Characterization and Error-Correcting Codes for TLC Flash Memories

Eitan Yaakobi, Laura Grupp, Paul H. Siegel, Steven Swanson, and Jack K. Wolf
University of California, San Diego La Jolla, CA 92093 – 0401, USA
Emails: {eyaakobi, lgrupp, psiegel, swanson, jwolf}@ucsd.edu

***Abstract*—Flash memory has become the storage medium of choice in portable consumer electronic applications, and high performance solid state drives (SSDs) are also being introduced into mobile computing, enterprise storage, data warehousing, and data-intensive computing systems. On the other hand, flash memory technologies present major challenges in the areas of device reliability, endurance, and energy efficiency. In this work, the error behavior of TLC flash is studied through an empirical database of errors which were induced by write, read, and erase operations. Based on this database, error characterization at the block and page level is given. To address the observed error behavior, a new error-correcting scheme for TLC flash is given and is compared with BCH and LDPC codes.**

I. Introduction

Flash memories are, by far, the most important type of non-volatile memory in use today. They are employed widely in mobile, embedded, and mass-storage applications, and the growth in this sector continues at a staggering pace. Moreover, since flash memories do not suffer from the mechanical limitations of magnetic disk drives, solid-state drives have the potential to upstage the magnetic recording industry in the foreseeable future.

Flash memory chips may use single-level cell (SLC) technology, where each cell can store one binary digit, or multi-level cell (MLC) technology, where each cell can store multiple binary digits. In this work, we assume that MLC chips store two bits in a cell and TLC chips store three bits in a cell. In early stages of flash memories, only low-redundancy error detection and correction (EDAC) codes, such as Hamming codes and error-detecting cyclic redundancy check (CRC) codes, were used. However, reducing the flash memory cell size and using MLC flash technology, has created the need for more powerful ECC methods, such as BCH codes and Low-Density Parity-Check (LDPC) codes.

The design of effective error-correcting codes requires a comprehensive understanding of the error mechanisms and error characteristics of flash memories. To help address this need, the works in [7] and [15] used an extensive empirical database of errors observed during erase, write, and read operations on a flash memory device. In this work, we gathered similar error statistics from several TLC flash memory chips. For each block, we repeated continuously the following process thousands of times:

1) Erase the block.
2) Write pseudo-random data into the block.
3) Read the block and identify errors by comparing the originally recorded data to the data that was read.

As mentioned in [15], we note that these experiments were conducted in a controlled laboratory environment and the error data was collected from only a few blocks on each chips. Therefore, the reported results do not account for possible variability among blocks on any given chip.

This research was funded in part by Toshiba Corporation and by the University of California Lab Fees Research Program, Award No. 09-LR-06-118620-SIEP.

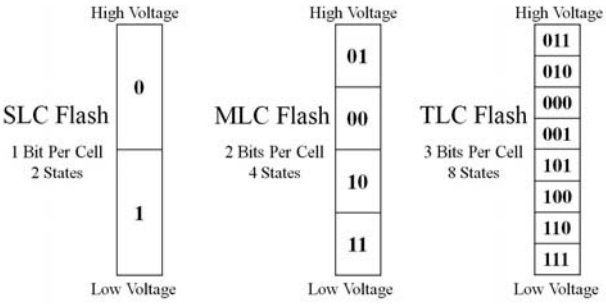


Fig. 1. Mappings of cell levels to binary representations in SLC, MLC, and TLC flash.

In this report, we extend the results reported in [7] and [15]. In Section II, a description of the structure of TLC flash memory is given. In Section III, we provide an error characterization of TLC flash at the block and page level. In Section IV, a comparison is made between different schemes that use a TLC block as an MLC or SLC block. Then, in Section V, we compare the performance of different ECC codes with respect to the measured error profile. In Section VI, we provide a new ECC scheme to be used in TLC flash and compare it with the other ECC codes. Section VII concludes the paper.

II. Flash Memory Structure

A flash memory chip is built from floating-gate cells which are organized in blocks. Each block typically contains either 64 pages (SLC), 128 pages (MLC) or 384 pages (TLC), where the size of a page can range from 2KB to 8KB.

An SLC flash memory cell has two levels and stores a single bit in each cell. An MLC flash memory cell has four levels and stores two bits, where the left bit among the two bits is called the Most Significant Bit (MSB) and the right bit is the Least Significant Bit (LSB). The mapping between charge values and bit values is depicted in Fig. 1. Similarly, in TLC flash, each cell stores three bits: MSB, CSB (Central Significant Bit), and LSB. The corresponding mapping between charge values and bit values is also given in Fig. 1 [14].

