UC San Diego

UC San Diego Electronic Theses and Dissertations

Title

Characterization of TLC Flash Memory

Permalink

https://escholarship.org/uc/item/0341c4wt

Author

Blair, Zachary Taylor

Publication Date

2017

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA, SAN DIEGO

Characterization of TLC Flash Memory

A thesis submitted in partial satisfaction of the
requirements for the degree
Master of Science

in

Computer Science (Computer Engineering)

by

Zachary Taylor Blair

Committee in charge:

Paul H. Siegel, Chair
Leonard E. Porter
Steven James Swanson
Michael Bedford Taylor

2017

Copyright

Zachary Taylor Blair, 2017

All rights reserved.

The thesis of Zachary Taylor Blair is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

Chair

University of California, San Diego

2017

DEDICATION

To Arthur, whose conception has been delayed

for pursuit of higher education.

EPIGRAPH

The laws of probability,

so true in general, so fallacious in particular.

Edward Gibbon

TABLE OF CONTENTS

Signature Page . . . iii

Dedication . . . iv

Epigraph . . . v

Table of Contents . . . vi

List of Figures . . . viii

List of Tables . . . x

Abstract of the Thesis . . . xi

Chapter 1 Introduction . . . 1

Chapter 2 Review of Flash Memory Research . . . 4
2.1 NAND Flash Errors . . . 4
2.2 Endurance . . . 5
2.2.1 SLC Endurance Codes . . . 7
2.2.2 MLC Endurance Codes . . . 7
2.3 Error Mitigation . . . 9
2.3.1 ICI-Mitigating Codes . . . 9
2.3.2 Retention Failure Mitigation . . . 12
2.4 Extending to TLC . . . 12

Chapter 3 Test Platform Architecture . . . 14
3.1 Overview . . . 14
3.2 Architecture . . . 15
3.3 Memory Controller . . . 19
3.4 Software API . . . 20
3.5 Measurement Method . . . 23
3.5.1 SBR Measurements . . . 24
3.5.2 Post-processing and data reporting . . . 28

Chapter 4 Experiments and Results . . . 30
4.1 Read Disturb . . . 30
4.1.1 Experiment . . . 31
4.1.2 Results . . . 32
4.1.3 Discussion . . . 40
4.2 Endurance . . . 43
4.2.1 Experiment . . . 43

4.2.2 Results 44
4.2.3 Discussion 53
4.3 Inter-Cell Interference (ICI) 57
4.3.1 Experiment 57
4.3.2 Results 58
4.3.3 Discussion 60
4.4 Cell Location 62
4.4.1 Wordline Location Dependence 62
4.4.2 Bitline Location Dependence 63

Chapter 5 Conclusions 66
5.1 From MLC Coding to TLC 66
5.2 Relationship Of Application and Coding 69
5.3 Relationship of Disturbs and Coding 70

Chapter 6 Final Notes 72

Bibliography 73

LIST OF FIGURES

Figure 2.1: Example MLC Flash write voltage distribution. 6

Figure 2.2: Example MLC Program Sequence. 8

Figure 2.3: Heat map showing percent of total cell errors for each VWENS pattern. 10

Figure 2.4: Percent of total errors accounted for by various aggressor patterns in MLC for all samples from 0 to 10k P/E cycles. 11

Figure 3.1: Test Platform Architecture 17

Figure 3.2: Top-level (command) state machine for memory controller. . . . 20

Figure 3.3: State machine for micro-operations within each command. 21

Figure 3.4: Sample soft bit data for cells written to B (Level 2). 26

Figure 3.5: Soft bit probability distribution at the beginning of a block's life. 27

Figure 4.1: Marginal error rate after soft bit reads. Each iteration consists of 1024 page reads, totaling 512k reads. 33

Figure 4.2: Shift in soft bit location of cells after 512k reads at end of block lifetime. "Aggregate Out of Range" represents frequency of cells shifted by greater than or equal to the amount shown. 34

Figure 4.3: Marginal error rate after soft bit reads to other pages in the same block. Each iteration consists of 1024 page reads, totaling 512k reads. 36

Figure 4.4: Error rate of cells written to each level after bitline read disturb, measured at end of life. 36

Figure 4.5: Shift in soft bit location of cells after 512k reads on different wordline at end of block lifetime. 37

Figure 4.6: Error rate of page N-1 after soft bit reads to page N, relative to error rate of other pages. 38

Figure 4.7: Marginal error multiplier by relative page location after bitline disturb. 38

Figure 4.8: Error rate as it increases during a sequential block read after inducing wear. 39

Figure 4.9: Error rate as it increases during a sequential block read after 10k P/E cycles. 40

Figure 4.10: Cell error rate of 8 blocks while P/E cycling from BOL to 12,000 P/E cycles. 45

Figure 4.11: Cell error rate for each program level. 46

Figure 4.12: ER and average cell error rate plotted against erase time. 47

Figure 4.13: Percent of total errors accounted for by each program level. . . . 48

Figure 4.14: Frequency of overprogramming and underprogramming failures during P/E cycling. 49

Figure 4.15: Frequency of overprogramming and underprogramming failures for each level after 10k P/E cycles. 50

Figure 4.16: Pseudorandom data error probability after inducing wear with data that is dominated by each programming level. 51

Figure 4.17: The encoding of Upper (U), Middle (M), and Lower (L) page bits into programming levels for each cell. 54

Figure 4.18: Percent of total errors accounted for by various aggressor patterns in TLC. 58

Figure 4.19: Percent of errors for each symbol accounted for by various wordline aggressor patterns in TLC, P/E cycles 0 to 12k. 59

Figure 4.20: Percent of errors for each symbol accounted for by various bitline aggressor patterns in TLC, P/E cycles 0 to 12k. 60

Figure 4.21: Error rate of each programming level by wordline index within a block. 63

Figure 4.22: Error rate across a wordline, by groups of 128 bitlines. 64

Figure 4.23: Error rate across a wordline, by bitline index modulo 32. 65

LIST OF TABLES

Table 4.1: Wear induced by each program level, normalized to random data. 53

ABSTRACT OF THE THESIS

Characterization of TLC Flash Memory

by

Zachary Taylor Blair

Master of Science in Computer Science (Computer Engineering)

University of California San Diego, 2017

Professor Paul H. Siegel, Chair

Flash memory manufacturers are increasing storage density by leveraging multi-bit per cell NAND flash technology in the latest process nodes. Error rates and endurance can be significantly altered in each generation, and the manufacturers provide conservative guidelines for these metrics. Deeper understanding of the physical characteristics of flash devices can be used to develop a usage model which improves lifetime and reliability.

We developed a System on Chip (SoC) platform for flash memory testing, including a hardware memory controller and software for pattern generation and data

processing. We have used the platform to characterize properties of a Triple Level Cell (TLC) flash memory device from a single vendor, and found behaviors have significantly changed from previous generations. Among these novel findings, manufacturers have built compensation mechanisms for some worst-case behaviors, such as inter-cell interference (ICI), but other effects become inherently more significant in the latest technologies.

With information about how certain behaviors affect reliability in both the short-term and toward the end of the product lifetime, this work provides speculation on the goals for coding schemes which target error mitigation.

Chapter 1

Introduction

During the formative years of flash memory, the low capacity and high cost were prohibitive despite the significant increase in performance over disk technologies. In recent technology nodes, flash-based devices have greatly improved in storage density and cost, which has made them a feasible replacement for disk drives. However, as density improves, a downward trend in endurance and reliability has drawn criticism [GDS12].

Cell endurance is generally characterized by error rate as a function of program cycles, and so rapid wear-out can restrict a device's usefulness to systems and applications with read-dominated workloads or short product lifetime. With significantly reduced endurance, the scope of applications for which the newest flash technologies are suited is narrowing.

As users of flash technology, we aim to broaden the horizon for future generations through the use of techniques such as wear and error mitigation codes to

mitigate the reliability problem. To develop an effective scheme which improves device lifetime, a system architect needs fine-grained information about error characteristics and effects that can lead to failures. Unfortunately, manufacturers provide only minimal information about errors — sometimes only the worst-case error rate at the end of product lifetime and the number of program cycles in the product's life. The datasheet for the device we analyze in this work provides no error characteristics, but a guideline that 60bit ECC per 1kB is recommended for a "highly reliable system." With little information about error characteristics, we cannot make determinations as to how we should compensate. We wish to know things like the relative frequency of symbol errors, the probability and direction of failures, and how stresses such as repeated reads disturb the programmed data.

In previous work within the CMRR, these fine-grained error data have been obtained for MLC devices from the same vendor and mitigation techniques have been tested [TUS15]. We are able to take these measurements due to a partnership with a memory manufacturer, who has provided us with access to special vendor commands to view soft-bit data. The manufacturer has been able to learn from previous generations and has compensated for them in the newer chips, so worst-case behaviors may have changed and new technologies may have significantly different properties. We repeat previous work, and expand upon it, for TLC flash memory to perform error analysis for a single product.

Benchmarking and reporting unofficial device characteristics is not the intent

of our work. We do not provide specific information such as command codes, page size, or pages per block so as to keep the identify of the manufacturer and product confidential. Instead, we use approximate values (>8kB pages and 80 pages per block) in our results and analysis that are relevant to the implications for coding.

Future work will involve development of coding mechanisms and use models that extend NAND lifetime. For example, we can extend MLC error mitigation or endurance codes that adapt to the current error characteristics, which vary throughout the life of a cell due to wear.

The rest of this thesis is organized as follows: Chapter 2 reviews previous work in studying the reliability of flash memory. Chapter 3 describes our platform and methodology of measurement for the flash devices. Chapter 4 describes metholodogy for each of the effects analyzed, and presents the results of our testing. Chapter 5 contains speculation on how our work can benefit future development of the technologies within and surrounding the flash device.

Chapter 2

Review of Flash Memory Research

Significant research has been conducted at UCSD, in both the Center for Memory and Recording Research and the Non-Volatile Systems Laboratory, in the characterization of and use models for NAND flash memory. Related works include extrapolation of error characteristics to predict the future of Flash memory [GDS12], characterization of errors and effects in SLC and MLC flash [CHMM12], and development of methodology to most effectively use flash for high performance or long lifetime [SA14, LS16]. This work is a direct extension of some of these projects, extended to the latest flash technology (TLC) and process node.

2.1 NAND Flash Errors

Early research found that while the density and cost metrics for NAND flash improved, nearly ever other aspect of the technology has sufferred. Write latency,

energy, program/erase lifetime [GDS12], and retention/reliability[CHMM12] have all decreased with each technology node. Most notably, multi-bit per cell storage reduces lifetime by a decimal order of magnitude for each additional bit. The already-short lifetime of SLC Flash (100k program/erase cycles) has been reduced to 10k and 1k program/erase cycles in MLC and TLC technologies, respectively. Manufacturers provide a minimum ECC correctability requirement for the device and a minimum number of P/E cycles for which each block will remain reliable if the ECC requirement is met. However, the error rates at end of life (EOL) are not published. In previous works [GDS12][TUS16], the bit error rate at EOL for MLC and TLC NAND has been approximately 10E-3.

In order to work around endurance limitations and to extend the lifetime of flash products, we need to collect more specific data about error characteristics over the lifetime of the product. There are various effects that can contribute to a cell error, and determining which are most significant and how they manifest is important to developing methodologies and use models for extended lifetime. These effects include gate wear, inter-cell interference (ICI) or adjacent write disturb, read dusturb, and value retention over time.

2.2 Endurance

A NAND Flash cell write is executed by injecting charge into the gate of a floating gate transistor. Data is read back by applying voltage to the bit line and

holding the control gate open for all word lines except the cells being measured, then reading the voltage of the bit line after charge drains through the measured cell to determine how much charge is held in the floating gate. Figure 2.1 is an example of the write level distribution for an MLC Flash cell. The default erase state (ER) holds no charge, and programming the cell to levels A, B, and C injects charge into the floating gate across the tunnel oxide, which over time causes wear of the gate.

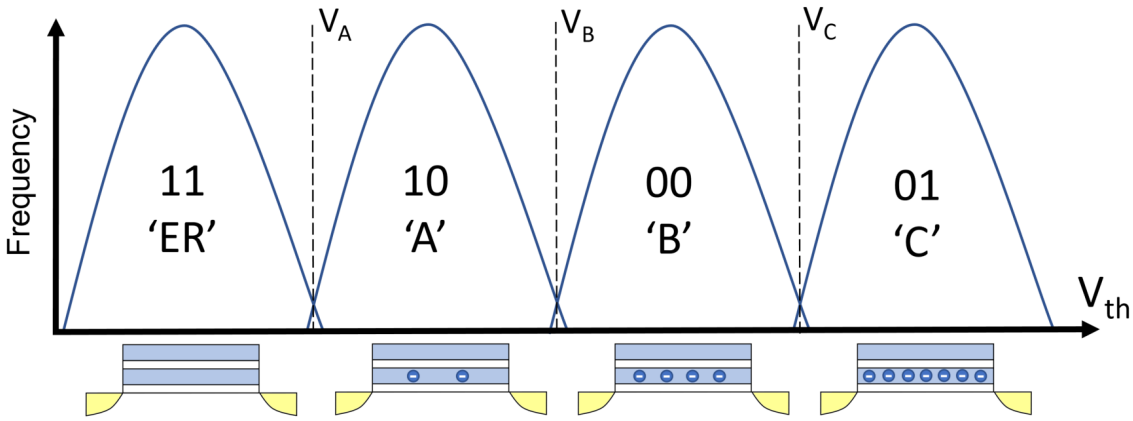

Figure 2.1: Example MLC Flash write voltage distribution.

Erosion of the cell's gate is the primary determinant of chip endurance, and the other effects (ICI, write/read disturb, charge retention, program/read interference) are the avenues by which errors occur. As the gates wear down, these effects become more significant and error rates increase [CHMM12]. Our objective as system architects is twofold: we must determine whether it is possible and valuable to reduce gate wear, and we must determine which error-inducing effects are most impactful and whether they can be mitigated.

2.2.1 SLC Endurance Codes

For structured data such as databases, spreadsheets, and documents, SanDisk demonstrated shaping codes for SLC flash that reduce gate wear by lowering the fraction of programmed cells [SA14]. The objective of these codes is to dynamically identify commonly occurring sequences and form a dictionary that maps high frequency entries to low-cost code words. By reducing the total charge injected into gates during the program cycle, wear is reduced and the block's endurance is improved. The table mapping source data to codewords is built dynamically as the data is written or read, which makes the approach attractive by eliminating buffering to build a dictionary and space to store it. Endurance codes are a comparable alternative to compression for structured data, where total charge is reduced by reducing the number of words instead of average cell level. With a fundamentally different relationship between program data patterns and cell level cost, multi-bit per cell technologies posed a challenge and presented an opportunity for more effective techniques of gate wear mitigation.

2.2.2 MLC Endurance Codes

To apply shaping codes to MLC devices, we must first understand which patterns are most desirable. Previous work has quantified the effects of gate wear and developed a cost model for MLC flash chips[SA14]. In general, the vector of cell level costs $[c_0,\ c_1,\ c_2,\ c_3]$ satisfies $c_0 \leq c_1 \leq c_2 \leq c_3$. Two features of the program

sequence shown in Figure 2.2 impacted the development of MLC codes. First, the lower page has a greater impact on cost than the upper. Second, the cost associated with an upper bit depends on the value of the lower page bit.

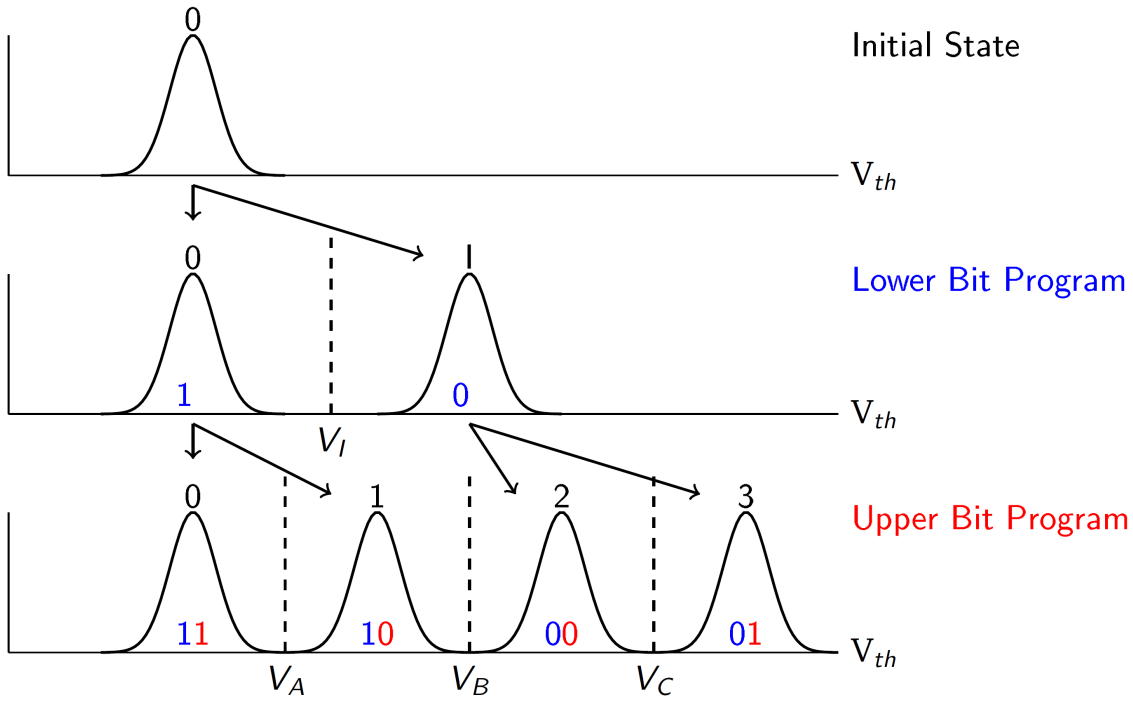

Figure 2.2: Example MLC Program Sequence.

