

Channel Coding Methods for Non-Volatile Memories

Lara Dolecek

Department of Electrical Engineering
University of California, Los Angeles
Los Angeles, CA, 90095-1594

Frederic Sala

Department of Electrical Engineering
University of California, Los Angeles
Los Angeles, CA, 90095-1594

now

the essence of knowledge

Boston — Delft



Documents

Document Sharing Hub

Foundations and Trends[®] in Communications and Information Theory

Published, sold and distributed by:

now Publishers Inc.
PO Box 1024
Hanover, MA 02339
United States
Tel. +1-781-985-4510
www.nowpublishers.com
sales@nowpublishers.com

Outside North America:

now Publishers Inc.
PO Box 179
2600 AD Delft
The Netherlands
Tel. +31-6-51115274

The preferred citation for this publication is

L. Dolecek and F. Sala. *Channel Coding Methods for Non-Volatile Memories*.
Foundations and Trends[®] in Communications and Information Theory, vol. 13,
no. 1, pp. 1–128, 2016.

This Foundations and Trends[®] issue was typeset in L^AT_EX using a class file designed by Neal Parikh. Printed on acid-free paper.

ISBN: 978-1-68083-103-0

© 2016 L. Dolecek and F. Sala

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, mechanical, photocopying, recording or otherwise, without prior written permission of the publishers.

Photocopying. In the USA: This journal is registered at the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923. Authorization to photocopy items for internal or personal use, or the internal or personal use of specific clients, is granted by now Publishers Inc for users registered with the Copyright Clearance Center (CCC). The 'services' for users can be found on the internet at: www.copyright.com

For those organizations that have been granted a photocopy license, a separate system of payment has been arranged. Authorization does not extend to other kinds of copying, such as that for general distribution, for advertising or promotional purposes, for creating new collective works, or for resale. In the rest of the world: Permission to photocopy must be obtained from the copyright owner. Please apply to now Publishers Inc., PO Box 1024, Hanover, MA 02339, USA; Tel. +1 781 871 0245; www.nowpublishers.com; sales@nowpublishers.com

now Publishers Inc. has an exclusive license to publish this material worldwide. Permission to use this content must be obtained from the copyright license holder. Please apply to now Publishers, PO Box 179, 2600 AD Delft, The Netherlands, www.nowpublishers.com; e-mail: sales@nowpublishers.com

Foundations and Trends® in Communications and Information Theory

Volume 13, Issue 1, 2016

Editorial Board

Editor-in-Chief

Sergio Verdú
Princeton University
United States

Editors

Venkat Anantharam
UC Berkeley

Helmut Bölcskei
ETH Zurich

Giuseppe Caire
USC

Daniel Costello
University of Notre Dame

Anthony Ephremides
University of Maryland

Alex Grant
*University of South
Australia*

Andrea Goldsmith
Stanford University

Albert Guillen i Fabregas
Pompeu Fabra University

Dongning Guo
Northwestern University

Dave Forney
MIT

Te Sun Han
University of Tokyo

Babak Hassibi
Caltech

Michael Honig
Northwestern University

Johannes Huber
University of Erlangen

Tara Javidi
UC San Diego

Ioannis Kontoyiannis
*Athens University
of Economy and Business*

Gerhard Kramer
TU Munich

Sanjeev Kulkarni
Princeton University

Amos Lapidoth
ETH Zurich

Bob McEliece
Caltech

Muriel Medard
MIT

Neri Merhav
Technion

David Neuhoff
University of Michigan

Alon Orlitsky
UC San Diego

Yury Polyanskiy
MIT

Vincent Poor
Princeton University

Maxim Raginsky
UIUC

Kannan Ramchandran
UC Berkeley

Shlomo Shamai
Technion

Amin Shokrollahi
EPF Lausanne

Yossef Steinberg
Technion

Wojciech Szpankowski
Purdue University

David Tse
UC Berkeley

Antonia Tulino
Alcatel-Lucent Bell Labs

Rüdiger Urbanke
EPF Lausanne

Emanuele Viterbo
Monash University

Tsachy Weissman
Stanford University

Frans Willems
TU Eindhoven

Raymond Yeung
CUHK

Bin Yu
UC Berkeley

Editorial Scope

Topics

Foundations and Trends® in Communications and Information Theory publishes survey and tutorial articles in the following topics:

- Coded modulation
- Coding theory and practice
- Communication complexity
- Communication system design
- Cryptology and data security
- Data compression
- Data networks
- Demodulation and Equalization
- Denoising
- Detection and estimation
- Information theory and statistics
- Information theory and computer science
- Joint source/channel coding
- Modulation and signal design
- Multiuser detection
- Multiuser information theory
- Optical communication channels
- Pattern recognition and learning
- Quantization
- Quantum information processing
- Rate-distortion theory
- Shannon theory
- Signal processing for communications
- Source coding
- Storage and recording codes
- Speech and Image Compression
- Wireless Communications

Information for Librarians

Foundations and Trends® in Communications and Information Theory, 2016, Volume 13, 4 issues. ISSN paper version 1567-2190. ISSN online version 1567-2328. Also available as a combined paper and online subscription.

Channel Coding Methods for Non-Volatile Memories

Lara Dolecek

Department of Electrical Engineering
University of California, Los Angeles
Los Angeles, CA, 90095-1594

Frederic Sala

Department of Electrical Engineering
University of California, Los Angeles
Los Angeles, CA, 90095-1594

Contents

1	Data-Driven Need For More Reliable Memories	3
2	Operational Characteristics of NVMs	7
2.1	Basics of flash memory operations	7
2.2	Basics of PCM operations	12
3	Exploiting Intra-Cell Variability For Improved Reliability: New Error Correction Coding	15
3.1	Algebraic-coding methods	16
3.2	Graph-based methods	35
3.3	Summary, additional research topics, and outlook	43
4	Maximizing the Number of Writes: WOM Codes	47
4.1	Coding for WOM	47
4.2	Summary, additional research topics, and outlook	68
5	A Novel Data Representation Scheme: Rank Modulation	71
5.1	Rank modulation Gray codes	73
5.2	Error-correcting codes	80
5.3	Extensions of rank modulation	87
5.4	Summary, additional research topics, and outlook	91

6	Dealing With Inter-Cell Interference: Constrained Coding	95
6.1	Constrained codes for flash: the “high-low-high” pattern	96
6.2	Constrained coding for phase-change memories	106
6.3	Summary, additional research topics, and outlook	110
7	Summary and Further Topics	113
	Acknowledgements	119
	References	121

Abstract

Non-volatile memories (NVMs) have emerged as the primary replacement of hard-disk drives for a variety of storage applications, including personal electronics, mobile computing, intelligent vehicles, enterprise storage, data warehousing, and data-intensive computing systems. Channel coding schemes are a necessary tool for ensuring target reliability and performance of NVMs. However, due to operational asymmetries in NVMs, conventional coding approaches - commonly based on designing for the Hamming metric - no longer apply. Given the immediate need for practical solutions and the shortfalls of existing methods, the fast-growing discipline of coding for NVMs has resulted in several key innovations that not only answer the needs of modern storage systems but also directly contribute to the analytical toolbox of coding theory at large.

This monograph discusses recent advances in coding for NVMs, covering topics such as error correction coding based on novel algebraic and graph-based methods, write-once memory (WOM) codes, rank modulation, and constrained coding. Our goal in this monograph is multifold: to illuminate the advantages - as well as challenges - associated with modern NVMs, to present a succinct overview of several exciting recent developments in coding for memories, and, by presenting numerous potential research directions, to inspire other researchers to contribute to this timely and thriving discipline.

L. Dolecek and F. Sala. *Channel Coding Methods for Non-Volatile Memories*. Foundations and Trends[®] in Communications and Information Theory, vol. 13, no. 1, pp. 1–128, 2016.
DOI: 10.1561/01000000084.

1

Data-Driven Need For More Reliable Memories

Welcome to the age of data! The amount of data generated in 2014 was estimated to be an unprecedented 4 Zettabytes (which is four billion Terabytes!), and is expected to increase to a whopping 40 Zettabytes by 2020 [2]. This data deluge has provided opportunities for scientific discoveries and technological innovations. To make full use of new data, the ability to store large-scale data reliably and efficiently has become the key challenge. Yet, available data storage systems have been grossly outpaced by data generation – the capacity of current storage systems is estimated at only 30% of generated data, and this percentage will decrease even further [2]. It is thus paramount to develop new approaches that make data storage systems fast, ultra-reliable, dense, and affordable.

Non-volatile memories (NVMs) are a class of memories that maintain stored data even after being disconnected from a power supply. NVMs offer faster data access, reduced power consumption, and improved physical resiliency relative to conventional hard disks. As a result, NVMs have emerged as the primary replacement of hard-disk drives for a variety of storage applications, including personal electronics, mobile computing, intelligent vehicles, enterprise storage, data warehousing, and data-intensive computing systems.

Flash memories are the most popular NVM technology today. Other NVM types, such as phase-change memory (PCM), magnetoresistive random access memory (MRAM), and spin-transfer-torque random access memory (STT-RAM), are also being actively pursued and may eventually offer viable alternatives to flash. Given the maturity and the ubiquity of flash devices, we will primarily focus on coding methods for flash memories; as appropriate, we will comment on the usage of related techniques for other technologies as well.

A flash device consists of floating gate transistors, organized in a two- or three-dimensional array. Each memory cell stores information: the amount of charge stored in the cell corresponds to a certain digital value. Improvements in fabrication technology over the past decade have resulted in decreased per-unit cost and increased areal densities of NVMs, and have thus made flash memories the primary replacement of hard disk drives in a range of modern applications. However, these positive trends have come at the expense of reduced reliability and limited device lifetime.