The memory cells are organized in blocks, where typically each block contains 64 pages (SLC), 128 pages (MLC) or 384 pages (TLC) and the size of a page can range from 2KB to 8KB. In MLC flash, the two bits within a single cell are not mapped to the same page. Rather, the collection of MSB's from a group of cells constitute a page called the MSB page and, similarly, the LSB's from the same group of cells form a page called the LSB page. A similar picture is derived for TLC flash. Here we distinguish between the MSB, CSB, and LSB page derived from the same group of cells [14]. The block layout of TLC block is depicted in Table I. Note that in the

TABLE I
TYPICAL LAYOUT OF A TLC BLOCK

Row Index	MSB of the first 2^{16} cells	CSB of the first 2^{16} cells	LSB of the first 2^{16} cells	MSB of the last 2^{16} cells	CSB of the last 2^{16} cells	LSB of the last 2^{16} cells
0	page 0			page 1		
1	page 2	page 6	page 12	page 3	page 7	page 13
2	page 4	page 10	page 18	page 5	page 11	page 19
⋮	⋮	⋮	⋮	⋮		
62	page 362	page 370	page 378	page 363	page 371	page 379
63	page 368	page 376		page 369	page 377	
64	page 374	page 382		page 375	page 383	
65	page 380			page 381		

first and last row only the MSB page is stored, and in rows 63 and 64 the LSB page is not stored. One possible explanation we assume for this structure is that the first and last rows are more vulnerable to errors and thus by storing only the MSB and CSB pages the BER is not too high. We will later see that in general the MSB pages have the lowest BER. In this work, we follow the scheme in [14] to program the three bits in a TLC cell.

Each page in a flash memory block contains a spare area. If the page size is 2KB then a typical spare area can be 64B. A portion of this spare area is used to store metadata in order to build the FTL once the flash memory is activated. The rest of the spare area is dedicated to storing the redundancy bytes of EDAC codes [6].

Remark 1. The organization of pages in a flash memory block may differ from one manufacturer to another. The configuration shown in Table I is consistent with the information available to us about the devices tested, as well as with most of the results of our experiments.

III. ERROR CHARACTERIZATION OF TLC FLASH

In [15], we provided a characterization of the error behavior in SLC and MLC flash. In particular, we analyzed the BER in the block-, page-, and bit-level. Here, we give a similar analysis for TLC flash. We first measured the average BER as a function of the number of program/erase cycles for several TLC blocks. The results are shown in Fig. 2.

Next, we examined the BER of individual pages within a block. Fig. 3 shows the average BER as a function of the number of program/erase cycles for each of the first 100 pages in the block. The same pattern of average BER repeats for the rest of the pages so we omit the details. We deduce that the MSB pages have the lowest BER. For the CSB and LSB pages, we see that the LSB pages on the left side of the block and the CSB pages on the right side of the block have higher BER than the LSB pages on the right side and the CSB pages on the left side. The consistency of this spatial variation in the page-level BER suggests that it is due to some property of the TLC block, but we do not have sufficient information about the device to offer a conclusive explanation.

At the bit level, we measured the total number of errors in each cell, just as we did in [15] for SLC memories. In contrast to the SLC case, we did not find that the errors were clustered along the bit lines; rather, the bit-error locations appear to be randomly distributed among the cells.

IV. PARTIAL CELL STATE USAGE IN TLC FLASH

In order to evaluate the effect on reliability of storing three bits (i.e., eight levels) per cell, we measured the block-level

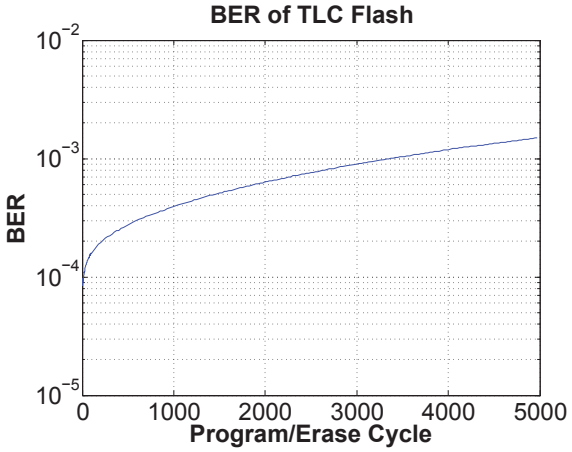


Fig. 2. BER of TLC chips as a function of the number of program/erase cycles.

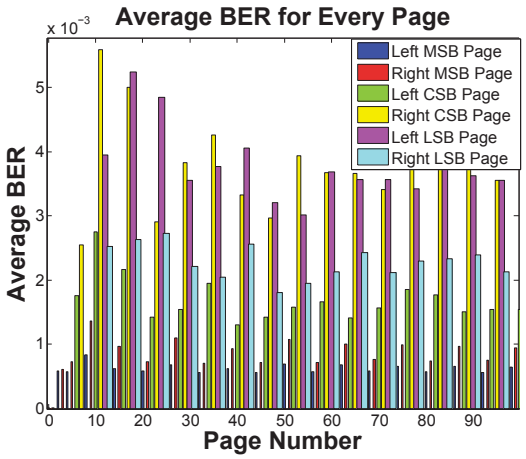


Fig. 3. Average BER of the pages of TLC block.

BER under the following restricted programming scenarios: (a) programming one bit per cell using only the MSB pages; or (b) programming two bits per cell using only the MSB and CSB pages. We considered two cases, the first being where the programming restrictions were introduced at the start of the experiment, and the second where they were introduced after 2000 normal TLC program/erase cycles. The results are depicted in Fig. 4.

V. ECC COMPARISON

We evaluated several different ECC schemes for use in flash memories, including BCH codes, several families of LDPC codes, and a new scheme designed specifically for TLC flash devices, which will be described in Section VI.