An unbuffered and adaptive code was developed which extends the SanDisk code for MLC flash. The lower page is programmed using the original SanDisk coding technique, but the upper page utilizes a lower-page dependent dictionary to encode the upper page[LS16]. For each sequence, the codeword stored in the lower page is used to determine upper page values. As with the SanDisk code, the upper page extension does not require buffering or dictionary storage. Using this technique, the shaping code was applied to the upper page despite the cost dependence on the lower

page, and results were shown to be comparable to that of the SLC code.

2.3 Error Mitigation

As an alternative to wear-mitigating codes, which improve error rate for future program cycles, error mitigation codes can improve the read error rate for the immediate block. To develop these codes, we must of course understand what causes errors and how they manifest. Some of the effects that induce errors are read disturb, inter-cell interference (ICI), and charge retention failures. Any of these errors that manifest non-uniformly can be coded against, as the worst-case patterns can be avoided. In this work we gather data about read disturbs and ICI, but not retention failures as we do not have an environment in which we can simulate device aging.

2.3.1 ICI-Mitigating Codes

Flash memory blocks are composed of a closely packed rectangular array of cells interconnected through horizontal wordlines and vertical bitlines. Wordlines are used to select a row, or page, and the bitlines are used to program and read individual cells. Due to the tight structure, there are non-negligible parasitic capacitance couplings between the gates of adjacent cells as shown on the left in Figure 2.3. The capacitance between cells leads to program and read interference.

ICI manifests as an increased voltage in victim cells (V) as they are affected by aggressor cells (WENS). Work has been done in the CMRR to perform charac-

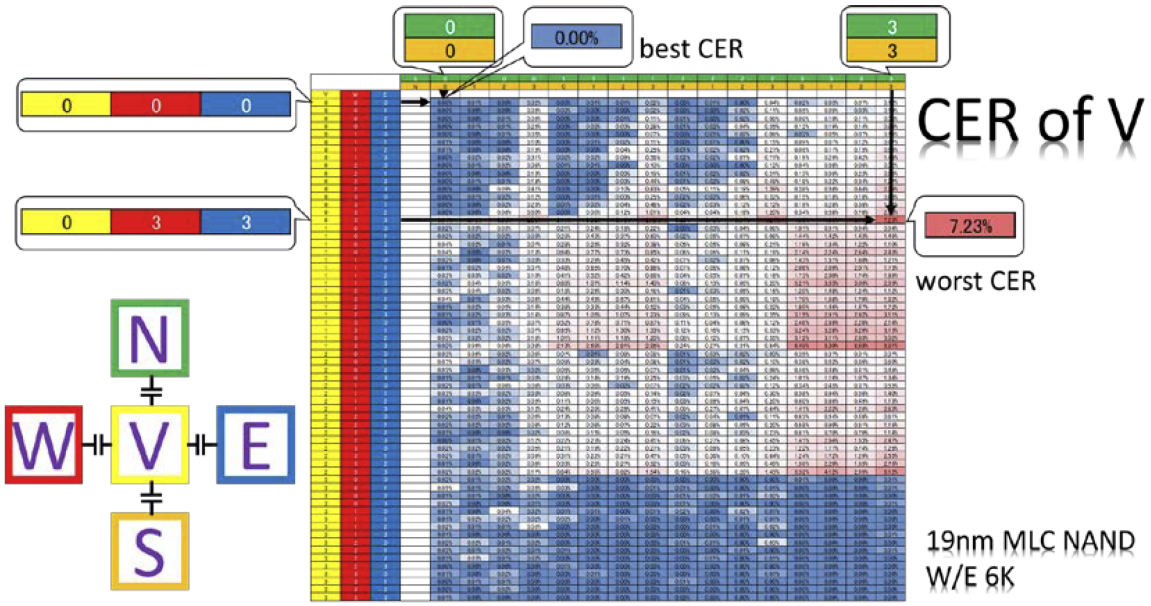

Figure 2.3: Heat map showing percent of total cell errors for each VWENS pattern.

terization of ICI in [TUS15], where it is shown that the error rate of the victim cell depends heavily on the VWENS cross pattern in a 19nm MLC NAND flash device. In particular, the neighbors with a shared bitline (N and S) have the highest impact on V. A graph of the victim cell contributions as a fraction of symbol errors is shown in Figure 2.4, and a heat map of the error rate for all patterns is shown on the right of Figure 2.3. It is clear from this data that there is an asymmetrical disturb on low-valued victim cells by high-valued bitline neighbors, particularly from the southern aggressor, which can be compounded by the northern cell. The shift is always toward a higher voltage, indicating that ICI allows neighboring cells to further open the channel of a victim cell, but not close an open channel.

Considering that the worst-case disturb from ICI is dependent on the pages above and below the victim, several codes have been proposed. ICI-mitigating codes

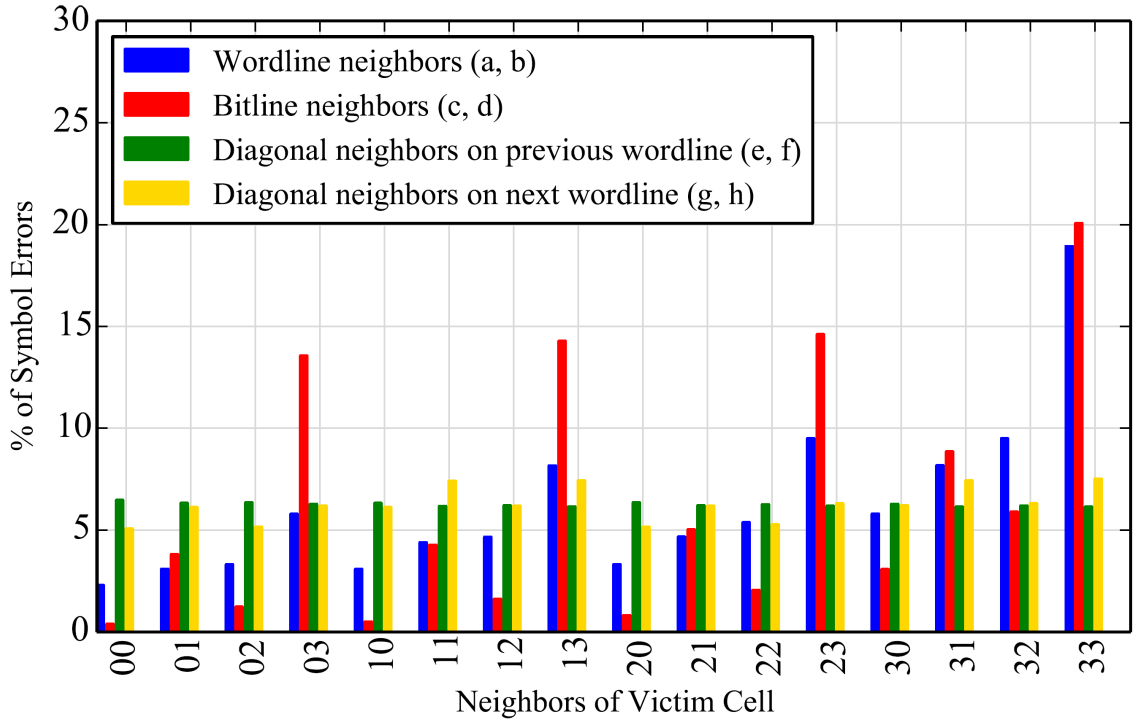

Figure 2.4: Percent of total errors accounted for by various aggressor patterns in MLC for all samples from 0 to 10k P/E cycles.

are considered with varying tradeoffs in granularity of decode operations (i.e. whether the lower and upper page of a wordline can be independently recovered) and performance overhead (decoding latency and readback of dependent pages). A few flavors of page and wordline row-by-row codes are proposed in which WL_i is dependent upon WL_{i-1} and optionally WL_{i-2}, in order to minimize BER based on the cost derived from the data in Figure 2.3. Another code which has no external dependencies simply tries to control the number of V=0 and V=1 occurrences, called a one-rich code[Tor16], has a lower read latency but sacrifices some BER reduction. The findings of these codes have yet to be published, as the efficacy has yet to be conclusively

demonstrated for multiple devices under various stress factors.

2.3.2 Retention Failure Mitigation

In contrast to ICI, retention failures result in the read-back voltage for a cell to be shifted down as charge escapes the floating gate over time. Just as ICI cannot cause errors by raising the voltage of a cell written to the highest level, loss of electrons cannot cause a retention failure in the lowest programming state, as there is no lower state to which it can transition. The solution is thus the opposite of ICI-mitigating codes, and selecting the appropriate code depends on both the use model for the system in which flash is integrated and the relative impact of retention and ICI on error rates. Retention failures have been characterized in [CHMM12], and we can qualitatively extend their results to TLC (without specific costs) for consideration of error mitigation codes. However, since we do not have an environment for simulating aging, we have not reproduced these measurements for TLC error characterization.

2.4 Extending to TLC

The coding objectives for MLC flash were closely tied to the physical behaviors and structures. We would like to determine whether previous work in coding to reduce error rate and prolong device lifetime is applicable to TLC flash in a smaller process node, and how we can adapt this work to match the latest chip characteristics. For example, TLC and QLC flash have an increased frame size for ICI-mitigating

codes that are dependent on the previous two wordlines. With an extra one or two pages per wordline, the frame can include up to 12 pages of readback, which may be impractical. Other techniques become less practical due to changing error characteristics. If a specific type of error becomes significantly more or less prominent, either as an inherent side effect of increased density or by manufacturer intervention, this information will be valuable when developing a programming cost model for codes.

Chapter 3

Test Platform Architecture

Previous work in the CMRR STAR group utilized a Xillinux and Xillybus-based test board on the Zedboard evaluation platform. The existing framework had reliability issues that became problematic for long-running tests, so we took the opportunity to fully redesign the hardware while making modifications to support the new TLC command sequences. In addition to upgrading for TLC chips, the flash memory test board has been redesigned with the intent to improve performance and debugging functionality.

3.1 Overview

Test performance for the MLC board was on the order of 1k P/E cycles per hour, limited by the bandwidth of the Xillybus framework for hardware-software data transfers. With an improved data transfer architecture and bare-metal application to

remove OS overhead, we have eliminated all bottlenecks from the test platform and successfully saturated the flash device's interface. Despite a new, highly redundant program sequence, the average test rate as a function of bytes programmed and erased has increased by 4X.

An intermittent bug existed in either the MLC flash chips or the test board that caused long-running tests to hang and/or return invalid data. The redesigned board has improved debugging functionality, including a watchdog timer, the ability to recover from faults, and recording waveforms from the memory interface using an Integrated Logic Analyzer.

3.2 Architecture

The new test board, upon which we have developed a framework for moving data between the Processing Subsystem (PS) and Programmable Logic (PL), has many benefits over the MLC board. Data transfers previously utilized the proprietary Xillybus IP core and Xillinux OS. These are useful for higher-level applications which require the I/O features provided by Linux and some simple communication with custom logic on the FPGA, but the performance is very low and customization is limited. Developing custom hardware has allowed us to both saturate the flash memory interface and improve debuggability.

We chose not to integrate the new hardware with an OS because there are more advantages to using bare-metal software for our application. The benefit of an

operating system is the filesystem and Ethernet drivers, which allow us to store test results on an SD card or network-attached storage. However, since the collected data is relatively small, we can write to a file on a host machine over a serial port without loss of performance. No other features of Linux were utilized in testing, and so the additional abstraction is not necessary. Xilinx provides a software development kit that aids in the development of bare-metal applications, the end result of which is a much simpler interface to the FPGA and significant performance gains. Modifications can also be made to the hardware design, then quickly implemented and tested in software without the long process of rebuilding an operating system with a new board support package.

Figure 3.1 shows the design implemented in hardware, the software/hardware interface, and I/O. An ARM processor controls the system and communicates with the host machine, and the programmable logic is only responsible for interfacing with the flash device. The PS and PL have separate communication channels for commands/status and data movement.

The primary interface between the PS and PL is the GP0 AXI Master, a slow interface by which the ARM core can write and read to memory mapped IP cores in the PL. The CSR receives commands, accepts configuration data, and responds to read requests for the status and error registers. When a command is received, the CSR triggers the Datamover IP to begin reading or writing data and the memory controller begins execution.

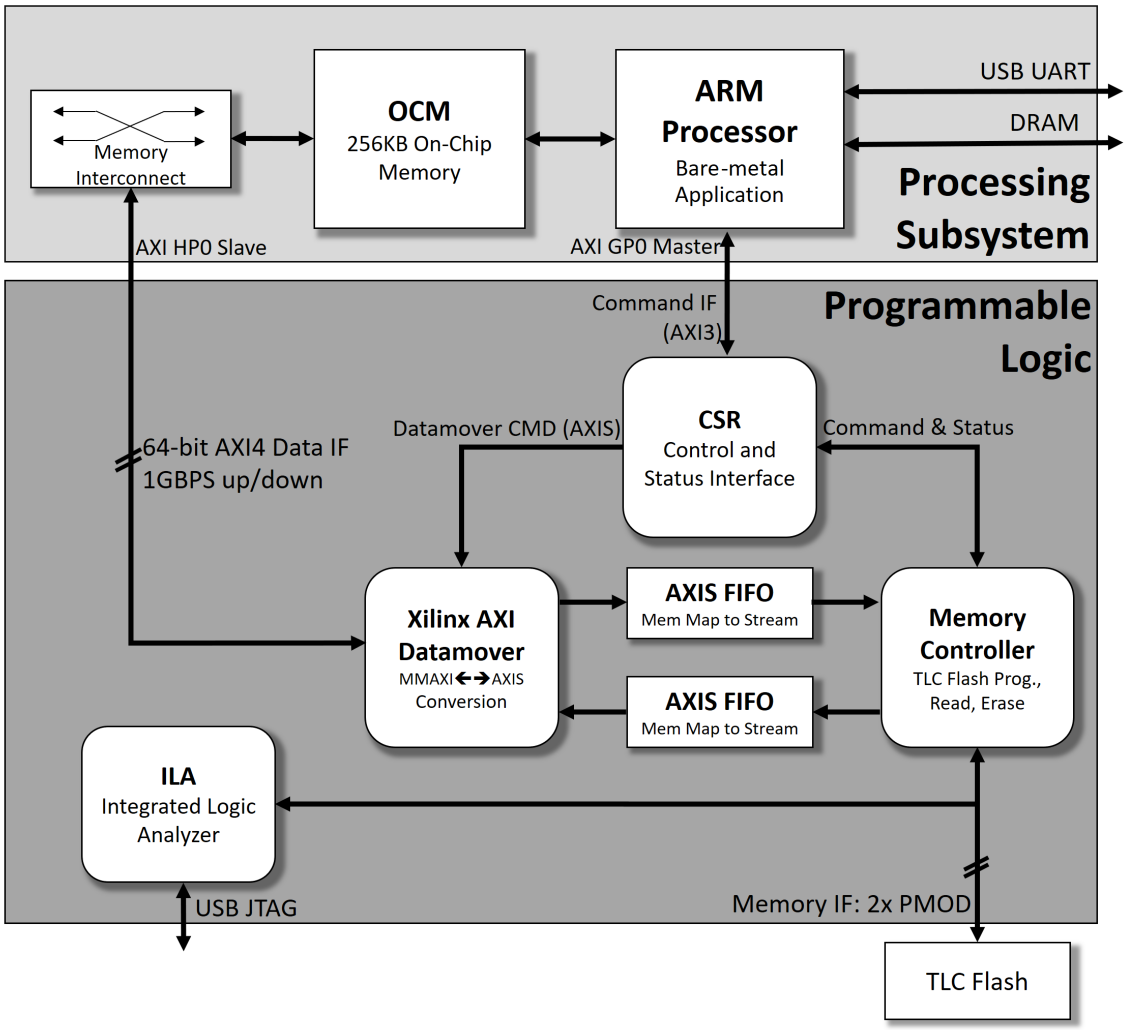

Figure 3.1: Test Platform Architecture

Data takes an indirect path from DRAM to the Flash memory when a page write is issued. Since using the ARM processor to write as a Master over AXI is both slow and utilizes the processor during transfer, we instead use a high power AXI port (HP0) in which the PL is the master. This interface can read from the on-chip memory (OCM) and DRAM at rates up to 1GBPS up and down, and requires no interaction with the processor, which makes it ideal for achieving high performance

and parallelizing the hardware/software.

