Graded Bit-Error-Correcting Codes With Applications to Flash Memory

Ryan Gabrys, Eitan Yaakobi, *Member, IEEE*, and Lara Dolecek, *Member, IEEE*

***Abstract*—Flash memory is a promising new storage technology. Supported by empirical data collected from a Flash memory device, we propose a class of codes that exploits the asymmetric nature of the error patterns in a Flash device using tensor product operations. We call these codes graded bit-error-correcting codes. As demonstrated on the data collected from a Flash chip, these codes significantly delay the onset of errors and therefore have the potential to prolong the lifetime of the memory device.**

***Index Terms*—Coding theory, error-correcting codes, flash memory, tensor product codes.**

I. Introduction

FLASH memory devices can be found almost everywhere today. They are lighter, faster, and more shock resistant than traditional magnetic hard drives. As this technology scales and the storage density increases, data errors become more prevalent, making error correction coding critical for maintaining data integrity.

The storage density of a Flash memory device is dependent on the number of discrete voltage levels the floating gate cell is capable of representing. In early generations, every memory cell could represent two voltage levels and thus store a single bit. The demand for increased storage capacity has created the need to store more than a single bit per cell by simply representing more than two voltage levels. In this work, we follow the commonly adopted nomenclature and assume that multiple level cell chips store multiple bits per cell, and that in particular triple level cell (TLC) chips store three bits per cell.

Recently, the subject of error-correction coding for Flash memory has received significant attention. In [18], trellis-coded modulation techniques were applied to Flash memory. In [13], the use of LDPC codes was investigated, and in [19], it was found that using soft information from multiple reads in the LDPC decoder lowered the error rate. In [9], algebraic error-correction codes were used for rewriting as well as for correcting errors. In [2], [5], [6], and [12], codes that correct limited magnitude asymmetric errors were constructed. In [23], this model was extended to correct graded error patterns. In [17], constructions were given for single error-correcting codes that can correct limited magnitude errors in two directions. In [25], a different error model was considered where the likelihood of an error occurring was directly related to the value of the cell being programmed. The problem was to construct codes that maximized the size of a codebook given some fixed tolerable error probability. In [11], a novel method of encoding information was introduced that reduced the occurrence of errors during programming.

The error model in this work is motivated by data collected from a TLC Flash device. As observed in [24], if the information from each Flash cell is interpreted as a triple-bit word, then the errors (referred to as graded bit errors) largely but not exclusively cause only a single bit in each word to change. From this observation, we suggest the use of a class of codes derived from tensor product codes [22] in the context of Flash memory. We refer to this class of codes as *graded bit-error-correcting codes*. The contribution of this work is to generalize the result of [24] to produce code constructions that correct errors that mostly have only a small number of bits in error for each cell error. In fact, some of the proposed codes indeed end up having the same algebraic structure as generalized tensor product codes (cf. [10]). The novelty of this work is to show that for certain parameters of the constituent codes, such constructions can correct graded bit errors.

Tensor product codes were first introduced in [22] and were generalized to produce efficient binary codes in [10]. In [20], these constructions were revisited and an efficient method of encoding was provided. More recently, tensor product codes were used in the context of magnetic recording [3], [4]. In a concatenated coding scheme, the use of a tensor product parity code as the inner code was shown to offer the performance advantages of a short length parity code but without the associated rate penalty. In [1], tensor product codes were used in conjunction with soft iterative decoding methods to manage the size of the syndrome table. In this work, a new type of generalized tensor product codes, the graded bit-error-correcting codes, is developed. These codes are demonstrated to correct the errors that occur within a TLC Flash device. In particular, generalized tensor product codes are shown to delay the onset of errors longer than conventional coding schemes. Delaying the onset of errors is significant since the device can potentially be used for a longer period of time.