In our analysis of BCH code performance, we assumed that if the code could correct t errors per page, then it would correct any page with at most t errors. If the number of errors

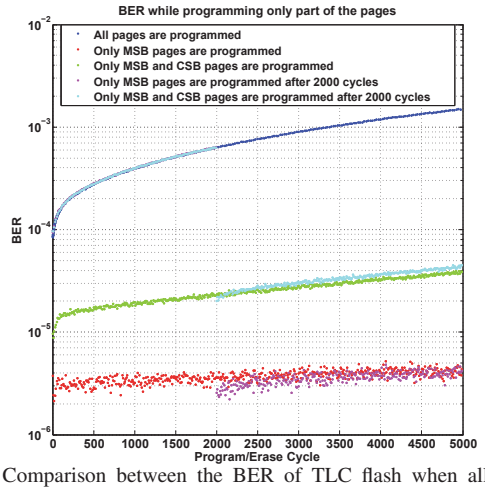


Fig. 4. Comparison between the BER of TLC flash when all pages are programmed, only the MSB and CSB pages, or only the MSB pages are programmed.

exceeded t, we assumed that the BCH decoder would detect this and leave the page unchanged.

For the LDPC code performance evaluation, we assumed that the all-zero codeword was stored, and that the memory introduced errors in the locations indicated by our empirical measurements. We treated the channel as a binary symmetric channel (BSC) with "crossover" error probability p equal to the average probability of error reflected in the measured error data. The decoder was based upon belief-propagation (BP) decoding, implemented in software as the floating-point sum-product algorithm (SPA).

The following families of LDPC codes were considered in this study.

1) LDPC $(3, k)$-regular Gallager codes.
   We used Gallager's method for designing "random" regular LDPC codes [5] to construct $(3, k)$-regular LDPC codes of block length 2^{16} bits and rates 0.8, 0.9, or 0.925. No attempt was made to eliminate length-4 cycles in the corresponding Tanner graphs. We refer to these codes in the sequel as "Gallager" codes.
2) Protograph-based low-density convolutional codes.
   A protograph [12] is a relatively small bipartite graph from which a larger graph can be obtained by a copy-and-permute procedure. The protograph is copied M times, and then the edges of the individual replicas are permuted among the M replicas to obtain a single, large bipartite graph referred to as the derived graph. Such an expansion preserves the degree distribution and hence can be used to construct LDPC codes. LDPC codes constructed through protograph expansions of convolutional codes have been shown to have belief-propagation thresholds close to the maximum a posteriori probability (MAP) thresholds [10].
   The rate-4/5 protograph-based LDPC code used in our simulation was obtained by expanding convolutional codes (with memory equal to two) using random permutation matrices, with matrix sizes chosen to ensure the required blocklengths. For this preliminary comparison, we did not make use of more refined design techniques, such as the progressive edge growth (PEG) method [8] or the approximate cycle extrinsic message degree (ACE) conditioning algorithm based upon the approximate cycle extrinsic message degree [13]. We will refer to this code as a "PCC" code.
3) AR4JA protograph-based LDPC codes.
   A family of protograph-based LDPC codes, known as Accumulate Repeat 4-Jagged Accumulate (AR4JA) codes, has been chosen by the Consultative Committee for Space Data Systems (CCSDS) as an experimental standard for "deep space" applications [2]. This set of codes, with code rates of 1/2, 2/3, and 4/5, offers higher rates than the previously standardized family of turbo codes. Three length options are available for each of the code rates. In our study, we considered rate-4/5 AR4JA codes of three different lengths, $(n, k) = (1280, 1024), (5120, 4096)$, and $(20480, 16384)$, all taken from the CCSDS standard.
   The AR4JA codes have a number of attractive features. Since they are constructed using protograph-based techniques [3], [4], [12], they provide an underlying regular structure that simplifies hardware implementation. They also allow for the introduction of degree-one variable nodes and the use of puncturing, both of which would be detrimental if applied to codes designed using "random" construction techniques. The AR4JA codes were created using two successive cyclic expansions of the base matrix, a procedure that helps to ensure a large minimum distance and a lower error floor. Two different optimization algorithms were used to select the expansions in order to prevent the clustering of short cycles [1]. Code designs were optimized using PEG and ACE techniques. The code designers also traded a small amount of threshold performance in the interest of further lowering error floors. We refer to these codes as "AR4JA" codes.
4) LDPC codes taken from MacKay's database of sparse graph codes [11].
   We used MacKay's constructions for LDPC codes with the following parameters:
   a) $n = 4095, r = 737, R = 0.82$, column weight 3, and no 4 cycles.
   b) $n = 4095, r = 738, R = 0.82$, column weight 4, and no 4 cycles.
   c) $n = 16383, r = 2130, R = 0.87$, column weight 3, and no 4 cycles.
   d) $n = 16383, r = 2131, R = 0.87$, column weight 4, and no 4 cycles.
   e) $n = 32000, r = 2240, R = 0.93$, column weight 3, and no 4 cycles.
   f) $n = 32000, r = 2241, R = 0.93$, column weight 4, and no 4 cycles.

   We will refer to the first, third, and fifth code as "DJCM-3" codes, and the second, fourth, and sixth code as "DJCM-4" codes.