The Master in the PL that controls data movement between the memory controller and OCM/DRAM is the AXI Datamover, a Xilinx IP that converts memory mapped AXI to AXI4 Streams. The Datamover stream fills a FIFO, which in turn feeds test data to the memory controller. The processor writes a length, source and destination address for test data to CSR address registers, and when a command is issued the CSR programs the Datamover to either fetch data from OCM and write to the FIFO or read from a FIFO and write to the OCM. With this framework in place for moving data, we can abstract the memory interaction from the flash controller such that it has simple data input and output streams that it can read/write depending on the issued command.

Once a command is received by the memory controller, it begins driving the control signals to the flash chip. With the exception of the new command sequences, the memory controller closely resembles that of the MLC test board. Notable exceptions are that the hardware has been designed such that the state machine will be much easier to modify/extend for future updates, and the chip timing parameters can be programmable at runtime to allow for tuning to the optimal performance.

If an error or fault does occur, it is now possible to debug the internal state of the FPGA using the integrated logic analyzer (ILA), which can record and upload waveforms for a selected set of signals to the host machine. However, the intermittent failures observed in prior MLC testing have not been observed with the new

architecture. Monitoring the flash interface with the ILA can still yield useful data about timing characteristics of the flash chip under various stress conditions, which can be utilized to tune performance beyond the worst-case latencies reported in the datasheet.

3.3 Memory Controller

Most of the development effort was focused on creation of a memory controller that was efficient, extensible, and configurable so that it could be tuned to improve bandwidth. The new memory controller is a redesign of the MLC hardware, extended to support new TLC command sequences. It has been designed with a 3-level hierarchical state machine, where interaction has been separated into three levels of abstraction.

The memory controller is triggered to begin execution of a command after the configuration registers are programmed with the source and target address, block and page numbers, and other data that is required for the next command. The commands will execute until completion, then notify the processor that the controller is idle and available again.

At the top level (Figure 3.2) is a command, such as write wordline, read/set feature, and block erase. We recognize that each macro-operation in the datasheet consists of a small set of operations, and so each command is broken up into a series of micro-operations (Figure 3.3), e.g. write address, read/write from/to data

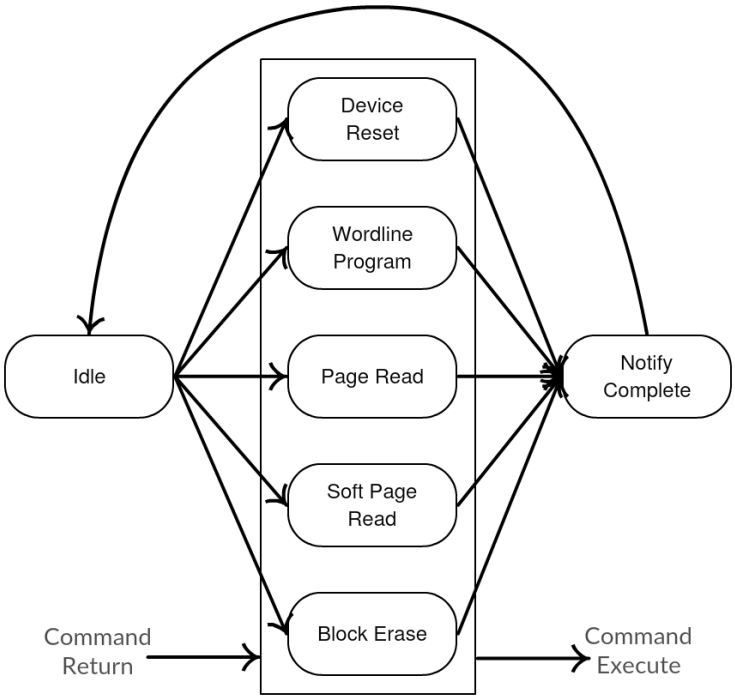

Figure 3.2: Top-level (command) state machine for memory controller.

stream, and write command. At the lowest level, each micro-operation consists of the individual control signals that must be toggled and their timing characteristics. The timing characterstics are configurable, so the device can be tuned for optimal performance.

3.4 Software API

We have developed a software API to abstract the low-level hardware interaction, as well as create some helper functions for test data generation and post-processing. The hardware interface consists of a set of functions to initiate flash commands in the memory controller, which run to completion without further inter-

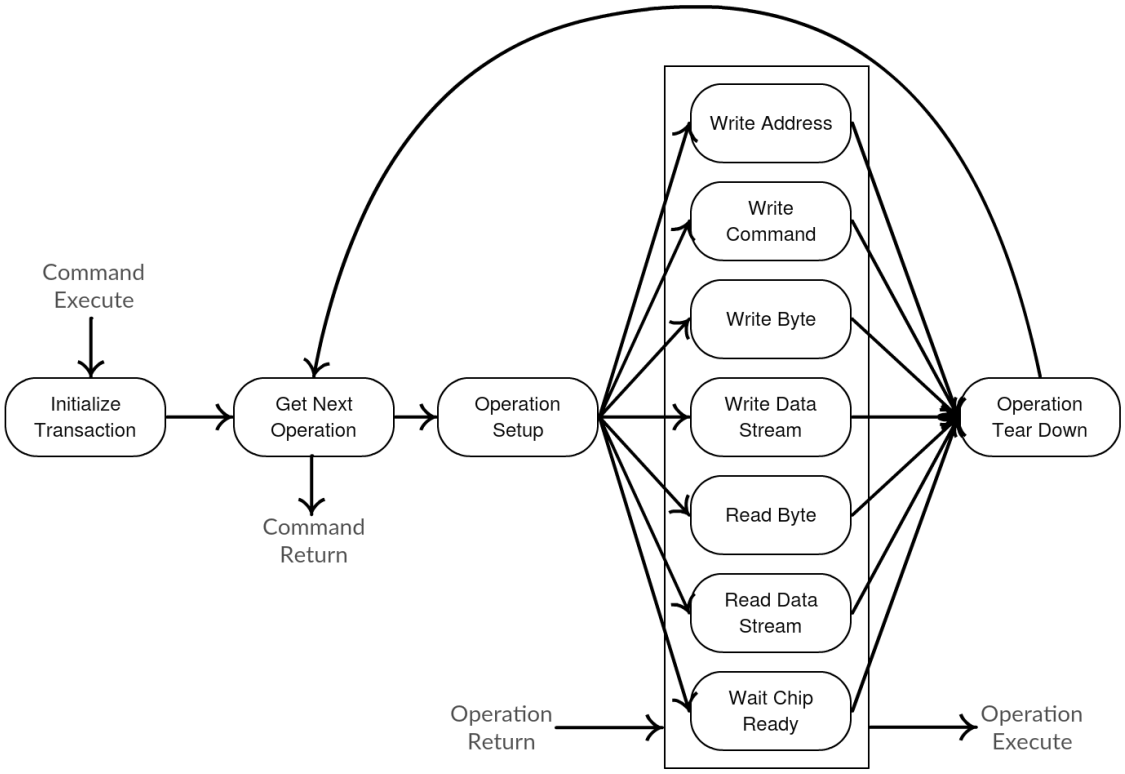

Figure 3.3: State machine for micro-operations within each command.

action from the software. These functions are listed:

- **Print Config/Status Registers:** Reads the complete configuration and status register file and reports their contents. Configuration includes watchdog timer length, test data length, source address, and destination address. Status includes the memory controller error flags, currently executing command and chip status from the previous command.

- **Wait on Chip Idle:** Blocks until the currently executing command completes.

- **Wordline Program:** Writes a single wordline, which is the minimum amount

of data that can be programmed to completion. A wordline for the device under test consists of 6 pages: the lower (L), middle (M), and upper (U) page on both the left plane (LP) and right plane (RP). Each wordline must be written three times as required by the TLC device's 3-stage program sequence.

- **Block Erase:** Erases a single block, which consist of approximately 80 wordlines.
- **Page Read:** Reads the hard bit data for a single page (one third of a wordline).
- **Soft Page Read:** Performs a sweep across a range of threshold voltages, yielding fine-grained information about the voltage written to each cell. See section 3.5 for further detail.
- **Read/Set Feature:** Read or program chip features. The chip ID, serial number, block lifetime, recommended ECC, page/block sizes, and various other features can be read. Chip configuration registers such as read/write thresholds can be programmed.

Once the memory controller has been triggered, the ARM core is available to perform test data pattern generation and post-processing of results in parallel. If we do not serialize data generation, flash interfacing, and post-processing, we can keep the memory interface saturated and perform experiments at the maximum rate allowed by the device under test. The features provided for post-processing and data generation are also listed:

- **Format Address:** Format the 5-byte address from block number, plane, and page.
- **Generate Random Data:** Generate pseudorandom test data for P/E cycling.
- **Generate Patterned Data:** Generates a block of contiguous data which is structured such that the input matrix of cell levels will be replicated across the block.
- **Process Soft-bit Read:** After an SBR, there is a duplicate of the page data for each shift threshold. These pages must be scanned to find the exact location where the threshold crosses the cell's voltage, which is the soft cell value.
- **Generate Histograms:** For hard and soft bit data, we can generate a histogram of the read back information based on various conditions. Some of these conditions include written value (for hard bit errors), soft cell values (for distribution graphs, P/E cycles, and the local VWENS pattern).

With the API we have developed, the NRE time required to develop tests to analyze a new effect or impact of a wear or error mitigation scheme is minimal.

3.5 Measurement Method

We simulate typical usage of flash devices by P/E cycling with pseudorandom data. At regular intervals, we read back data before erasing the block, process the

patterns, and report histograms of the data to a host machine. The typical reporting interval is 100 P/E cycles and we test for over 10k cycles, which is 10x the lifetime of a TLC block.

Much of the information we would like to gather about errors can be gleaned from hard bit data, or the eight "bins" returned from normal page reads. With these data, we can determine the rate at which the read back values for a cell are in a different bin than the value written. However, it only provides a macro-level perspective on the impact of various disturbs. For a more fine-grained perspective on the distribution of cell values, the manufacturer provides the capability of reading "soft" bit data, which have about 64x higher granularity than hard bit reads. That is to say we can read approximately 512 "bins" instead of 8. Soft bit reads provide us the ability to observe how various effects such as wear, read disturbs, and ICI shift the distribution of cell voltages.

3.5.1 SBR Measurements

Hard bit reads determine the bin a cell's voltage falls into by checking whether the voltage is greater than or less than each threshold, of which there are 7 to divide the 8 bins in TLC flash. Soft bit data can be gathered by shifting the thresholds by increments that are approximately 1/64th the distance between each hard bit threshold and reading the data again. The method that this manufacturer provides involves determining whether a cell is within a specific range of voltages. We first

shift a threshold to the low end of the range, then to the high end of the range and perform another page read. This yields two pages of data which indicate whether each cell's voltage is above or below the two selected thresholds. These pages are XORed, and the positive bits are those which reside within the specified range, above the lower and below the high threshold.

The hardware used for a SBR is highly sensitive, but in reading minute differences in voltage for flash cells that are prone to disturbs, we cannot accurately measure shift values of N and N+/-1 to check whether a cell falls into each small bin. Instead, we hold the low threshold constant at the maximum negative shift while performing a sweep across the full range of shift values for the upper threshold. For example, to measure cells adjacent to V_B, or those written with symbols A or B, we use the following procedure:

- Set the bottom of the range V_{min} to V_B-128, the maximum negative shift.
- Set the top of the range V_{max} to the bottom of the measured range, e.g. V_B-100.
- Read two pages, one for each V_{min} and V_{max}.
- XOR the pages, and read the data back resultant data.
- Repeat the measurements, incrementing V_{max} until the high end of the measured range is reached, e.g. V_B+100.

For each cell, this yields an array of bits indicating whether its value lies within [-128,V_{max}] for V_{max} from V_B-100 to V_B+100. Some sample data for cells with

values between V_B and V_C is shown in Figure 3.4.

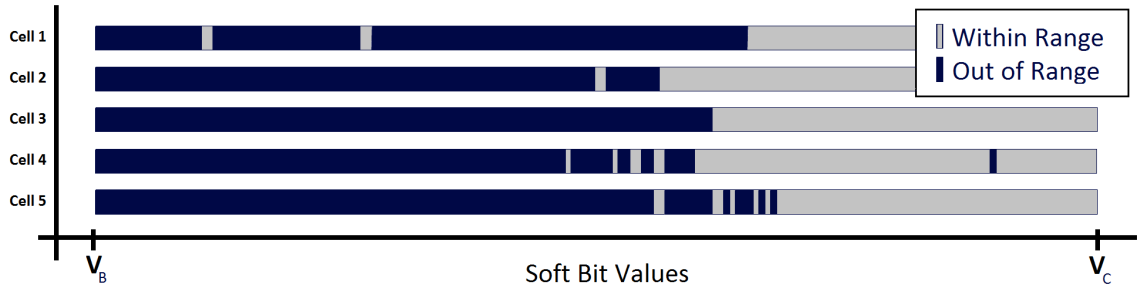

Figure 3.4: Sample soft bit data for cells written to B (Level 2).

Looking at cell 3, we can see the low end of the measurements which are "within range" begins at $V_{max}=V_B+40$, which means the cell's voltage lies within that bin. Ideally, all cells would have only one such transition and it would be clear into which bin they fall. However, there are frequently outliers and transitions that are not smooth. We applied a few methods to better approximate the voltage written to each cell, including a smoothing filter to remove outliers and taking a weighted average of the transitions based on the length of continuous segments before and after the transition. The efficacy of these methods was limited because there were always somewhat common patterns which would yield obviously incorrect results.

We cannot correlate the hard or soft bit thresholds to absolute voltages, as this information is proprietary and not provided in the technical specification. However, we do know that each soft threshold is evenly spaced and we can use this knowledge to plot the frequency distribution without specific voltage information. An example of the soft-bit distributions is shown in Figure 3.5, which includes the full curve for

cells written with symbols A-G, but only a small portion of ER because the fully erased state lies below the measurable range. Each point represents the probability that a cell's value resides within a small bin, or 1/64th of the space between hard bit thresholds. The distribution for each symbol appears to be gaussian-like with the exception of some asymmetry in the highest and lowest levels.

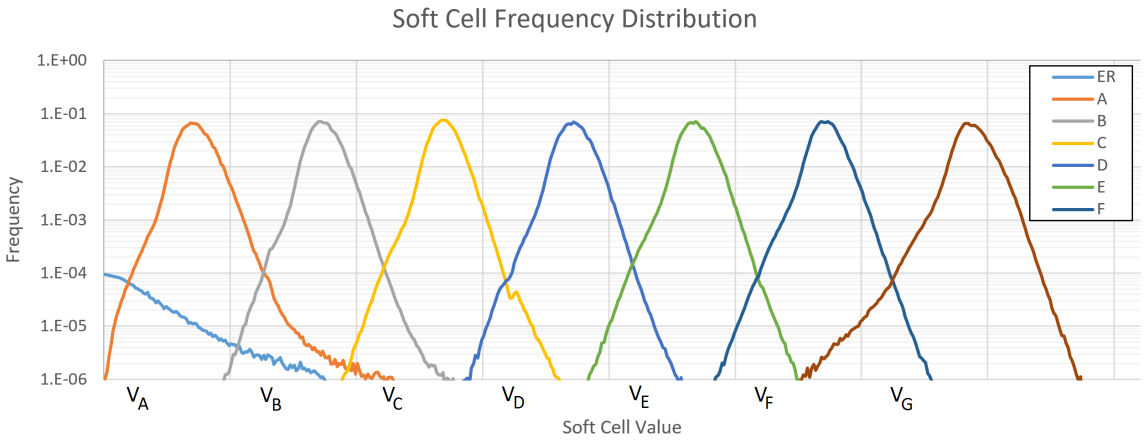

Figure 3.5: Soft bit probability distribution at the beginning of a block's life.

The soft and hard bit data also appear to vary in that cells with a soft level near a threshold can have hard bits indicating they are on the opposite side. This behavior is observed either due to disturb induced during measurement taking, or simply imprecision in the physical mechanism for measurement. Due to the imprecision and lack of absolute voltage information, our objective is not to perfectly measure each cell, but to process the soft bit arrays with consistency such that we can create a probabilistic curve for each written symbol and compare the relative shape and position of write distributions under various stresses. We found that the most consistent method simply sums the number of samples that are out of range

and adds this offset to the low threshold.

3.5.2 Post-processing and data reporting

The soft bit data is many hundreds of times larger than the source data, and we take measurements for multiple blocks consisting of thousands of page reads, so storing the complete test results in their raw form is not feasible. We process the data into a set of histograms for certain behaviors such as error direction, error frequency, and soft levels. These histograms are sorted by number of P/E cycles, the local VWENS write pattern, and the hard read value of the victim celll.

There are two soft measurement behaviors that we record separately from the typical samples. The first is an expected behavior – the ideal erase state (ER or Level 0) is below the minimum measurable voltage when shifting from the lowest threshold V_A. We record these measurements as out of bounds (OOB) in a separate histogram, and they do not appear in distribution graphs. Sometimes an OOB error also occurs for cells read as something other than ER, and we record these separately. The second behavior occurs when we identify the soft bit samples as unreliable. There is always a range in which the value of a soft bit measurement is questionable, but some data appears to be completely random. When more than one quarter of adjacent sample pairs differ, we record the cell read as invalid.