Unlike hard disks, NVMs can only be used for a certain number of program and erase (P/E) cycles, after which a device is deemed unusable. Currently, there is an unfavorable trade-off between storage density and endurance: as the cell density doubles, the lifetime drops by 10x - 20x [67]. Additionally, with the increase in storage capacity, due to a variety of physical impairments, memory reliability also rapidly decreases.

A particularly promising approach to overcome these physical limitations in NVMs is to apply error correction schemes. The field of channel coding deals with the development of practical error correction methods that introduce controlled, carefully designed redundancy into a data stream to make transmission or storage over a noisy medium as reliable as possible. Channel coding methods have been used with great success in data storage technologies and have helped make computer storage ubiquitous. However, NVMs present unique operational constraints that make existing channel coding methods inadequate. In particular, due to operational asymmetries in NVMs, conventional coding approaches – commonly based on designing for the Hamming met-

ric – no longer apply. Given the immediate need for practical solutions and the shortfalls of the existing methods, the fast-growing discipline of coding for NVMs has resulted in several key innovations that not only answer the needs of modern storage systems but also directly contribute to the analytical toolbox of coding theory at large.

This manuscript discusses recent advances in coding for NVMs, covering topics such as error correction coding based on novel algebraic and graph-based methods, write-once memory (WOM) codes, rank modulation, and constrained coding. Our goal for this work is multifold: to illuminate the advantages – as well as challenges – associated with modern NVMs, to present a succinct overview of several exciting recent developments in coding for memories, and, by presenting several possible research directions, to inspire other researchers to contribute to this timely and thriving discipline.

In Chapter 2 we will discuss the key operational features associated with flash memories. Motivated by these idiosyncrasies, subsequent chapters will overview recent developments in channel coding methods (Chapter 3), WOM codes (Chapter 4), rank modulation schemes (Chapter 5), and constrained codes (Chapter 6). The concluding Chapter 7 will summarize key results and will also list several research topics of interest to broad coding/information theory and signal processing community.

2

Operational Characteristics of NVMs

In this chapter we summarize the key operational features of non-volatile memories, focusing primarily on NAND flash memories, followed by a brief overview of phase-change memories (PCM).

2.1 Basics of flash memory operations

The basic building block of a flash memory is the memory cell. A memory cell corresponds to a transistor that has a control gate and a floating gate, separated by insulating layers. The value of data stored in a memory cell corresponds to the amount of charge on the floating gate. There are two major types of flash technologies: NAND flash and NOR flash. The terminology NAND and NOR refers to the logical NAND-like and NOR-like behavior of the device. The main difference between NOR and NAND flash is related to the structure of the connections between transistors. NOR flash allows for reading individual cells, while cells are accessed by page in NAND. On the other hand, NOR flash has a much higher cost per bit and slower write speeds compared to NAND. As a result, NAND flash is the dominant technology; from now on, when we use the term flash, we implicitly refer to NAND flash. See Figure 2.1 for a NAND cell illustration.

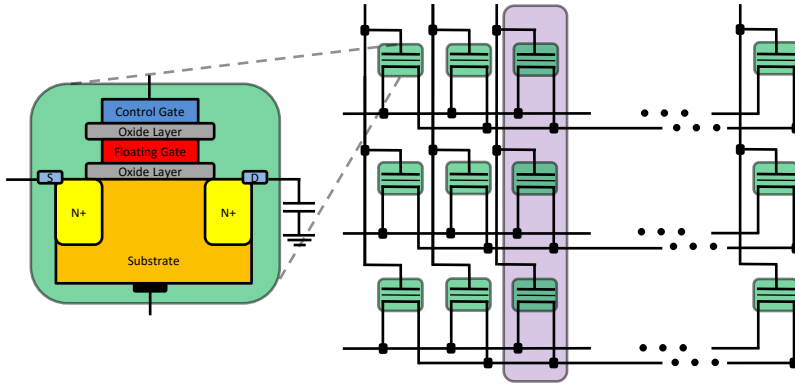


Figure 2.1: NAND Flash array organization with a single cell highlighted.

The process of writing and erasing a NAND cell consists of applying a sufficiently high voltage, called the *threshold voltage*, to the control gate to alter the charge on the floating gate. Depending on the direction in which the electrons flow, the cell is programmed (electrons flow towards the floating gate) or erased (electrons flow away from the floating gate). This process is known as Fowler-Nordheim tunneling. The process of reading is dual: it is related to determining the amount of charge stored on the floating gate and is performed as follows. Input voltage is applied to the control gate and the drain current is measured. If the drain current is below/above a certain threshold, as measured by a sense amp comparator, we conclude that the charge on the floating gate (i.e., the information stored) is below/above a prescribed threshold i.e., the transistor was turned off/on. Therefore, a *single threshold* scheme tells us whether the stored charge is below or above a certain level but does not tell us the exact amount of stored charge. The input voltage is applied to all cells in the page together, thus parallelizing the read operation.

In current NAND flash architectures, memory cells are organized into two dimensional arrays. Cells are organized into pages, which are

further combined into blocks. Many blocks are put together to produce one flash device. For example a 2Gb flash device consists of 2048 blocks with 64 pages per block, and 2112 bytes per page; other combinations of page/block sizes are also possible and they yield different overall device capacities. More recently, 3D (vertical) NAND flash has been developed, which has a more complex architecture due to the 3D structure.

Additionally, flash devices can be categorized by the number of bits memory cells can store. Single-level cell (SLC) devices store a single bit per cell; the SLC nomenclature comes from the fact that a single threshold is needed to distinguish the two ranges, corresponding to bit value '0' and bit value '1'. Multiple level cell (MLC) devices store multiple bits per cell, and multiple thresholds are needed to distinguish different levels. As somewhat of a misnomer, triple level cell (TLC) refers to a multilevel flash device storing three bits per cell – here seven thresholds are needed to distinguish among eight levels (one for each combination of a binary triplet). Flash devices storing four bits per cell (i.e., sixteen levels) are considered the densest solutions currently available. For each of these cell densities, the lowest level is the fully erased state and the highest level is the fully programmed state. See Figure 2.2 for an illustration of SLC, MLC, and TLC devices. As the

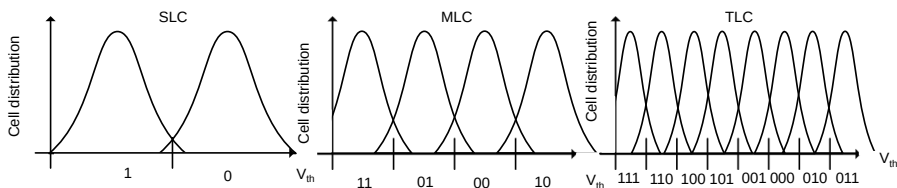


Figure 2.2: SLC, MLC, and TLC cells, with information associated with each region.

figure shows, there is variation among different cells of the same type; for a variety of reasons (summarized below), the cells do not always behave in exactly the same way. The value of the threshold voltage needed to turn on a cell varies across cells, and this induces variation

in cell behavior. As a result, we observe a distribution associated with different cell levels. It is apparent from the figure that as the number of levels represented in a cell increases, the reliability of the device decreases since the adjacent distributions experience more overlaps.

An important feature of widely popular NAND flash is that the write operation by way of adding charge is done at the page level, but the erase operation is done at the block level. As a result, if a cell value needs to be decreased, the entire block of cells this cell belongs to must also be erased and re-written. Unfortunately, frequent erases wear out the device to the point that it can no longer be used with confidence. The usable lifetime is typically expressed in terms of the number of program and erase (P/E) operations that can be executed before the device is no longer considered reliable. The denser the device is, the fewer P/E cycles it can sustain before it is deemed unreliable. For example, for SLC devices, the expected lifetime is around 10^5 P/E cycles. For MLC devices storing two bits per cell, the lifetime drops to 10^4 P/E cycles, and with three bits per cell TLC memory, it drops even further to a mere 10^3 P/E cycles or even fewer [67], translating to only months of usable lifetime for frequently written disks! The P/E lifetime issue is further exacerbated by the reliability requirements: the raw bit error rate (BER) can be unacceptably high, even as high as $10^{-2} - 10^{-1}$ [67], while the product specification requires the device to operate at the undetected BER (UBER) level of 10^{-16} and lower [1].

Therefore, despite their many advantages including faster access and lower power consumption compared to conventional hard disks, flash memories also present unique characteristics that can present a major roadblock in the quest for reliable and dense storage. We now summarize the main issues:

- **overshooting:** In order to prevent accidental overshooting that would consequently result in an expensive block erase, writing to a higher value (or a lower value, when preceded by an erase) must be done carefully. In practice, incremental step pulse programming (ISPP) is applied as follows. Starting from the current level, a small amount of charge is added, and the cell is read back to test if it holds the desired target value. If the cell is still

below the target level, another small amount of charge is repeatedly applied until the target level is reached. Since ISPP operates in very small increments, write latency inevitably increases with the number of pulses that need to be applied. This issue has become especially pronounced in newer dense memories that have many levels. We note that in the ISPP technique, charge is added to cells in parallel, so that cells' charge levels increase together step-by-step until their target levels are reached.

