DESIGN AND CONSTRUCTION OF ALGEBRAIC CODES FOR ENHANCING INFORMATION INTEGRITY IN DATA STORAGE SYSTEMS

Thesis

Submitted in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

RAJESH SHETTY K

Register No. EC06F01

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA,

SURATHKAL, MANGALORE- 575 025

May - 2013

DECLARATION

I hereby declare that the Research Thesis entitled **Design and Construction of Algebraic Codes for Enhancing Information Integrity in Data Storage Systems** which is being submitted to the **National Institute of Technology Karnataka**, **Surathkal** in partial fulfillment of the requirements for the award of the Degree of **Doctor of Philosophy in Electronics and Communication Engineering** is a bonafide report of the research work carried out by me. The material contained in this Research Thesis has not been submitted to any University or Institution for the award of any degree.

RAJESH SHETTY K

EC06F01

Department of Electronics and Communication Engineering

Place: NITK, Surathkal

Date: 31- 05-2013

CERTIFICATE

This is to certify that the Research Thesis entitled **"Design and Construction of Algebraic Codes for Enhancing Information Integrity in Data Storage Systems"** submitted by **Rajesh Shetty K (Register Number : EC06F01)** as the record of the research work carried out by him, is accepted as the Research Thesis submission in partial fulfillment of the requirements for the award of degree of Doctor of Philosophy.

Dr. U. Sripati
Research Guide

Dr. Muralidhar Kulkarni
Chairman, DRPC

Dedicated
to
Poojya Shree Bhagawan

ACKNOWLEDGEMENTS

I would like to express my deep sense of gratitude and respect to my research advisor *Dr. U. Sripati* for the guidance, encouragement, unflinching support, advice and help during the period of my research. His undying inspiration and motivation has helped me through the ups and downs of the rough terrain of the research path. This work has taken shape because of his efforts and inputs.

I wish to express my gratitude to *Dr. B. Shankarananda*, my associate advisor, for giving me an opportunity to pursue research work in the Department of Electronics & Communication at NITK, Surathkal.

I am thankful to *Dr. Muralidhar Kulkarni*, Head of the Department of Electronics and Communication, for his support and encouragement.

I would like to thank my ex-HOD *Dr. Sumam David* for her help, advice and constant support during this period.

I would also like to acknowledge my doctoral committee members, *Dr. B. R. Shankar* and *Dr. John D'Souza* for reading and evaluating my research reports as well as for their precious time, suggestions, feedback and interest in this work.

I would also like to express my sincere thanks to *Sri. N. Vinaya Hegde*, President of Nitte Education Trust and Chancellor Nitte University for all his support and encouragement throughout my career.

Special thanks to my friends, *Prashantha Kumar, Ramakrishna* for all the discussions and help. I would also like to express my gratitude to all the staff members (teaching and non teaching) of Electronics and Communication Department at NITK, Suratkal for their help and support.

Most of all, I wish to thank my parents, my wife *Mamatha*, for the love, encouragement, moral and emotional support, patience as well as to my daughter *Sumedha* for bringing joy and happiness in our lives.

Rajesh Shetty K

ABSTRACT

Data storage devices have become ubiquitous in present day information driven society. It is essential that storage devices exhibit very high levels of data integrity. Therefore, data integrity is a fundamental aspect of storage, security and reliability. NAND and NOR Flash memories [Chen, Y. 2008], [Mielke, N et al. 2008], [Gal, E. et al. 2005], [Jiang, A et al. 2010] are widely used for data storage because of their compactness and low power consumption. Data stored in non-volatile memory is usually critical to proper system operation, and corruption of data can lead to system failure. Hence data corruption is a major concern in applications that rely on non-volatile memory for long-term data storage. Many techniques have been employed to improve the reliability of these devices. These techniques can be divided into two categories. In the first approach, improvements are carried out in the fabrication process to reduce the Raw Bit Error Rate (RBER). The second option is to use Error Correction Techniques to improve the RBER level to levels that are deemed acceptable to most users [Sun, F. et al. 2007], [Sun,F et al. 2006], [Chen, Y. and Parhi, K. 2004], [Mielke, N. et al. 2008].

Error Control Code (ECC) techniques (i.e., techniques capable of detecting and correcting errors in processed and stored data by using redundant bits in addition to information bits according to a given coding strategy) [Pless, V. and Huffman, W.C. 1998] have been commonly used at board level for many years to enhance the reliability of memory systems [Bertozzi, D. et al. 2005]. However, as memory chips become denser, they also become more prone to errors, as a consequence of both the reduced cell size and the increased cell count within a single die. Moreover, read and write operations are made more critical by both technology scaling down and higher speed requirements. On the other hand, higher and higher reliability is required for storage systems in a large variety of applications.

Generally high storage density is achieved by reducing the size of the elementary memory cell. However, for non-volatile memories, some physical phenomena makes

an aggressive reduction of the memory cell size difficult [Atwood, G. et al. 1997], [Wang, Z. and Karpovsky, M. 2011]. An alternative solution to reduce the cost per bit and increase the storage density is to adopt the multilevel approach. It consists of placing a multiplicity of charge amount in the floating gate, thus allowing the cell to store more than one bit. However, the multilevel storage requires the consideration of three basic issues:

(i) accuracy of write operation (necessary to place the correct amount of charge of the floating gate).

(ii) precision of the charge sensing (required to discriminate the different threshold voltages).

(iii) stability of charge over an extended time period. Although Multi Level Cell (MLC) memory has higher density than Single Level Cell (SLC) memory, MLC is more vulnerable to errors because small fluctuation of the charge amount in the floating gate and slight variation of gate voltage result in misreading of stored data [Sun, F. et al. 2007], [Sun,F et al. 2006], [Maeda, Y et al. 2009], [Lin, H et al. 2002], [Ankolekar, P. P et al.2010].

ECC is a cost effective method to enhance the integrity of data storage systems. Very stringent values of application BER, which would ordinarily require complex and expensive fabrication techniques as well as expensive materials, can be met very easily by employing ECC. Storage devices characterized by high RBER values can be made to yield application BERs as small as desired by the use of suitable ECC techniques. The fraction of erroneous bits that remain uncorrected after applying ECC constitute the uncorrectable bit error rate (UBER). UBER is a useful reliability metric for data storage devices and is used to specify the data corruption rate in the information given to the user after correction by ECC algorithms. ECC algorithms can also correct errors that may manifest at any later stage during the life of the device. Hence use of ECC techniques has been widely accepted by the semiconductor manufacturing industry to enhance the RBER to levels demanded by applications.

In this thesis, we have made an attempt to synthesize a number of codes for use in data storage systems with error correcting capability exceeding the state of art as

specified in the industry documentation. In the initial part of the thesis, the focus is on the synthesis of codes for enhancing data integrity in flash memories composed of SLCs. While studying the flash memory organization, two memory models, namely Memory model 1 and Memory model 2 are identified and the codes are synthesized separately for these memory models. As compared to the current standard, [Mehnert, A. 2008] where six bits in errors can be corrected over a span of 4096 information bits (one sector), we propose codes that can correct up to nine bits in error per sector. The various generator polynomials are computed. As the performance of the error control code improves with increase in length, we were motivated to consider the combination of two sectors to constitute the information block. For this scenario, we propose codes that can correct up to eighteen bits in error over a span of 8192 bits (two sectors). Further, using Memory model 2, we have synthesized and proposed codes that can correct up to eighteen bits in errors per sector. The performance of these codes is quantified by computing values of the probability of decoding error.

To summarize, the main objective of this work has been to design, construct and synthesize a large group of codes which can be used to enhance the data integrity levels associated with flash memory devices so as to make them useful in a wider class of applications. With a view to make these synthesized codes, readily acceptable to industry, we have strictly adhered to the memory architecture specified in the literature.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AG	Algebraic Geometry
AWGN	additive white Gaussian noise
BCH	Bose-Chaudhury-Hocquenghem
BER	bit error rate
BIOS	basic input output system
ECC	error control coding
EPROM	erasable programmable read only memory
ISI	inter symbol interference
LCM	least common multiple
LDPC	low density parity check
ML	maximum likelihood
MLC	multi level cell
RBER	raw bit error rate
RS	Reed-Solomon
RSER	raw symbol error rate
SEU	single event upset
SLC	single level cell
UBER	uncorrectable bit error rate
USER	uncorrectable symbol error rate

List of Notations

F_q	Galois field $GF(q)$
C	Cyclic code
H	parity check matrix
G	generator matrix
P	sub-matrix
I	identity matrix
n	codeword length
k	dimension of the code
c	Codeword
r	Redundancy
$d_{\min}$	minimum distance of the code
t	error correcting capability
δ	design distance
$g(x)$	generator polynomial
S	Syndrome
W_r	row weight of parity check matrix
W_c	column weight of parity check matrix

Chapter 1

Introduction

1.1 Preliminaries

The revolution in the area of communication and the advent of internet has produced enormous demand for increase in information storage capacity and density. Physical/Media improvements along with sophisticated signal processing and coding techniques have played a critical role in the constant augmentation of storage/communication channel capacities [Costello, D. and Forney, D. 2007]. Every computer memory and data storage system has adopted some type of error detection or error correction code in order to enhance system reliability. The reliability levels required by the storage devices are extremely high. This is because, unlike communication systems, retransmission is generally not possible. It is expected that the user will be able to save data and be able to retrieve it perfectly at any time whenever required.

Storage devices have become an integral part of modern electronic, communication and computing devices. Today, a number of portable hand held devices are required to perform complex mathematical computations. They are also intelligent enough to communicate with each other. The processor performing complex computations has to be supported by storage devices possessing very high level of data integrity. Designers of storage devices have employed many techniques to improve the capacity and reliability of these devices. In recent years, increasingly sophisticated Error Control Coding (ECC) algorithms have been employed to increase the reliability levels of data storage systems.

By definition, a storage device is designed to store and retain information without corruption for long period of time. However, all storage devices have the potential to return information different from what was originally stored. Data integrity is a fundamental aspect of storage security and reliability. NAND Flash memories [Choi, H.et

al. 2010], [Liu, W. et al. 2006], [Sun, F et al. 2006] are widely employed for data storage because of their compactness, low power, low cost, high data throughput and reliability. Scaling and MLC technology have enabled NAND flash memories to replace hard disk drives (HDD) in portable devices and in some computers.

Data stored in the nonvolatile memory is usually critical to proper system operation, and corruption of that data can lead to system failure, hardware damage, and even unsafe operating conditions. Hence data corruption is a major concern in applications that rely on nonvolatile memory for long-term data storage. By implementing proper data protection techniques, both in hardware and software, the chances of data corruption can be greatly reduced. One of the most efficient methods is the use of ECC. In data storage applications/systems, it is important that we are able to save the data and retrieve it correctly without any errors. It is the art of ECC that makes it possible. Permanent and temporal faults are the major sources of errors in modern digital storage systems [Yamada, J. 1987]. Power supply break down, defective open or short circuits, open lines, electron migration etc. can cause permanent faults. Permanent faults lead to hard errors; they therefore affect the system functions for a long period of time. Temporal faults can be transient or intermittent [Massengill, L.W. 1996]. Transient faults occur randomly and externally because of external noise, namely electromagnetic waves and particles such as α particles and neutrons. Intermittent faults occur randomly but internally because of unstable or marginally stable hardware, varying hardware or software state as a function of load, or signal coupling between adjacent signal lines. Some of the intermittent faults may be due to glitches which are unpredictable spike noise pulses occurring and propagated especially in large combinational digital circuits [Calvel, P. et al. 1994]. Temporal faults lead to soft errors and these interrupt system functions for a very short period of time. In today's ultra-high density RAMs, it has been recognized that multiple cosmic ray induced transient errors are a serious problem [Lo, J.C and Fujiwara, E. 2005], [Fujiwara, E. 2006]. Hence error control algorithms with

error detecting/correcting capabilities are required in data storage systems/applications to preserve data integrity.

With the addition of error detection and correction, the risk of system failure due to data corruption in a nonvolatile memory can be minimized. Like HDDs, NAND memories are not intrinsically error-free but rely on ECC to correct raw bits. During process of data readout, the fraction of bits that contain incorrect data (prior to application of ECC) is called the raw bit error rate (RBER) [Mielke, N. et al. 2008]. The residual error rate (pertaining to data that remains uncorrected) after applying ECC is called the uncorrectable bit error rate (UBER) [Mielke, N. et al. 2008]. UBER is a useful reliability metric for mass storage devices such as HDDs and flash memory devices. UBER is used to specify the data-corruption rate after the application of ECC algorithm. The relationship between RBER and UBER for error control algorithms having differing error correcting capabilities is shown in Figure 1.1 [Cooke, J. 2007]. It is observed that, as RBER increases, matching the ECC to the target BER of the application become more important.

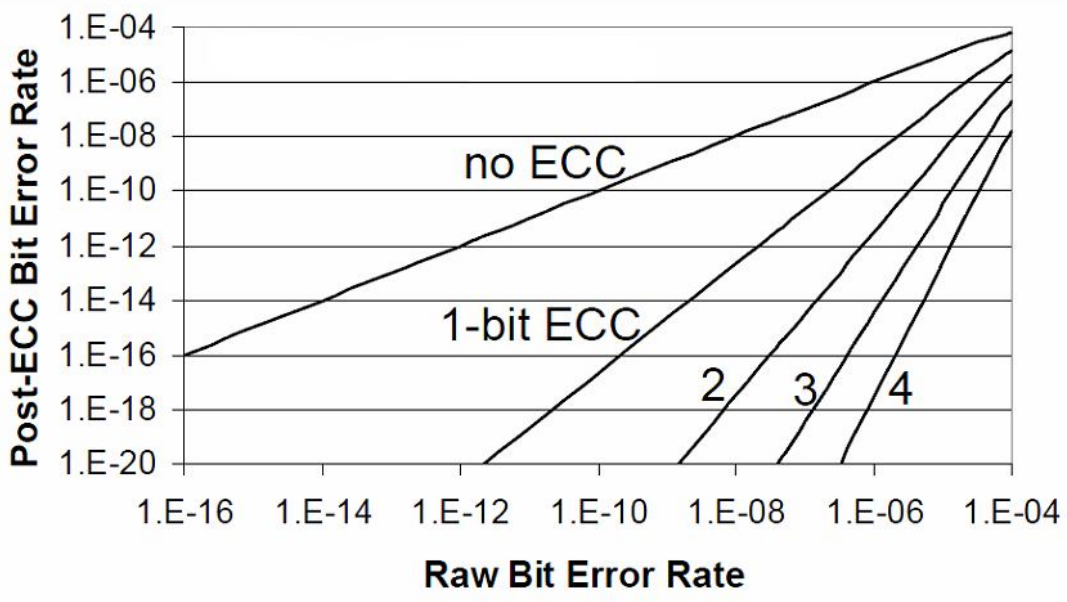


Figure1.1: Raw Bit Error Rate versus Post-ECC Bit Error [Cooke, J. 2007]

A huge number of crucial documents containing privileged and sensitive information are routinely transacted using digital communication systems in the modern era. This information is often conveyed over open channels and stored in digital media. These transactions often have little tolerance for error. Therefore, reliability and accuracy are essential to flash memories. Data can be verified immediately after it is written, but this process will not identify bit errors that may manifest later. To ensure long-term data integrity, modern storage and information processing devices must also use a suitable Error Control Code.

ECC schemes supplement user data with parity bits which store enough extra information for the data to be reconstructed if one or more bits are corrupted. Many ECC algorithms have been employed to correct errors in storage systems [Carrasco, R.A. and Johnston, M. 2008], [Cassuto, Y. et al. 2010], [Chen, Y. and Parhi, K. 2004], [Fujiwara, E. 2006], [Im, S. and Shin, D. 2009], [Lin, H., Chen, T. and Chang, J. 2002], [Micheloni, R et al. 2008]. We have synthesized a number of codes with error correcting capabilities exceeding the state of art in our work. We have developed a suitable architecture for the VLSI implementation of encoder/decoder modules of these codes.

Finite fields are the key tool in the design of powerful linear block codes. The theory of finite fields as applicable to the study of ECC is discussed in [Wicker, S.B. 1995], [Blahut, R.E. 2003] and [Moon, T.K. 2006].

1.2 Motivation for the use of Error Control Codes in Storage Systems

As the recording density increase, a very large number of bits have to be packaged into a very small physical area. Consequently, the physical space available to accommodate a bit has become smaller and smaller over the years with increase in storage density. This results in Inter Symbol Interference (ISI) in the sense that the detection of an information bit (or symbol) is influenced by bits (or symbols) that are present in the recording

medium in the immediate vicinity. This problem becomes acute when the recording density increases. The use of powerful error control algorithms can protect the integrity of user information against errors caused by ageing, wear out due to repeated read and write operations and manufacturing defects. In this work we are proposing several error control codes based on Bose-Chaudhury-Hocquenghem (BCH) and Reed-Solomon (RS) codes for use in semiconductor memories. BCH codes are considered to be good linear error correction codes because of their rich algebraic structure which enables the synthesis of simple and elegant encoding/ decoding algorithms. Further, the scheme of BCH coding and decoding integrated with interleaving can correct both the random errors and the burst errors. As compared to the current standard [Mehnert, A. 2008], where six bits in error can be corrected over a span of 4096 information bits, we propose codes that can correct eight and nine errors (t = 8, t = 9). Memory is organized into blocks, pages and sectors (Refer Table 1 which gives the organization of a 2GB Flash memory device). There are two memory models proposed in semiconductor data storage industry which are widely used in current practice. The smallest unit is a sector. In Memory model 1, each sector has 512 bytes reserved for storing user information and 16 bytes reserved for storing parity check information. In Memory model 2, each sector has 512 bytes reserved for storing user information and 32 bytes reserved for storing parity check (redundant) information.

1.3 Flash Memory Organization

A look at how flash memory is organized reveals some of the challenges involved in managing flash memory. The smallest logical/administrative unit is a sector. Each sector contains a storage area (512 bytes) plus a small overhead area (16 bytes). Sectors are grouped into pages, and blocks include multiple pages (32 pages of 512 bytes or more recently, 64 pages of 2048 bytes). Blocks contain a defined number of sectors, and there are typically 1000 to 8000 blocks per chip. Table 1 shows memory organization for a 2 GB flash device.

Table 1: Memory organization for a 2 GB flash drive

Sector size	512 bytes
Sectors/page	8
Pages/block	64
Page size	4 KB
Block size	256 KB
Blocks/die	4096
Dies/chip	2
Total capacity	2 GB

ECC is performed on a bit/byte level within sectors. Depending on cell structure and quality, this task is becoming more important in flash devices. We have attempted to synthesize suitable BCH and RS codes for use in flash memories for ensuring data integrity. We have developed suitable decoding architecture to perform decoding operations for these codes. The choice of the error control code to be used in a particular application depends upon many factors. These are:

(i) Organization of information and overhead data in the storage devices.

(ii) Types of errors (i.e. whether the error mechanism generates random errors or burst errors.

(iii) Computational complexity permitted by architecture of storage devices. All mathematical operations have to be performed by suitable hardware. An additional arithmetic processor to handle encoding/decoding operations will usually not be feasible.

(iv) Decoding delay (latency) versus computational complexity. Decoding can be considered on a page by page basis. However complexity of computation is high because of large size of data block and corresponding need to do computations in a large field F_{2^8} (or larger). To avoid excessive computational complexity, we have devised codes where one sector or at most two sectors are encoded into one codeword.

A significant difference between the BCH and RS codes is the basic correction unit. The BCH code corrects bits, while the Reed-Solomon code corrects symbols. Therefore, a BCH code corrects *t* bit errors, while the Reed-Solomon code over F_{2^m} corrects *t* symbols of *s* bit errors, which can be a variable number of bits: from *t* bits if errors occur in different symbols up to *st* bits in case errors occur consecutively. Reed-Solomon codes generally have a greater correction capability for burst type errors while BCH codes possess simpler decoding architectures and can correct random errors efficiently. As the trend in semiconductor memory design continues towards higher chip density and large storage capacity, ECCs are becoming most cost effective means of maintaining a high level of system reliability [Bertozzi, D. et al. 2005]. A memory system can be made fault tolerant with the application of an error-correcting code; i.e., the mean time between "failures" of a properly designed memory system can be significantly increased with ECC. In this context, a system "fails" only when the errors exceed the error correcting capability of the code. Also, in order to optimize data integrity, the ECC should have the capability of correcting error patterns that are most likely to occur.

A bursty channels is defined as a channel over which errors tend to occur in bunches or "bursts" as opposed to the random patterns. Bursty channels usually contain some error causing agents in the physical medium whose effective time constant exceeds the symbol transmission rate of the channel. A random error correcting code can correct up to *t* symbol errors per code word, regardless of the placement of these errors. Both random and burst errors are encountered in data storage systems. Burst errors are frequently encountered in optical storage systems (e.g., Compact Disc (CD), DVDs etc). Without error correcting codes, digital audio/video/data storage would not be technically feasible. The recording channel in a CD play back system consists of a transmitting laser and the disc to be recorded. The play back channel consists of the recorded disc and a photo-detector. Assuming that the player is working properly, the primary contributor to errors on this channel is contamination of the surface of the disc. As the surface contamination

affects an area that is usually quite large compared to the surface used to record a single bit, channel errors occur in bursts when the disc is played.

The various computer applications for error control can be grouped into four categories: memory (random access memory and read-only memory), disk storage, tape storage, and inter-processor communication. Each type of communication has its unique characteristic and requires the use of certain types of codes. The earliest memory storage units were mechanical relays. The contents of these relays could be verified through visual inspection. Simple parity check codes provided error detection in these systems. The relays were replaced by ferrite cores in which the magnetic field could be induced and its orientation read. The core memory had to be refreshed periodically and their contents verified to see if any errors had crept in since the last refresh cycle.

Semiconductor memory is currently the predominant choice for random access and read-only memory applications. Read/write errors in semiconductor memory are generally labeled as either “hard” or “soft” errors. Hard errors are device failures that are permanent, while soft errors are transient, and are sometime called “single event upsets” (SEUs). SEUs are frequently caused by radiation; atomic particles leave an ion trail, or “tunnel” as they pass through a device substrate. SEUs can also be caused by electronic transients induced by lighting, adjacent electrical machinery, and the ionic scintillation generated by thermonuclear weapons. Some of these error sources can also cause hard failures. Magnetic disks have been traditionally used for mass storage applications. As with the Compact Disc, these storage devices involve moving media and/or record/read heads. Though not as prone to radiation-induced SEUs as semiconductor memory, disks do suffer from burst errors caused by surface contamination and material defects.

Magnetic tape suffers from error-causing mechanisms similar to those encountered in magnetic and optical disk systems. In addition, the flexibility of the magnetic tape can create its own unique set of problems. In most digital systems the data on the tape is

organized in parallel tracks that run the length of the tape, hence contaminants tend to corrupt one track or a small number of adjacent tracks during the reading process.

Semiconductor memories can be divided into two major categories: RAM and ROM. The category of non-volatile memories [Ricco, B. et al. 1998] includes all the memory devices whose content can be changed electrically but is held when the power supply is switched off. The history of non-volatile memories began in the seventies, with the introduction of the first EPROM (Erasable Programmable Read Only Memory). Since then, non-volatile memories have always been considered one of the most important families of semiconductor devices. With the introduction of non-volatile Flash memories into portable products like mobile phones, palmtop, digital cameras and so on, the market of these memories has seen a staggering increase. Flash memories are non-volatile memories characterized by the fact that the erase operation (writing of the logic "1") must occur simultaneously on a set of cells, called sector or block; the program operation instead (writing of the logic "0") acts on the single cell. These devices have become the most widely used non-volatile electronic memories. It would not be an exaggeration to state that Flash memories are a milestone in the development of the data storage technology. The applications of flash memories have expanded widely in recent years, and flash memories have become the dominating member in the family of non-volatile memories. Compared to magnetic recording and optical recording, flash memories are more suitable for many mobile-embedded and mass-storage applications. The reasons include their high speed, physical robustness, and easy integration with circuits. The representation of data plays a key role in storage systems. Like magnetic recording and optical recording, flash memories have their own distinct properties, including block erasure, iterative cell programming, etc. These distinct properties introduce very interesting information representation and coding problems that address many aspects of a successful storage system, such as efficient data modification, error correction, etc.
Actually, the name "flash" is associated with the speed at which large data blocks can be erased which is similar to that of the flash of a camera. In contrast, old-style EEPROM

allows only the simultaneous erasure of bytes. For these reasons, flash memories are becoming dominant as secondary memory devices in digital cameras, digital audio players, mobile phones, and PC basic input/output system (BIOS) chips. A flash memory is an array of cells that consist of floating gate transistors. Information is stored as an electric charge in each cell.

The increase of the storage density and the reduction of the cost per bit of flash memories were traditionally achieved by the aggressive scaling of the memory cell transistor until the MLC technology was developed and implemented in 1997 [Atwood, G. et al. 1997], [Wang, Z. and Karpovsky, M. 2011]. In MLC devices [Grossi, M. et al. 2003], a cell can assume several possible voltage levels.

The amount of charge in a cell determines its threshold voltage, which can be measured. The operation of injecting charge into a cell is called *writing* (or *programming*), removing charge is called *erasing*, and measuring the charge level is called *reading*. If we use two discrete charge levels to store data, the cell is called SLC and can store one bit. If we use $Q > 2$ discrete charge levels to store data, the cell is called MLC and can store $\log_2 Q$ bits. A prominent property of flash memories is *block erasure*. In a flash memory, cells are organized as blocks, each containing about 10^5 cells. While it is relatively easy to inject charge into a cell, to remove charge from any cell, the whole block containing it must be erased to the ground level (and then reprogrammed). This is called block erasure. The block erasure operation not only significantly reduces speed, but also reduces the lifetime of the flash memory [Atwood, G et.al 1997]. This is because a block can only endure about $10^4 - 10^6$ erasures, after which the block may break down. Since the breaking down of a single block can make the whole memory stop working, it is important to balance the erasures performed to different blocks. This is called *wear leveling*. A commonly used wear leveling technique is to balance erasures by moving data among the blocks, especially when the data are revised [Gal, E et al. 2005].

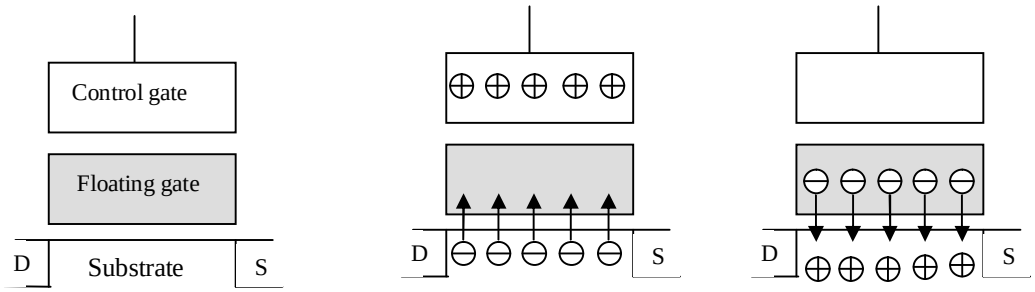
For the Flash memories currently in the market, the architectures used are NOR and NAND flash memories [Ricco, B. et al. 1998], [Bez, R. et al. 2003], [Gal, E. et al. 2005], [Jiang, A. et al. 2010]. A NOR flash memory allows random access to its cells. NAND flash partitions every block into multiple sections called pages, and a page is the unit of a read or write operation. Compared to NOR flash, NAND flash may be much more restrictive on how its pages can be programmed, such as allowing a page to be programmed only a few times before erasure [Gal, E. et al. 2005]. However, NAND flash enjoys the advantage of higher cell density. Between them, NAND flash is currently used much more often due to its higher data density.

However, there remain many technical challenges in flash memories. Adding charge to a single cell is easy, but removing charge from a cell requires erasing the entire block containing that cell and reprogramming all cells in that block. These block erasures are time-consuming and can also cause physical degradation and shorten memory life. Therefore, it is important to reduce the frequency of block-erasures [Gal, E. and Toledo, S. 2005]. Coding techniques have been introduced to accomplish this. A variety of coding schemes for flash memories were introduced such as floating codes [Jiang, A. et al. 2010], [Mahadavifar, H. et al. 2009], buffer codes [Bohossian, V. et al. 2007], [Jiang, A. et al. 2009a], trajectory codes [Jiang, A. et al. 2009b], multidimensional flash codes [Yaakobi, E. et al. 2008], and rank modulation codes [Jiang, A. et al. 2009], [Wang, Z. and Bruck, J. 2010].

Error control codes are widely used in almost all digital communications. This is due to the higher performance that the market demands for achieving reliable communications over noisy channels. BCH codes, a very important family of block codes that can be decoded using algebraic techniques with affordable complexity, have been in wide use for decades, especially in storage channels in various forms either as Hamming codes or as Reed-Solomon (RS) codes. BCH codes are in wide use in concatenated coding techniques, concatenated either with Convolutional codes or with other block codes such

as Low-density parity check (LDPC) codes [Ryan,W. E. and Lin, S]. Second generation Digital Video Broadcast (DVB) standards are adopting BCH codes as part of their concatenated coded strategy. Binary BCH codes have some advantages over RS codes, especially if the noise is random. The read channel in a MLC based flash memory exhibits a random noise channel and BCH is a favorite code for error correction in flash memory products.

1.4 Flash Memory Structure and Error Characteristics [Maeda, Y. et al. 2009]



(a) Structure (b) Program (c) Erase

Figure 1.2: Flash Memory Cell

Figure 1.2 shows flash cell structure having control gate, floating gate, source and drain. The floating gate is insulated from the substrate. The cell is programmed by applying a high voltage to the control gate, which moves electrons from the substrate into the floating gate. The cell is erased by applying a high voltage to the substrate, which removes electrons from the floating gate. In order to read out the data from the cell, the charge level of the floating gate is identified by applying specific voltages to the control gate. If the floating gate has few electrons, the drain-source current flows with a low control gate voltage, while if the floating gate has many electrons, the drain-source current flows only when the control gate voltage is sufficiently high.

Conventional SLC can hold two distinct levels of charge, and hence it can store one bit of data in each memory cell, while a MLC can reliably hold $Q = 2^b$ levels of charge, and

hence it can store *b* bits of data, where *b* is typically 2 or 3 [Rossi, D. et al. 2002], [Rossi, D. et al. 2003], [Gregori, S. et al. 2003], [Sun, F et al. 2007], [Maeda, Y. et al. 2009]. Although MLC memory has higher density than SLC memory, MLC is more vulnerable to errors because small fluctuation of the charge amount in the floating gate and slight variation of gate voltage can result in misreading of stored data. Multilevel flash memory cells have found application in efforts to increase density of bits per unit area in recent years [Lin, H. et al. 2002].

1.5 Scope of the research work presented in the thesis

Our aim is to synthesize high rate (hence efficient) algebraic codes for enhancing data integrity in storage systems. In the initial part of our work, we have concentrated on the synthesis of codes for enhancing data integrity in Flash memories composed of SLCs. In the second part, we have synthesized the codes applicable in memories composed of MLCs. Here we have considered cells with four levels, eight levels and sixteen levels. Channel Matrices, also called P-matrices which quantify the probability of various forms of error in MLCs have been quantified. These give insight into the type of errors that can occur in MLCs. We have also investigated the role of interleavers to improve data integrity during occurrence of burst errors in storage devices. Mainly, block interleavers and convolutional interleavers are considered for the analysis. Further, we have also analyzed and adapted the decoder architecture for the codes synthesized by us. This decoder makes use of inversion free Berlekamp - Massey algorithm.

1.6 Organization of the thesis

The contents of this thesis are organized into chapters as follows: Chapter 2 provides a brief overview of the necessary mathematical background. We have provided a brief description of finite fields because traditional BCH codes are constructed over finite fields. The BCH bound has been invoked in the design of BCH codes. Synthesis of BCH codes and RS codes for different memory models as applicable to single level cells are discussed in Chapter 3. In this chapter, we have also obtained the performance plots of

the codes that are synthesized. In Chapter 4, we have discussed the modeling of multilevel cell as a channel and synthesized codes for different memory models as applicable to multilevel cells. A suitable decoder architecture based on the inversion less Berlekemp-Massey algorithm has been adapted to study the decoding in Chapter 5. The use of interleavers and its ability to enhance data integrity is also discussed in this chapter. Further the codes are also synthesized using LDPC and the reason for not adapting these codes in flash memory is highlighted. We conclude the thesis in Chapter 6 by summarizing the obtained results, and giving some directions for further research work in this area.

Chapter 2

Mathematical background

2.1 Cyclic Codes

Cyclic codes have been to be a constant focus of interest for mathematicians and engineers for the past five decades. Cyclic codes [Lin, S. and Costello, D. 2004], [Moon, T.K. 2006] are important practical error control codes for a variety of reasons. Cyclic codes are a class of error correcting codes that can be efficiently encoded and decoded using simple shift registers and combinatorial elements, on the basis of their representation using polynomials. Within the family of cyclic codes there are certain special families of codes that are extremely powerful. These include the Golay, BCH and the RS codes.

Definition 2.1.1 – *Linear Cyclic Block Codes* [Wicker, S.B. 1995]

An (*n, k*) linear block code **C** is said to be cyclic if for every code word $c = (c_0, c_1, ...c_{n-1}) \in$ **C**, there is also a code word $c' = (c_{n-1}, c_0, c_1, ..., c_{n-2}) \in$ **C.** The code word c' is a right cyclic shift of the code word *c*. Since *c* has been arbitrarily selected from among the code words in **C**, it follows that all *n* of the distinct cyclic shifts of *c* must also be code words in **C**.

The key to the underlying structure of cyclic codes lies in the association of a code polynomial $c(x) = c_0 + c_1 x + c_2 x^2 + ... + c_{n-1} x^{n-1}$ with every code word **c** = $(c_0, c_1, c_2, ..., c_{n-2}, c_{n-1}) \in$ **C**. If **C** is a *q*-ary (*n, k*) code, then the collection of code words in **C** forms a vector space of dimension *k* within the space of all *n*-tuples over F_q.

$$
\begin{aligned}
c &= (c_0, c_1, c_2, \ldots\ldots\ldots\ldots c_{n-1}) \leftrightarrow c(x) \\
c^{(1)} &= (c_{n-1}, c_0, c_1, \ldots\ldots\ldots\ldots\ldots c_{n-2}) \leftrightarrow c^1(x) \\
c^{(2)} &= (c_{n-2}, c_{n-1}, c_0, \ldots\ldots\ldots\ldots\ldots c_{n-3}) \leftrightarrow c^2(x) \\
&\vdots \\
c^{(k)} &= (c_{n-k}, c_{n-k+1}, c_{n-k+2}, \ldots\ldots\ldots\ldots\ldots\ldots c_{n-k-1}) \leftrightarrow c^k(x)
\end{aligned} \tag{2.1}
$$

$$
\begin{aligned}
c(x) &= c_0 + c_1 x + c_2 x^2 + \ldots\ldots\ldots\ldots\ldots + c_{n-1} x^{n-1} \\
xc(x) &= c_0 x + c_1 x^2 + c_2 x^3 + \ldots\ldots\ldots\ldots + c_{n-1} x^n \\
x^2 c(x) &= c_0 x + c_1 x^2 + c_2 x^3 + \ldots\ldots\ldots\ldots + c_{n-1} x^{n+1}
\end{aligned} \tag{2.2}
$$

Hence, it may be concluded that

$$
\begin{aligned}
xc(x) \bmod (x^n - 1) &= c^1(x) \\
x^2 c(x) \bmod (x^n - 1) &= c^2(x) \\
&\vdots \\
x^k c(x) \bmod (x^n - 1) &= c^k(x)
\end{aligned} \tag{2.3}
$$

Set: A set is an arbitrary collection of objects, or elements, without any predefined operation between set elements. A set may be finite, countably infinite, or uncountably infinite. The primary characteristic of a set is its cardinality, which is defined as the number of objects/elements contained in the set.

Group: A group is a set of objects G on which a binary operation '$\bullet$' has been defined. The binary operation takes any two elements in G and generates as its result an element that is also in G.

Definition 2.1.2 - *Group:*

A set G on which a binary operation '$\bullet$' is defined is called a group if the following conditions are satisfied.

(i) The binary operation '$\bullet$' is associative.

(ii) G contains an element e such that for any a in G, $a \bullet e = e \bullet a = a$. This element e is called an identity element of G.

(iii) For any element a in G, there exists another element a' in G such that $a \bullet a' = a' \bullet a = e$. The element a is called an inverse of a.

(iv) A group G is said to be commutative (or Abelian) if its binary operation '•' satisfies a '•' b=b '•' a.

Definition 2.1.3 – *Irreducible polynomial*

A polynomial $f(x)$ is said to be irreducible in the field F_q if it cannot be factored into a product of polynomials with coefficients in F_q. It can be easily verified that the polynomials $x^2 + x + 1$ and $x^3 + x + 1$ are irreducible over $F_2[x]$.

Definition 2.1.4 – *Primitive polynomial*

An irreducible polynomial $p(x)$ is said to be primitive if the smallest positive integer 'n' for which $p(x) \mid x^{n-1}$ is $n = p^m - 1$.

i.e., $p(x) \mid x^{p^m - 1} - 1$ and $p(x) \nmid x^n - 1$ for $n < p^m - 1$.

A primitive polynomial for F_{q^m} is the minimal polynomial of some primitive element of F_{q^m}. It is the polynomial of smallest non-zero degree with coefficients from F_q having a certain primitive element of F_{q^m} as a root.

$x^2 + x + 1$ and $x^3 + x + 1$ are examples of polynomials that are irreducible as well as primitive. They are used to generate the finite fields F_{2^2} and F_{2^3} respectively.

Definition 2.1.5 - *Minimal polynomial*

Let α be a primitive element in the field F_{q^m}. The minimal polynomial of α with respect to F_q is the smallest degree nonzero polynomial $p(x)$ in $F_q[x]$ such that $p(\alpha) = 0$.

Definition 2.1.6– *Conjugates of Field Elements*

Let β be an element of F_{q^m}. Then the conjugates of β with respect to F_q are $\beta^q, \beta^{q^2}, \beta^{q^3}, \ldots\ldots\ldots$. Since the field is finite, this process cannot keep yielding new elements indefinitely.

Let us consider an example to illustrate this. The elements of F_{2^3} are $F_{2^3} = \{0,1,\alpha,\alpha^2,\alpha^3,\alpha^4,\alpha^5,\alpha^6\}$. Here $q = 2$ and $m = 3$. The conjugates of α are:

$$\alpha^{2^1} = \alpha^2,$$
$$\alpha^{2^2} = \alpha^4$$
$$\alpha^{2^3} = \alpha^8 = \alpha$$

Therefore the distinct conjugates of α in F_{2^3} are α^2 and α^4**.** Conjugates of α with respect to F_q form a set which is called the conjugacy class of α with respect to F_q. The conjugacy class of α is $\{\alpha, \alpha^{2^1} = \alpha^2, \alpha^{2^2} = \alpha^4\} = \{\alpha, \alpha^2, \alpha^4\}$.