In contrast to conventional symmetric error-correcting codes (e.g., [15, Chs. 5 and 8]) that in general correct symbol errors of arbitrary magnitudes, the error correction capability of the

Manuscript received May 04, 2012; revised November 01, 2012; accepted November 29, 2012. Date of publication January 04, 2013; date of current version March 13, 2013. This work was supported in part by the SMART scholarship, in part by the National Science Foundation under Grants CCF-1029030 and CCF-1150212, in part by the International Sephardic Education Foundation, and in part by the Lester Deutsch Fellowship. This paper was presented in part at the 2012 IEEE International Symposium on Information Theory.

R. Gabrys and L. Dolecek are with the University of California at Los Angeles, Los Angeles, CA 90095 USA (e-mail: rgabrys@ucla.edu; dolecek@ee.ucla.edu).

E. Yaakobi is with the California Institute of Technology, Pasadena, CA 91106 USA (e-mail: yaakobi@caltech.edu).

Communicated by O. Milenkovic, Associate Editor for Coding Theory.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2012.2234207

TABLE I
MAPPING BETWEEN VOLTAGE LEVELS AND TRIPLE-BIT WORDS

Voltage Level	Triple-bit Word
0	111
1	110
2	100
3	101
4	001
5	000
6	010
7	011

proposed graded bit-error-correcting codes is developed only in terms of certain error patterns. This restriction offers performance advantage over symmetric error-correcting codes for applications, such as Flash, where the errors occur in an asymmetric fashion.

This paper is organized as follows. In Section II, the data collected from a TLC Flash chip are summarized. In Section III, the error model, motivated by the experimental data, is proposed. In Section IV, code constructions for this model are given. Section V analyzes the performance of these new codes in terms of their redundancy. In Section VI, these constructions are shown through simulation to delay the onset of errors within the memory device longer than traditionally used error correction codes. Section VII concludes this paper.

II. STRUCTURE AND ERROR CHARACTERIZATION OF TLC FLASH

In this section, we report on the observed errors measured from a TLC chip provided by an anonymous vendor. Before analyzing the results, let us briefly review the typical structure of a TLC chip.

A TLC chip is divided into multiple planes. Each plane is divided into a set of blocks and these blocks are further decomposed into pages. For the particular TLC chip measured, there are 384 pages within a block and 8 KB within a page. The eight discrete voltage levels from the cell are represented as a triple-bit word. We refer to the first bit in the word as the most significant bit (MSB), the second bit in the word as the center significant bit (CSB), and the third bit in the word as the least significant bit (LSB). The mapping between voltage levels and triple-bit words is depicted in Table I. (For more details on the structure of a TLC chip, see [24].)

The errors were measured from 16 blocks evenly divided across two planes. In a manner analogous to [8], the following testing procedure was repeatedly performed. On the first cycle of every 100 program/erase (P/E) cycles, a block was erased, and random data were then written and finally read back for errors. On the other 99 cycles, the block was simply erased and every page was programmed to simulate the aging of the device.

In Fig. 1, the bit error rate (BER) is illustrated for the TLC chip tested over the course of its lifetime. It can be seen that over time, the BER increases dramatically. Furthermore, the increase in error rate varies across the LSB, CSB, and MSB. The "symbol error rate" curve refers to the symbol error rate when each cell is represented as a symbol over $GF(8)$, and LSB, CSB and MSB curves correspond to the respective bitwise errors.

Fig. 1 plots the result of measurements taken from 3.3×10^7 cells monitored across 5000 P/E cycles. The dominant trend from Fig. 1 is that the "symbol error rate" appears to be roughly

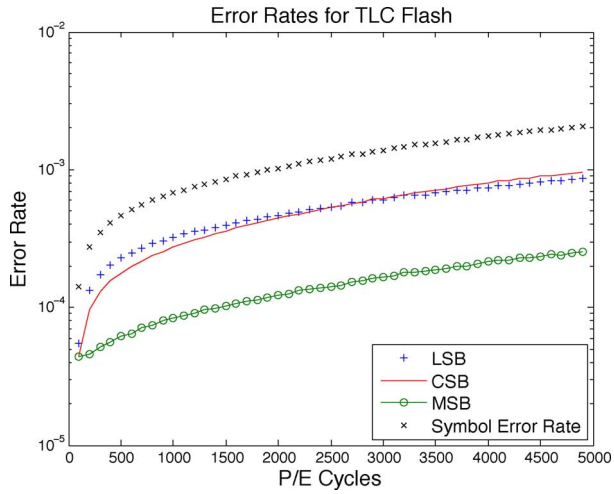

Fig. 1. Error rates measured from a TLC Flash device.