Since we do not have an OS or file system on the embedded platform, we write the results to a host machine after processing hard and soft bit read data. The

host sorts the data and stores it in a database that can be accessed by visualization software for plotting and comparing the effect of various patterns and disturbs. We analyze this data to identify patterns and anomalies, which we explore in depth in the next sections. The patterns we observe are used to describe how errors manifest and quantify their impact, which enables us to apply coding mechanisms to reduce errors through mitigation for worst-case patterns and disturbs.

Chapter 4

Experiments and Results

In this chapter, we describe the experimental method for testing various aspects of read disturb, gate wear, and inter-cell interference. For each effect, we provide quantitative analysis of how the effect manifests over the lifetime of the product. We discuss how the reliability of a device varies over time and how this impacts coding objectives. As each effect can be coded against differently, and decreasing errors through one effect may increase them through another, we also discuss how the properties relate and which are most impactful.

4.1 Read Disturb

In order to isolate the results for later experiments, we first characterize read disturb. We read entire blocks during wear and ICI analysis, so understanding the impact of a soft bit read first is important to observing the marginal difference of

other effects and how they interact.

4.1.1 Experiment

When we read a page in NAND flash, the current allowed through bitlines is dependent upon the charge held in the floating gates for the selected wordline. This is accomplished by raising the control gates for all other wordlines while leaving the control for the selected page low. There is commonly an observable effect on the unselected cells due to raising the control gate and allowing current to flow through, and we quantify this effect. The effect of reading a page is typically higher on adjacent cells than on the selected wordline, and we show that this is also the case for this device. Both reading a wordline (WL disturb) and reading cells on a common bitline (BL disturb) are analyzed.

Wordline Disturb

To demonstrate that reading a wordline does not disturb its cells, we first program the cell with pseudorandom data. An initial soft page read is performed and the soft level at each index in the page is stored. For one soft page read, we read the full measurable range from each threshold, which is the range [-128,127] from two sets of thresholds (the lower and upper pages). With each range requiring two reads, one for the lower and one for the upper end of the range, the total number of page reads is 1024 for each soft bit data set. This is an unavoidable minimum unit of disturb that we must apply to make a measurement. This number of reads is

relatively small compared to the typical number of reads required to disturb a cell to the point of failing, and should cause a negligible shift in cells. We repeat the soft bit read 512 times for a total of 512K reads and analyze the results at several intervals. This test is repeated after inducing wear through P/E cycling. Measurements are taken at the beginning of the chip's lifetime, the manufacturer's specified end of life, and 10x the end of life.

Bitline Disturb

We quantify the effect of bitline read disturbs in two ways. First, we show the relative error rate of pages in a block as we perform soft bit reads of those pages sequentially. This results in a total of 1024 page reads per page as they are read, and so the Nth page will have N*1k reads on the bitline prior to being read back. This test is important to quantitatively establish the read disturbs incurred when reading back blocks for ICI and wear analysis.

The second experiment is performed similarly to the Wordline Read Disturb test. At each interval where we would collect data about the wordline being stressed with repeated reads, we instead read the other pages in the block before and after the read stress is applied and measure their marginal error rates.

4.1.2 Results

For each experiment, we consider the error rate at the beginning, end, and well beyond the end of life, with samples taken at various intervals of read disturb.

Wordline Disturb

Based on historical experiments in SLC and MLC, we expect that repeated reads of a single wordline will have little impact on the contents of the page being read. Figure 4.1 shows the marginal cell error rate after repeated soft bit read sweeps at exponential intervals. Each iteration repeats the soft bit measurement technique outlined in Section 3.5.1, so the cell error rate is recorded from 2^9-2^{18} page reads.

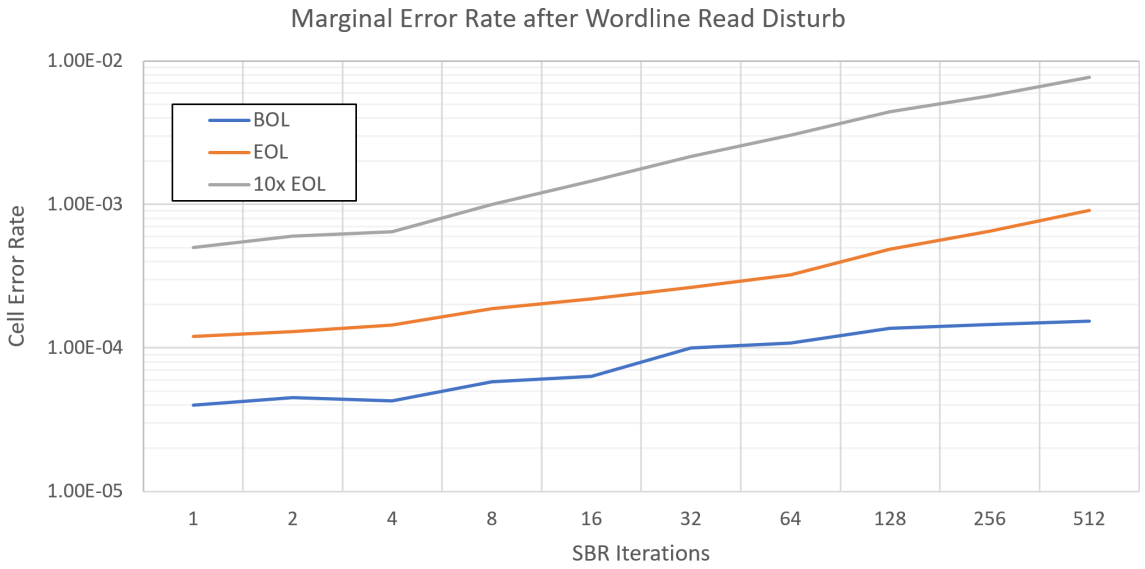

Figure 4.1: Marginal error rate after soft bit reads. Each iteration consists of 1024 page reads, totaling 512k reads.

This data shows that there is a measurable wordline read disturb, and that it is proportional to the write error rate incurred by wear. The hard bit write error rates prior to any disturb are 1.1E-3, 2.3E-3, and 1.8E-2 for the BOL, EOL, and 10xEOL measurements, respectively. Comparatively, the wordline read disturb is negligible early in the device lifetime with a marginal error rate of less than 15%

at 512k page reads. The marginal error rate is 40% at EOL and 44% at 10xEOL, which leads us to interpret this effect as becoming consistently proportional to the error rate after the block has endured some wear. The error trends also appear to be linear, so we can extrapolate to predict a marginal error rate of 7.5E-7 per read.

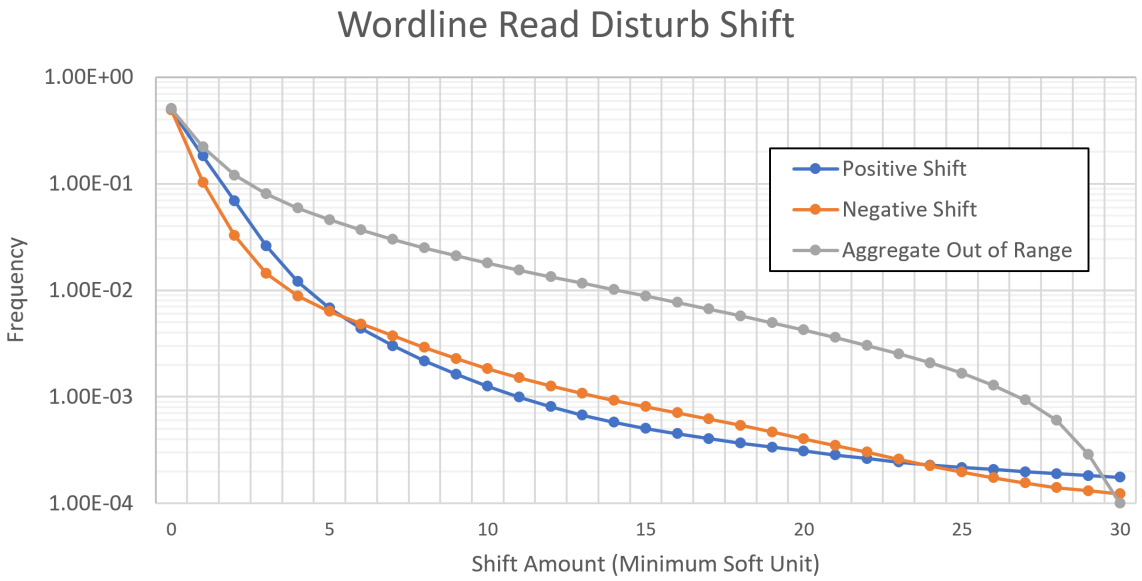

Figure 4.2: Shift in soft bit location of cells after 512k reads at end of block lifetime. "Aggregate Out of Range" represents frequency of cells shifted by greater than or equal to the amount shown.

Looking beyond the raw cell error rate, we can gain more insight into how read disturb manifests by analyzing the shift patterns in the soft bit data. For each cell index in the page, we store the precise voltage after one soft bit read, then at the end of our read disturb experiment we compare the final soft bit positions to the initial value at that index. Figure 4.2 shows this data at the block's EOL (1k P/E cycles). We can note two features of the graph that affect coding objectives. First, each write level is not represented individually because the shift patterns are

nearly identical. The exception to this is of course the erase state (ER), which is not within the measurable range for soft bit reads. Second, we note that the positive and negative shift are almost symmetrical, which indicates to us that unlike ICI and retention errors, wordline read disturb causes cells to be read as a higher or lower value with equal probability.

Bitline Disturb

Read disturbs are more commonly associated with reading other pages in a block that share a common bitline. This is intuitive because the control gate is opened with a stronger charge than the fully programmed state, and current is flowing through the cell, which applies stress to the charge held in the floating gate. We have gathered data on several aspects of this effect that shows how relative page position within a block, victim cell level, and block wear determine the probability and extent of a bitline read disturb.

Similarly to the wordline read disturb, the bitline disturb increases cell error rate proportionally to the write error rate before any disturbs. With the initial error rates for the block being analyzed at 1.2E-3, 2.5E-3, 1.8E-2, the marginal error rates are 13% at BOL, 59% at EOL, and 128% at 10x EOL. In contrast to the wordline read disturb, bitline read disturbs continue to worsen relative to the baseline error rate after the product EOL.

The symmetry of wordline read disturb direction and similar impact on each level is interesting, as most disturb effects manifest asymmetrically. Figures 4.4 and

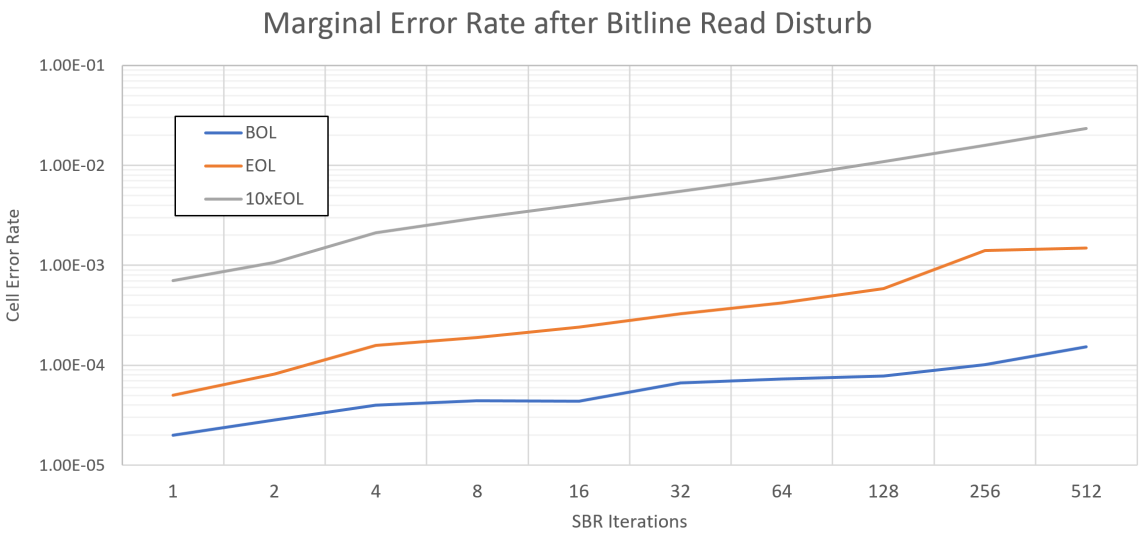

Figure 4.3: Marginal error rate after soft bit reads to other pages in the same block. Each iteration consists of 1024 page reads, totaling 512k reads.

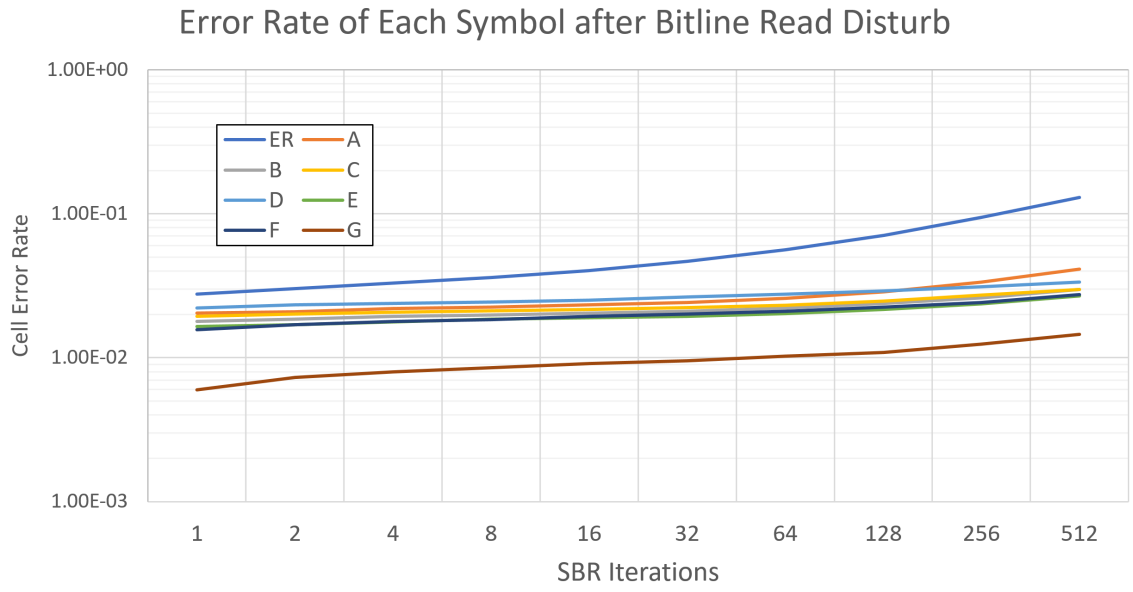

Figure 4.4: Error rate of cells written to each level after bitline read disturb, measured at end of life.

4.5 show that symmetry does not hold true for bitline read disturbs. The impact is greatest on the fully programmed (G) and fully erased (ER) levels, which tend to

shift toward a partially programmed state. Cells written to G shift downward with a 6x greater probability than they gain charge, while the lowest soft-bit measurable level (A), which is less impacted than the erase state, has a 5x greater probability to shift upward. The cells programmed to a middling state tend to have an equal probability of a less dramatic upward and downward shift, resulting in a widening of their probability distributions but not a significant increase in cell error rate.

We also consider the relative location of pages as they are impacted by bitline read disturb. Previous work in MLC [TUS15] has found that the South aggressor cells cause the most impact on a victim through ICI, and we anticipated that this effect may come into play for read disturb. In this experiment, we repeatedly read a single page in the middle of a block to disturb cells on the shared bitline. The

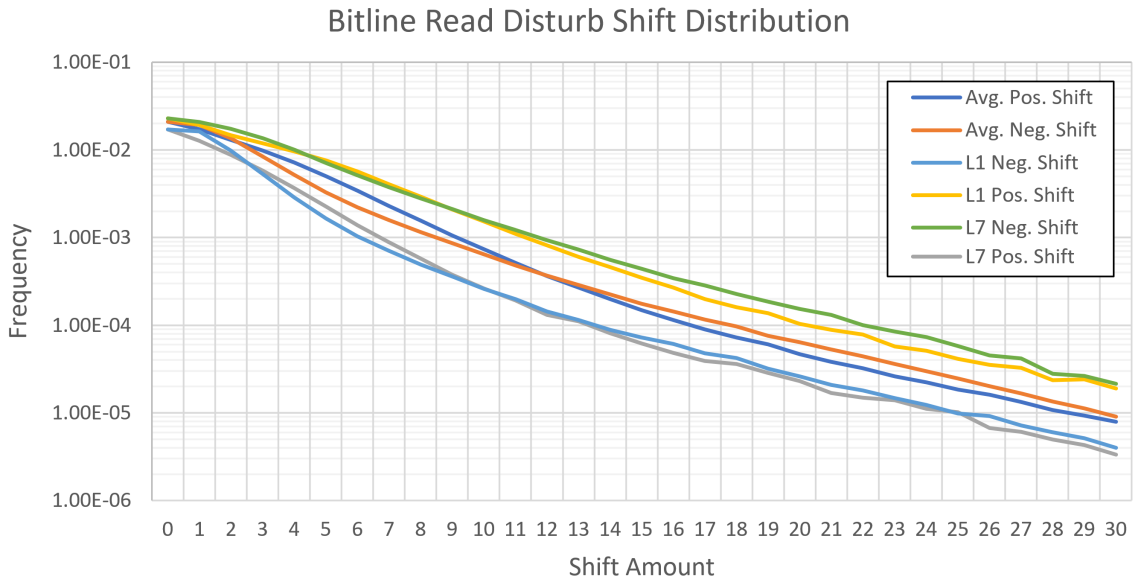

Figure 4.5: Shift in soft bit location of cells after 512k reads on different wordline at end of block lifetime.

