a different frequency. Transponder availability and bandwidth constraints have limited this growth, because transponder capacity is determined by the selected modulation scheme and Forward Error Correction (FEC) rate. FEC is a system of error control for data transmission. [4]

1.2.3 Importance of interleaving

The adverse environment of wireless channel causes long burst errors frequently and where bandwidth is limited, digital data must be greatly compressed before transmission. The multimedia data suffer from burst errors badly and the transmission quality is very poor [1].

FEC coding provides a prevailing technique for transmitting information-bearing data reliably from a source to a sink across the wireless channel. However, to achieve the maximum benefit from FEC coding in many wireless channels, an additional technique known as *interleaving* is required. The need for this new technique is justified based on the fact that wireless channels have memory due to multipath fading which is described as the arrival of signals at the receiver via multiple propagation paths at different lengths [16]. The significant applications that require the interleaving are time-division multiplexing (TDM) in telecommunications, disk storage and data transmission.

1.2.4 Applications of interleaving

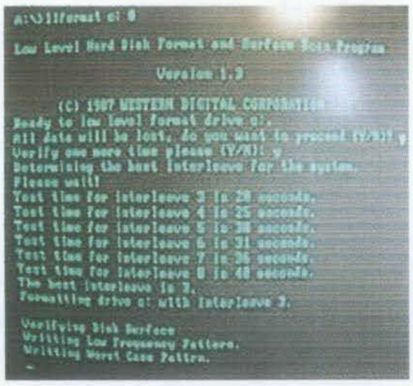
Time-division Multiplexing (TDM) in Telecommunication

Synchronous time division multiplexing is possible when the achievable data rate (or bandwidth) of the medium exceeds the data rate of digital signals to be transmitted. Multiple digital signals (or analog signals carrying digital data) can be carried on a single transmission path by interleaving portions of each signal in time. The interleaving can be at the bit level or in blocks of bytes or larger quantities [10].

Disk Storage

Historically, interleaving was used in ordering block storage on disk-based storage devices such as the floppy disk and the hard disk. The primary purpose of interleaving was to adjust the timing differences between when the computer was ready to transfer data, and when that data was actually arriving at the drive head to be read. Interleaving was very common prior to the 1990s, but faded from use as processing speeds increased. Modern disk storage is not interleaved [31].

Interleaving was used to arrange the sectors in the most efficient manner possible, so that after reading a sector, time would be permitted for processing, and then the next sector in sequence is ready to be read just as the computer is ready to do so. Matching the sector interleave to the processing speed therefore accelerates the data transfer, but an incorrect interleave can make the system perform markedly slower.



```
A:\>llformat c: 0

Low Level Hard Disk Format and Surface Scan Program

                    Version 1.3

        (C) 1987 WESTERN DIGITAL CORPORATION
Ready to low level format drive c:.
All data will be lost, do you want to proceed (Y/N)? y
Verify one more time please (Y/N): y
Determining the best interleave for the system.
Please wait!
Test time for interleave 3 is 20 seconds.
Test time for interleave 4 is 25 seconds.
Test time for interleave 5 is 30 seconds.
Test time for interleave 6 is 31 seconds.
Test time for interleave 7 is 36 seconds.
Test time for interleave 8 is 40 seconds.
The best interleave is 3.
Formatting drive c: with interleave 3.

 Verifying Disk Surface
 Writing Low Frequency Pattern.
 Writing Worst Case Pattrn.
 _
```

Figure 2: Low-level format utility performing interleave speed tests on a 10-megabyte IBM PC XT hard drive [31]

Information is commonly stored on disk storage in very small pieces referred to as *sectors* or *blocks*. These are arranged in concentric rings referred to as *tracks* or *cylinders* across the surface of each disk. While it may seem easiest to order these blocks in direct serial order in each track, such as 1 2 3 4 5 6 7 8 9, for early computing devices this ordering was not practical.

Data to be written or read is put into a special region of reusable memory referred to as a *buffer* [10], [31]. When data needed to be written, it was moved into the buffer, and then written from the buffer to the disk. When data was read, the reverse took place, transferring first into the buffer and then moved to where it was needed. Most early computers were not fast enough to read a sector, move the data from the buffer to somewhere else, and be ready to read the next sector by the time that next sector was appearing under the read head.

When sectors were arranged in direct serial order, after the first sector was read the computer may spend the time it takes for three sectors to pass by before it is ready to receive data again. However with the sectors in direct order, sector two, three, and four have already passed by. The computer doesn't need sectors 4, 5, 6, 7, 8, 9, or 1, and must wait for these to pass by, before reading sector two. This waiting for the disk spin around to the right spot slows the data transfer rate.

To correct for the processing delays, the ideal interleave for this system would be 1:4, ordering the sectors like this: 1 8 6 4 2 9 7 5 3. It reads sector 1, processes for three sectors whereby 8 6 and 4 pass by, and just as the computer becomes ready again, sector two is arriving just as it is needed.

Modern disk storage does not need interleaving since the buffer space is now so much larger. Data is now more commonly stored as clusters which are groups of sectors, and the data buffer is sufficiently large to allow all sectors in a block to be read at once without any delay between sectors.

Data Transmission

Interleaving is used in digital data transmission technology to protect the transmission against burst errors. These errors overwrite a lot of bits in a row, so a typical error correction scheme that expects errors to be more uniformly distributed can be overwhelmed. Interleaving is used to help stop this from happening.

Data is often transmitted with error control bits that enable the receiver to correct a certain number of errors that occur during transmission. If a burst error occurs, too many errors can be made in one code word, and that codeword cannot be correctly decoded. To reduce the effect of such burst errors, the bits of a number of codewords are interleaved before being transmitted. This way, a burst error affects only a correctable number of bits in each codeword, and the decoder can decode the codewords correctly.

This method is popular because it is a less complex and cheaper way to handle burst errors than directly increasing the power of the error correction scheme.

Below is an example as an error correcting code is applied so that the channel codeword has four bits and one-bit errors can be corrected. The channel codewords are put into a block like this: aaaabbbbccccddddeeeeffffgggg.

Consider transmission without interleaving:

```
Error-free message:
aaaabbbbccccddddeeeeffffgggg

Transmission with a burst error:
aaaabbbbccc____deeeeffffgggg
```

The codeword dddd is altered in three bits, so either it cannot be decoded at all (decoding failure) or it might be decoded into the wrong codeword (false decoding). Any of the two happens depends on the error correcting code applied.

Now, let's do the same **with interleaving**:

```
Error-free code words:
aaaabbbbccccddddeeeeffffgggg

Interleaved:
abcdefgabcdefgabcdefgabcdefg

Transmission with a burst error:
abcdefgabcd____bcdefgabcdefg

Received code words after deinterleaving:
aa_abbbbccccdddde_eef_ffg_gg
```

In each of the codewords aaaa, eeee, ffff, gggg, only one bit is altered, so our **one-bit-error-correcting-code** will decode everything correctly.

Of course, latency is increased by interleaving because we cannot send the second bit of codeword aaaa before awaiting the first bit of codeword gggg.

For a different example, consider a meaningful sentence like: ThisIsAnExampleOfInterleaving, and suppose we get a burst error corrupting six letters. First, let us see what the sentence looks like without interleaving.

Consider transmission without interleaving:

```
Original transmitted sentence:
ThisIsAnExampleOfInterleaving

Received sentence with a burst error:
ThisIs______pleOfInterleaving
```

We find that the term "AnExample" is lost or unintelligible.

Now we repeat this example but interleave the sentence prior to transmission. The message is interleaved by transmitting every fourth letter starting at the first letter, then every fourth letter starting at the second, an so on. To make the message a multiple of four letters, three dots have been added to the end. (This is an example of block interleaving.)

Consider transmission with interleaving:

```
Transmitted sentence:
ThisIsAnExampleOfInterleaving...

Error-free transmission:
TIEpfeaghsxlIrv.iAaenli.snmOten.

Received sentence with a burst error:
TIEpfe______Irv.iAaenli.snmOten.

Received sentence after deinterleaving:
T_isI_AnE_amp_eOfInterle_vin_...
```

No single word is completely lost and it is easy to recover them.

1.2.5 Significant of the project

The Bose-Chadhuri-Hocquenghem (BCH) code is an error correcting code which is a method of transmitting message over a noisy transmission channel. In computer science and information theory, the issue of error correction and detection has great practical importance. The error detection is the ability to detect errors that are made due to noise or other impairments during the transmission from the transmitter to the receiver. Error correction has the feature of enabling localization of the errors and correcting them.

Interleaving technique is simple and effective in dispersing error clusters. It works by spreading the bits to be transmitted throughout the entire message and it mainly includes block interleaving and bit interleaving. The latter has slightly better performance than the former, but the former has lower complexity to implement [1]. In this project, the author prefers block interleaving in the system because of the fixed code length of block codes.

This project will introduce the comparison study for non-interleaved and interleaved BCH codes. The encoding and decoding techniques of the BCH codes would be simulated by using MATLAB simulation tool. The comparison study involved error performance, effect of noise variance, channels performance and effect of data rates on the bit error rate (BER). The comparison study is important so that proper applications could be implemented based on the projects' constraints such as time for decoding, cost and codes' complexity.

1.3 OBJECTIVES

- To investigate the effect of block interleaving technique in forward error correction.
- To compare the performance of non-interleaved and interleaved BCH codes in various environment.

1.4 SCOPES OF STUDY

This project covers the research on the utilization of BCH codes in multimedia communication and the performance of BCH codes if combined with interleaving. The channel mode used is AWGN and BPSK modulation. However, we are also going to study other channel models such as binary symmetric channel and Rayleigh channel. Fast fading channel introduces errors which will degrade the quality of transmission. In this project, random integers are used as information to be transferred. The codes could minimize the probability of lost information transmitted. Interleaving and deinterleaving are applied to the encoded data. The Matlab software is used for simulating the encoding/decoding and interleaving/deinterleaving for both the error correcting codes.

CHAPTER 2
LITERATURE REVIEW AND THEORY

2.1 Supporting information

After fifty years since the first coding engines of error-correction and detection were introduced, almost all communication and processing systems went through developments with a variety of error control coding sub-systems [2].

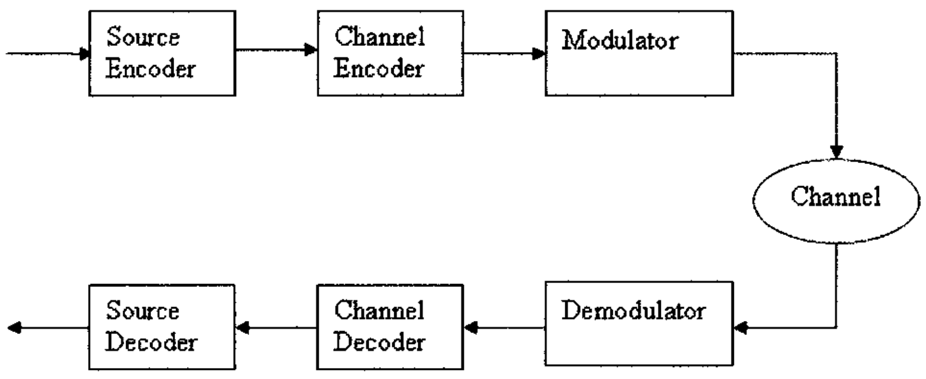


Figure 3: Block Diagram of a general communication system

Coding is the conversion of information to another form. From Figure 3, source coding is conducted for lowering the redundancy in the information, for example; ZIP, JPEG and MPEG2. The purpose of channel coding is to defeat the channel noise. The application of redundant symbols to correct data errors could be implemented by channel encoding. Modulation is the conversion of symbols to a waveform for transmission. The conversion of the waveform back to symbols is done by demodulation. The decoding uses the redundant symbols to correct errors. Several parameters for code performance evaluations are code rate (R), Signal – to – noise ratio (E_b/N_o) and Bit Error Rate (BER). The coding gain is the saving in E_b/N_o required to achieve a given BER when coding is used compared to the other with no coding. Generally, the lower the code rate, the higher the coding gain. [4].

2.2 Title Definition

BCH code is an error correction code while *interleaving* is a technique for handling burst errors in the transmission path whereby data streams containing error correction functions are dispersed. Even if *burst errors* occur, the error correction function can be used effectively for decoding at the receiving equipment end. The operation performed at the receiving end to return the signal to its original state is called *deinterleaving*.

Interleaved BCH codes are BCH codes combined with interleaving technique to disperse errors in data transmission [1] while *non-interleaved BCH* codes are BCH codes alone without combining with interleaving technique.

Bursts (or clusters) of errors are defined as a group of successive error bits in the one-dimensional (1-D) case or linked error bits in multi-dimensional (M-D) cases [17].