In Fig. 14, we compare the performance of the BCH and LDPC codes described above. The BER results were computed for the first 100 iterations and then every 25th iteration thereafter, and the data were averaged over six TLC blocks. Comparisons were made between codes having approximately the same rate. For the rate-4/5, protograph-based PCC code, we did not see any errors. Likewise, for the AR4JA codes with

codeword lengths 5120 and 20480, we did not observe any errors. The length-1280 AR4JA code experienced errors after about 2500 program-erase cycles. The Gallager code failed at only one iteration after about 9000 cycles, while the rate-0.82 DCJM-3 and DCJM-4 codes began to fail after about 2600 and 9000 program-erase iterations, respectively; nevertheless, starting approximately at iteration 8500, the performance of all of these LDPC codes is superior to that of the BCH code. For higher-rate codes, the rate-0.9 Gallager code and the rate-0.87 DCJM-3 and DCJM-4 codes outperformed their rate-0.9 BCH counterpart. Finally, for yet higher rates near 0.925, we can see that the rate-0.93 DJCM-4 code outperformed the rate-0.93 DJCM-3 code, as well as the rate-0.925 BCH and Gallager codes.

VI. New ECC for TLC Flash

We next propose a new ECC design that reflects the cell-level error characteristics of our TLC flash memory chips. Our motivation comes from our previous work on the design of new codes for MLC flash, as reported in [15]. The MLC codes operate on pairs of pages which share the same group of cells, and they reflect the fact that the dominant error types that we observed involved a single-level cell-state change, specifically from 10 to 00 or from 00 to 01. Initially, we expected to see a similar behavior in TLC flash, with dominant errors corresponding to an increase in the cell state by one or, at most, two levels. Somewhat surprisingly, our measurements did not support this expectation. In fact, we saw frequent errors where the cell state changed from the lowest level to the highest level. However, there were some properties shared by the most frequently observed errors:

1) If a cell is in error, then with high probability only one out of its three bits is in error.
2) The probability of a bit being in error does not depend on the target cell state.

One possible explanation for this error behavior is as follows. The three bits in a cell are not programmed all at once, but rather one at a time. Therefore, an error in programming the first or second bit can cause the cell to change its state by more than one or two levels. For example, assume that we seek to program the three bits in a cell to (1,1,1) and that an error occurred while programming the MSB. Then, the CSB and LSB will be programmed based upon the erroneous measurement of the MSB as a 0 rather than as a 1. This results in the programming of the cell to its highest level, corresponding to bit values (0,1,1).

A. Code Construction

We will now show how the knowledge of the cell-state error behavior can be used to construct a new error correction coding scheme. The scheme works as follows. Let $\mathcal{C}_1$ be an $[n+r_1, n]$ t_1-error-correcting code over $GF(4)$ and let $\mathcal{C}_2$ be a binary $[n+r_2, n]$ t_2-error-correcting code, where $t_1 > t_2$. Assume that both codes are systematic. We will make use of a mapping, $\phi : \{0,1\}^3 \to GF(4)$, from binary triplets to elements of $GF(4)$, defined by:

$$\begin{array}{llll} \phi(1,1,1)=0, & \phi(1,1,0)=1, & \phi(1,0,0)=2, & \phi(1,0,1)=3, \\ \phi(0,0,0)=0, & \phi(0,0,1)=1, & \phi(0,1,1)=2, & \phi(0,1,0)=3. \end{array}$$

The mapping ϕ extends naturally to triplets of binary vectors of the same length.

Let $\boldsymbol{p}_{MSB} = (p_1, \ldots, p_n), \boldsymbol{p}_{CSB} = (c_1, \ldots, c_n), \boldsymbol{p}_{LSB} = (\ell_1, \ldots, \ell_n)$ be a group of MSB, CSB, LSB pages, respectively, sharing the same group of cells. The encoding procedure, depicted in Fig. 6, is as follows.

Encoding:

1) Calculate and store $\boldsymbol{s}_2$, the r_2 redundancy bits of $\mathcal{C}_2$ corresponding to the information page $\boldsymbol{p}_{MSB}$.
2) Calculate (without storing) $\boldsymbol{u} = \phi(\boldsymbol{p}_{MSB}, \boldsymbol{p}_{CSB}, \boldsymbol{p}_{LSB})$ over $GF(4)$.
3) Calculate and store $\boldsymbol{s}_1$, the r_1 redundancy symbols over $GF(4)$ of $\mathcal{C}_1$ corresponding to the information page $\boldsymbol{u}$.

For the decoding procedure, we let $\boldsymbol{p}'_{MSB} = (p'_1, \ldots, p'_n)$, $\boldsymbol{p}'_{CSB} = (c'_1, \ldots, c'_n), \boldsymbol{p}'_{LSB} = (\ell'_1, \ldots, \ell'_n)$ be the received MSB, CSB, LSB pages, respectively. We define a mapping $\psi : \{0,1\}^3 \times GF(4) \to \{0,1\}^3$ which takes a binary triplet (p', c', ℓ') and an error symbol e' in $GF(4)$ and returns a new binary triplet (p'', c'', ℓ'') as follows.