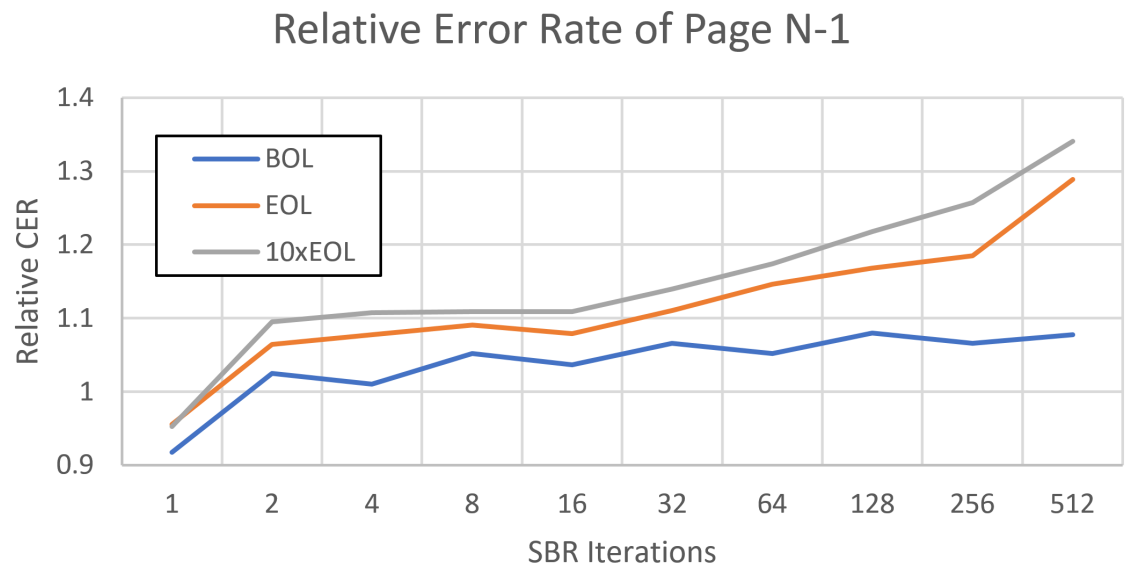

Figure 4.6: Error rate of page N-1 after soft bit reads to page N, relative to error rate of other pages.

measurements are taken on 10 pages with relative locations +/-1, 5, 10, 20, and 40. The wordlines above (lower index) the read page have a higher rate of error, as well as the wordlines immediately adjacent, with the highest error rate being the wordline

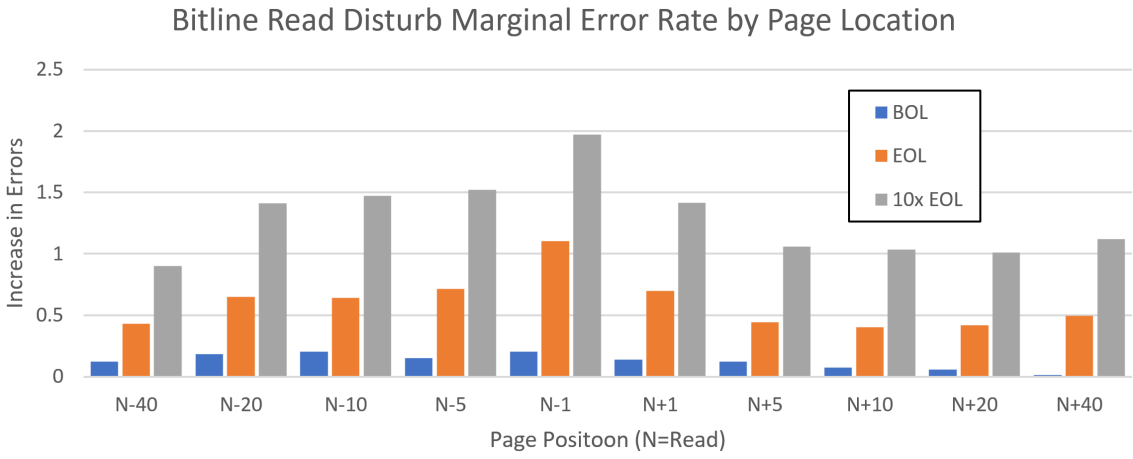

Figure 4.7: Marginal error multiplier by relative page location after bitline disturb.

immediately above the line used to apply stress. The exception is a page very near to the top of the block, which appears to be less impacted.

Figure 4.6 shows the relative error rate of wordline N-1 compared to the average of the other 9 lines. This data is in line with our expectation that the greatest effect of ICI comes from the adjacent cell in the South position. Chart 4.7 shows the error rate for each page at BOL, EOL, and 10x EOL as a function of the initial error rate, taken after 512k page reads (512 soft bit read sweeps).

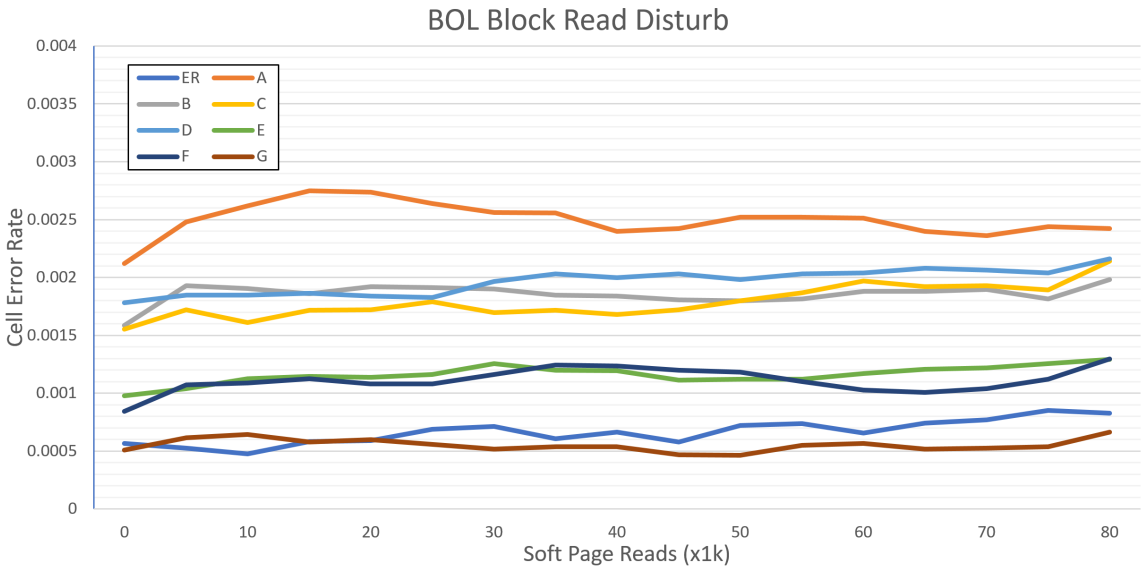

Figure 4.8: Error rate as it increases during a sequential block read after inducing wear.

The impact of this read disturb is also shown (Figures 4.8 and 4.9) for our regular use model in later experiments, which is to perform soft bit reads sequentially across the entire block. This differs from a single-page repeated read disturb in that each bitline is more likely to have some wear. We observe a greater effect of read

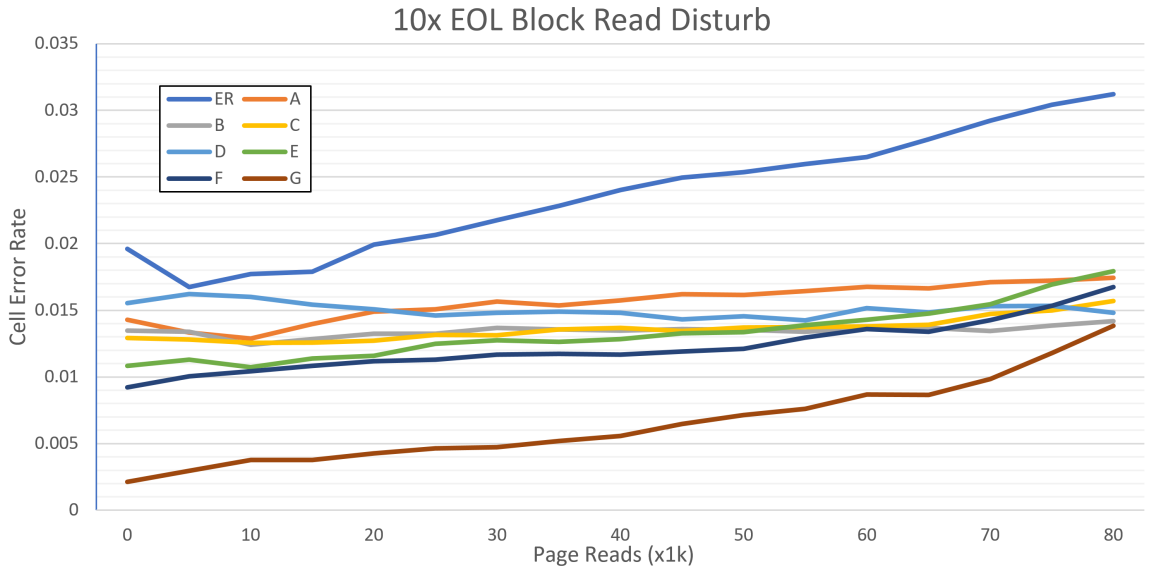

Figure 4.9: Error rate as it increases during a sequential block read after 10k P/E cycles.

disturb during sequential block reads than single-page repeated read because each wordline contains different data, and so the probability of a bitline having been used to drain current increases with each page. It is important for us to note this data so that we can understand how read disturbs appear within the results of our other experiments, allowing us to isolate other effects such as programming faults from failures caused during readback.

4.1.3 Discussion

Our results show that read disturb is a non-negligible effect in TLC flash memory, but only after significant wear is induced (well beyond the lifetime of the block). Though reads do not have a great effect on the contents of the selected

wordline, adjacent wordlines are heavily impacted, and all other pages in the block to a lesser extent. Bitline read disturbs are more common and occur asymmetrically, and each page read causes a disturb on all other pages in the block, which ultimately means 100x fewer page reads are required to cause the same amount of damage as wordline disturb.

In quantifying wordline disturb, we found that no particular cell level is more susceptible to the effect than others, and that errors are equally likely to shift a cell to a higher and lower voltage, yielding a wider probability distribution for each program level. The disturb thus has no specific implications for coding, but the quantification of error rate as a function of current wear and number of reads is useful in modeling the coding or refresh requirements for read-dominated applications.

Bitline disturb is shown to have a much higher global impact per read, and the errors manifest asymmetrically with respect to cell value, page location, and failure direction. In contrast to the wordline read disturb, cells tend to shift toward a partially programmed state through gain or loss of charge. The BOL and EOL (not shown) data are very similar, so the significant growth of cell error rate in the fully programmed and erased cells is not apparent until well after the manufacturer-specified EOL. Later data will demonstrate this in more detail.

The physical layout of the blocks makes it clear as to why pages above the selected wordline have a higher error rate than those below. When a cell is programmed to a high level, current drains through the entire bitline. The cells in an

erased or partially programmed state allow little current to drain through the pages south of the selected wordline, but the cells above that line are all fully opened by the control gate and are thus always charged while performing a read. Read disturb will always be greater on the lower-indexed (North) pages. For a read of the Nth page, the minimum current (assuming the selected cell is fully erased) to drain through wordline 0 is the capacitive load of N cells, while the N-1st page only needs to charge the bitline between itself and page N.

Moving on to the analysis of other effects, the most important data from this section is observed in Figures 4.8 and 4.9, which show that we will be influencing the results of ICI and wear analysis while gathering the data with soft bit reads. In particular, we must consider the effects of read disturb on the results for cells written to the highest and lowest levels in order to isolate the effect of ICI and tunnel oxide wear because we will observe higher error rates than were originally present after programming.

To develop coding methods tailored to improving the lifetime of a block, consideration must be taken for read-dominant workloads because of how error characteristics change as cells become worn. To compensate for the new distribution of errors and their direction, we can use shaping codes to reduce the error prone symbols or enforce a refresh policy to ensure that pages are not too heavily corrupted by repeated reads without writes.

4.2 Endurance

In this section, we determine how the wear of tunnel oxide induced by program/erase cycling influences program faults. We characterize the error rate for each programming level and show the probability distribution of soft levels as a function of P/E cycles. As with previous work in endurance coding for MLC flash [LS16], we aim to develop a cost function for the amount of wear induced by each programming level.

4.2.1 Experiment

The manufacturer specifies that the device use model requires writing pseudorandom data. We first characterize the wear in the average case P/E cycling with pseudorandom data and reading back the soft level distribution at regular intervals. We also perform tests using highly regular data to show the specific impact of each programming level on the life of a block.

Random Data

For the typical use case (e.g. highly random, unstructured, or encrypted data), we induce wear with the same method used in the section on read disturbs. We generate random data, program an entire block, then erase it. This process is repeated for up to 12x the EOL, and we read back the soft level data on 100-cycle intervals. This data is stored as a histogram of the probability mass function, sortable

by the hard level written, hard level read, P/E cycle in which data was sampled, and levels written to the neighboring data.

Patterned Data

To gather the necessary information about how much wear each program level induces, we need to perform P/E cycling with patterned data that is dominated by one level. However, the simple approach of writing a constant value to all cells is not effective. We tested this approach and found that the result is an extremely high error rate, in some cases (such as writing each cell in a page to L7), all cells fail to program. After some trial and error, we found that randomizing one in eight cells yields a cell error rate close to that of the the pseudorandom data, and so we use this pattern in our experiment.

To stress each block, we write the entire block to a constant cell level, with the exception of one random index in each eight-cell word, which is written to a random level. As with the average use case, we perform cycling with this data for up to 12,000 iterations. However, we program pseudorandom data on each 100th cycle and read back the hard and soft level data so that we can compare the error rates for the normal use model after wear by very structured data.

4.2.2 Results

We analyze the cell error rates and soft distributions for both random and patterned wear. We characterize the marginal impact of wear on cell errors, which

means we must estimate and remove the errors caused by the read disturb in order to consider solely the program failure rate.

Random Data

For error mitigation techniques, we need to know the error rate and types of errors that occur in the device. Raw cell error rate is the simplest metric of how wear affects failures in NAND flash. We show this rate from the beginning of life to 12x the end of life in Figure 4.10.

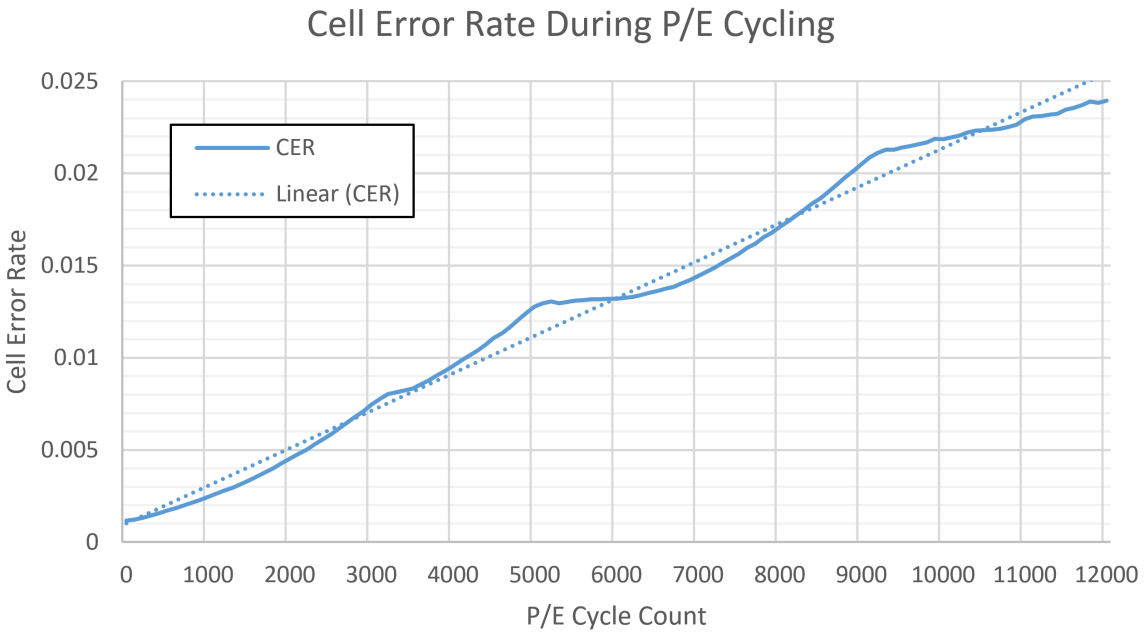

Figure 4.10: Cell error rate of 8 blocks while P/E cycling from BOL to 12,000 P/E cycles.