Determining the conjugates of α^3,

$$(\alpha^3)^{2^0} = \alpha^3$$
$$(\alpha^3)^{2^1} = \alpha^6$$
$$(\alpha^3)^{2^2} = (\alpha^3)^4 = \alpha^{12} = \alpha^5$$
$$(\alpha^3)^{2^3} = (\alpha^3)^8 = \alpha^{24} = \alpha^3$$

Therefore the conjugacy class of $\alpha^3 \in F_{2^3}$ is $\{\alpha^3, \alpha^6, \alpha^5\}$. Hence we write the conjugacy classes of F_{2^3} are $\{0\}, \{1\}, \{\alpha, \alpha^2, \alpha^4\}, \{\alpha^3, \alpha^6, \alpha^5\}$.

Theorem 2.1.1: Let α be an element of F_{q^m}. Let $p(x)$ be the minimal polynomial of α with respect to F_q. The roots of $p(x)$ are exactly the conjugates of α with respect to F_q.

Let $\alpha \in F_{2^3}$. The conjugacy class of α is $\{\alpha, \alpha^2, \alpha^4\}$. So the minimal polynomial of α is $p(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^4) = x^3 + x + 1$.

Example 2.1.1: The minimal polynomials of the elements in F_{2^3} with respect to F_2.

Exponential Representation	Polynomial Representation
α^0	1
α^1	α
α^2	α^2
α^3	$\alpha+1$
α^4	$\alpha^2+\alpha$
α^5	$\alpha^2+\alpha+1$
α^6	α^2+1
0	0

The eight elements are arranged in conjugacy classes and their minimal polynomials are computed as follows:

Conjugacy Class	Minimal Polynomial
$\{0\}$	$M_*(x)=(x-0)=x$
$\{\alpha^0=1\}$	$M_0(x)=(x-1)=x+1$
$\{\alpha,\alpha^2,\alpha^4\}$	$M_1(x)=(x-\alpha)(x-\alpha^2)(x-\alpha^4)=x^3+x+1$
$\{\alpha^3,\alpha^6,\alpha^5\}$	$M_3(x)=(x-\alpha^3)(x-\alpha^6)(x-\alpha^5)=x^3+x^2+1$

Definition 2.1.7– *Cyclotomic Cosets* [Wicker,S.B. 1995]

The cyclotomic cosets *modulo n* with respect to F_q are partitioning of the integers $\{0,1,.....,n-1\}$ into the sets of the form $\{a, aq, aq^2, aq^3, ..., aq^{d-1}\}$. The cyclotomic cosets *modulo n* with respect to F_q thus contain the exponents of the *n* distinct powers of a primitive n^{th} root of unity with respect to F_q. Referring to the example 2.1.1, the conjugacy class and the cyclotomic cosets are shown below.

Conjugacy Class	Cyclotomic Cosets
$\{\alpha^0=1\}$	$\{0\}$
$\{\alpha,\alpha^2,\alpha^4\}$	$\{1,2,4\}$
$\{\alpha^3,\alpha^6,\alpha^5\}$	$\{3,6,5\}$

The most commonly used cyclic error correcting codes are the BCH and Reed-Solomon codes. Hence these codes may be specified by a generator polynomial. A BCH code over F_q of length *n* capable of correcting at least *t* errors is synthesized as follows [Wicker, S.B. 1995]:

(i) Determine the smallest *m* such that F_{q^m} has a primitive *n*th root of unity β.

(ii) Select a non-negative integer *b*. Frequently, *b* = 1.

(iii) List the *2t* consecutive powers of β.

$$\beta^b, \beta^{b+1}, ..., \beta^{b+2t-1}$$

The minimal polynomial with respect to F_q of each of these powers of β are determined.

(iv) The generator polynomial *g(x)* is the least common multiple (LCM) of these minimal polynomials. The code is a *(n, n- deg(g(x))* cyclic code.

2.2 Bounds on Codes [Moon, T.K., 2006]

Let **C** be an *(n ,k)* block code with minimum distance $d_{\min}$ over a field with *q* elements with redundancy $r = n - k$. There are relationships that must be satisfied among the code length *n*, the dimension *k*, the minimum distance $d_{\min}$, and the field size *q*.

Theorem 2.2.1- (**The Singleton bound).** *The minimum distance for an (n, k) linear code is bounded by*

$$d_{\min} \leq n - k + 1 \tag{2.4}$$

Proof

An *(n, k)* linear code has a parity check matrix **H** with $n - k$ linearly independent rows. Since the row rank of a matrix is equal to its column rank, rank (**H**) = $n - k$. Any collection of $n - k + 1$ columns must therefore be linearly dependent. Thus, the minimum distance cannot be larger than $n - k + 1$.

A code for which $d_{\min} = n - k + 1$ is called a maximum distance separable (MDS) code.

Theorem 2.2.2 - (The Hamming Bound). *A t-error correcting q-ary code* **C** *must have redundancy r satisfying*

$$r \geq \log_q V_q(n,t) \tag{2.5}$$

Proof

Each of *M* spheres in **C** has radius *t*. The spheres do not overlap, or else it would not be possible to decode *t* errors. The total number of points enclosed by the spheres must be $\leq q^n$. We must have

$$MV_q(n,t) \leq q^n \tag{2.6}$$

where $V_q(n,t)$ is the number of points in a Hamming sphere of radius $t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor$

so

$$\frac{q^n}{M} \geq V_q(n,t) \tag{2.7}$$

from which the result follows by taking $\log_q$ of both sides. A code satisfying the Hamming bound with equality is said to be a perfect code. In this section, we are seeking theoretical limits without regard to the feasibility of a code for any particular use.

Definition 2.2.1

Let $A_q(n, d_{\min})$ be the maximum number of codewords in any code over F_q of length *n* with minimum distance $d_{\min}$. For a linear code the dimension of the code is

$$k = \log_q A_q(n, d_{\min}).$$

Definition 2.2.2

For a code with length n and minimum distance $d_{\min}$, let $\delta = d_{\min}/n$ be the relative distance of the code. For a code with relative distance δ, the distance is $d_{\min} \approx \delta n = \delta n + O(1)$

Definition 2.2.3

Let

$$\alpha_q(\delta) = \underbrace{\limsup}_{n\to\infty} \frac{1}{n} \log_q A_q(n, \lfloor \delta n \rfloor \tag{2.8}$$

For a linear code, $\log_q A_q(n, d_{\min})$ is the dimension of the code and $\frac{1}{n}\log_q A_q(n, d_{\min})$ is the code rate, so $\alpha_q(\delta)$ is the maximum possible code rate that an arbitrary long code can have while maintaining a relative distance δ. We call this the asymptotic rate.

The functions $A_q(n, d_{\min})$ and $\alpha_q(\delta)$ are not known in general, but the upper and lower bounds on these functions can be established. For example, the Singleton bound can be expressed in terms if these functions as

$$A_q(n, d_{\min}) \le q^{n-d_{\min}+1} \tag{2.9}$$

and, asymptotically,

$$\alpha_q(\delta) \le 1-\delta \tag{2.10}$$

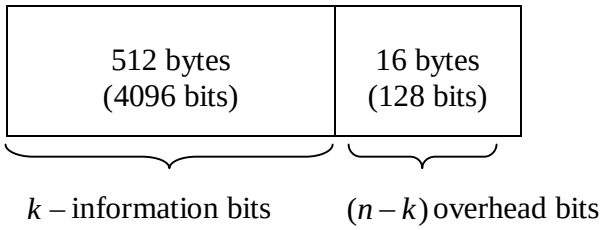
Many of the bounds presented here are expressed in terms of $\alpha_q(\delta)$. A lower bound is the Gilbert-Varshamov bound. As upper bounds on $\alpha_q(\delta)$, we have the Hamming and the Singleton bounds, the Plotkin bound, the Elias bound, and the McEliece-Rodemich-Rumsey-Welch bound.

Chapter 3

Application of Error Correcting Codes to Single Level Cells

3.1 Introduction

In Chapter 1, we had discussed memory organization where we had specified the manner in which sectors, pages and blocks are integrated to give rise to a Flash memory device. It was mentioned that two memory models have been extensively discussed in literature [Mehnert, A. 2008]. Accordingly, let us consider the first model (Memory model 1) with 512 bytes of data and 16 bytes of overhead.



The design of BCH codes has been discussed in [Chapter 2, section 2.1]. As discussed there, to design a BCH code of length n, we choose the smallest m such that $n|q^m-1$, where q is a power of prime. In practical applications usually $q = 2$. Hence we choose the smallest m such that $n|\ 2^m-1$. BCH codes, take advantage of a useful result that ensures a minimum "design distance δ", given a particular constraint on the generator polynomial.

3.2 Synthesis of BCH Codes

3.2.1 Memory model 1

The BCH Bound:

To recapitulate, let **C** be a q-ary (n, k) cyclic code with a generator polynomial $g(x)$. Let m be a multiplicative order of q modulo n (F_{q^m} is thus the smallest extension field of F_q

that contains a primitive n^{th} root of unity). Let α be a primitive n^{th} root of unity. Select g*(x)* to be a minimal-degree polynomial in F_q such that $g(\alpha^b) = g(\alpha^{b+1}) = g(\alpha^{b+2}) = \ldots\ldots\ldots\ldots = g(\alpha^{b+\delta-2}) = 0$ for some integers $b \geq 0$ and $\delta \geq 1$. *g(x)* thus has $(\delta - 1)$ consecutive powers of α as zeros. If these requirements are taken care of, the code **C** defined by *g(x)* has minimum distance $d_{\min} \geq \delta$. The parameter δ is the *design distance* of the BCH code defined by the generator polynomial *g(x)*.

A (*n, k*) block code is characterized by the value of *n, k*. The process used for synthesis of codes is as follows:

(i) Based on the value of (n, k), $n-k$ represents the number of parity symbols ($n > k$). Choose the smallest value of *m* such that $n \mid 2^m - 1$ (i.e. if k = 4096 bits $= 2^{12}$, then shortest primitive length BCH code with $n > 4096$ is $n = 2^{13} - 1 = 8191$ bits).

(ii) Select the appropriate extension field of F_2 in which the primitive n^{th} root of unity can be found (in this case $F_{2^{13}}$).

(iii) Determine the conjugacy classes and associated minimal polynomials.

(iv) Synthesize generator polynomial by following requirements of BCH bound.

Let the natural length $n = 2^{13} - 1 = 8191$. The elements of $F_{2^{13}}$ are

$F_{2^{13}} = \{0, 1, \alpha, \alpha^2, \alpha^3, \ldots\ldots\ldots\ldots, \alpha^{8190}\}$. We have used the primitive polynomial $p(x) = 1 + x + x^3 + x^4 + x^{13}$, to generate the representation of the elements of $F_{2^{13}}$.The conjugacy classes are listed below.

1. $\{\alpha, \alpha^2, \alpha^4, \alpha^8, \alpha^{16}, \alpha^{32}, \alpha^{64}, \alpha^{128}, \alpha^{256}, \alpha^{512}, \alpha^{1024}, \alpha^{2048}, \alpha^{4096}\}$
2. $\{\alpha^3, \alpha^6, \alpha^{12}, \alpha^{24}, \alpha^{48}, \alpha^{96}, \alpha^{192}, \alpha^{384}, \alpha^{768}, \alpha^{1536}, \alpha^{3072}, \alpha^{6144}, \alpha^{4097}\}$
3. $\{\alpha^5, \alpha^{10}, \alpha^{20}, \alpha^{40}, \alpha^{80}, \alpha^{160}, \alpha^{320}, \alpha^{640}, \alpha^{1280}, \alpha^{2560}, \alpha^{5120}, \alpha^{2049}, \alpha^{4098}\}$
4. $\{\alpha^7, \alpha^{14}, \alpha^{28}, \alpha^{56}, \alpha^{112}, \alpha^{224}, \alpha^{448}, \alpha^{896}, \alpha^{1792}, \alpha^{3584}, \alpha^{7168}, \alpha^{6145}, \alpha^{4099}\}$

5. $\{\ \alpha^{9},\alpha^{18},\alpha^{36},\alpha^{72},\alpha^{144},\alpha^{288},\alpha^{576},\alpha^{1152},\alpha^{2304},\alpha^{4608},\alpha^{1025},\alpha^{2050},\alpha^{4100}\}$ (3.1)

6. $\{\ \alpha^{11},\alpha^{22},\alpha^{44},\alpha^{88},\alpha^{176.},\alpha^{352},\alpha^{704},\alpha^{1408},\alpha^{2816},\alpha^{5632},\alpha^{3073},\alpha^{6146},\alpha^{4101}\}$

7. $\{\ \alpha^{13},\alpha^{26},\alpha^{52},\alpha^{104},\alpha^{208},\alpha^{416},\alpha^{832},\alpha^{1664},\alpha^{3328},\alpha^{6656},\alpha^{5121},\alpha^{2051},\alpha^{4102}\}$

8. $\{\ \alpha^{15},\alpha^{30},\alpha^{60},\alpha^{120},\alpha^{240},\alpha^{480},\alpha^{960},\alpha^{1920},\alpha^{3840},\alpha^{7680},\alpha^{7169},\alpha^{6147},\alpha^{4103}\}$

9. $\{\ \alpha^{17},\alpha^{34},\alpha^{68},\alpha^{136},\alpha^{272},\alpha^{544},\alpha^{1088},\alpha^{2176},\alpha^{4352},\alpha^{513},\alpha^{1026},\alpha^{2052},\alpha^{4104}\}$

10. $\{\ \alpha^{19},\alpha^{38},\alpha^{76},\alpha^{152},\alpha^{304},\alpha^{608},\alpha^{1216},\alpha^{2432},\alpha^{4864},\alpha^{1537},\alpha^{3074},\alpha^{6148},\alpha^{4105}\}$

Corresponding to each conjugacy class is a minimal polynomial. This polynomial has the elements of the conjugacy class as its roots. For example, the minimal polynomial of conjugacy class 1 is specified as,

$$\begin{aligned}&(x-\alpha)(x-\alpha^{2})(x-\alpha^{4})(x-\alpha^{8})(x-\alpha^{16})(x-\alpha^{32})(x-\alpha^{64})(x-\alpha^{128})(x-\alpha^{256})\\&(x-\alpha^{512})(x-\alpha^{1024})(x-\alpha^{2048})(x-\alpha^{4096})\\&=1+x+x^{3}+x^{4}+x^{13}\\&=M_{1}(x)\end{aligned} \tag{3.2}$$

Similarly, the minimal polynomial of conjugacy class 2 is specified as

$$\begin{aligned}&(x-\alpha^{3})(x-\alpha^{6})(x-\alpha^{12})(x-\alpha^{24})(x-\alpha^{48})(x-\alpha^{96})(x-\alpha^{192})(x-\alpha^{384})\\&(x-\alpha^{768})(x-\alpha^{1536})(x-\alpha^{3072})(x-\alpha^{6144})(x-\alpha^{4097})\\&=1+x+x^{4}+x^{5}+x^{7}+x^{9}+x^{10}+x^{13}\\&=M_{3}(x)\end{aligned} \tag{3.3}$$

The minimal polynomial of the conjugacy classes listed in (3.1) is enumerated in (3.4).

$$M_{1}(x)=1+x+x^{3}+x^{4}+x^{13}$$

$$M_{3}(x)=1+x+x^{4}+x^{5}+x^{7}+x^{9}+x^{10}+x^{13}$$

$$M_{5}(x)=1+x+x^{4}+x^{7}+x^{8}+x^{11}+x^{13}$$

$$M_{7}(x)=1+x+x^{2}+x^{3}+x^{6}+x^{8}+x^{9}+x^{10}+x^{13}$$

$$M_{9}(x)=1+x^{5}+x^{6}+x^{7}+x^{8}+x^{12}+x^{13} \tag{3.4 a}$$

$$M_{11}(x) = 1 + x + x^5 + x^7 + x^8 + x^9 + x^{13}$$
$$M_{13}(x) = 1 + x^3 + x^4 + x^5 + x^6 + x^{12} + x^{13}$$
$$M_{15}(x) = 1 + x + x^2 + x^3 + x^4 + x^5 + x^7 + x^9 + x^{13} \qquad (3.4\text{ b})$$
$$M_{17}(x) = 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^8 + x^9 + x^{10} + x^{11} + x^{13}$$
$$M_{19}(x) = 1 + x^3 + x^5 + x^9 + x^{11} + x^{12} + x^{13}$$

Let the requirement be to synthesize a $t = 8$ BCH code with $k = 4096$ bits per sector. As per the BCH bound, the design distance $\delta = 2t + 1 = 17$. Assuming $b = 1$ (narrow sense BCH code) the required roots of $g(x)$ are: $\{\alpha, \alpha^2, \alpha^3, \ldots\ldots\ldots\ldots\ldots, \alpha^{b+\delta-2}\}$.

$$= \{\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}, \alpha^{13}, \alpha^{14}, \alpha^{15}, \alpha^{16}\} \qquad (3.5)$$

Hence $g(x)$ is computed as the *LCM* of the minimal polynomials of the required roots. For correction of 8-bit ($t = 8$), the generator polynomial

$$g(x) = LCM\{M_1(x), M_3(x), M_5(x), M_7(x), M_9(x), M_{11}(x), M_{13}(x), M_{15}(x)\} \qquad (3.6)$$

Therefore, the generator polynomial $g(x)$ for $t = 8$ is

$$\begin{aligned} g(x) = {} & 1 + x + x^5 + x^8 + x^9 + x^{11} + x^{12} + x^{13} + x^{14} + x^{15} + x^{18} + x^{22} + \\ & x^{23} + x^{24} + x^{26} + x^{30} + x^{31} + x^{32} + x^{38} + x^{40} + x^{41} + x^{42} + x^{47} + x^{48} + \\ & x^{49} + x^{52} + x^{58} + x^{59} + x^{64} + x^{65} + x^{67} + x^{68} + x^{69} + x^{70} + x^{77} + x^{78} + \\ & x^{79} + x^{82} + x^{84} + x^{88} + x^{91} + x^{92} + x^{93} + x^{94} + x^{95} + x^{96} + x^{98} + x^{100} + x^{104} \end{aligned} \qquad (3.7)$$

Here the *degree* $\{g(x)\}$ = 104 = $n - k$. $g(x)$ will define a (8191, 8087) BCH code. This code has to be shortened.

$n - k$ = 104, $\therefore n = k + 104$. But $k = 4096$. Hence $n = 4200$.

To be used in this application meeting the requirements of memory model 1, the BCH code with parameters (8191, 8087) has to be shortened by eliminating 3991 (i.e. 8087 – 4096) higher order positions which are set to zero. After this we obtain a shortened (8191-3991, 8087-3991) = (4200, 4096) BCH code.

Hence, for t = 8, the parameters of shortened BCH code are (4200, 4096).

The encoding process is briefly explained below. The message polynomial is expressed as

$$u(x) = u_0 + u_1 x + u_2 x^2 + \ldots\ldots\ldots\ldots\ldots\ldots + u_{k-1} x^{k-1}$$

$$= u_0 + u_1 x + u_2 x^2 + \ldots\ldots\ldots\ldots\ldots\ldots + u_{4095} x^{4095} + u_{4096} x^{4096} + \ldots\ldots\ldots + u_{8086} x^{8086} \quad (3.8)$$

Since the sector has only 4096 bits, the positions $u_{4096} \ldots\ldots\ldots u_{8086}$ are occupied by zeros.

Effectively $u(x) = u_0 + u_1 x + u_2 x^2 + \ldots\ldots\ldots\ldots\ldots\ldots + u_{4095} x^{4095}$ (3.9)

Pre multiplying $u(x)$ by $x^{n-k} = x^{104}$

$$x^{104} u(x) = u_0 x^{104} + u_1 x^{105} + u_2 x^{106} + \ldots\ldots + u_{4095} x^{4199} + u_{4096} x^{4200} + \ldots\ldots + u_{8086} x^{8190} \quad (3.10)$$

Again the coefficients $u_{4095} = \cdots\cdots\cdots = u_{8086} = 0$.

Dividing $x^{104} u(x)$ by $g(x)$ and extracting the remainder,

$$x^{104} u(x) = a(x) g(x) + r(x)$$

$$-r(x) + x^{104} u(x) = a(x) g(x) = v(x) \quad (3.11)$$

The negative sign is ignored because the computation is F_2, in which subtraction is same as addition.

$$\therefore \underbrace{r_0 + r_1 x + r_2 x^2 + \ldots\ldots r_{103} x^{103}}_{\text{redundant bits}} + \underbrace{u_0 x^{104} + \ldots\ldots\ldots\ldots + u_{4095} x^{4199}}_{\text{message bits}} + \underbrace{u_{4096} x^{4200} + \ldots\ldots\ldots + u_{8086} x^{8190}}_{\text{zeros}}$$

Similarly for t = 9, the design distance $\delta = 2t + 1 = 19$, $b = 1$, $b + \delta - 2 = 18$.

Hence the required roots are:

$\{\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}, \alpha^{13}, \alpha^{14}, \alpha^{15}, \alpha^{16}, \alpha^{17}, \alpha^{18}\}$

Therefore,

$$g(x) = LCM\left\{ M_1(x), M_3(x), M_5(x), M_7(x), M_9(x), M_{11}(x), M_{13}(x), M_{15}(x), M_{17}(x) \right\} \qquad (3.12)$$

$$\begin{aligned} g(x) = 1 &+ x^5 + x^6 + x^7 + x^9 + x^{10} + x^{13} + x^{17} + x^{18} + x^{19} + x^{20} + x^{21} + \\ &x^{23} + x^{25} + x^{26} + x^{29} + x^{30} + x^{31} + x^{32} + x^{33} + x^{36} + x^{37} + x^{39} + x^{40} + \\ &x^{44} + x^{45} + x^{47} + x^{49} + x^{51} + x^{54} + x^{55} + x^{58} + x^{59} + x^{60} + x^{63} + x^{64} + \\ &x^{65} + x^{67} + x^{69} + x^{70} + x^{71} + x^{72} + x^{76} + x^{78} + x^{81} + x^{82} + x^{83} + x^{84} + \\ &x^{85} + x^{86} + x^{87} + x^{89} + x^{90} + x^{91} + x^{96} + x^{101} + x^{103} + x^{105} + x^{107} + x^{111} + \\ &x^{112} + x^{114} + x^{115} + x^{117} \end{aligned} \qquad (3.13)$$

Let us now consider synthesis of a BCH code characterized by $t = 9$. According to the requirement of the BCH bound, the required roots are

$\{\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}, \alpha^{13}, \alpha^{14}, \alpha^{15}, \alpha^{16}, \alpha^{17}, \alpha^{18}\}$.

Hence, *degree* $\{g(x)\} = 117 = n - k$. This implies, $n = k + 117 = 4096 + 117 = 4213$ bits.

The original BCH code over $F_{2^{13}}$ has a natural length $n = 8191$. We need to start with a primitive BCH code with parameters (8191, 8191-117) = (8191, 8074). Again, $k = 4096$ bits. So, $8074 - 4096 = 3978$.

Therefore the parameters of the shortened BCH code are:

$$(8191 - 3978, 8074 - 3978) = (4213, 4096).$$

For $t = 10$, the design distance $\delta = 2t + 1 = 21, \quad b = 1, \quad b + \delta - 2 = 20$.

Hence the required roots are:

$\{\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}, \alpha^{13}, \alpha^{14}, \alpha^{15}, \alpha^{16}, \alpha^{17}, \alpha^{18}, \alpha^{19}, \alpha^{20}\}$

$$g(x) = LCM\begin{Bmatrix} M_1(x), M_3(x), M_5(x), M_7(x), M_9(x), M_{11}(x), \\ M_{13}(x), M_{15}(x), M_{17}(x), M_{19}(x) \end{Bmatrix} \qquad (3.14)$$

$$g(x) = 1 + x^3 + x^6 + x^7 + x^8 + x^9 + x^{10} + x^{12} + x^{13} + x^{16} + x^{17} + x^{19} + x^{25} + x^{26} + x^{27} + x^{28} + x^{29} + x^{30} + x^{32} + x^{34} + x^{36} + x^{44} + x^{45} + x^{47} + x^{49} + x^{51} + x^{57} + x^{58} + x^{59} + x^{60} + x^{62} + x^{63} + x^{65} + x^{66} + x^{67} + x^{70} + x^{73} + x^{77} + x^{79} + x^{80} + x^{83} + x^{86} + x^{87} + x^{88} + x^{89} + x^{90} + x^{95} + x^{97} + x^{100} + x^{102} + x^{103} + x^{108} + x^{109} + x^{110} + x^{111} + x^{113} + x^{114} + x^{115} + x^{118} + x^{121} + x^{123} + x^{124} + x^{126} + x^{129} + x^{130} \quad (3.15)$$

Here *degree* $\{ g(x) \}$ = 130 = $n - k$,

$n = k + 130 = 4096 + 130 = 4226$ bits. Since the original length of the BCH code over $F_{2^{13}}$ is 8191, we have to start with a primitive BCH code with parameters (8191, 8191-130) = (8191, 8061). As per the memory model in discussion, k= 4096 bits. Therefore, 8061 – 4096 = 3965. Therefore the parameters of the shortened BCH code are (4226, 4096). The memory model 1 allocates an overhead of 16 bytes (128 bits) for each sector (512 bytes). The number of overhead bits required equals $degree\{g(x)\}$. Table 3.1 summarizes the results of the synthesis.

Table 3.1. Number of overhead bits required for various values of t (Memory model 1)

t	**Number of overhead bits required (*degree{g(x)}*)**
8	104
9	117
10	130

The BCH code capable of correcting $t = 8$ errors requires 104 bits of overhead and the BCH code capable of correcting $t = 9$ errors requires 117 bits. These requirements are easily met by the given memory model where 128 bits are available for storing redundant bits generated by the error control code. However the BCH code specified by (3.15) for $t = 10$ requires 130 bits of overhead which is not supported in this model.

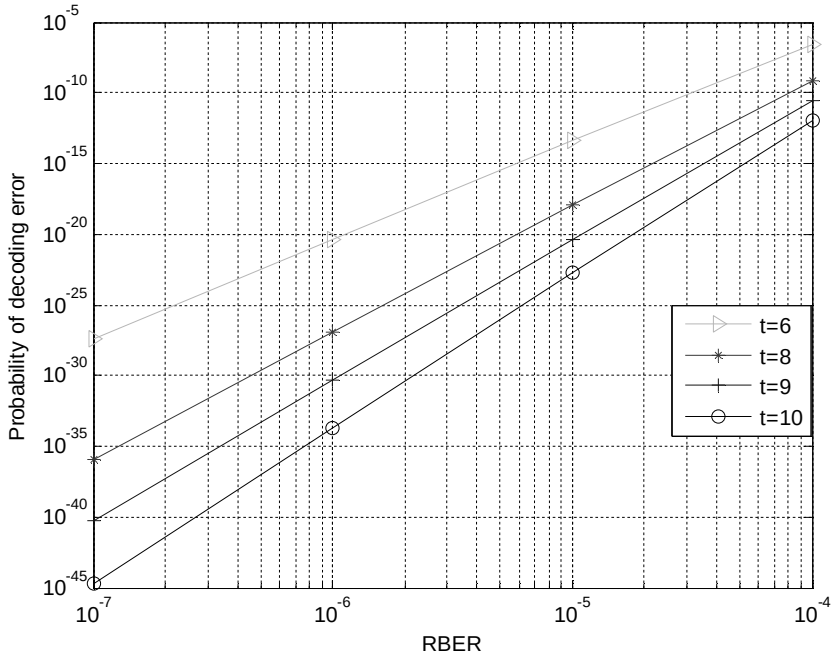


Figure 3.1: Performance of BCH Codes for enhancing Data Integrity in Flash memories for Memory model 1

The probability of decoding error which can also be called as UBER associated with a code that can correct t errors can be expressed as

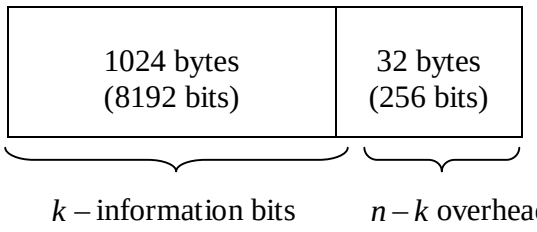
$$P_{decoding\ error} = \sum_{k=t+1}^{n} \binom{n}{k} (RBER)^k (1 - RBER)^{n-k}$$

The performance of theses codes is shown in Figure 3.1. We observe that the t = 9 code can transform a RBER of 10^{-5} to an UBER of 10^{-21}. Similarly, it can be observed that the same RBER can be transformed to an UBER of 10^{-13},10^{-18} and 10^{-23} by the codes with t = 6, t = 8 and t = 10 respectively. To be able to use this code we require overhead space of 17 bytes per sector, which is not available in this memory model. Hence, with the use of BCH codes, it is not possible to correct more than t = 9 errors per sector with memory model 1. Generally, the performance of an Error Control Code improves with increase in its length. This gives us the motivation to consider combining two sectors to constitute the information block which gives rise to one codeword. One codeword comprises of two

information bearing sectors (1024 bytes) and the corresponding overhead is increased to 32 bytes. We have investigated the possibility of constructing more powerful BCH codes with this adaptation.

3.2.2 Two Sectors as one Information Block

Suppose the contents of two sectors are taken as one information block, we have 1024 bytes of information and 32 bytes of overhead. We wish to design a shortened BCH code to meet this requirement.



The number of information bits k = 8192. Hence n > 8192. We have to start with a primitive BCH code of length $n = 2^{14} - 1 = 16383$. A primitive $(2^{14}-1)^{th}$ root of unity can be found in $F_{2^{14}}$. Hence the roots of $g(x)$ can be located in this field.

$n - k = 32 \times 8 = 256$ bits, $k - n - 256 = 16383 - 256 = 16127$ bits.

Parameters of the original BCH code over $F_{2^{14}}$ are (16383, 16127).

For t = 16, design distance $\delta = 2t + 1 = 33$, $b = 1$, $b + \delta - 2 = 32$

Required roots are:

$\{\alpha, \alpha^2, \alpha^3, .\alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}, \alpha^{13}, \alpha^{14}, \alpha^{15}, \alpha^{16}, \alpha^{17}, \alpha^{18}, \alpha^{19}, \alpha^{20},$
$\alpha^{21}, \alpha^{22}, \alpha^{23}, \alpha^{24}, \alpha^{25}, \alpha^{26}, \alpha^{27}, \alpha^{28}, \alpha^{29}, \alpha^{30}, \alpha^{31}, \alpha^{32}\}$

Similarly for t = 17, design distance $\delta = 2t + 1 = 35$, $b = 1$, $b + \delta - 2 = 34$.

Hence the required roots are:

$\{\alpha, \alpha^2, \alpha^3, .\alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}, \alpha^{13}, \alpha^{14}, \alpha^{15}, \alpha^{16}, \alpha^{17}, \alpha^{18}, \alpha^{19}, \alpha^{20},$
$\alpha^{21}, \alpha^{22}, \alpha^{23}, \alpha^{24}, \alpha^{25}, \alpha^{26}, \alpha^{27}, \alpha^{28}, \alpha^{29}, \alpha^{30}, \alpha^{31}, \alpha^{32}, \alpha^{33}, \alpha^{34}\}$

The required roots for other values of *t* can similarly be computed by invoking the BCH bound.

The conjugacy classes for $F_{2^{14}}$ are enumerated in (3.16).

1. $\{\alpha, \alpha^2, \alpha^4, \alpha^8, \alpha^{16}, \alpha^{32}, \alpha^{64}, \alpha^{128}, \alpha^{256}, \alpha^{512}, \alpha^{1024}, \alpha^{2048}, \alpha^{4096}, \alpha^{8192}\}$
2. $\{\alpha^3, \alpha^6, \alpha^{12}, \alpha^{24}, \alpha^{48}, \alpha^{96}, \alpha^{192}, \alpha^{384}, \alpha^{768}, \alpha^{1536}, \alpha^{3072}, \alpha^{6144}, \alpha^{12288}, \alpha^{8193}\}$
3. $\{\alpha^5, \alpha^{10}, \alpha^{20}, \alpha^{40}, \alpha^{80}, \alpha^{160}, \alpha^{320}, \alpha^{640}, \alpha^{1280}, \alpha^{2560}, \alpha^{5120}, \alpha^{2049}, \alpha^{10240}, \alpha^{4097}, \alpha^{8194}\}$
4. $\{\alpha^7, \alpha^{14}, \alpha^{28}, \alpha^{56}, \alpha^{112}, \alpha^{224}, \alpha^{448}, \alpha^{896}, \alpha^{1792}, \alpha^{3584}, \alpha^{7168}, \alpha^{14336}, \alpha^{12289}, \alpha^{8195}\}$
5. $\{\alpha^9, \alpha^{18}, \alpha^{36}, \alpha^{72}, \alpha^{144}, \alpha^{288}, \alpha^{576}, \alpha^{1152}, \alpha^{2304}, \alpha^{4608}, \alpha^{9216}, \alpha^{2049}, \alpha^{4098}, \alpha^{8196}\}$
6. $\{\alpha^{11}, \alpha^{22}, \alpha^{44}, \alpha^{88}, \alpha^{176.}, \alpha^{352}, \alpha^{704}, \alpha^{1408}, \alpha^{2816}, \alpha^{5632}, \alpha^{11264}, \alpha^{6145}, \alpha^{12290}, \alpha^{8197}\}$
7. $\{\alpha^{13}, \alpha^{26}, \alpha^{52}, \alpha^{104}, \alpha^{208}, \alpha^{416}, \alpha^{832}, \alpha^{1664}, \alpha^{3328}, \alpha^{6656}, \alpha^{13312}, \alpha^{10241}, \alpha^{4099}, \alpha^{8198}\}$
8. $\{\alpha^{15}, \alpha^{30}, \alpha^{60}, \alpha^{120}, \alpha^{240}, \alpha^{480}, \alpha^{960}, \alpha^{1920}, \alpha^{3840}, \alpha^{7680}, \alpha^{15360}, \alpha^{14337}, \alpha^{12291}, \alpha^{8199}\}$ (3.16)
9. $\{\alpha^{17}, \alpha^{34}, \alpha^{68}, \alpha^{136}, \alpha^{272}, \alpha^{544}, \alpha^{1088}, \alpha^{2176}, \alpha^{4352}, \alpha^{8704}, \alpha^{1025}, \alpha^{2050}, \alpha^{4100}, \alpha^{8200}\}$
10. $\{\alpha^{19}, \alpha^{38}, \alpha^{76}, \alpha^{152}, \alpha^{304}, \alpha^{608}, \alpha^{1216}, \alpha^{2432}, \alpha^{4864}, \alpha^{9728}, \alpha^{3077}, \alpha^{6154}, \alpha^{12308}, \alpha^{8201}\}$
11. $\{\alpha^{21}, \alpha^{42}, \alpha^{84}, \alpha^{168}, \alpha^{336}, \alpha^{672}, \alpha^{1344}, \alpha^{2688}, \alpha^{5376}, \alpha^{10752}, \alpha^{5121}, \alpha^{10242}, \alpha^{4101}, \alpha^{8202}\}$
12. $\{\alpha^{23}, \alpha^{46}, \alpha^{92}, \alpha^{184}, \alpha^{368}, \alpha^{736}, \alpha^{1472}, \alpha^{2944}, \alpha^{5888}, \alpha^{11776}, \alpha^{7169}, \alpha^{14338}, \alpha^{12293}, \alpha^{8203}\}$
13. $\{\alpha^{25}, \alpha^{50}, \alpha^{100}, \alpha^{200}, \alpha^{400}, \alpha^{800}, \alpha^{1600}, \alpha^{3200}, \alpha^{6400}, \alpha^{12800}, \alpha^{9217}, \alpha^{2051}, \alpha^{4102}, \alpha^{8204}\}$
14. $\{\alpha^{27}, \alpha^{54}, \alpha^{108}, \alpha^{216}, \alpha^{432}, \alpha^{864}, \alpha^{1728}, \alpha^{3456}, \alpha^{6912}, \alpha^{13824}, \alpha^{11265}, \alpha^{6147}, \alpha^{12294}, \alpha^{8205}\}$
15. $\{\alpha^{29}, \alpha^{58}, \alpha^{116}, \alpha^{232}, \alpha^{464}, \alpha^{928}, \alpha^{1856}, \alpha^{3712}, \alpha^{7424}, \alpha^{14848}, \alpha^{13313}, \alpha^{10243}, \alpha^{4103}, \alpha^{8206}\}$
16. $\{\alpha^{31}, \alpha^{62}, \alpha^{124}, \alpha^{248}, \alpha^{496}, \alpha^{992}, \alpha^{1984}, \alpha^{3968}, \alpha^{7936}, \alpha^{15872}, \alpha^{15361}, \alpha^{14339}, \alpha^{12295}, \alpha^{8207}\}$
17. $\{\alpha^{33}, \alpha^{66}, \alpha^{132}, \alpha^{264}, \alpha^{528}, \alpha^{1056}, \alpha^{2112}, \alpha^{4224}, \alpha^{8448}, \alpha^{513}, \alpha^{1026}, \alpha^{2052}, \alpha^{4104}, \alpha^{8208}\}$
18. $\{\alpha^{35}, \alpha^{70}, \alpha^{140}, \alpha^{280}, \alpha^{560}, \alpha^{1120}, \alpha^{2240}, \alpha^{4480}, \alpha^{8960}, \alpha^{1537}, \alpha^{3074}, \alpha^{6148}, \alpha^{12296}, \alpha^{8209}\}$
19. $\{\alpha^{37}, \alpha^{74}, \alpha^{148}, \alpha^{296}, \alpha^{592}, \alpha^{1184}, \alpha^{2368}, \alpha^{4736}, \alpha^{9472}, \alpha^{2561}, \alpha^{5122}, \alpha^{10244}, \alpha^{4105}, \alpha^{8210}\}$

20. $\{\ \alpha^{39},\alpha^{78},\alpha^{156},\alpha^{312},\alpha^{624},\alpha^{1248},\alpha^{2496},\alpha^{4992},\alpha^{9984},\alpha^{3585},\alpha^{7170},\alpha^{14340},\alpha^{12297},\alpha^{8211}\}$

21. $\{\alpha^{41},\alpha^{82},\alpha^{164},\alpha^{328},\alpha^{656},\alpha^{1312},\alpha^{2624},\alpha^{5248},\alpha^{10496},\alpha^{4609},\alpha^{9218},\alpha^{2053},\alpha^{4106},\alpha^{8212}\}$

22. $\{\ \alpha^{43},\alpha^{86},\alpha^{172},\alpha^{344},\alpha^{688},\alpha^{1376},\alpha^{2752},\alpha^{5504},\alpha^{11008},\alpha^{5633},\alpha^{11266},\alpha^{6149},\alpha^{12298},\alpha^{8213}\}$

The corresponding minimal polynomials obtained are:

$$
\begin{aligned}
M_1(x) &= 1 + x + x^6 + x^{10} + x^{14} \\
M_3(x) &= 1 + x + x^2 + x^5 + x^8 + x^{14} \\
M_5(x) &= 1 + x + x^3 + x^4 + x^6 + x^7 + x^9 + x^{10} + x^{14} \\
M_7(x) &= 1 + x^4 + x^5 + x^6 + x^7 + x^8 + x^9 + x^{12} + x^{14} \\
M_9(x) &= 1 + x^2 + x^3 + x^5 + x^{11} + x^{12} + x^{14} \\
M_{11}(x) &= 1 + x\ \ + x^6 + x^8 + x^{14} \\
M_{13}(x) &= 1 + x^5 + x^6 + x^9 + x^{10} + x^{11} + x^{12} + x^{13} + x^{14} \\
M_{15}(x) &= 1 + x^2 + x^5 + x^7 + x^8 + x^9 + x^{10} + x^{11} + x^{14} \\
M_{17}(x) &= 1 + x + x^2 + x^3 + x^4 + x^5 + x^7 + x^8 + x^{10} + x^{11} + x^{14} \\
M_{19}(x) &= 1 + x + x^6 + x^{11} + x^{14} \\
M_{21}(x) &= 1 + x^2 + x^4 + x^8 + x^{10} + x^{11} + x^{14} \\
M_{23}(x) &= 1 + x^2 + x^5 + x^6 + x^9 + x^{11} + x^{14} \\
M_{25}(x) &= 1 + x + x^6 + x^7 + x^{10} + x^{11} + x^{12} + x^{13} + x^{14} \\
M_{27}(x) &= 1 + x + x^7 + x^8 + x^{10} + x^{13} + x^{14} \\
M_{29}(x) &= 1 + x + x^2 + x^3 + x^5 + x^8 + x^{11} + x^{13} + x^{14} \\
M_{31}(x) &= 1 + x^3 + x^4 + x^7 + x^9 + x^{10} + x^{11} + x^{12} + x^{14} \\
M_{33}(x) &= 1 + x + x^2 + x^3 + x^4 + x^6 + x^7 + x^8 + x^{10} + x^{12} + x^{14} \\
M_{35}(x) &= 1 + x + x^2 + x^4 + x^5 + x^6 + x^{11} + x^{13} + x^{14} \\
M_{37}(x) &= 1 + x + x^3 + x^5 + x^6 + x^{13} + x^{14}
\end{aligned}
\tag{3.17}
$$

The generator polynomial for codes designed to handle two sectors as one information block is computed by evaluating the *LCM* of minimal polynomials corresponding to the required roots.