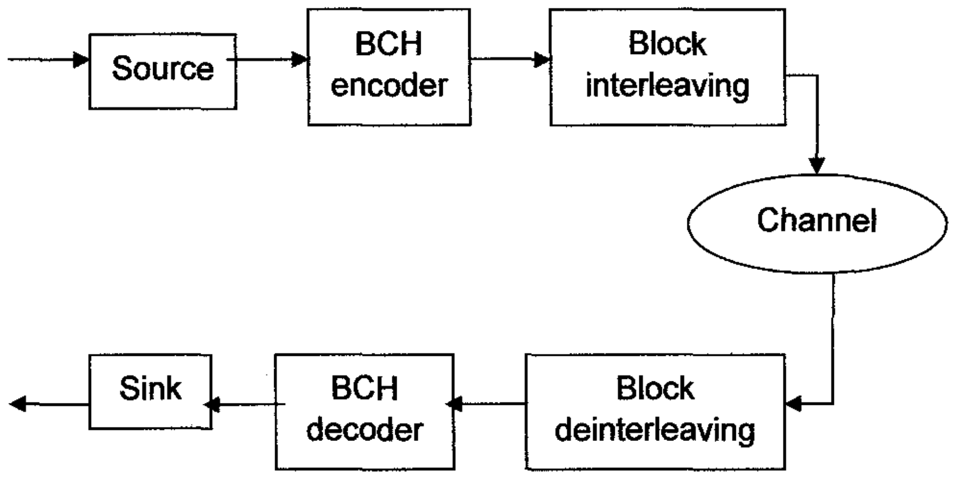


Figure 4: Model of the Project

2.3 BCH Codes

Bose – Chaudhuri – Hocquenghem (BCH) codes are an important subclass of cyclic codes, which have some efficient decoding algorithm due to the strict algebraic architecture [1]. The BCH codes which are a generalization of Hamming distance codes that allow multiple error correction provide a wide variety of block lengths and corresponding code rates. They are important because of their flexibility in the choice of their code parameters and at a block lengths of a few hundred, BCH codes could outperform all other block codes with the same block length and code rate [3].

2.3.1 BCH Codes Parameters

The BCH codes have the following parameters for any positive integers ‘m’ and ‘t’, where $m \geq 3$ and $t < 2^{m-1}$.

Block Length: $n = 2^{m} - 1$

Parity Check Bits: $n - k \leq mt$

Minimum Distance: $d \geq 2t + 1$

This code is capable of correcting combinations of ‘t’ or fewer errors in a block of $n = 2^{m} - 1$ digits. The generator polynomial of this code is specified in terms of its roots from the Galois field, $GF(2^m)$. The generator polynomial g(X) of the t – error – correcting BCH code of length $2^m - 1$ is the lowest – degree polynomial over GF(2) that has: “$a, a^2, a^3, \ldots, a^{2t}$” as its roots. Let $\Phi(X)$ be the minimal polynomial of a^i. Then, g(X) must be a least common multiple (LCM) of $\Phi_1(X), \Phi_2(X), \ldots, \Phi_{2t}(X)$, which is $g(X) = LCM\{\Phi_1(X), \Phi_2(X), \ldots, \Phi_{2t}(X)\}$.

Hence, every even power of ‘a’ in the sequence of “$a, a^2, a^3, \ldots, a^{2t}$” has the same minimal polynomial as the preceding odd power of ‘a’ in the sequence. As a result, the generator polynomial g(X) of the binary t – error – correcting BCH code of length $2^{m} - 1$ can be reduced from $g(X) = LCM\{\Phi_1(X), \Phi_2(X), \ldots, \Phi_{2t}(X)\}$ to $g(X) = LCM\{\Phi_1(X), \Phi_2(X), \ldots, \Phi_{2t-1}(X)\}$.

Due to the degree of each minimal polynomial is ‘m’ or less, the degree of g(X) is at most ‘mt’; that is, the number of parity – check digits, n – k, of the code is at most equal to ‘mt’. ‘n’ represents the block size, ‘n – k’ represents the parity – check digits and ‘t’ represents the number of errors that could be corrected with BCH codes. If the value of ‘t’ is small, n – k is exactly equal to ‘mt’. The BCH codes defined are usually called primitive BCH codes, where its parameters are code length of $2^m - 1$ with $m \geq 10$.

The single – error – correcting BCH code of length $2^m - 1$ is generated by $g(X) = \Phi_1(X)$. Because ‘a’ is primitive element of $GF(2^m)$, $\Phi_1(X)$ is a primitive polynomial of degree 'm'. Therefore, the single – error – correcting BCH code of length $2^m - 1$ is a Hamming code.

Let $v(X) = v_o + v_1a^i + \ldots + v_{n-1}a^{(n-1)i} = 0$ be a code polynomial in a t – error – correcting BCH code of length $n = 2^m - 1$. This equality can be written as a matrix product as follows:

$$(v_o, v_1, \ldots, v_{n-1}) \cdot \begin{bmatrix} 1 \\ ai \\ a2i \\ . \\ . \\ a(n-1)i \end{bmatrix} = 0$$

for $i \leq I \leq 2t$. The condition given as above shows that the inner product of $(v_o, v_1, \ldots, v_{n-1})$ and $(1, a^i, a^{2t}, \ldots, a^{(n-1)i})$ is equal to zero. Therefore, as 'v' is the codeword in the BCH code, then

$$v \,.\, H^T = 0$$

2.3.2 Galois Array

Galois Theory, named after Évariste Galois, is important in BCH codes encoding and decoding algorithms. In abstract algebra, certain Galois Theory problems in field theory can be reduced to group theory, which is simpler and straightforward. Abstract algebra is the field of mathematics that studies algebraic structures, such as groups, rings, fields, modules, vector spaces, and algebras.

Group theory is a branch of mathematics concerned with the study of groups. Galois Theory uses groups to describe the symmetries of the equations satisfied by the solutions to a polynomial equation [7].

A group G is a collection of objects with an operation · satisfying the following rules:

1) For any two elements x and y in the group G we also have x·y in the group G.
2) There is an element, which is usually written 1 or e, but sometimes 0, called the identity in G such that for any x in the group G we have 1·x = x = x·1.

3) For any elements x,y,z in G we have $(x \cdot y) \cdot z = x \cdot (y \cdot z)$. This property is called associativity, which means, we can write x·y·z unambiguously.

4) Every element x in G has a unique inverse y (sometimes weitten –x or x – 1) so that $x \cdot y = y \cdot x = 1$.

Field theory is a branch of mathematics which studies the properties of fields. A field is a mathematical entity for which addition, subtraction, multiplication and division are well-defined [8].

Originally, Galois used permutation groups to describe how the various roots of a given polynomial equation are related to each other. The modern approach to Galois Theory, developed by Richard Dedekind, Leopold Kronecker and Emil Artin involves studying automorphisms of field extensions. The set of all automorphisms of an object forms a group called the automorphism group which is the symmetry group of the object [8].

Galois Theory is concerned with symmetries in the roots of polynomial p(x). Symmetry of the roots is a way of swapping the solutions around in a way which does not matter in some sense. Therefore, $\sqrt{2}$ and - $\sqrt{2}$ are the same because any polynomial expression involving $\sqrt{2}$ will be the same if $\sqrt{2}$ is replaced by - $\sqrt{2}$ [8].

2.3.3 Decoding of BCH Codes

There are several decoding scheme available for BCH codes:

i. **Berlekamp – Massey algorithm (BMA)**

The BMA was invented by Berlekamp and Massey. This is a computationally efficient method to solve the syndrome equation, in terms of the number of operations in $GF(2^m)$. The BMA is important for BCH decoders' implementation in software.

ii. **Euclideon algorithm (EA)**

Euclideon algorithm involves determining the greatest common divisor (GCD) of two integers of elements of any Euclidean domain by repeatedly dividing the two numbers and the remainder in turns. Due to its regular structure, the EA is widely used in hardware implementation for BCH decoders.

iii. Direct method

This method was proposed by Peterson. It directly finds the coefficients of error locator polynomial as a set of linear equations. The term Peterson – Gorenstein – Zierler decodes was used in literature. As the complecity of inverting a matrix grows with the cube of the error – correcting capability, the direct solution method works for small values of 't'.

For this project, the **Berlekamp-Massey** decoding scheme would be implemented for decoding the BCH codes.

2.4 Berlekamp-Massey Algorithm (BMA)

The Berlekamp-Massey algorithm is an algorithm for finding the minimal polynomial of a linearly recurrent sequence. The algorithm was invented by Elwyn Berlekamp in 1968 [5]. Its connection to linear codes was observed by James Massey the following year. It became the key to practical application of the now ubiquitous Reed-Solomon (RS) code. In 1967 E. Berlekamp demonstrated an extremely efficient decoding algorithm for both BCH and RS codes. In 1967, Massey showed that the BCH decoding problem is equivalent to the problem of synthesizing the shortest linear-feedback shift register which is capable of generating a given sequence [2].

2.5 Interleaving Technique

Interleaving is a type of time diversity that lessens the effects of error bursts over the radio fading channel. Several diversity techniques aim at dropping channel effects either by providing the receiver with independent imitations of the transmitted sequence or by randomizing channel errors [18]. In the design of a reliable wireless communication system, we are confronted with two conflicting phenomena: a wireless channel that produces bursts of correlated errors and a convolutional decoder that cannot handle error bursts.

Interleaving is an effective technique to resolve this conflict by converting burst of errors into random-like errors [16], [17]. Interleaving has the net effect of breaking up error bursts that occur during the course of data transmission over the wireless channel and spreading them over the duration of operation of the interleaver. There are three types of interleaving which are block interleaving, convolutional interleaving and random interleaving [16].

2.5.1 Block interleaving

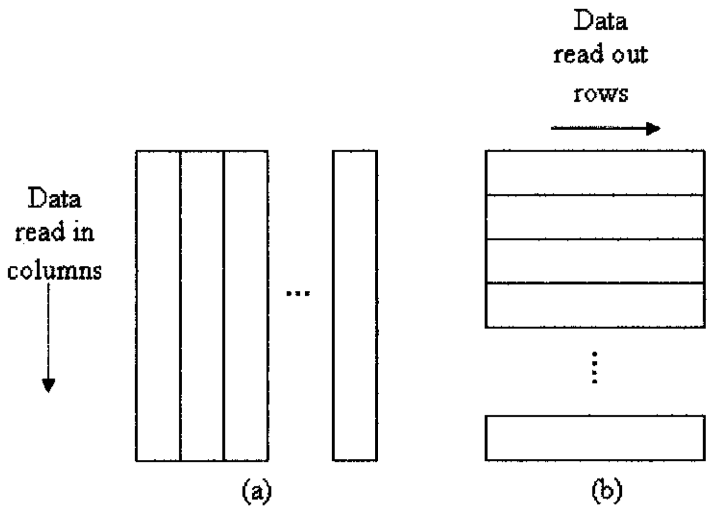


Figure 5: Block interleaver sturucture [16]
(a) Data "read in"
(b) Data "read out"

Classical block interleaver functions as memory buffer, as shown in **Figure 5**, where data are written into this *N x L* rectangular array columnwise from the channel encoder and its substance are sent to the transmitter. At the receiver, the inverse operation is performed, which are, data are written into the contents of the array in the receiver in row manner. Once the array is filled, it is read out in column manner into the Viterbi decoder.

The (N,L) interleaver and deinterleaver for block interleaver are both periodic with fundamental period $T = NL$. For the correlation time or error-burst-length time that corresponds to L received bits, the effect of an error burst would corrupt the equivalent of one row of the deinterleaver block at the receiver.

2.5.2 Convolutional interleaving

Defining the period

$$T = LN,$$

we refer to the interleaver as an $(L \ x \ N)$ convolutional interleaver, which has proper ties similar to those of the $(L \ x \ N)$ block interleaver.

The sequence of encoded bits to be interleaved in the transmitter is arranged in blocks of L bits. For each block, the encoded bits are sequentially shifted into and out of a bank of N registers by means of two synchronized input and output *commutators*. The interleaver is structured as follows:

1. The zeroth shift register provides no storage; that is, the incoming encoded symbol is transmitted immediately.
2. Each successive shift register provides a storage capacity of L symbols more than the preceding shift register.
3. Each shift register is visited regularly on a periodic basis.

With each new encoded symbol, the commutators switch to a new shift register. The new symbol is shifted into the register, and the oldest symbol stored in that register is shifted out. After finishing with the $(N - 1)$th shift register (i.e., the last register), the commutators return to the zeroth shift register. Thus the switching – shifting procedure is repeated periodically on a regular basis.

The deinterleaver in the receiver also uses N shift registers and a pair of input/output commutators that are synchronized with those in the interleaver. The shift registers are stacked in reverse orde of those in the interleaver, resulting in the deinterleaver performs the inverse operation in the receiver.

An advantage of convolutional over block interleaving is that in convolutional interleaving, the total end-to-end delay is $L(N - 1)$ symbols and the memory requirement is $L(N - 1)/2$ in both the interleaver and deinterleaver, which are one-half of the correspondinf values in a block interleaver/deinterleaver for a similar level of interleaving.

2.5.3 Random interleaving

In a random interleaver, a block of N input bits is written into the interleaver in the order in which the bits are received, but they are reas out in a random manner. Typically, the permutation of the input bits is defined by a uniform distributuin. Let $\Pi(i)$ denote the permuted location of the ith input bit where $i = 1,2,\ldots,N$. The set of integers denoted by $\{\Pi(i)\}_{i=1}^{N}$, defining the order in which the stored input bits are read out of the interleaver, is generated according to the following two-step algorithm:

1. Choose an integer i_1 from the uniformly distributed set $A = \{1,2,\ldots,N\}$, with the probability of choosing i_1 being $P(i_1) = 1/N$. the chosen integer i_1 is set to $\Pi(1)$.