There are two features of the CER graph that we should note. First, failures become more probable approximately linearly at a marginal rate of 2E-6 errors per P/E cycle. This is the minimum information required to determine ECC require-

ments at any time during the block's use. The second feature is the apparent leveling off of errors at certain intervals, as if P/E cycling induces less wear at certain points.

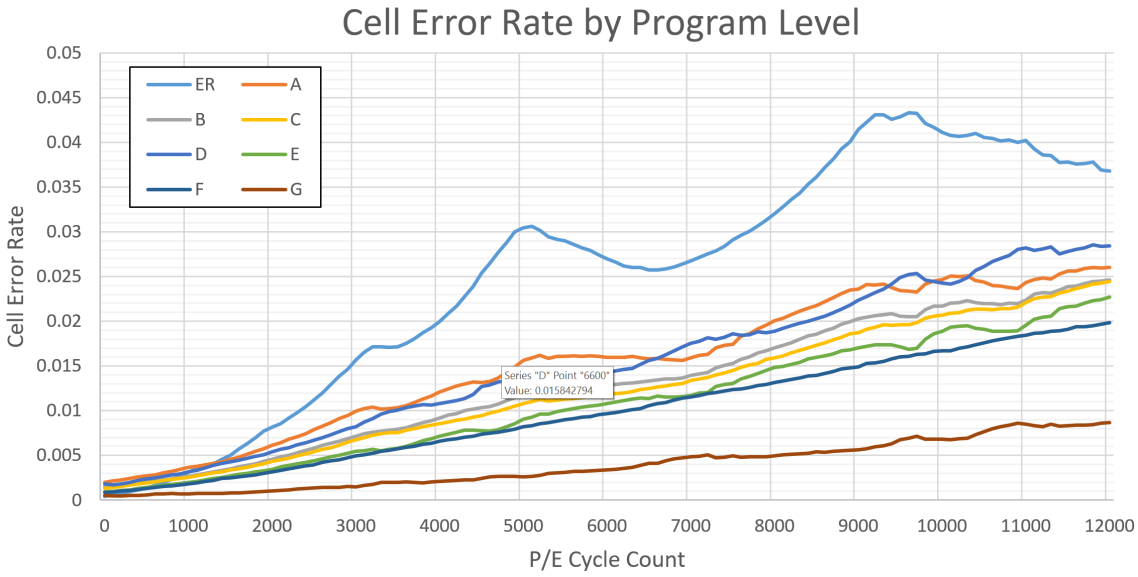

Figure 4.11: Cell error rate for each program level.

The cause of the leveling off of errors becomes more clear in Figure 4.11, which shows the CER of each program level. At certain points during cycling, the block appears to heal and certain levels (most notably ER) are less likely to fail than in previous P/E cycles. We do not know exactly what mechanism drives this unusual effect, but we are confident that it is not caused by our testing platform because we have analyzed various blocks on multiple devices at different ambient temperatures, and always found the same result.

We speculated that the ER state might be unique in the presence of these errors due to erasure failures, when cells that remain above the erased state after a block erase. In the next program sequence, cells that were on partially erased

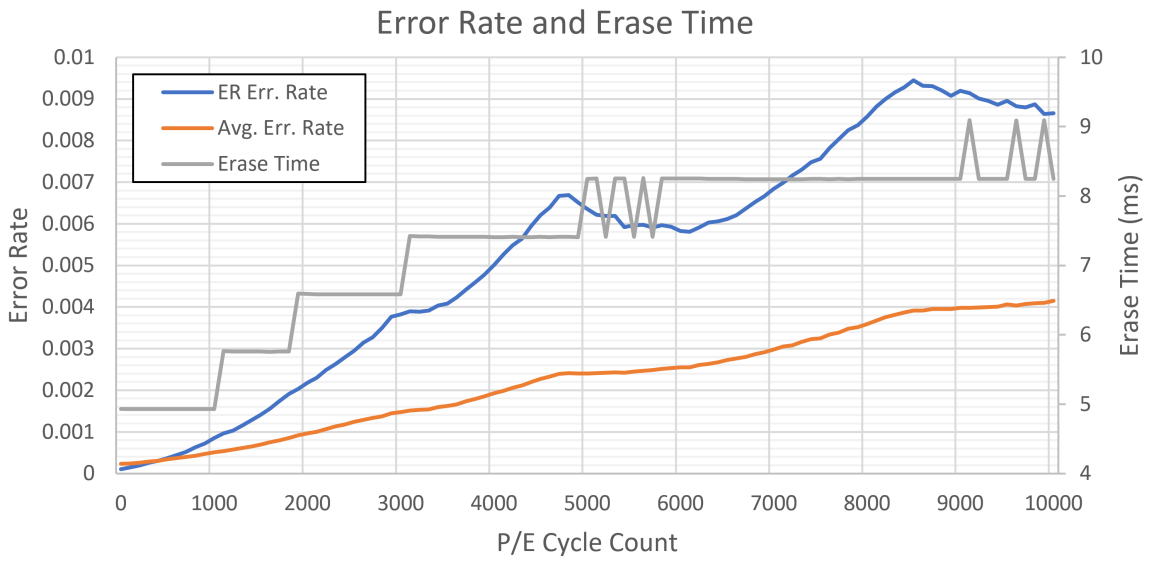

Figure 4.12: ER and average cell error rate plotted against erase time.

would likely have a low amount of charge, e.g. a low A. We checked this assertion by reading back data after a block erase, but found that erasure failures virtually never occur. If we look at the time to complete an erase operation, we can speculate as to why there are no erasure failures and how this impacts the error rate.

We can see in 4.12 that the ER error rate and erase time are highly correlated. We believe that there is a mechanism in place to guarantee no erasure failures by internally reading back data and repeating the erase operation until all cells are fully erased, thus increasing the amount of time required to erase the block. When the erase time increases, it appears to jump by a discrete amount to perform an additional iteration, which takes approximately 0.84 seconds.

As difficulty erasing some cells causes the BER operation time to increase, the cells which were correctly erased are discharged even further, widening the gap

between the cell's charge and the lowest threshold V_A. This would reduce the ER error rate by making the cells less susceptible to disturbs which would pull them up into a partially programmed state. This mechanism would partially explain the phenomenon we observe, but it is not clear why the error reduction is gradual rather than a jump at any instance where the number of erase iterations is increased.

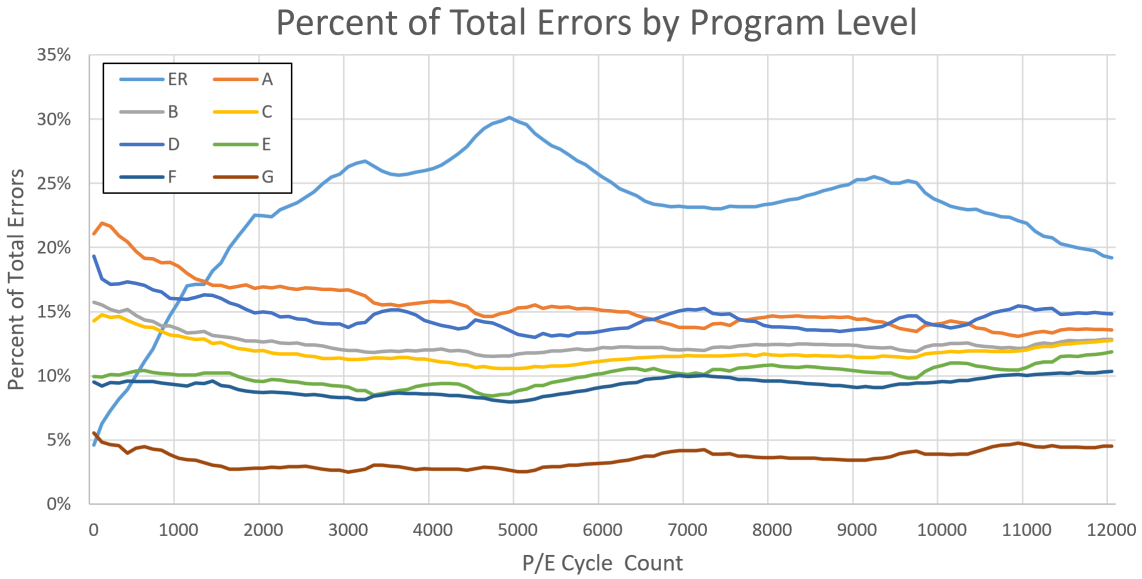

Figure 4.13: Percent of total errors accounted for by each program level.

Another visualization for the cell error rate by program level is given in Figure 4.13, where we show the percentage of total errors for which each program level is responsible. This graph makes it more clear what the relative cost of each level is in terms of the immediate cell error rate.

Even after accounting for the average increase in ER errors caused by read disturb, which is approximately 25% for an entire block read after significant wear, the relative failure rate of ER differs significantly from other cell levels in how it

varies with wear. At the beginning of life, it has the lowest error rate, then rapidly climbs until it is consistently the dominant error. Given the current level of wear, we can use these data to develop a cost function for each symbol, and use the costs to develop an ideal shaping code for error mitigation.

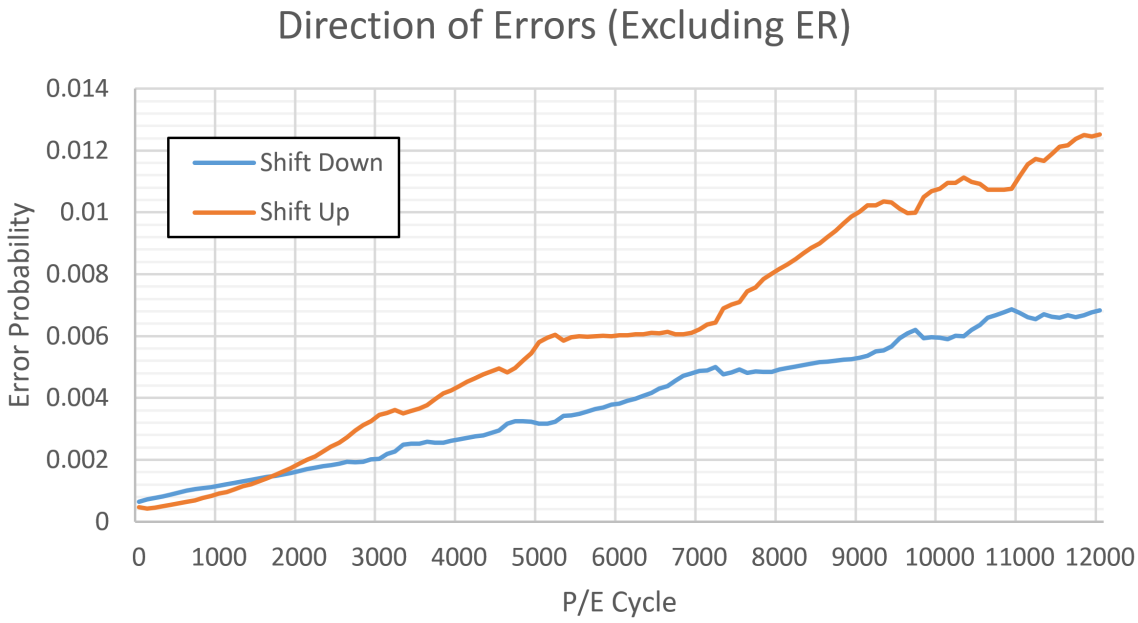

Figure 4.14: Frequency of overprogramming and underprogramming failures during P/E cycling.

In addition to the frequency of errors of each program level, we can show the direction of failures. Levels A-F can be shifted one or more hard levels up or down, while ER and G can only be shifted up and down, respectively. Since we already know that ER has anomalous error frequencies, we consider only the trend of upward and downward shift in levels A-F in Figure 4.14. For these levels, there is a much higher propensity for upward shifts, or cell overprogramming.

We can discard the effects of read disturb, which are minimal for symbols

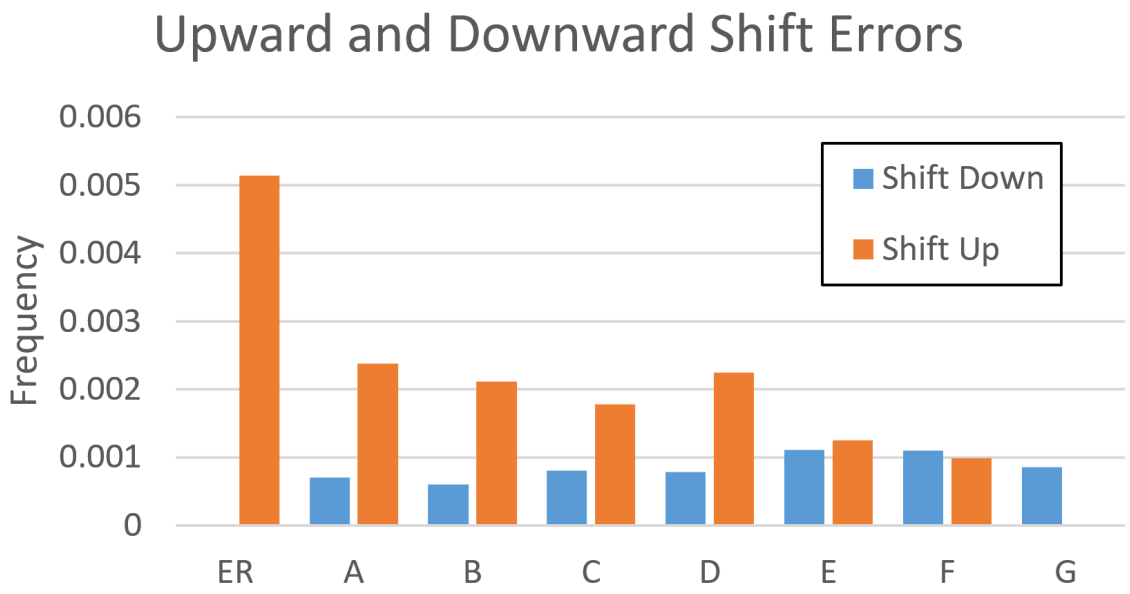

Figure 4.15: Frequency of overprogramming and underprogramming failures for each level after 10k P/E cycles.

A-F and significant but always downward for cells programmed to G, and we see that overprogramming failures are nearly twice as likely as underprogramming after the block has been significantly worn. However, this effect is not consistent across all levels. We expect that lower cells are more likely to have an upward shift than higher cells, and this is indeed the case. As we can see in Figure 4.15, levels A-D have 3.4X, 3.5X, 2.2X, and 2.8X higher probability to shift upward than downward, while levels E and F have very similar probabilities.

Patterned Data

Endurance codes have a different cost metric than error mitigation codes in that the objective is to choose the patterns which cause the least wear and thus

affect error rate for all future P/E cycles, rather than those which will reduce errors for the immediate page. In this section we find the wear cost for each program level by analyzing the error rate after wearing pages with the pattern described in section 4.2.1.

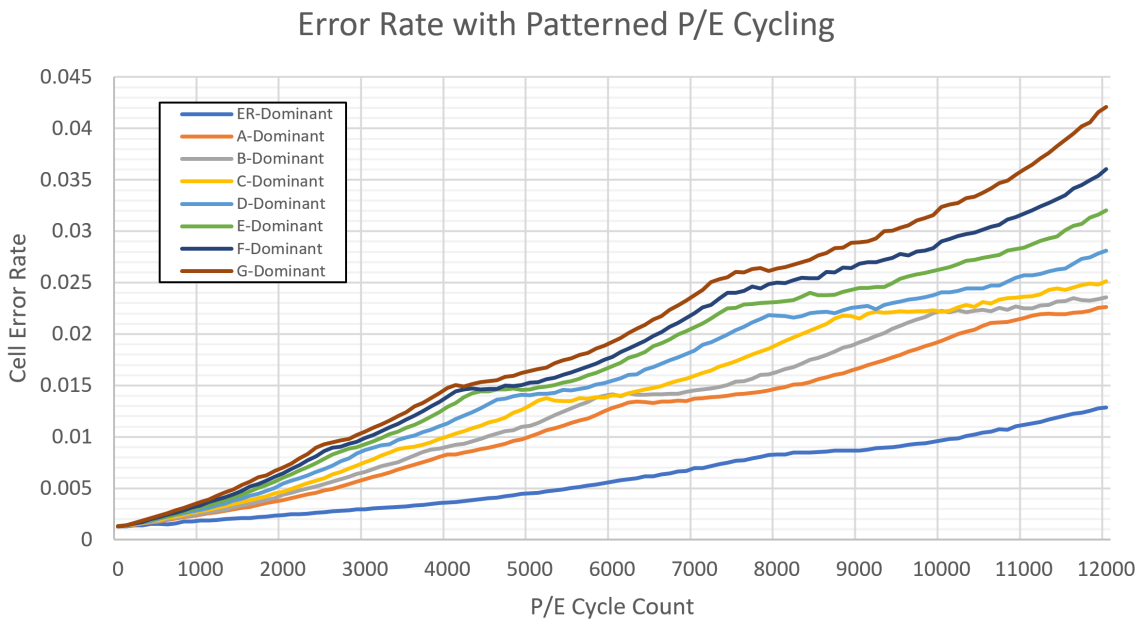

Figure 4.16: Pseudorandom data error probability after inducing wear with data that is dominated by each programming level.