(i) for t =16,

$$g(x) = LCM\begin{Bmatrix} M_1(x), M_3(x), M_5(x), M_7(x), M_9(x), M_{11}(x), M_{13}(x), M_{15}(x), \\ M_{17}(x), M_{19}(x), M_{21}(x), M_{23}(x), M_{25}(x), M_{27}(x), M_{29}(x), M_{31}(x) \end{Bmatrix} \quad (3.18)$$

Therefore,

$$\begin{aligned} g(x) = 1 + x + x^2 + x^3 + x^6 + x^7 + x^9 + x^{10} + x^{12} + x^{13} + x^{14} + x^{16} + x^{17} + \\ x^{23} + x^{25} + x^{27} + x^{29} + x^{31} + x^{32} + x^{35} + x^{39} + x^{40} + x^{42} + x^{43} + x^{45} + \\ x^{46} + x^{47} + x^{48} + x^{51} + x^{53} + x^{54} + x^{55} + x^{58} + x^{62} + x^{67} + x^{68} + x^{71} + \\ x^{72} + x^{74} + x^{76} + x^{78} + x^{79} + x^{80} + x^{81} + x^{86} + x^{87} + x^{88} + x^{89} + x^{93} + \\ x^{94} + x^{95} + x^{96} + x^{97} + x^{98} + x^{100} + x^{102} + x^{103} + x^{104} + x^{105} + x^{106} + \\ x^{107} + x^{108} + x^{109} + x^{110} + x^{112} + x^{113} + x^{114} + x^{115} + x^{118} + x^{120} + x^{123} + \\ x^{124} + x^{126} + x^{133} + x^{136} + x^{140} + x^{143} + x^{145} + x^{149} + x^{151} + x^{153} + x^{158} + \\ x^{159} + x^{162} + x^{164} + x^{168} + x^{171} + x^{174} + x^{176} + x^{177} + x^{181} + x^{183} + x^{185} + \\ x^{187} + x^{190} + x^{196} + x^{197} + x^{198} + x^{202} + x^{203} + x^{210} + x^{213} + x^{218} + x^{219} + \\ x^{222} + x^{223} + x^{224} \end{aligned} \quad (3.19)$$

Here *degree* $\{g(x)\}$ = 224 = $n - k$, $n = k + 224 = 8192 + 224 = 8416$ bits. Since the original length of the BCH code over $F_{2^{14}}$ is $2^{14} - 1 = 16383$, the primitive BCH code will have the parameter (16383, 16383 – 224) = (16383, 16159).

Since k = 8192 bits, 16159 – 8192 =7967 bits. Therefore the parameters of the shortened BCH code are (8416, 8192).

Since ($n - k$) = 256 and *degree* $\{g(x)\}$ = 224, the code can correct sixteen bit errors.

(ii) for t = 17,

$$g(x) = LCM\begin{Bmatrix} M_1(x), M_3(x), M_5(x), M_7(x), M_9(x), M_{11}(x), M_{13}(x), \\ M_{15}(x), M_{17}(x), M_{19}(x), M_{21}(x), M_{23}(x), M_{25}(x), \\ M_{27}(x), M_{29}(x), M_{31}(x), M_{33}(x) \end{Bmatrix} \quad (3.20)$$

Therefore

$$g(x) = 1 + x^2 + x^5 + x^7 + x^8 + x^{10} + x^{11} + x^{15} + x^{20} + x^{21} + x^{23} + x^{26} + x^{27} + x^{29} +$$
$$x^{34} + x^{35} + x^{36} + x^{38} + x^{39} + x^{40} + x^{42} + x^{43} + x^{46} + x^{47} + x^{53} + x^{57} + x^{60} + x^{62} +$$
$$x^{64} + x^{65} + x^{66} + x^{67} + x^{68} + x^{71} + x^{72} + x^{73} + x^{75} + x^{76} + x^{80} + x^{81} + x^{83} + x^{86} +$$
$$x^{87} + x^{90} + x^{91} + x^{92} + x^{95} + x^{96} + x^{97} + x^{98} + x^{99} + x^{102} + x^{106} + x^{107} + x^{109} + x^{110} +$$
$$x^{115} + x^{119} + x^{121} + x^{122} + x^{123} + x^{124} + x^{125} + x^{126} + x^{128} + x^{129} + x^{130} + x^{133} + x^{137} + \quad (3.21)$$
$$x^{138} + x^{144} + x^{145} + x^{148} + x^{149} + x^{153} + x^{154} + x^{155} + x^{157} + x^{160} + x^{161} + x^{162} + x^{163} +$$
$$x^{164} + x^{167} + x^{169} + x^{172} + x^{174} + x^{175} + x^{176} + x^{179} + x^{183} + x^{184} + x^{185} + x^{188} + x^{189} +$$
$$x^{193} + x^{199} + x^{201} + x^{208} + x^{209} + x^{210} + x^{211} + x^{212} + x^{213} + x^{214} + x^{216} + x^{217} + x^{219} +$$
$$x^{221} + x^{224} + x^{226} + x^{228} + x^{229} + x^{231} + x^{232} + x^{235} + x^{237} + x^{238}$$

In this case *degree* $\{g(x)\}$ = 238 = $n - k$. $n = k + 238 = 8192 + 238 = 8430$ bits. Therefore the primitive code parameter is (16383, 16383 – 238) = (16383, 16145).

Since $k = 8192$, $16145 - 8192 = 7953$. Therefore the parameters of the shortened BCH code are (8430, 8192). Since *degree* $\{g(x)\} < (n - k)$, the code can correct up to seventeen bits in error over two sectors.

(iii) for $t = 18$, the design distance $\delta = 2t + 1 = 37$, $b = 1$, $b + \delta - 2 = 36$.

Hence the required roots are:

$$\{\alpha, \alpha^2, \alpha^3, .\alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}, \alpha^{13}, \alpha^{14}, \alpha^{15}, \alpha^{16}, \alpha^{17}, \alpha^{18}, \alpha^{19}, \alpha^{20},$$
$$\alpha^{21}, \alpha^{22}, \alpha^{23}, \alpha^{24}, \alpha^{25}, \alpha^{26}, \alpha^{27}, \alpha^{28}, \alpha^{29}, \alpha^{30}, \alpha^{31}, \alpha^{32}, \alpha^{33}, \alpha^{34}, \alpha^{35}, \alpha^{36}\}$$

$$g(x) = LCM\left\{\begin{matrix} M_1(x), M_3(x), M_5(x), M_7(x), M_9(x), M_{11}(x), M_{13}(x), M_{15}(x), \\ M_{17}(x), M_{19}(x), M_{21}(x), M_{23}(x), M_{25}(x), M_{27}(x), M_{29}(x), \\ M_{31}(x), M_{33}(x), M_{35}(x) \end{matrix}\right\} \quad (3.22)$$

$$g(x) = 1 + x + x^3 + x^6 + x^7 + x^8 + x^9 + x^{10} + x^{11} + x^{13} + x^{14} + x^{16} + x^{19} +$$
$$x^{20} + x^{23} + x^{32} + x^{33} + x^{34} + x^{39} + x^{40} + x^{41} + x^{42} + x^{43} + x^{44} + x^{49} +$$
$$x^{50} + x^{53} + x^{54} + x^{58} + x^{59} + x^{60} + x^{61} + x^{62} + x^{65} + x^{67} + x^{70} + x^{72} +$$
$$x^{73} + x^{75} + x^{77} + x^{79} + x^{80} + x^{82} + x^{83} + x^{85} + x^{90} + x^{91} + x^{94} + x^{97} +$$
$$x^{98} + x^{99} + x^{102} + x^{104} + x^{105} + x^{108} + x^{109} + x^{112} + x^{116} + x^{118} + x^{119} +$$
$$x^{120} + x^{121} + x^{122} + x^{124} + x^{125} + x^{127} + x^{130} + x^{132} + x^{133} + x^{137} + x^{138} + \quad (3.23)$$
$$x^{140} + x^{141} + x^{146} + x^{148} + x^{149} + x^{150} + x^{153} + x^{155} + x^{156} + x^{158} + x^{161} +$$
$$x^{163} + x^{164} + x^{165} + x^{168} + x^{169} + x^{170} + x^{172} + x^{173} + x^{177} + x^{178} + x^{179} +$$
$$x^{184} + x^{187} + x^{191} + x^{192} + x^{193} + x^{195} + x^{197} + x^{200} + x^{202} + x^{203} + x^{208} +$$
$$x^{211} + x^{214} + x^{216} + x^{217} + x^{218} + x^{219} + x^{224} + x^{226} + x^{227} + x^{228} + x^{230} +$$
$$x^{233} + x^{235} + x^{236} + x^{239} + x^{241} + x^{242} + x^{250} + x^{252}$$

Here *degree* $\{g(x)\}$ = 252 = $n - k$. $n = k + 252 = 8192 + 252 = 8444$ bits. Therefore the primitive code parameter is (16383, 16383 – 252) = (16383, 16131). According to the memory model considered, k = 8192. Hence 16,131 – 8192 = 7939. Therefore the parameters of the shortened BCH code are (8444, 8192). Since *degree* $\{ g(x) \} < (n - k)$, the overhead requirements of the code can be satisfied by the architecture and the code can support correction of up to eighteen bits in error over two sectors.

(iv) for $t = 19$, $\delta = 2t + 1 = 39$, $b = 1$, $b + \delta - 2 = 38$.

Hence the required roots are:

$$\{\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8, \alpha^9, \alpha^{10}, \alpha^{11}, \alpha^{12}, \alpha^{13}, \alpha^{14}, \alpha^{15}, \alpha^{16}, \alpha^{17}, \alpha^{18}, \alpha^{19}, \alpha^{20},$$
$$\alpha^{21}, \alpha^{22}, \alpha^{23}, \alpha^{24}, \alpha^{25}, \alpha^{26}, \alpha^{27}, \alpha^{28}, \alpha^{29}, \alpha^{30}, \alpha^{31}, \alpha^{32}, \alpha^{33}, \alpha^{34}, \alpha^{35}, \alpha^{36}, \alpha^{37}, \alpha^{38}\}$$

$$g(x) = LCM \begin{Bmatrix} M_1(x), M_3(x), M_5(x), M_7(x), M_9(x), M_{11}(x), M_{13}(x), M_{15}(x), \\ M_{17}(x), M_{19}(x), M_{21}(x), M_{23}(x), M_{25}(x), M_{27}(x), M_{29}(x), \\ M_{31}(x), M_{33}(x), M_{35}(x), M_{37}(x) \end{Bmatrix} \quad (3.24)$$

$$g(x) = 1 + x^2 + x^5 + x^7 + x^8 + x^{10} + x^{13} + x^{14} + x^{16} + x^{18} + x^{19} + x^{20} + x^{25} +$$
$$x^{26} + x^{30} + x^{34} + x^{37} + x^{39} + x^{40} + x^{42} + x^{43} + x^{45} + x^{46} + x^{47} + x^{48} + x^{49} +$$
$$x^{50} + x^{54} + x^{55} + x^{57} + x^{58} + x^{60} + x^{61} + x^{63} + x^{67} + x^{70} + x^{71} + x^{74} + x^{75} +$$
$$x^{78} + x^{79} + x^{80} + x^{81} + x^{82} + x^{83} + x^{84} + x^{85} + x^{86} + x^{90} + x^{93} + x^{94} + x^{95} +$$
$$x^{98} + x^{100} + x^{101} + x^{102} + x^{104} + x^{105} + x^{106} + x^{107} + x^{109} + x^{113} + x^{115} + x^{117} + \quad (3.25)$$
$$x^{121} + x^{128} + x^{130} + x^{131} + x^{134} + x^{136} + x^{137} + x^{139} + x^{140} + x^{144} + x^{146} + x^{147} +$$
$$x^{148} + x^{149} + x^{150} + x^{151} + x^{152} + x^{154} + x^{157} + x^{158} + x^{161} + x^{164} + x^{167} + x^{170} +$$
$$x^{172} + x^{176} + x^{185} + x^{187} + x^{188} + x^{189} + x^{190} + x^{191} + x^{192} + x^{196} + x^{198} + x^{199} +$$
$$x^{201} + x^{209} + x^{210} + x^{211} + x^{212} + x^{217} + x^{220} + x^{221} + x^{225} + x^{226} + x^{228} + x^{230} +$$
$$x^{231} + x^{233} + x^{238} + x^{241} + x^{242} + x^{244} + x^{247} + x^{251} + x^{253} + x^{254} + x^{257} + x^{258} +$$
$$x^{263} + x^{264} + x^{265} + x^{266}$$

It is to be noted that *degree* $\{g(x)\}$ = 266 = $n - k$. $n = k + 266 = 8192 + 266 = 8458$ bits.
Primitive code parameter is (16383, 16383 – 266) = (16383, 16117).
k = 8192, 16117 – 8192 = 7925.
Parameters of shortened BCH code are: (8458, 8192).

Since *degree* $\{g(x)\}$ > 256 (i.e. $n - k$ value as per the model), a $t = 19$ error correcting BCH code has a overhead requirement that cannot be supported with this memory model. However, if the memory allocated to store redundant overhead bits is increased from 32 to 34 bits then this solution can be employed. Further, in a similar manner, we have also worked out the generator polynomials of BCH codes capable of correcting $t = 20, 21,$ and 22 errors per 1024 bytes which can be used if memories with higher data integrity are required. However, this will necessitate the use of a greater overhead allocation for storing redundant bits generated by the code.

(v) for t = 20, the design distance $\delta = 2t + 1 = 41$, $b = 1$, $b + \delta - 2 = 40$.
Hence the required roots are: $\{\alpha, \alpha^2, \alpha^3, ..., \alpha^{40}\}$.

$$g(x) = 1 + x^3 + x^4 + x^6 + x^8 + x^9 + x^{14} + x^{15} + x^{16} + x^{20} + x^{23} + x^{24} +$$
$$x^{27} + x^{29} + x^{30} + x^{33} + x^{35} + x^{36} + x^{38} + x^{40} + x^{41} + x^{42} + x^{44} + x^{46} +$$
$$x^{48} + x^{49} + x^{50} + x^{51} + x^{52} + x^{56} + x^{60} + x^{62} + x^{64} + x^{65} + x^{68} + x^{69} +$$
$$x^{73} + x^{76} + x^{77} + x^{79} + x^{80} + x^{82} + x^{84} + x^{90} + x^{91} + x^{92} + x^{93} + x^{95} +$$
$$x^{99} + x^{100} + x^{101} + x^{105} + x^{110} + x^{113} + x^{114} + x^{116} + x^{118} + x^{119} + x^{120} +$$
$$x^{121} + x^{122} + x^{124} + x^{125} + x^{126} + x^{127} + x^{129} + x^{130} + x^{132} + x^{135} + x^{136} +$$
$$x^{137} + x^{139} + x^{141} + x^{143} + x^{148} + x^{149} + x^{150} + x^{152} + x^{153} + x^{156} + x^{157} + \quad (3.26)$$
$$x^{159} + x^{161} + x^{162} + x^{167} + x^{169} + x^{171} + x^{172} + x^{174} + x^{178} + x^{179} + x^{181} +$$
$$x^{183} + x^{184} + x^{189} + x^{192} + x^{193} + x^{194} + x^{195} + x^{198} + x^{200} + x^{202} + x^{203} +$$
$$x^{205} + x^{207} + x^{208} + x^{209} + x^{212} + x^{213} + x^{214} + x^{215} + x^{216} + x^{218} + x^{220} +$$
$$x^{222} + x^{223} + x^{224} + x^{225} + x^{226} + x^{227} + x^{228} + x^{229} + x^{231} + x^{235} + x^{236} +$$
$$x^{237} + x^{238} + x^{239} + x^{240} + x^{243} + x^{249} + x^{251} + x^{253} + x^{255} + x^{256} + x^{257} +$$
$$x^{261} + x^{263} + x^{265} + x^{267} + x^{271} + x^{273} + x^{274} + x^{275} + x^{280}$$

In this case, *degree* $\{g(x)\}$ = 280 = (n – k). n = k + 280 = 8192 +280 =8472 bits.

Primitive code parameter is (16383, 16383 – 280) = (16383, 16103).

k = 8192, 16103– 8192 = 7911.

Parameters of shortened BCH code are: (8472, 8192).

(vi) for t = 21, the design distance $\delta = 2t + 1 = 43$, $b = 1$, $b + \delta - 2 = 42$.

$$g(x) = 1 + x \ + x^6 + x^7 + x^8 + x^{10} + x^{11} + x^{12} + x^{13} + x^{15} + x^{16} + x^{19} + x^{21} + x^{27} +$$
$$x^{28} + x^{29} + x^{32} + x^{34} + x^{36} + x^{38} + x^{39} + x^{41} + x^{44} + x^{46} + x^{47} + x^{48} + x^{49} + x^{50} +$$
$$x^{51} + x^{53} + x^{55} + x^{56} + x^{57} + x^{59} + x^{61} + x^{62} + x^{66} + x^{68} + x^{69} + x^{70} + x^{71} + x^{73} +$$
$$x^{75} + x^{76} + x^{77} + x^{79} + x^{83} + x^{84} + x^{85} + x^{86} + x^{88} + x^{90} + x^{91} + x^{97} + x^{99} + x^{105} +$$
$$x^{107} + x^{109} + x^{112} + x^{113} + x^{115} + x^{118} + x^{119} + x^{121} + x^{122} + x^{123} + x^{124} + x^{125} + x^{126} +$$
$$x^{129} + x^{131} + x^{132} + x^{133} + x^{137} + x^{1141} + x^{145} + x^{151} + x^{156} + x^{157} + x^{158} + x^{163} + x^{164} + \quad (3.27)$$
$$x^{165} + x^{172} + x^{173} + x^{174} + x^{175} + x^{178} + x^{180} + x^{185} + x^{189} + x^{191} + x^{193} + x^{195} + x^{201} +$$
$$x^{204} + x^{206} + x^{208} + x^{209} + x^{211} + x^{212} + x^{214} + x^{215} + x^{217} + x^{220} + x^{221} + x^{223} + x^{224} +$$
$$x^{227} + x^{229} + x^{232} + x^{234} + x^{235} + x^{236} + x^{239} + x^{247} + x^{248} + x^{250} + x^{251} + x^{252} + x^{254} +$$
$$x^{255} + x^{256} + x^{257} + x^{258} + x^{259} + x^{260} + x^{261} + x^{265} + x^{266} + x^{267} + x^{270} + x^{272} + x^{275} +$$
$$x^{276} + x^{279} + x^{280} + x^{281} + x^{286} + x^{287} + x^{291} + x^{292} + x^{294}$$

(vii) for $t = 22$, the design distance $\delta = 2t + 1 = 45$, $b = 1$, $b + \delta - 2 = 44$

$$
\begin{aligned}
g(x) = 1 &+ x^3 + x^4 + x^6 + x^9 + x^{12} + x^{13} + x^{15} + x^{16} + x^{19} + x^{20} + x^{25} + x^{26} + x^{29} + \\
&x^{30} + x^{32} + x^{34} + x^{35} + x^{37} + x^{41} + x^{43} + x^{45} + x^{48} + x^{51} + x^{53} + x^{54} + x^{55} + x^{56} + \\
&x^{57} + x^{59} + x^{62} + x^{65} + x^{67} + x^{69} + x^{70} + x^{72} + x^{73} + x^{74} + x^{75} + x^{76} + x^{77} + x^{78} + \\
&x^{80} + x^{82} + x^{83} + x^{84} + x^{85} + x^{86} + x^{87} + x^{88} + x^{90} + x^{91} + x^{92} + x^{94} + x^{95} + x^{99} + \\
&x^{103} + x^{105} + x^{106} + x^{108} + x^{109} + x^{113} + x^{117} + x^{121} + x^{122} + x^{123} + x^{125} + x^{126} + x^{129} + \\
&x^{136} + x^{139} + x^{141} + x^{142} + x^{146} + x^{147} + x^{149} + x^{150} + x^{151} + x^{152} + x^{153} + x^{154} + x^{155} + \\
&x^{156} + x^{157} + x^{162} + x^{165} + x^{167} + x^{168} + x^{169} + x^{170} + x^{171} + x^{172} + x^{175} + x^{176} + x^{177} + \\
&x^{179} + x^{180} + x^{184} + x^{186} + x^{189} + x^{190} + x^{191} + x^{192} + x^{193} + x^{196} + x^{198} + x^{203} + x^{205} + \\
&x^{206} + x^{207} + x^{210} + x^{211} + x^{218} + x^{219} + x^{223} + x^{225} + x^{226} + x^{227} + x^{228} + x^{229} + x^{230} + \\
&x^{231} + x^{232} + x^{233} + x^{234} + x^{235} + x^{238} + x^{241} + x^{242} + x^{243} + x^{244} + x^{246} + x^{248} + x^{249} + \\
&x^{250} + x^{252} + x^{253} + x^{254} + x^{256} + x^{261} + x^{262} + x^{263} + x^{267} + x^{268} + x^{269} + x^{272} + x^{273} + \\
&x^{274} + x^{279} + x^{280} + x^{285} + x^{286} + x^{288} + x^{289} + x^{290} + x^{293} + x^{294} + x^{296} + x^{297} + x^{299} + \\
&x^{300} + x^{303} + x^{304} + x^{307} + x^{308}
\end{aligned}
\tag{3.28}
$$

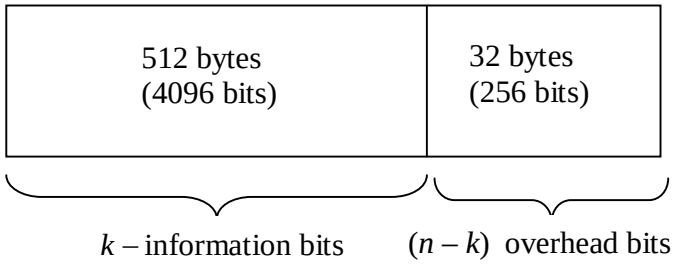
Table 3.2 shows the summary of the results obtained after synthesizing the codes by taking two information sectors as one information block. The codes specified in the shaded blocks cannot be supported by the device architecture.

Table 3.2. Number of overhead bits required for various values of t (Combining two sectors as one information block)

t	**Number of overhead bits required (*degree{g(x)}*)**
16	224
17	238
18	252
19	266
20	280
21	294
22	308

3.2.3 Memory Model 2

Let us consider the second model (memory model 2) with 512 bytes of data and 32 bytes of overhead. The natural length $n = 2^{13} - 1$ = 8191 bits.



Choosing n = 8192 – 1 = 8191 = $(2^{13} - 1)$.

$n - k = 256, \quad k = 7935$.

Parameters of this code are (8191, 7935).

From the model, $n = k + 256 = 4096 + 256 = 4352$.

$(n, k) = (4352, 4096)$.

Therefore the required shortened BCH code is (4352, 4096).

It is known that b = 1, design distance $\delta = 2t + 1$.

For t = 15, $\delta = 2t + 1$ = 31.

Required roots are: $\{\alpha, \alpha^2, \alpha^3, \ldots\ldots\ldots \alpha^{b+\delta-2} = \alpha^{30}\}$.

Similarly, for t = 16, the required roots are :$\{\alpha, \alpha^2, \alpha^3, \ldots\ldots\ldots \alpha^{b+\delta-2} = \alpha^{32}\}$

for t = 17, the required roots are :$\{\alpha, \alpha^2, \alpha^3, \ldots\ldots\ldots \alpha^{b+\delta-2} = \alpha^{34}\}$

for t = 18, the required roots are :$\{\alpha, \alpha^2, \alpha^3, \ldots\ldots\ldots \alpha^{b+\delta-2} = \alpha^{36}\}$

for t = 19, the required roots are :$\{\alpha, \alpha^2, \alpha^3, \ldots\ldots\ldots \alpha^{b+\delta-2} = \alpha^{38}\}$

for t = 20, the required roots are :$\{\alpha, \alpha^2, \alpha^3, \ldots\ldots\ldots \alpha^{b+\delta-2} = \alpha^{40}\}$

for t = 21, the required roots are :$\{\alpha, \alpha^2, \alpha^3, \ldots\ldots\ldots \alpha^{b+\delta-2} = \alpha^{42}\}$

for t = 22, the required roots are :$\{\alpha, \alpha^2, \alpha^3, \ldots\ldots\ldots \alpha^{b+\delta-2} = \alpha^{44}\}$

Conjugacy classes of the elements of $F_{2^{13}}$ are listed in (3.1). The conjugacy classes of the remaining elements in $F_{2^{13}}$ are listed below.

10. $\{\alpha^{21},\alpha^{42},\alpha^{84},\alpha^{168},\alpha^{336},\alpha^{672},\alpha^{1344},\alpha^{2688},\alpha^{5372},\alpha^{2561},\alpha^{5122},\alpha^{2053},\alpha^{4106}\}$
11. $\{\alpha^{23},\alpha^{46},\alpha^{92},\alpha^{184},\alpha^{368},\alpha^{736},\alpha^{1472},\alpha^{2944},\alpha^{5888},\alpha^{3585},\alpha^{7170},\alpha^{6149},\alpha^{4107}\}$
12. $\{\alpha^{25},\alpha^{50},\alpha^{100},\alpha^{200},\alpha^{400},\alpha^{800},\alpha^{1600},\alpha^{3200},\alpha^{6400},\alpha^{4609},\alpha^{1027},\alpha^{2054},\alpha^{4108}\}$
13. $\{\alpha^{27},\alpha^{54},\alpha^{108},\alpha^{216},\alpha^{432},\alpha^{864},\alpha^{1728},\alpha^{3456},\alpha^{6912},\alpha^{5633},\alpha^{3075},\alpha^{6150},\alpha^{4109}\}$
14. $\{\alpha^{29},\alpha^{58},\alpha^{116},\alpha^{232},\alpha^{464},\alpha^{928},\alpha^{1856},\alpha^{3712},\alpha^{7424},\alpha^{6657},\alpha^{5123},\alpha^{2055},\alpha^{4110}\}$
15. $\{\alpha^{31},\alpha^{62},\alpha^{124},\alpha^{248},\alpha^{496},\alpha^{992},\alpha^{1984},\alpha^{3968},\alpha^{7936},\alpha^{7681},\alpha^{7171},\alpha^{6151},\alpha^{4111}\}$ (3.29)
16. $\{\alpha^{33},\alpha^{66},\alpha^{132},\alpha^{264},\alpha^{528},\alpha^{1056},\alpha^{2112},\alpha^{4224},\alpha^{257},\alpha^{514},\alpha^{1028},\alpha^{2056},\alpha^{4112}\}$
17. $\{\alpha^{35},\alpha^{70},\alpha^{140},\alpha^{280},\alpha^{560},\alpha^{1120},\alpha^{2240},\alpha^{4480},\alpha^{769},\alpha^{1538},\alpha^{3076},\alpha^{66152},\alpha^{4113}\}$
18. $\{\alpha^{37},\alpha^{74},\alpha^{148},\alpha^{296},\alpha^{592},\alpha^{1184},\alpha^{2368},\alpha^{4732},\alpha^{1281},\alpha^{2562},\alpha^{5124},\alpha^{2057},\alpha^{4114}\}$
19. $\{\alpha^{39},\alpha^{78},\alpha^{156},\alpha^{312},\alpha^{624},\alpha^{1248},\alpha^{2496},\alpha^{4992},\alpha^{1793},\alpha^{3586},\alpha^{7172},\alpha^{6153},\alpha^{4115}\}$
20. $\{\alpha^{41},\alpha^{82},\alpha^{164},\alpha^{328},\alpha^{656},\alpha^{1312},\alpha^{2624},\alpha^{5248},\alpha^{2305},\alpha^{4610},\alpha^{1029},\alpha^{2058},\alpha^{4116}\}$
21. $\{\alpha^{43},\alpha^{86},\alpha^{172},\alpha^{344},\alpha^{688},\alpha^{1376},\alpha^{2752},\alpha^{5504},\alpha^{2817},\alpha^{5634},\alpha^{3077},\alpha^{6154},\alpha^{4117}\}$

The minimal polynomials are shown in equation (3.4). The other minimal polynomials obtained are:

$$
\begin{aligned}
M_{21}(x) &= 1+x+x^4+x^6+x^7+x^8+x^{11}+x^{12}+x^{13}\\
M_{23}(x) &= 1+x+x^2+x^5+x^{11}+x^{12}+x^{13}\\
M_{25}(x) &= 1+x^2+x^3+x^4+x^6+x^8+x^{10}+x^{12}+x^{13}\\
M_{27}(x) &= 1+x^2+x^4+x^8+x^9+x^{12}+x^{13}\\
M_{29}(x) &= 1+x^2+x^6+x^8+x^9+x^{10}+x^{11}+x^{12}+x^{13}\\
M_{31}(x) &= 1+x^2+x^3+x^6+x^8+x^{10}+x^{11}+x^{12}+x^{13}\\
M_{33}(x) &= 1+x+x^2+x^3+x^4+x^5+x^{10}+x^{11}+x^{13}\\
M_{35}(x) &= 1+x+x^3+x^4+x^5+x^7+x^8+x^9+x^{11}+x^{12}+x^{13}\\
M_{37}(x) &= 1+x^2+x^3+x^4+x^7+x^8+x^{13}
\end{aligned}
\qquad (3.30\text{ a})
$$

$$M_{39}(x) = 1 + x + x^4 + x^5 + x^6 + x^7 + x^8 + x^9 + x^{10} + x^{12} + x^{13}$$
$$M_{41}(x) = 1 + x^2 + x^5 + x^6 + x^7 + x^8 + x^9 + x^{11} + x^{13} \qquad (3.30\text{ b})$$
$$M_{43}(x) = 1 + x^2 + x^3 + x^4 + x^5 + x^6 + x^8 + x^{10} + x^{11} + x^{12} + x^{13}$$

The generator polynomial for codes designed by considering memory model 2 is computed by evaluating the *LCM* of minimal polynomials corresponding to the required roots

(i) For $t = 15$

$$g(x) = LCM\left\{\begin{matrix} M_1(x), M_3(x), M_5(x), M_7(x), M_9(x), M_{11}(x), M_{13}(x), M_{15}(x), \\ M_{17}(x), M_{19}(x), M_{21}(x), M_{23}(x), M_{25}(x), M_{27}(x), M_{29}(x) \end{matrix}\right\} \qquad (3.31)$$

Therefore,

$$\begin{aligned} g(x) = 1 &+ x^3 + x^4 + x^5 + x^8 + x^{10} + x^{11} + x^{13} + x^{15} + x^{17} + x^{19} + x^{24} + x^{28} + x^{29} + \\ &x^{30} + x^{31} + x^{32} + x^{34} + x^{35} + x^{37} + x^{38} + x^{39} + x^{40} + x^{41} + x^{44} + x^{45} + x^{47} + x^{49} + \\ &x^{50} + x^{53} + x^{55} + x^{56} + x^{60} + x^{61} + x^{62} + x^{63} + x^{64} + x^{66} + x^{68} + x^{72} + x^{74} + x^{76} + \\ &x^{78} + x^{79} + x^{81} + x^{83} + x^{84} + x^{86} + x^{90} + x^{92} + x^{95} + x^{96} + x^{100} + x^{101} + x^{102} + x^{103} + \\ &x^{104} + x^{105} + x^{107} + x^{109} + x^{110} + x^{113} + x^{114} + x^{120} + x^{122} + x^{123} + x^{125} + x^{127} + x^{132} + \\ &x^{134} + x^{135} + x^{139} + x^{141} + x^{143} + x^{144} + x^{146} + x^{151} + x^{157} + x^{159} + x^{165} + x^{167} + x^{168} + \\ &x^{169} + x^{173} + x^{174} + x^{175} + x^{176} + x^{178} + x^{181} + x^{182} + x^{183} + x^{184} + x^{185} + x^{186} + x^{188} + \\ &x^{189} + x^{190} + x^{192} + x^{194} + x^{195} \end{aligned} \qquad (3.32)$$

In this case *degree* $\{g(x)\}$ =195 = $n - k$.

$n = k + 195 = 4096 + 195 = 4291$ bits.

$\therefore$ Primitive code parameter over $F_{2^{13}}$: (8191, 8191 – 195) = (8191, 7996)

k = 4096 $\Rightarrow$ 7996 – 4096 =3900.

$\therefore$ The shortened BCH code has the parameter:

(8191 – 3900, 7996 – 3900) = (4291, 4096)

Since *degree* $\{g(x)\}$ < $(n - k)$, a $t = 15$ error correcting BCH code can be supported by this memory model.

(ii) For $t = 16$

$$g(x) = LCM\begin{Bmatrix} M_1(x), M_3(x), M_5(x), M_7(x), M_9(x), M_{11}(x), M_{13}(x), M_{15}(x), \\ M_{17}(x), M_{19}(x), M_{21}(x), M_{23}(x), M_{25}(x), M_{27}(x), M_{29}(x), M_{31}(x) \end{Bmatrix} \quad (3.33)$$

$$\begin{aligned} g(x) = 1 + x^4 + x^6 + x^7 + x^8 + x^{10} + x^{11} + x^{12} + x^{13} + x^{14} + x^{15} + x^{20} + x^{22} + x^{23} + \\ x^{24} + x^{26} + x^{27} + x^{30} + x^{31} + x^{32} + x^{41} + x^{42} + x^{45} + x^{47} + x^{51} + x^{52} + x^{53} + x^{54} + \\ x^{56} + x^{57} + x^{60} + x^{61} + x^{63} + x^{64} + x^{65} + x^{66} + x^{67} + x^{69} + x^{70} + x^{73} + x^{80} + x^{81} + \\ x^{82} + x^{83} + x^{84} + x^{85} + x^{86} + x^{87} + x^{88} + x^{90} + x^{94} + x^{97} + x^{99} + x^{101} + x^{103} + x^{104} + \\ x^{105} + x^{106} + x^{108} + x^{109} + x^{110} + x^{111} + x^{112} + x^{113} + x^{114} + x^{120} + x^{121} + x^{122} + x^{123} + \\ x^{124} + x^{125} + x^{126} + x^{127} + x^{133} + x^{136} + x^{137} + x^{139} + x^{140} + x^{141} + x^{142} + x^{143} + x^{144} + \\ x^{146} + x^{148} + x^{149} + x^{151} + x^{152} + x^{153} + x^{157} + x^{158} + x^{160} + x^{162} + x^{166} + x^{167} + x^{169} + \\ x^{170} + x^{171} + x^{172} + x^{173} + x^{175} + x^{176} + x^{178} + x^{179} + x^{184} + x^{185} + x^{186} + x^{187} + x^{188} + \\ x^{189} + x^{193} + x^{194} + x^{195} + x^{196} + x^{197} + x^{199} + x^{200} + x^{201} + x^{203} + x^{206} + x^{207} + x^{208} \end{aligned} \quad (3.34)$$

In this case *degree* $\{ g(x) \} = 208 = n - k$. $\therefore n = k + 208 = 4096 + 208 = 4304$.

$\therefore$ Primitive code parameter over $F_{2^{13}}$: (8191, 8191 – 208) = (8191, 7983).

$k = 4096 \quad \Rightarrow \quad 7983 - 4096 = 3887$.

$\therefore$ The shortened BCH code has the parameters:

$$(8191 - 3887, 7983 - 3887) = (4304, 4096)$$

Since *degree* $\{g(x)\} < (n - k)$, a $t = 16$ error correcting BCH code can be supported by this model.

(iii) For $t = 17$

$$g(x) = LCM\begin{Bmatrix} M_1(x), M_3(x), M_5(x), M_7(x), M_9(x), M_{11}(x), M_{13}(x), \\ M_{15}(x), M_{17}(x), M_{19}(x), M_{21}(x), M_{23}(x), M_{25}(x), \\ M_{27}(x), M_{29}(x), M_{31}(x), M_{33}(x) \end{Bmatrix} \quad (3.37)$$

$$g(x) = 1 + x + x^2 + x^3 + x^7 + x^{10} + x^{12} + x^{15} + x^{17} + x^{18} + x^{24} + x^{25} + x^{28} + x^{29} +$$
$$x^{31} + x^{32} + x^{34} + x^{35} + x^{38} + x^{39} + x^{41} + x^{44} + x^{46} + x^{47} + x^{51} + x^{52} + x^{54} + x^{59} +$$
$$x^{61} + x^{62} + x^{63} + x^{64} + x^{65} + x^{66} + x^{70} + x^{73} + x^{78} + x^{81} + x^{86} + x^{89} + x^{92} + x^{93} +$$
$$x^{94} + x^{95} + x^{97} + x^{98} + x^{99} + x^{101} + x^{102} + x^{104} + x^{105} + x^{106} + x^{110} + x^{112} + x^{113} +$$
$$x^{114} + x^{115} + x^{116} + x^{117} + x^{120} + x^{121} + x^{123} + x^{126} + x^{127} + x^{128} + x^{132} + x^{136} + x^{138} + \quad (3.36)$$
$$x^{140} + x^{141} + x^{142} + x^{144} + x^{145} + x^{146} + x^{148} + x^{152} + x^{154} + x^{155} + x^{156} + x^{157} + x^{159} +$$
$$x^{161} + x^{162} + x^{163} + x^{165} + x^{166} + x^{171} + x^{173} + x^{174} + x^{178} + x^{180} + x^{181} + x^{182} + x^{183} +$$
$$x^{187} + x^{188} + x^{189} + x^{191} + x^{193} + x^{195} + x^{200} + x^{202} + x^{203} + x^{205} + x^{209} + x^{211} + x^{213} +$$
$$x^{220} + x^{221}$$

In this case *degree* $\{g(x)\}$ =221 = $n - k$.