First, for all $(p', c', \ell') \in \{0,1\}^3$, we set $\psi((p', c', \ell'), 0) = (p', c', \ell')$. We then specify

$$\begin{array}{ll} \psi((1,1,1),1)=(1,0,1), & \psi((1,1,0),1)=(1,1,1), \\ \psi((1,0,0),1)=(1,1,0), & \psi((1,0,1),1)=(1,0,0), \\ \psi((1,1,1),2)=(0,1,1), & \psi((1,1,0),2)=(0,1,0), \\ \psi((1,0,0),2)=(0,0,0), & \psi((1,0,1),2)=(0,0,1), \\ \psi((1,1,1),3)=(1,1,0), & \psi((1,1,0),3)=(1,0,0), \\ \psi((1,0,0),3)=(1,0,1), & \psi((1,0,1),3)=(1,1,1). \end{array}$$

We then extend the mapping to the rest of the domain by demanding that, for $e' \neq 0$, if $\psi((p', c', \ell'), e') = (p'', c'', \ell'')$, then $\psi((\overline{p'}, \overline{c'}, \overline{\ell'}), e') = (\overline{p''}, \overline{c''}, \overline{\ell''})$. The role of the mapping ψ will be to return the bit values that were stored in a cell, assuming there was only a single bit error. The mapping ψ also extends naturally to a triplet of binary vectors and a vector over $GF(4)$, all of the same length. The decoding procedure, depicted in Fig. 7, is as follows.

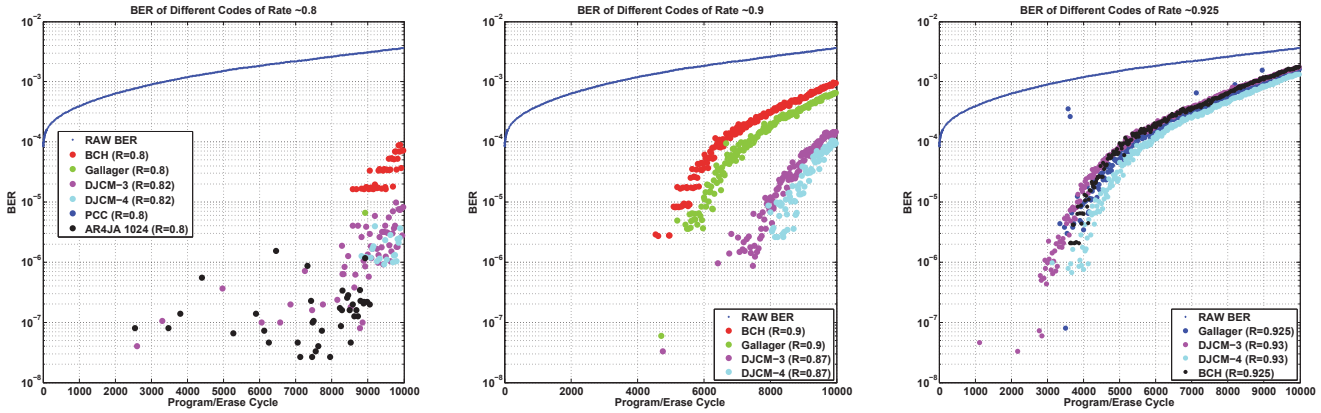
Decoding:

1) Calculate $\boldsymbol{u}' = \phi(\boldsymbol{p}'_{MSB}, \boldsymbol{p}'_{CSB}, \boldsymbol{p}'_{LSB})$ over $GF(4)$.
2) Using the r_1 redundancy symbols over $GF(4)$ and a decoder for the code $\mathcal{C}_1$, find up to t_1 symbol errors in $\boldsymbol{u}'$ and let $\boldsymbol{e}'$ denote the resulting error vector over $GF(4)$.
3) Calculate three binary vectors $\boldsymbol{p}''_{MSB}, \boldsymbol{p}''_{CSB}, \boldsymbol{p}''_{LSB}$ according to
$$(\boldsymbol{p}''_{MSB}, \boldsymbol{p}''_{CSB}, \boldsymbol{p}''_{LSB}) = \psi((\boldsymbol{p}'_{MSB}, \boldsymbol{p}'_{CSB}, \boldsymbol{p}'_{LSB}), \boldsymbol{e}').$$
4) Using the r_2 redundancy bits and a decoder for the code $\mathcal{C}_2$, find up to t_2 errors in $\boldsymbol{p}''_{MSB}$ and let $\boldsymbol{e}''$ denote the resulting binary error vector.
5) Return $(\boldsymbol{p}''_{MSB} + \boldsymbol{e}'', \boldsymbol{p}''_{CSB} + \boldsymbol{e}'', \boldsymbol{p}''_{LSB} + \boldsymbol{e}'')$ as the decoded triplet of pages.

B. Code Analysis

In order to analyze the error capability of this construction, we use the following definitions for different cell-errors.

1) A *type-one cell-error* is a cell-error where exactly one out of the three bits in the cell is in error.
2) A *type-two cell-error* is a cell-error where exactly two out of the three bits in the cell are in error.



(a) (b) (c)

Fig. 5. ECC performance comparisons for rates (a) $R \approx 0.8$, (b) $R \approx 0.9$, and (c) $R \approx 0.925$.

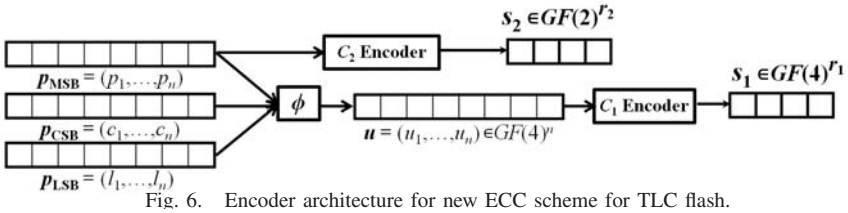


Fig. 6. Encoder architecture for new ECC scheme for TLC flash.