2. for $k > 1$, choose an integer i_k from the uniformly distributed set $A_k = \{ i \in A, i \neq i_1, i_2,\ldots, i_{k-1}\}$, with the probability of choosing i_k being $P(i_k) = 1/(N - \mathrm{k} + 1)$. The chosen integer i_k is set to $\Pi(k)$. Note that the size of the set A_k is progressively reduced for $k > 1$. When $k = N$, we are left with a single integer, i_N, that is set to $\Pi(N)$.

2.6 Finite-State Channel (FSC) Model

Four-state simply partitioned Markov model is used to represent a typical fast fading channel. This kind of finite-state channel (FSC) model can run easily and resemble the real communication environment in effect. The selected channel has parameters below: FSK (modulation), 100 km/h (vehicle velocity) and 300 bits/s (data rate). The Markov transition matrix related to the model is as follows [1]:

$$P = \begin{bmatrix} 0.974932 & 0 & 0 & 0.025068 \\ 0 & 0.515248 & 0 & 0.484752 \\ 0 & 0 & 0.997782 & 0.002218 \\ 0.039832 & 0.450840 & 0.052737 & 0.456590 \end{bmatrix}$$

CHAPTER 3
METHODOLOGY

3.1 Procedure Identification

The main objective of this project is to investigate the utilization of BCH codes in multimedia communication with performance comparison of non-interleaved and interleaved BCH codes. Input message used is random integers. For performance comparison, the encoded message is also decoded without interleaving and the rate of error will be compared. AWGN channel is used.

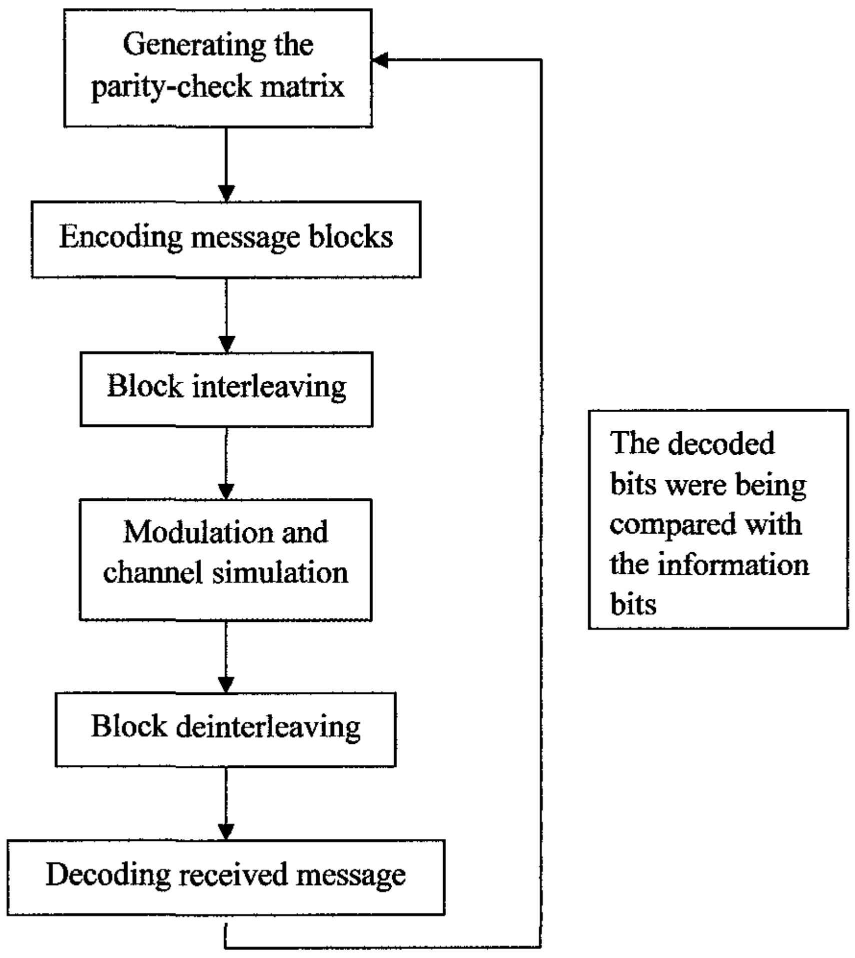

Generating the parity-check matrix
Encoding message blocks
Block interleaving
Modulation and channel simulation
Block deinterleaving
Decoding received message
The decoded bits were being compared with the information bits

Figure 6: Step-by-step System Methodology

3.1.1 Interleaved BCH Codes Algorithm

The BCH codes algorithm was divided into several parts for more detailed explanations. Referring to the Matlab communication toolbox functions, the algorithm for BCH code is listed as follows:

Step 1: Construct the codeword

```
m=4;
n=2^m-1;
k=5;
nwords=10;
```

From the codes above, n represents the codeword length, k is the message length, and the nwords represents the number of words to encode for this program.

Step 2: Create states for random number generator

```
st1 = 27221; st2 = 4831;
```

st1 and st2 are states for random number generator.

Step 3: Create Galois field array

```
msg=gf(randint(nwords,k,st1));
```

From the code above, randint(10,5) generates a 10 by 5 matrix of random binary numbers. “0” and “1” occur with equal probability.

‘GF’ function creates a Galois field array. The msg=randint(nwords,k,st1)) creates a Galois field array from the matrix randint(nwords,k). The Galois field has 2^m elements, where for this program, the value of m is set to default value 1. Each element of x must be 0 or 1. The output for msg is a variable that MATLAB recognizes as a Galois field array, rather than an array of integers. [19]

```
x = double(msg.x);
```

The above code converts msg from Galois array to integer for error statistics because biterr (code in interleaving stage) codes integers.

Step 4: Create generator polynomial

```
[genpoly,t] = bchgenpoly(n,k)
```

The function bchgenpoly gets generator polynomial and error-correction capability. genpoly = bchgenpoly (n,k) returns the narrow-sense generator polynomial of a BCH code with code length n and message length k. The codeword length n must have the form 2^m-1 for some integer m between 3 and 16. The output genpoly is a Galois row vector that represents the coefficients of the generator polynomial in order of descending powers. The narrow-sense generator polynomial is (X-alpha)*(X-alpha^2)*...*(X-alpha^(N-K)), where alpha is the root of the default primitive polynomial for the field GF (N+1). [20]

Step 5: Encode the message

```
code = bchenc(msg,n,k);
```

code = bchenc(msg,n,k) encodes the message in msg using an [n,k] BCH encoder with the narrow-sense generator polynomial. msg is a Galois array of symbols over GF(2). Each k-element row of msg represents a message word, where the leftmost symbol is the most significant symbol. Parity symbols are at the end of each word in the output Galois array code. [21]

```
y = double(code.x);
```

The above code converts code from Galois array to integer (in complex double form) for interleaving.

Step 6: Create burst errors

```
errors = zeros(size(code)); errors(n-2:n+3) = [1 1 1 1 1 1];
```

The above codes create burst error that will corrupt two adjacent codewords.

Step 7: Interleave encoded message

```
inter = randintrlv(y,st2);.
```

intrlvd = randintrlv(data,state) rearranges the elements in data using a random permutation. The state parameter initializes the random number generator that the function uses to determine the permutation. The function is predictable and invertible for a given state, but different states produce different permutations. If data is a matrix with multiple rows and columns, then the function processes the columns independently. [25]

```
inter_err = bitxor(inter,errors);
```

The code inter_err = bitxor(inter,errors) includes burst errors created earlier into the interleaved encoded message.

C = bitxor(A, B) returns the bitwise XOR of the two arguments A and B. Both A and B must be unsigned integers. [26]

Step 8: BPSK modulation

```
mod = pskmod(code_err,2);
```

The pskmod function represents phase shift keying modulation.

y = pskmod(x,M) outputs the complex envelope y of the modulation of the message signal x using phase shift keying modulation. M is the alphabet size and must be an integer power of 2. The message signal must consist of integers between 0 and M-1. The initial phase of the modulation is zero. For two-dimensional signals, the function treats each column as 1 channel. [22]

Step 9: AWGN channel simulation

```
channel = awgn(mod,snr);
```

AWGN adds white Gaussian noise to a signal.

y = awgn(x,snr) add white Gaussian noise to x. the snr is in dB.

The power of x is assumed to be 0 dBW. If x is complex, then awgn adds complex noise [30].

Step 10: BPSK demodulation

```
r = pskdemod(ncode_dbl,2);
```

Demodulation is basically the reverse of modulation.

z = pskdemod(y,M) demodulates the complex envelope y of a PSK modulated signal. M is the alphabet size and must be an integer power of 2. The demodulator, which is designed specifically for the symbol-set used by the modulator, determines the phase of the received signal and maps it back to the symbol it represents, thus recovering the original data. If y is a matrix with multiple rows and columns, then the function processes the columns independently. In this case, y3 is processed independently. [24]

Step 11: Deinterleave the interleaved message

```
D = randdeintrlv(C,st2); % Deinterleave.

deinter_gf = gf(D);
```

deintrlvd = randdeintrlv(data,state) restores the original ordering of the elements in data by inverting a random permutation. To use this function as an inverse of the randintrlv function, the same state input is used in both functions. In that case, the two functions are inverses in the sense that applying randintrlv followed by randdeintrlv leaves data unchanged. [27]

Step 12: Decode the received message

```
[newmsg1,err1,ccode1] = bchdec(deinter_gf,n,k)
```

The function bchdec represents the BCH decoder.

decoded = bchdec(code,n,k) attempts to decode the received signal in code using an [n,k] BCH decoder with the narrow-sense generator polynomial. code is a Galois array of symbols over GF(2). Each n-element row of code represents a corrupted systematic codeword, where the parity symbols are at the end and the leftmost symbol is the most significant symbol.

In the Galois array decoded, each row represents the attempt at decoding the corresponding row in code. A decoding failure occurs if bchdec detects more than t errors in a row of code, where t is the number of correctable errors as reported by bchgenpoly. In the case of a decoding failure, bchdec forms the corresponding row of decoded by merely removing n-k symbols from the end of the row of code.

[decoded,cnumerr,ccode] = bchdec(deinter_gf,n,k) returns ccode, the corrected version of code. The Galois array ccode has the same format as code. If a decoding failure occurs in a certain row of code, then the corresponding row in ccode contains that row unchanged. [28]

Step 13: Error Statistics

```
z1 = double(newmsg1.x);

disp('Number of errors and error rate, with interleaving:');

[number_with,rate_with] = biterr(x,z1)
```

newmsg1 is the decoded message that is converted to integer (in complex double form) from Galois array by double for use in error statistics.

The biterr function compares unsigned binary representations of elements in x with those in z1. [29]

BCH Matlab source could be referred in Appendix A.

3.2 Tools and Software Identification

This project requires Matlab simulation tool for producing results of encoding and decoding for the error correction codes as well as interleaved and non-interleaved BCH Codes.

CHAPTER 4
RESULTS AND DISCUSSION

4.1 PERFORMANCE OF BCH CODES WITH VARYING SNR

The author utilized random integer as message and applied all the BCH algorithms. Channel used is AWGN and the performances of bit error rate versus signal-to-noise ratio (SNR) of the BCH codes were observed.

Signal-to-noise ratio is an engineering term for the ratio of power in a signal (significant information) to the power contained in a noise that is present during transmission [4], [10].

$$SNR = \frac{P_{signal}}{P_{noise}}$$

SNR are usually expressed in terms of logarithmic decibel scale because many signals have a very wide dynamic range. In decibels, the *SNR* is 20 times the base 10 logarithm of the amplitude ratio or 10 times the logarithm of the power ratio:

$$SNR = 10 \log_{10} \left(\frac{P_{signal}}{P_{noise}}\right)$$

For this project, the *SNR* was related to the noise variance (N_o), which is:

$$SNR = 10 \log_{10} \left(\frac{1}{N_o}\right)$$

An error ratio is the ratio of the number of bits, or blocks incorrectly received to the total number of bits, or blocks sent during a specified time interval [4], [10]. The error ratio is usually expressed in scientific notation. For example, 2.5 erroneous bits out of 100,000 bits transmitted would be 2.5 out of 10^5 or 2.5×10^{-5}.

Besides that, the bit error ratio for the transmission is the number of erroneous bits received divided by the total number of bits transmitted. For the information BER, the number of erroneous decoded bits is divided by the total number of decoded bits.

Below are the results that show the error performances of both non-interleaved and interleaved BCH codes and their performance comparison for a certain amount of burst errors which are different from each other.