- **charge leakage:** In the ideal case, the electrons are trapped in the floating gate indefinitely. However, as time passes after the initial programming, the electrons slowly leak out, and with sufficiently large amounts of leakage, the amount of charge stored in the cell will end up being below the threshold separating the originally stored level from a level below – the cell will be read out as having a value lower than the intended value. In addition, as the devices age, the total amount of charge that can be stored in cell gates gets reduced because of worsening defects.
- **inter-cell coupling:** While the cells in principle can be accessed individually, the process of writing into a cell may inadvertently affect the levels in adjacent cells due to the parasitic capacitances of neighboring cells through so-called inter-cell coupling. This inter-cell interference is particularly pronounced when a low-valued cell is in between two cells being programmed to high levels. Programming the outer cells can result in the unintentional programming of the middle (victim) cell.
- **randomness and compound effects:** Each digital value corresponds to a certain amount of charge stored in the cell. As the device technology scales down, and as more bits are packed in each cell, the ranges of the voltage values associated with a particular level shrink to the point that only a handful of electrons correspond to the particular level. At this scale, even minute, random changes can result in the alteration of the cell value. Taken collectively, the voltage distributions associated with different cell values will have overlapping tails, further compromising reliability.

An intriguing observation is that these characteristics are highly asymmetric. For example, overwriting only causes errors in the upward direction, whereas charge leakage only causes errors in the downward direction. Likewise, inter-cell coupling affects low levels more profoundly than high levels. Additionally, empirical evidence shows that the distribution associated with the lowest state (fully erased) tends to have a broader shape than the distributions associated with other levels [111]. (Figure 2.2 is the cartoon of the idealized, equal-width distributions). All these effects negatively impact device reliability and lifetime.

2.2 Basics of PCM operations

Phase-change memory (PCM) devices offer another viable NVM alternative. Like flash memories, individual cells store digital information via a physical cell state. PCM cells have two possible states, an amorphous and a crystalline state – each state denotes a bit value stored. When PCM devices are implemented with partially crystalline states, multiple bits per cell can be stored [19].

Unlike flash memories where a decrease of a value of a single cell requires the entire block to be erased (with a million cells per block), in PCM memories a cell value can be individually lowered. However, this operation must first decrease the cell value to state zero and can only then program the desired new value (that is lower than the previously stored value). The sequence of the operations is called RESET (for erase) and SET (for programming). As with flash, however, the number of allowable RESET operations is limited.

The RESET operation involves applying a large temperature to cells in order to bring them back into the initial amorphous state. In the SET operation, a more moderate temperature increases the crystallinity of cells. Such temperature operations have several negative effects. The first, known as *thermal crosstalk*, occurs when the large temperature (used to reset cells to the amorphous state) impacts adjacent cells, raising their levels (that is, making them more crystallized.) Note that this is essentially the opposite effect of inter-cell coupling in

flash: a large downwards (rather than upwards) operation inadvertently raises the levels of neighboring cells. Another problem is known as *thermal accumulation*. This effect occurs when a set of physically adjacent cells is repeatedly programmed, resulting in heat accumulating in that area. The effect is especially harmful for highly-dense PCMs.

In the sequel, we will discuss several recent developments in coding for memories and demonstrate how they can quite successfully improve lifetime and reliability of NVMs by intelligently taking advantage of the underlying asymmetries and dominant error patterns.

3

Exploiting Intra-Cell Variability For Improved Reliability: New Error Correction Coding

In this chapter we discuss several recent developments in error correction coding for NVMs. In the first part, we focus our attention on algebraic coding methods attuned to the physical characteristics of flash, and in the second part we demonstrate how graph-based codes can be effectively used in flash under enhanced precision mechanisms.

Early generations of flash memories used simple single-error correcting Hamming codes. As the device geometry was scaled down, and the frequency of errors increased, the need for more sophisticated error control schemes emerged. Well-known traditional BCH codes have long been used in flash memories, since they offer a crisp error correction capability characterization that memory designers need: A BCH code of given length $p^m - 1$, where p is prime and m is a positive integer, is easily designed to be able to correct a prescribed number of errors t . Such a BCH code is defined over $GF(p^m)$. Its parity check matrix is composed of powers of α , for α a primitive element of $GF(p^m)$, and has $p^m - 1$ columns and $2t$ rows [103]. While BCH codes possess the desirable feature of guaranteed error correction capability with small redundancy overhead, they are intrinsically designed to correct symmetric errors, where all error patterns up to prescribed weight are essentially treated as equal.

However, as recently recognized, and as we explained in the previous chapter, errors in NVMs, and most notably in flash memories, behave in a highly asymmetric way. Thus, under such intricate error patterns, symmetric error correcting codes – including BCH codes and the like – may be suboptimal in scenarios where the errors are not symmetric. In this chapter we will first survey several recent developments in algebraic ECC methods for flash; these codes embrace, rather than ignore, the underlying asymmetry in errors. Additionally, we will also summarize recent results on the usage of graph-based codes for flash and we will demonstrate that the LDPC code design targeted for the ubiquitous additive white Gaussian noise (AWGN) model may not be the correct approach for the flash channel code design. This insight will further support our overall premise that the departure from the canonical channels is necessary for the development of high-performance error control schemes that are appropriate for flash-like channels.

3.1 Algebraic-coding methods

In this section we explore the use of various advanced algebraic techniques that improve the performance of NVM devices. We focus on constructions that have a special property: while offering a more focused error correction capability than blanket symmetric error correction, the resulting codes are in fact built by carefully combining constituent symmetric error correcting codes. The resulting codes thus inherit desirable structural properties (e.g., rate) and well-known decoding methods of the underlying constituent codes. We present two prime exemplars of such constructions, graded-bit-error correcting codes and asymmetric limited-magnitude error correcting codes.

3.1.1 Graded-bit-error correcting codes

The motivation for introducing this new class of algebraic codes came from experimental measurements on a TLC flash device [66; 151]. These measurements revealed strong asymmetry among the error patterns. Recall that memory cells are organized into pages and that pages are grouped into blocks. The data collection was performed on a TLC de-

vice with 8KB of data per page, and 384 pages per block (multiple blocks constitute a plane, and several planes form the overall TLC chip). In each P/E cycle, blocks are erased and random data is written. As a matter of convenience, on the first out of every 100 such cycles, the data is read back after being written and subsequently checked for errors. The overall procedure served to model the aging of the TLC device.

Since a TLC device stores 3 bits per cell, we can refer to these 3 bits as MSB, CSB, and LSB, where MSB stands for the most significant bit, CSB stands for the center significant bit, and LSB stands for the least significant bit. To minimize the errors, levels are, as shown in Figure 3.1, mapped into Gray-coded symbols, starting with 111 for the lowest (unprogrammed) level (see also Figure 2.2). It was observed in [151] that the bit error rate (BER) increases over time; as previously described, wearout makes the memory cells less reliable. Furthermore, the overall error rate was recorded as roughly the sum of the error rates across the three bits (MSB, CSB, and LSB); that is, when a cell error occurs, it is mostly due to having a single bit in error. A more detailed investigation of the error types revealed that about 96% of symbol errors were indeed due to one bit being in error, about 3% of symbol errors were due to two bits being in error, and the remaining fraction (less than 1%) of errors are due to all three bits being in error.

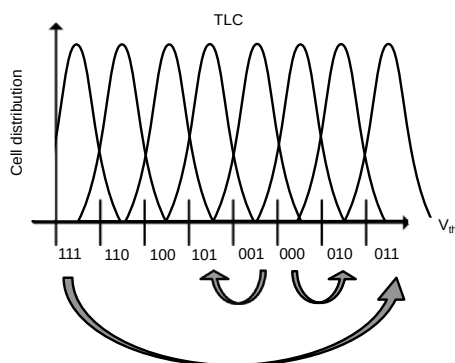


Figure 3.1: Gray coded TLC flash cell. Arrows indicate some of possible erroneous transitions.

This interesting error distribution can be in part attributed to the programming sequence wherein MSB bits are programmed first, followed by programming of CSB bits, and then LSB bits: contiguous MSB, CSB, and LSB physical pages correspond to non-contiguous MSB, CSB, and LSB logical pages [151; 154]. For example, suppose we wish to write $(1, 1, 1)$ into a TLC cell. Suppose that the MSB was erroneously programmed into 0 so the read value is $(0, 1, 1)$. As a result, we would observe a single bit error. Multiple bit errors, albeit far less likely, can be caused by a combination of events: programming error, intercell interference, wearout, and so on.

Since the majority of the errors are single bit errors, a reasonable approach might be to use three (possibly distinct) binary codes to protect each of the MSB, CSB, and LSB pages separately. However, a closer inspection of the collected data reveals the fact that this approach is not suitable: treating each of the errors in the MSB, CSB, and LSB pages as the result of separate, independent binary channels underestimates the number of symbol errors which contain multiple bits in error. This occurs because errors impact cells, in which the MSB, CSB, and LSB are stored together simultaneously. If our goal is to reach a particular overall symbol error rate target, we would have to overprovision by making the independent binary codes stronger, reducing the overall rate. As a result, we require a single non-binary code to protect the (MSB, CSB, LSB) triplet simultaneously.

We are therefore tasked with designing a code that can correct a large number of single bit per symbol errors and some smaller number of multiple bit per symbol errors. This is in contrast to the conventional symmetric error correcting coding schemes – including popular BCH codes – that do not differentiate among different symbol-to-symbol errors, and instead focus on the total error correction capability, expressed either in symbols or in bits. Our goal is therefore to offer a third alternative to parallel binary codes and standard non-binary codes. One such designed code is called a *graded-bit-error correcting code* and was introduced in [151] and further developed in [64]. The code uses tensor product operations to control the target error correction. In the sequel, we summarize the key properties of the graded-

bit-error correcting codes and discuss their benefits over conventional BCH-like designs.

In order to introduce a code that captures the fine-grained features we are after, let us first specify the error patterns we are interested in. We focus on memory devices storing m bits per cell; therefore $q = 2^m$ levels are needed to distinguish all possible values. Since we are interested in controlling for a prescribed number of erroneous bits per erroneous symbol, let us express each 2^m -ary cell in its binary representation with m bits. We concatenate values from n 2^m -ary cells to produce a length nm binary sequence in $(GF(2)^m)^n$. Here, bits $m(i-1)+1, \dots, m(i-1)+m$ represent the i th cell for $1 \leq i \leq n$. We write $wt(\mathbf{x}) = |\{i : \mathbf{x}_i \neq \mathbf{0}\}|$ to denote the weight (number of nonzero components) of a vector $\mathbf{x}$.