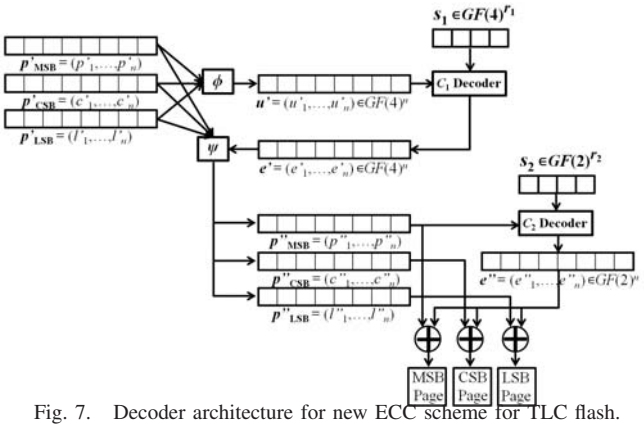


Fig. 7. Decoder architecture for new ECC scheme for TLC flash.

3) A *type-three cell-error* is a cell-error where all three bits in the cell are in error.

Let $\boldsymbol{p}_{MSB} = (p_1,\ldots,p_n), \boldsymbol{p}_{CSB} = (c_1,\ldots,c_n), \boldsymbol{p}_{LSB} = (\ell_1,\ldots,\ell_n)$ be the stored MSB, CSB, LSB page, respectively, which share the same group of cells, and $\boldsymbol{u} = \phi(\boldsymbol{p}_{MSB}, \boldsymbol{p}_{CSB}, \boldsymbol{p}_{LSB})$. Let $\boldsymbol{p}'_{MSB} = (p'_1,\ldots,p'_n), \boldsymbol{p}'_{CSB} = (c'_1,\ldots,c'_n), \boldsymbol{p}'_{LSB} = (\ell'_1,\ldots,\ell'_n)$ be the corresponding received MSB, CSB, LSB page, respectively, and $\boldsymbol{u}' = \phi(\boldsymbol{p}'_{MSB}, \boldsymbol{p}'_{CSB}, \boldsymbol{p}'_{LSB})$. Let $\boldsymbol{e}_{MSB}, \boldsymbol{e}_{CSB}, \boldsymbol{e}_{LSB}$ be the error vector in the MSB, CSB, LSB page, respectively, i.e.,

$$\boldsymbol{p}'_{MSB} = \boldsymbol{p}_{MSB} + \boldsymbol{e}_{MSB}, \boldsymbol{p}'_{CSB} = \boldsymbol{p}_{CSB} + \boldsymbol{e}_{CSB}, \boldsymbol{p}'_{LSB} = \boldsymbol{p}_{LSB} + \boldsymbol{e}_{LSB}.$$

Assume that the number of type-one cell-errors is e_1, the number of type-two cell-errors is e_2, and the number of type-three cell-errors is e_3. For a vector $\boldsymbol{v}$ (binary or non-binary) we denote by $w_H(\boldsymbol{v})$ its Hamming weight. First, we prove the following two lemmas.

Lemma 1. *The vectors $\boldsymbol{u}$ and $\boldsymbol{u}'$ satisfy the following property:*

$$w_H(\boldsymbol{u}' - \boldsymbol{u}) = e_1 + e_2.$$

Proof: For all $1 \leqslant i \leqslant n$, if there is no error in the i-th cell then $(p'_i, c'_i, \ell'_i) = (p_i, c_i, \ell_i)$ and

$$u_i = \phi(p_i, c_i, \ell_i) = \phi(p'_i, c'_i, \ell'_i) = u'_i.$$

If all three bits in the i-th cell are in error then $(p'_i, c'_i, \ell'_i) = (\overline{p_i}, \overline{c_i}, \overline{\ell_i})$ and

$$u_i = \phi(p_i, c_i, \ell_i) = \phi(\overline{p_i}, \overline{c_i}, \overline{\ell_i}) = \phi(p'_i, c'_i, \ell'_i) = u'_i.$$

If exactly one or two out of the three bits are in error then it is easy to verify that

$$u_i = \phi(p_i, c_i, \ell_i) \neq \phi(p'_i, c'_i, \ell'_i) = u'_i.$$

■

Assume that $\boldsymbol{e}' = \boldsymbol{u}' - \boldsymbol{u}$, and let

$$(\boldsymbol{p}''_{MSB}, \boldsymbol{p}''_{CSB}, \boldsymbol{p}''_{LSB}) = \psi((\boldsymbol{p}'_{MSB}, \boldsymbol{p}'_{CSB}, \boldsymbol{p}'_{LSB}), \boldsymbol{e}'),$$

and $\boldsymbol{e}'_{MSB}, \boldsymbol{e}'_{CSB}, \boldsymbol{e}'_{LSB}$ be the new error vectors in the MSB, CSB, LSB page, respectively, satisfying

$$\boldsymbol{p}''_{MSB} = \boldsymbol{p}_{MSB} + \boldsymbol{e}'_{MSB}, \boldsymbol{p}''_{CSB} = \boldsymbol{p}_{CSB} + \boldsymbol{e}'_{CSB}, \boldsymbol{p}''_{LSB} = \boldsymbol{p}_{LSB} + \boldsymbol{e}'_{LSB}. \quad (1)$$