TABLE II
PROMINENT ERROR PATTERNS OBSERVED

Programmed state	Errored state	Percentage of errors
000	010	0.2467
000	001	0.2444
111	101	0.0820
111	110	0.0807
000	100	0.0669
011	001	0.0556
100	110	0.0550
011	010	0.0547
100	101	0.0540
111	011	0.0217

TABLE III
FLASH CELL ERROR PATTERNS

Number of bits in Flash cell that err	Percentage of errors
1	0.9617
2	0.0314
3	0.0069

the sum of the individual BERs of the MSB, CSB, and LSB. This suggests that whenever a cell error occurs, with high probability only one of the three bits in the cell errs. More specifically, as shown in Table II, most of the bit errors in each cell followed a certain pattern. In each row, erroneous bits are marked in red. Table II also offers an insight on why the BER curves of the LSB and CSB in Fig. 1 are close to each other: every dominant error of the CSB has an equivalent one of the LSB with almost the same percentage. Upon closer inspection, Table III shows that 96.17% of cell errors measured from across the device's lifetime only had a single bit in error.

We note that this distribution of error patterns is a result of the special programming property of the bits where the three bits are not programmed all at once but rather one at a time. This special programming property is due to organization of information in a Flash device: every Flash cell is spread across multiple pages and data are accessed/modified one page at a time. Suppose the bits are programmed in the following order: first MSB, then CSB, and finally LSB. Then, a programming error in the MSB causes the LSB and CSB to base their program on the erroneous measurement of the MSB. Thus, if the value $(1, 1, 1)$ is written and the MSB is programmed as a 0, then the CSB and LSB will

read 0 and will program the cell to its highest level, resulting in the state $(0, 1, 1)$. For more information regarding this property and the mappings of cell values to voltages, see [24].

The observed distribution is the key motivation for the proposed code constructions that correct a large number of cell errors with one bit in error and a smaller number of cell errors with more than one bit in error. Note that this error model is considerably different than the one of asymmetric limited magnitude errors, studied in previous works, e.g., [2] and [6]. That is, the observed errors are not limited in their magnitude as one may expect to see. Interestingly, as shown in the last row of Table II, we observed frequent errors from the lowest level to highest level (cf. Table I). We note that in all these dominant errors in Table II, only one bit was in error.

III. Model and Definitions

In this section, the relevant error models as well as code definitions are introduced.

Definition 1: A linear code $\mathcal{C}$ of length n and dimension k over an alphabet of size q that can correct t or less errors is referred to as an $[n, k, t]_q$ code.

In this paper, the *redundancy* of a code $\mathcal{C}$ represents the number of parity bits if the code is binary, and, more generally, the number of parity symbols if the code is nonbinary. All codes considered in this work have alphabets whose cardinality is $q = 2^m$ where m is some positive integer. Each cell can take on 2^m possible values and is displayed as an m-bit vector. Thus, a word of length n is represented as a length-nm binary vector where bits $mi, \ldots, m(i+1)-1$ represent the ith cell for $1 \leq i \leq n$.

Accordingly, every cell error is represented as a length-m vector $\boldsymbol{e}_i$. For a fixed ℓ, if $wt(\boldsymbol{e}_i) \leq \ell$, then such an error is called an *ℓ-bit cell error*, where the Hamming weight of a vector $\boldsymbol{x}$ is denoted by $wt(\boldsymbol{x})$. Motivated by the nature of the errors observed, it is useful to define the following class of error vectors and codes.