Definition 1. Given positive integers t and ℓ , an error vector $\mathbf{e} = (e_1, e_1, \dots, e_n)$ over $(GF(2)^m)^n$ is called a $[t; \ell]$ -**bit-error vector** if the following holds:

1. $wt(\mathbf{e}) \leq t$, and
2. $\forall i, wt(e_i) \leq \ell$.

Definition 2. A linear code $\mathcal{C}$ that can correct every $[t; \ell]$ bit-error vector is called a $[t; \ell]$ -**bit-error correcting code**.

The previous definitions constrain the error correction to patterns with the maximum number of erroneous bits in a particular erroneous cell. Since we are also interested in the cases where some (albeit fewer) cells may have more than one bit in error, the following definitions offer a more refined characterization of error patterns of interest.

Definition 3. Let $0 < \ell_1 < \ell_2 \leq m$, $t_1, t_2 > 0$. Then, a vector $\mathbf{e} = (e_1, e_2, \dots, e_n)$ over $(GF(2)^m)^n$ is called a $[t_1, t_2; \ell_1, \ell_2]$ -**graded-bit-error vector** if the following holds:

1. $wt(\mathbf{e}) \leq t_1 + t_2$,
2. $\forall i, wt(e_i) \leq \ell_2$, and
3. $|\{i : wt(e_i) > \ell_1\}| \leq t_2$.

Graded-bit-error correcting codes are described as follows.

Definition 4. Let $0 < \ell_1 < \ell_2 \leq m$, $t_1, t_2 > 0$. Then, a code $\mathcal{C}$ is said to be a $[t_1, t_2; \ell_1, \ell_2]$ -**graded-bit-error correcting code** if it can correct every $[t_1, t_2; \ell_1, \ell_2]$ -graded-bit-error-vector.

It is clear that these error vector definitions offer a more refined error characterization than what is afforded by the conventional designs that primarily optimize for the total Hamming distance. Here is an example. Suppose the following data was stored in 6 TLC cells: (000 110 010 101 000 111) (we group the values into triplets, with each triplet corresponding to data stored in one TLC cell). Suppose the following was retrieved: (101 110 000 101 000 011). The difference is (101 000 010 000 000 100). If we were to only consider symbol errors, we would declare that the pattern has 3 erroneous symbols, and we would seek to design 2^3 -ary codes that can correct t symbol errors for some target t . Indeed, this is exactly what BCH codes offer. With the error vector definitions above, we can be more specific. The error pattern (101 000 010 000 000 100) can be classified as a $[3; 2]$ -bit-error vector, in which case we would seek to design a $[t; \ell]$ -bit-error correcting code for target t and ℓ . Taking it further, we observe that the error pattern can also be classified as a $[2, 1; 1, 2]$ -graded-bit-error vector since there are $2 + 1$ ($t_1 + t_2$) symbols in error, all symbols have at most 2 (ℓ_2) erroneous bits, and there is 1 (t_2) cell with more than 1 (ℓ_1) erroneous bit. We also remark that the above error pattern can also be classified as a $[t; \ell]$ -bit-error-vector for any allowable choice of $t \geq 3$ and $\ell \geq 2$, for example, it can also be classified as a $[3; 3]$ -bit-error vector, or a $[4; 2]$ -bit-error vector. A similar property of course holds for the error pattern classification in terms of t_1, t_2, ℓ_1 , and ℓ_2 . This classification is consistent with our target error correction capability: we seek to correct all errors that are within a certain distance from the stored codeword, and all such errors can be simultaneously described by the same (maximal) t and ℓ (or t_1, t_2, ℓ_1 , and ℓ_2).

It is helpful to recognize that an appropriately specified tensor product of two parity check matrices (of conventional symmetric error correcting codes) indeed captures the symbol error/bit error count.

We use the notation $[n, k, t]_q$ to denote a linear code of length n , dimension k (that is, there are q^k codewords) correcting t errors over an alphabet of size q . Let H_1 be a parity check matrix of a binary $[m, k_1, \ell]_2$ code $\mathcal{C}_1$, and let H_2 be a parity check matrix of a 2^{m-k_1} -ary $[n, k_2, t]_{2^{m-k_1}}$ code $\mathcal{C}_2$.

We then have the following result.

Construction 1. Let $\mathcal{C}_A$ be a code with the parity-check matrix

$$H_A = [H_2 \otimes H_1]. \quad (3.1)$$

We will show that the code built according to Construction 1 is indeed a $[t; \ell]$ -bit-error correcting code $\mathcal{C}_A$.

This construction is due to Wolf, who originally proposed it for burst error correction [147]. It was recently recognized also by Wolf [148] that this construction can be used in modern data storage systems.

Observe that the order of the finite field over which $\mathcal{C}_2$ is defined is governed by the dimension of the parity check matrix of $\mathcal{C}_1$: $m - k_1$. A very practical feature of this construction is that a code design targeting more refined error patterns can be achieved using two constituent codes designed to correct symmetric errors.

Here is an example of this code construction. Before we begin, we note that the operations are always performed in $GF(2^{m-k_1})$. However, the codewords in $\mathcal{C}_A$ are required to be binary. These ideas will be clarified in the example.

Example 3.1. Let $\mathcal{C}_1$ be a Hamming code $[3, 1, 1]_2$ with the parity check matrix $H_1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$. Let $\mathcal{C}_2$ be a 2^2 -ary code $[4, 2, 1]_{2^2}$ with the parity check matrix $H_2 = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & \alpha \end{bmatrix}$, where α is a primitive element of $GF(2^2)$, and thus satisfies $\alpha^2 = \alpha + 1$. We can express the elements in $GF(2^2)$ as $\alpha^0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\alpha^1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $\alpha^2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, and $0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$. Then, $H_1 = [1 \ \alpha \ \alpha^2]$ so that

$$H_A = H_2 \otimes H_1 = \begin{bmatrix} 1 & \alpha & \alpha^2 & 0 & 0 & 0 & 1 & \alpha & \alpha^2 & 1 & \alpha & \alpha^2 \\ 0 & 0 & 0 & 1 & \alpha & \alpha^2 & 1 & \alpha & \alpha^2 & \alpha & \alpha^2 & 1 \end{bmatrix}.$$

The binary representation of H_A is $H_{A|\text{bin}}$, and is given by

$$H_{A|\text{bin}} = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}.$$

By the rank properties of the tensor product operation, H_A has rank 2, i.e., it is full rank. We partition the columns of H_A into 4 groups (here $n = 4$) of 3 symbols (here $m = 3$) so that columns 1, 2, and 3 constitute the first group, columns 4, 5, and 6 constitute the second group, columns 7, 8, and 9 constitute the third group, and columns 10, 11, and 12 constitute the fourth group. Observe that since all columns of $H_{A|\text{bin}}$ are distinct, $\mathcal{C}_A$ can indeed correct 1 bit error. On the other hand, $\mathcal{C}_A$ cannot always correct a single symbol error. For example, error vectors $(000 \ 000 \ \alpha^2 00 \ 000)^T$ and $(000 \ 000 \ 001 \ 000)^T$ produce the same syndrome $(\alpha^2 \ \alpha^2)^T$. However, the critical idea here is that we need not correct all symbol errors; we are interested only in binary codewords and binary error vectors of length mn . In fact, we require codewords in $\mathcal{C}_A$ to be binary; and we perform the operations $0 \cdot x = 0$ and $1 \cdot x = x$ for elements $x \in GF(2^{m-k_1})$. We use this convention throughout the remainder of this section.

We now demonstrate that a code $\mathcal{C}_A$ built according to Construction 1 is indeed a $[t; \ell]$ -bit-error correcting code; the proof will essentially follow from the decoding algorithm which we describe next.

The decoding algorithm is based on the 2-step syndrome decoder detailed in [147] and [64]. All operations will be performed in $GF(2^{m-k_1})$. The input to the decoder is $\mathbf{y}$, where $\mathbf{y} = \mathbf{c} + \mathbf{e}$ for $\mathbf{c}$ a codeword in $\mathcal{C}_A$ and $\mathbf{e}$ a $[t, \ell]$ -bit-error vector. It is convenient to write $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n)$, $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_n)$, and $\mathbf{e} = (\mathbf{e}_1, \mathbf{e}_2, \dots, \mathbf{e}_n)$, where for each i , $1 \leq i \leq n$, $\mathbf{y}_i$, $\mathbf{c}_i$, and $\mathbf{e}_i \in GF(2^m)$. The output of the first step is based on syndrome decoding for the constituent code

$\mathcal{C}_2$; this output, consisting of a collection of syndromes, is then fed into the second step of the decoder, itself a syndrome decoder for $\mathcal{C}_1$. The output of the second syndrome decoder produces an estimate of $\mathbf{e}$.