Lemma 2. *The error vectors $\boldsymbol{e}'_{MSB}, \boldsymbol{e}'_{CSB}, \boldsymbol{e}'_{LSB}$ satisfy*

$$\boldsymbol{e}'_{MSB} = \boldsymbol{e}'_{CSB} = \boldsymbol{e}'_{LSB},$$

and

$$w_H(\boldsymbol{e}'_{MSB}) = w_H(\boldsymbol{e}'_{MSB}) = w_H(\boldsymbol{e}'_{MSB}) = e_2 + e_3.$$

Proof: For all $1 \leqslant i \leqslant n$, consider the following cases:

1) If there is no error in the i-th cell, then $u_i = u'_i$ and $e_{MSB,i} = e_{CSB,i} = e_{LSB,i} = 0$ and $e'_{MSB,i} = e'_{CSB,i} = e'_{LSB,i} = 0$.
2) If all three bits are in error in the i-th cell then $u_i = u'_i$ and $e_{MSB,i} = e_{CSB,i} = e_{LSB,i} = 1$ and $e'_{MSB,i} = e'_{CSB,i} = e'_{LSB,i} = 1$.
3) If only one bit is in error in the i-th cell, then $u_i \neq u'_i$; however, one can verify that $\psi((p'_i, c'_i, \ell'_i), e'_i) = (p_i, c_i, \ell_i)$, and so $e'_{MSB,i} = e'_{CSB,i} = e'_{LSB,i} = 0$.
4) If two bits are in error in the i-th cell, then $u_i \neq u'_i$; now we conclude that $\psi((p'_i, c'_i, \ell'_i), e'_i) = (\overline{p_i}, \overline{c_i}, \overline{\ell_i})$, and so $e'_{MSB,i} = e'_{CSB,i} = e'_{LSB,i} = 1$.

In each case, we find that for all $1 \leqslant i \leqslant n$, $e'_{MSB,i} = e'_{CSB,i} = e'_{LSB,i}$ and thus $\boldsymbol{e}'_{MSB} = \boldsymbol{e}'_{CSB} = \boldsymbol{e}'_{LSB}$. Also, $e'_{MSB,i} = e'_{CSB,i} = e'_{LSB,i} = 1$ if and only if there are two or three bits in error in the cell, and thus $w_H(\boldsymbol{e}'_{MSB}) = w_H(\boldsymbol{e}'_{MSB}) = w_H(\boldsymbol{e}'_{MSB}) = e_2 + e_3$. ■

Next, we verify the error-correction capability of the new code construction.

Theorem 3. *Let $\mathcal{C}_1$ be an $[n + r_1, n]$ t_1-error-correcting code over $GF(4)$ and let $\mathcal{C}_2$ be a binary $[n + r_2, n]$ t_2-error-correcting code. If $e_1 + e_2 \leqslant t_1$ and $e_2 + e_3 \leqslant t_2$, then the decoding procedure described above successfully decodes the correct value of the three pages.*

Proof: According to Lemma 1, if $e_1 + e_2 \leqslant t_1$, then $w_H(\boldsymbol{u}' - \boldsymbol{u}) \leqslant t_1$. Therefore, Step 2 of the decoding procedure succeeds and $\boldsymbol{e}' = \boldsymbol{u}' - \boldsymbol{u}$. Then, according to Lemma 2, $w_H(\boldsymbol{e}'_{MSB}) = e_2 + e_3 \leqslant t_2$. Thus, Step 4 succeeds and $\boldsymbol{e}'' = \boldsymbol{p}''_{MSB} - \boldsymbol{p}_{MSB} = \boldsymbol{e}'_{MSB}$, from which we recover the correct right value of the MSB page. Finally, again invoking Lemma 2, we deduce that that $\boldsymbol{e}'' = \boldsymbol{e}'_{CSB} = \boldsymbol{e}'_{LSB}$, from which we recover the correct values of the CSB and LSB pages. ■

C. Performance Evaluation

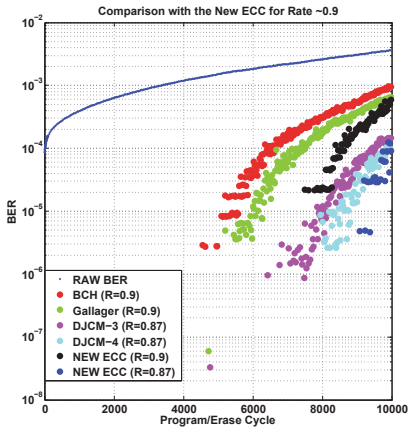
We compared the BER performance of the new ECC scheme to that of the BCH and LDPC codes considered above. We assumed a page size of 8KB, i.e., 2^{16} bits. In order to specify a rate for the new ECC scheme, we assumed that the code $\mathcal{C}_2$ is a binary t_2-error correcting BCH code with redundancy $t_2 \log_2(n) = 16t_2$, where we have used the fact that, with the page size specified above, $\log_2(n) = 16$. For the code $\mathcal{C}_1$, we note that for a t_1-error correcting code over $GF(4)$, the minimum redundancy according to the sphere packing bound is on the order of $t_2 \log_2(3n) = t_2 \log_2(n) + t_2 \log_2(3)$. Since efficient constructions of such codes are not readily found, we assumed conservatively that $\mathcal{C}_1$ has redundancy $t_2 \log_2(n) + 2t_2 = 18t_2$. Using this redundancy calculation, we evaluated codes for rates 0.9 and 0.925. The results are shown in Fig. 17, where we can see that, for both rates, the new ECC scheme outperforms the BCH code as well as the Gallager, DJCM-3, and DJCM-4 LDPC codes.