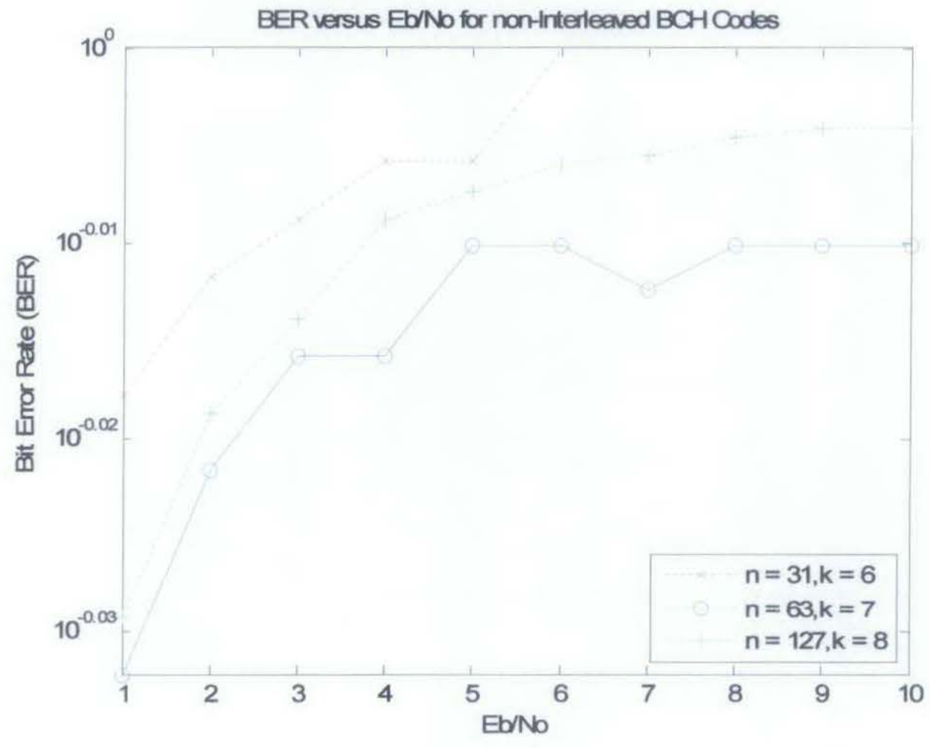


Figure 7: BER vs Eb/No for non-interleaved BCH codes

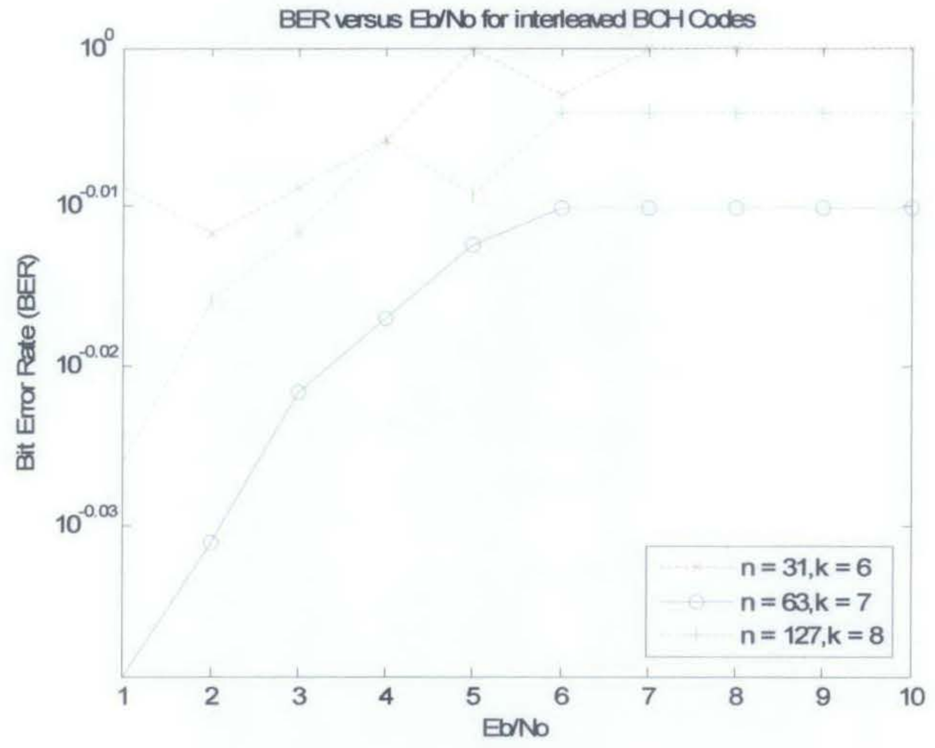


Figure 8: BER vs Eb/No for interleaved BCH codes

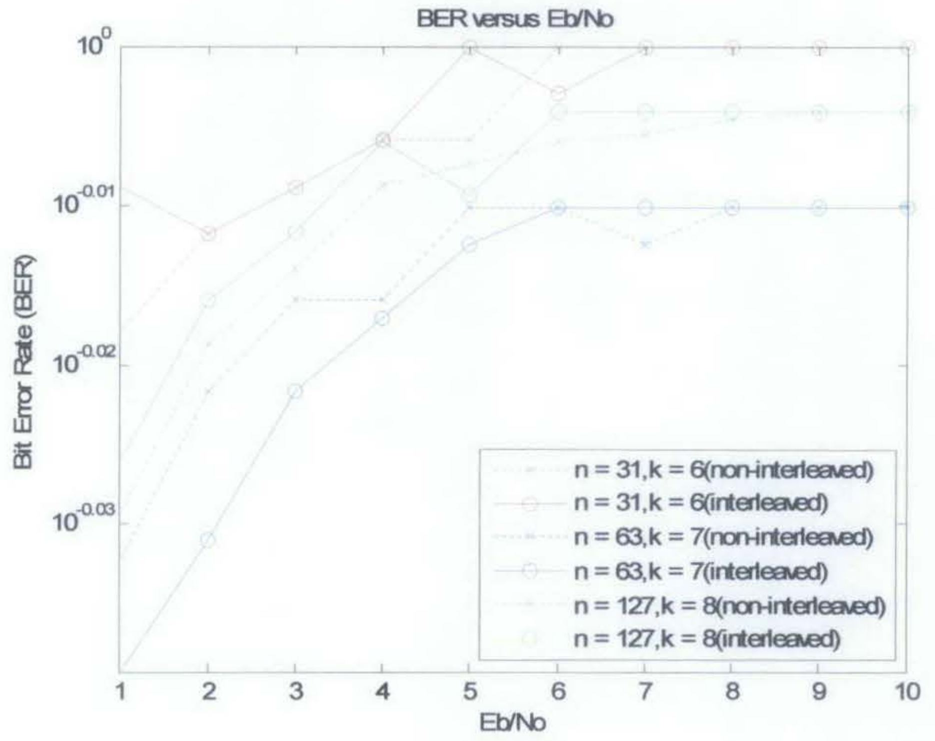


Figure 9: Comparison of BER vs SNR for interleaved and non-interleaved BCH Codes

4.1.1 Discussions

From **Figure 7,** bit error rate (BER) of each BCH codes increases as signal-to-noise ratio (SNR) increases. It is different compared to the theoretical BER vs SNR shown in **Figure 14 (Appendix C)**. This is because of the bursts of errors added in BCH codes simulation for non-interleaving.

Comparing between BCH codes performances, codeword, $n = 31$ shows the highest bit error rate while $n = 63$ shows the lowest bit error rate. **Figure 9** presents a clearer comparison of non-interleaved and interleaved BCH codes in terms of BER over SNR. This occurs due to the instability of burst errors included where each BCH code is provided with different amount of burst errors. Errors for n = 63 and n = 127 are not that enough to be detected.

Observing from these graphs, it could be seen that there are several limitations that influence the success of interleaving technique. Firstly, it is based on the size of burst of errors. For example, to combat bursts of errors of size t equal to a specific given burst error size t_o, one needs to implement an algorithm with a set of parameters to construct an interleaving code. When size t increases, that is, $t > t_o$, one needs to implement an algorithm with a new set of parameters to construct another interleaving code. This means, the interleaved array constructed for a specific t_o may not be able to correct a burst of errors of size t as $t > t_o$.

Secondly, when the actual size of a burst, t, is less than t_o, with which the interleaving algorithm is applied, the technique is no longer optimal which means that the interleaving degree reaches its lower bound.

4.2 PERFORMANCE OF BCH CODES WITH SMALL BURST OF ERRORS

To prove that size of errors influence interleaving, another simulation is performed with reduced size of errors. Figures below show the error performances of both non-interleaved and interleaved BCH codes and their performance comparison as amount of burst errors included is reduced. For a given communication system, the bit error ratio which is the ratio of the number of bits incorrectly received to the total number of bits sent during a specified time interval, will be affected by both the data transmission rate and the signal power margin.