It is helpful to first observe that $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_n)$ is a codeword of $\mathcal{C}_A$ if and only if $(H_1 \cdot \mathbf{c}_1^T, H_1 \cdot \mathbf{c}_2^T, \dots, H_1 \cdot \mathbf{c}_n^T)$ is a codeword in $\mathcal{C}_2$. Let us quickly verify this convenient fact. Write $H_A \cdot \mathbf{c}^T = [H_2 \otimes H_1] \cdot \mathbf{c}^T$. Let $r = m - k_1$ and $r_2 = n - k_2$ denote the numbers of rows in H_1 and H_2 , respectively. Recall that H_2 is an $r_2 \times n$ matrix over $GF(2^r)$. We may write

$$\begin{aligned} H_A \cdot \mathbf{c}^T &= \begin{bmatrix} h_{1,1}H_1 & h_{1,2}H_1 & \dots & h_{1,n}H_1 \\ h_{2,1}H_1 & h_{2,2}H_1 & \dots & h_{2,n}H_1 \\ & & \ddots & \\ h_{r_2,1}H_1 & h_{r_2,2}H_1 & \dots & h_{r_2,n}H_1 \end{bmatrix} \cdot \mathbf{c}^T \\ &= \begin{bmatrix} h_{1,1} & h_{1,2} & \dots & h_{1,n} \\ h_{2,1} & h_{2,2} & \dots & h_{2,n} \\ & & \ddots & \\ h_{r_2,1} & h_{r_2,2} & \dots & h_{r_2,n} \end{bmatrix} \cdot \begin{bmatrix} H_1 \cdot \mathbf{c}_1^T \\ H_1 \cdot \mathbf{c}_2^T \\ \vdots \\ H_1 \cdot \mathbf{c}_n^T \end{bmatrix}, \end{aligned}$$

where each $h_{i,j}$ entry, $1 \leq i \leq r_2$, $1 \leq j \leq n$, is an entry from the matrix H_2 . Note that H_1 is a length- m row vector of elements in $GF(2^r)$, so that $H_1 \cdot \mathbf{c}_j^T$, $1 \leq j \leq n$ produces a symbol in $GF(2^r)$. Now, if

$$H_A \cdot \mathbf{c}^T = 0, \text{ we must also have } H_2 \cdot \begin{bmatrix} H_1 \cdot \mathbf{c}_1^T \\ H_1 \cdot \mathbf{c}_2^T \\ \vdots \\ H_1 \cdot \mathbf{c}_n^T \end{bmatrix} = 0, \text{ so that indeed}$$

$(H_1 \cdot \mathbf{c}_1^T, H_1 \cdot \mathbf{c}_2^T, \dots, H_1 \cdot \mathbf{c}_n^T)$ is a codeword in $\mathcal{C}_2$.

Let us now return to the overall decoder. Upon receiving $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n)$, the two-step syndrome decoder computes in the first step the syndrome $(\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_n) = \mathcal{D}_2(H_2 \cdot (H_1 \cdot \mathbf{y}_1^T, H_1 \cdot \mathbf{y}_2^T, \dots, H_1 \cdot \mathbf{y}_n^T))$. Here, $\mathcal{D}_2$ is the decoding map for code $\mathcal{C}_2$. Recall that $(H_1 \cdot \mathbf{c}_1^T, H_1 \cdot \mathbf{c}_2^T, \dots, H_1 \cdot \mathbf{c}_n^T)$ is a codeword in $\mathcal{C}_2$ and that $(H_1 \cdot \mathbf{y}_1^T, H_1 \cdot \mathbf{y}_2^T, \dots, H_1 \cdot \mathbf{y}_n^T) = (H_1 \cdot \mathbf{c}_1^T, H_1 \cdot \mathbf{c}_2^T, \dots, H_1 \cdot \mathbf{c}_n^T) + (H_1 \cdot \mathbf{e}_1^T, H_1 \cdot \mathbf{e}_2^T, \dots, H_1 \cdot \mathbf{e}_n^T)$.

Now, the vector $(H_1 \cdot \mathbf{e}_1^T, H_1 \cdot \mathbf{e}_2^T, \dots, H_1 \cdot \mathbf{e}_n^T)$ has weight at most t , so that the decoder $\mathcal{D}_2$ can indeed retrieve the correct codeword $(H_1 \cdot$

$\mathbf{c}_1^T, H_1 \cdot \mathbf{c}_2^T, \dots, H_1 \cdot \mathbf{c}_n^T$). Let us say, for convenience, that the decoding map produces an error estimate, rather than the estimated codeword. Then, we may write that $(\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_n) = (H_1 \cdot \mathbf{e}_1^T, H_1 \cdot \mathbf{e}_2^T, \dots, H_1 \cdot \mathbf{e}_n^T)$.

All that remains is for us to decode each of the $H_1 \cdot \mathbf{y}_i^T$ individually. Since we have found $\mathbf{s}_i = H_1 \cdot \mathbf{e}_i^T$, for all $1 \leq i \leq n$, we have the syndromes for codewords in $\mathcal{C}_1$. Therefore, we can use the decoding map $\mathcal{D}_1$ to find $\mathbf{e}_i$ for all $1 \leq i \leq n$ whenever the weight of $\mathbf{e}_i$ is at most ℓ . This concludes the decoding process.

Motivated by the error patterns discussed earlier wherein a relatively small fraction of cells have a large number of bits in error, and inspired by the compactness of the Wolf construction, the following code construction was proposed in [64] for the correction of error patterns encountered in multilevel flash devices. With this more refined error characterization, additional care needs to be taken when choosing constituent symmetric error correcting codes.

Let us again start with a binary code as one of our constituent codes. Suppose $\mathcal{C}_1$ is an $[m, k, \ell_2]_2$ code with a parity check matrix H_1 . (We have now replaced ℓ by ℓ_2 since we need both ℓ_2 and ℓ_1 in the construction of a $[t_1, t_2; \ell_1, \ell_2]$ -graded-bit-error correcting code.) Let $r = m - k$. Suppose H_1 is chosen such that the topmost r_3 rows of H_1 constitute a parity check matrix of an $[m, m - r_3, \ell_1]_2$ code for some $r_3 < r$. Call this code $\mathcal{C}_3$ and this parity check matrix H_3 . Let H_5 be the submatrix of H_1 consisting of the bottom $r_5 = r - r_3$ rows of H_1 , i.e., $H_1 = \begin{bmatrix} H_3 \\ H_5 \end{bmatrix}$.

Let H_2 be a parity check matrix of an 2^{r_3} -ary $[n, k_2, t_1 + t_2]_{2^{r_3}}$ code $\mathcal{C}_2$ and let H_4 be a parity check matrix of an 2^{r_5} -ary $[n, k_4, t_2]_{2^{r_5}}$ code $\mathcal{C}_4$. Note that dimensions k_2 and k_4 of codes $\mathcal{C}_2$ and $\mathcal{C}_4$ are whatever values are needed to achieve target error correction capability ($t_1 + t_2$ and t_2 , respectively) of these codes at length n .

We now have the following.

Construction 2. The code $\mathcal{C}_B$ with the parity check matrix

$$H_B = \begin{bmatrix} H_2 \otimes H_3 \\ H_4 \otimes H_5 \end{bmatrix} \quad (3.2)$$

is a $[t_1, t_2; \ell_1, \ell_2]$ -graded-bit-error correcting code $\mathcal{C}_B$ of length n . Here, the operation $H_2 \otimes H_3$ is performed in $GF(2^{r_3})$, while $H_4 \otimes H_5$ is performed in $GF(2^{r_5})$ and the result is afterwards converted to $GF(2^{r_3})$.

We start off our discussion with an illustrative example.

Example 3.2 (see also [64; 151]). Let $H_3 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$ be a parity check matrix for a $[3, 1, 1]_2$ Hamming code and let $H_5 = [1 \ 1 \ 1]$ be a parity check matrix of a single parity check code. The parity check matrix $H_1 = \begin{bmatrix} H_3 \\ H_5 \end{bmatrix}$ then corresponds to a “trivial” code $\mathcal{C}_1$, that has all-zeros as its only codeword; the code has length 3 and corrects 3 errors. Note that $\ell_1 = 1$, $\ell_2 = 3$, $r_3 = 2$, and $r_5 = 1$.

Let $\mathcal{C}_2$ be a $[15, 9, 2]_4$ code with a parity check matrix H_2 ,

$$H_2 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & \alpha^2 & \alpha & \alpha & \alpha^2 & \alpha & 0 & \alpha^2 & \alpha^2 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & \alpha^2 & 0 & \alpha & 1 & \alpha & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & \alpha^2 & \alpha^2 & \alpha & \alpha & 1 & 0 & 1 & \alpha & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & \alpha^2 & \alpha^2 & \alpha & \alpha & 1 & 0 & 1 & \alpha \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & \alpha^2 & 0 & \alpha & 1 & \alpha & 0 & 1 & \alpha \\ 0 & 0 & 0 & 0 & 0 & 1 & \alpha^2 & \alpha^2 & 1 & \alpha^2 & 0 & 1 & 1 & \alpha^2 & \alpha^2 \end{bmatrix}.$$

Here α is the primitive element of $GF(2^2)$; $\alpha^2 = \alpha + 1$. Furthermore, let $\mathcal{C}_5$ be a $[15, 11, 1]_2$ Hamming code with a parity check matrix H_4 . Then, using Construction 2, the code $\mathcal{C}_B$ has a parity check matrix

$$H_B = \begin{bmatrix} H_2 \otimes H_3 \\ H_4 \otimes H_5 \end{bmatrix}. \quad (3.3)$$

According to our construction, H_B is a parity check matrix for a $[1, 1; 1, 3]$ -graded-bit-error correcting code. We can express elements in $GF(2^2)$ as $\alpha^0 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$, $\alpha^1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $\alpha^2 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$, and $0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, as we have done in Example 3.1.

As a result, we may express H_3 as $(\alpha^0 \ \alpha \ \alpha^2)$. We perform the operation $H_2 \otimes H_3$ over $GF(2^2)$. The output is a 6×45 matrix over $GF(2^2)$. The operation $H_4 \otimes H_5$ is performed over $GF(2)$, and its output

is a 4×45 matrix over $GF(2)$. We then convert this output to a 2×45 matrix over $GF(4)$, allowing us to form a final H_B .