Definition 2: Given the positive integers t and ℓ, an error vector $\boldsymbol{e} = (\boldsymbol{e}_1, \boldsymbol{e}_1, \ldots, \boldsymbol{e}_n)$ over $(GF(2)^m)^n$ is called a *$[t; \ell]_{2^m}$-bit-error vector* if the following holds:

1) $wt(\boldsymbol{e}) = |\{i : \boldsymbol{e}_i \neq \boldsymbol{0}\}| \leq t$; and
2) $\forall i$, $wt(\boldsymbol{e}_i) \leq \ell$.

Definition 3: A 2^m-ary linear code $\mathcal{C}$ that can correct every $[t; \ell]_{2^m}$-bit-error vector is called a *$[t; \ell]_{2^m}$-bit-error-correcting code*.

From the data collected from the TLC flash device, it was observed that while most cell errors suffered a single bit error, a small number of cells had double or triple bit errors. Therefore, to correct all observed errors, it is useful to define the following refined error vectors and corresponding codes.

Definition 4: Let $0 < \ell_1 < \ell_2 \leq m$, $t_1, t_2 > 0$. Then, a vector $\boldsymbol{e} = (\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_n)$ over $(GF(2)^m)^n$ is called a *$[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error vector* if the following holds:

1) $wt(\boldsymbol{e}) = |\{i : \boldsymbol{e}_i \neq \boldsymbol{0}\}| \leq t_1 + t_2$;
2) $\forall i$, $wt(\boldsymbol{e}_i) \leq \ell_2$; and
3) $|\{i : wt(\boldsymbol{e}_i) > \ell_1\}| \leq t_2$.

The vector $(000\ \ 001\ \ 000\ \ 011\ \ 000)$ is an example of a $[1, 1; 1, 2]_{2^3}$-bit-error vector of length 5, since there are at most $1 + 1 = 2$ cells of nonzero Hamming weight (here, these are the second and fourth cell), the maximum weight per cell is 2 (achieved in the fourth cell), and there is at most one cell of weight more than one (again achieved in the fourth cell).

Definition 5: Let $0 < \ell_1 < \ell_2 \leq m$, $t_1, t_2 > 0$. Then, a 2^m-ary code $\mathcal{C}$ is said to be a *$[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-correcting code* if it can correct every $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error vector.

We collectively refer to codes introduced in Definition 5 as *graded bit-error-correcting codes*.

We complete the section with the following definition useful in determining the parity-check matrices of (graded) bit-error-correcting codes.

Definition 6: Let $A \in GF(q)^{m \times n}$ and $B \in GF(q)^{p \times r}$. Then, the tensor product of A and B is defined as the matrix

$$A \otimes B = \begin{pmatrix} a_{1,1}B & \ldots & a_{1,n}B \\ \vdots & \ddots & \vdots \\ a_{m,1}B & \ldots & a_{m,n}B \end{pmatrix} \in GF(q)^{mp \times nr}.$$

Note that the rank of the tensor product can be expressed in terms of the ranks of constituent matrices as

$$\text{rank}(A \otimes B) = \text{rank}(A) \cdot \text{rank}(B).$$

IV. Code Constructions

In this section, code constructions are given for (graded) bit-error-correcting codes. The section begins by revisiting a result from [22] that can be used to create $[t; \ell]_{2^m}$-bit-error-correcting codes. This idea is extended to create $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-correcting codes.

A. Tensor Product Codes

We start by presenting a construction of a type of tensor product codes which we also refer to as a $[t; \ell]_{2^m}$-bit-error-correcting code.

Construction A: (cf. [22]) Let $\mathcal{C}_1$ be an $[m, k_1, \ell]_2$ code with a parity-check matrix H_1. Let $\mathcal{C}_2$ be an $[n, k_2, t]_{2^{m-k_1}}$ code with a parity-check matrix H_2. Then, the code $\mathcal{C}_A$ with the parity-check matrix

$$H_A = (H_2 \otimes H_1) \tag{1}$$

is a $[t; \ell]_{2^m}$-bit-error-correcting code of length n.

Remark 1: Note that the parity-check matrix $H_A = (H_2 \otimes H_1)$ corresponds to a binary code of length nm. This matrix can also be converted into a parity-check matrix of a code of length n over $GF(2^m)$ by simply partitioning each row of the matrix H_A into consecutive groups of m bits each. Each such group then corresponds to an element in $GF(2^m)$.