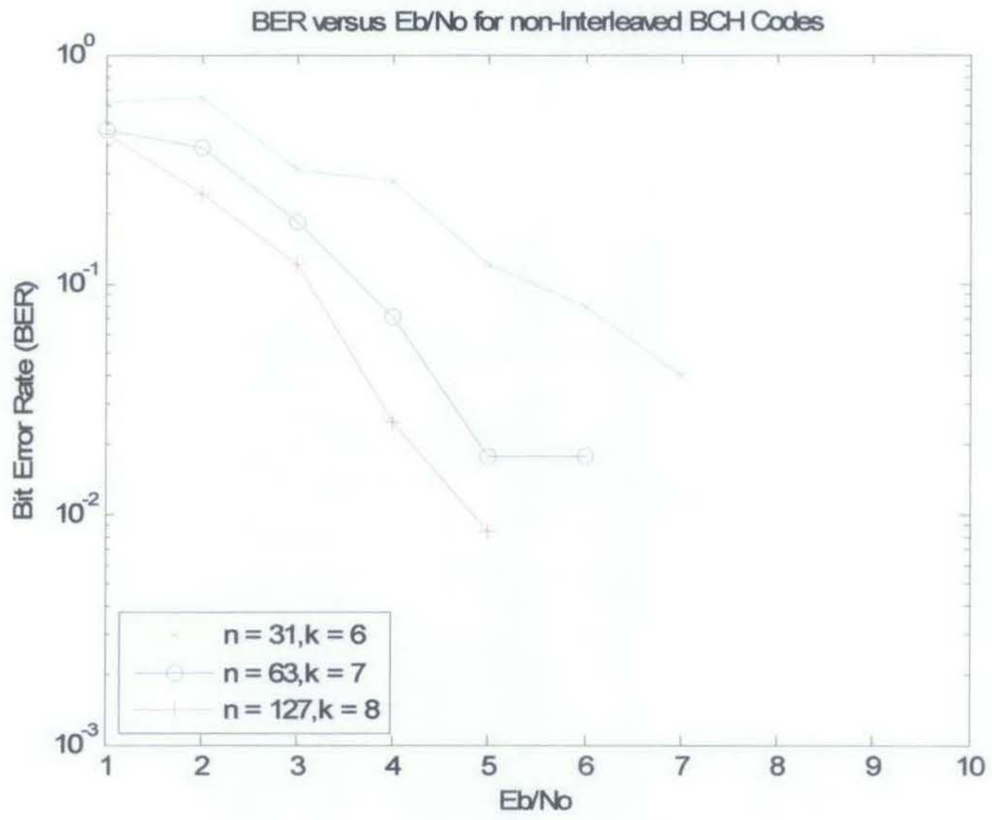


Figure 10: BER vs Eb/No for non-interleaved BCH codes for smaller error burst

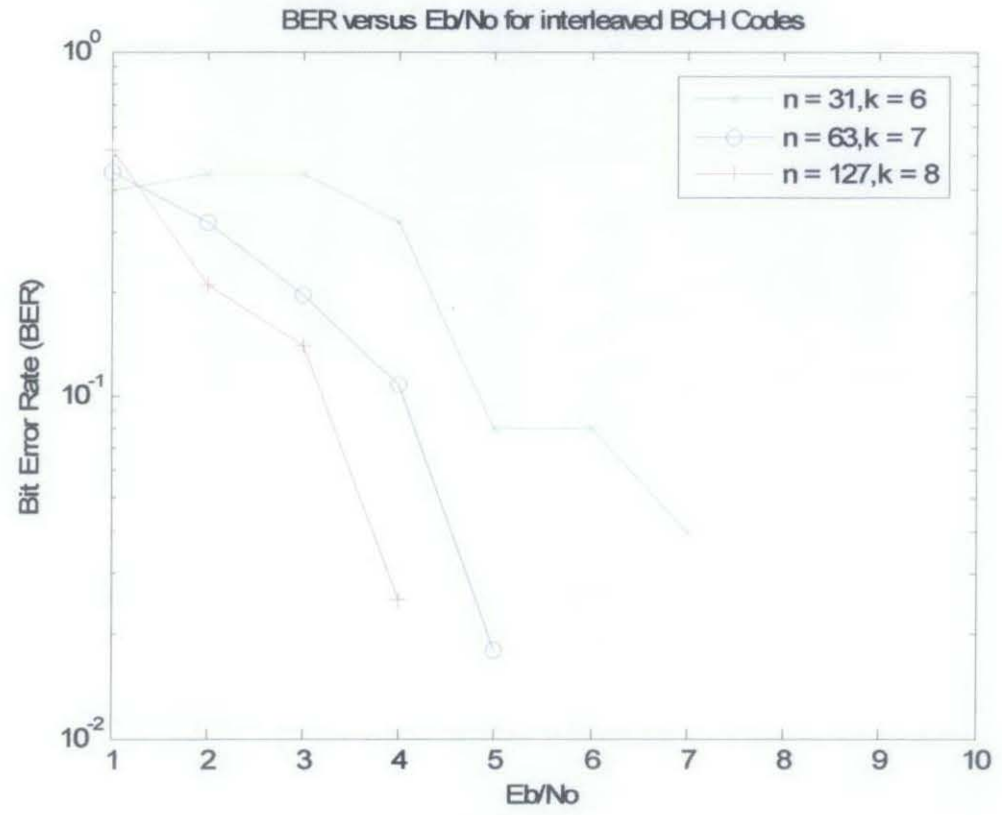


Figure 11: BER vs Eb/No for interleaved BCH codes for smaller error burst

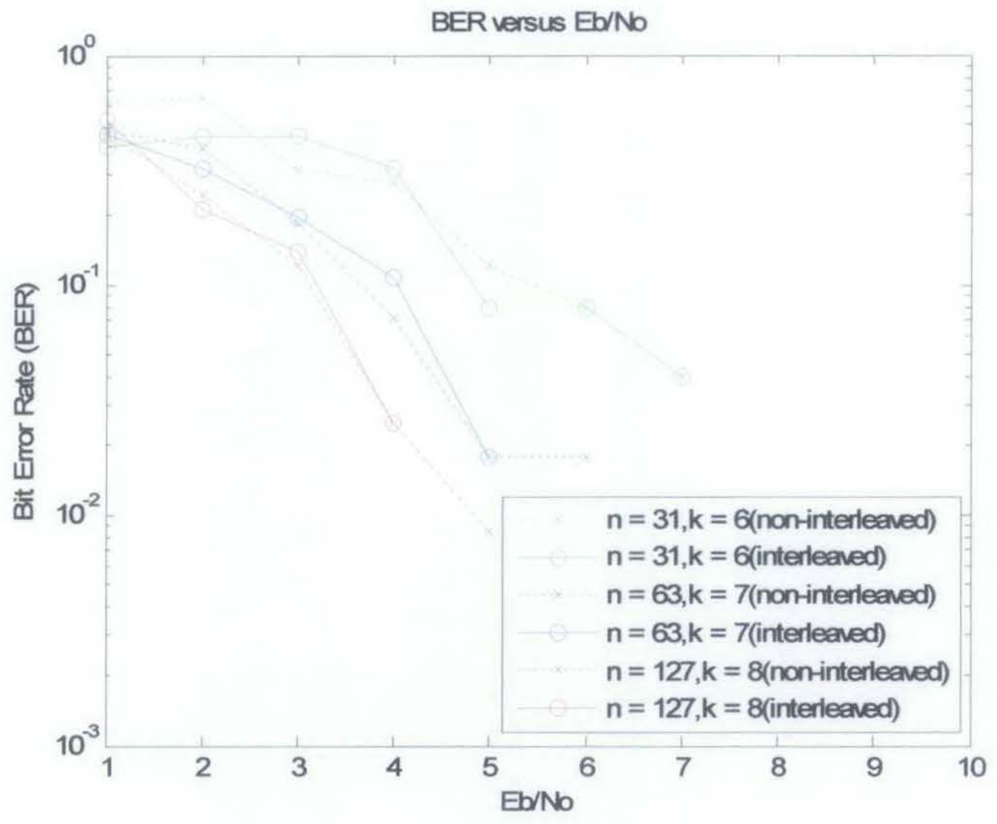


Figure 12: Comparison of BER vs SNR for interleaved and non-interleaved BCH Codes

4.2.1 Discussions

The result comparison analysis was performed. **Figure 10** shows the error performances for non-interleaved BCH codes which indicate that the higher the value of Signal to Noise Ratio (SNR), the lower the Bit Error Rate (BER). This proves the theoretical BER vs SNR shown in **Figure 14 (Appendix D)**. The same goes for interleaved BCH codes error performance shown in **Figure 11**. This is because, as SNR increases, the signal power becomes stronger compared to the noise power. Therefore, larger and clearer signal could be detected by the receiver.

Comparing between BCH codes performances, codeword, n = 31 shows the highest bit error rate while n = 127 shows the lowest bit error rate, which once again match the theoretical BER vs SNR graph.

From **Figure 12**, it could be seen that the BER for interleaved is lower than the BER for non-interleaved BCH codes. This shows that interleaved BCH codes is more efficient where no single data are missing at the receiver for a pack of data sent.

Instead, only a little part of a single data is missing as the data were interleaved before being sent through channel where burst of errors occurs. Thus, received message could be recovered easily. However, there are several points of SNR that shows a reverse of the BER performance of interleaved and non-interleaved BCH codes. This occurs due to the instability of burst errors included as each BCH code is provided with different amount of burst errors.

4.3 BLOCK LENGTH VS. SIMULATION TIME

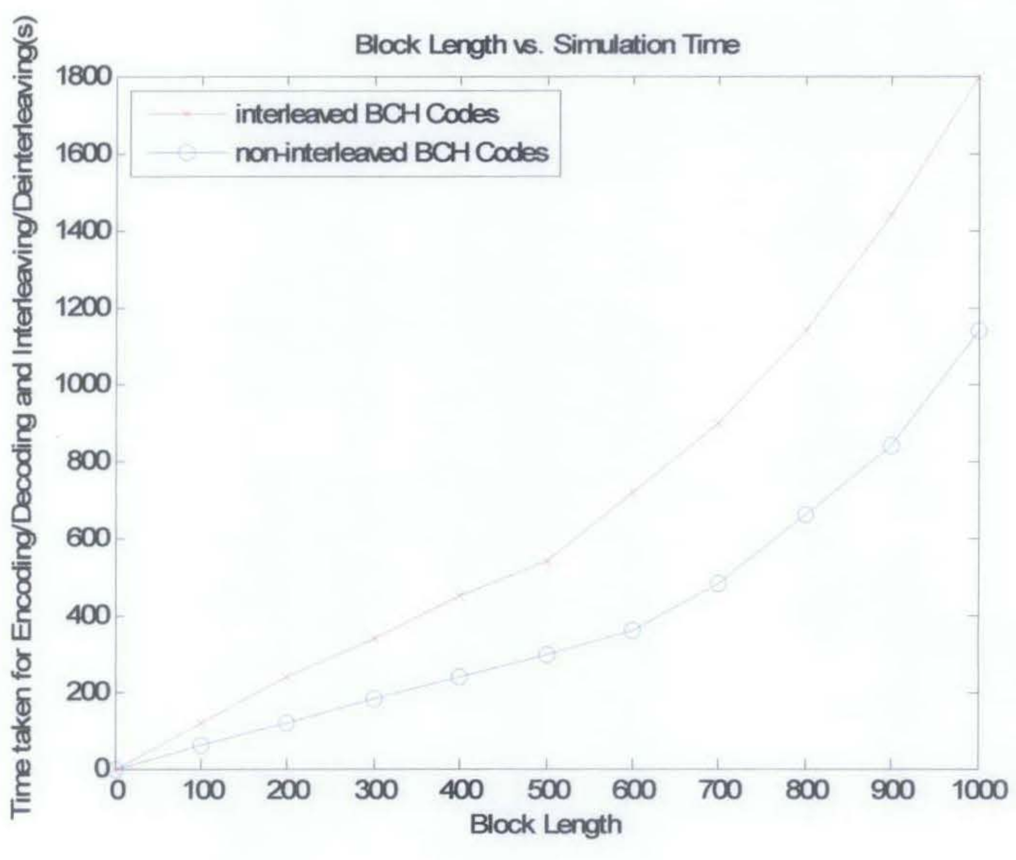


Figure 13: Block length versus simulation time

4.3.1 Discussions

Figure 13 shows that interleaved BCH codes took longer time for the additional interleaving/deinterleaving to the encoding/decoding compared to non-interleaved BCH codes. This is due to the interleaving codes process which adds to the codes complexity.

Interleaving BCH codes have more codes algorithm to be performed, which takes times. Interleaving as mentioned in *1.2.4* is a way to protect data transmission from burst errors. Encoded data is first interleaved and then burst errors are included. Interleaved message with burst errors then enters modulation and channel simulation and is deinterleaved before being decoded to obtain transmitted data.

From the graph, it could also be seen that non-interleaved BCH codes takes lesser time to complete. This is due to encoding/decoding was performed without interleaving, thus the codes algorithm is much simpler than interleaved BCH codes algorithm.

CHAPTER 5
CONCLUSION & RECOMMENDATION

5.1 CONCLUSION

Throughout this project, the author have learnt one of the powerful error correcting codes called BCH codes and affect with varying SNR and error size when combined with interleaving technique. BCH codes are used in data transmission over noisy transmission channel, while interleaving is a technique to protect transmission data from burst errors. This error detection would detect errors that are made due to noise during the transmission from the transmitter to the receiver and eliminate the noise. Interleaving helps improve the performance of error correcting process by rearranging encoded data randomly before being transmitted. Burst errors occurred during transmission where several data will be missing. When data was deinterleaved and decoded, received data can be recovered easily.

All stages of system methodology are applied and channel used is AWGN. Based on the simulation results, it shows that practical BER vs. SNR does not agree with theoretical because of the addition of error bursts for the practical simulation. Due to frequency of error bursts and the occurrence of other errors caused by channel noise, the interleaver should ideally be made as large as possible. Nevertheless, interleaver introduces delay into the transmission of the message signal. Therefore, the *nwords* x *k* array must be filled before it can be transmitted [16]. This is an issue in real-time applications concerning voice because it limits the udable block size of the interleaver and requires a compromise solution.

Simulation time increases as number of codewords processed increases. During simulation, long BCH codes consumes longer simulation period compared to short BCH codes due to number of codewords processed (nwords) i.e. *nwords* = *n* – *k*. However, for any BCH codes, longer simulation period also occurred if nwords is set to be very high for example *nwords* = 1000. The very long codeword length are producing good decoding efficiency, in other words, the longer the codeword that is sent through the channel modulation, the more accurate the decoded data received.

The implementations of interleaved BCH codes and non-interleaved BCH codes in Matlab simulation software for this Final Year project were nearly successful. The codes enabled us to analyze the error correction codes and interleaving in further detail and research were conducted successfully.

With interleaving, burst errors in forward error correction can be dispersed. Although, interleaving technique would be much more effective if applied in mobile fading channel instead of AWGN. This is due to burst of errors usually occurs in mobile fading environment.

5.2 RECOMMENDATIONS

1. Use QPSK, FSK as modulation.
2. Channel simulation with fast fading channel such as Binary Symmetric Channel, Rayleigh and Rician.
3. Input used to be digital image and video.

REFERENCES

[1] Lijun Zhang, Victor O. K. Li and Zhigang Cao, "Short BCH Codes for Wireless Multimedia Data", *IEEE,* pp. 220-222, 2002.

[2] Irving S.Reed and Xuemin Chen, "Error-Control Coding for Data Networks", *Kluwer Academic Publishers Boston/Dorrdrecht/London,* 1999.

[3] Hank Wallace, "*Error Detection and Correction Using the BCH Code*", 2001

[4] Wong Hui Yin, DEC "*A Comparison Study of LDPC and BCH Codes*", Final Year Proj., Universiti Teknologi PETRONAS, 2006

[5] Wikipedia, <http://en.wikipedia.org/wiki/Berlekamp-Massey_algorithm>, last accessed 30 April 2007

[6] Matlab Lecture 10, <http://www.aquaphoenix.com/lecture/matlab10/page5.html#10.4>, last accessed 18 Sept 2007

[7] Wikipedia, <http://en.wikipedia.org/wiki/Galois_theory>, last accessed 20 Sept 2007

[8] Z. Moe, X. X. Qiu, R. A. Scholtz, and V. O. K. Li, 1999 "ATM-based TH-SSMA network for multimedia PCS", *IEEE J. Select. Areas Commun.,* vol. 17, pp. 824-836.

[9] MATLAB 7.1 SP3, 2 August 2005 "*DCT and Image Compression*", Image Processing Toolbox.

[10] William Stallings, "*Data and Computer Communications*", Pearson Education 8th Ed., 2007

[11] MATLAB 7.1 SP3, 2 August 2005 "*Interleaver*", Communication Toolbox.

[12] MATLAB 7.1 SP3, 2 August 2005 "*Discrete Cosine Transform*", Image Processing Toolbox.

[13] Wikipedia, <http://en.wikipedia.org/wiki/error_detection_and_correction>

[14] Daniel j. Costello, Jr. and Shu Lin, "*Error Control Coding*", Prentice Hall 2nd Ed., 2004.

[15] P. R. Chang, and C. F. Lin, "Wireless ATM-based multicode CDMA transport architecture for MPEG-2 video transmission", *Proc. IEEE,* vol. 87, pp. 1807-1824, 1999.

[16] Michael Moher, Simon Haykin, "*Modern Wireless Communications*", Prentice Hall

[17] Yun Q. Shi, Xi Min Zhang, Zhi-Cheng Ni and Nirwan Ansari, "Interleaving for Combating Bursts of Errors" *Circuits and Systems Mag., IEEE,* pp. 29-42, first quarter 2004.

[18] S. A. Hanna, "Convolutional Interleaving for Digital Radio Communications" *Conf. IEEE,* pp 443 – 447, ICUPC '93.