Expressing the quaternary symbols as binary vectors of length 2, we can now stack up the two tensor products and produce the binary parity check matrix H_B of size 16×45 .

The proof of correctness of Construction 2, as for Construction 1, relies on specifying the multi-step syndrome decoding and proving that the decoder indeed produces the correct codeword as long as the errors fall into the class of graded bit error patterns. Here are the key steps. We first perform syndrome decoding for $\mathcal{C}_2$ based on the received word. The output of this decoder will correctly identify the locations of erroneous 2^{r_3} -ary symbols since the code $\mathcal{C}_2$ can correct up to $t_1 + t_2$ symbol errors (and by definition the error vector is a $[t_1, t_2; \ell_1, \ell_2]$ -graded-bit-error vector). The output will be a length- n vector over $GF(2^{r_3})$. We then apply the syndrome decoder for $\mathcal{C}_3$ to each of these n symbols, viewing them each as a binary syndrome. Since the error correction capability of $\mathcal{C}_3$ is ℓ_1 bits, the errors with weight at most ℓ_1 will be correctly identified. However, errors are allowed to have weight up to ℓ_2 , by construction of the error vector. The errors that have weight larger than ℓ_1 will be miscorrected. The output of the syndrome decoder for $\mathcal{C}_3$ for such errors will be binary words of length m and weight at most $\ell_1 + \ell_2$. Call the result of this step $\mathbf{y}_2$ (the syndrome decoder for $\mathcal{C}_3$ is invoked n times). Now, since the code $\mathcal{C}_1$ can correct up to ℓ_2 erroneous bits, the minimum weight of a non-zero codeword in $\mathcal{C}_1$ is $2\ell_2 + 1$. Since $\ell_1 + \ell_2 < 2\ell_2 + 1$ by construction, all such errors can be detected by the decoder of $\mathcal{C}_1$ and the syndrome decoder for $\mathcal{C}_4$ can be used to correct identified error groups (up to t_2 of them) of weight at most ℓ_2 each.

We observe that every $[t_1 + t_2; \ell_2]$ -bit-error correcting code, given by Construction 1, is also a $[t_1, t_2; \ell_1, \ell_2]$ -graded-bit-error correcting code (here, as usual $\ell_1 < \ell_2$), given by Construction 2; the other direction does not hold. Several extensions of the presented construction are possible: instead of just using constituent codes that correct errors, we may also employ error detecting and error detecting/correcting codes as constituents – the overall error correction/detection capability will (in a suitable sense) carry over to the resultant code, see also [64; 147;

148]. This extension now gives us a mix-and-match flexibility to pick appropriate constituent codes to achieve error protection against target error patterns.

We now provide simulation results in Figure 3.2 illustrating the benefits of using algebraic codes tailored to error patterns encountered in flash. The following codes were simulated: an 2^3 -ary extended

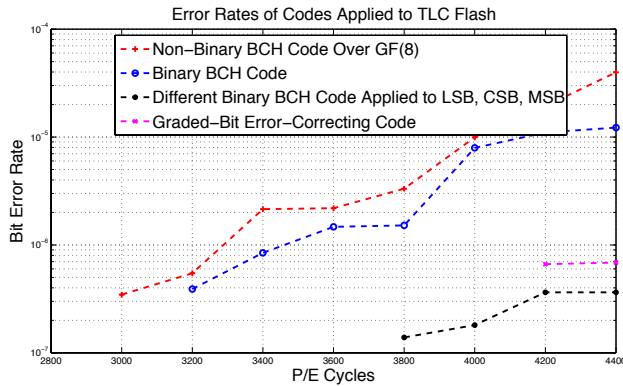


Figure 3.2: Simulation results showing the benefits of using graded-bit error correcting codes in flash. Figure derived based on results from [64].

BCH code of length $n = 4096$, dimension $k = 3534$ that corrects $t = 80$ errors (red curve), a binary extended BCH code of length $n = 4096$, dimension $k = 3531$ that corrects $t = 41$ errors applied to each of the MSB/CSB/LSB pages separately (blue curve), a code consisting of three BCH codes, with relative error correction capabilities chosen according to observed error patterns (black curve): $[n, k, t]_2$ is $[4096, 3915, 15]_2$ for MSB, $[4096, 3339, 63]_2$ for CSB, and $[4096, 3351, 62]_2$ for LSB (recall that MSB/CSB/LSB stands for most/center/least significant bit). The graded-bit-error correcting code (pink curve) is also of length 4096, and it corrects up to 88 symbol errors of which at most 7 have up to 3 bits in error, and the rest have 1 bit in error. All the codes have approximately the same rate, 0.86. Observe that the graded-bit-error correcting code defers the onset of

errors and thus significantly extends the lifetime of the memory device beyond what is afforded by standard codes; in this example, the lifetime is extended by about 40%.

3.1.2 Asymmetric, limited-magnitude error correcting codes

As previously discussed, there is strong operational asymmetry in flash devices. Recall from Chapter 2 that one practical way to circumvent the particularly expensive block erase operation during programming is to employ ISPP – this programming technique however requires many small increments of adding charge, which for programming multilevel cells can translate into unacceptable write latency. What if we were to allow programming errors to occur but use appropriately defined codes to correct them? This fundamental question was explored in [30], and we now summarize the main findings of this award-winning work.

Let us first observe that the programming errors (as well as errors caused by inter-cell interference) can only occur in the upward direction. Moreover, these errors are also typically of limited magnitude. Suppose we seek to correct errors of limited magnitude 1. One naïve approach would be to only encode information into odd levels. Then, when reading out a group of cells, the parity of the cell value would immediately help us conclude where the errors are. However, this approach is fairly wasteful for our target application as we are only using roughly half of the available levels for encoding; in this section we shall describe a coding method due to [30] that achieves the same goal (in fact, it does so even more generally), albeit with far less redundancy.

As in the previous subsection, let us first formalize the notion of the error vector.

Definition 5. A vector $e \in Q^n$ of non-negative integers $e = (e_1, e_2, \dots, e_n)$ is called a t -asymmetric, ℓ -limited magnitude error vector if

1. $|i : e_i \neq 0| \leq t$, and
2. $\forall i, 0 \leq e_i \leq \ell$.

For example, suppose $n = 6$, $Q = \mathbb{Z}_8$, and the error vector e is specified as

$$e = (0 \quad 1 \quad 2 \quad 0 \quad 0 \quad 3).$$

Then, the vector $\mathbf{e}$ can be classified as a 3-asymmetric, 3-limited magnitude error vector. As in the discussion on the graded-bit-error correcting codes, the error vector of interest can be parametrized by different choices of its parameters. For example, this $\mathbf{e}$ can also be viewed as a 3-asymmetric, 4-limited magnitude error vector, or as a 4-asymmetric, 4-limited magnitude error vector, among other choices.

Since we are interested in correcting this specialized type of errors in the context of NVMs where there is a maximum charge stored per cell, it is natural to adopt the following channel model. Suppose $\mathbf{x} \in Q^n$ is stored. The vector $\mathbf{y} = \mathbf{x} + \mathbf{e} \in Q^n$ is retrieved. The condition $\mathbf{y} \in Q^n$ ensures that the maximal charge in the stored cells does not exceed the physical limits of the cells.

In order to develop codes suitable for this channel, we need to first define an appropriate notion of distance.

Definition 6. Let $\mathbf{x} = (x_1, x_2, \dots, x_n) \in Q^n$, and $\mathbf{z} = (z_1, z_2, \dots, z_n) \in Q^n$. Define $N(\mathbf{x}, \mathbf{z}) = |\{i : x_i > z_i\}|$. Then,

$$d_\ell(\mathbf{x}, \mathbf{z}) = \begin{cases} n + 1 & \text{if } \max_{1 \leq i \leq n} |x_i - z_i| > \ell, \\ \max\{N(\mathbf{x}, \mathbf{z}), N(\mathbf{z}, \mathbf{x})\} & \text{otherwise.} \end{cases}$$

For example, suppose $n = 6$, $Q = \mathbb{Z}_8$, and $\ell = 2$. Let $\mathbf{x} = (0 \ 1 \ 0 \ 2 \ 2 \ 3)$ and $\mathbf{z} = (0 \ 2 \ 1 \ 2 \ 3 \ 1)$. Then, $N(\mathbf{x}, \mathbf{z}) = 1$, and $N(\mathbf{z}, \mathbf{x}) = 3$. Also, $\max_{1 \leq i \leq 6} \{|x_i - z_i|\} = 2 \leq \ell$. Thus, $d_\ell(\mathbf{x}, \mathbf{z}) = 3$.

Let $\mathbf{u} = (0 \ 2 \ 3 \ 2 \ 3 \ 1)$. Note that $\max_{1 \leq i \leq 6} \{|x_i - u_i|\} = 3 > \ell$. Thus, $d_\ell(\mathbf{x}, \mathbf{u}) = 7$.

We now show that this refined definition of distance is indeed the right measure in the context of the errors we seek to correct.

Lemma 3.1. A code $\mathcal{C} \subset Q^n$ can correct t -asymmetric, ℓ -limited magnitude errors if and only if $d_\ell(\mathbf{x}, \mathbf{z}) \geq t + 1$, $\forall \mathbf{x}, \mathbf{z} \in \mathcal{C}$, $\mathbf{x} \neq \mathbf{z}$.

First, a code $\mathcal{C}$ fails to correct such errors if $\exists \mathbf{x}, \mathbf{z} \in \mathcal{C}$, $\mathbf{x} \neq \mathbf{z}$, such that $\mathbf{x} + \mathbf{e} = \mathbf{z} + \mathbf{f}$, where $\mathbf{e}$ and $\mathbf{f}$ are t -asymmetric, ℓ -limited magnitude error vectors. We prove the claim by contradiction.