We first provide an example illustrating how these operations are performed.

Example 1: (cf. [20]) Let H_2 be the following parity-check matrix for a $[5,3,1]_4$ code where α is a primitive element from $GF(4)$

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

Let H_1 be a Hamming code of length 3 with the following parity-check matrix:

$$H_1 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}.$$

Representing the elements of $GF(4)$ as $\alpha^0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \alpha^1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \alpha^2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, and $0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, we have

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

Using the same symbol-to-binary vector mapping, we can write H_A in the binary representation as

$$H_A = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}.$$

Note that since the parity-check matrix of the code $\mathcal{C}_A$ is the tensor product of the matrices H_1 and H_2, and $\text{rank}(H_2 \otimes H_1) = \text{rank}(H_2) \cdot \text{rank}(H_1)$, we get that the redundancy of the code $\mathcal{C}_A$ is $r_1 r_2$, where $r_1 = m - k_1$ and $r_2 = n - k_2$. While the correctness of the error-correction capability was proved in [22], for completeness and in the interest of the subsequent discussion, let us describe here a decoder for the code $\mathcal{C}_A$.

Suppose $\boldsymbol{c} \in \mathcal{C}_A$, where $\boldsymbol{c} = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n) \in (GF(2)^m)^n$. Then

$$\begin{aligned} H_A \cdot \boldsymbol{c}^T &= (H_2 \otimes H_1) \cdot \boldsymbol{c}^T \\ &= \begin{pmatrix} h_{1,1}H_1 & \ldots & h_{1,n}H_1 \\ \vdots & \ddots & \vdots \\ h_{r_1,1}H_1 & \ldots & h_{r_1,n}H_1 \end{pmatrix} \cdot \boldsymbol{c}^T \\ &= \begin{pmatrix} h_{1,1} & \ldots & h_{1,n} \\ \vdots & \ddots & \vdots \\ h_{r_1,1} & \ldots & h_{r_1,n} \end{pmatrix} \cdot \begin{pmatrix} H_1 \cdot {\boldsymbol{c}_1}^T \\ \vdots \\ H_1 \cdot {\boldsymbol{c}_n}^T \end{pmatrix} \end{aligned}$$

where $h_{i,j}$ represents the symbol in position row i, column j of H_2. Thus, $\boldsymbol{c} \in \mathcal{C}_A$ if and only if $(H_1 \cdot {\boldsymbol{c}_1}^T, \ldots, H_1 \cdot {\boldsymbol{c}_n}^T) \in \mathcal{C}_2$. The code $\mathcal{C}_A$ can be expressed as follows:

$$\begin{aligned} \mathcal{C}_A = \{\boldsymbol{c} = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n) \in (GF(2)^m)^n : \\ (H_1 \cdot \boldsymbol{c}_1^T, \ldots, H_1 \cdot \boldsymbol{c}_n^T) \in \mathcal{C}_2\}. \end{aligned}$$

Let

$$\mathcal{D}_1 : \{0,1\}^{r_1} \to \{0,1\}^m, \ \mathcal{D}_2 : (\{0,1\}^{r_1})^{r_2} \to (\{0,1\}^{r_1})^n$$

be the decoders of the codes $\mathcal{C}_1, \mathcal{C}_2$, respectively. Here, and henceforth we assume that the input to each constituent decoder is the syndrome of the received vector and the output is the detected error vector. We note that the input to the constituent decoders will be the corresponding syndrome, whereas the input to the decoder of Construction A as well as the upcoming constructions will be the erroneous word. For the ease of the code presentation, we use both descriptions of the decoder input. We also assume that if the code can correct t errors, then the weight of the output error vector is at most t. If the decoder finds an error vector of weight greater than t, then the all-zero vector is returned as the output.