[19] MATLAB 7.1 SP3, 2 August 2005, *"Galois array"* Communication Toolbox

[20] MATLAB 7.1 SP3, 2 August 2005, *"generator polynomial"* Communication Toolbox

[21] MATLAB 7.1 SP3, 2 August 2005, *"bchenc"* Communication Toolbox

[22] MATLAB 7.1 SP3, 2 August 2005, *"pskmod"* Communication Toolbox

[23] MATLAB 7.1 SP3, 2 August 2005, *"sum"* Communication Toolbox

[24] MATLAB 7.1 SP3, 2 August 2005, *"pskdemod"* Communication Toolbox

[25] MATLAB 7.1 SP3, 2 August 2005, *"randintrlv"* Communication Toolbox

[26] MATLAB 7.1 SP3, 2 August 2005, *"bitxor"* Communication Toolbox

[27] MATLAB 7.1 SP3, 2 August 2005, *"randdeintrlv"* Communication Toolbox

[28] MATLAB 7.1 SP3, 2 August 2005, *"bchdec"* Communication Toolbox

[29] MATLAB 7.1 SP3, 2 August 2005, *"biterr"* Communication Toolbox

[30] MATLAB 7.1 SP3, 2 August 2005, *"awgn"* Communication Toolbox

[31] Wikipedia, <http://en.wikipedia.org/wiki/Interleaving>, last accessed April 30, 2008

[32] Yuan Dongfeng, " The Estimating on Performance to Interleaved BCH Codes applied to the Mobile Communication Channel", *IEEE,* pp. 208 – 212

[33] R. Xiujie, W. Chengxiang, Y. Dongfeng and Y. Qi, " Performance Research of the Convolutional Code Using a Novel Interleaving Scheme in Mobile Image Communication Systems and the Comparison with Interleaved BCH Code", *IEEE*, pp. 998 – 1001.

APPENDIX A
SOURCE CODE

```
clear; clc;
st1 = rand('state');  st2=rand('state'); % States for random number
generator

% Codeword length and message length
for n = 31
    k = 6;
    nwords = n-k; %number of words to process

    msg = gf(randint(nwords,k,2,st1));
    x_in = double(msg.x);
    [genpoly,t] = bchgenpoly(n,k); % t is error-correction
    capability
    code = bchenc(msg,n,k); % Encode the data
    A = double(code.x);

    % Create a burst error that will corrupt two adjacent codewords.
    errors = zeros(size(code));

    for snr = 1:10
        x(snr,1) = snr;
        errors(1:100) = [1 1 1 1 1 1 1 1 ...........1 1 1 1 1 1 1 1 1 ];
        errors(101:174) = [1 1 1 1 1 1 1 ...........1 1 1 1 1 1 1 1 1 ];

        % With Interleaving
        %-------------------
        B = randintrlv(A,st2); % Interleave.
        inter_err = bitxor(B,errors); % Include burst error.

        % Enter modulation and channel simulation
        mod = pskmod(inter_err,2);
        channel = awgn(mod,snr);
        C = pskdemod(channel,2);

        D = randdeintrlv(C,st2); % Deinterleave.
        deinter_gf = gf(D);
        [newmsg1,err1,ccode1] = bchdec(deinter_gf,n,k); % Decode
        x_out_with = double(newmsg1.x);

        %disp('Number of errors and error rate, with
        interleaving:');
        [number_with,rate_with] = biterr(x_in,x_out_with); % Error
        statistics
        zi1(snr,1) = number_with;
        yi1(snr,1) = rate_with;

        % Without Interleaving
        %---------------------
        code_err = bitxor(A,errors); % Include burst error.

        % Enter modulation and channel simulation
        mod = pskmod(code_err,2);
        channel = awgn(mod,snr);
        r = pskdemod(channel,2);

        r2 = gf(r);
        [newmsg2,err2,ccode2] = bchdec(r2,n,k); % Decode
        x_out_without = double(newmsg2.x);
        %disp('Number of errors and error rate, without
        interleaving:');
        [number_without,rate_without] = biterr(x_in,x_out_without);
        % Error statistics
        z1(snr,1) = number_without;
        y1(snr,1) = rate_without;
    end
end

n
k
disp('[x,y1,yi1]');
[x,y1,yi1]

for n = 63
    k = 7;
    nwords = n-k; %number of words to process

    msg = gf(randint(nwords,k,2,st1));
    x_in = double(msg.x);
    [genpoly,t] = bchgenpoly(n,k); % t is error-correction
    capability
    code = bchenc(msg,n,k); % Encode the data
    A = double(code.x);

    % Create a burst error that will corrupt two adjacent codewords.
    errors = zeros(size(code));

    for snr = 1:10
        x(snr,1) = snr;
        errors(10:399) = [1 1 1 1 1 ........1 1 1 1 1 1 1 1 1 1 1 1 1 1];
        .
        .
        errors(700:784) = [1 1 1 1 1 ........1 1 1 1 1 1 1 1 1 1 1 1 1];

        % With Interleaving
        %------------------
        B = randintrlv(A,st2); % Interleave.
        inter_err = bitxor(B,errors); % Include burst error.

        % Enter modulation and channel simulation
        mod = pskmod(inter_err,2);
        channel = awgn(mod,snr);
        C = pskdemod(channel,2);

        D = randdeintrlv(C,st2); % Deinterleave.
        deinter_gf = gf(D);
        [newmsg1,err1,ccode1] = bchdec(deinter_gf,n,k); % Decode
        x_out_with = double(newmsg1.x);

        %disp('Number of errors and error rate, with
        interleaving:');
        [number_with,rate_with] = biterr(x_in,x_out_with); % Error
        statistics
        zi2(snr,1) = number_with;
        yi2(snr,1) = rate_with;

        % Without Interleaving
        %---------------------
        code_err = bitxor(A,errors); % Include burst error.

        % Enter modulation and channel simulation
        mod = pskmod(code_err,2);
        channel = awgn(mod,snr);
        r = pskdemod(channel,2);

        r2 = gf(r);
        [newmsg2,err2,ccode2] = bchdec(r2,n,k); % Decode
        x_out_without = double(newmsg2.x);
        %disp('Number of errors and error rate, without
        interleaving:');
        [number_without,rate_without] = biterr(x_in,x_out_without);
        % Error statistics
        z2(snr,1) = number_without;
        y2(snr,1) = rate_without;
    end
end

n
k
disp('[x,y2,yi2]');
[x,y2,yi2]

for n = 127
    k = 8;
    nwords = n-k; %number of words to process

    msg = gf(randint(nwords,k,2,st1));
    x_in = double(msg.x);
    [genpoly,t] = bchgenpoly(n,k); % t is error-correction
    capability
    code = bchenc(msg,n,k); % Encode the data
    A = double(code.x);

    % Create a burst error that will corrupt two adjacent codewords.
    errors = zeros(size(code));

    for snr = 1:10
        x(snr,1) = snr;
        errors(10:399) = [1 1 1 1 1 1 1 ………1 1 1 1 1 1 1 1 1 1 1 1];
        .
        .
        errors(3600:3299) = [1 1 1 1 1 1 1 1 ……1 1 1 1 1 1 1 1 1 1];

        % With Interleaving
        %------------------
        B = randintrlv(A,st2); % Interleave.
        inter_err = bitxor(B,errors); % Include burst error.

        % Enter modulation and channel simulation
        mod = pskmod(inter_err,2);
        channel = awgn(mod,snr);
        C = pskdemod(channel,2);

        D = randdeintrlv(C,st2); % Deinterleave.
        deinter_gf = gf(D);
        [newmsg1,err1,ccode1] = bchdec(deinter_gf,n,k); % Decode
        x_out_with = double(newmsg1.x);

        %disp('Number of errors and error rate, with
        interleaving:');
        [number_with,rate_with] = biterr(x_in,x_out_with); % Error
        statistics
        zi3(snr,1) = number_with;
        yi3(snr,1) = rate_with;

        % Without Interleaving
        %----------------------
        code_err = bitxor(A,errors); % Include burst error.

        % Enter modulation and channel simulation
        mod = pskmod(code_err,2);
        channel = awgn(mod,snr);
        r = pskdemod(channel,2);

        r2 = gf(r);
        [newmsg2,err2,ccode2] = bchdec(r2,n,k); % Decode
        x_out_without = double(newmsg2.x);
        %disp('Number of errors and error rate, without
        interleaving:');
        [number_without,rate_without] = biterr(x_in,x_out_without);
        % Error statistics
        z3(snr,1) = number_without;
        y3(snr,1) = rate_without;
    end
end

n
k
disp('[x,y3,yi3]');
[x,y3,yi3]

figure, semilogy(x,y1,'-xr', x,y2,'-ob', x,y3,'-+g')
h = legend('n = 31,k = 6','n = 63,k = 7', 'n = 127,k = 8',3);
set(h,'Interpreter','none')
xlabel('Eb/No');
ylabel('Bit Error Rate (BER)');
title('BER versus Eb/No for non-Interleaved BCH Codes');

figure, semilogy(x,yi1,'-xr', x,yi2,'-ob', x,yi3,'-+g')
h = legend('n = 31,k = 6','n = 63,k = 7', 'n = 127,k = 8',3);
set(h,'Interpreter','none')
xlabel('Eb/No');
ylabel('Bit Error Rate (BER)');
title('BER versus Eb/No for interleaved BCH Codes');

figure, semilogy(x,y1,':xr',x,yi1,'-or', x,y2,':xb', x,yi2,'-ob',
x,y3,':xg', x,yi3,'-og')
h = legend('n = 31,k = 6(non-interleaved)','n = 31,k =
6(interleaved)', 'n = 63,k = 7(non-interleaved)','n = 63,k =
7(interleaved)',  'n = 127,k = 8(non-interleaved)', 'n = 127,k =
8(interleaved)',6);
set(h,'Interpreter','none')
xlabel('Eb/No');
ylabel('Bit Error Rate (BER)');
title('BER versus Eb/No');
```

APPENDIX B
RESULT

```
n =

    31


k =

     6

[x,y1,yi1]

ans =

    1.0000    0.6200    0.4000
    2.0000    0.6533    0.4400
    3.0000    0.3133    0.4400
    4.0000    0.2800    0.3200
    5.0000    0.1200    0.0800
    6.0000    0.0800    0.0800
    7.0000    0.0400    0.0400
    8.0000         0         0
    9.0000         0         0
   10.0000         0         0


n =

    63


k =

     7

[x,y2,yi2]

ans =
     1.0000    0.4668    0.4464
     2.0000    0.3903    0.3189
     3.0000    0.1862    0.1964
     4.0000    0.0714    0.1071
     5.0000    0.0179    0.0179
     6.0000    0.0179         0
     7.0000         0         0
     8.0000         0         0
     9.0000         0         0
    10.0000         0         0


n =

   127


k =

     8

 [x,y3,yi3]

ans =

     1.0000    0.4548    0.5189
     2.0000    0.2458    0.2122
     3.0000    0.1208    0.1408
     4.0000    0.0252    0.0252
     5.0000    0.0084       0
     6.0000      0          0
     7.0000      0          0
     8.0000      0          0
     9.0000      0          0
     10.0000     0          0
```

APPENDIX C
SUBROUTINE MATLAB SOURCE CODE

1) bchgenpoly.m

```
% Initial checks
error(nargchk(2,3,nargin));

t = bchnumerr(N,K);
t2 = 2*t;

prim_poly = 1;

m = log2(N+1);

if ~isempty(varargin)
    prim_poly = varargin{1};
    % Check prim_poly
    if isempty(prim_poly)
        if ~isnumeric(prim_poly)
            error('To use the default PRIM_POLY, it must be marked
by [].');
        end
    else
        if ~isnumeric(prim_poly) || ~isscalar(prim_poly) ||
(floor(prim_poly) ~= prim_poly)
            error('PRIM_POLY must be a scalar integer.');
        end

        if ~isprimitive(prim_poly)
            error('PRIM_POLY must be a primitive polynomial.');
        end
    end

end

% Determine the cosets for this field
if prim_poly == 1
    coset = cosets(m,[],'nodisplay');
else
    coset = cosets(m,prim_poly,'nodisplay');
end

% For each coset that contains a power of alpha < 2t, add the
corresponding
% minimum polynomial to the list of minimum polynomials. Then
convolve all the
% minimum polynomials to make the generator polynomial.
minpol_list = [];
for idx1 = 2 : numel(coset)
    if(any(find(log(coset{idx1})<t2)))  % coset contains a power of
alpha < 2t

        % Compute the minimum polynomial for this coset
        tempPoly = 1;
        thisCoset = coset{idx1};
        for idx2 = 1 : length(thisCoset);
            tempPoly = conv(tempPoly, [1 thisCoset(idx2)]);
        end

        % Zero pad polynomial if necessary
        minPol = gf([zeros(1,m+1-length(tempPoly))  tempPoly.x],1);

        % add polynomial to list
        minpol_list = [minpol_list;minPol];
    end
end

% Convolve all the rows of the minpol_list with each other.
len = size(minpol_list,1);
genpoly  = 1;
for i = 1:len,
    genpoly = conv(genpoly,minpol_list(i,:));
end

% Strip any leading zeros
% The size of the generator polynomial should be N-K+1
genpoly = genpoly( end-(N-K) :end);
```