Let us first suppose that $\mathbf{x}, \mathbf{z} \in \mathcal{C}$ such that $d_\ell(\mathbf{x}, \mathbf{z}) \geq t + 1$. Now, if $d_\ell(\mathbf{x}, \mathbf{z}) \geq t + 1$ it must be the case that either 1) $\max\{N(\mathbf{x}, \mathbf{z}), N(\mathbf{z}, \mathbf{x})\} > t$ or 2) $\exists i, 1 \leq i \leq n$ such that $|x_i - z_i| > \ell$.

In case 1) we have that either $N(\mathbf{x}, \mathbf{z}) > t$ or $N(\mathbf{z}, \mathbf{x}) > t$. Now if $\mathbf{x} + \mathbf{e} = \mathbf{z} + \mathbf{f}$, then $\mathbf{x} - \mathbf{z} = \mathbf{f} - \mathbf{e}$. Then, if $N(\mathbf{x}, \mathbf{z}) > t$, $\mathbf{x} - \mathbf{z}$ has more than t positive entries. However, $\mathbf{f}$ and $\mathbf{e}$ by assumption can have at most t positive entries each so that $\mathbf{f} - \mathbf{e}$ also has at most t positive entries, and we have reached a contradiction. If $N(\mathbf{z}, \mathbf{x}) > t$, the proof follows analogously.

In case 2), we have that either $x_i - z_i > \ell$ or $z_i - x_i > \ell$. For the former, it then follows that $f_i - e_i > \ell$ and for the latter, it likewise follows that $e_i - f_i > \ell$. Neither choice is possible, as these choices would imply that either $\mathbf{e}$ or $\mathbf{f}$ has an entry greater than ℓ , which is not possible. This completes the ‘if’ direction of the proof.

Now suppose that $d_\ell(\mathbf{x}, \mathbf{z}) \leq t$. We need to show that there exist two distinct codewords $\mathbf{x}$ and $\mathbf{z}$ that cannot be uniquely recovered under some error vectors $\mathbf{e}$ and $\mathbf{f}$, respectively. Since the distance is at most t , it must be the case that $\forall i, 1 \leq i \leq n, |x_i - z_i| \leq \ell$. Therefore, $\max\{N(\mathbf{x}, \mathbf{z}), N(\mathbf{z}, \mathbf{x})\} \leq t$, that is, $N(\mathbf{x}, \mathbf{z}) \leq t$ and $N(\mathbf{z}, \mathbf{x}) \leq t$. Let us now construct $\mathbf{e}$ and $\mathbf{f}$ as follows. In positions i , $1 \leq i \leq n$, where $x_i > z_i$, we set $f_i = x_i - z_i$. In positions i , $1 \leq i \leq n$, where $z_i > x_i$, we set $e_i = z_i - x_i$. In all other positions, we set $e_i = f_i = 0$. Note that this choice of $\mathbf{e}$ and $\mathbf{f}$ is legal as both vectors have at most t non-zero entries and all entries have value at most ℓ .

We are now ready to present a construction of a code capable of correcting t -asymmetric, ℓ -limited magnitude errors.

Construction 3. Suppose $\tilde{\mathcal{C}}$ is a code of length n over the alphabet $\tilde{A}$, with the alphabet size $\tilde{q} = |\tilde{A}| = \ell + 1$. Define code $\mathcal{C}$ of length n over alphabet A of size $q = |A|$, with $q > \ell + 1$, as

$$\mathcal{C} = \{\mathbf{x} = (x_1, x_2, \dots, x_n) \in A^n \mid \mathbf{x} \bmod \tilde{q} \in \tilde{\mathcal{C}}\}.$$

We now prove that this code construction indeed possesses the target error correction capability.

Theorem 3.2. Code $\mathcal{C}$ is t -asymmetric, ℓ -limited magnitude-error correcting code if code $\tilde{\mathcal{C}}$ can correct t symmetric errors. Additionally, if $q > 2\ell$, the converse is also true.

To prove the ‘if’ part, suppose first that $\mathbf{x}, \mathbf{z} \in \mathcal{C}$, and $\mathbf{x} \neq \mathbf{z}$ map to the same codeword $\mathbf{u} \in \tilde{\mathcal{C}}$ via the modding operation. Then

it must be the case that $\forall i, 1 \leq i \leq n, |x_i - z_i|$ is an integer multiple of $\tilde{q} = \ell + 1$, and there exists at least one such i where $|x_i - z_i|$ is strictly positive. Then, as a result $d_\ell(\mathbf{x}, \mathbf{z}) = n + 1$, and $\mathbf{x}$ and $\mathbf{z}$ cannot be confused under the t -asymmetric, ℓ -limited magnitude-error model.

Suppose now that $\mathbf{x}, \mathbf{z} \in \mathcal{C}$ map to two different codewords in $\tilde{\mathcal{C}}$. Call these codewords $\tilde{\mathbf{x}}$ and $\tilde{\mathbf{z}}$. By definition of code $\tilde{\mathcal{C}}$, all of its codewords must differ in at least $2t + 1$ places. Since the (inverse) mod operation in the presented construction preserves these differences, it then also must be the case that the Hamming distance between $\mathbf{x}$ and $\mathbf{z}$ is at least $2t + 1$. If $\mathbf{x}$ and $\mathbf{z}$ differ in at least $2t + 1$ locations, it moreover must be the case that at least one of $N(\mathbf{x}, \mathbf{z})$ and $N(\mathbf{z}, \mathbf{x})$ is at least $t + 1$. (Furthermore, in the positions where they differ, the gap may be larger than ℓ). As a result, again, $\mathbf{x}$ and $\mathbf{z}$ cannot be confused under the t -asymmetric, ℓ -limited magnitude-error model, and we conclude the first part of the proof.

We leave the ‘only if’ part to the reader (see also [30]).

Note that in the above we set $\tilde{q} = \ell + 1$; it is clear that $\tilde{q} > \ell$ is sufficient for the resultant construction to yield target error correction capability.

Note that like in the previous section on (graded)-bit-error correcting codes, we can build codes with target error correction capability using existing symmetric error correcting codes. Here is one example.

Example 3.3. Suppose we seek to construct a code over $\mathbb{Z}_6$ of length $n = 13$ that corrects one error of limited magnitude 2, i.e., $t = 1$ and $\ell = 2$. It suffices to identify a ternary single error correcting code to be the constituent code $\tilde{\mathcal{C}}$ in Construction 3. One such code is a single error correcting Hamming code over $GF(3)$. This code corrects 1 error and can be described by a 3×13 parity check matrix

$$\tilde{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 2 & 1 & 2 & 2 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 0 & 2 & 1 & 2 \end{bmatrix}.$$

Our resultant code $\mathcal{C}$ is of length $n = 13$, and consists of all strings $\mathbf{x} \in \mathbb{Z}_6^{13}$ such that $\mathbf{x} \bmod 3 \in \tilde{\mathcal{C}}$. Here are some possible codewords:

(2 2 2 0 0 0 1 0 0 0 0 0 0),
 (5 5 2 0 0 0 4 0 0 0 0 0 0), and
 (5 4 5 0 0 0 0 0 0 0 0 1 0).

The code $\tilde{\mathcal{C}}$ has a total of 3^{10} codewords. For a given codeword $\mathbf{u} \in \tilde{\mathcal{C}}$, the value in each coordinate can be mapped to two different choices in the corresponding codeword in $\mathcal{C}$: 0 maps to 0 and 3, 1 maps to 1 and 4, and 2 maps to 2 and 5. There are therefore $3^{10} \times 2^{13}$ codewords in $\mathcal{C}$. It is instructive to compare this cardinality to that of a naïve scheme that only encodes into levels 0 and 3; the latter has cardinality only 2^{13} .

The previous example also illustrates how well we can do cardinality-wise with a proposed construction. The cardinality sandwich bound in [30] tells us that the cardinality of code $\mathcal{C}$ lies between $\lfloor \frac{q}{q} \rfloor^n \cdot |\tilde{\mathcal{C}}|$ and $\lceil \frac{q}{q} \rceil^n \cdot |\tilde{\mathcal{C}}|$. For our example, this inequality becomes equality and tells us that code $\mathcal{C}$ has exactly $2^{13} \times 3^{10}$ codewords, precisely as we have already calculated directly.

As shown in [30], significant savings in programming latency can be achieved using this coding approach; for example the savings are already around 20% with $\ell = 1$, and this fraction increases with the increase in the parameter ℓ . The presented coding idea, that allows for intentional write errors, can therefore offset the inferior write latency of flash-based drives relative to that of conventional hard disk drives, thus bringing flash even closer to broader deployment in hard disk-centric storage applications.

It is possible to symmetrize the coding problem presented here by allowing for the error wrap-around. In this case, we would modify Definition 6 to

Definition 7.

$$d_\ell(\mathbf{x}, \mathbf{z}) = \begin{cases} n + 1 & \text{if} \\ \max_{1 \leq i \leq n} \min\{(x_i - z_i \bmod q), (z_i - x_i \bmod q)\} > \ell, \\ \max\{N(\mathbf{x}, \mathbf{z}), N(\mathbf{z}, \mathbf{x})\} & \text{otherwise.} \end{cases}$$

It is interesting to investigate alternative coding approaches for this symmetrized model; it is expected that tools from (symmetric) group theory could be used in corresponding code constructions.