The decoder $\mathcal{D}_A : (\{0,1\}^m)^n \to (\{0,1\}^m)^n$ of the code $\mathcal{C}_A$ gets as an input a word of the form $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$, where $\boldsymbol{c} \in \mathcal{C}_A$ and $\boldsymbol{e} \in (GF(2)^m)^n$ is a $[t;\ell]_{2^m}$-bit-error vector. The output of the decoder is the estimate of the error vector, obtained in two steps. First, $\mathcal{D}_2$ produces a set of syndromes $\boldsymbol{s}_i$, $1 \le i \le n$, based on the received string $\boldsymbol{y}$. Each of the syndromes $\boldsymbol{s}_i$ serves as the input to $\mathcal{D}_1$ to collectively produce the estimate of the error vector $\widehat{\boldsymbol{e}}$. The overall decoding algorithm can be described in these two steps.

1) $\mathcal{D}_2(H_2 \cdot (H_1 \cdot \boldsymbol{y}_1^T, \ldots, H_1 \cdot \boldsymbol{y}_n^T)^T) = (\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n)$.
2) $\widehat{\boldsymbol{e}} = (\mathcal{D}_1(\boldsymbol{s}_1), \ldots, \mathcal{D}_1(\boldsymbol{s}_n))$.

Lemma 1: Assume that $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$, where $\boldsymbol{c} \in \mathcal{C}_A$ and $\boldsymbol{e} \in (GF(2)^m)^n$ is a $[t;\ell]_{2^m}$-bit-error vector. The decoder output satisfies $\mathcal{D}_A(\boldsymbol{y}) = \widehat{\boldsymbol{e}} = \boldsymbol{e}$.

Proof: According to the definition of the code $\mathcal{C}_A$, we have $(H_1 \cdot \boldsymbol{c}_1^T, \ldots, H_1 \cdot \boldsymbol{c}_n^T) \in \mathcal{C}_2$ and we can write

$$\begin{aligned} &(H_1 \cdot \boldsymbol{y}_1^T, \ldots, H_1 \cdot \boldsymbol{y}_n^T) \\ &= (H_1 \cdot \boldsymbol{c}_1^T, \ldots, H_1 \cdot \boldsymbol{c}_n^T) + (H_1 \cdot \boldsymbol{e}_1^T, \ldots, H_1 \cdot \boldsymbol{e}_n^T). \end{aligned}$$

The vector $(H_1 \cdot \boldsymbol{e}_1^T, \ldots, H_1 \cdot \boldsymbol{e}_n^T)$ has weight at most t and since $\mathcal{C}_2$ can correct t errors, we get that

$$(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) = (H_1 \cdot \boldsymbol{e}_1^T, \ldots, H_1 \cdot \boldsymbol{e}_n^T).$$

Next, for every $1 \le i \le n$

$$H_1 \cdot \boldsymbol{y}_i^T = H_1 \cdot (\boldsymbol{c}_i^T + \boldsymbol{e}_i^T) = H_1 \cdot \boldsymbol{c}_i^T + H_1 \cdot \boldsymbol{e}_i^T$$

and since $\boldsymbol{s}_i = H_1 \cdot \boldsymbol{e}_i^T$ and the weight of $\boldsymbol{e}_i$ is at most ℓ, we get that $\mathcal{D}_1(\boldsymbol{s}_i) = \boldsymbol{e}_i$, that is, $\boldsymbol{e} = \widehat{\boldsymbol{e}}$. ■

B. Graded Bit-Error-Correcting Codes

The codes given in Construction A correct error patterns according to the maximum number of bit errors in every cell (or within an m-bit symbol). Recall that in the TLC data it was observed that while most cells suffer a small number of bit errors, a few cell errors may in fact have a larger number of bit errors. This observation motivates the following construction of $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-correcting codes (also referred to as graded bit-error-correcting codes).

Construction B: Let $\mathcal{C}_1$ be an $[m, k, \ell_2]_2$ code with a parity-check matrix H_1.

1) Let $r = m - k$ be such that the following holds.
   a) There exists $0 \le r' < r$ such that the matrix H_1' comprised of the first r' rows of H_1 is a parity-check matrix for an $[m, m - r', \ell_1]_2$ code $\mathcal{C}_1'$.

b) The matrix H_1'' is an $r'' \times m$ matrix consisting of the last r'' rows of H_1, where $r'' = r - r'$.