2) bchenc.m

```
% Initial checks
error(nargchk(3,4,nargin));

% Number of optional input arguments
nvarargin = nargin - 3;

% % Fundamental checks on parameter data types
if ~isa(msg,'gf')
   error('MSG must be a Galois array.');
end

if(msg.m ~=1)
    error('MSG must be in GF(2).');
end

%set and check the parity position
if(nargin>3)
    parityPos = varargin{1};
else
    parityPos = 'end';
end

if( ~strcmp(parityPos,'beginning') && ~strcmp(parityPos, 'end') )
    error('PARITYPOS must be either ''beginning'' or ''end''  ')
end


[m_msg, n_msg] = size(msg);

if (n_msg ~= K)
    error('The message length must equal K.')
end
```

```
% get the generator polynomial
genpoly = bchgenpoly(N,K);

% get the generator matrix
[h, gen] = cyclgen(N, (double(genpoly.x)));

% do the coding
code = msg * gen;

% rearrange parity if necessary
%if(isempty(varargin) || strcmp(lower(varargin{1}), 'beginning'))
if(strcmp(parityPos, 'end'))
    code = [code(:,N-K+1:end), code(:,1:N-K)];
end
```

3) <u>**randintrlv.m**</u>

```
% --- Usual error checks
error(nargchk(2,2,nargin));
error(nargoutchk(0,1,nargout));

data_size = size(data);          % Obtains size of DATA
orig_data = data;
dimState  = size(state);         % Obtains dims of STATE

% --- Checks if DATA is 1-D row vector
if (data_size(1) == 1)
    data = data(:);              % Converts sequence in DATA to a
column vector
    data_size = size(data);
end

% --- Error checking on input arguments
if isempty(data)
    error('comm:randintrlv:DataIsEmpty','DATA cannot be empty.')
end

if (~isnumeric(data) && ~isa(data,'gf'))
    error('comm:randintrlv:DataIsNotNumeric','DATA must be
numeric.');
end

if isempty(state)
    error('comm:randintrlv:StateIsEmpty','STATE cannot be empty.')
end

if ~isnumeric(state)
    error('comm:randintrlv:StateIsNotNumeric','STATE must be
numeric.')
end

if ~(all(dimState == [1 1]) || all(dimState == [35 1]))
    error('comm:randintrlv:InvalidState','STATE must be scalar or
35-by-1.')
end

% Get the current state of rand, for restoral purposes later
originalState = rand('state');
```

```
rand('state',state);                    % Set the current state of
the uniform generator
int_vec = (randperm(data_size(1)));     % Return a random
permutation of the integers 1:data_size(2)

% Reset the state of rand to its original state
rand('state', originalState);

% --- Reorder sequence of symbols
intrlved = intrlv(orig_data,int_vec);
```

4) pskmod.m

```
% Error checks
if(nargin<2)
    error('comm:pskmod:numarg','Too few input arguments.');
end

if (nargin > 4)
    error('comm:pskmod:numarg', 'Too many input arguments. ');
end

% Check that x is a positive integer
if (~isreal(x) || any(any(ceil(x) ~= x)) || ~isnumeric(x))
    error('comm:pskmod:xreal', 'Elements of input X must be integers
in the range [0, M-1].');
end

% Check that M is a positive integer
if (~isreal(M) || ~isscalar(M) || M<=0 || (ceil(M)~=M) ||
~isnumeric(M))
    error('comm:pskmod:Mreal', 'M must be a scalar positive
integer.');
end

% Check that M is of the form 2^K
if(~isnumeric(M) || (ceil(log2(M)) ~= log2(M)))
    error('comm:pskmod:Mpow2', 'M must be in the form of M = 2^K,
where K is an integer. ');
end

% Check that x is within range
if ((min(min(x)) < 0) || (max(max(x)) > (M-1)))
    error('comm:pskmod:xreal', 'Elements of input X must be integers
in [0, M-1].');
end

% Determine initial phase. The default value is 0
if (nargin >= 3)
    ini_phase = varargin{1};
    if (isempty(ini_phase))
        ini_phase = 0;
    elseif (~isreal(ini_phase) || ~isscalar(ini_phase))
        error('comm:pskmod:ini_phaseReal', 'INI_PHASE must be a real
scalar. ');
    end
else
    ini_phase = 0;
end
```

```
% Check SYMBOL_ORDER
if(nargin==2 || nargin==3)
   Symbol_Ordering = 'bin'; % default
else
    Symbol_Ordering = varargin{2};
    if (~ischar(Symbol_Ordering)) ||
(~strcmpi(Symbol_Ordering,'GRAY')) &&
(~strcmpi(Symbol_Ordering,'BIN'))
        error('comm:pskmod:SymbolOrder','Invalid symbol set
ordering.');
    end
end

% --- Assure that X, if one dimensional, has the correct orientation
--- %
wid = size(x,1);
if (wid == 1)
    x = x(:);
end

% Gray encode if necessary
if (strcmpi(Symbol_Ordering,'GRAY'))
    [x_gray,gray_map] = bin2gray(x,'psk',M);   % Gray encode
    [tf,index]=ismember(x,gray_map);
     x=index-1;
end

% Evaluate the phase angle based on M and the input value. The phase
angle
% lies between 0 - 2*pi.
theta = 2*pi*x/M;

% The complex envelope is (cos(theta) + j*sin(theta)). This can be
% expressed as exp(j*theta). If there is an initial phase, it is
added
% to the existing phase angle
y = exp(j*(theta + ini_phase));

% --- modulator output must be complex
if isreal(y)
    y = complex(y,0);
end

% --- restore the output signal to the original orientation --- %
if(wid == 1)
    y = y.';
end
```

5) awgn.m

```
% --- Initial checks
error(nargchk(2,5,nargin));

% --- Value set indicators (used for the string flags)
pModeSet    = 0;
measModeSet = 0;

% --- Set default values
reqSNR   = [];
```

```
sig      = [];
sigPower = 0;
pMode    = 'db';
measMode = 'specify';
state    = [];

% --- Placeholder for the signature string
sigStr = '';

% --- Identify string and numeric arguments
for n=1:nargin
   if(n>1)
      sigStr(size(sigStr,2)+1) = '/';
   end
   % --- Assign the string and numeric flags
   if(ischar(varargin{n}))
      sigStr(size(sigStr,2)+1) = 's';
   elseif(isnumeric(varargin{n}))
      sigStr(size(sigStr,2)+1) = 'n';
   else
      error('Only string and numeric arguments are allowed.');
   end
end

% --- Identify parameter signatures and assign values to variables
switch sigStr
   % --- awgn(x, snr)
   case 'n/n'
      sig      = varargin{1};
      reqSNR   = varargin{2};

   % --- awgn(x, snr, sigPower)
   case 'n/n/n'
      sig      = varargin{1};
      reqSNR   = varargin{2};
      sigPower = varargin{3};

   % --- awgn(x, snr, 'measured')
   case 'n/n/s'
      sig      = varargin{1};
      reqSNR   = varargin{2};
      measMode = lower(varargin{3});

      measModeSet = 1;

   % --- awgn(x, snr, sigPower, state)
   case 'n/n/n/n'
      sig      = varargin{1};
      reqSNR   = varargin{2};
      sigPower = varargin{3};
      state    = varargin{4};

   % --- awgn(x, snr, 'measured', state)
   case 'n/n/s/n'
      sig      = varargin{1};
      reqSNR   = varargin{2};
      measMode = lower(varargin{3});
      state    = varargin{4};

      measModeSet = 1;
   % --- awgn(x, snr, sigPower, 'db|linear')
   case 'n/n/n/s'
      sig      = varargin{1};
      reqSNR   = varargin{2};
      sigPower = varargin{3};
      pMode    = lower(varargin{4});

      pModeSet = 1;

   % --- awgn(x, snr, 'measured', 'db|linear')
   case 'n/n/s/s'
      sig      = varargin{1};
      reqSNR   = varargin{2};
      measMode = lower(varargin{3});
      pMode    = lower(varargin{4});

      measModeSet = 1;
      pModeSet    = 1;

   % --- awgn(x, snr, sigPower, state, 'db|linear')
   case 'n/n/n/n/s'
      sig      = varargin{1};
      reqSNR   = varargin{2};
      sigPower = varargin{3};
      state    = varargin{4};
      pMode    = lower(varargin{5});

      pModeSet = 1;

   % --- awgn(x, snr, 'measured', state, 'db|linear')
   case 'n/n/s/n/s'
      sig      = varargin{1};
      reqSNR   = varargin{2};
      measMode = lower(varargin{3});
      state    = varargin{4};
      pMode    = lower(varargin{5});

      measModeSet = 1;
      pModeSet    = 1;

   otherwise
      error('Syntax error.');
end

% --- Parameters have all been set, either to their defaults or by
the values passed in,
%     so perform range and type checks

% --- sig
if(isempty(sig))
   error('An input signal must be given.');
end

if(ndims(sig)>2)
   error('The input signal must have 2 or fewer dimensions.');
end

% --- measMode
if(measModeSet)
```

```
   if(~strcmp(measMode,'measured'))
      error('The signal power parameter must be numeric or
''measured''.');
   end
end

% --- pMode
if(pModeSet)
   switch pMode
   case {'db' 'linear'}
   otherwise
      error('The signal power mode must be ''db'' or ''linear''.');
   end
end

% -- reqSNR
if(any([~isreal(reqSNR) (length(reqSNR)>1) (length(reqSNR)==0)]))
   error('The signal-to-noise ratio must be a real scalar.');
end

if(strcmp(pMode,'linear'))
   if(reqSNR<=0)
      error('In linear mode, the signal-to-noise ratio must be >
0.');
   end
end

% --- sigPower
if(~strcmp(measMode,'measured'))

   % --- If measMode is not 'measured', then the signal power must
be specified
   if(any([~isreal(sigPower) (length(sigPower)>1)
(length(sigPower)==0)]))
      error('The signal power value must be a real scalar.');
   end

   if(strcmp(pMode,'linear'))
      if(sigPower<0)
         error('In linear mode, the signal power must be >= 0.');
      end
   end

end

% --- state
if(~isempty(state))
   if(any([~isreal(state) (length(state)>1) (length(state)==0)
any((state-floor(state))~=0)]))
      error('The State must be a real, integer scalar.');
   end
end

% --- All parameters are valid, so no extra checking is required

% --- Check the signal power.  This needs to consider power
measurements on matrices
if(strcmp(measMode,'measured'))
   sigPower = sum(abs(sig(:)).^2)/length(sig(:));
   if(strcmp(pMode,'db'))
      sigPower = 10*log10(sigPower);
   end
end

% --- Compute the required noise power
switch lower(pMode)
   case 'linear'
      noisePower = sigPower/reqSNR;
   case 'db'
      noisePower = sigPower-reqSNR;
      pMode = 'dbw';
end

% --- Add the noise
if(isreal(sig))
   opType = 'real';
else
   opType = 'complex';
end

y = sig+wgn(size(sig,1), size(sig,2), noisePower, 1, state, pMode,
opType);
```

6) pskdemod.m

```
% Error checks
if(nargin<2)
    error('comm:pskdemod:numarg','Too few input arguments.');
end

if (nargin > 4)
    error('comm:pskdemod:numarg', 'Too many input arguments. ');
end

%Check y, m
if( ~isnumeric(y))
    error('comm:pskdemod:Ynum','Y must be numeric.');
end

% Checks that M is positive integer
if (~isreal(M) || ~isscalar(M) || M<=0 || (ceil(M)~=M) ||
~isnumeric(M))
    error('comm:pskdemod:Mreal','M must be a scalar positive
integer.');
end

% Checks that M is in of the form 2^K
if(~isnumeric(M) || (ceil(log2(M)) ~= log2(M)))
    error('comm:pskdemod:Mpow2', 'M must be in the form of M = 2^K,
where K is an integer. ');
end

% Determine INI_PHASE. The default value is 0
if (nargin >= 3)
    ini_phase = varargin{1};
    if (isempty(ini_phase))
        ini_phase = 0;
    elseif (~isreal(ini_phase) || ~isscalar(ini_phase))
```

```
        error('comm:pskdemod:Ini_phaseReal', 'INI_PHASE must be a
real scalar. ');
    end
else
    ini_phase = 0;
end

% Check SYMBOL_ORDER
if(nargin==2 || nargin==3 )
   Symbol_Ordering = 'bin'; % default
else
    Symbol_Ordering = varargin{2};
    if (~ischar(Symbol_Ordering)) ||
(~strcmpi(Symbol_Ordering,'GRAY')) &&
(~strcmpi(Symbol_Ordering,'BIN'))
        error('comm:pskdemod:SymbolOrder','Invalid symbol set
ordering.');
    end
end

% End error checks

% Assure that Y, if one dimensional, has the correct orientation
wid = size(y,1);
if(wid==1)
    y = y(:);
end

% De-rotate
y = y .* exp(-i*ini_phase);

% Demodulate
normFactor = M/(2*pi); % normalization factor to convert from PI-
domain to
                       % linear domain
% convert input signal angle to linear domain; round the value to
get ideal
% constellation points
z = round((angle(y) .* normFactor));
% move all the negative integers by M
z(z < 0) = M + z(z < 0);

% --- restore the output signal to the original orientation --- %
if(wid == 1)
    z = z';
end

% Gray decode if necessary
if (strcmpi(Symbol_Ordering,'GRAY'))
    [z_degray,gray_map] = gray2bin(z,'psk',M);   % Gray decode
    % --- Assure that X, if one dimensional, has the correct
orientation --- %
    if(size(z,1) == 1)
        temp = zeros(size(y));
        temp(:) = gray_map(z+1);
        z(:) = temp(:);
    else
        z = gray_map(z+1);
    end
end
```