$n = k + 221 = 4096 + 221 = 4317$.

$\therefore$ Primitive code parameter over $F_{2^{13}}$: (8191, 8191 – 221) = (8191, 7970)

$k = 4096 \quad \Rightarrow \quad 7970 - 4096 = 3874$

$\therefore$ The shortened BCH code has the parameter:

(8191 – 3874, 7970 – 3874) = (4317, 4096).

Since *degree* $\{ g(x) \} < (n - k)$, a $t = 17$ error correcting BCH code can be supported by this model.

(iv) For $t = 18$

$$g(x) = LCM\left\{\begin{matrix} M_1(x), M_3(x), M_5(x), M_7(x), M_9(x), M_{11}(x), M_{13}(x), \\ M_{15}(x), M_{17}(x), M_{19}(x), M_{21}(x), M_{23}(x), M_{25}(x), M_{27}(x), \\ M_{29}(x), M_{31}(x), M_{33}(x), M_{35}(x) \end{matrix}\right\} \quad (3.37)$$

$$g(x) = 1 + x^3 + x^4 + x^5 + x^6 + x^9 + x^{10} + x^{11} + x^{12} + x^{13} + x^{14} + x^{17} +$$
$$x^{18} + x^{19} + x^{25} + x^{27} + x^{28} + x^{33} + x^{36} + x^{39} + x^{42} + x^{45} + x^{47} + x^{48} +$$
$$x^{50} + x^{54} + x^{61} + x^{62} + x^{66} + x^{68} + x^{69} + x^{70} + x^{72} + x^{73} + x^{74} + x^{75} +$$
$$x^{77} + x^{78} + x^{79} + x^{80} + x^{85} + x^{88} + x^{92} + x^{93} + x^{94} + x^{98} + x^{101} + x^{102} +$$
$$x^{104} + x^{105} + x^{107} + x^{109} + x^{112} + x^{114} + x^{115} + x^{117} + x^{118} + x^{123} + x^{125} +$$
$$x^{126} + x^{127} + x^{130} + x^{131} + x^{133} + x^{135} + x^{136} + x^{138} + x^{140} + x^{142} + x^{143} +$$
$$x^{144} + x^{146} + x^{149} + x^{150} + x^{154} + x^{155} + x^{158} + x^{159} + x^{163} + x^{164} + x^{165} +$$
$$x^{166} + x^{167} + x^{168} + x^{170} + x^{171} + x^{172} + x^{173} + x^{175} + x^{180} + x^{181} + x^{184} +$$
$$x^{188} + x^{191} + x^{192} + x^{194} + x^{196} + x^{197} + x^{198} + x^{200} + x^{202} + x^{203} + x^{204} +$$
$$x^{206} + x^{212} + x^{213} + x^{215} + x^{216} + x^{217} + x^{219} + x^{225} + x^{227} + x^{230} + x^{231} + x^{234} \quad (3.38)$$

Here *degree* $\{g(x)\}$ = 234 = $n - k$. $\therefore$ $n = k + 234 = 4096 + 234 = 4330$.

Primitive code parameter over $F_{2^{13}}$: (8191, 8191 – 234) = (8191, 7957)

k = 4096. $\Rightarrow$ 7957 – 4096 =3861

$\therefore$ The shortened BCH code has the parameter:

(8191 – 3861, 7957 – 3861) = (4330, 4096).

Since *degree* $\{g(x)\}$ < ($n - k$), a t = 18 error correcting BCH code can be supported by this memory model.

(v) For t = 19

$$g(x) = LCM\left\{\begin{matrix} M_1(x), M_3(x), M_5(x), M_7(x), M_9(x), M_{11}(x), M_{13}(x), \\ M_{15}(x), M_{17}(x), M_{19}(x), M_{21}(x), M_{23}(x), M_{25}(x), \\ M_{27}(x), M_{29}(x), M_{31}(x), M_{33}(x), M_{35}(x), M_{37}(x) \end{matrix}\right\} \quad (3.39)$$

$$g(x) = 1 + x^2 + x^6 + x^9 + x^{10} + x^{12} + x^{13} + x^{14} + x^{15} + x^{16} + x^{18} + x^{19} + x^{21} + x^{26} +$$
$$x^{30} + x^{31} + x^{32} + x^{34} + x^{35} + x^{36} + x^{37} + x^{40} + x^{41} + x^{46} + x^{47} + x^{54} + x^{55} + x^{58} +$$
$$x^{60} + x^{61} + x^{67} + x^{68} + x^{74} + x^{75} + x^{76} + x^{79} + x^{80} + x^{86} + x^{89} + x^{93} + x^{95} + x^{98} +$$
$$x^{99} + x^{100} + x^{101} + x^{102} + x^{103} + x^{104} + x^{105} + x^{108} + x^{109} + x^{116} + x^{117} + x^{119} + x^{121} +$$
$$x^{124} + x^{125} + x^{126} + x^{128} + x^{129} + x^{130} + x^{133} + x^{136} + x^{137} + x^{140} + x^{143} + x^{144} + x^{145} + \quad (3.40)$$
$$x^{147} + x^{152} + x^{154} + x^{156} + x^{157} + x^{159} + x^{160} + x^{161} + x^{162} + x^{164} + x^{165} + x^{166} + x^{168} +$$
$$x^{169} + x^{170} + x^{173} + x^{174} + x^{181} + x^{188} + x^{189} + x^{190} + x^{191} + x^{192} + x^{195} + x^{197} + x^{200} +$$
$$x^{203} + x^{205} + x^{206} + x^{208} + x^{209} + x^{211} + x^{213} + x^{221} + x^{223} + x^{225} + x^{227} + x^{229} + x^{230} +$$
$$x^{232} + x^{233} + x^{236} + x^{239} + x^{240} + x^{241} + x^{242} + x^{243} + x^{244} + x^{247}$$

In this case *degree* $\{g(x)\}$ = 247 = $n - k$

$\therefore$ $n = k + 247 = 4096 + 247 = 4343$

$\therefore$ Primitive code parameter over $F_{2^{13}}$: (8191, 8191 – 247) = (8191, 7944)

$k = 4096 \quad \Rightarrow \quad 7944 - 4096 = 3848$

$\therefore$ The shortened BCH code has the parameter:

(8191 – 3848, 7944 – 3848) = (4343, 4096)

Since *degree* $\{g(x)\} < (n - k)$, a t = 19 error correcting BCH code can be supported by this model.

(vi) For t = 20

$$g(x) = LCM\left\{\begin{matrix} M_1(x), M_3(x), M_5(x), M_7(x), M_9(x), M_{11}(x), M_{13}(x), \\ M_{15}(x), M_{17}(x), M_{19}(x), M_{21}(x), M_{23}(x), M_{25}(x), \\ M_{27}(x), M_{29}(x), M_{31}(x), M_{33}(x), M_{35}(x), M_{37}(x), M_{39}(x) \end{matrix}\right\} \quad (3.41)$$

$$g(x) = 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 + x^7 + x^9 + x^{10} + x^{11} + x^{13} + x^{17} + x^{18} +$$
$$x^{19} + x^{20} + x^{21} + x^{22} + x^{23} + x^{25} + x^{26} + x^{28} + x^{29} + x^{33} + x^{35} + x^{37} + x^{40} +$$
$$x^{42} + x^{44} + x^{47} + x^{48} + x^{51} + x^{52} + x^{56} + x^{57} + x^{58} + x^{59} + x^{63} + x^{69} + x^{70} +$$
$$x^{71} + x^{72} + x^{77} + x^{80} + x^{81} + x^{82} + x^{84} + x^{86} + x^{87} + x^{90} + x^{92} + x^{94} + x^{95} +$$
$$x^{96} + x^{101} + x^{106} + x^{107} + x^{108} + x^{109} + x^{111} + x^{112} + x^{113} + x^{115} + x^{116} + x^{120} +$$
$$x^{121} + x^{123} + x^{128} + x^{130} + x^{132} + x^{133} + x^{135} + x^{138} + x^{140} + x^{142} + x^{143} + x^{144} +$$
$$x^{146} + x^{148} + x^{150} + x^{156} + x^{163} + x^{164} + x^{166} + x^{167} + x^{168} + x^{170} + x^{171} + x^{172} +$$
$$x^{175} + x^{176} + x^{178} + x^{179} + x^{181} + x^{182} + x^{183} + x^{184} + x^{186} + x^{189} + x^{190} + x^{191} +$$
$$x^{192} + x^{195} + x^{199} + x^{201} + x^{202} + x^{203} + x^{205} + x^{207} + x^{213} + x^{214} + x^{216} + x^{217} +$$
$$x^{220} + x^{221} + x^{222} + x^{223} + x^{225} + x^{226} + x^{227} + x^{228} + x^{230} + x^{231} + x^{233} + x^{235} +$$
$$x^{237} + x^{238} + x^{240} + x^{242} + x^{245} + x^{249} + x^{250} + x^{253} + x^{255} + x^{256} + x^{259} + x^{260} \quad (3.42)$$

Here *degree* $\{g(x)\}$ = 260 = $n - k$.

$\therefore\ n = k + 260 = 4096 + 260 = 4356$

$\therefore$ Primitive code parameter over $F_{2^{13}}$: (8191, 8191 – 260) = (8191, 7931)

k = 4096 $\Rightarrow$ 7931 – 4096 =3835

Hence, the shortened BCH code has the parameter are:

(8191 – 3835, 7931 – 3835) = (4356, 4096)

Since *degree* $\{g(x)\} > (n - k)$, a t = 20 error correcting BCH code cannot be supported by this memory model. However, if overhead space is increased to 33 bytes, this code can still be used. Further, we have also worked out the generator polynomials of BCH codes capable of correcting $t = 21$and 22 errors per 512 bytes which can be used if memories with higher data integrity are required. However, this will necessitate the use of a greater overhead allocation for storing redundant bits generated by the code.

(vii) For t = 21

$$g(x) = LCM\left\{\begin{matrix} M_1(x), M_3(x), M_5(x), M_7(x), M_9(x), M_{11}(x), M_{13}(x), M_{15}(x), \\ M_{17}(x), M_{19}(x), M_{21}(x), M_{23}(x), M_{25}(x), M_{27}(x), M_{29}(x), M_{31}(x), \\ M_{33}(x), M_{35}(x), M_{37}(x), M_{39}(x), M_{41}(x) \end{matrix}\right\} \quad (3.43)$$

$$g(x) = 1 + x + x^5 + x^7 + x^8 + x^9 + x^{12} + x^{15} + x^{18} + x^{23} + x^{26} + x^{30} + x^{31} + x^{32} +$$
$$x^{37} + x^{38} + x^{41} + x^{44} + x^{46} + x^{47} + x^{48} + x^{52} + x^{54} + x^{57} + x^{58} + x^{59} + x^{62} + x^{63} +$$
$$x^{64} + x^{66} + x^{67} + x^{68} + x^{73} + x^{74} + x^{77} + x^{81} + x^{82} + x^{85} + x^{86} + x^{88} + x^{89} + x^{90} +$$
$$x^{93} + x^{94} + x^{97} + x^{102} + x^{105} + x^{106} + x^{111} + x^{113} + x^{116} + x^{119} + x^{120} + x^{123} + x^{127} +$$
$$x^{128} + x^{133} + x^{136} + x^{140} + x^{141} + x^{143} + x^{144} + x^{147} + x^{149} + x^{150} + x^{153} + x^{154} + x^{155} +$$
$$x^{159} + x^{161} + x^{162} + x^{163} + x^{168} + x^{173} + x^{174} + x^{177} + x^{180} + x^{181} + x^{183} + x^{184} + x^{185} +$$
$$x^{187} + x^{189} + x^{191} + x^{192} + x^{194} + x^{197} + x^{199} + x^{201} + x^{203} + x^{206} + x^{207} + x^{208} + x^{210} +$$
$$x^{211} + x^{212} + x^{214} + x^{215} + x^{216} + x^{217} + x^{219} + x^{223} + x^{226} + x^{227} + x^{228} + x^{230} + x^{235} +$$
$$x^{246} + x^{251} + x^{255} + x^{257} + x^{258} + x^{259} + x^{261} + x^{262} + x^{263} + x^{265} +$$
$$x^{267} + x^{268} + x^{270} + x^{271} + x^{272} + x^{273} \qquad (3.44)$$

Here *degree* $\{g(x)\}$ = 273 = $n - k$. $\therefore$ $n = k + 273 = 4096 + 273 = 4369$

$\therefore$ Primitive code parameter over $F_{2^{13}}$: (8191, 8191 – 273) = (8191, 7918)

k = 4096 $\Rightarrow$ 7918 – 4096 = 3822

$\therefore$ The shortened BCH code has the parameter:

(8191 – 3822, 7918 – 3822) = (4369, 4096)

(viii) For t = 22

$$g(x) = LCM\left\{\begin{matrix} M_1(x), M_3(x), M_5(x), M_7(x), M_9(x), M_{11}(x), M_{13}(x), M_{15}(x), \\ M_{17}(x), M_{19}(x), M_{21}(x), M_{23}(x), M_{25}(x), M_{27}(x), M_{29}(x), \\ M_{31}(x), M_{33}(x), M_{35}(x), M_{37}(x), M_{39}(x), M_{41}(x), M_{43}(x) \end{matrix}\right\} \qquad (3.45)$$

$$g(x) = 1 + x + x^2 + x^5 + x^7 + x^8 + x^{15} + x^{16} + x^{17} + x^{22} + x^{25} + x^{29} +$$
$$x^{30} + x^{32} + x^{33} + x^{34} + x^{36} + x^{39} + x^{41} + x^{42} + x^{44} + x^{45} + x^{48} + x^{51} +$$
$$x^{53} + x^{55} + x^{56} + x^{63} + x^{67} + x^{68} + x^{69} + x^{73} + x^{75} + x^{78} + x^{82} + x^{83} +$$
$$x^{86} + x^{87} + x^{88} + x^{89} + x^{91} + x^{93} + x^{94} + x^{95} + x^{100} + x^{101} + x^{102} + x^{103} +$$
$$x^{104} + x^{105} + x^{109} + x^{111} + x^{114} + x^{116} + x^{119} + x^{123} + x^{124} + x^{131} + x^{133} + x^{135} +$$
$$x^{144} + x^{147} + x^{151} + x^{153} + x^{155} + x^{156} + x^{158} + x^{160} + x^{162} + x^{164} + x^{165} + x^{169} + \quad (3.46)$$
$$x^{170} + x^{173} + x^{174} + x^{175} + x^{177} + x^{1778} + x^{182} + x^{183} + x^{184} + x^{185} + x^{187} + x^{188} +$$
$$x^{189} + x^{190} + x^{191} + x^{193} + x^{194} + x^{195} + x^{200} + x^{201} + x^{204} + x^{211} + x^{212} + x^{217} +$$
$$x^{219} + x^{221} + x^{222} + x^{223} + x^{224} + x^{226} + x^{227} + x^{231} + x^{232} + x^{237} + x^{238} + x^{241} +$$
$$x^{242} + x^{245} + x^{247} + x^{249} + x^{250} + x^{252} + x^{253} + x^{258} + x^{259} + x^{260} + x^{262} + x^{263} +$$
$$x^{264} + x^{265} + x^{266} + x^{268} + x^{269} + x^{272} + x^{273} + x^{274} + x^{276} + x^{277} + x^{282} + x^{284} + x^{286}$$

Now, for t =22, $degree\ \{g(x)\}$ = 286 = $n - k$.

$\therefore$ $n = k + 286 = 4096 + 286 = 4382$.

$\therefore$ Primitive code parameter over $F_{2^{13}}$: (8191, 8191 – 286) = (8191, 7905)

$k = 4096 \quad \Rightarrow \quad 7905 - 4096 = 3809$

$\therefore$ The shortened BCH code has the parameter: (8191 – 3809, 7905 – 3809)

= (4382, 4096)

Table 3.3. Number of overhead bits required for various values of t (Memory model 2)

T	**Number of overhead bits required (*degree{g(x)}*)**	**Shortened BCH code parameter**
15	195	(4291, 4096)
16	208	(4204, 4096)
17	221	(4317, 4096)
18	234	(4330, 4096)
19	247	(4343, 4096)
20	260	(4356, 4096)
21	273	(4369, 4096)
22	286	(4382, 4096)

Table 3.3 shows the summary of the results obtained after synthesizing the codes by considering memory model 2. The performance of these codes is shown in Figure 3.2.

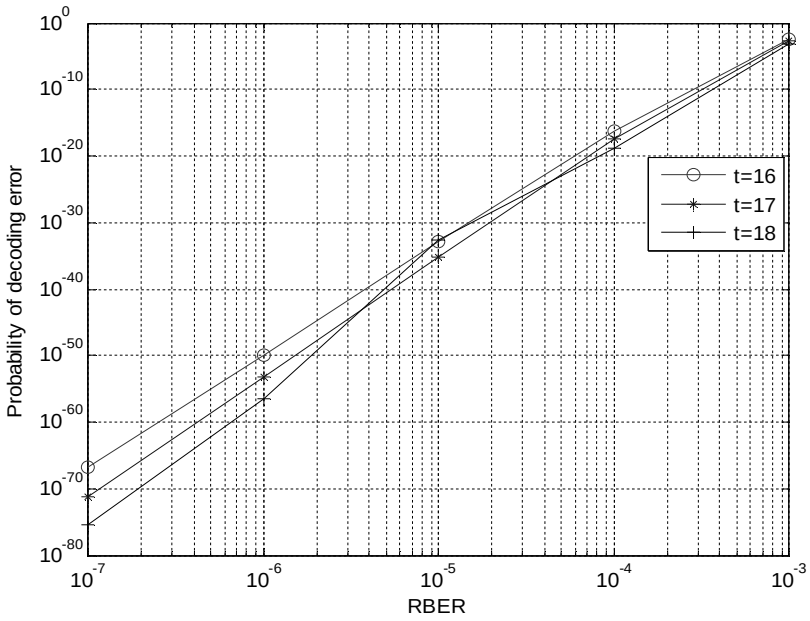


Figure 3.2: Performance of BCH Codes for enhancing Data Integrity in Flash memories for Memory model 2

3.3 Synthesis of RS Code for Memory Model 1

(i) Sector size = 512 bytes, Overhead = 16 bytes

Consider the architecture of memory model 1, where, each sector is further partitioned into four subsectors, each with 128 bytes of information and 4 bytes of overhead.

128 bytes	4 bytes
128 bytes	4 bytes
128 bytes	4 bytes
128 bytes	4 bytes

Natural length of a RS code over F_{2^8}, $n = 2^8 - 1$.

$n - k = 4$ bytes. This leads to the selection of (255, 251) RS code over F_{2^8} .

$$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor = \left\lfloor \frac{n-k}{2} \right\rfloor = 2$$

For *t* = 2, we have 8 bit symbols. Therefore the shortened RS code over F_{2^8} is (132, 128).

$\delta = 2t + 1 = 5, \qquad b = 1, \qquad b + \delta - 2 = 4.$

So the required roots are $\{\alpha, \alpha^2, ..., \alpha^{b+\delta-2}\} = \{\alpha, \alpha^2, \alpha^3, \alpha^4\}$

$$\begin{aligned} \therefore g(x) &= (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4) \\ &= x^4 + \alpha^{30}x^3 + \alpha^{216}x^2 + \alpha^{231}x + \alpha^{116} \end{aligned} \qquad (3.47)$$

This is of the form $g(x) = x^4 + g_3x^3 + g_2x^2 + g_1x + 1$

Since RS codes are cyclic codes, encoding in systematic form is analogous to the binary encoding procedure. One can think of shifting a message polynomial, *m(x)*, into the rightmost *k* stages of a codeword register and then appending a parity polynomial, *p(x)*, by placing it in the leftmost $n - k$ stages. So the steps are:

(i) Compute $x^{n-k}m(x)$.

(ii) Divide $x^{n-k}m(x)$ by the generator polynomial $g(x)$ and compute the remainder.

i.e. $x^{n-k}m(x) = q(x)g(x) + r(x)$,

where $q(x)$ is the quotient and $r(x)$ is the remainder.

(iii) Compute $x^{n-k}m(x) - r(x)$.

(ii) Another type of partition can also be thought of. In this case, the sector is partitioned into two sub-sectors, each containing 256 bytes of information and 8 bytes of overhead. This is illustrated below.

256 bytes = 2048 bits	8 bytes = 64 bits
256 bytes = 2048 bits	8 bytes = 64 bits

In this solution, it should be noted that each symbol used in the RS code is 9 bits wide. Each information block consists of 228 nine bit symbols. The first 228 symbols comprise of user data. The last symbol has five bits of user data to which four additional bits are padded. The original RS code over F_{2^9} has length $n = 2^9 - 1 = 511$. Therefore the primitive length RS code has parameters (511, 504).

$d_{\min} = 235 - 228 + 1 = 8$ nine bit symbols.

$t = \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor = 3$ nine bit symbols. $\quad \delta = 2t + 1 = 7, \quad b + \delta - 2 = 6$

So the required roots are $\{\alpha, \alpha^2, ...\alpha^{b+\delta-2}\} = \{\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6\}$

$$\therefore g(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4)(x-\alpha^5)(x-\alpha^6)$$

$$= x^7 + \alpha^{254}x^6 + \alpha^{17}x^5 + \alpha^{178}x^4 + \alpha^{373}x^3 + \alpha^{76}x^2 + \alpha^{298}x + \alpha^{291} \qquad (3.48)$$

This is of the form $x^7 + g_6x^6 + g_5x^5 + g_4x^4 + g_3x^3 + g_2x^2 + g_1x + g_0$

This is shortened to (235, 228).

3.4 Synthesis of RS code for Memory Model 2

128 bytes	8 bytes
128 bytes	8 bytes
128 bytes	8 bytes
128 bytes	8 bytes

In memory model 2, k = 512 bytes and overhead $(n-k)$ = 32 bytes. So if each sector is partitioned into four subsectors, then each subsector will have $k = 128$ bytes and $(n-k) = 8$ bytes.

$n-k = 8, \qquad k = 128, \qquad \therefore n = 128+8 = 136$ *bytes*

$n-k+1 = 9, \qquad t = \left\lfloor \frac{n-k}{2} \right\rfloor = 4.$

The original RS code over F_{2^8} has length $n' = 2^8 - 1 = 255$. Hence the code has the parameters (255, 247). The shortened RS code over F_{2^8} has parameters (136, 128).

Design distance $\delta = 2t+1 = 9, \qquad b+\delta-2 = 8$

Required roots are: $\{\alpha, \alpha^2, ..., \alpha^{b+\delta-2}\} = \{\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6, \alpha^7, \alpha^8\}$

$$\therefore g(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4)(x-\alpha^5)(x-\alpha^6)(x-\alpha^7)(x-\alpha^8)$$
$$= x^8 + \alpha^{227}x^7 + \alpha^{44}x^6 + \alpha^{178}x^5 + \alpha^{71}x^4 + \alpha^{172}x^3 + \alpha^8 x^2 + \alpha^{224}x + \alpha^{37} \qquad (3.49)$$

This is of the form $x^8 + g_7x^7 + g_6x^6 + g_5x^5 + g_4x^4 + g_3x^3 + g_2x^2 + g\ x + \alpha^{37}$

Thus, this chapter has been devoted to the synthesis of various BCH and RS codes which can be employed to protect information integrity in Single level Cell (SLC) based Flash memories. In the next chapter, we will focus our attention to the synthesis of codes which can be used to protect information integrity on Multi Level Cell (MLC) based Flash memories.

This page is intentionally left blank

Chapter 4

Application of Error Correcting Codes to Multi Level Cells

4.1 Introduction

In recent years, flash memory has been increasingly used in many embedded systems. This is because the performance and capacity of NAND flash memory has improved over the years. Hard disk drive (HDD) based storage devices can save large amounts of data at low price, but they have disadvantages such as large size, lesser durability, high power consumption and long response time. Flash memory has been appropriated for small portable devices because it overcomes these disadvantages. However, Flash devices have not achieved the reliability of their HDD counterparts. Attempts have been made by many researchers to address the reliability problem by incorporating powerful ECC algorithms [Agarwal, A. et al. 2005], [Slayman, C.W. 2005], [Bajura, M et al. 2007]. In this chapter, we have proposed a few solutions to mitigate this problem.

MLC flash memories have replaced SLC flash memories in some applications in recent years. Because multiple bits are stored per memory cell in MLC flash memory, there is an improvement in the storage density. Unfortunately, this is accompanied by an increase in the probability of error. This means as the number of distinct levels that can be stored in a cell increases, one can expect an increase in the RBER. Hence powerful ECC algorithms that can detect and correct these error patterns have to be developed and deployed if MLC devices are to find widespread acceptability.

Details of flash memory organization and its structure and characteristics have been discussed in Chapter 1. As mentioned there, conventional SLC can hold two distinct levels of charge, and hence it can store one bit of data in each memory cell, while a MLC can reliably hold $Q = 2^b$ levels of charge, and hence it can store b bits of data, where b is typically 2 or 3 [Rossi, D. and Metra, C. 2003], [Rossi, D. et al. 2001], [Sun, F.et al.

2007], [Sun, F. et al. 2006]. Although MLC memory has higher density than SLC memory, MLC is more vulnerable to errors because small fluctuation of the charge amount in the floating gate and slight variation of gate voltage result in misreading of stored data. Multilevel flash memory cells have found application in efforts to increase density of bits per unit area [Lin, H. et al. 2002] in recent years.

4.2 Modeling of Multi Level Cell as a Channel

In this section, we describe the modeling of MLC cell as a channel. This will allow us to quantify the performance of these codes and measure the degree of improvement in data integrity provided by the use of these codes. In MLC memory devices, *b*-bits of data are stored in a single cell by designing the cell to hold $Q = 2^b$ distinct levels of charge in the floating gate. The amount of charge residing on the floating gate is identified by observing the relation between the control gate voltage, V_{CG} and the current value from the drain to source, I_D. Figure 4.1 [Maeda, Y. et al. 2009] depicts schematic relation between the control gate voltage and the drain-source current I_D where $Q = 2^2 = 4$. During the read operation, a gradually increasing gate to source voltage (V_{CG}) is applied and the drain to source current (I_D) is monitored. When the value of I_D just exceeds I_{TH}, further increment in V_{CG} is stopped. This value of V_{CG} denotes the read out voltage V_{R_i}. It can be inferred from Figure 4.1 that as the charge level in the floating gate is increased, larger control gate voltages (V_{CG}) have to be applied before significant drain-source current can flow. As can be seen from the Figure 4.1, if the charge level in the floating gate corresponds to level 3, then the drain-source current (I_D) exceeds the threshold value (I_{TH}) only when $V_{CG} \geq V_{TH3}$. The charge amount levels in *Q*-level cell are expressed by integers {0,1,……., *Q*-1}. For a charge amount level *i*, threshold voltage V_{THi}, is determined as the minimum control gate voltage V_{CG} which gives the drain-source current I_D greater than I_{TH}. From Figure 4.1, it is apparent that $V_{TH0} < V_{TH1} < \cdots\cdots < V_{THQ-1}$. I_{TH} is a predetermined threshold drain-source current. In a *Q*-level MLC the distinct symbols that can be stored are 0,1,2,….., *Q*-1. During the read process, the

mapping between the readout voltage V_{R_i} and the symbol present in the location being accessed can be specified as in Table 4.1.

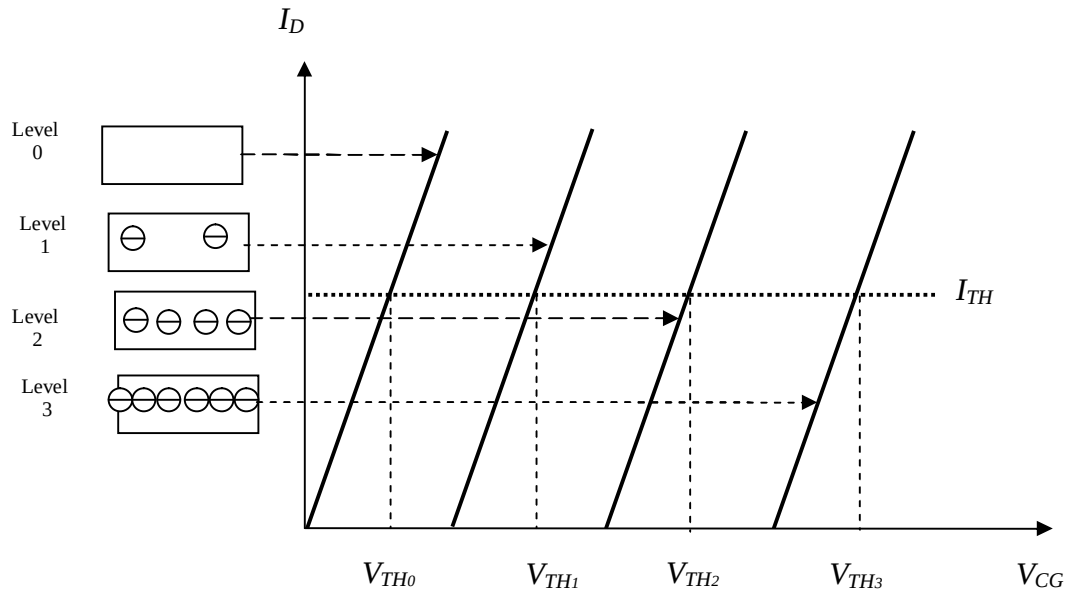


Figure 4.1: Schematic relation between control voltage and drain-source current in 4-level memory cell.

Table 4.1: Mapping between the readout voltage V_{R_i} and the symbol stored in MLC

Range of V_{R_i}	Symbols
$0 \le V_{R_i} \le V_{TH_0}$	0
$V_{TH_0} \le V_{R_i} \le V_{TH_1}$	1
$V_{TH_1} \le V_{R_i} \le V_{TH_2}$	2
⋮	⋮
$V_{TH_{Q-1}} \le V_{R_i} \le V_{TH_Q}$	Q-1

The probability distribution of the readout voltages V_{R_i} is usually approximated by Gaussian distribution [Maeda, Y. and Kaneko, H. 2009], [Sun, F. et al. 2006], [Sun, F.et al. 2007] whose probability density function is defined as follows:

$$p_i(v_{R_i}) = \frac{1}{\sqrt{2\pi\sigma_i^2}} e^{-\frac{(v_{ri}-\mu_i)^2}{2\sigma_i^2}} \tag{4.1}$$

Here, μ_i and σ_i are the mean and standard deviation of V_{THi} respectively. v_{R_i} represents the sample value of the readout voltage V_{R_i}. The variance σ_i (which is a measure of the width of the Gaussian curve in Figure 4.2) is an indication of the likelihood of the stored data being read erroneously. A large value of σ_i indicates that the probability of reading back a symbol that is different from that which was written into that location is relatively large. Thus, a large value of σ gives rise to a large value of RSER.

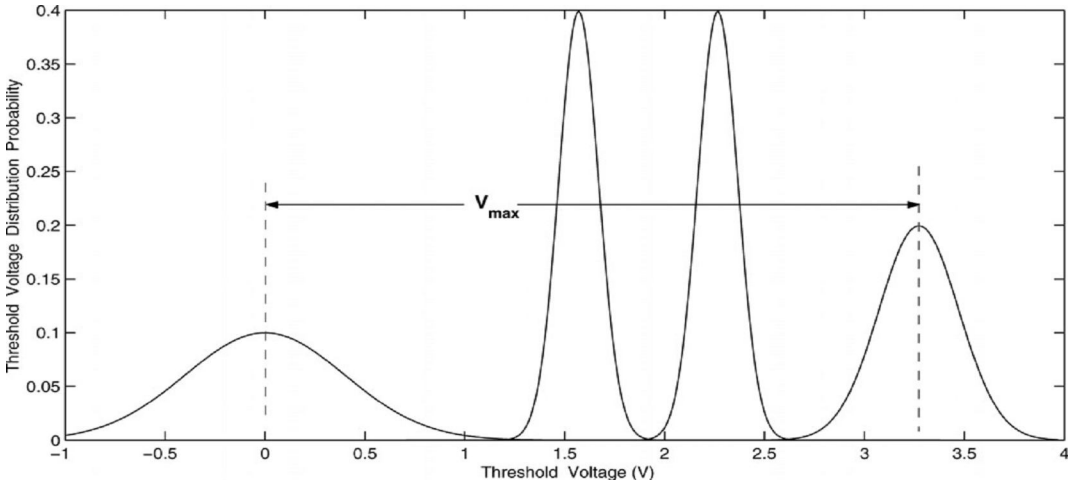


Figure 4.2: Probability density functions in 4-level cell [Maeda, Y. et al. 2009], [Sun, F. et al. 2007].

Figure 4.2 represents the typical probability distribution associated with the readout voltage corresponding to the four different stored symbols in a 4-level cell. The two inner distributions have the same standard deviation, denoted as σ, the standard deviations of the outer two distributions are 1.5σ and 1.2σ respectively. Since standard deviations of $P_0(v_{R_i})$ and $P_{Q-1}(v_{R_i})$ are usually larger than that of $P_i(v_{R_i})$ [Gregori, S. et al. 2003], [Sun, F et al. 2007], [Maeda, Y. et al. 2009] the errors in MLC cannot be expressed by any conventional channel model, such as additive white Gaussian noise (AWGN)

channel. Thus a suitable channel matrix has to be determined to describe the various errors encountered during reading process in a MLC and their relative probabilities of occurrence.

The error probabilities in the MLC are expressed by the following channel matrix:

$$P = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,Q-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,Q-1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{Q-1,0} & p_{Q-1,1} & \cdots & p_{Q-1,Q-1} \end{bmatrix} \tag{4.2}$$

where $p_{i,j}$ indicates the probability that cell data of original level *i* is read out as level *j*. Here, $p_{i,i}$ is the probability that the original data of level *i* is read out without any error. Under the assumption of the Gaussian distribution, the probability $p_{i,j}$ is calculated as

$$p_{i,j} = \int_{V_{R_{j-1}}}^{V_{R_j}} P_i(x)dx \tag{4.3}$$

where $i, j \in \{0,1,.....,Q-1\}$, $V_{R_{-1}} = -\infty$, $V_{R_{Q-1}} = \infty$.
The channel matrices computed for different values of σ for a four level MLC is shown in (4.5) – (4.13). A sample calculation showing the steps used to compute the entries of the *P* matrix for $\sigma = 0.2$ is worked out.