2) The matrix H_2 is a parity-check matrix for an $[n, k_2, t_1 + t_2]_{2^{r'}}$ code $\mathcal{C}_2$, and $r_2 = n - k_2$.

3) The matrix H_3 is a parity-check matrix for an $[n, k_3, t_2]_{2^{r''}}$ code $\mathcal{C}_3$, and $r_3 = n - k_3$.

Then, a parity-check matrix for a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-correcting code $\mathcal{C}_B$ of length n is

$$H_B = \begin{pmatrix} H_2 \otimes H_1' \\ H_3 \otimes H_1'' \end{pmatrix}. \tag{2}$$

Remark 2: The parity-check matrix H_1 of the code $\mathcal{C}_1$ that corrects ℓ_2 errors needs to satisfy the property that it can be decomposed into two matrices H_1' and H_1'', where the first matrix is a parity-check matrix of an ℓ_1-error-correcting code $\mathcal{C}_1'$. We note this requirement is not hard to satisfy as many codes can follow this structure, and in particular Bose–Chaudhuri–Hocquenghem (BCH) codes.

Remark 3: The resulting parity-check matrix in Construction B has the same structure as a parity-check matrix for the generalized tensor product code in [10]. The focus in [10] was to create rate-efficient binary codes with a guaranteed minimum distance and low decoding complexity. Here, the goal is to correct specific error patterns not captured by traditional definitions of minimum distance.

Before proving the error correction capability of $\mathcal{C}_B$, we provide an illustrative example.

Example 2: Suppose $\mathcal{C}_1$ is a triple error-correcting code with a parity-check matrix $H_1 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$. Let $r' = 2$ so that $H_1' = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$ is a parity-check matrix for a $[3, 1, 1]_2$ Hamming code and $H_1'' = (\,1\,1\,1\,)$. Let $\mathcal{C}_2$ be a $[15, 9, 2]_4$ code with a parity-check matrix H_2. Furthermore, let $\mathcal{C}_3$ be a $[15, 11, 1]_2$ Hamming code with a parity-check matrix H_3. Then, using Construction B, the code $\mathcal{C}_B$ has a parity-check matrix

$$H_B = \begin{pmatrix} H_2 \otimes H_1' \\ H_3 \otimes H_1'' \end{pmatrix}. \tag{3}$$

H_B is a parity-check matrix for a $[1, 1; 1, 3]_{2^3}$-bit-error-correcting code. Let α be the primitive element of $GF(4)$. Using the field element representation from Example 1, the matrix H_1' can now be written as $H_1'' = (\,\alpha^0\ \alpha\ \alpha^2\,)$, and the operation $H_2 \otimes H_1'$ can be performed over $GF(4)$. The operation $H_3 \otimes H_1''$ is defined over $GF(2)$. We remark that the particular choice of $\mathcal{C}_1$ in this example results in the same code that was previously proposed in [24].

Note that $\boldsymbol{c} \in \mathcal{C}_B$ if and only if

$$\mathbf{0} = H_B \cdot \boldsymbol{c}^T = \begin{pmatrix} H_2 \otimes H_1' \\ H_3 \otimes H_1'' \end{pmatrix} \cdot \boldsymbol{c}^T = \begin{pmatrix} H_2 \cdot (\,H_1' \cdot \boldsymbol{c}_1{}^T, \ldots, H_1' \cdot \boldsymbol{c}_n{}^T\,)^T \\ H_3 \cdot (\,H_1'' \cdot \boldsymbol{c}_1{}^T, \ldots, H_1'' \cdot \boldsymbol{c}_n{}^T\,)^T \end{pmatrix}.$$

Hence, the code $\mathcal{C}_B$ can be expressed as

$$\mathcal{C}_B = \{\boldsymbol{c} = (\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n) \in (GF(2)^m)^n : (H_1' \cdot \boldsymbol{c}_1^T, \ldots, H_1' \cdot \boldsymbol{c}_n^T) \in \mathcal{C}_2, (H_1'' \cdot \boldsymbol{c}_1^T, \ldots, H_1'' \cdot \boldsymbol{c}_n^T) \in \mathcal{C}_3\}$$

and its redundancy is $r'r_2 + r''r_3$ (see [10]).