7) randdeintrlv.m

```
% --- Usual error checks
error(nargchk(2,2,nargin));
error(nargoutchk(0,1,nargout));

data_size = size(data);         % Obtains size of DATA
orig_data = data;
dimState  = size(state);        % Obtains dims of STATE

% --- Checks if DATA is 1-D column vector
if (data_size(1) == 1)
    data = data(:);             % Converts sequence in DATA to a
column vector
    data_size = size(data);
end

% --- Error checking on input arguments
if isempty(data)
    error('comm:randdeintrlv:DataIsEmpty','DATA cannot be empty.')
end

if (~isnumeric(data) && ~isa(data,'gf'))
    error('comm:randdeintrlv:DataIsNotNumeric','DATA must be
numeric.');
end

if isempty(state)
    error('comm:randdeintrlv:StateIsEmpty','STATE cannot be empty.')
end

if ~isnumeric(state)
    error('comm:randdeintrlv:StateIsNotNumeric','STATE must be
numeric.')
end

if ~(all(dimState == [1 1]) || all(dimState == [35 1]))
    error('comm:randdeintrlv:InvalidState','STATE must be scalar or
35-by-1.')
end

% Get the current state of rand, for restoral purposes later
originalState = rand('state');

rand('state',state);                    % Set the current state of
the uniform generator
int_vec = (randperm(data_size(1)));     % Return a random
permutation of the integers 1:data_size(2)

% Reset the state of rand to its original state
rand('state', originalState);

% --- Rearrange sequence of symbols
deintrlved = deintrlv(orig_data,int_vec);
```

8) bchdec.m

```
error(nargchk(3,4,nargin));

% Fundamental checks on parameter data types
if ~isa(code,'gf')
   error('CODE must be a Galois array.');
end

if(code.m~=1)
    error('Code must be in GF(2).');
end

% Check mandatory parameters : code, N, K, t

% --- code
if isempty(code.x)
    error('CODE must be a nonempty Galois array.');
end;

% --- width of code
[m_code, n_code] = size(code);
if N ~= n_code
    error('CODE must be either a N-element row vector or a matrix
with N columns.');
end

% Set and check the parity position
if(nargin>3)
    parityPos = varargin{1};
else
    parityPos = 'end';
end

if( ~strcmp(parityPos,'beginning') && ~strcmp(parityPos, 'end') )
    error('PARITYPOS must be either ''beginning'' or ''end''  ')
end

% Get the number of errors we can correct
t = bchnumerr(N,K);

% Bring the code word into the extension field
M = log2(N+1);
code = gf(code.x,M);

% Ensure that the code format into the berlekamp function is [msg
parity], since
% the function works only in that mode.  The berlekamp function also
takes care
% of prepending zeros for shortened codes.
if strcmp(parityPos, 'beginning')
    code = [code(:,N-K+1:n_code) code(:,1:N-K)];
end

% Pre-allocate memory.  Each element in this column vector holds the
number of
% errors in the corresponding row
decoded = gf(zeros(m_code, K));
cnumerr = zeros(m_code,1);
ccode   = gf(zeros(m_code, N));
```

```
for j = 1 : m_code,

    % Call to core algorithm BERLEKAMP
    inputCode    = code(j,:);
    inputCodeVal = inputCode.x;
    b            = 1;  % narrow-sense codeword
    shortened    = 0;  % no shortened codewords
    inWidth      = length(code(j,:));
    [decodedInt cnumerr(j) ccodeInt] = berlekamp(inputCodeVal, ...
                                                 N, ...
                                                 K, ...
                                                 M, ...
                                                 t, ...
                                                 b, ...
                                                 shortened, ...
                                                 inWidth);
    decoded(j,:) = gf(decodedInt);
    ccode(j,:)   = gf(ccodeInt);
end

% If necessary, flip message and parity symbols in corrected
codeword, undoing
% the flip prior to decoding.
if strcmp(parityPos, 'beginning')
    ccode = [ccode(:,K+1:n_code) ccode(:,1:K)];  %#ok
end
```

9) biterr.m

```
% --- Typical error checking.
error(nargchk(2,4,nargin));

% --- Placeholder for the signature string.
sigStr = '';
flag = '';
K = [];

% --- Identify string and numeric arguments
for n=1:nargin
    if(n>1)
        sigStr(size(sigStr,2)+1) = '/';
    end
    % --- Assign the string and numeric flags
    if(ischar(varargin{n}))
        sigStr(size(sigStr,2)+1) = 's';
    elseif(isnumeric(varargin{n}))
        sigStr(size(sigStr,2)+1) = 'n';
    else
        error('Only string and numeric arguments are accepted.');
    end
end

% --- Identify parameter signitures and assign values to variables
switch sigStr
    % --- biterr(a, b)
    case 'n/n'
        a       = varargin{1};
        b       = varargin{2};
        % --- biterr(a, b, K)
    case 'n/n/n'
        a        = varargin{1};
        b        = varargin{2};
        K        = varargin{3};


        % --- biterr(a, b, flag)
    case 'n/n/s'
        a        = varargin{1};
        b        = varargin{2};
        flag     = varargin{3};

        % --- biterr(a, b, K, flag)
    case 'n/n/n/s'
        a        = varargin{1};
        b        = varargin{2};
        K        = varargin{3};
        flag     = varargin{4};

        % --- biterr(a, b, flag, K)
    case 'n/n/s/n'
        a        = varargin{1};
        b        = varargin{2};
        flag     = varargin{3};
        K        = varargin{4};

        % --- If the parameter list does not match one of these
signatures.
    otherwise
        error('Syntax error.');
end

if (isempty(a)) || (isempty(b))
    error('Required parameter empty.');
end

if ~(min(min(isfinite(a))) && min(min(isfinite(b)))) || ~(isreal(a)
& isreal(b)) || max(max(a<0)) || max(max(b<0)) ||
max(max(floor(a)~=a)) || max(max(floor(b)~=b))
    error('Inputs must be finite, real, positive integers.');
end

% Determine the sizes of the input matrices.
[am, an] = size(a);
[bm, bn] = size(b);

% If one of the inputs is a vector, it can be either the first or
second input.
% This conditional swap ensures that the first input is the matrix
and the second is the vector.
if ((am == 1) && (bm > 1)) || ((an == 1) && (bn > 1))
    [a, b] = deal(b, a);
    [am, an] = size(a);
    [bm, bn] = size(b);
end

% Check the sizes of the inputs to determine the default mode of
operation.
if ((bm == 1) && (am > 1))
```

```
    default_mode = 'row-wise';
    if (an ~= bn)
        error('Input row vector must contain as many elements as
there are columns in the input matrix.');
    end
elseif ((bn == 1) && (an > 1))
    default_mode = 'column-wise';
    if (am ~= bm)
        error('Input column vector must contain as many elements as
there are rows in the input matrix.');
    end
else
    default_mode = 'overall';
    if (am ~= bm) || (an ~= bn)
        error('Input matrices must be the same size.');
    end
end

% Check that the user specified mode of operation is valid.
if isempty(flag)
    flag = default_mode;
elseif ~(strcmp(flag,'column-wise') || strcmp(flag,'row-wise') ||
strcmp(flag,'overall'))
    error('Invalid string flag.');
elseif strcmp(default_mode,'row-wise') && strcmp(flag,'column-wise')
    error('A column-wise comparison is not possible with a row
vector input.');
elseif strcmp(default_mode,'column-wise') && strcmp(flag,'row-wise')
    error('A row-wise comparison is not possible with a column
vector input.');
end

% Determine the minimum number of bits needed to represent the
matrices.
tmp = max( max(max(a)), max(max(b)) );
if (tmp > 0)
    sym_len = floor( log(tmp) / log(2) ) + 1;
else
    sym_len = 1;
end

% Check that the user specified 'symbol length' is valid.
if ~isempty(K)
    if max(size(K)) > 1
        error('Word length must be a scalar.');
    elseif (~isfinite(K)) || (floor(K)~=K) || (~isreal(K))
        error('Word length must be a finite, real integer.');
    elseif K < sym_len
        error('The specified word length is too short for the matrix
elements.');
    else
        sym_len = K;
    end
end

a2 = toBinary(a, sym_len);
b2 = toBinary(b, sym_len);

% Two separate flags are needed for the function to operate
efficiently.
```

```
% 'default_mode' specifices if one of the inputs is actually a
vector while
% the other is a matrix, meaning that the vector should be compared
with each
% individual row or column of the matrix.  'flag' (which the user
specifies)
% specifies how the results of this comparison are reported.

if strcmp(default_mode,'overall')
    if strcmp(flag,'column-wise')
        for i = 1:an
            num(1,i) = sum(sum(a2(:,((i-1)*sym_len+1):(i*sym_len))
~= b2(:,((i-1)*sym_len+1):(i*sym_len))));
        end
        rat = num / (am*sym_len);
    elseif strcmp(flag,'row-wise')
        num = sum(a2~=b2,2);
        rat = num / (an*sym_len);
    else
        num = sum(sum(a2~=b2));
        rat = num / (am*an*sym_len);
    end
    if (nargout > 2)
        loc = zeros(am,an);
        for i = 1:an
            loc(:,i) = sum( (a2(:,((i-1)*sym_len+1):(i*sym_len)) ~=
b2(:,((i-1)*sym_len+1):(i*sym_len))), 2);
        end
    end
elseif strcmp(default_mode,'column-wise')
    if (nargout < 3)
        for i = 1:an,
            num(1,i) = sum(sum(a2(:,((i-
1)*sym_len+1):(i*sym_len))~=b2));
        end
    else
        loc = zeros(am,an);
        for i = 1:an,
            loc(:,i) = sum((a2(:,((i-1)*sym_len+1):(i*sym_len)) ~=
b2), 2);
            num(1,i) = sum(loc(:,i));
        end
    end
    if strcmp(flag,'overall')
        num = sum(num);
        rat = num / (am*an*sym_len);
    else
        rat = num / (am*sym_len);
    end
else
    if (nargout < 3)
        for i = 1:am,
            num(i,1) = sum(a2(i,:)~=b2);
        end
    else
        loc = zeros(am,an);
        for i = 1:an
            for j = 1:am
                loc(j,i) = sum( (a2(j,((i-1)*sym_len+1):(i*sym_len))
~= b2(1,((i-1)*sym_len+1):(i*sym_len))), 2);
            end
        end
        num(:,1) = sum(loc,2);
    end
    if strcmp(flag,'overall')
        num = sum(num);
        rat = num / (am*an*sym_len);
    else
        rat = num / (an*sym_len);
    end
end

%%%
function b = toBinary(a, sym_len)
% Convert matrix to binary representation

[am an] = size(a);
b = de2bi(a(:), sym_len);

% block transpose
b = reshape(permute(reshape(b', sym_len, am, an), [2 1 3]), am,
sym_len*an);
```

APPENDIX D

GRAPH OF THEORETICAL BER VS SNR

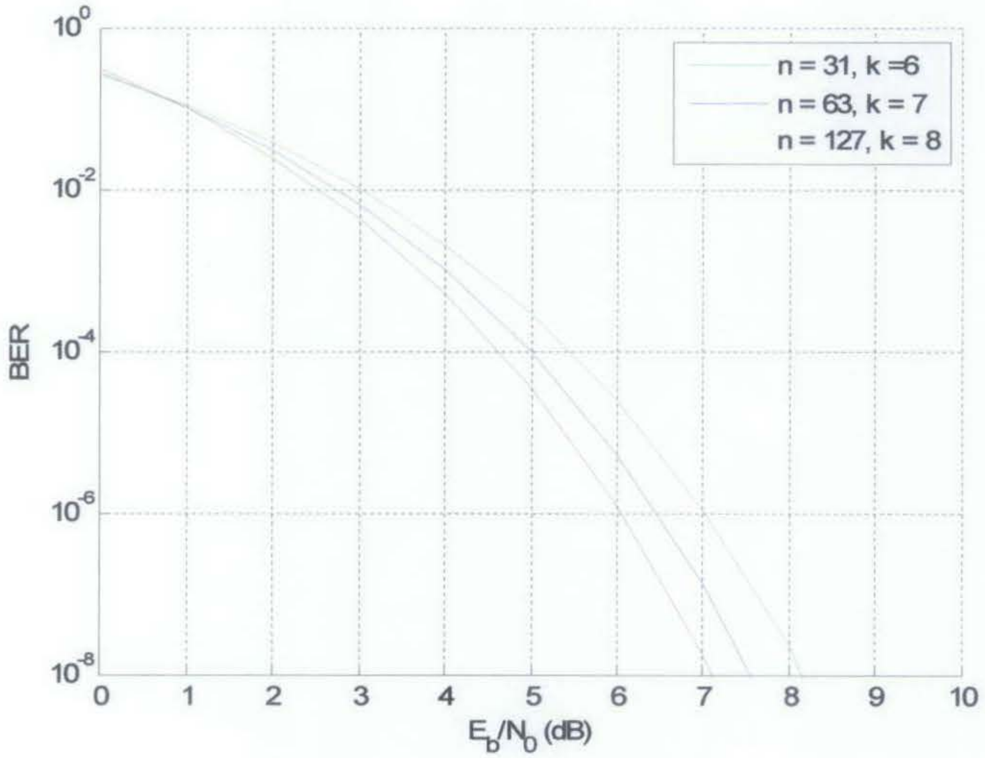


Figure 14: Theoretical BER vs Eb/No