Figure 4.16 shows the relative rate at which wear is induced by programming with the majority of cells written to one level. There are two ways of looking at this data. We can compare it to the raw cell error rate in Figure 4.10, or we can take note of where the peaks occur in the ER error rate and compare these to the peaks from random data P/E cycling in Figure 4.11. Using the first method, it's difficult to correlate the wear to raw error rate because of how it varies with the peaks and valleys in ER caused by an unknown effect.

The occurrence of the peaks are consistent across all blocks, and the location also appears to be dependent on the wear rather than number of P/E cycles, so we choose to use these easily identifiable points to compare the relative wear induced by each pattern. First, we select the samples where the three peaks occur for each pattern in Figure 4.11. We can compare these cycle counts to those of the peaks during random wear, which occur at 3200, 5100, and 9200 P/E cycles. Considering that each wear pattern consists of random data for one in eight cells, and the other seven cells are set to the same level, we use the following equation to find the relative cost C_L for each programming level L. The number of P/E cycles of the Nth peak for each programming pattern, either random or dominated by level L, is denoted by $PE_{N,L}$.

$$\sum_{peak=1}^{3} C_{rand} * PE_{peak,rand} = \sum_{peak=1}^{3} [1/8 * C_{rand} * PE_{peak,L} + 7/8 * C_L * PE_{peak,L}]$$

Plugging in the location of each peak yields the cost of each programming level relative to the cost of random data, which we show in the table below. The costs are calculated independently of each other, and confirm the accuracy of the ER error peak method of calculation with an average of $1.019C_{rand}$, which at least implies that if the cost model were applied to pseudorandom data, we would be within 2% of the actual wear. We report the cost function as wear relative to the average case so that the equivalent number of "random" P/E cycles can be calculated by accumulating the average cost of each page program and then used to index into the other data sets from this work to find the expected error characteristics at any

point in the block's life.

Table 4.1: Wear induced by each program level, normalized to random data.

	ER	A	B	C	D	E	F	G
Cost	0.538	0.814	0.870	0.988	1.129	1.205	1.273	1.331

4.2.3 Discussion

Several observations can be made from the data that we have reported in this section. As the cell gates are worn, error patterns are changed and previously low-cost symbols can become the primary contributors of errors. The wear incurred by writing each symbol also varies, and we have quantified their individual impacts.

We note the symbol encoding in Figure 4.17, which is different from the MLC encoding in how the pages divide the range of voltages. There are only three thresholds for two pages in MLC, so the center threshold is assigned to one page and the two remaining thresholds divide the left and right halves with the bit from the upper page. TLC devices have 7 total thresholds, but given that the vast majority of errors only shift up or down one programming level, it is unreasonable to divide the range in halves three times.

If the three pages were assigned 1, 2, and 4 thresholds by subdividing voltage ranges with each page (e.g. if the encoding was sequential from 3'b000 to 3'b111), one page would account for 12% of errors while the page with 4 subdivisions would account for 64% of errors. Instead, the encoding splits the thresholds across the

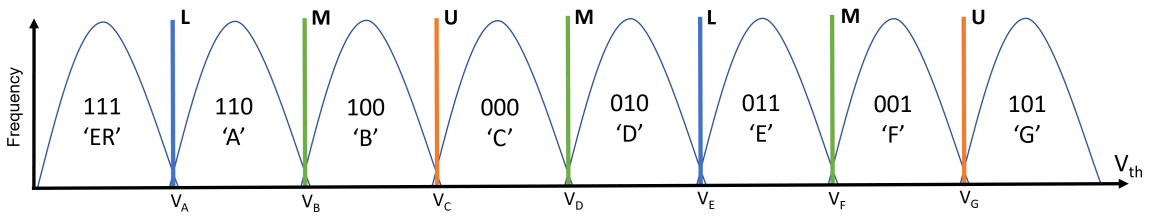

Figure 4.17: The encoding of Upper (U), Middle (M), and Lower (L) page bits into programming levels for each cell.

pages with 3 assigned to the middle and 2 assigned to the upper and lower pages. Errors are distributed more evenly across the pages, with 22%, 36%, and 42% of total errors attributed to the upper, middle, and lower pages, respectively.

The encoding of pages to cell levels in this device can be beneficial for both endurance and error mitigation codes. Because of the location of threshold V_C programmed during the upper page, we can make the decision in the upper page, after the lower and middle pages are known, to reshape cells with higher cost to wear or immediate error rate to a much lower level. This is vastly different from MLC, in which the upper page only shifted up/down one level. With the TLC encoding, the number of levels shifted by inverting the upper page bit is 1, 1, 3, and 5 for the four lower/middle page bit pairs. Endurance codes can use this to prioritize the shaping away from D and E, which become A and ER with an upper page bit of 1. Error mitigating codes can perform shaping in the opposite direction because the levels ER and A have much higher error probabilities than D and E.

With this technique in mind, the question then becomes how to adapt to the changing error characteristics as blocks become worn. If we first identify the point in

random data P/E cycling at which error mitigation codes become strictly necessary, we can utilize endurance codes until that level of wear has been achieved. The use of endurance codes changes the amount of wear applied by each P/E cycle, so we would have to use the cost model developed above to keep track of how much wear has actually been induced. Once the crossover point at the threshold of the ECC mechanism's capabilities is reached, endurance codes could be replaced with error mitigation shaping codes to shift the distribution of cells in the opposite direction.

Ideally, the lifetime of a block could be extended by using a hybrid endurance/error mitigation shaping code. Such a code would require the system to keep track of current wear on a wordline or block by accumulating the cost with the function outlined in this section, then look up the current error characteristics using the data in Figure 4.11. With the error characteristics known, the cost function as an input to a shaping code can be updated to permit the maximum acceptable error rate while minimizing wear. Initially, the error rate is low enough that no error mitigation is necessary and wear is minimized. Over time, the cost function will be shifted away from endurance in favor of shaping away from errors, until the error rate with a code purely designed to reduce errors exceeds the capabilities of the ECC mechanism and the block is decommissioned.

We can also note the direction of errors is asymmetrical, and that this information can potentially be leveraged in error correcting codes. Though we do not explore this in more detail, it is valuable to identify the relative probabilities

of certain symbol errors and their direction. The key points here are that the vast majority of errors are a single shift above or below the intended programming level, and that for the lower levels, the likelihood of an upward shift is much higher than a downward shift.

Although we do not have the facilities to test retention failures through simulation of device aging between programming and reading back the contents of a block, we can comment on one recommendation by the manufacturer and how it may be related to the direction of cell errors. A self-healing effect is reported that may improve the error characteristics of a block if there is a reasonably long period between each P/E cycle. This may be related to retention failures, in which charge slowly leaks out of the floating gate and reduces the level of the written cell.

Cells with significantly worn gates are more susceptible to overprogramming errors, which explains the trend of upward shifts being much more common than downward shifts. In our testing environment, we read back data immediately after writing, which does not allow any time for retention failures to occur. However, the cells that are overprogrammed because they allow charge to flow too freely into the floating gate may be equally likely to lose that charge over time. Thus, the overprogramming of the lower-charge levels may be healed as charge then leaks back out of them and a program failure in conjunction with a retention failure cause the correct data to be read back. It is also possible that overprogramming errors in lower-charge cells would be mirrored by retention failures in the cells with the highest

charge, but we are not able to test and quantify the accuracy of these conjectures.

4.3 Inter-Cell Interference (ICI)

Previous work quantified the effect of inter-cell interference in MLC flash, and Figure 2.4 shows how significantly neighboring cells impact error rate. The most apparent relationship in MLC was between the victim (V) cell and its northern (N) nieghbor, with up to 40x difference in symbol errors between the fully erased and fully programmed states. In this section, we extend ICI analysis to TLC flash and determine whether the effect has worsened or improved.

4.3.1 Experiment

The data required for ICI analysis is not gathered by programming special patterns and recording error rate. We wish to observe how ICI impacts the typical use case, which is writing random data, so we perform P/E cycling and add additional fields to sort the histogram of results so that we can view error patterns for each local VWENS pattern. Each aggressor cell (WENS) is stored as one of four values (ER and A, B and C, etc.), rather than all eight levels simply to reduce the size of the data set. As with our other experiments, we sample data after each 100 P/E cycles from the beginning of life to 12,000 cycles.

4.3.2 Results

Before analyzing specific effects and identifying worst-case patterns, we first compare the effect of ICI in TLC directly to MLC. The format of Figure 2.4 is replicated for TLC in Figure 4.18, where the symbols 0-3 represent {0/1, 2/3, 4/5, 6/7} in the TLC device. We did not collect significant data on the diagonal neighbors because they have been shown to only minimally impact victim cells, so only bitline and wordline neighbors are shown. The MLC data is taken for 10k cycles, which is the lifetime of a block. We show both the TLC block lifetime (first 1k cycles) and all 12k cycles of our testing.

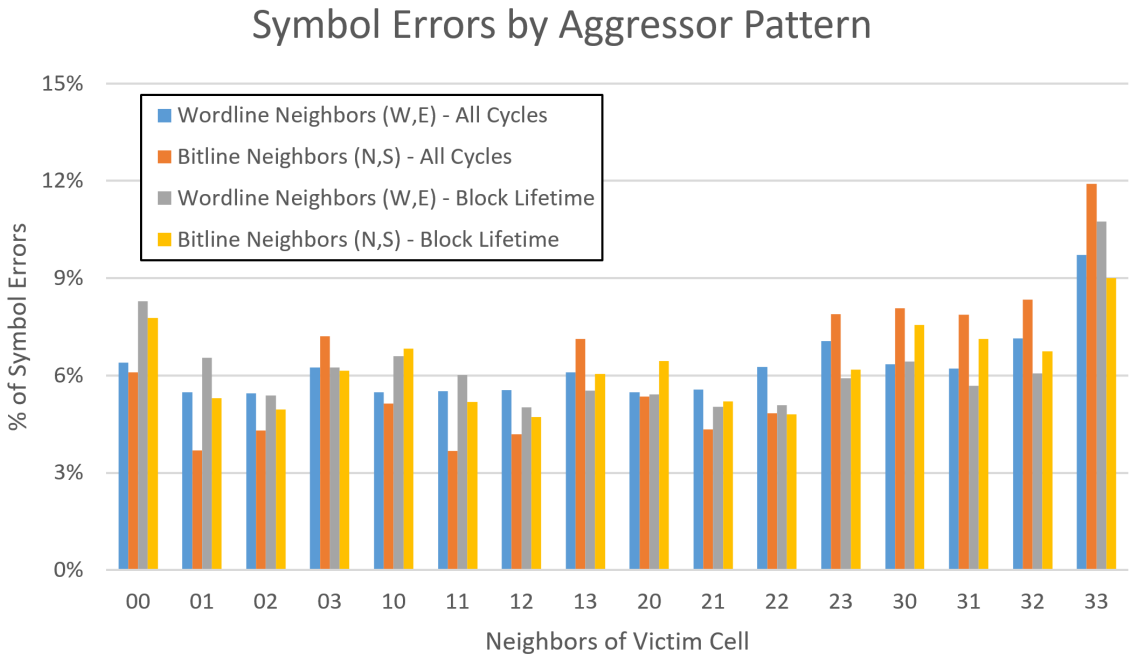

Figure 4.18: Percent of total errors accounted for by various aggressor patterns in TLC.

Compared to the MLC chart, ICI is significantly reduced in TLC flash both

during and well beyond the lifetime of a block. During the normal lifetime in MLC, the maximum difference is found in the bitline neighbor patterns of (0,0) and (3,3), with a 40x higher error rate in fully programmed neighbors. The best and worst-case patterns in TLC are (1,2) and (3,3), with only a 1.9x difference in errors.

Though this data shows that wordline neighbors don't have a significant impact on ICI, there is still a significant and observable effect that can be used in coding. This data only shows the overall error rate, not the error rate for each symbol. If we look more closely, we can see that ICI significantly changes error rates for individual victim cell levels. Figures 4.19 and 4.20 show the percent of symbol errors for each wordline and bitline aggressor pattern, with each cell level represented independently.

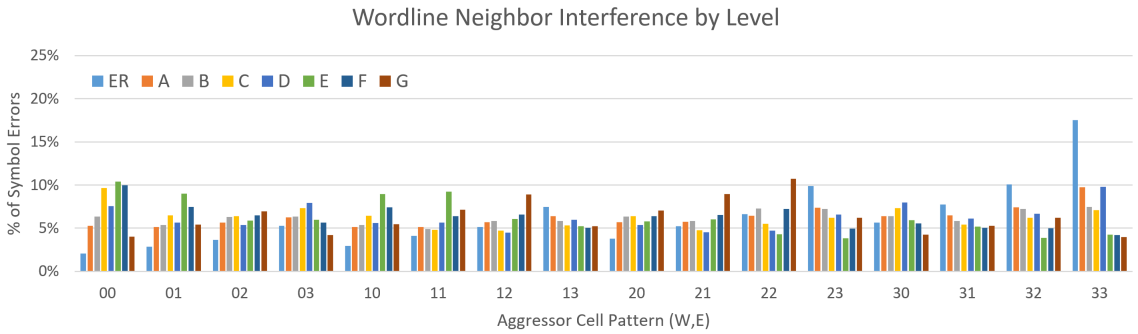

Figure 4.19: Percent of errors for each symbol accounted for by various wordline aggressor patterns in TLC, P/E cycles 0 to 12k.

While there is nothing as drastic as the 40x difference in total cell error rate observed in MLC, the varying cost of certain patterns can be significant for individual cells. The most distinct is the 42x difference in ER errors with bitline aggressors

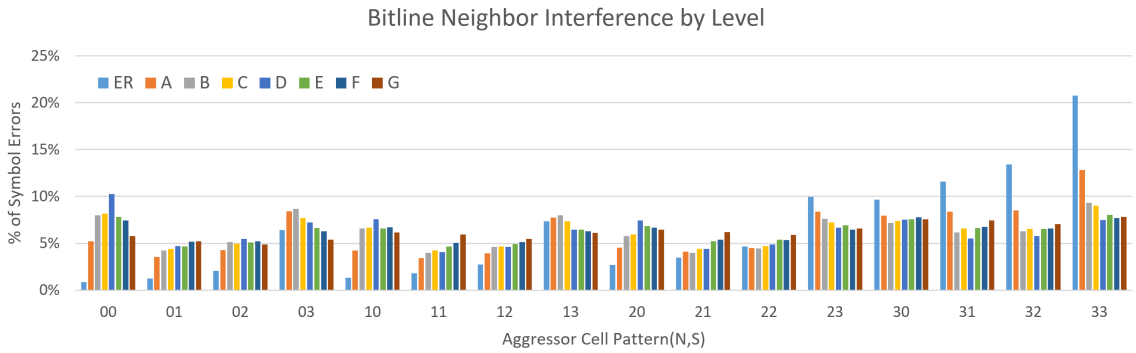

Figure 4.20: Percent of errors for each symbol accounted for by various bitline aggressor patterns in TLC, P/E cycles 0 to 12k.

(0,0) and (3,3). If all four aggressors of a cell programmed to ER are level 3 or 0, the average error probabilities across all cycles are 2.05E-1 and 7.99E-4, which is a difference of 245x.

Some other less expected patterns are also observed. For example, cells in the fully programmed state do not mirror the behavior of the fully erased state in that they are less likely to fail if surrounded by cells programmed to the opposite level than if surrounded by cells with similar values. However, the mid-level symbol D does appear to be moderately impacted by low neighbor cells. A last observation is that the symbols E and G are uniquely affected by wordline neighbors moreso than bitline neighbors.

4.3.3 Discussion

Compared to our previous findings in MLC flash devices, inter-cell interference has been significantly reduced. We believe that, as a result of ICI being significant

in previous products, the manufacturer has specifically taken measures to counteract the disturb caused by programming adjacent cells. This would explain why the programming sequence has been significantly changed in this generation.

Previous devices have had a simple programming sequence: pages are programmed sequentially within a block, and once programmed they are ready to be read back. In the TLC devices, each page must be programmed three times in total. Additionally, the two following wordlines must be known before the program sequence can be completed. Each line is programmed one iteration behind the line above it, and so programming a line is not completed until the next two lines have been programmed for two and one iterations. Finally, data cannot be read back from a wordline until the next line has completed the program sequence, meaning that you can not read data until the next three wordlines are known. Since a wordline consists of three >8kB pages in both the left and right plane, the total frame size that must be buffered is >140kB. This can be inconvenient, but has some beneficial implications for coding that we will discuss later.

The feature to note about the new program sequence is that the pages above and below must be known in order to complete the sequence, and programming takes place in three steps. We believe this is so compensation can be made for interference from the adjacent cells while programming, the result being that ICI is greatly reduced. Despite ICI no longer being the dominant factor in error rate, the data in figures 4.19 and 4.20 can still be used to develop a cost function for the

row-by-row ICI-mitigating coding techniques in [LS16].

4.4 Cell Location

Although not related to any disturb or wear, a cell's physical location within a block may be related to its error characteristics. Location dependence was observed in [CHMM12], which reports a relationship between the wordline and both retention and program interference errors. This paper does not mention the location within a wordline, i.e. the bitline index, having an effect on the error frequencies.