VII. Summary and Conclusions

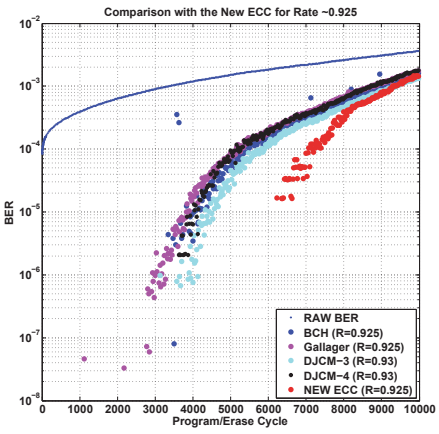
In this paper, we extended our previous study of error characteristics in SLC and MLC chips [15] to 3-bit per cell (TLC) flash memories. After analyzing the BER at the block-level and page-level, we measured the improvement in BER that could be achieved with a reduction in the nominal chip storage capacity, as obtained by programming only the most-significant-bit (MSB) page or both the MSB and central-significant-bit (CSB) pages. We then compared the BER performance of BCH codes and several families of LDPC codes. We also introduced and evaluated a novel error-correction scheme that takes into account our characterization of the dominant cell-level errors found in TLC devices. The new code operates on all three pages stored in a physical row of TLC cells. Our results provided valuable insights into the advantages offered by certain classes of high-rate LDPC codes, as well as by our proposed cell-level coding technique.

Acknowledgment

The authors wish to thank Aman Bhatia, Brian K. Butler, Aravind Iyengar, and Minghai Qin for their help in processing the error measurement results and, in particular, for the LDPC code performance simulations. The authors also wish to thank Jeff Ohshima and Hironori Uchikawa of Toshiba Corporation for their support of this work.



(a)



(b)

Fig. 8. BER performance of BCH codes, selected LDPC codes, and the new ECC scheme for TLC flash at rates (a) $R \approx 0.9$, and (b) $R \approx 0.925$.

References

[1] K. Andrews, et. al., "The Development of Turbo and LDPC Codes for Deep-Space Applications," *Proceedings of the IEEE*, November 2007.

[2] CCSDS, "Low Density Parity Check Codes for use in Near-Earth and Deep Space Applications," *Experimental Specification CCSDS 131.1-O-2*, September 2007.

[3] D. Divsalar, et. al., "Construction of protograph LDPC codes with linear minimum distance," *Proc. IEEE Int. Symp. Inf. Theory*, June 2006.

[4] D. Divsalar, et. al., "Capacity-Approaching Protograph Codes," *IEEE J. Selected Areas in Commun.*, vol. 27, pp. 876–888, August 2009.

[5] R.G. Gallager, "Low-density parity-check codes," *IRE Trans. Inform. Theory*, vol. 8, no. 1, pp. 21–28, January 1962.

[6] S. Gregori, A. Cabrini, O. Khouri, and G. Torelli, "On-chip error correcting techniques for new-generation flash memories," *Proceedings of The IEEE*, vol. 91, no. 4, pp. 602–616, April 2003.

[7] L. Grupp, A. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P.H. Siegel, and J.K. Wolf, "Characterizing flash memory : anomalies, observations, and applications," *MICRO-42*, pp. 24–33, December 2009.

[8] X.-Y. Hu, et. al., "Regular and irregular progressive edge-growth Tanner graphs," *IEEE Trans. Inf. Theory*, vol. 51, no. 1, pp. 386-398, Jan. 2005.

[9] JEDEC, Preconditioning of Plastic Surface Mount Devices Prior to Reliability Testing.

[10] S. Kudekar, T. Richardson, and R.L. Urbanke, "Threshold saturation via spatial coupling: Why convolutional LDPC ensembles perform so well over the BEC," *CoRR, vol. abs/1001.1826*, 2010.

[11] D.J.C. MacKay, "Encyclopedia of sparse graph codes," http://www.inference.phy.cam.ac.uk/mackay/codes/data.html.

[12] J. Thorpe, "Low Density Parity Check (LDPC) Codes Constructed from Protographs," *JPL INP Progress Report*, pp. 42–154, August 15, 2003.

[13] T. Tian, et. al., "Selective avoidance of cycles in irregular LDPC code construction," *IEEE Trans. Comm.*, vol. 52, pp. 1242-1247, August 2004.

[14] H. Weingarten, "New strategies to overcome 3bpc challenges," *Flash Memory Summit*, Santa Clara, August 2010.

[15] E. Yaakobi, et. al., "Error characterization and coding schemes for flash memories," in *Proc. Workshop on the Application of Communication Theory to Emerging Memory Technologies*, Miami, Florida, Dec. 2010.