Computation of Threshold voltage for a 4-level MLC

$\mu_0 = -2.5,\ \mu_1 = -0.45,\ \mu_1 = 1.19,\ \mu_2 = 3.0$

For = $\sigma = 0.20,\ \sigma_0 = 1.5\sigma,\ \sigma_1 = \sigma,$

$$P_0(x) = \frac{1}{\sqrt{2\pi\sigma_0^2}} e^{-\frac{(x-\mu_0)^2}{2\sigma_0^2}} \quad \text{and} \quad P_1(x) = \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}}$$

Solving two equations, the value of $x(V_{TH_0})$ is obtained.

$$P_0(x) = \frac{1}{\sqrt{2\pi\sigma_0^2}} e^{-\frac{(x-\mu_0)^2}{2\sigma_0^2}} = P_1(x) = \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}}$$

$$\frac{e^{-\frac{(x-\mu_0)^2}{2\omega_0^2}}}{e^{-\frac{(x-\mu_1)}{2\sigma_1^2}}} = \frac{\sigma_0}{\sigma_1}$$

$$e^{-\left[\frac{(x-\mu_0)^2}{2\sigma_0^2} - \frac{(x-\mu_1)^2}{2\sigma_1^2}\right]} = \frac{\sigma_0}{\sigma_1}$$

$$e^{-\left[\frac{(x+2.5)^2}{2(0.09)} - \frac{(x+0.45)^2}{2(0.04)}\right]} = 1.5$$

$$x = -1.28184$$

The channel matrix for $Q = 4$ level cell memory is calculated as follows.

$$P = \begin{bmatrix} p_{0,0} & 1.3\times10^{-16} & 5\times10^{-82} & 2.5\times10^{-199} \\ 8.6\times10^{-17} & p_{1,1} & 1.3\times10^{-16} & 3.7\times10^{-134} \\ 2.9\times10^{-134} & 1.3\times10^{-16} & p_{2,2} & 9.3\times10^{-17} \\ 8.6\times10^{-278} & 1.4\times10^{-106} & 1.1\times10^{-16} & p_{3,3} \end{bmatrix} \tag{4.4}$$

Here the standard deviations of the threshold voltages are assumed to be $\sigma_0 = 0.15,\ \sigma_1 = 0.1 = \sigma_2, \sigma_3 = 0.12$ and the averages $\mu_0 = -2.50, \mu_1 = -0.45, \mu_2 = 1.19$ and $\mu_3 = 3$. The readout voltages are $V_{TH_0} = -1.27$, $V_{TH_1} = 0.37$ and $V_{TH_2} = 2.01$. The above channel matrix shows that, MLC suffers from asymmetric errors in which error probability between adjacent levels is high, while that between separate levels is relatively low. The channel matrix (P) computed for different values of σ is enumerated below:

For $\sigma = 0.12,\ V_{TH_0} = -1.27427,\ V_{TH_1} = 0.37,\ V_{TH_2} = 2.01447$

$$P = \begin{bmatrix} 1 & 4.89315\times10^{-12} & 1.55699\times10^{-37} & 4.0705\times10^{-139} \\ 3.2456\times10^{-12} & 1 & 4.14932\times10^{-12} & 5.0035\times10^{-94} \\ 5.17817\times10^{-94} & 4.14819\times10^{-12} & 1 & 3.19699\times10^{-12} \\ 6.47236\times10^{-194} & 8.02228\times10^{-75} & 3.85172\times10-12 & 1 \end{bmatrix} \quad (4.6)$$

For $\sigma = 0.14,\ V_{TH_0} = -1.27581,\ V_{TH_1} = 0.37,\ V_{TH_2} = 2.0157$

$$P = \begin{bmatrix} 1 & 2.78008\times10^{-9} & 8.02922\times10^{-43} & 7.70494\times10^{-143} \\ 1.83252\times10^{-9} & 1 & 2.35449\times10^{-9} & 1.07205\times10^{-69} \\ 9.80174\times10^{-70} & 2.35449\times10^{-9} & 1 & 1.88968\times10^{-9} \\ 3.41905\times10^{-143} & 4.86358\times10^{-47} & 2.27975\times10^{-9} & 1 \end{bmatrix} \quad 4.7)$$

For $\sigma = 0.16,\ V_{TH_0} = -1.27758,\ V_{TH_1} = 0.37,\ V_{TH_2} = 2.01582$

$$P = \begin{bmatrix} 1 & 1.75834\times10^{-7} & 2.93644\times10^{-33} & 2.79663\times10^{-79} \\ 1.15569\times10^{-7} & 1 & 1.48769\times10^{-7} & 6.863\times10^{-54} \\ 5.78837\times10^{-54} & 1.48769\times10^{-7} & 1 & 1.22571\times10^{-7} \\ 2.94897\times10^{-110} & 5.22396\times10^{-43} & 1.4803\times10^{-7} & 1 \end{bmatrix} \quad (4.8)$$

For $\sigma = 0.18,\ V_{TH_0} = -1.27959,\ V_{TH_1} = 0.37,\ V_{TH_2} = 2.01664$

$$P = \begin{bmatrix} 0.999997 & 3.09144\times10^{-6} & 3.50639\times10^{-18} & 4.07687\times10^{-63} \\ 2.02467\times10^{-6} & 0.999995 & 2.61237\times10^{-6} & 4.8336\times10^{-43} \\ 3.85622\times10^{-43} & 2.61237\times10^{-6} & 0.999995 & 2.19042\times10^{-6} \\ 1.15173\times10^{-87} & 2.08814\times10^{-34} & 2.64944\times10^{-6} & 0.999997 \end{bmatrix} \quad (4.9)$$

For $\sigma = 0.20,\ V_{TH_0} = -1.28184,\ V_{TH_1} = 0.37,\ V_{TH_2} = 2.01756$

$$P = \begin{bmatrix} 0.999976 & 2..44804\times10^{-5} & 5.5204\times10^{-22} & 1.51722\times10^{-51} \\ 1.59682\times10^{-5} & 0.999963 & 2.06575\times10^{-5} & 2.83418\times10^{-35} \\ 2.17229\times10^{-35} & 2.06575\times10^{-5} & 0.999962 & 7.75326\times10^{-5} \\ 1.69782\times10^{-71} & 3.03026\times10^{-28} & 2.12444\times10^{-5} & 0.999979 \end{bmatrix} \quad (4.10)$$

For $\sigma = 0.22,\ V_{TH_0} = -1.28342,\ V_{TH_1} = 0.37,\ V_{TH_2} = 2.01857$

$$P = \begin{bmatrix} 0.999885 & 1.14856\times10^{-4} & 1.70432\times10^{-18} & 5.61787\times10^{-43} \\ 7.46101\times10^{-5} & 0.999829 & 9.67815\times10^{-5} & 1.6122\times10^{-29} \\ 1.19925\times10^{-29} & 9.67815\times10^{-5} & 0.99982 & 8.28665\times10^{-5} \\ 1.5859\times10^{-59} & 1.11624\times10^{-23} & 1.00587\times10^{-4} & 0.999892 \end{bmatrix} \quad (4.11)$$

For $\sigma = 0.24,\ V_{TH_0} = -1.28703,\ V_{TH_1} = 0.37,\ V_{TH_2} = 2.01968$

$$P = \begin{bmatrix} 0.999623 & 3.76713\times10^{-4} & 7.79231\times10^{-16} & 1.87378\times10^{-36} \\ 2.43666\times10^{-4} & 0.999439 & 3.16964\times10^{-4} & 3.89531\times10^{-25} \\ 2.83274\times10^{-25} & 3.16964\times10^{-4} & 0.99941 & 2.73112\times10^{-4} \\ 2.04643\times10^{-50} & 3.36399\times10^{-20} & 3.32169\times10^{-4} & 0.999668 \end{bmatrix} \quad (4.12)$$

For $\sigma = 0.26,\ V_{TH_0} = -1.28997,\ V_{TH_1} = 0.37,\ V_{TH_2} = 2.02089$

$$P = \begin{bmatrix} 0.999041 & 9.59011\times10^{-4} & 9.26643\times10^{-14} & 2.26154\times10^{-31} \\ 6.17537\times10^{-4} & 0.998577 & 8.0567\times10^{-4} & 1.01551\times10^{-21} \\ 7.25641\times10^{-22} & 8.0567\times10^{-4} & 0.998497 & 6.97386\times10^{-4} \\ 2.76235\times10^{-43} & 1.88058\times10^{-17} & 9.20857\times10^{-4} & 0.998421 \end{bmatrix} \quad (4.13)$$

Let us assume that symbol i is written into a MLC cell and that the contents of this cell are read out in error during a read cycle. As a result of the asymmetric error mechanism, the probability of symbol $i-1$ or $i+1$ appearing at the MLC output is significantly higher than the probability of any other symbol from the set of symbols that can be stored in the MLC. This can also be inferred by a careful examination of the P matrices. To illustrate this idea, let us examine the second row of any P matrix in equations (4.5) to (4.13). We observe that the entry $p_{2,2}$ takes on a value which is close to 1. The entries $p_{2,1}$ and $p_{2,3}$ have magnitudes which are significantly higher than the other entries in this row. This implies that if symbol 1 is written into a cell which is read out in error, the erroneous read out values are most likely to be symbol 0 or symbol 2. The probability of seeing any other symbol at the output of this cell is vanishingly small. In general, for a Q-level cell, the RSER (Raw Symbol Error Rate) can be computed as,

$$P(e) = \sum_{i=0}^{Q-1} P(i)P(e / i) \tag{4.14}$$

where $P(e / i)$ denotes the probability of reading out the contents of a cell in error when symbol i has been written into the cell. Thus, for Q = 4, the equation representing the RSER is computed as follows:

$$P(e) = P(0)\ P\left(e/0\right) + P(1)\ P\left(e/1\right) + P(2)\ P\left(e/2\right) + P(3)\ P\left(e/3\right) \tag{4.15}$$

4.3 Code Synthesis for Multilevel Flash

Referring to memory model 1, (512 bytes of information and 16 bytes of overhead), with Q = 2 level MLC, for t = 9, the shortened BCH code parameters are (4213, 4096) over $F_{2^{13}}$.

$$v_0 v_1 \mid v_2 v_3 \mid v_4 v_5 \mid \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots v_{4210} v_{4211} \mid v_{4212}$$

$$S_0 \quad S_1 \quad S_2 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad S_{2106}$$

If Gray code is used to map binary 2 – tuples to symbols over F_{2^2}, then an error event (i.e, a symbol being readout in error) is almost always due to a single bit change in two bit representation. Error events involving both the binary digits in error are extremely improbable (i.e, the probability of such events are of the order of 10^{-82} or smaller). Thus by assigning Gray map to assign bit pattern to symbols, the (4213, 4096) binary BCH code capable of correcting t = 9 errors over a span of 4213 bits can effectively correct t = 9 symbol errors over a span of 2107 *4-ary* symbols.

Now combining two sectors, the memory model will have 1024 bytes (8192 bits) of information and 32 bytes (256 bits) of overhead. If this model is used to synthesize the codes for Q = 4 level MLC, then length of the code $n = 2^{14} - 1 = 16383$. Also, $n - k \le 256$. For t = 18, the shortened BCH code parameters are (8444, 8192). This code can correct eighteen single bit errors over a span of 8192 bits (one sector in Q = 2 case). It can also correct eighteen symbols over a span of 8444 *4-ary* symbols with Gray mapping.

4.3.1 RS Codes

Referring to the memory model 1 (with 512 bytes of information and 16 bytes of overheads), a t = 9 BCH code will have the parameters of (4213, 4096) after shortening. So, in this case k = 512, n > 512, $n = 2^{10} - 1$ = 1023. Let us synthesize a RS code over $F_{2^{10}}$.

$k = 512 = 512 \times 8 = 4096$ bits ~ 410 ten bit symbols.

Considering $t = 6,\ b = 1,\ \delta = 2t + 1 = 13,\ b + \delta - 2 = 12$. The generator polynomial is

$$g(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4)(x-\alpha^5)(x-\alpha^6)(x-\alpha^7)(x-\alpha^8)(x-\alpha^9)(x-\alpha^{10})$$
$$(x-\alpha^{11})(x-\alpha^{12})$$

$$g(x) = \alpha^6 + \alpha^{800}x + \alpha^{861}x^2 + \alpha^{676}x^3 + \alpha^{244}x^4 + \alpha^{419}x^5 + \alpha^{444}x^6 + \alpha^{968}x^7 +$$
$$\alpha^{400}x^8 + \alpha^{256}x^9 + \alpha^{36}x^{10} + \alpha^{961}x^{11} + x^{12} \qquad (4.16)$$

Here the degree of *g(x)* =12. Therefore, $n - k = 12$ ten bit symbols i.e. 120 bits. This code is an n=422, k= 410, t=6 RS code over $F_{2^{10}}$. So, six symbols in errors can be corrected.

If two sectors are combined, then k = 1024 bytes = 8192 bits and $n - k|_{\max}$ =32 bytes = 256 bits. The RS code should be from the field $F_{2^{10}} = \{0,1,\alpha,\alpha^2,.....,\alpha^{1022}\}$.

k = 819.2 ~ 820 ten bit symbols and $n - k$ ~ 25 ten bit symbols.

For $t = 12$, $\delta = 2t + 1 = 25$, $b + \delta - 2 = 24$.

Therefore,

$$g(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4)(x-\alpha^5)(x-\alpha^6)(x-\alpha^7)(x-\alpha^8)(x-\alpha^9)$$
$$(x-\alpha^{10})(x-\alpha^{11})(x-\alpha^{12})(x-\alpha^{13})(x-\alpha^{14})(x-\alpha^{15})(x-\alpha^{16})(x-\alpha^{17})(x-\alpha^{18})$$
$$(x-\alpha^{19})(x-\alpha^{20})(x-\alpha^{21})(x-\alpha^{22})(x-\alpha^{23})(x-\alpha^{24})$$

$$g(x) = \alpha^{866} + \alpha^{212}x + \alpha^{917}x^2 + \alpha^{105}x^3 + \alpha^{573}x^4 + \alpha^{32}x^5 + \alpha^{940}x^6 + \alpha^{948}x^7 + \alpha^{130}x^8 + \alpha^{493}x^9 + \alpha^{1009}x^{10} + \alpha^{386}x^{11} + \alpha^{475}x^{12} + \alpha^{475}x^{12} + \alpha^{366}x^{13} + \alpha^{234}x^{14} + \alpha^{363}x^{15} + \alpha^{254}x^{16} + \alpha^{562}x^{17} + \alpha^{475}x^{18} + \alpha^{185}x^{19} + \alpha^{393}x^{20} + \alpha^{704}x^{21} + \alpha^{224}x^{22} + \alpha^{16}x^{23} + x^{24} \quad (4.17)$$

Here degree of *g(x)* = 24 = $n - k$. Since k = 820, n = 24 + k = 844. The suitable code is (844, 820) RS code over $F_{2^{10}}$, with t = 12. The code can correct twelve ten bit symbols over a span of 844 ten bit symbols.

4.3.2 Code synthesis for 8-level MLC

Let us consider 8-level MLC flash with memory model 1 architecture.

512 bytes (4096 8-ary cells) 4096 x 3 = 12288 bits	16 bytes 16x8x3 = 384 bits

In this case $k = 12288$ and $n - k|_{\max} = 384$.

$n = 2^{14} - 1 = 16383$

Let $t = 25$, $b = 1$, $\delta = 2t + 1 = 51$, $b + \delta - 2 = 50$.

The required roots are: $\{\alpha, \alpha^2, \alpha^3, ..., \alpha^{50}\}$. After deriving the conjugacy classes, the minimal polynomials that are obtained are listed in equation (4.18).

$$
\begin{aligned}
M_1(x) &= 1 + x + x^6 + x^{10} + x^{14} \\
M_3(x) &= 1 + x + x^2 + x^5 + x^8 + x^{14} \\
M_5(x) &= 1 + x + x^3 + x^4 + x^6 + x^7 + x^9 + x^{10} + x^{14} \\
M_7(x) &= 1 + x^4 + x^5 + x^6 + x^7 + x^8 + x^9 + x^{12} + x^{14} \\
M_9(x) &= 1 + x^2 + x^3 + x^5 + x^{11} + x^{12} + x^{14} \\
M_{11}(x) &= 1 + x + x^6 + x^8 + x^{14} \\
M_{13}(x) &= 1 + x^5 + x^6 + x^9 + x^{10} + x^{11} + x^{12} + x^{13} + x^{14} \\
M_{15}(x) &= 1 + x^2 + x^5 + x^7 + x^8 + x^9 + x^{10} + x^{11} + x^{14} \\
M_{17}(x) &= 1 + x + x^2 + x^3 + x^4 + x^5 + x^7 + x^8 + x^{10} + x^{11} + x^{14} \\
M_{19}(x) &= 1 + x + x^6 + x^{11} + x^{14} \\
M_{21}(x) &= 1 + x^2 + x^4 + x^8 + x^{10} + x^{11} + x^{14} \\
M_{23}(x) &= 1 + x^2 + x^5 + x^6 + x^9 + x^{11} + x^{14} \\
M_{25}(x) &= 1 + x + x^6 + x^7 + x^{10} + x^{11} + x^{12} + x^{13} + x^{14} \\
M_{27}(x) &= 1 + x + x^7 + x^8 + x^{10} + x^{13} + x^{14} \\
M_{29}(x) &= 1 + x + x^2 + x^3 + x^5 + x^8 + x^{11} + x^{13} + x^{14} \\
M_{31}(x) &= 1 + x^3 + x^4 + x^7 + x^9 + x^{10} + x^{11} + x^{12} + x^{14} \\
M_{33}(x) &= 1 + x + x^2 + x^3 + x^4 + x^6 + x^7 + x^8 + x^{10} + x^{12} + x^{14} \\
M_{35}(x) &= 1 + x + x^2 + x^4 + x^5 + x^6 + x^{11} + x^{13} + x^{14} \\
M_{37}(x) &= 1 + x + x^3 + x^5 + x^6 + x^{13} + x^{14} \\
M_{39}(x) &= 1 + x^2 + x^3 + x^6 + x^{10} + x^{13} + x^{14} \\
M_{41}(x) &= 1 + x + x^3 + x^5 + x^6 + x^7 + x^8 + x^9 + x^{11} + x^{12} + x^{14} \\
M_{43}(x) &= 1 + x + x^2 + x^4 + x^8 + x^{10} + x^{12} + x^{13} + x^{14}
\end{aligned}
\qquad \text{(4.18 a)}
$$

$$M_{45}(x) = 1 + x^2 + x^3 + x^6 + x^7 + x^8 + x^{11} + x^{13} + x^{14}$$
$$M_{47}(x) = 1 + x + x^4 + x^8 + x^{12} + x^{13} + x^{14}$$
$$M_{49}(x) = 1 + x + x^3 + x^5 + x^6 + x^9 + x^{10} + x^{11} + x^{14} \qquad (4.18b)$$
$$M_{51}(x) = 1 + x + x^3 + x^5 + x^7 + x^8 + x^9 + x^{10} + x^{11} + x^{13} + x^{14}$$
$$M_{53}(x) = 1 + x^2 + x^3 + x^5 + x^7 + x^9 + x^{10} + x^{12} + x^{14}$$

Therefore,

$$\begin{aligned} g(x) = 1 &+ x^3 + x^5 + x^6 + x^9 + x^{11} + x^{13} + x^{16} + x^{19} + x^{22} + x^{27} + x^{28} + \\ &x^{29} + x^{36} + x^{38} + x^{39} + x^{41} + x^{43} + x^{44} + x^{45} + x^{46} + x^{47} + x^{49} + \\ &x^{50} + x^{52} + x^{54} + x^{55} + x^{56} + x^{58} + x^{59} + x^{60} + x^{63} + x^{64} + x^{69} + \\ &x^{70} + x^{73} + x^{76} + x^{77} + x^{81} + x^{82} + x^{87} + x^{92} + x^{93} + x^{95} + x^{100} + \\ &x^{101} + x^{102} + x^{103} + x^{104} + x^{106} + x^{108} + x^{109} + x^{110} + x^{114} + x^{115} + x^{116} + \\ &x^{122} + x^{124} + x^{125} + x^{127} + x^{128} + x^{131} + x^{135} + x^{136} + x^{137} + x^{138} + x^{140} + \\ &x^{144} + x^{145} + x^{147} + x^{148} + x^{149} + x^{151} + x^{155} + x^{158} + x^{160} + x^{163} + x^{164} + \\ &x^{168} + x^{169} + x^{174} + x^{176} + x^{177} + x^{179} + x^{180} + x^{183} + x^{184} + x^{186} + x^{189} + \\ &x^{190} + x^{191} + x^{192} + x^{193} + x^{194} + x^{195} + x^{196} + x^{198} + x^{201} + x^{204} + x^{206} + \\ &x^{207} + x^{210} + x^{213} + x^{217} + x^{219} + x^{221} + x^{222} + x^{224} + x^{225} + x^{227} + x^{229} + \\ &x^{231} + x^{232} + x^{233} + x^{235} + x^{236} + x^{237} + x^{238} + x^{239} + x^{240} + x^{241} + x^{242} + \\ &x^{243} + x^{246} + x^{247} + x^{248} + x^{248} + x^{250} + x^{251} + x^{252} + x^{253} + x^{254} + x^{255} + \\ &x^{257} + x^{259} + x^{260} + x^{261} + x^{263} + x^{264} + x^{265} + x^{269} + x^{271} + x^{277} + x^{278} + \\ &x^{280} + x^{284} + x^{286} + x^{287} + x^{290} + x^{291} + x^{293} + x^{294} + x^{295} + x^{298} + x^{299} + \\ &x^{301} + x^{302} + x^{303} + x^{305} + x^{306} + x^{307} + x^{308} + x^{310} + x^{311} + x^{316} + x^{317} + \\ &x^{318} + x^{319} + x^{321} + x^{322} + x^{324} + x^{325} + x^{326} + x^{327} + x^{328} + x^{330} + x^{332} + \\ &x^{333} + x^{334} + x^{337} + x^{339} + x^{341} + x^{343} + x^{345} + x^{347} + x^{349} + x^{350} \end{aligned} \qquad (4.19)$$

Let $t = 26$, $b = 1$, $\delta = 2t + 1 = 53$, $b + \delta - 2 = 52$.
The required roots are: $\{\alpha, \alpha^2, \alpha^3, ..., \alpha^{52}\}$ and the generator polynomial is

$$g(x) = 1 + x^4 + x^6 + x^8 + x^9 + x^{13} + x^{14} + x^{16} + x^{20} + x^{21} + x^{22} + x^{25} +$$
$$x^{26} + x^{27} + x^{28} + x^{29} + x^{30} + x^{33} + x^{35} + x^{36} + x^{39} + x^{40} + x^{41} + x^{44} +$$
$$x^{46} + x^{49} + x^{52} + x^{53} + x^{55} + x^{56} + x^{57} + x^{58} + x^{59} + x^{61} + x^{62} + x^{63} +$$
$$x^{64} + x^{65} + x^{69} + x^{71} + x^{72} + x^{74} + x^{78} + x^{79} + x^{82} + x^{83} + x^{84} + x^{85} +$$
$$x^{87} + x^{88} + x^{89} + x^{90} + x^{912} + x^{93} + x^{94} + x^{95} + x^{98} + x^{99} + x^{100} + x^{101} +$$
$$x^{102} + x^{104} + x^{105} + x^{107} + x^{108} + x^{112} + x^{113} + x^{114} + x^{116} + x^{118} + x^{122} +$$
$$x^{124} + x^{126} + x^{127} + x^{128} + x^{129} + x^{132} + x^{136} + x^{137} + x^{139} + x^{140} + x^{144} +$$
$$x^{146} + x^{148} + x^{149} + x^{155} + x^{159} + x^{160} + x^{161} + x^{163} + x^{165} + x^{166} + x^{167} +$$
$$x^{168} + x^{170} + x^{171} + x^{174} + x^{175} + x^{177} + x^{179} + x^{182} + x^{183} + x^{185} + x^{186} +$$
$$x^{187} + x^{188} + x^{190} + x^{191} + x^{196} + x^{198} + x^{200} + x^{202} + x^{205} + x^{206} + x^{207} +$$
$$x^{208} + x^{211} + x^{214} + x^{215} + x^{216} + x^{217} + x^{218} + x^{219} + x^{222} + x^{223} + x^{226} +$$
$$x^{228} + x^{229} + x^{230} + x^{231} + x^{232} + x^{234} + x^{235} + x^{238} + x^{239} + x^{240} + x^{242} +$$
$$x^{243} + x^{247} + x^{248} + x^{252} + x^{253} + x^{254} + x^{255} + x^{256} + x^{257} + x^{260} + x^{261} +$$
$$x^{270} + x^{272} + x^{273} + x^{274} + x^{275} + x^{281} + x^{282} + x^{283} + x^{284} + x^{285} + x^{286} +$$
$$x^{291} + x^{293} + x^{294} + x^{296} + x^{297} + x^{298} + x^{300} + x^{301} + x^{302} + x^{303} + x^{307} +$$
$$x^{308} + x^{309} + x^{310} + x^{313} + x^{315} + x^{317} + x^{318} + x^{321} + x^{324} + x^{325} + x^{326} + \qquad (4.20)$$
$$x^{328} + x^{333} + x^{334} + x^{338} + x^{339} + x^{340} + x^{341} + x^{342} + x^{343} + x^{344} + x^{346} +$$
$$x^{347} + x^{348} + x^{349} + x^{354} + x^{359} + x^{360} + x^{362} + x^{364}$$

Let $t = 27$, $b = 1$, $\delta = 2t + 1 = 55$, $b + \delta - 2 = 54$.
The required roots are: $\{\alpha, \alpha^2, \alpha^3, ..., \alpha^{54}\}$. Hence the generator polynomial is

$$g(x) = 1 + x^2 + x^5 + x^6 + x^8 + x^{10} + x^{11} + x^{13} + x^{19} + x^{24} + x^{25} + x^{27} +$$
$$x^{29} + x^{30} + x^{31} + x^{33} + x^{34} + x^{35} + x^{37} + x^{41} + x^{44} + x^{46} + x^{50} + x^{51} +$$
$$x^{52} + x^{53} + x^{55} + x^{56} + x^{57} + x^{58} + x^{59} + x^{60} + x^{64} + x^{65} + x^{66} + x^{67} +$$
$$x^{70} + x^{71} + x^{73} + x^{74} + x^{76} + x^{79} + x^{82} + x^{83} + x^{84} + x^{85} + x^{86} + x^{87} +$$
$$x^{89} + x^{92} + x^{93} + x^{94} + x^{95} + x^{98} + x^{99} + x^{100} + x^{101} + x^{102} + x^{104} + x^{105} +$$
$$x^{107} + x^{108} + x^{112} + x^{113} + x^{114} + x^{116} + x^{118} + x^{119} + x^{120} + x^{121} + x^{126} +$$
$$x^{127} + x^{128} + x^{129} + x^{131} + x^{132} + x^{133} + x^{134} + x^{135} + x^{138} + x^{141} + x^{142} +$$
$$x^{143} + x^{147} + x^{150} + x^{154} + x^{155} + x^{156} + x^{158} + x^{161} + x^{164} + x^{169} + x^{170} +$$
$$x^{171} + x^{172} + x^{173} + x^{174} + x^{175} + x^{183} + x^{185} + x^{186} + x^{187} + x^{190} + x^{192} +$$
$$x^{195} + x^{196} + x^{197} + x^{198} + x^{207} + x^{208} + x^{212} + x^{213} + x^{222} + x^{223} + x^{224} +$$
$$x^{228} + x^{230} + x^{231} + x^{234} + x^{236} + x^{239} + x^{240} + x^{242} + x^{247} + x^{248} + x^{249} +$$
$$x^{250} + x^{251} + x^{254} + x^{255} + x^{256} + x^{257} + x^{259} + x^{260} + x^{261} + x^{263} + x^{264} +$$
$$x^{266} + x^{268} + x^{269} + x^{272} + x^{273} + x^{274} + x^{275} + x^{279} + x^{280} + x^{281} + x^{282} +$$
$$x^{284} + x^{285} + x^{287} + x^{288} + x^{290} + x^{291} + x^{292} + x^{294} + x^{295} + x^{296} + x^{301} +$$
$$x^{305} + x^{306} + x^{309} + x^{311} + x^{312} + x^{313} + x^{318} + x^{322} + x^{324} + x^{328} + x^{331} +$$
$$x^{333} + x^{334} + x^{335} + x^{337} + x^{338} + x^{339} + x^{340} + x^{342} + x^{344} + x^{345} + x^{348} +$$
$$x^{349} + x^{351} + x^{360} + x^{363} + x^{366} + x^{367} + x^{370} + x^{371} + x^{374} + x^{378} \quad (4.21)$$

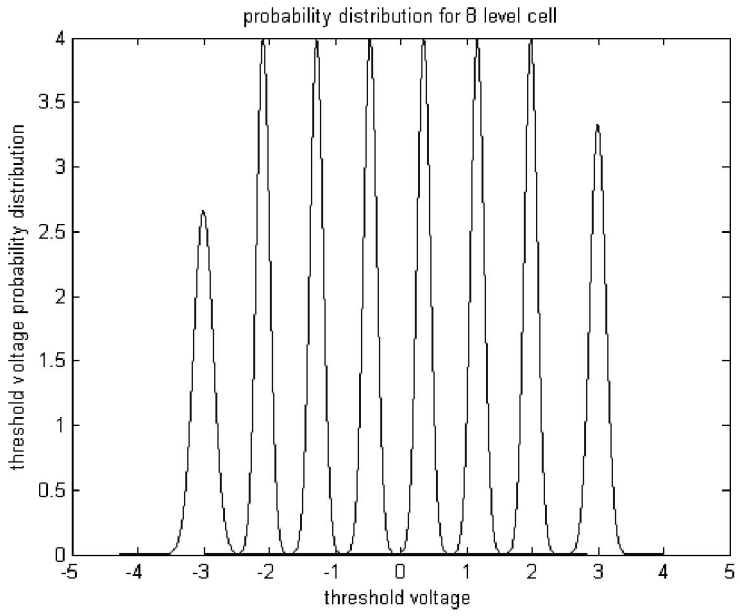


Figure 4.3: Probability density functions in 8-level cell.

Fig.4.3 describes the probability density functions in 8-level cell [Maeda, Y. et al. 2009]. The P matrix computed for for $Q = 8$ with $\sigma = 0.1$ and 0.125 is shown below.

P =

$$\begin{bmatrix} 0.999826 & 1.7355\times10^{-4} & 1.03571\times10^{-18} & 5.54008\times10^{-46} & 5.21766\times10^{-86} & 8.02884\times10^{-139} & 1.95744\times10^{-204} & 6.27304\times10^{-289} \\ 1.12569\times10^{-4} & 0.999864 & 2.30072\times10^{-5} & 1.14291\times10^{-34} & 1.39358\times10^{-92} & 2.86903\times10^{-179} & 9.14323\times10^{-295} & 0 \\ 1.2268\times10^{-32} & 2.30072\times10^{-5} & 0.999954 & 2.30072\times10^{-5} & 1.1429\times10^{-34} & 1.39358\times10^{-92} & 2.86903\times10^{-179} & 4.3151\times10^{-304} \\ 3.4329\times10^{-89} & 1.1429\times10^{-34} & 2.30072\times10^{-5} & 0.999954 & 2.30072\times10^{-5} & 1.14291\times10^{-34} & 1.39358\times10^{-92} & 1.3291\times10^{-186} \\ 1.6336\times10^{-174} & 1.39358\times10^{-92} & 1.14291\times10^{-34} & 2.30072\times10^{-5} & 0.999954 & 2.30072\times10^{-5} & 1.14291\times10^{-34} & 8.3582\times10^{-98} \\ 1.20627\times10^{-288} & 2.86903\times10^{-179} & 1.39358\times10^{-92} & 1.14291\times10^{-34} & 2.30072\times10^{-5} & 0.999954 & 2.30072\times10^{-5} & 7.63694\times10^{-34} \\ 0 & 9.14323\times10^{-295} & 2.86903\times10^{-179} & 1.39358\times10^{-92} & 1.14291\times10^{-34} & 2.30072\times10^{-5} & 0.999975 & 1.61598\times10^{-6} \\ 0 & 0 & 1.02694\times10^{-228} & 1.86579\times10^{-143} & 3.38296\times10^{-78} & 6.53853\times10^{-33} & 1.95368\times10^{-6} & 0.999998 \end{bmatrix}$$

P

=

$$\begin{bmatrix} 0.997758 & 2.224214\times10^{-3} & 1.25568\times10^{-12} & 3.73652\times10^{-30} & 8.0542\times10^{-56} & 1.16947\times10^{-89} & 1.10971\times10^{-131} & 7.24794\times10^{-186} \\ 1.43598\times10^{-3} & 0.998007 & 5.57061\times10^{-4} & 6.86053\times10^{-23} & 4.93461\times10^{-60} & 1.4515\times10^{-115} & 1.60277\times10^{-189} & 2.34423\times10^{-289} \\ 1.03822\times10^{-4} & 5.57061\times10^{-4} & 0.998886 & 5.57061\times10^{-4} & 6.86053\times10^{-23} & 4.93461\times10^{-60} & 1.4515\times10^{-125} & 1.2832\times10^{-195} \\ 4.54982\times10^{-58} & 6.86053\times10^{-23} & 5.57061\times10^{-4} & 0.998886 & 5.57061\times10^{-4} & 6.86053\times10^{-23} & 4.93461\times10^{-60} & 2.54507\times10^{-120} \\ 8.20572\times10^{-113} & 4.93461\times10^{-60} & 6.86053\times10^{-23} & 5.57061\times10^{-4} & 0.998886 & 5.57061\times10^{-4} & 6.86053\times10^{-63} & 1.88831\times10^{-69} \\ 5.56757\times10^{-186} & 1.4515\times10^{-115} & 4.93461\times10^{-60} & 6.86053\times10^{-23} & 5.57061\times10^{-4} & 0.998886 & 5.57061\times10^{-4} & 5.67256\times10^{-85} \\ 1.37229\times10^{-277} & 1.23893\times10^{-214} & 1.4515\times10^{-115} & 4.93461\times10^{-60} & 6.86053\times10^{-23} & 5.57061\times10^{-4} & 0.999349 & 9.42013\times10^{-5} \\ 3.75438\times10^{-291} & 0 & 3.14117\times10^{-147} & 1.25828\times10^{-92} & 8.18713\times10^{-51} & 9.23245\times10^{-22} & 1.14381\times10^{-4} & 0.999866 \end{bmatrix}$$

RSER can be computes as follows:

$$P(e) = P(0)P(e/0) + P(1)P(e/1) + P(2)P(e/2) + P(3)P(e/3) + P(4)P(e/4) + P(5)P(e/5) + P(6)P(e/6) + P(7)P(e/7)$$

Assuming that symbols 0 – 7 can exist with equal apriori probability in the cell,

$P(0) = P(1) = .\cdots\cdots = P(7) = \frac{1}{8}$, the RSER simplifies to,

$$P(e) = \frac{1}{8}\begin{bmatrix} P(e/0) + P(e/1) + P(e/2) + P(e/3) + \\ P(e/4) + P(e/5) + P(e/6) + P(e/7) \end{bmatrix} \quad (4.23)$$

The probability of the decoding error associated with a code that can correct t errors can be expressed as

$$P_{\,decoding\ error} = \sum_{k=t+1}^{n} \binom{n}{k} (RSER)^k (1 - RSER)^{n-k} \tag{4.24}$$

Probability of decoding error computed for different values of σ are shown in Table 4.2 and Table 4.3.

Table 4.2: Computed values of RSER and Probability of decoding error for 4-level MLC

σ	RSER	Probability of decoding error (Memory model 1) for code with parameters (8444, 8192) & t = 18	Probability of decoding error (Memory model 2) for code with parameters (8696, 8192) & t = 36
0.18	3.7951×10^{-6}	3.1822×10^{-46}	1.0053×10^{-99}
0.19	1.1577×10^{-5}	4.7775×10^{-37}	7.8451×10^{-81}
0.20	3.0135×10^{-5}	3.2230×10^{-29}	1.5854×10^{-65}
0.21	6.8909×10^{-5}	1.5793×10^{-22}	2.2225×10^{-52}
0.22	1.4162×10^{-4}	7.7686×10^{-17}	4.5486×10^{-41}
0.23	2.6632×10^{-4}	4.6789×10^{-12}	2.2312×10^{-31}
0.24	4.6489×10^{-4}	3.8230×10^{-8}	3.7709×10^{-23}
0.25	7.4679×10^{-4}	3.3731×10^{-5}	1.4532×10^{-16}
0.26	1.1015×10^{-3}	8.2444×10^{-4}	1.4151×10^{-10}

Table 4.3: Computed values of RSER and Probability of decoding error for 8-level MLC

σ	RSER	Probability of decoding error (Memory model 1) for code with parameters (12666, 12288) & t = 27	Probability of decoding error (Memory model 2) for code with parameters (13044, 12288) & t = 54
0.1	6.4951×10^{-9}	6.0980×10^{-33}	3.3642×10^{-78}
0.11	2.5685×10^{-4}	3.0872×10^{-17}	2.0185×10^{-46}
0.12	7.5244×10^{-4}	9.0401×10^{-7}	1.7272×10^{-23}
0.125	1.1821×10^{-3}	1.6625×10^{-3}	4.5435×10^{-15}

Hence in majority of the cases a symbol error is due to one bit in the representation of the symbol being readout erroneously. Error events involving more than one binary digit in

error are highly improbable. Hence we have estimated the value of the UBER as the same as Uncorrectable Symbol Error Rate (USER) which is calculated from (4.24). This can be observed by an examination of the *P* matrices (4.5) - (4.13). Let us examine the second row of (4.10). We see that if symbol 1 is written into the cell, the probability of it being read out as symbol 0 or 2 is of the order of 10^{-5} while the probability of the contents being read out as symbol 3 is of the order of 10^{-35}. Thus by assigning Gray map to assign bit pattern to symbols, the (8444, 8192) binary BCH code capable of correcting $t = 9$ bit errors over a span of 8444 bits can effectively correct $t = 9$ symbol errors over a span of 4222 *4-ary* symbols. The performance plots of the codes synthesized are shown in Figure 4.4, Figure 4.5, Figure 4.6 and Figure 4.7. It is observed that as vrange/sigma increases, the values of UBER becomes smaller and smaller with the use of these codes.

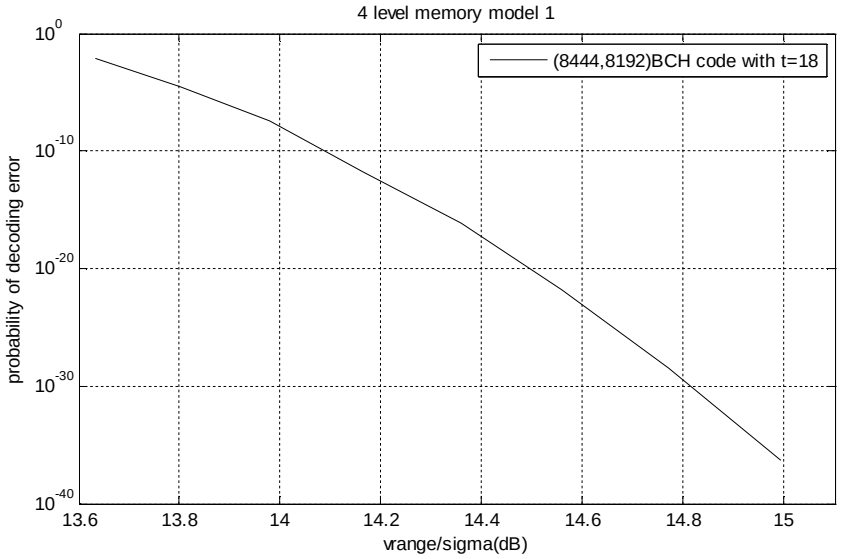


Figure 4.4: Performance of $t = 18$ BCH code for MLC (4-level)

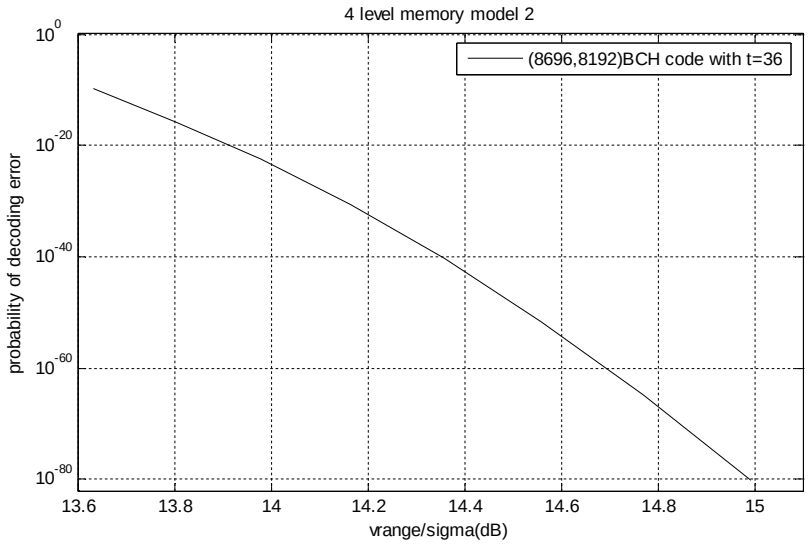


Figure 4.5: Performance of t = 36 BCH code for MLC (4-level)

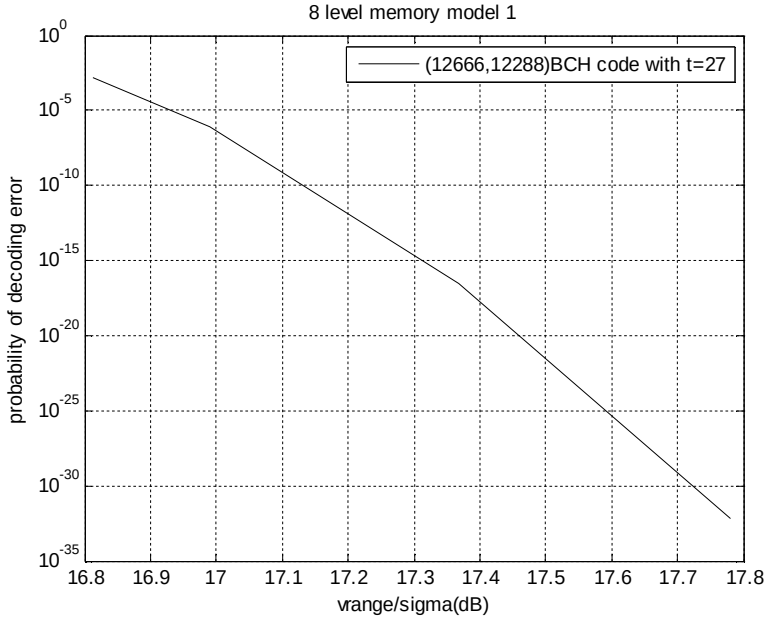


Figure 4.6: Performance of t = 27 BCH code for MLC (8-level)

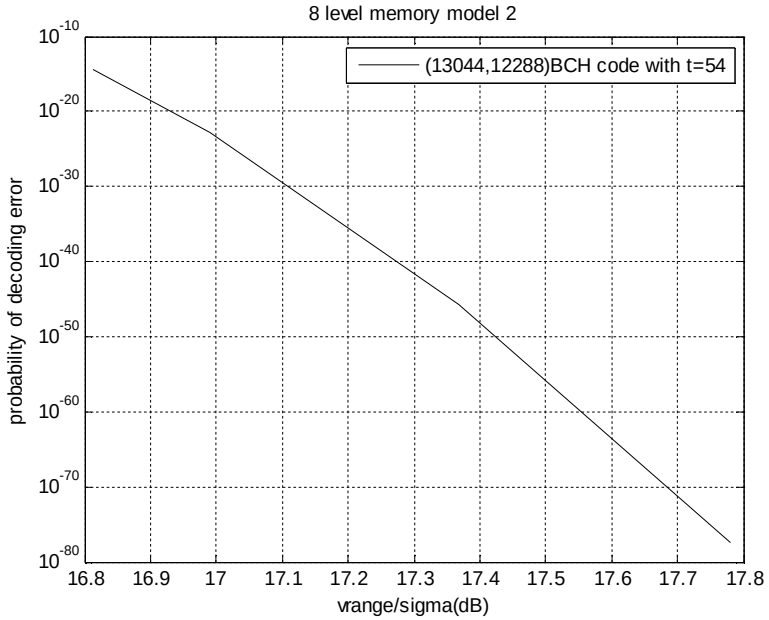


Figure 4.7: Performance of t = 54 BCH code for MLC (8-level)

From Table 4.2, we infer that in the case of 4-level MLC, use of codes based on Memory model 1 can enhance device reliability significantly for values of $\sigma \leq 0.22$. Use of codes based on Memory model 2 can improve device reliability even when σ increases to 0.25. Similarly from Table 4.3, we infer that for 8-level MLC, use of suitable codes based on Memory model 1 can enhance device reliability for values of $\sigma \leq 0.11$. Codes based on Memory model 2 can improve reliability even when value of σ increases to 0.125. The generator polynomials computed are shown in (4.25), (4.26), (4.27), (4.28).