In this section, we analyze error distributions by both wordline and bitline index. The data is collected during normal wear analysis, where we record histograms for both column and row. Read disturb is virtually eliminated in the data collection for this test, as we are not utilizing any soft bit data.

4.4.1 Wordline Location Dependence

We found that there is a relationship between the wordline index and programming error rate, although not nearly as drastic as that found in [CHMM12]. Figure 4.21 shows the error rate of each programming level across a block.

The first similarity we observe is in the first and last wordlines, which have a lower error rate due to the lack of either N or S aggressor cells, and therefore reduced ICI. We can also see that the frequency of errors increases across the page, but the difference is not significant for most levels.

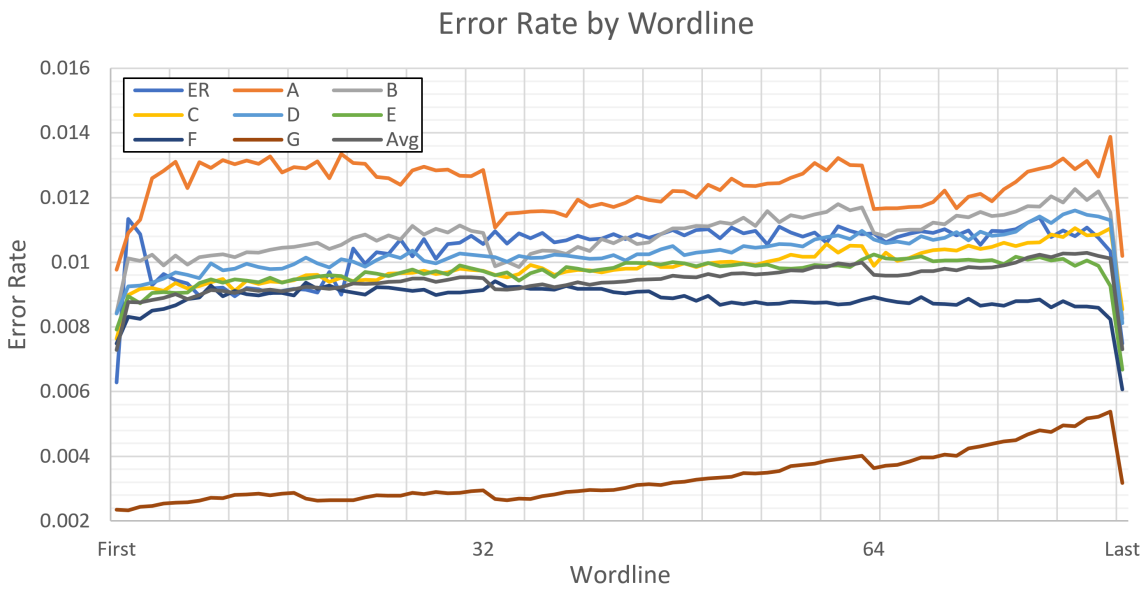

Figure 4.21: Error rate of each programming level by wordline index within a block.

We also note that there is a noticable drop in error rate for levels A, B, and G at indices 32 and 64. The decrease is less than 10%, which is also negligible compared to other factors. From this data we can infer some information about the structure of a block, as wordlines appear to be grouped by 32 rows, but it ultimately has few implications for coding.

4.4.2 Bitline Location Dependence

In contrast to the wordline location dependence, we found the bitline index to have a significant impact on error rate. However, this impact is not from a global pattern across a wordline.

Figure 4.22 shows the average error rate of each group of 128 contiguous cells across the wordline, which has more than 128K cells. It is clear that there is no

difference in the error rate across a page, as each cell group falls within the same range.

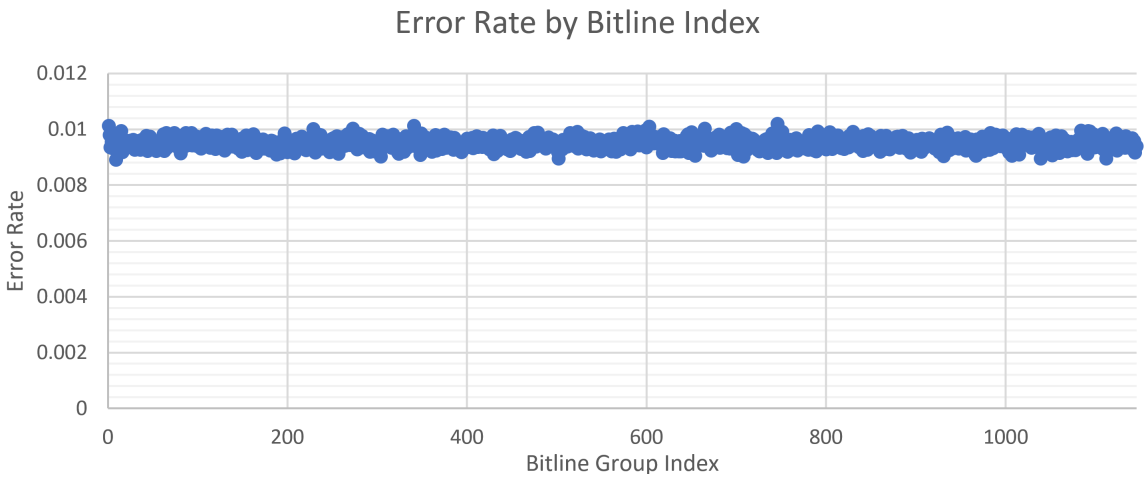

Figure 4.22: Error rate across a wordline, by groups of 128 bitlines.

Instead of looking at the global index, we can observe errors within each cell group by plotting the error rate against the index modulo some interval. We found that the smallest interval for which there was a consistent pattern is 32 bits, which is shown in Figure 4.23.

Although the difference is slight, the pattern of the fourth byte in each 32 bit sequence is always distinguishable from that of the first three bytes. The more obvious patterns are observed in groups of 2 and 8 contiguous cells.

The greatest difference is in the even- and odd-indexed bitlines, in which there is a maximum of a 55

For the purposes of generating a cost function to use in coding, we found that the general location of a cell within a block is mostly negligible. However, the

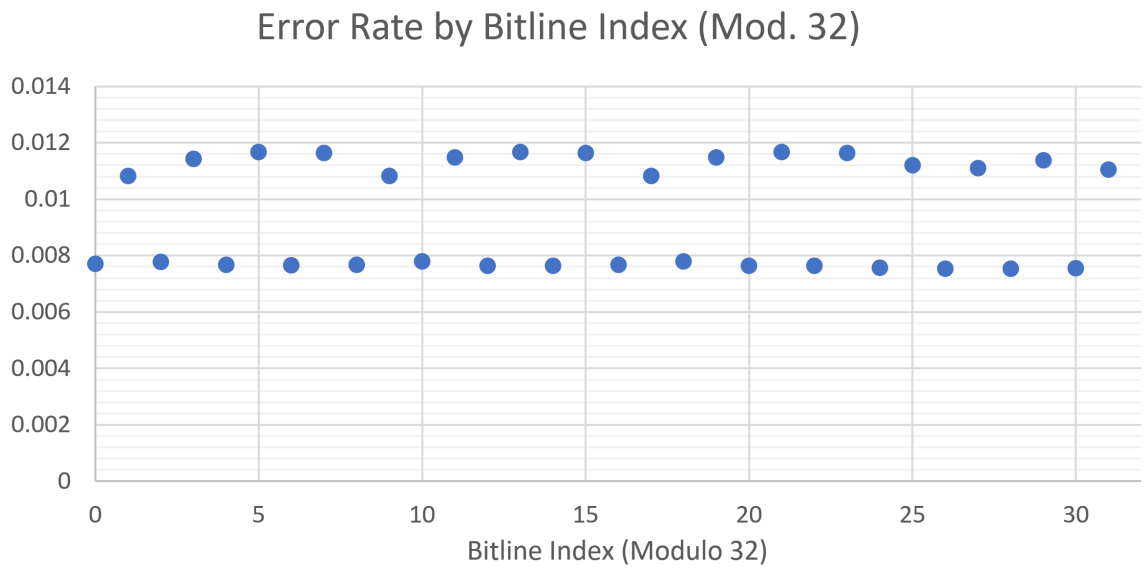

Figure 4.23: Error rate across a wordline, by bitline index modulo 32.

even/odd bitlines have significantly differing error characteristics which affect code development. To accurately predict the error rate of each cell, the codeword length must be an even number of cells, with a marginal improvement if a multiple of eight.

Shaping codes could also use the bitline index to shift the distribution of the more error-prone or wear-inducing programming levels onto the even-indexed bitlines so as the mitigate their impact. With a maximum difference of 55% between the indices with the highest and lowest error rates, this feature becomes extremely important in developing an accurate cost.

Chapter 5

Conclusions

In this work, we describe the extension of MLC NAND flash characterization to the newer TLC devices. To achieve greater density, the block lifetime has again been reduced by an order of magnitude. With a significantly shortened life, understanding how the various disturbs manifest as failures throughout the device lifetime is paramount in developing codes which extend the usage. Our objective was to determine which disturbs and effects have worsened, remained the same, or have improved through manufacturer intervention or intrinsic properties of the new technology.

5.1 From MLC Coding to TLC

Previous work in the CMRR has identified codes which address various disturbs in the MLC flash. Endurance codes are developed to extend lifetime by reduc-

ing wear, error-mitigating shaping codes can reduce errors by avoiding programming levels with the highest failure rate, and ICI-mitigating codes can use the contents of adjacent pages or bits within pages to avoid the worst-case aggressor-victim cell patterns. With the data we have gathered on how failures manifest in TLC flash, we can adapt these codes to the new characteristics of TLC.

The simplest way to utilize the results of this work is to update the cost function of endurance and error mitigation codes to accomodate the new cell level encoding and failure rate of each level. Our error analysis yields the data necessary to apply the work of [LS16] to TLC with minor adjustments. For example, we can use these shaping codes for both the middle and upper page, where it was previously only used for the upper page to eliminate buffering of the lower page, and see much more benefit in TLC because the encoding scheme allows for greater shifts toward or away from the highest/lowest levels, as we outlined in section 4.2.3.

We note, however, that the relative failure rate of each level changes as blocks become worn. For example, the fully erased state has the fewest errors at BOL, then rapidly climbs to become the dominant error at a point beyond the specified EOL. The failure rate continues to climb until ER is more than twice as likely to cause an error as any other state, after which the other states begin to catch up. The changing relative error rates may merit some adaptive codes that change their objective over the lifetime of a block, using the current amount of wear to dynamically update a cost function.

Another option is a hybrid error-mitigating/endurance code that gradually transitions from shaping toward the lower states, which reduces wear, to higher states, which have lower error frequencies after significant wear is induced. The wear cost model we presented in 4.2.2 can be used to track the amount of wear accumulated, and index into the dataset from Figure 4.11 to find the current error characteristics. With the failure rates known, data can be shaped such that the maximum endurance code is applied without exceeding the capability of the ECC mechanism. Combining these codes could adapt to the changing characteristics to extend the usability of a block well beyond the lifetime of the block with only endurance or error mitigation.

We also characterized the effect of inter-cell interference to determine whether the ICI-mitigating codes for MLC considered in [BSY15] and [TUS15] are applicable to TLC. One of the major concerns in developinng these codes was the large frame size. For coding against ICI within the wordline, we must store or read back the lower page in order to determine the symbol pattern in the wordline. For a row-by-row code, the frame increases to include a total of three wordlines. This requirement is considerably worse in TLC, as each wordline consists of the left and right plane, and there are three pages per plane in the wordline, for a total of 6 pages. If we must consider a total of 3 wordlines, 18 pages are required for the code.

Such a large buffering requirement could be considered prohibitively costly, and considering the page below the wordline being encoded is not possible if we

enforce a policy of low or no additional latency. However, the new programming sequence for TLC already requires that we buffer three wordlines in order to complete the 3-stage programming. Since this is already necessary for programming in the typical use model, we can take advantage of the available data in the north, west, and east aggressor cells to code for reduction of ICI with no additional cost.

Although the new programming sequence makes ICI-mitigating codes more viable, our data shows a significantly lessened effect in TLC. It appears that the 3-stage programming model was developed by the manufacturer to reduce ICI by compensating during writes, which renders coding against it unnecessary. There is still a noticable effect as shown in 4.3, but other coding techniques may be more beneficial because ICI is no longer the dominant source of errors. This would be a valuable determination to make in future experiments.

5.2 Relationship Of Application and Coding

Throughout our analysis we have noted that certain codes will be differently impacted depending on the application's use of the device. There is no single coding solution that is ideal for all applications due to the varying types of stress that are placed on a block.

The first effect we characterized is read disturb. Though the effect can be significant for a read-dominated workload, it appears that simply refreshing the page would be more beneficial than coding against this type of error. The highest rate of

failure caused by read disturb is a downward and upward shift from the highest and lowest states, which at the beginning of life generally have the lowest programming failure rate. It would be counterproductive to shape the distribution of cell values away from ER and G. On the other hand, ER becomes the dominant error later in a block's life, and reducing the frequency of cells written to ER would have a compounded effect of reducing both read disturb and overprogramming failures.

In addition to the number of reads between overwriting a block, another significant feature of the system or application would be the amount of time between writing a page and reading it back. Although we do not characterize retention failures in this work, it is understood that over long periods of time, charge can leak out of the floating gate of a cell and cause a shift to a lower state. This effect mirrors the overprogramming errors which are dominant in low-voltage cells, causing an increase in error rate in the highest cells and becoming more prevalent when a block is worn. It may even be the case that retention failures mitigate cells that have been overprogrammed given the correct amount of time between a write and read. We do not know the extent to which this effect occurs in TLC, but it should be considered when selecting the coding objectives which best suit a particular system.

5.3 Relationship of Disturbs and Coding

As a final note, we wish to discuss how the various disturbs interrelate and how coding to mitigate a disturb can have a compounded effect when the other

characteristics are taken into consideration. It is sometimes the case that a coding technique can have unintended positive or negative side effects through the other disturbs.

Consider first an endurance code that minimizes the total wear cost using the data in section 4.2.2. The lowest wear is incurred by the fully erased state, with each consecutive increase in level having a higher cost. If data is reshaped such that more low-level cells are written, the effect of ICI will also be lessened by reducing the frequency of fully-programmed aggressor cells. If ICI is lessened significantly, the dominant ER overprogramming errors are reduced, and it is possible that both wear and immediate error rate are reduced.

The converse case arises when an error mitigating code is utilized late in a block's lifetime. Such a code would shape data away from the lower states that are prone to errors. Though there are fewer cells programmed to ER, the probability that they are surrounded by high-level aggressor cells is increased. In this case, an error mitigation scheme could exacerbate ICI and be much less effective than anticipated, possibly even causing a higher overall error rate.

We cannot be sure of the efficacy of a code without implementing and testing, or at least simulating its effect using the data we have collected. Some codes may cause more harm than good, while others may be much more effective than predicted.

Chapter 6

Final Notes

In this work we have characterized several effects in one TLC NAND Flash device, and how they interact. We use this information to discuss how the effects might impact coding and development of cost models, but we do not specifically propose coding solutions. This thesis has laid the groundwork and makes data available for future coding efforts, which can utilize different aspects of this report based on the capabilities of the code and objectives of the user. For example, if a code is able to take advantage of asymmetrical direction of failures, or the user doesn't have latency requirements and can utilize a large frame buffer, the cost model varies and different sections of this report become pertinent. With the exception of retention failure analysis, we have provided most of the information necessary to predict the type and frequency of errors throughout the device's lifetime if the workload is well undersood.

Bibliography

[BSY15] S. Buzaglo, P. H. Siegel, and E. Yaakobi. Coding schemes for inter-cell interference in flash memory. In Proc. *2015 IEEE International Symposium on Information Theory (ISIT)*, pages 1736–1740, June 2015.

[CHMM12] Yu Cai, Erich F. Haratsch, Onur Mutlu, and Ken Mai. Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '12, pages 521–526, San Jose, CA, USA, 2012. EDA Consortium.

[GDS12] Laura M. Grupp, John D. Davis, and Steven Swanson. The bleak future of NAND flash memory. In *Proceedings of the 10th USENIX Conference on File and Storage Technologies*, FAST'12, pages 2–2, Berkeley, CA, USA, 2012. USENIX Association.

[LS16] Y. Liu and P. H. Siegel. Shaping codes for structured data. In Proc. *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6, Dec 2016.

[SA14] E. Sharon and I. Alrod. Data shaping for improving endurance and reliability in sub-20nm NAND. In *Non-Volatile Memories Workshop*, March 2014.

[Tor16] O. Torii. Practical row-by-row code design for 19nm mlc nand flash memory. In Proc. *CMRR Research Review, UC San Diego*, October 2016.

[TUS15] V. Taranalli, H. Uchikawa, and P. H. Siegel. Error analysis and inter-cell interference mitigation in multi-level cell flash memories. In Proc. *2015 IEEE International Conference on Communications (ICC)*, pages 271–276, June 2015.

[TUS16] V. Taranalli, H. Uchikawa, and P. H. Siegel. Channel models for multi-level cell flash memories based on empirical error analysis. *IEEE Transactions on Communications*, 64(8):3169–3181, Aug 2016.