3.1.3 Additional asymmetric coding research

In this subsection, we briefly discuss related research focused on correcting asymmetric and multiple level errors. The concept underlying codes correcting multiple level asymmetric errors (that is, errors that can be divided into blocks, where each erroneous block satisfies some particular error pattern) date back to as far as fifty years ago. Such classes of codes include concatenated codes and tensor product codes. Concatenated codes, studied by Forney [56] in his seminal dissertation, combine an inner code with an outer code by replacing each symbol in the outer code with a corresponding codeword in the inner code. Tensor product codes, initially studied by Wolf [147], operate in a different manner, by replacing each entry in the parity check matrix of an outer code with a (scalar multiplied by) the parity check matrix of the outer code (this is in fact the tensor product of the two matrices.) Tensor product codes are of course used as a building block in Construction 2. The main similarity between the two types of codes are that both produce a code of length $n_1 n_2$ if the lengths of the constituent codes are n_1 and n_2 . Both techniques produce a minimum distance that is (at least) $d_1 d_2$, where d_1, d_2 are the minimum distances of the constituent codes. However, the resulting dimensions are different with the two techniques. If the constituent codes for the tensor product construction have dimensions k_1 and k_2 , respectively, the dimension of the tensor product code is $n_1 n_2 - (n_1 - k_1)(n_2 - k_2)$. However, the dimension for concatenated codes is $k_1 k_2$.

Both types of approaches (along with schemes that generalize them) can produce codes with various properties. The resultant codes are particularly good at handling, for example, burst errors, where interleaving helps spread the errors into single errors among larger blocks. The multiple level idea then comes into play: the errors in each block are corrected using one of the two codes, while remaining errors or those errors introduced by miscorrections are corrected by the other code. In addition to burst errors, tensor product based codes were also previously considered for error correction over partial response channels, frequently used to model magnetic recording operations, as in e.g., hard disk drives [8; 32]. Concatenated codes have also been used in the

context of coding for magnetic data storage devices. For example, integrated interleaving schemes [69], using a generalization of concatenated codes similar to that of [16], allow for a final construction with distance parameter tailored to the specific error statistics of a particular data storage channel. This scheme is the culmination of the work initially performed for magnetic storage systems in [113] and [3].

There are also many other approaches tackling the types of asymmetric, specific errors we have described. In [60], the authors divide symbol errors into common and uncommon errors and provide constructions of *focused codes* that correct more of the common errors compared to uncommon errors. The focused code constructions mix three linear block codes. Such codes can efficiently correct more exotic errors, such as, for example, odd-weight errors and double-adjacent errors.

Although many of the types of problems we have mentioned have a similar flavor, there is no single overarching framework that encompasses all of them. Similarly, there is no immediate basis for comparison between these schemes, other than a direct comparison to standard error correcting code constructions. Therefore, each problem is typically tackled individually, and the most suitable approach is found. Although a concatenated-coding based solution could conceivably be found, for the case of correcting graded bit errors, code constructions based on tensor product operations are quite useful, as the approach used for Construction 2 is perfectly suitable for correcting the types of errors seen to occur in TLC flash channels.

Finally, we comment on additional recent works that have explored variants of channel models with asymmetric/symmetric errors of limited magnitude and the associated code constructions. Work in [44] designed optimal systematic codes that can correct asymmetric errors of prescribed magnitude in all coordinates, and the work in [157] studied codes for a more restrictive model of graded asymmetric errors (thus intersecting the setups presented in Sections 3.1.1 and 3.1.2). Limited-magnitude error correcting codes recovering from a single error were explored in [90]. The channel model (and codes for it) studied in [160] is akin to the one discussed in Section 3.1.2 with the difference that the

error magnitude is allowed to be different in either direction of the errors. A related problem of constructing error detecting codes of limited magnitude was recently studied in [45]. Constructions of perfect codes and the non-existence of perfect codes for certain parameters for the asymmetric, limited-magnitude error channel were presented in [21].

3.2 Graph-based methods

In the previous section we presented several recent ECC techniques for NVMs that relied on tools from algebraic coding theory. These codes enjoy firm error correction capability of the exact type that is needed in modern NVMs. The accompanying decoding algorithms used bounded-distance decoding, and are based on taking a hard decision at the input to the decoder. On the other hand, graph-based codes can offer superior performance in situations where the decoder can take advantage of so-called soft information from the channel: using quantized channel outputs, the decoder operates on the likelihoods of the received bits.

Graph-based codes have already been deployed with great success in many data storage and communications systems. They are only recently starting to be considered for use in flash memories. In this chapter, we explain how LDPC codes, as the prime exemplars of graph-based codes, can be used in flash channels, when more than one read is allowed. The key point is that a code should be carefully designed for the induced quantized channel and that this quantized channel itself should be chosen to maximize certain design parameters; this choice will in fact come from the choice made in the placement of voltage reference thresholds (which we refer to as quantization thresholds, to avoid confusion with other meanings of the term threshold.)

While on average, LDPC codes offer excellent performance, they do not enjoy the guaranteed error correction capability of their algebraic cousins; this issue becomes particularly important in the region of low error rates where the sudden performance degradation results in the well-known error floor [120]. Given the intricacies of the flash channel and low-error rate deficiencies of LDPC codes (in the canonical models),

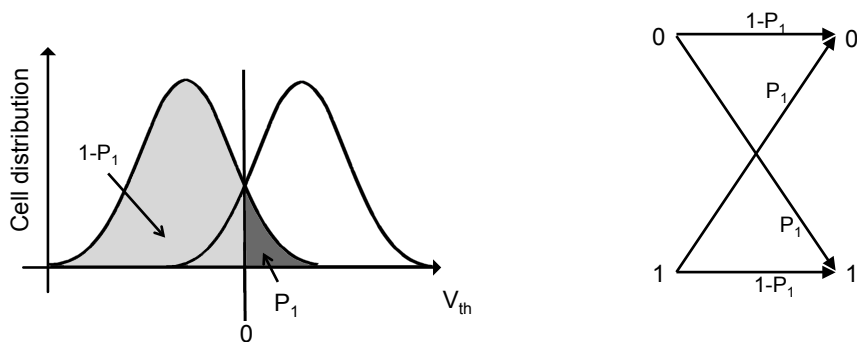


Figure 3.3: SLC threshold distributions with one read (left) and the induced BSC (right).

the design of LDPC codes for modern flash applications is currently a very active area of investigation among leading flash manufacturers.

As previously described, the read operation in flash uses the floating gate voltage to determine the amount of charge stored in the cell. Recall that this quantization threshold voltage varies from cell to cell due to a number of reasons, including read/write disturb from other cells, charge leakage, and write overcharge. The overall effect is captured through a distribution, which in practice may have a fairly complicated form. In order to extract useful information about these distributions that can then be used to form likelihoods, multiple reads are typically performed. How to place the quantization thresholds so that these expensive reads offer valuable information has thus become a key problem.

One recent, intuitive yet effective approach for threshold computation is based on the maximization of the mutual information (MMI) of the induced quantized channel [143]. Suppose that the threshold distributions in an SLC cell are distributed as shown in Figure 3.3, left panel. In practice of course the two distributions will not have the same parameters (see for example [28; 29]), but we deliberately use the simplified model to illustrate the MMI method presented below.

If only one read is available, by symmetry, the threshold will clearly be placed at the intersection point. The induced quantized channel is a BSC with parameter p_1 , where p_1 denotes the area under the distribution tail, see left and right panels in Figure 3.3.

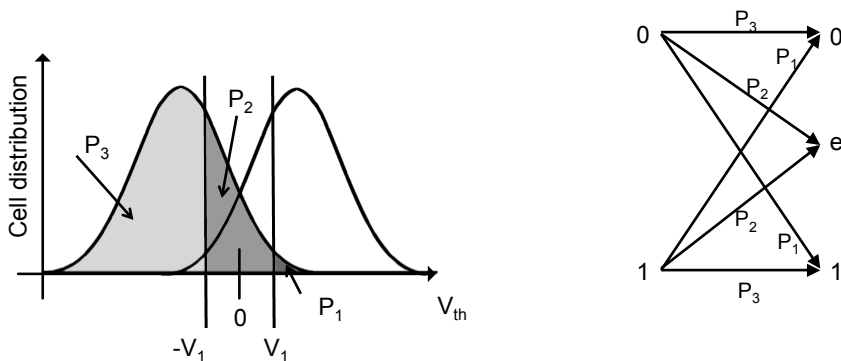


Figure 3.4: SLC threshold distributions with two reads (left) and the induced quantized channel (right).

If two reads are available, we will again by symmetry place the thresholds at v_1 and $-v_1$, see Figure 3.4, left panel. The region in between the two thresholds has very poor reliability; we may therefore mark the output associated with this region as an “erasure”. The induced quantized channel is now as shown in Figure 3.4, right panel. The choice of v_1 and $-v_1$ determines transition probabilities in the channel, marked in the figure with p_1 , p_2 , and p_3 . The goal is then to choose v_1 and $-v_1$ so that the induced quantized channel maximizes mutual information $I(X;Y) = H(Y) - H(Y|X)$, where X is the input and Y is the output of the channel, and H is the entropy function. It is clear from Figure 3.4 that for the uniform input, $I(X;Y) = H((p_1 + p_3)/2, p_2, (p_1 + p_3)/2) - H(p_1, p_2, p_3)$.

Maximizing $I(X;Y)$ with respect to v_1 can be done analytically or numerically, depending on what is known about the underlying quantization threshold distributions. For example if we assume that both distributions are Gaussian with means -1 (for the lower level) and 1 (for the upper level) and variances σ^2 , along with a small v_1 , I is a quasi-concave function of v_1 at the fixed value of the signal-to-noise ratio SNR, $\text{SNR} = (10 \log_{10} 1/\sigma^2)$. Therefore dI/dv_1 can be computed analytically. The plot showing mutual information at $\text{SNR} = 4$ dB is shown in Figure 3.5. The maximum value of I (with respect to v_1) is also specified.