For $t = 18$

$$g(x) = 1 + x + x^3 + x^6 + x^7 + x^8 + x^9 + x^{10} + x^{11} + x^{13} + x^{14} + x^{16} + x^{19} + x^{20} + x^{23} + x^{32} + x^{33} + x^{34} + x^{39} + x^{40} + x^{41} + x^{42} + x^{43} + x^{44} + x^{49} + x^{50} + x^{53} + x^{54} + x^{58} + x^{59} + x^{60} + x^{61} + x^{62} + x^{65} + x^{67} + x^{70} + x^{72} + x^{73} + x^{75} + x^{77} + x^{79} + x^{80} + x^{82} + x^{83} + x^{85} + x^{90} + x^{91} + x^{94} + x^{97} + x^{98} + x^{99} + x^{102} + x^{104} + x^{105} + x^{108} + x^{109} + x^{112} + x^{116} + x^{118} + x^{119} + x^{120} + x^{121} + x^{122} + x^{124} + x^{125} + x^{127} + x^{130} + x^{132} + x^{133} + x^{137} + x^{138} + x^{140} + x^{141} + x^{146} + x^{148} + x^{149} + x^{150} + x^{153} + x^{155} + x^{156} + x^{158} + x^{161} + x^{163} + x^{164} + x^{165} + x^{168} + x^{169} + x^{170} + x^{172} + x^{173} + x^{177} + x^{178} + x^{179} + x^{184} + x^{187} + x^{191} + x^{192} + x^{193} + x^{195} + x^{197} + x^{200} + x^{202} + x^{203} + x^{208} + x^{211} + x^{214} + x^{216} + x^{217} + x^{218} + x^{219} + x^{224} + x^{226} + x^{227} + x^{228} + x^{230} + x^{233} + x^{235} + x^{236} + x^{239} + x^{241} + x^{242} + x^{250} + x^{252} \quad (4.25)$$

For $t = 27$

$$g(x) = 1 + x^2 + x^5 + x^6 + x^8 + x^{10} + x^{11} + x^{13} + x^{19} + x^{24} + x^{25} + x^{27} + x^{29} + x^{30} + x^{31} + x^{33} + x^{34} + x^{35} + x^{37} + x^{41} + x^{43} + x^{44} + x^{46} + x^{50} + x^{51} + x^{52} + x^{53} + x^{55} + x^{56} + x^{57} + x^{58} + x^{59} + x^{60} + x^{66} + x^{65} + x^{66} + x^{67} + x^{70} + x^{71} + x^{73} + x^{74} + x^{76} + x^{79} + x^{82} + x^{83} + x^{84} + x^{85} + x^{86} + x^{87} + x^{89} + x^{92} + x^{93} + x^{94} + x^{96} + x^{97} + x^{98} + x^{100} + x^{101} + x^{102} + x^{103} + x^{104} + x^{105} + x^{111} + x^{113} + x^{114} + x^{115} + x^{117} + x^{118} + x^{119} + x^{120} + x^{121} + x^{126} + x^{127} + x^{128} + x^{129} + x^{131} + x^{132} + x^{133} + x^{134} + x^{135} + x^{138} + x^{141} + x^{142} + x^{143} + x^{147} + x^{150} + x^{154} + x^{155} + x^{156} + x^{158} + x^{161} + x^{164} + x^{169} + x^{170} + x^{171} + x^{172} + x^{173} + x^{174} + x^{175} + x^{183} + x^{185} + x^{186} + x^{187} + x^{190} + x^{192} + x^{195} + x^{196} + x^{197} + x^{198} + x^{207} + x^{208} + x^{212} + x^{213} + x^{222} + x^{223} + x^{224} + x^{228} + x^{230} + x^{231} + x^{234} + x^{236} + x^{239} + x^{240} + x^{242} + x^{247} + x^{248} + x^{249} + x^{250} + x^{251} + x^{254} + x^{255} + x^{256} + x^{257} + x^{259} + x^{260} + x^{261} + x^{263} + x^{264} + x^{266} + x^{268} + x^{269} + x^{272} + x^{273} + x^{274} + x^{275} + x^{279} + x^{280} + x^{281} + x^{282} + x^{284} + x^{285} + x^{287} + x^{288} + x^{290} + x^{291} + x^{292} + x^{294} + x^{295} + x^{296} + x^{301} + x^{305} + x^{306} + x^{309} + x^{311} + x^{312} + x^{313} + x^{318} + x^{322} + x^{324} + x^{328} + x^{331} + x^{333} + x^{334} + x^{335} + x^{337} + x^{338} + x^{339} + x^{340} + x^{342} + x^{344} + x^{345} + x^{348} + x^{349} + x^{351} + x^{360} + x^{363} + x^{366} + x^{367} + x^{370} + x^{371} + x^{374} + x^{378} \quad (4.26)$$

For $t = 36$

$$
\begin{aligned}
g(x) = 1 &+ x + x^3 + x^5 + x^6 + x^{11} + x^{12} + x^{14} + x^{16} + x^{17} + x^{18} + x^{19} + \\
&x^{20} + x^{21} + x^{23} + x^{26} + x^{29} + x^{32} + x^{34} + x^{35} + x^{37} + x^{39} + x^{40} + x^{41} + \\
&x^{42} + x^{43} + x^{45} + x^{47} + x^{49} + x^{50} + x^{52} + x^{59} + x^{60} + x^{61} + x^{62} + x^{64} + \\
&x^{67} + x^{68} + x^{70} + x^{76} + x^{78} + x^{79} + x^{81} + x^{82} + x^{83} + x^{85} + x^{86} + x^{88} + \\
&x^{89} + x^{90} + x^{91} + x^{92} + x^{95} + x^{96} + x^{98} + x^{99} + x^{102} + x^{107} + x^{116} + x^{119} + \\
&x^{120} + x^{123} + x^{125} + x^{127} + x^{129} + x^{130} + x^{131} + x^{133} + x^{134} + x^{135} + x^{136} + \\
&x^{137} + x^{138} + x^{140} + x^{141} + x^{142} + x^{143} + x^{145} + x^{149} + x^{150} + x^{151} + x^{152} + \\
&x^{154} + x^{156} + x^{157} + x^{160} + x^{161} + x^{162} + x^{163} + x^{166} + x^{172} + x^{173} + x^{174} + \\
&x^{175} + x^{176} + x^{178} + x^{179} + x^{181} + x^{183} + x^{184} + x^{185} + x^{190} + x^{191} + x^{193} + \\
&x^{195} + x^{196} + x^{197} + x^{200} + x^{202} + x^{203} + x^{204} + x^{206} + x^{207} + x^{208} + x^{210} + \\
&x^{213} + x^{217} + x^{218} + x^{219} + x^{223} + x^{224} + x^{225} + x^{226} + x^{227} + x^{230} + x^{235} + \\
&x^{236} + x^{237} + x^{238} + x^{241} + x^{242} + x^{243} + x^{247} + x^{249} + x^{250} + x^{251} + x^{253} + \\
&x^{256} + x^{257} + x^{260} + x^{262} + x^{263} + x^{266} + x^{267} + x^{268} + x^{271} + x^{279} + x^{280} + \\
&x^{281} + x^{284} + x^{286} + x^{290} + x^{291} + x^{293} + x^{294} + x^{295} + x^{296} + x^{301} + x^{303} + \\
&x^{304} + x^{305} + x^{309} + x^{310} + x^{312} + x^{313} + x^{314} + x^{317} + x^{322} + x^{323} + x^{324} + \\
&x^{325} + x^{326} + x^{327} + x^{328} + x^{330} + x^{332} + x^{334} + x^{339} + x^{341} + x^{342} + x^{343} + \\
&x^{345} + x^{347} + x^{348} + x^{353} + x^{360} + x^{362} + x^{364} + x^{368} + x^{372} + x^{375} + x^{378} + \\
&x^{381} + x^{386} + x^{389} + x^{391} + x^{394} + x^{398} + x^{402} + x^{403} + x^{404} + x^{405} + x^{406} + \\
&x^{407} + x^{408} + x^{409} + x^{410} + x^{413} + x^{419} + x^{421} + x^{423} + x^{425} + x^{426} + x^{428} + \\
&x^{432} + x^{433} + x^{434} + x^{435} + x^{436} + x^{437} + x^{438} + x^{440} + x^{441} + x^{442} + x^{445} + \\
&x^{446} + x^{447} + x^{448} + x^{450} + x^{453} + x^{454} + x^{455} + x^{456} + x^{458} + x^{459} + x^{461} \\
&+ x^{465} + x^{466} + x^{467} + x^{469} + x^{470} + x^{471} + x^{472} + x^{474} + x^{475} + x^{476} + x^{477} + \\
&x^{483} + x^{489} + x^{491} + x^{492} + x^{493} + x^{494} + x^{495} + x^{496} + x^{499} + x^{500} + x^{501} + x^{504}
\end{aligned}
\tag{4.27}
$$

For $t = 54$

$$g(x) = 1 + x + x^3 + x^4 + x^5 + x^9 + x^{12} + x^{14} + x^{16} + x^{17} + x^{19} + x^{21} + x^{23} + x^{24} + x^{30} + x^{31} + x^{32} + x^{33} + x^{37} + x^{38} + x^{39} + x^{42} + x^{43} + x^{45} + x^{48} + x^{50} + x^{53} + x^{54} + x^{55} + x^{57} + x^{59} + x^{60} + x^{62} + x^{63} + x^{65} + x^{69} + x^{70} + x^{76} + x^{77} + x^{78} + x^{81} + x^{82} + x^{83} + x^{85} + x^{89} + x^{90} + x^{92} + x^{93} + x^{95} + x^{96} + x^{99} + x^{100} + x^{106} + x^{107} + x^{108} + x^{109} + x^{113} + x^{114} + x^{115} + x^{118} + x^{120} + x^{121} + x^{123} + x^{124} + x^{125} + x^{127} + x^{128} + x^{132} + x^{133} + x^{135} + x^{138} + x^{140} + x^{143} + x^{147} + x^{148} + x^{152} + x^{153} + x^{155} + x^{160} + x^{163} + x^{164} + x^{172} + x^{178} + x^{183} + x^{185} + x^{186} + x^{188} + x^{189} + x^{194} + x^{195} + x^{198} + x^{199} + x^{203} + x^{207} + x^{210} + x^{212} + x^{213} + x^{214} + x^{215} + x^{218} + x^{220} + x^{221} + x^{222} + x^{223} + x^{224} + x^{225} + x^{228} + x^{229} + x^{230} + x^{234} + x^{235} + x^{236} + x^{239} + x^{243} + x^{244} + x^{245} + x^{249} + x^{251} + x^{252} + x^{254} + x^{256} + x^{257} + x^{259} + x^{260} + x^{262} + x^{264} + x^{265} + x^{266} + x^{267} + x^{268} + x^{269} + x^{272} + x^{273} + x^{274} + x^{275} + x^{276} + x^{278} + x^{281} + x^{282} + x^{283} + x^{285} + x^{289} + x^{292} + x^{296} + x^{297} + x^{299} + x^{300} + x^{302} + x^{303} + x^{304} + x^{306} + x^{307} + x^{309} + x^{310} + x^{313} + x^{317} + x^{318} + x^{319} + x^{324} + x^{325} + x^{327} + x^{329} + x^{330} + x^{331} + x^{332} + x^{335} + x^{338} + x^{339} + x^{341} + x^{343} + x^{344} + x^{345} + x^{346} + x^{349} + x^{350} + x^{353} + x^{356} + x^{358} + x^{359} + x^{360} + x^{361} + x^{363} + x^{366} + x^{368} + x^{370} + x^{371} + x^{373} + x^{374} + x^{375} + x^{376} + x^{377} + x^{378} + x^{379} + x^{381} + x^{382} + x^{383} + x^{384} + x^{385} + x^{386} + x^{387} + x^{390} + x^{393} + x^{395} + x^{396} + x^{402} + x^{404} + x^{408} + x^{410} + x^{411} + x^{412} + x^{414} + x^{418} + x^{420} + x^{422} + x^{423} + x^{424} + x^{426} + x^{427} + x^{428} + x^{429} + x^{430} + x^{432} + x^{433} + x^{434} + x^{436} + x^{438} + x^{439} + x^{440} + x^{441} + x^{444} + x^{447} + x^{449} + x^{450} + x^{451} + x^{453} + x^{456} + x^{458} + x^{459} + x^{461} + x^{462} + x^{464} + x^{470} + x^{472} + x^{473} + x^{474} + x^{475} + x^{477} + x^{478} + x^{479} + x^{480} + x^{482} + x^{483} + x^{484} + x^{485} + x^{487} + x^{488} + x^{490} + x^{491} + x^{492} + x^{493} + x^{494} + x^{496} + x^{497} + x^{499} + x^{500} + x^{504} + x^{505} + x^{506} + x^{508} + x^{509} + x^{510} + x^{512} + x^{514} + x^{515} + x^{517} + x^{518} + x^{519} + x^{520} + x^{522} + x^{523} + x^{524} + x^{525} + x^{527} + x^{528} + x^{529} + x^{533} + x^{536} + x^{538} + x^{540} + x^{541} + x^{544} + x^{546} + x^{547} + x^{548} + x^{549} + x^{550} + x^{552} + x^{553} + x^{555} + x^{558} + x^{560} + x^{564} + x^{571} + x^{574} + x^{575} + x^{577} + x^{579} + x^{582} + x^{583} + x^{584} + x^{585} + x^{588} + x^{590} + x^{593} + x^{594} + x^{596} + x^{597} + x^{598} + x^{605} + x^{607} + x^{610} + x^{613} + x^{619} + x^{623} + x^{624} + x^{625} + x^{626} + x^{627} + x^{628} + x^{629} + x^{632} + x^{633} + x^{634} + x^{636} + x^{637} + x^{638} + x^{640} + x^{642} + x^{643} + x^{647} + x^{654} + x^{655} + x^{657} + x^{659} + x^{666} + x^{669} + x^{671} + x^{672} + x^{675} + x^{677} + x^{678} + x^{679} + x^{682} + x^{683} + x^{684} + x^{688} + x^{690} + x^{691} + x^{694} + x^{695} + x^{696} + x^{697} + x^{698} + x^{699} + x^{703} + x^{704} + x^{705} + x^{708} + x^{709} + x^{716} + x^{717} + x^{720} + x^{721} + x^{722} + x^{723} + x^{724} + x^{725} + x^{727} + x^{728} + x^{729} + x^{730} + x^{731} + x^{733} + x^{734} + x^{737} + x^{738} + x^{739} + x^{741} + x^{742} + x^{743} + x^{746} + x^{748} + x^{749} + x^{751} + x^{752} + x^{755} + x^{756} \quad (4.28)$$

4.3.3 Code synthesis for 16-level MLC

Consider 16-level MLC flash with memory model 1 architecture.

512 bytes (4096 16-ary cells) 4096 x 4 = 16384 bits	16 bytes 16x8x4= 512 bits

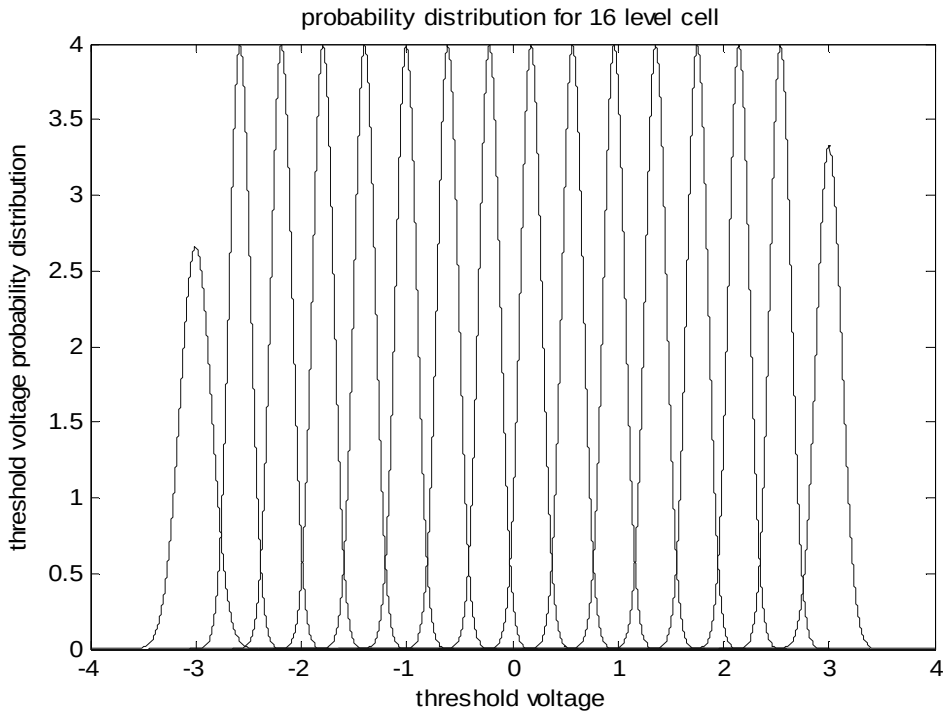


Figure 4.8: Probability distribution for 16-level cell

We have investigated that the best possible BCH code in 16 level flash memory with memory model 1 is the shortened (16894, 16384) over $F_{2^{15}}$ with error correcting capability t = 34. Here *degree* $\{g(x)\}$ = 510 = $n - k$, hence $n = k + 510 = 16384 + 510 = 16894$. Therefore the primitive BCH code parameters are

$$(32767,\ 32767-510) = (32767,\ 32257)$$

k = 16384. Hence, 32257 – 16384 = 15873. Therefore the parameters of the shortened BCH code are (16894, 16384). Since *degree* {*g(x)*} = 510 < 512, the overhead requirements of the code can be met by the Memory model 1. Hence, for t = 34, the parameters of shortened BCH code are n=16894, k=16384. The BCH code specified by these parameters can correct t = 34 errors over a span of one sector ($512\times4\times8$=16384 bits) in 16 level flash memory.

RBER can be computed as follows:

$$\begin{aligned}P(e) = {} & P(0)P(e/0)+P(1)P(e/1)+P(2)P(e/2)+P(3)P(e/3)+\\ & P(4)P(e/4)+P(5)P(e/5)+P(6)P(e/6)+P(7)P(e/7)+\\ & P(8)P(e/8)+P(9)P(e/9)+P(10)P(e/10)+P(11)P(e/11)+\\ & P(12)P(e/12)+P(13)P(e/14)+P(15)P(e/15)\end{aligned}$$

Assuming that symbols 0 – 16 can exist with equal probability in the cell,

$$P(e)=\frac{1}{8}\left[\begin{array}{l}P(e/0)+P(e/1)+P(e/2)+P(e/3)+P(e/4)+P(e/5)+\\ P(e/6)+P(e/7)+P(e/8)+P(e/9)+P(e/10)+P(e/11)+\\ P(e/12)+P(e/13)+P(e/14)+P(e/15)\end{array}\right] \tag{4.29}$$

Consider 16-level MLC flash with memory model 2 architecture.

512 bytes (4096 16-ary cells) 4096 x 4 = 16384 bits	32 bytes 32x8x4= 1024 bits

We investigated that the best possible BCH code in 16 level flash memory with memory model 2 is the shortened (17404, 16384) over $F_{2^{15}}$ with error correcting capability t = 68. Here *degree* {*g(x)*} = 1020 = $n - k$, hence $n = k + 1020 = 16384 + 1020 = 17404$. Therefore the primitive BCH code parameters are

$$(32767,\ 32767-1020) = (32767, 31747)$$

k = 16384, 31747– 16384 = 15404. Therefore the parameters of the shortened BCH code are (17404, 16384). Since *degree* $\{g(x)\}$ = 1020 < 1024, the overhead requirements of the code can be met by the Memory model 2. Hence, for t = 68, the parameters of shortened BCH code are n = 17404 and k = 16384. The BCH code specified by these parameters can correct t = 68 errors over a span of one sector ($512\times4\times8$=16384 bits) in 16-level flash memory. The performance plots of 16-level flash are shown in Fig.4.9 and Fig.4.10. The generator polynomial for t = 34 BCH code is computed and is shown in (4.30) and the generator polynomial for t = 68 BCH code is computed and is shown in (4.31).

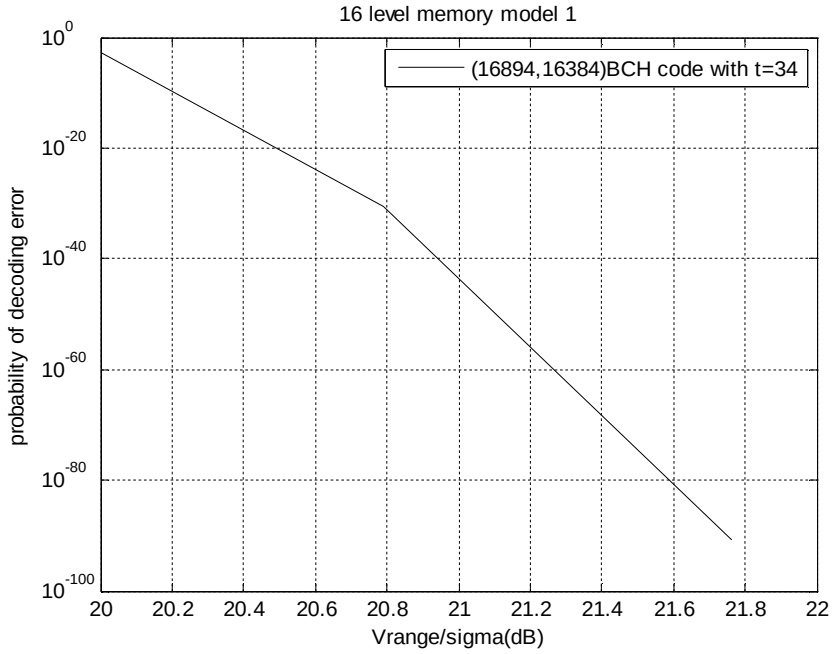


Figure 4.9: Performance of t = 34 BCH code for MLC (16-level) Memory model 1

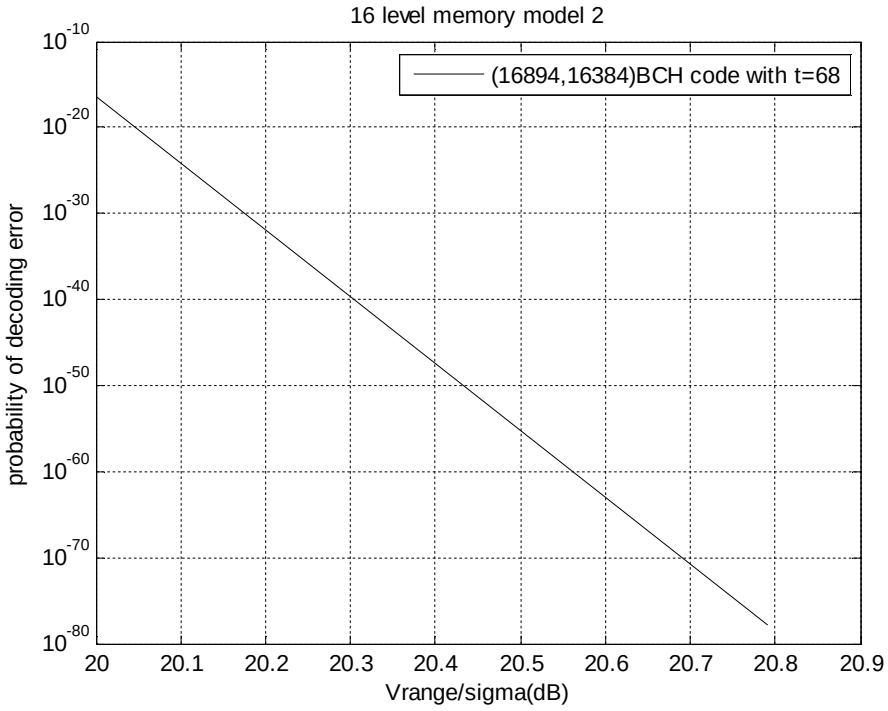


Figure 4.10: Performance of $t = 68$ BCH code for MLC (16-level) Memory model 2

$$g(x) = 1 + x + x^2 + x^3 + x^4 + x^{13} + x^{16} + x^{17} + x^{20} + x^{21} + x^{22} + x^{23} +$$
$$x^{24} + x^{25} + x^{35} + x^{36} + x^{38} + x^{39} + x^{42} + x^{44} + x^{45} + x^{46} + x^{47} + x^{48} +$$
$$x^{50} + x^{54} + x^{55} + x^{57} + x^{58} + x^{60} + x^{61} + x^{62} + x^{63} + x^{65} + x^{66} + x^{67} +$$
$$x^{68} + x^{69} + x^{70} + x^{72} + x^{73} + x^{74} + x^{75} + x^{76} + x^{80} + x^{81} + x^{82} + x^{85} +$$
$$x^{86} + x^{88} + x^{90} + x^{91} + x^{92} + x^{93} + x^{94} + x^{95} + x^{98} + x^{99} + x^{100} + x^{107} +$$
$$x^{108} + x^{109} + x^{110} + x^{111} + x^{112} + x^{122} + x^{123} + x^{124} + x^{125} + x^{127} + x^{128} +$$
$$x^{132} + x^{136} + x^{137} + x^{138} + x^{139} + x^{142} + x^{143} + x^{144} + x^{150} + x^{151} + x^{152} +$$
$$x^{153} + x^{154} + x^{155} + x^{156} + x^{157} + x^{160} + x^{161} + x^{162} + x^{165} + x^{168} + x^{169} +$$
$$x^{171} + x^{173} + x^{174} + x^{175} + x^{177} + x^{180} + x^{182} + x^{183} + x^{186} + x^{188} + x^{189} +$$
$$x^{190} + x^{191} + x^{192} + x^{193} + x^{194} + x^{196} + x^{198} + x^{203} + x^{207} + x^{212} + x^{214} +$$
$$x^{215} + x^{216} + x^{218} + x^{222} + x^{225} + x^{227} + x^{228} + x^{229} + x^{231} + x^{234} + x^{238} +$$
$$x^{239} + x^{240} + x^{243} + x^{244} + x^{247} + x^{249} + x^{250} + x^{252} + x^{253} + x^{255} + x^{262} +$$
$$x^{267} + x^{268} + x^{269} + x^{270} + x^{271} + x^{273} + x^{275} + x^{283} + x^{287} + x^{290} + x^{291} +$$
$$x^{292} + x^{294} + x^{296} + x^{301} + x^{303} + x^{304} + x^{305} + x^{307} + x^{308} + x^{309} + x^{311} +$$
$$x^{312} + x^{314} + x^{315} + x^{316} + x^{319} + x^{320} + x^{321} + x^{323} + x^{324} + x^{328} + x^{329} +$$
$$x^{331} + x^{332} + x^{334} + x^{335} + x^{336} + x^{337} + x^{339} + x^{341} + x^{342} + x^{346} + x^{347} +$$
$$x^{350} + x^{351} + x^{357} + x^{360} + x^{361} + x^{370} + x^{375} + x^{376} + x^{381} + x^{383} + x^{385} +$$
$$x^{388} + x^{391} + x^{392} + x^{393} + x^{394} + x^{395} + x^{399} + x^{400} + x^{402} + x^{403} + x^{404} +$$
$$x^{406} + x^{407} + x^{409} + x^{411} + x^{412} + x^{413} + x^{415} + x^{416} + x^{417} + x^{419} + x^{420} +$$
$$x^{422} + x^{423} + x^{424} + x^{425} + x^{426} + x^{429} + x^{434} + x^{437} + x^{438} + x^{440} + x^{441} +$$
$$x^{443} + x^{446} + x^{447} + x^{448} + x^{453} + x^{454} + x^{458} + x^{459} + x^{462} + x^{463} + x^{465} +$$
$$x^{470} + x^{471} + x^{473} + x^{476} + x^{481} + x^{482} + x^{486} + x^{487} + x^{489} + x^{490} + x^{491} +$$
$$x^{497} + x^{499} + x^{500} + x^{501} + x^{502} + x^{503} + x^{504} + x^{505} + x^{506} + x^{507} + x^{510} \quad (4.30)$$

$$g(x) = 1 + x^2 + x^5 + x^7 + x^8 + x^{12} + x^{13} + x^{14} + x^{22} + x^{24} + x^{26} + x^{28} +$$
$$x^{33} + x^{36} + x^{38} + x^{41} + x^{43} + x^{44} + x^{47} + x^{48} + x^{49} + x^{50} + x^{51} + x^{52} + x^{53} +$$
$$x^{54} + x^{58} + x^{60} + x^{61} + x^{62} + x^{64} + x^{65} + x^{66} + x^{67} + x^{69} + x^{72} + x^{74} + x^{78} +$$
$$x^{83} + x^{84} + x^{86} + x^{87} + x^{89} + x^{93} + x^{94} + x^{98} + x^{102} + x^{112} + x^{114} + x^{120} +$$
$$x^{123} + x^{124} + x^{126} + x^{127} + x^{131} + x^{132} + x^{133} + x^{136} + x^{137} + x^{138} + x^{140} +$$
$$x^{142} + x^{143} + x^{145} + x^{148} + x^{150} + x^{151} + x^{152} + x^{153} + x^{154} + x^{155} + x^{156} +$$
$$x^{159} + x^{163} + x^{164} + x^{165} + x^{166} + x^{168} + x^{170} + x^{171} + x^{172} + x^{179} + x^{180} +$$
$$x^{181} + x^{185} + x^{186} + x^{187} + x^{189} + x^{191} + x^{194} + x^{195} + x^{196} + x^{197} + x^{198} +$$
$$x^{199} + x^{200} + x^{202} + x^{206} + x^{210} + x^{211} + x^{215} + x^{216} + x^{217} + x^{218} + x^{221} +$$
$$x^{224} + x^{225} + x^{226} + x^{228} + x^{229} + x^{231} + x^{233} + x^{234} + x^{235} + x^{238} + x^{240} +$$
$$x^{243} + x^{247} + x^{248} + x^{251} + x^{253} + x^{255} + x^{259} + x^{260} + x^{262} + x^{263} + x^{264} +$$
$$x^{267} + x^{268} + x^{269} + x^{272} + x^{274} + x^{275} + x^{277} + x^{281} + x^{282} + x^{283} + x^{285} +$$
$$x^{286} + x^{287} + x^{292} + x^{293} + x^{294} + x^{295} + x^{296} + x^{300} + x^{301} + x^{303} + x^{305} +$$
$$x^{309} + x^{311} + x^{312} + x^{313} + x^{314} + x^{315} + x^{317} + x^{319} + x^{321} + x^{323} + x^{324} +$$
$$x^{325} + x^{326} + x^{327} + x^{329} + x^{331} + x^{332} + x^{333} + x^{336} + x^{337} + x^{338} + x^{339} +$$
$$x^{341} + x^{343} + x^{344} + x^{345} + x^{346} + x^{349} + x^{352} + x^{353} + x^{354} + x^{357} + x^{358} +$$
$$x^{360} + x^{363} + x^{366} + x^{367} + x^{368} + x^{369} + x^{371} + x^{372} + x^{374} + x^{375} + x^{378} +$$
$$x^{382} + x^{383} + x^{389} + x^{391} + x^{394} + x^{398} + x^{399} + x^{400} + x^{401} + x^{403} + x^{404} +$$
$$x^{405} + x^{406} + x^{407} + x^{408} + x^{409} + x^{411} + x^{415} + x^{417} + x^{420} + x^{421} + x^{424} +$$
$$x^{425} + x^{428} + x^{429} + x^{430} + x^{431} + x^{432} + x^{436} + x^{437} + x^{440} + x^{443} + x^{444} +$$
$$x^{445} + x^{446} + x^{448} + x^{449} + x^{451} + x^{452} + x^{453} + x^{454} + x^{455} + x^{456} + x^{457} +$$
$$x^{460} + x^{462} + x^{463} + x^{464} + x^{465} + x^{469} + x^{470} + x^{474} + x^{475} + x^{481} + x^{483} +$$
$$x^{487} + x^{488} + x^{490} + x^{491} + x^{494} + x^{497} + x^{500} + x^{502} + x^{504} + x^{505} + x^{508} +$$
$$x^{515} + x^{517} + x^{518} + x^{519} + x^{520} + x^{521} + x^{522} + x^{523} + x^{525} + x^{526} + x^{527} +$$
$$x^{532} + x^{533} + x^{535} + x^{536} + x^{537} + x^{539} + x^{540} \tag{4.31}$$

This chapter has been devoted to the synthesis of various BCH and RS codes which can be employed to protect information integrity in Multi level Cell (MLC) based Flash memories. The next chapter is focused on the decoder architecture and an insight to LDPC codes followed by interleavers.

This page is intentionally left blank

Chapter 5

Decoder Architecture and Interleaver

5.1 Decoder architecture

The iterative algorithm discovered by Berlekamp and Massey for the decoding of BCH codes is the best known technique for finding the error-locator polynomial [Chen, Y. and Parhi, K. 2004], [Sun, F. et al. 2006]. The computation of inverses in a finite field needed in the original Berlekamp – Massey method is a complex and time consuming exercise. An inversion less decoding method for binary BCH codes was proposed [Reed, I.S. et al. 1991] to simplify the Berlekamp – Massey algorithm. Here, we have adapted this algorithm to obtain decoding architectures which are suitable for use in Flash memory applications. The algorithm is briefly described below.

Let $\mu^{(0)}(D)$ be the error locator polynomial, $\lambda^{(0)}(D)$ be the previous polynomial, $\ell^{(0)}(D)$ be the length of LFSR and $\gamma^{(k)}$ be the scaling factor.

Let us assume the initial values to be $\mu^{(0)}(D) = 1$, $\lambda^{(0)}(D) = 1$, $\ell^{(0)}(D) = 0$, $\gamma^{(k)} = 1$.

The following recursive procedure is used to compute the LFSR. If the syndrome S_k is unknown, stop; otherwise define the discrepancy,

$$\delta^{(k+1)} = \sum_{j=0}^{\ell^{(k)}} \mu_j^{(k)} S_{k-j} \tag{5.1}$$

Using the knowledge of $\delta^{(k+1)}$, the polynomial $\mu^{(k)}(D)$ is updated as,

$$\mu^{(k+1)}(D) = \gamma^{(k)} \cdot \mu^{(k)}(D) - \delta^{(k+1)} \cdot \lambda^{(k)} \cdot (D) \cdot D \tag{5.2}$$

Further, the previous polynomial $\lambda^{(k)}(D)$ is updated as,

$$\lambda^{(k+1)}(D) = \begin{cases} D \cdot \lambda^{(k)}(D), & \delta^{(k+1)} = 0 \quad or \quad 2\ell^{(k)} > k \\ \mu^{(k)}(D), & \delta^{(k+1)} \neq 0 \quad and \quad 2\ell^{(k)} \leq k \end{cases} \tag{5.3}$$

Now the length $\ell^{(k)}$ is updated as,

$$\ell^{(k+1)} = \begin{cases} \ell^{(k)} & \delta^{(k+1)} = 0 \quad or \quad 2\ell^{(k)} > k \\ (k+1) - \ell^{(k)} & \delta^{(k+1)} \neq 0 \quad and \quad 2\ell^{(k)} \leq k \end{cases} \tag{5.4}$$

Finally, the scaling factor $\gamma^{(k)}$ is updated as,

$$\gamma^{(k+1)} = \begin{cases} \gamma^{(k)} & \delta^{(k+1)} = 0 \quad or \quad 2\ell^{(k)} > k \\ \delta^{(k+1)} & \delta^{(k+1)} \neq 0 \quad and \quad 2\ell^{(k)} \leq k \end{cases} \tag{5.5}$$

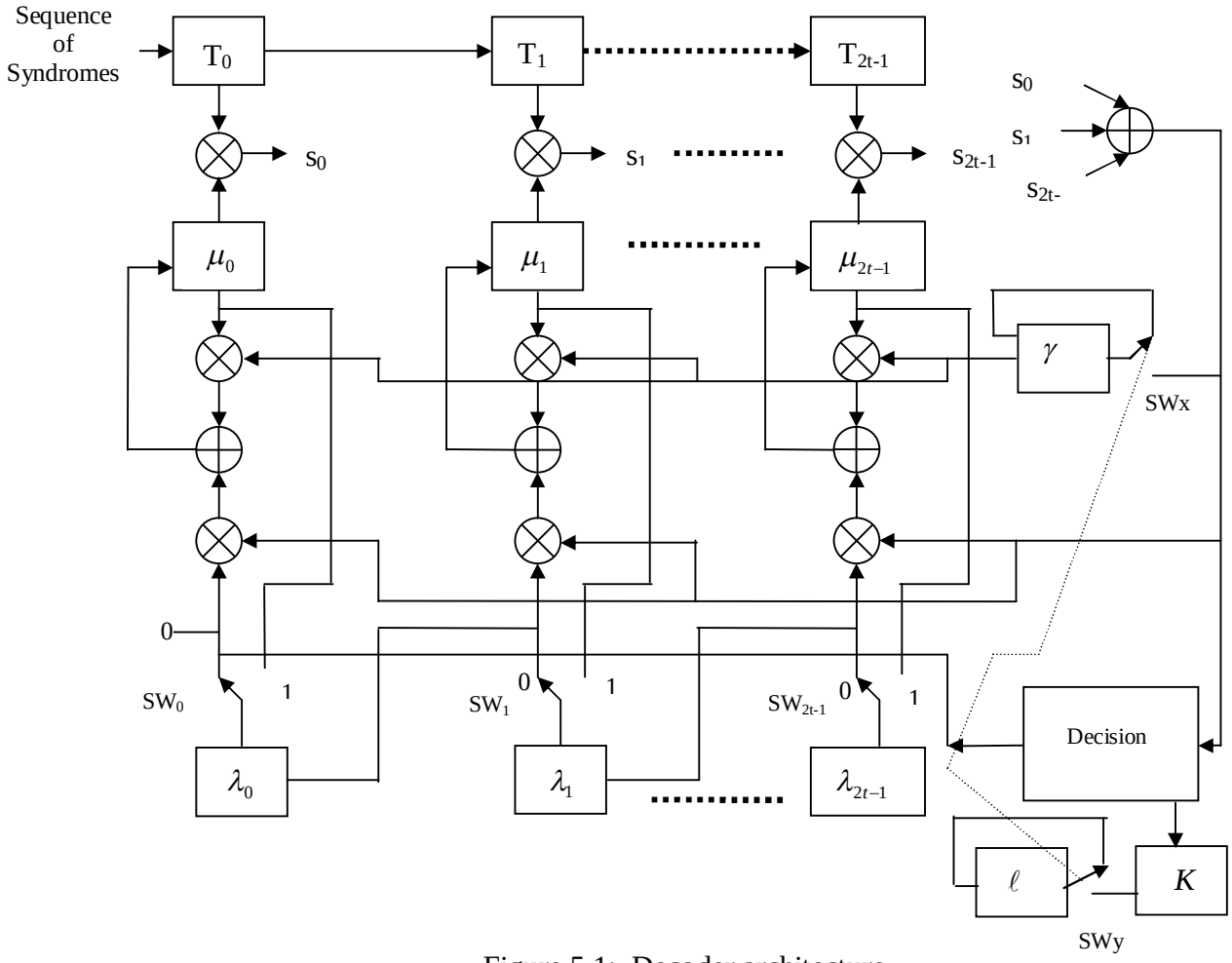


Figure 5.1: Decoder architecture

Figure 5.1 shows the decoder architecture. There are $2t+1$ switches in the circuit. Each switch can be in position A or B (position 0 or 1). The operation of the circuit is as follows:

(i) During initialization the switches are to be connected to position A. Once this is done the discrepancy is computed.

(ii) With switches SW_0, SW_1, ……., SW_{2t-1} in position A, update the value of the μ register. i.e. compute $\left(\mu_0^{(1)}, \mu_1^{(1)}, ..., \mu_{2t-1}^{(1)}\right)$

(iii) For the given value of k (now k = 0) check whether $2\ell^{(k)} \le k$. If $\delta^{(k+1)} \neq 0$ and $2l^{(k)} > k$, keep the position of the unchanged at position A. Upon applying the CLK signal, the contents of the λ register shifts one position to the right. Thus $\lambda^{(1)}(D) = D\lambda^{(0)}(D)$.

(iv) Switch SWx is in position A. Hence upon application of CLK, $\gamma^{(1)} = \gamma^{(0)}$.

(v) Switch SWy is in position A. Hence upon application of CLK, $l^{(1)} = l^{(0)}$.

(vi) On the other hand, if $\delta^{(k+1)} = 0$ or $2l^{(k)} \le k$, push switches to position B.

In this case,

$$\lambda_0^{(1)} = \mu_0^{(0)}$$
$$\lambda_1^{(1)} = \mu_1^{(0)}$$
$$\lambda_2^{(1)} = \mu_2^{(0)}$$

(vii) Switch SWy is shifted to position B. Hence $l^{(1)}$ is updated by $k+1-l^{(0)} = 0+1-1 = 1$. Thus $l^{(1)}$=1.

(viii) Switch SWx is shifted to position B. Hence, $\gamma^{(2)} = \delta^{(2)}$

This sequence is repeated.

The following example illustrates the working of this algorithm.

Example: Consider a t = 2 BCH code with length n = 15, over F_{2^4}. Choosing b = 1, $\delta = 2t+1 = 5$, $b+\delta-2 = 4$.

Therefore the required roots are: $\alpha, \alpha^2, \alpha^3, \alpha^4$.

Let the received word be $r(x) = x^7 + x^{13}$.

The syndromes are:

$$\begin{aligned} S_1 &= r(\alpha) = \alpha^7 + \alpha^{13} = \alpha^5 \\ S_2 &= r(\alpha^2) = \alpha^{14} + \alpha^{26} = \alpha^{10} \\ S_3 &= r(\alpha^3) = \alpha^{21} + \alpha^{39} = \alpha^5 \\ S_4 &= r(\alpha^4) = \alpha^{28} + \alpha^{52} = \alpha^5 \end{aligned} \tag{5.6}$$

When $k = 0$,

$$\delta^{(k+1)} = \delta^{(1)} = \sum_{j=0}^{l^{(0)}} \mu_j^{(k)} S_{b+k-j} = \sum_{j=0}^{0} \mu_j^{(0)} S_1 = 1 \cdot S_1 = \alpha^5 \neq 0$$

$$\mu^{(1)}(D) = \gamma^{(0)} \mu^0(D) - \delta^{(1)} \lambda^{(0)}(D) D = 1 \cdot 1 + \alpha^5 \cdot 1.D = 1 + \alpha^5 D$$

The conditions to be checked are:

(i) $\delta^{(k+1)} = 0 \quad or \quad 2\ell^{(k)} > k$

(ii) $\delta^{(k+1)} \neq 0 \quad and \quad 2\ell^{(k)} \leq k$

Since $\delta^{(1)} \neq 0$ *and* $2\ell^{(0)} = 0$ (i.e.) $\leq k$,

Previous polynomial $\lambda^{(1)}(D) = \mu^{(0)}(D) = 1$

Update of length $\ell^{(1)} = (0+1) - \ell^{(0)} = 1 - 0 = 1$

Scaling factor $\gamma^{(1)}(D) = \delta^{(1)} = \alpha^5$

When $k = 1$,

$$\delta^{(k+1)} = \delta^{(2)} = \sum_{j=0}^{l^{(1)}} \mu_j^{(k)} S_{b+k-j} = \sum_{j=0}^{1} \mu_j^{(0)} S_{2-j} = 1 \cdot \alpha^{10} + \alpha^5 \cdot \alpha^5 = 0$$

Since the discrepancy $\delta^{(2)} = 0$, $2\ell^{(1)} = 2$ i.e. $> k$

$$\mu^{(2)}(D) = \gamma^{(1)} \mu^{(1)}(D) - \delta^{(2)} \lambda^{(1)}(D) D = \alpha^5 (1 + \alpha^5 D) = \alpha^5 + \alpha^{10} D$$

$$\lambda^{(2)}(D) = D \lambda^{(1)}(D) = D$$

$$\ell^{(2)} = \ell^{(1)} = 1$$

$$\gamma^{(2)} = \gamma^{(1)} = \alpha^5$$

When $k = 2$,

$$\delta^{(3)} = \sum_{j=0}^{1} \mu_j^{(2)} S_{3-j} = 1 \neq 0$$

Since $\delta^{(3)} = 1 \neq 0$ and $2\ell^{(2)} = 2$ (i.e. $\leq k$),

$$\mu^{(3)}(D) = \gamma^{(2)}\mu^{(2)}(D) - \delta^{(3)}\lambda^{(2)}(D)D = \alpha^5(\alpha^5 + \alpha^{10}D) = \alpha^{10} + D + D^2$$
$$\lambda^{(3)}(D) = \lambda^{(2)}(D) = \alpha^5 + \alpha^{10}D$$
$$\ell^{(3)} = (k+1) - \ell^{(2)} = 3 - 1 = 2$$
$$\gamma^{(3)} = \delta^{(3)} = 1$$

When $k = 3$,

$$\delta^{(4)} = 0$$
$$\mu^{(4)}(D) = \gamma^{(3)}\mu^{(3)}(D) - \delta^{(4)}\lambda^{(3)}(D)D = 1(\alpha^{10} + D + D^2) = \alpha^{10} + D + D^2$$

Table 5.1 shows the various values obtained during each iteration in the above example.

Table 5.1: Values of various parameters

k	$\delta^{(k)}$	$\mu^{(k)}(D)$	$\lambda^{(k)}(D)$	$\ell^{(k)}$	$\gamma^{(k)}$
0	-	1	1	0	1
1	α^5	$1 + \alpha^5 D$	1	1	α^5
2	0	$\alpha^5 + \alpha^{10} D$	D	1	α^5
3	1	$\alpha^{10} + D + D^2$	$\alpha^5 + \alpha^{10} D$	2	1
4	0	-	-	-	-

Therefore, the error location polynomial is $\mu(D) = \mu^{(4)}(D) = \alpha^{10} + D + D^2$

$$\mu(\alpha^2) = \alpha^{10} + \alpha^2 + \alpha^4 = 0$$
$$\mu(\alpha^3) = \alpha^{10} + \alpha^3 + \alpha^6 = \alpha^4$$
$$\mu(\alpha^4) = \alpha^{10} + \alpha^4 + \alpha^8 = 1$$
$$\mu(\alpha^5) = \alpha^{10} + \alpha^5 + \alpha^{10} = \alpha^5$$
$$\mu(\alpha^6) = \alpha^{10} + \alpha^6 + \alpha^{12} = \alpha^2$$
$$\mu(\alpha^7) = \alpha^{10} + \alpha^7 + \alpha^{14} = \alpha^8$$
$$\mu(\alpha^8) = \alpha^{10} + \alpha^8 + \alpha^{16} = 0$$

Therefore, α^2 and α^8 are zeros of $\mu(D)$. These points to errors at locations x^{13} and x^2 respectively.

$$\therefore e(x) = x^7 + x^{13} \tag{5.7}$$

5.2 Code synthesis using LDPC Codes

Low-density parity-check (LDPC) codes are a class of linear block codes. The name comes from the characteristic of their parity-check matrix which contains only a few 1's in comparison to the number of 0's. Their main advantage is that they provide a performance which is very close to the capacity for a lot of different channels and linear time complex algorithms for decoding. Furthermore they are suited for implementations that make heavy use of parallelism.

They were first introduced by Gallager in his Ph.D thesis in 1960. But due to the computational effort in implementing coder and encoder for such codes and the introduction of Reed-Solomon codes, they were mostly ignored until about fifteen years ago.

The feature of LDPC codes [Ryan, W. E. and Lin, S] to perform near the Shannon limit of a channel exists only for large block lengths. For example there have been simulations that perform within 0.04 dB of the Shannon limit at a bit error rate of 10^{-6} with block length of 10^7. An interesting fact is that those high performance codes are irregular. The large block length results also in large parity-check and generator matrices. The complexity of multiplying a codeword with a matrix depends on the amount of 1's in the matrix. If we put the sparse matrix $\mathbf{H}$ in the form $[\mathbf{P}^T \mathbf{I}]$ via Gaussian elimination the generator matrix $\mathbf{G}$ can be calculated as $\mathbf{G} = \mathbf{[I\ P]}$. The sub-matrix $\mathbf{P}$ is generally not sparse so that the encoding complexity will be quite high. Since the complexity grows, even sparse matrices do not result in a good performance if the block length gets very high. So iterative decoding (and encoding) algorithms are used. Those algorithms perform local calculations and pass those local results via messages. This step is typically repeated several times. The term "local calculations" already indicates that a divide and

conquer strategy, which separates a complex problem into manageable sub-problems, is realized. A sparse parity-check matrix now helps this algorithm in several ways. First it helps to keep both the local calculations simple and also reduces the complexity of combining the sub-problems by reducing the number of needed messages to exchange all the information. Furthermore, it is observed that iterative decoding algorithms of sparse codes perform very close to the optimal Maximum Likelihood (ML) decoder.

Basically there are two different possibilities to represent LDPC codes. Like all linear block codes they can be described via matrices. The second possibility is a graphical representation.

Matrix Representation

The following matrix is an example for a low-density parity-check matrix. The matrix defined in equation (5.8) is a parity-check matrix with dimension $n \times m$ for a (8, 4) code. We can now define two numbers describing this matrix. W_r for the number of 1's in each row and W_c for the columns. For a matrix to be called low-density the two conditions $W_c << n$ and $W_r << m$ must be satisfied.

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix} \tag{5.8}$$

Graphical Representation

Tanner introduced an effective graphical representation for LDPC codes. Tanner graphs are bipartite graphs. That means that the nodes of the graph are separated into two distinctive sets and edges are only connecting nodes of two different types. The two types

of nodes in a Tanner graph are called variable nodes (*v*-nodes) and check nodes (*c*-nodes). Figure 5.2 is an example for such a Tanner graph and represents the same code as the matrix in 5.8. The creation of such a graph is rather straight forward. It consists of *m* check nodes (the number of parity bits) and *n* variable nodes (the number of bits in a codeword). Check node f_i is connected to variable node c_j if the element h_{ij} of **H** is a 1. The marked path $c_2 \to f_1 \to c_5 \to f_2 \to c_2$ is an example for a short cycle. Those should usually be avoided since they are bad for decoding performance.

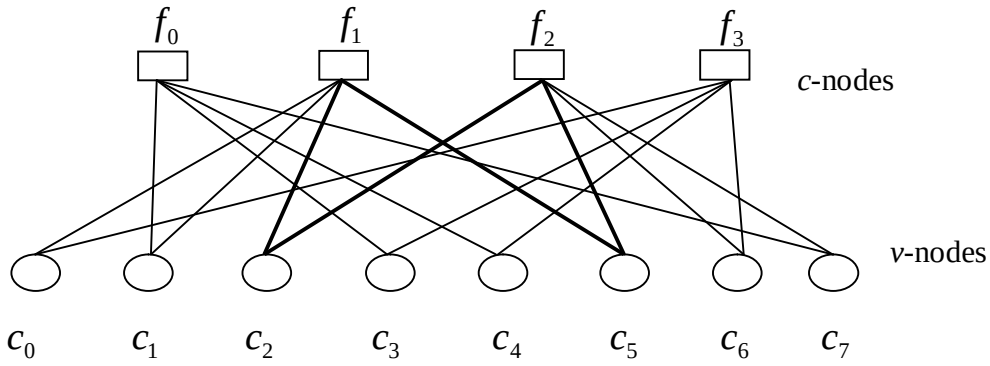


Figure 5.2: Tanner graph corresponding to the parity check matrix in Equation (5.8).

Regular and irregular LDPC Codes

A LDPC code is called regular if W_c is constant for every column and $W_r = W_c$ is also constant for every row. The example matrix from equation (5.8) is regular with $W_c = 2$ and $W_r = 4$. It is also possible to see the regularity of this code while looking at the graphical representation. There is the same number of incoming edges for every *v*-node and also for all the *c*-nodes. If **H** is low density but the numbers of 1's in each row or column aren't constant the code is called a irregular LDPC code.

Gallager construction of parity check matrix (H)

In this method of construction, the parity check matrix (**H***)* is constructed as follows. The original parity check matrix (**H**) is divided into 'W_c' sub-matrices as follows:

$$\mathbf{H} = \begin{bmatrix} \mathbf{H_1} \\ \hline \mathbf{H_2} \\ \hline \mathbf{H_3} \\ \hline \\ \\ \\ \hline \mathbf{H_{W_c}} \\ \hline \end{bmatrix}$$

The sub-matrix $\mathbf{H_1}$ is given by

Number of one's in each row = W_r

$$\begin{bmatrix} 1\,1 \ldots\ldots\ldots 1\,1\,0\,0\,0\,0\,0\,0\,0 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots .0\,0\,0\,0 \\ 0\,0 \ldots\ldots\ldots 0\,0\,1\,1 \ldots\ldots\ldots .1\,1\,0\,0\,0\,0 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 0\,0\,0\,0 \\ \\ \\ \\ \\ 0\,0\,0\,0\,0 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots .1\,1 \ldots\ldots 1\,0\,0 \ldots\ldots\ldots\ldots .0\,0 \\ 0\,0\,0\,0\,0 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots .0\,0\,0\,0\,1\,1 \ldots\ldots\ldots 1\,1 \end{bmatrix}$$

The matrix $\mathbf{H}_2$ is random permutations of columns of $\mathbf{H}_1$

The matrix $\mathbf{H}_3$ is another random permutation of columns of $\mathbf{H}_1$

The matrix $\mathbf{H}_{W_c}$ is another random permutation of columns of $\mathbf{H}_1$

In this way Parity check matrix can be constructed using Gallager approach.

5.3 Application of LDPC Codes to Flash Memories

We synthesized regular LDPC codes for the available standard memory models. In memory model 1, each sector has 512 bytes reserved for storing information and 16 bytes reserved for storing parity check information. In memory model 2, each sector has 512 bytes reserved for storing information and 32 bytes reserved for storing parity check (redundant) information.

Regular LDPC codes are characterized by three parameters.

(i) Block length (n)

(ii) Each column of parity check matrix should have the same weight W_c

(iii) Each row of parity check matrix should have the same weight W_r

5.3.1 Code synthesis for Memory Model 1

For this model, we have number of information bits/sector k =512 bytes = 4096 bits.

Number of Parity bits/sector $(n-k)$ = 16 bytes = 128 bits.

Total number of bits/sector = 528 bytes = 4224 bits.

For LDPC codes,

$$\text{Rate of the code} \quad R = \frac{k}{n} = 1 - \frac{W_c}{W_r}$$

For the given model, n = 4224 bits and k = 4096 bits. Hence,

$$1 - \frac{W_c}{W_r} = \frac{4096}{4224}$$

Simplifying the above equation

$$==> \quad \frac{W_c}{W_r} = 1 - \frac{4096}{4224} \tag{5.9}$$

$$==> \quad W_r = 33W_c$$

Assuming $W_c = 4$, $\quad W_r = 33W_c = 33 \times 4 = 132$

Therefore the code parameters are:

Block length n = 4224,

Weight of each column of parity check matrix $W_c = 4$

Weight of each row of parity check matrix $W_r = 132$

Hence we have (4224, 4, 132) regular LDPC code. We have generated full rank sparse parity check matrix of order 128×4224 for above specified code using Gallager construction of parity check matrix

5.3.2 Code synthesis for Memory Model 2

For memory model 2, we have number of information bits/sector k = 512 bytes = 4096 bits.

Number of Parity bits/sector $(n-k)$ = 32 bytes = 256 bits.

Total number of bits/sector = 544 bytes = 4352 bits.

We have

$$\text{Rate of the code } R = \frac{k}{n} = 1 - \frac{W_c}{W_r}$$

For the given model, n = 4352 bits and k = 4096 bits

Hence we have $1 - \frac{W_c}{W_r} = \frac{4096}{4352}$

$$==> \quad \frac{W_c}{W_r} = 1 - \frac{4096}{4352} \tag{5.10}$$

$$==> \quad W_r = 17W_c$$

Assuming $W_c = 4, \quad W_r = 17 \times 4 = 68$

Therefore the code parameters are

Block length n = 4352

Weight of each column of parity check matrix $W_c = 4$

Weight of each row of parity check matrix $W_r = 68$

Hence we have (4352, 4, 68) regular LDPC code. We have generated full rank sparse parity check matrix of order 256×4352 for above specified code using Gallager construction of parity check matrix.

5.3.3 Code Synthesis for Multi Level Flash Memories

4-level MLC with Memory Model 1

Here each cell will hold 2 bits and hence by taking one sector at a time to synthesize regular LDPC codes we have number of information bits/sector = 8192 and number of parity bits/sector = 256. Since the ratio of information bits to the total bits remains unchanged, the condition between row weight W_r and column weight W_c of parity check matrix remains same as single level cell (SLC) i.e. $W_r = 33W_c$. Assuming $W_c = 4$ which gives $W_r = 132$ and total bits n = 8448. Hence this results (8448, 4, 132) regular LDPC code and the order of parity check matrix is 256×8448.

8-level MLC with Memory Model 1

In this case each cell will hold 3 bits and hence by taking one sector at a time to synthesize regular LDPC code we have number of information bits/sector = 12288 and number of parity bits/sector = 384. Since the ratio of information bits to the total bits remains unchanged, the condition between row weight W_r and column weight W_c remains of parity check matrix remains same as single level cell(SLC) i.e. $W_r = 33 \times W_c$. Assuming W_c = 4 which gives W_r = 132 and total bits n = 12672. Hence this results (12672, 4, 132) regular LDPC code and the order of parity check matrix is 384×12672.

16-level MLC with Memory Model 1

In this case each cell will hold 4 bits and hence by taking one sector at a time to synthesize regular LDPC code we have number of information bits/sector=16384 and number of parity bits/sector=512. Since the ratio of information bits to the total bits remains unchanged, the condition between row weight W_r and column weight W_c of parity check matrix remains same as SLC i.e. $W_r = 33 \times W_c$. Assuming $W_c = 4$ which gives $W_r = 132$ and total bits n = 16896. Hence this results (16896, 4, 132) regular LDPC code and the order of parity check matrix is 512×16896. We have generated full rank sparse parity check matrices for all the above specified codes using Gallager construction of parity check matrix.

4-level MLC Memory Model 2

In this case each cell will hold 2 bits and hence by taking one sector at a time to synthesize regular LDPC code we have number of information bits/sector = 8192 and number of parity bits/sector = 512. Since the ratio of information bits to the total bits remains unchanged, the condition between row weight W_r and column weight W_c of parity check matrix remains same as SLC i.e. $W_r = 17 \times W_c$. Assuming $W_c = 4$ which gives $W_r = 68$ and total bits n = 8704. Hence this results (8704, 4, 68) regular LDPC code and the order of matrix is 512×8704.

8-level MLC Memory Model 2

In this case each cell will hold 3 bits and hence by taking one sector at a time to synthesize regular LDPC code we have number of information bits/sector = 12288 and number of parity bits/sector = 768. Since the ratio of information bits to the total bits remains unchanged, the condition between row weight W_r and column weight W_c of parity check matrix remains same as SLC i.e. $W_r = 17 \times W_c$. Assuming $W_c = 4$ gives

$W_r = 68$ and total bits n =13056. Hence this results (13056, 4, 68) regular LDPC code and the order of matrix is 768×13056.

16-level MLC Memory Model 2

In this case each cell will hold 4 bits and hence by taking one sector at a time to synthesize regular LDPC code we have number of information bits/sector = 16384 and number of parity bits/sector = 1024. Since the ratio of information bits to the total bits remains unchanged, the condition between row weight W_r and column weight W_c of parity-check matrix remains same as SLC i.e. $W_r = 17 \times W_c$. Assuming $W_c = 4$ which gives $W_r = 68$ and total bits n = 17408. Hence this results (17408, 4, 68) regular LDPC code and the order of matrix is 1024×17408.

We have generated full rank sparse parity check matrices for all the above specified codes using Gallager construction of parity check matrix.

5.4 Interleavers

A bursty channel is defined as a channel over which errors tend to occur in bunches, or "bursts", as opposed to random patterns associated with a Bernoulli distributed process. Bursty channels usually contain some error causing agent in the physical medium whose effective time constant exceeds the symbol transmission rate of the channel. For example a scratch on a Compact Disc may obscure several consecutive bits on each of the adjacent tracks, thus causing multiple error bursts when the disc is played.

Most binary block codes have been designed as random error correcting. A random error correcting code can correct up to t symbol errors per code word, regardless of the placement of those errors. A problem arises with these codes whenever the channel encountered in the application is bursty. An error burst focuses several symbol errors within a small number of received codewords, while the other codewords may not be corrupted by any errors at all. The performance of convolutional code is also sensitive to

bursty channels. In the decoding of convolutional codes, error event occur whenever the received codeword is closer to an incorrect code word than the codeword that was transmitted. A convolutional code may be able to correct an arbitrarily large number of well spaced-errors, while at the same time unable to handle a short burst.

An Interleaver is a device that rearranges the ordering of a sequence of symbols in some one-to-one deterministic manner. Alternately, an Interleaver [Wicker, S. B. 1995] can be viewed as device that mixes up the symbols from several codewords so that the symbols from any given code word are well separated during transmission. Associated with any interleaver is a de-interleaver, which is the device that restores the reordered sequence to its original ordering. When the codewords are reconstructed by the deinterleaver, error bursts introduced by the channel are broken up and spread across several codewords. The interleaver/deinterleaver pair thus creates an effectively random channel. Interleavers and deinterleavers have a variety of applications in cryptography and communication technology.

In many of the applications to communication technology, interleaving is used as an adjunct to coding for error correction. One technique which is useful for some types of burst error channels is to insert an interleaver between the channel encoder and the channel. The interleaver redistributes the channel symbols so that the symbols from a codeword are mutually separated by somewhat more than the length of a typical burst of errors. Thus, interleaving effectively makes the channel appear like a random error channel to the decoder.

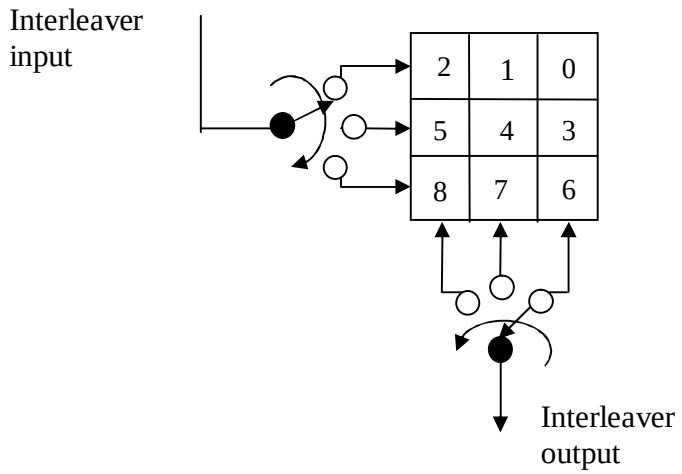
Block and cross-interleave are the most frequently used types of interleavers. The cross-interleave is sometimes called a convolutional or periodic interleaver. An $(n \times m)$ block interleaver and the corresponding deinterleaver are shown in Figure. The two circuits are identical, each consisting of n rows of m memory elements. The coded data stream is read into the block interleaver rows in the order noted in the figure. The interleaver

contents are then read by columns. Any two adjacent symbols at the input are thus separated by $(n-1)$ other symbols at the output. The row length *m* is frequently selected so that each row holds an entire codeword. A burst of *b* errors causes a maximum of $\left\lceil \frac{b}{n} \right\rceil$ errors to occur in one or more codewords. The efficiency of an interleaver can be measured in a number of ways. Efficiency γ may be defined to be the ratio of the length of the smallest burst of errors that can cause the error correcting capability *t* of the code to be exceeded to the number of memory elements used in the interleaver. Thus, for an $(n \times m)$ block interleave we have $\gamma = (nt+1)/nm \approx t/m$.

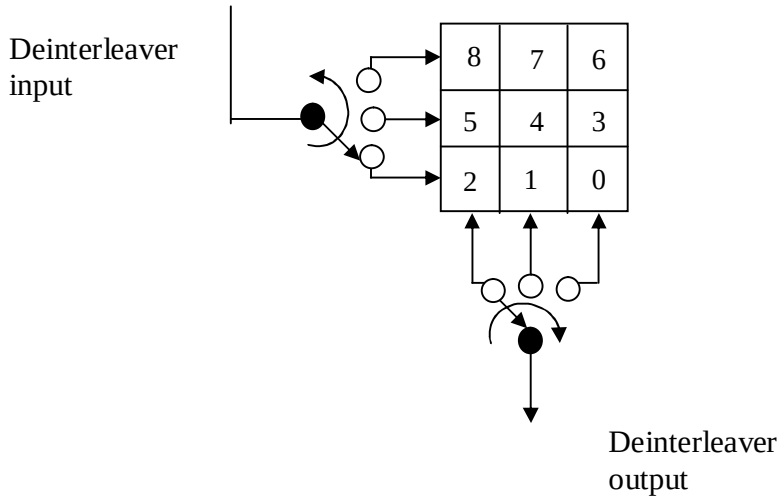
A cross-interleave circuit is shown in Figure 5.3. The circuit is characterized by the index *m*, the number of delay lines. Each block *D* corresponds to a *D*-symbol delay. The input symbols are read onto the delay lines in the order shown in the Figure 5.3. The output of the delay line is read in the same order. Consider a pair of consecutive input symbols x_0 and x_1. These two symbols are placed on adjacent delay lines one with delay *tD* and the other with delay $(t+1)D$. When x_0 reaches the output position of its delay line, x_1 will still be *D* delay elements short of the output of its own line. After x_0 is read, all of the *m* delay line outputs are read *D* times before x_1 is output. Thus there are *mD* symbols separating adjacent codeword symbols at the output of the interleaver.

Suppose that *m* is chosen so that it equals or exceeds the length of the codewords. Each symbol in a codeword is placed on a different delay line. At the output of the interleaver, the codeword symbols will appear in order, with each symbol separated from its neighbor(s) in the codeword by *mD* symbols from *mD* other codewords.

$\ldots x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, \ldots$



$\ldots x_6, x_3, x_0, x_7, x_4, x_1, x_8, x_5, x_2, x_{15}, x_{12}, x_9, x_{16}, x_{13}, \ldots$



$\ldots x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, \ldots$

Figure 5.3: A 3×3 Block Interleaver and Deinterleaver

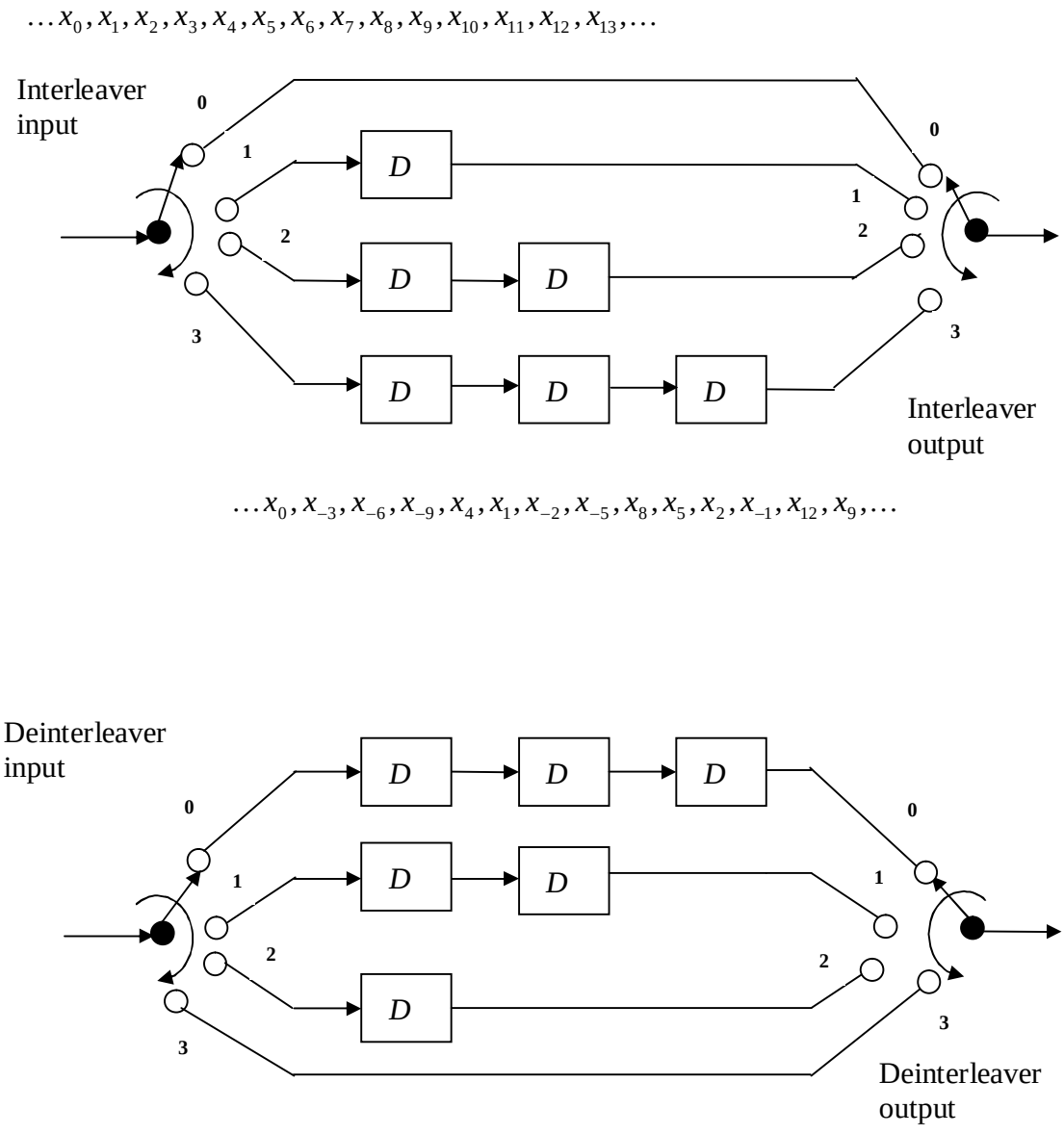


Figure 5.4: A Cross Interleave circuit and corresponding Deinterleaver (m = 4, D =1 symbol)

A length *b* burst of errors may thus cause $\lceil b/(mD+1) \rceil$ errors in one or more codewords. Given a *t*-error correcting code, decoder errors are possible when the length of the error bursts equals or exceeds $(mD+1)(t-1)+1$.

The efficiency of the cross-interleaver is [Wicker, S.B. 1995]]

$$\gamma = \frac{(mD+1)(t-1)+1}{[0+1+2+\cdots\cdots+(m-1)]D} = \frac{mD(t-1)+t}{\left\lceil \frac{m(m-1)D}{2} \right\rceil} \approx \frac{2t}{m-1} \qquad (5.11)$$

The cross-interleaver is thus slightly more than twice as efficient as the block interleaver.

Let us consider a (260, 255) RS code over F_{2^8}. Let t = 2. Therefore, the design distance $\delta = 2t+1 = 5$. This code is shortened to (132, 128) over F_{2^8}. As per memory model 1, one sector holds 4096 bits of information (512 bytes) and 128 bits redundancy (16 bytes). Each sector is now divided into four partitions, with $n - k$ = 4 bytes.

128 bytes	4 bytes
128 bytes	4 bytes
128 bytes	4 bytes
128 bytes	4 bytes

We see that each partition has 128 information bytes and 4 bytes of overhead. Let the contents of these partitions be respectively represented by P_0, P_1, P_2, P_3. The symbols (bytes) contained in these partitions may be represented as,

$$\begin{aligned}
P_0 &= v_{0,0}, v_{0,1}, v_{0,2}, \cdots\cdots, v_{0,127}, v_{0,128}, v_{0,129}, v_{0,130}, v_{0,131} \\
P_1 &= v_{1,0}, v_{1,1}, v_{1,2}, \cdots\cdots\cdots, v_{1,127}, v_{1,128}, v_{1,129}, v_{1,130}, v_{1,131} \\
P_2 &= v_{2,0}, v_{2,1}, v_{2,2,}, \cdots\cdots\cdots, v_{2,127}, v_{2,128}, v_{2,129}, v_{2,130}, v_{2,131} \\
P_3 &= v_{3,0}, v_{3,1}, v_{3,2} \cdots\cdots\cdots, v_{3,127}, v_{3,128}, v_{3,129}, v_{3,130}, v_{3,131}
\end{aligned} \qquad (5.12)$$

While writing data into the memory, we pass it through a block interleaver which reorders the data and writes into the memory in the following way.

$$[v_{0,0}, v_{1,0}, v_{2,0}, v_{3,0} \vdots v_{0,1}, v_{1,1}, v_{2,1}, v_{3,1} \vdots \cdots\cdots\cdots \vdots v_{0,127}, v_{1,127}, v_{2,127}, v_{3,127},$$
$$v_{0,128}, v_{1,128}, v_{2,128}, v_{3,128}, \vdots \cdots\cdots\cdots\cdots \vdots v_{0,131}, v_{1,131}, v_{2,131}, v_{3,131}] \quad (5.13)$$

Let us assume that an error burst involving eight consecutive bytes in a sector has occurred. As an example, let us consider that $v_{0,0}, v_{1,0}, v_{2,0}, v_{3,0} \vdots v_{0,1}, v_{1,1}, v_{2,1}, v_{3,1}$ are in error. Now, while reading from the memory these symbols are read in the following manner.

$$\underline{v_{0,0}, v_{0,1}}, v_{0,2}, \cdots\cdots\cdots, v_{0,127}, v_{0,128}, v_{0,129}, v_{0,130}, v_{0,131}$$
$$\underline{v_{1,0}, v_{1,1}}, v_{1,2}, \cdots\cdots\cdots, v_{1,127}, v_{1,128}, v_{1,129}, v_{1,130}, v_{1,131}$$
$$\underline{v_{2,0}, v_{2,1}}, v_{2,2}, \cdots\cdots\cdots, v_{2,127}, v_{2,128}, v_{2,129}, v_{2,130}, v_{2,131}$$
$$\underline{v_{3,0}, v_{3,1}}, v_{3,2}, \cdots\cdots\cdots, v_{3,127}, v_{3,128}, v_{3,129}, v_{3,130}, v_{3,131} \quad (5.14)$$

It is observed that even though eight consecutive bytes per sector are in error, after deinterleaving these erroneous symbols are redistributed so that each code word contains only two erroneous symbols. These errors can be easily corrected by the RS code. However, we should note that a burst of length containing nine or more symbols in error after redistribution by the deinterleaver will result in erroneous decoding because under these circumstances, it cannot be guaranteed that the number of symbols in error in each codeword is less than or equal to two.

Let us now consider the case where we increase the interleaving depth to accommodate two sectors (as represented in Figure 5.5). Let the codewords contained in these sectors be labeled as $C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7$. These code words can be represented as shown in (5.15)

128 bytes	4 bytes
128 bytes	4 bytes
128 bytes	4 bytes
128 bytes	4 bytes

128 bytes	4 bytes
128 bytes	4 bytes
128 bytes	4 bytes
128 bytes	4 bytes

Figure 5.5: Representation of two sectors, each with 512 bytes of data and 16 bytes of overhead

$$
\begin{aligned}
C_0 &= v_{0,0}, v_{0,1}, v_{0,2}, v_{0,3}, v_{0,4}, v_{0,5}, \cdots\cdots\cdots, v_{0,127}, v_{0,128}, \cdots, v_{0,131} \\
C_1 &= v_{1,0}, v_{1,1}, v_{1,2}, v_{1,3}, v_{1,4}, v_{1,5}, \cdots\cdots\cdots, v_{1,127}, v_{1,128}, \cdots, v_{1,131} \\
C_2 &= v_{2,0}, v_{2,1}, v_{2,2}, v_{2,3}, v_{2,4}, v_{2,5}, \cdots\cdots\cdots, v_{2,127}, v_{2,128}, \cdots, v_{2,131} \\
C_3 &= v_{3,0}, v_{3,1}, v_{3,2}, v_{3,3}, v_{3,4}, v_{3,5}, \cdots\cdots\cdots, v_{3,127}, v_{3,128}, \cdots, v_{3,131} \\
C_4 &= v_{4,0}, v_{4,1}, v_{4,2}, v_{4,3}, v_{4,4}, v_{4,5}, \cdots\cdots\cdots, v_{4,127}, v_{4,128}, \cdots, v_{4,131} \\
C_5 &= v_{5,0}, v_{5,1}, v_{5,2}, v_{5,3}, v_{5,4}, v_{5,5}, \cdots\cdots\cdots, v_{5,127}, v_{5,128}, \cdots, v_{5,131} \\
C_6 &= v_{6,0}, v_{6,1}, v_{6,2}, v_{6,3}, v_{6,4}, v_{6,5}, \cdots\cdots\cdots, v_{6,127}, v_{6,128}, \cdots, v_{6,131} \\
C_7 &= v_{7,0}, v_{7,1}, v_{7,2}, v_{7,3}, v_{7,4}, v_{7,5}, \cdots\cdots\cdots, v_{7,127}, v_{7,128}, \cdots, v_{7,131}
\end{aligned} \tag{5.15}
$$

These symbols are interleaved and written into memory in the following sequences.

$$
\begin{aligned}
&v_{0,0}, v_{1,0}, v_{2,0}, v_{3,0}, v_{4,0}, v_{5,0}, v_{6,0}, v_{7,0}, \vdots v_{0,1}, v_{1,1}, v_{2,1}, v_{3,1}, v_{4,1}, v_{5,1}, v_{6,1}, v_{7,1} \vdots \\
&v_{0,2}, v_{1,2}, v_{2,2}, v_{3,2}, v_{4,2}, v_{5,2}, v_{6,2}, v_{7,2}, \vdots \cdots\cdots, v_{0,127}, v_{1,127}, v_{2,127}, v_{3,127}, v_{4,127}, \\
&v_{5,127}, v_{6,127}, v_{7,127}, \vdots v_{0,128}, v_{1,128}, v_{2,128}, v_{3,128}, v_{4,128}, v_{5,128}, v_{6,128}, v_{7,128}, \vdots \cdots\cdots\cdots \vdots \\
&v_{0,131}, v_{1,131}, v_{2,131}, v_{3,131}, v_{4,131}, v_{5,131}, v_{6,131}, v_{7,131} \vdots
\end{aligned} \tag{5.16}
$$

If there are sixteen consecutive bytes in errors say

$$
v_{0,1}, v_{1,1}, v_{2,1}, v_{3,1}, v_{4,1}, v_{5,1}, v_{6,1}, v_{7,1}, \vdots v_{0,2}, v_{1,2}, v_{2,2}, v_{3,2}, v_{4,2}, v_{5,2}, v_{6,2}, v_{7,2} \tag{5.17}
$$

then upon reading from the memory (deinterleaving) it is observed that there are only two bytes of error per codeword.

$$
\begin{aligned}
C_0 &= v_{0,0}, \underline{v_{0,1}, v_{0,2}}, v_{0,3}, v_{0,4}, v_{0,5}, \cdots\cdots\cdots, v_{0,127}, v_{0,128}, \cdots, v_{0,131} \\
C_1 &= v_{1,0}, \underline{v_{1,1}, v_{1,2}}, v_{1,3}, v_{1,4}, v_{1,5}, \cdots\cdots\cdots, v_{1,127}, v_{1,128}, \cdots, v_{1,131} \\
C_2 &= v_{2,0}, \underline{v_{2,1}, v_{2,2}}, v_{2,3}, v_{2,4}, v_{2,5}, \cdots\cdots\cdots, v_{2,127}, v_{2,128}, \cdots, v_{2,131} \\
C_3 &= v_{3,0}, \underline{v_{3,1}, v_{3,2}}, v_{3,3}, v_{3,4}, v_{3,5}, \cdots\cdots\cdots, v_{3,127}, v_{3,128}, \cdots, v_{3,131} \\
C_4 &= v_{4,0}, \underline{v_{4,1}, v_{4,2}}, v_{4,3}, v_{4,4}, v_{4,5}, \cdots\cdots\cdots, v_{4,127}, v_{4,128}, \cdots, v_{4,131} \\
C_5 &= v_{5,0}, \underline{v_{5,1}, v_{5,2}}, v_{5,3}, v_{5,4}, v_{5,5}, \cdots\cdots\cdots, v_{5,127}, v_{5,128}, \cdots, v_{5,131} \\
C_6 &= v_{6,0}, \underline{v_{6,1}, v_{6,2}}, v_{6,3}, v_{6,4}, v_{6,5}, \cdots\cdots\cdots, v_{6,127}, v_{6,128}, \cdots, v_{6,131} \\
C_7 &= v_{7,0}, \underline{v_{7,1}, v_{7,2}}, v_{7,3}, v_{7,4}, v_{7,5}, \cdots\cdots\cdots, v_{7,127}, v_{7,128}, \cdots, v_{7,131}
\end{aligned} \tag{5.18}
$$

From this example, it is clear that as a result of the doubling of the interleaver depth, burst errors involving up to sixteen symbols could be corrected. Hence, a doubling of the interleaving depth results in a doubling of the length of bursts that can be corrected.

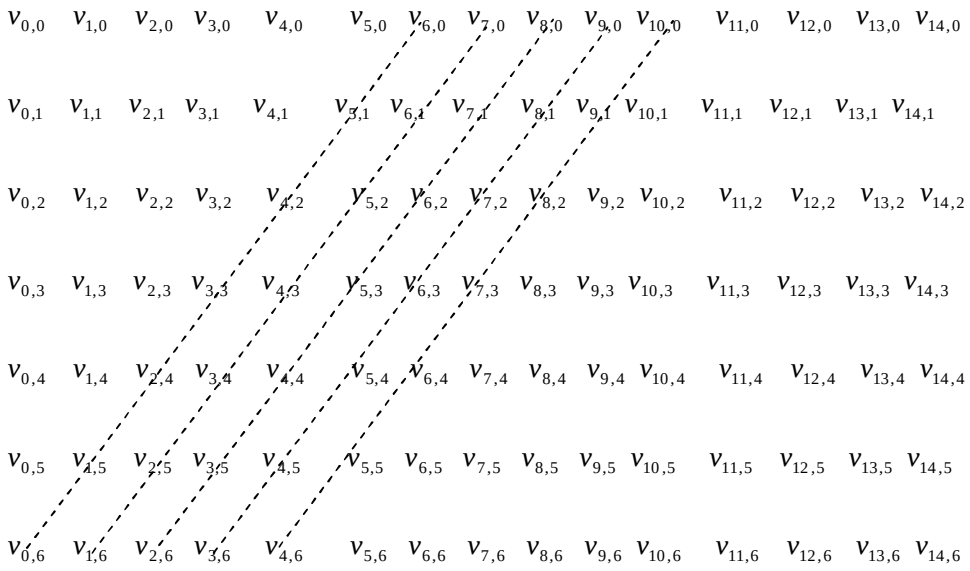
We have additionally considered the use of convolutional interleavers in this application and quantified the improvement in the burst error correcting capabilities of the code with these adaptations. The structure of convolutional interleaver (cross interleaver) is shown in Figure 5.4. Let us consider a (7, 3) RS code over F_{2^3} with an error correcting capability t = 2. It is shown in [Wicker, S.B. 1995] that the burst error correcting capability of a convolutional interleaver with interleaving depth m and error correcting capability t is quantified as

$$B = (mD+1)(t-1)+1 \tag{5.19}$$

where D denotes one symbol delay. If we choose $m = n = 7$, then the burst error correcting capability $B = 9$. Let us consider nine successive codewords produced by the code as shown in (5.20).

$$\begin{aligned}
P_0 &= v_{0,0}, v_{0,1}, v_{0,2}, v_{0,3}, v_{0,4}, v_{0,5}, v_{0,6} \\
P_1 &= v_{1,0}, v_{1,1}, v_{1,2}, v_{1,3}, v_{1,4}, v_{1,5}, v_{1,6} \\
P_2 &= v_{2,0}, v_{2,1}, v_{2,2}, v_{2,3}, v_{2,4}, v_{2,5}, v_{2,6} \\
P_3 &= v_{3,0}, v_{3,1}, v_{3,2}, v_{3,3}, v_{3,4}, v_{3,5}, v_{3,6} \\
P_4 &= v_{4,0}, v_{4,1}, v_{4,2}, v_{4,3}, v_{4,4}, v_{4,5}, v_{4,6} \\
P_5 &= v_{5,0}, v_{5,1}, v_{5,2}, v_{5,3}, v_{5,4}, v_{5,5}, v_{5,6} \\
P_6 &= v_{6,0}, v_{6,1}, v_{6,2}, v_{6,3}, v_{6,4}, v_{6,5}, v_{6,6} \\
P_7 &= v_{7,0}, v_{7,1}, v_{7,2}, v_{7,3}, v_{7,4}, v_{7,5}, v_{7,6} \\
P_8 &= v_{8,0}, v_{8,1}, v_{8,2}, v_{8,3}, v_{8,4}, v_{8,5}, v_{8,6}
\end{aligned} \tag{5.20}$$

This is presented to the convolutional interleaver in the following order,



(5.21)

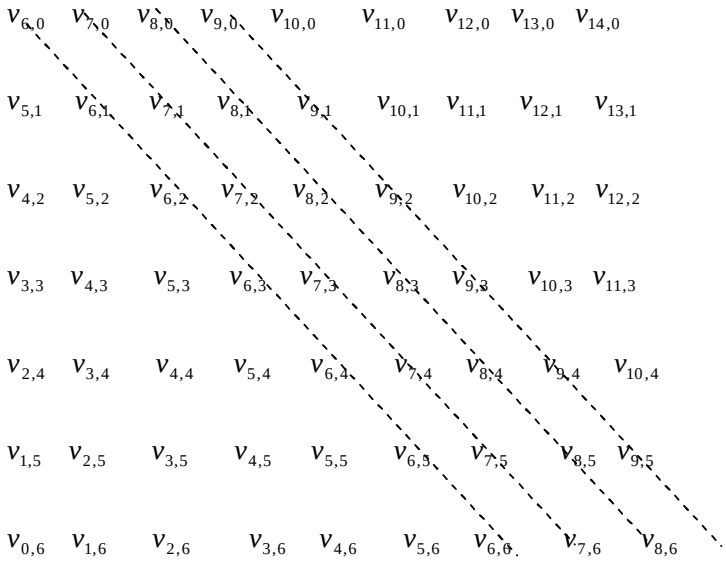
After interleaving, the data is available at the output of the interleaver in the following order as shown in (5.22). This is fed to the channel which might exhibit bursty behavior.

$$
\begin{array}{l}
v_{6,0}\ v_{5,1}\ v_{4,2}\ v_{3,3}\ v_{2,4}\ v_{1,5}\ v_{0,6} \vdots v_{7,0}\ v_{6,1}\ v_{5,2}\ v_{4,3}\ v_{3,4}\ v_{2,5}\ v_{1,6} \vdots \\
v_{8,0}\ v_{7,1}\ v_{6,2}\ v_{5,3}\ v_{4,4}\ v_{3,5}\ v_{2,6} \vdots v_{9,0}\ v_{8,1}\ v_{7,2}\ v_{6,3}\ v_{5,4}\ v_{4,5}\ v_{3,6} \vdots \\
v_{10,0}\ v_{9,1}\ v_{8,2}\ v_{7,3}\ v_{6,4}\ v_{5,5}\ v_{4,6} \vdots v_{11,0}\ v_{10,1}\ v_{9,2}\ v_{8,3}\ v_{7,4}\ v_{6,5}\ v_{5,6} \vdots \\
v_{12,0}\ v_{11,1}\ v_{10,2}\ v_{9,3}\ v_{8,4}\ v_{7,5}\ v_{6,6} \vdots v_{13,0}\ v_{12,1}\ v_{11,2}\ v_{10,3}\ v_{9,4}\ v_{8,5}\ v_{7,6} \vdots \\
v_{14,0}\ v_{13,1}\ v_{12,2}\ v_{11,3}\ v_{10,4}\ v_{9,5}\ v_{8,6} \vdots
\end{array}
\tag{5.22}
$$

Let us assume that during readout, the symbols are affected by a burst of error spanning nine successive symbols represented by

$$v_{12,0}\ v_{11,1}\ v_{10,2}\ v_{9,3}\ v_{8,4}\ v_{7,5}\ v_{6,6} \vdots v_{13,0}\ v_{12,1} \tag{5.23}$$

After deinterleaving, these erroneous symbols are redistributed back so that no more than two erroneous symbols are present per codeword. Thus the entire error burst can be corrected. This process is illustrated in (5.24) and the error location after deinterleaving is shown in (5.25).

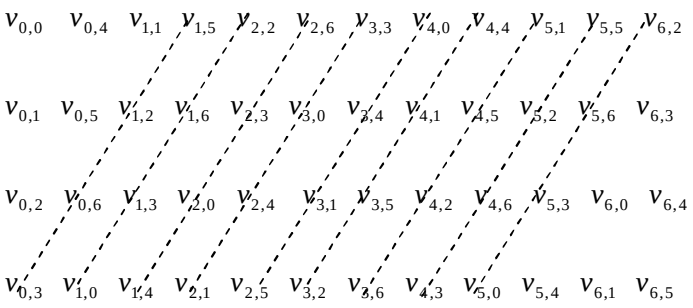


(5.24)

The output of the deinterleaver is

$$
\begin{array}{l}
v_{6,0}\ v_{6,1}\ v_{6,2}\ v_{6,3}\ v_{6,4}\ v_{6,5}\ \underline{v_{6,6}} \vdots v_{7,0}\ v_{7,1}\ v_{7,2}\ v_{7,3}\ v_{7,4}\ \underline{v_{7,5}}\ v_{7,6} \vdots \\
v_{8,0}\ v_{8,1}\ v_{8,2}\ v_{8,3}\ \underline{v_{8,4}}\ v_{8,5}\ v_{8,6} \vdots v_{9,0}\ v_{9,1}\ v_{9,2}\ \underline{v_{9,3}}\ v_{9,4}\ v_{9,5}\ v_{9,6} \vdots \\
v_{10,0}\ v_{10,1}\ \underline{v_{10,2}}\ v_{10,3}\ v_{10,4}\ v_{10,5}\ v_{10,6} \vdots v_{11,0}\ \underline{v_{11,1}}\ v_{11,2}\ v_{11,3}\ v_{11,4}\ v_{11,5}\ v_{11,6} \vdots \\
\underline{v_{12,0}}\ \underline{v_{12,1}}\ v_{12,2}\ v_{12,3}\ v_{12,4}\ v_{12,5}\ v_{12,6} \vdots \underline{v_{13,0}}\ v_{13,1}\ v_{13,2}\ v_{13,3}\ v_{13,4}\ v_{13,5}\ v_{13,6} \vdots \\
v_{14,0}\ v_{14,1}\ v_{14,2} \vdots v_{14,3}\ v_{14,4}\ v_{14,5}\ v_{14,6} \vdots
\end{array}
\qquad (5.25)
$$

Considering the case when m = 4, the burst error correcting capability is 6. Let us consider the codewords as in (5.20). These codewords are presented to the interleaver with a depth of interleaving of four as follows:



(5.26)

After interleaving, the data is available as

$$\begin{array}{l} v_{1,5}\ v_{1,2}\ v_{0,6}\ v_{0,3}\vdots v_{2,2}\ v_{1,6}\ v_{1,3}\ v_{1,0}\vdots v_{2,6}\ v_{2,3}\ v_{2,0}\ v_{1,4}\vdots \\ v_{3,3}\ v_{3,0}\ v_{2,4}\ v_{2,1}\vdots v_{4,0}\ v_{3,4}\ v_{3,1}\ v_{2,5}\vdots \underline{v_{4,4}\ v_{4,1}\ v_{3,5}\ v_{3,2}}\vdots \underline{v_{5,1}\ v_{4,5}}\ v_{4,2}\ v_{3,6}\vdots \\ v_{5,5}\ v_{5,2}\ v_{4,6}\ v_{4,3}\vdots v_{6,2}\ v_{5,6}\ v_{5,3}\ v_{5,0}\vdots v_{6,6}\ v_{6,3}\ v_{6,0}\ v_{5,4}\vdots \ldots\ldots\ldots \end{array} \tag{5.27}$$

Let us assume that during readout, the symbols are affected by a burst of error spanning six successive symbols represented by

$$\underline{v_{4,4}\ v_{4,1}\ v_{3,5}\ v_{3,2}\vdots v_{5,1}\ v_{4,5}}$$

This is presented to the deinterleaver.

$$\begin{array}{l} v_{1,5}\ v_{2,2}\ v_{2,6}\ v_{3,3}\ v_{4,0}\ v_{4,4}\ v_{5,1}\ v_{5,5}\ v_{6,2}\ v_{6,6} \\ v_{1,2}\ v_{1,6}\ v_{2,3}\ v_{3,0}\ v_{3,4}\ v_{4,1}\ v_{4,5}\ v_{5,2}\ v_{5,6}\ v_{6,3} \\ v_{0,6}\ v_{1,3}\ v_{2,0}\ v_{2,4}\ v_{3,1}\ v_{3,5}\ v_{4,2}\ v_{4,6}\ v_{5,3}\ v_{6,0} \\ v_{0,3}\ v_{1,0}\ v_{1,4}\ v_{2,1}\ v_{2,5}\ v_{3,2}\ v_{3,6}\ v_{4,3}\ v_{5,0}\ v_{5,4} \end{array} \tag{5.28}$$

The output of the deinterleaver is

$$\begin{array}{l} v_{1,5}\ v_{1,6}\ v_{2,0}\ v_{2,1}\vdots v_{2,2}\ v_{2,3}\ v_{2,4}\ v_{2,5}\vdots v_{2,6}\ v_{3,0}\ v_{3,1}\ \underline{v_{3,2}}\vdots v_{3,3}\ v_{3,4}\ \underline{v_{3,5}}\ v_{3,6}\vdots \\ v_{4,0}\ \underline{v_{4,1}}\ v_{4,2}\ v_{4,3}\vdots \underline{v_{4,4}}\ \underline{v_{4,5}}\ v_{4,6}\ v_{5,0}\vdots \underline{v_{5,1}}\ v_{5,2}\ v_{5,3}\ v_{5,4}\vdots \ldots\ldots\ldots \end{array} \tag{5.29}$$

After deinterleaving, these erroneous symbols are redistributed such that one or at the most two symbols per codeword are in error. This shows that integration of coding and decoding with interleaver helps in improving the data integrity of the device.

This chapter has been devoted to the study of the decoder architecture and investigation of the role of interleavers to improve the data integrity when burst errors occur in storage systems. In the next chapter highlights the conclusion with the summary of the research results and scope for future work.

This page is intentionally left blank

Chapter 6

Conclusion

6.1 Summary of the Results

The research work forming the body of this thesis started with a study of array codes [Roth, R. M. 1991] with an idea to apply them to correct burst errors in data storage systems. During this phase, we studied the rank distance properties of $[n \times n, k]$ array codes and their applications. We started our study of flash memory architecture with a view to apply $[n \times n, k]$ array codes for error detection and correction. However, we found that the device architecture does not support the use of array codes readily. Traditionally flash memories were designed using SLC as building blocks. SLC can store one bit of information. In recent years, (after 2010), the focus of researchers has shifted to the design of flash memories based on MLC. MLC can store more than one bit of information. MLC's that can store two bits, three bits and four bits of information per cell have been discussed in literature. There are two models which define the amount of overhead (in terms of redundant bits) that is used to protect the information stored in each sector. These models have been referred to in the thesis as Memory model 1 and Memory model 2 respectively. A perusal of technical literature revealed that BCH codes have been extensively used to detect and correct in flash memories. In 2008 [Mehnert, A. 2008], [Chen, Y, 2008], the state of art employed a BCH code that could correct six errors over a span of one sector (4096 bits). A detailed study of device architecture and an understanding of the structure of the BCH codes revealed that the device architecture has enough redundancy to support a more powerful BCH code. In chapter 3, we have synthesized BCH codes that can correct as many as eight bits and nine bits in error over the span of one sector. We also have attempted to synthesize codes with greater error correcting capability by combining two sectors. These results have been obtained for memory model 1. Further we have synthesized BCH codes that can correct up to eighteen bits in errors over a span of one sector by using the architecture specified in Memory model 2.

A perusal of technical literature pertaining to flash memories after 2010 reveals that a number of researchers are interested in designing MLC based flash memories. The basic organization of the flash memory remains unaltered. This fact leads us to introspect as to whether BCH codes could be synthesized to correct errors in MLC's.

The errors in MLC cannot be expressed by any conventional channel model, such as additive white Gaussian noise (AWGN) channel. Thus a suitable channel matrix has been determined to describe the various errors encountered during reading process in a MLC and their relative probabilities of occurrence. The channel matrices are computed for different values of σ for a four level MLC and eight level MLC. The channel matrices computed show that MLC suffers from asymmetric errors.

Encoding of BCH codes is relatively simple process. As far as the decoding is concerned, inversion free Berlekamp-Massey algorithm is used. We have employed this architecture for the codes described in the thesis. Burst errors are the dominant mode of errors in the storage systems. This can happen due to faulty sectors in a storage device and can result in destruction of large blocks of data. To strengthen the error correction codes so that it can meet the challenges posed by burst mode errors, interleaving is necessary. We have employed both block interleaver and convolutional interleaver to enhance error correction capability of various codes synthesized by us.

In conclusion, the work done in this thesis has yielded several BCH and RS codes that can be employed to enhance the integrity of flash memory devices. A comparison with the state of art reveals that these codes when employed can reduce the error rates in flash memory significantly and hence make flash technology suitable for use in application were data integrity is very critical. If sufficient improvement in flash memory reliability can be achieved, they could be considered as worthy competitors to Hard Disk Drives.

6.2 Directions for further research

Algebraic geometry (AG)[Silverman, J. H. and Tate, J. 1992], [Carrasco, R.A. and Johnston, M. (2008)], [Blake, I. et al. 1998], [Shibuya, T. et al. 1996], [Shibuya, T. et al. 1997], [Johnston, M. et al. 2004] is a powerful mathematical tool for constructing very long non-binary block codes with excellent parameters such as high code rate and large Hamming distance. These codes are constructed from the affine points of an irreducible projective curve and a set of rational functions defined on that curve. The length of an AG code is equal to the number of affine points. However, constructing AG codes requires an in-depth knowledge of the theory of algebraic geometry. The well known Reed-Solomon codes are the simplest class of AG code, constructed from the affine points of the projective line. Consequently, they have the shortest block lengths of all AG codes, and there are not many Reed-Solomon codes that can be constructed. However, Reed-Solomon codes are maximum distance separable (MDS) unlike other AG codes, where the genus of the curve can reduce the actual minimum Hamming distance. Despite this penalty, AG codes still have much larger minimum Hamming distances than Reed-Solomon codes defined over the same finite field and consequently AG codes can correct much longer bursts of errors, which are common in data storage systems. A disadvantage of AG codes is their higher decoding complexity. AG codes could be a possible candidate for the error correcting schemes in future data storage devices.

In chapter 3 and chapter 4 of the thesis, we have synthesized high rate long length BCH codes for correcting errors encountered in SLC's and MLC's. Abelian codes constitute a class of codes that includes cyclic codes as a special case. It has been shown by Berman [Berman, S.D. 1967] that under certain conditions, general class of Abelian codes has better error correction capability than the class of cyclic codes. Abelian codes are well suited in applications that require the use of codewords having long length and high rate. Therefore they are well suited for use in this application. However, a design procedure such as the BCH bound in the case of cyclic codes which enables the ready design of Abelian codes with good distance property is not available. If such a result allows the

synthesis of Abelian codes with good distance property (the counterpart of the BCH bound for Abelian codes) is discovered, then Abelian codes may find potential application not only in this field but in any field that requires the use of high rate, long length codes.

In chapter 5, we have discussed an inversion free Berlekamp-Massey decoding architecture for decoding BCH codes. Most of the steps in decoding BCH and RS codes are identical. The major difference is that error magnitudes have to be computed in case of RS codes. In our study we have confined ourselves to the decoding of BCH codes with the inversion free algorithm. Interested researchers can derive the steps required to apply Forney's algorithm (used to compute error magnitudes) and map them into simple steps that can be carried out by digital hardware to complete the decoding of RS codes proposed in the synthesis.

We have also synthesized several LDPC codes for use in this application by using Gallager constructions. Theses have resulted in codes being specified by **H** matrix with very large dimensions. This makes the process of evaluating the distance properties very difficult. However, researchers can look at alternative approaches to synthesize LDPC codes that may be useful in similar applications. In a similar manner, researchers could also investigate possible use of Turbo codes in this application.

References

Agarwal, A., Paul, B. C., Mukhopadhyay, S. and Roy, K. (2005). "Process variation in embedded memories: failure analysis and variation aware architecture", *IEEE Journal on Solid-Stae Circuits*, Vol. 40, pp. 1804-1814.

Ankolekar, P. P., Isaac, R. and Bredow, J. W. (2010). "Multibit Error Correction Methods for Latency Constrained Flash Memory Systems", *IEEE Trans. on Device and Materials Reliability*, Vol. 10, No. 1, pp. 33-39.

Atwood, G., Fazio, A., Mills, D.and Reaves, B. (1997). "Intel Strata memory technology overview", *Intel Technology Journal*., Vol. 1 [Online].
Available: http://www.intel.com/technology/itj/archive/1997.htm.

Bajura, M. et al. (2007). "Models and Algorithmic Limits for an ECC based APPROACH TO Harden Sub-100-nm SRAMs", *IEEE Trans. Nuclear Science*, Vol. 54, No. 4, pp. 935-945.

Berman, S. D. (1967). "On the theory of group codes", *Kibernetika*, Vol. 3, No. 1, pp. 31-39.

Berman, S. D. (1967). "Semisimple cyclic and Abelian codes", *Kibernetika*, Vol.3, No.3, pp. 21-30.

Bertozzi, D., Benini, L. and Micheli, G.D. (2005). "Error Control Schemes for On Chip Communication Links: The Energy-Reliability Tradeoff", *IEEE Trans. On Computer-Aided Design of Integrated Circuits and Systems*, Vol.24, No.6.

Bez, R., Camerlenghi, E., Modelli, A. and Visconti, A. (2003). "Introduction to Flash memory", Proc. of the IEEE, Vol.91, No.4, pp.489-502.

Blahut, R. E. (2003). “Algebraic Codes for Data Transmission”, First Edition, Cambridge University Press.

Blake, I., Heegard, C., Hohold, T. and Wei, V. (1998). “Algebraic Geometry Codes”, *IEEE Trans. on Information Theory*, Vol.44, No.6, pp. 2596-2618.

Bohossian, V., Jiang, A. and Bruck, J. (2007). “Buffer Codes for Asymmetric Multi-Level Memory”, *in Proc. IEEE Int. Symp. Information Theory (ISIT)*, France, pp. 1186-1190.

Calvel, P. et al. (1994). “Space radiation evaluation of 16 Mbit DRAMs for mass memory applications”, *IEEE Transactions on Nuclear Science*, pp. 576-593.

Carrasco, R. A. and Johnston, M. (2008). “Non-Binary Error Control Coding for Wireless Communication and Data Storage”, John Wiley & Sons, Ltd.

Cassuto, Y., Schwartz, M., Bohossian, V. and Bruck, J. (2010). “Codes for assymetric limited-magnitude errors with application to multilevel flash memories”, *IEEE Trans.on Information Theory*, Vol.56, No.4, pp. 1582-1594.

Chen, Y. (2008). “Flash Memory Reliability”, *NEPP 2008 Report, California Institute of Technology*.

Chen, Y. and Parhi, K. (2004). “Area efficient parallel decoder architecture for long BCH codes”, *Proceedings of IEEE International Conference on Speech and Signal Processing (ICASSP’04)*, Vol 5, pp. V-73-76.

Choi, H., Liu, W. and Sung, W. (2010). "VLSI Implementation of BCH Error Correction for Multilevel Cell NAND Flash Memory", *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 18, No. 5, pp. 843-847.

Cooke, J. (2007). "The Truth of NAND Flash Memory", *Micron Technology, Inc* available at http://download.micron.com/pdf/presentation/events/WinHEC.Cooke.pdf

Costello, D and Forney, D. (2007). "Channel Coding: The road to channel capacity", *Proceedings of IEEE*, Vol. 95, No. 6, pp. 1150-1177.

Fujiwara, E. (2006). "Code Design for Dependable Systems", Wiley Interscience.

Gal, E. and Toledo, S. (2005). "Algorithms and data structures for flash memories", *ACM Comput. Surv*., Vol. 37, pp.138-163.

Gregori, S., Cabrini, A., Khouri, O. and Torelli,G. (2003). *'On-Chip Error Correcting Techniques for New-Generation Flash Memories'*, *Proceedings of IEEE*, Vol. 91, No. 4, pp. 602-616.

Jiang, A., Mateescu, R., Schwartz, M. and Bruck, J. (2009a). "Rank modulation for flash memories," *IEEE Trans. Inf. Theory*, Vol. 55, No. 6, pp. 2659–2673.

Jiang, A., Langberg, M., Schwartz, M. and Bruck, J. (2009b). "Universal rewriting in constrained memories," in *Proc. IEEE Int. Symp. Inf.Theory*, Seoul, Korea, Jun.–July, pp. 1219–1223.

Jiang, A., Mateescu, R., Yaakobi, E., Bruck, J., Siegel, P. H., Vardy, A. and Wolf, J. K. (2010a). "Storage Coding for Wear Leveling in Flash Memories", *IEEE Transactions on Information Theory*, Vol. 56, No. 10, pp. 5290-5299.

Jiang, A., Bohossian, V. and Bruck, J.(2010b). “Rewriting codes for joint information storage in flash memories,” *IEEE Trans. Inf. Theory*, Vol. 56, No. 10, pp. 5300–5313.

Hsu, I. S., Reed, I. S., Truong, T.K., Wang, K., Yeh, C. S., Deutsch, L. J. (1984). “The VLSI Implementation of a Reed-Solomon encoder using Berlekamps Bit-Serial Multiplier Algorithm”, *IEEE Trans. on Computers*, Vol. c-33, No. 10, pp. 906-911.

Huang, Q., Lin, S. and Abdel-Ghaffer, K. A. S. (2011). “Error-Correcting Codes for Flash Coding”, *IEEE Trans. on Information Theory*, Vol.57, No.9, pp.6097-6108.

Im, S. and Shin, D. (2009). “Storage architecture and software support for SLC/MLC combined flash memory”, *Proc. of 24th ACM Symposium on Applied Computing (SAC ’09)*, Honolulu, Hawaii, pp. 1664-1669.

Johnston, M., Carrasco, R.A. and Burrows, B.L.(2004). “Design of Algebraic geometric codes over fading channels”, *Electronics Letter*, Vol. 40, No. 21.

Lin, H., Chen, T. and Chang, J. (2002). “Investigation of disturbance for new dual floating gate multilevel flash cells’, *Solid-State Electronics* , Vol. 46, pp. 1145 – 1150.

Lin, S. and Costello, D. (2004). “Error Control Coding”, Second Edition, Prentice Hall**.**

Liu, W., Rhi, J. and Sung, W. (2006). “Low-power high-throughput BCH error correction VLSI design for multi-level cell NAND flash memories*”, in Proc. Int.Workshop SiPS*, 2006, pp. 248-253.

Lo, J.C. and Fujiwara, E. (2005). “Transient behavior of the encoding/decoding circuits of error control code”, *Proc. IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems*.

MacKay, D. J. C. (2003). “Information Theory, Inference, and Learning Algorithm, Cambridge University Press.

Maeda, Y. and Kaneko, H. (2009). “Error Control Coding for Multilevel Cell Flash Memories Using Nonbinary Low-Density Parity-Check Codes”, *24th International Symposium on Defect and Fault Tolerance in VLSI Systems,* pp. 367-375.

Mahdavifar, H., Siegel, P.H., Vardy, A., Wolf,J.K. and Yaakobi, E. (2009). “A nearly optimal construction of flash codes,” in *Proc. IEEE Int. Symp. Inf. Theory*, Seoul, Korea, June – July, pp. 1239–1243.

Massengill, L.W. (1996). “Cosmic and terrestrial single event radiation effects in dynamic random access memories”, *IEEE Transactions on Nuclear Science,* pp. 576-593.

Mehnert, A. (2008). “Managing Flash memory with intelligence”, *Industrial Embedded Systems E-Letter*, March 2008 (Hyperstone GmbH).

Micheloni, R., Marelli, A. and Ravasio, R. (2008). “Error Correction Codes for Non-Volatile Memories”, Springer.

Mielke, N., Marquart, T., Wu, N., Kessenich, J., Belgal, H., Schares, E., Trivedi, F., Goodness, E. and Nevill, L.R. (2008). “Bit Error Rate in NAND Flash Memories”, *IEEE 46th Annual International Reliability Physics Symposium,* pp. 9-19.

Moon, T, K. (2006). “Error Correction Coding”, Wiley Interscience.

Pless, V. and Huffman, W.C. (1998). “Handbook of Coding Theory”, Vol. 2, Elsevier.

Reed, I. S., Shih, M. T. and Truong, T. K. (1991). “VLSI design of Inverse-free Berlekamp-Massey algorithm”, *IEEE proceedings-E*, Vol. 138, No. 5, pp. 295-298.

Ricco, B., Torelli, G., Lanzoni, M., Manstretta, A., Maes, H.E., Montanari, D. and Modelli, A. (1998). “Nonvolatile Multilevel Memories for Digital Applications”, *Proceedings of IEEE*, Vol. 86, No. 12, pp. 2399-2421.

Rossi, D., Metra, C. and Ricco, B.(2002). “Fast and Compact Error Correcting Scheme for Reliable Multilevel Flash Memories”, *Proceedings of the Eighth International On-Line Testing Workshop (IOLTW’02)*.

Rossi, D. and Metra, C. (2003). “Error Correcting Strategy for High Speed and High Density Reliable Flash Memories”, *Journal of Electronic Testing: Theory and Applications*, Vol. 19, pp. 511-521.

Roth, R. M. (1991). “Maximum-Rank Array Code and their application to Criss Cross Error Correction”, *IEEE Trans. on Information Theory*, Vol. 37. pp.328-336.

Ryan, W. E. and Lin, S. (2009). “Channel Codes: Classical and Modern”, Cambridge University Press.

Shao, H.M., Truong, T.K., Deutsch, L.J., Yuen, J.H. and Reed, I.S. (1985). “A VLSI Design of a Pipeline Reed-Solomon Decoder”, *IEEE Trans. on Computers*, Vol. C-34, No.5.

Silverman, J. H. and Tate, J. (1992). “Rational Points on Elliptic Curves”, Springer-Verlag New York Inc.

Shibuya, T., Matsumoto, R. and Sakaniwa, K. (1997). “Simple Estimation for Dimension of Subfield Subcodes of AG Codes”, *IEICE Trans. Fundamentals*, Vol. E80-A, No. 11, pp. 2058-2065.

Shibuya, T., Jinushi, H., Miura, S. and Sakaniwa, K. (1996). “On the performance of Algebraic Geometric codes*”, IEICE Trans. Fundamentals,* Vol. E79-A, No. 6, pp.928-937.

Slayman, C. W. (2005). “Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations”, *IEEE Trans. on Devices and Materials Reliability*”, Vol. 5, pp. 397-404.

Sun, F., Devarajan, S., Rose, K. and Zhang, T.(2007). “Design of on-chip error correction systems for multilevel NOR and NAND flash memories”, *IET Circuits Devices Syst.,* Vol. 1, No.3, pp 241-249.

Sun, F., Rose, K. and Zhang, T. (2006). “On the use of Strong BCH Codes for improving multilevel NAND flash memory storage capacity”, *IEEE workshop on Signal Processing Systems (SiPS): Design and Implementation.*

Tanzawa, T., Tanaka, T., Takeuchi, K., Shirota, R., Aritome, S., Watanabe, H., Hemink, G., Shimizu, K., Sato, S., Takeuchi, Y. and Ohuchi, K. (1997). “A Compact On-Chip ECC for Low Cost Flash Memories”, *IEEE journal on Solid State Circuits,* Vol. 32, No. 5, pp. 662-669.

Wicker, S. B. (1995). “Error Control Systems for Digital Communication and Storage”, Prentice Hall.

Wang, Z. and Bruck, J. (2010). “Partial rank modulation for flash memories”, in *Proc. IEEE Int. Symp. Inf. Theory*, Austin, TX, Jun. 13–18, pp. 864–868.

Wang, Z. and Karpovsky, M. (2012). “Nonlinear Multi-Error Correction Codes for Reliable MLC NAND Flash Memories”, *IEEE Transactions on VLSI Systems*, Vol.20, No. 7, pp. 1221-1234.

Yaakobi, E., Vardy, A., Siegel, P.H. and Wolf, J.K. (2008). “Multidimensional flash codes,” in *Proc. Annual Allerton Conf. Communication, Control and Computing*, Monticello, IL, Sep. 23–26, pp. 392–399.

Yamada,J. (1987). “Selector-line merged built-in ECC technique for DRAM’s”, *IEEE Journal of Solid State Circuits*, Vol.SC-22, No.5, pp. 868-873.

Publications Based on the Research Work Described in this Thesis

International Journal Papers

1. Rajesh Shetty K, Sripati U, Prashantha Kumar H, Shankarananda B, "Design and Construction of Algebraic Codes for Enhancing Data Integrity in Flash Memories", International Journal of Advances in Communication Engineering, Vol. 2, No. 2, ISSN: 0975-6094, pp. 51-56, July-December 2010.

2. Rajesh Shetty K, Ramakrishna, Sripati U, Prashantha Kumar H, "Design and Construction of BCH Codes for Enhancing Data Integrity in Multilevel Flash Memories", International Journal of Information and Communication Technology, Vol.4, Issue 1, pp.40-60, 2012 (InderScience Publications).

International Conference Papers

3. Rajesh Shetty K, Sripati U, Prashantha Kumar H, Shankarananda B, "Design and Construction of Codes for MIMO Block Fading Channels", Proceedings of International Conference on Advanced Computing and Communication (ICACC−2010), May 3-4, 2010, Amal Jyothi College of Engineering, Kanjirapally, Kottayam), pp. 201-205 (secured best paper award).

4. Rajesh Shetty K, Sripati U, Prashantha Kumar H, Shankarananda B, "Synthesis of BCH Codes for Enhancing Data Integrity in Flash Memories", 5th International Conference on Industrial and Information Systems (ICIIS−2010), 29th July to 1st August 2010, National Institute of Technology Karnataka, Surathkal, pp.119-124 (Available on IEEE Xplore).

This page is intentionally left blank

Bio data of Rajesh Shetty K

Rajesh Shetty K
Department of Electronics & Communication Engineering,
N.M.A.M Institute of Technology, Nitte – 574 110.
Email : krshetty_nitte@yahoo.co.in

Qualifications:

- B.E (1990) (Electronics & Communication) – First Class
  N.M.A.M Institute of Technology, Nitte – 574 110.
  (Mangalore University)

- M.Tech (2004) (Digital Electronics & Advanced Communication)
  (Secured First Class with Distinction)
  National Institute of Technology Karnataka, Surathkal.

Total experience: 22 years (Teaching 17 years and industry 5 years)

Duration	Positions Held	Employer
November 2013 to till date	Associate Professor, Dept. of E & C Engineering	N.M.A.M Institute of Technology, Nitte – 574110. Karkala Taluk, Udupi Dist
December 2004 to October 2013	Asst. Professor, Dept. of E & C Engineering	N.M.A.M Institute of Technology, Nitte – 574110. Karkala Taluk, Udupi Dist.
October 2000 to November 2004	Lecturer, Sr.Lecturer, Dept. of E & C Engineering	N.M.A.M Institute of Technology, Nitte – 574110. Karkala Taluk, Udupi Dist
December 1995 to October 2000	Sr. Engineer	Eagle Sales Corporation, R.M.R.Road, Bangalore -25.
November 1990 to December 1995	Lecturer, Dept. of E & C Engineering	N.M.A.M Institute of Technology, Nitte – 574110. Karkala Taluk, Udupi Dist.

No. of International Journal Publications: 9
No. of International Conference proceedings/ publications: 7
No. of National Conference proceedings / publications: 2
No. of Patent Applications filed: 2
No. of Expert Talk Delivered: 1
No. of Professional Training undergone: 1
No. of Workshops Attended: 3