Let us denote by

$$\mathcal{D}_1 : \{0,1\}^r \to \{0,1\}^m,\ \mathcal{D}_1' : \{0,1\}^{r'} \to \{0,1\}^m$$
$$\mathcal{D}_2 : (\{0,1\}^{r'})^{r_2} \to (\{0,1\}^{r'})^n$$
$$\mathcal{D}_3 : (\{0,1\}^{r''})^{r_3} \to (\{0,1\}^{r''})^n$$

the decoders of the codes $\mathcal{C}_1, \mathcal{C}_1', \mathcal{C}_2$, and $\mathcal{C}_3$, respectively. As earlier, the input to each decoder is the syndrome and the output is the error vector whose weight is no greater than the guaranteed error-correction capability of the corresponding code.

Before presenting the decoding steps, let us explain the idea behind this construction and its decoding procedure. We start in a similar fashion as the decoder for Construction A, where at most t (now interpreted as $t_1 + t_2$) cell errors each of weight at most ℓ_1 are found. Clearly, it may not be possible to correct all cell errors this way since t_2 errors can have weight ℓ_2, and $\ell_2 > \ell_1$. If a cell error has weight at most ℓ_1, then it is corrected. Otherwise, it is miscorrected to a cell-error vector with weight at most $\ell_1 + \ell_2$ since the weight of each miscorrection has been restricted to be ℓ_1. This operation in turn guarantees that the new cell-error vector is not a codeword in $\mathcal{C}_1$, since the minimum distance of $\mathcal{C}_1$ is at least $2\ell_2 + 1$. Thus, the next step is to detect the cells which were miscorrected. For cell errors with more than ℓ_1 bits in error, the remaining part of the syndrome according to the code $\mathcal{C}_1$ is recovered. The decoder $\mathcal{D}_1$ is then used to recover the remaining errors.

The decoder $\mathcal{D}_B : (\{0,1\}^m)^n \to (\{0,1\}^m)^n$ of the code $\mathcal{C}_B$ gets as an input a word of the form $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$, where $\boldsymbol{c} \in \mathcal{C}_B$ and $\boldsymbol{e} \in (GF(2)^m)^n$ is a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error vector. The decoder output $\widehat{\boldsymbol{e}}$ is an estimate of the error vector $\boldsymbol{e}$, that is, $\mathcal{D}_B(\boldsymbol{y}) = \widehat{\boldsymbol{e}}$. The decoder $\mathcal{D}_B$ performs the following sequence of steps.

1) $\mathcal{D}_2(H_2 \cdot (H_1' \cdot \boldsymbol{y}_1^T, \ldots, H_1' \cdot \boldsymbol{y}_n^T)^T) = (\boldsymbol{s}_1^0, \ldots, \boldsymbol{s}_n^0)$.
2) $\widehat{\boldsymbol{e}}^* = (\mathcal{D}_1'(\boldsymbol{s}_1^0), \ldots, \mathcal{D}_1'(\boldsymbol{s}_n^0))$.
3) $\boldsymbol{y}' = \boldsymbol{y} + \widehat{\boldsymbol{e}}^*$.
4) $\mathcal{D}_2(H_2 \cdot (H_1' \cdot \boldsymbol{y}_1'^T, \ldots, H_1' \cdot \boldsymbol{y}_n'^T)^T) = (\boldsymbol{s}_1', \ldots, \boldsymbol{s}_n')$.
5) $\mathcal{D}_3(H_3 \cdot (H_1'' \cdot \boldsymbol{y}_1'^T, \ldots, H_1'' \cdot \boldsymbol{y}_n'^T)^T) = (\boldsymbol{s}_1'', \ldots, \boldsymbol{s}_n'')$.
6) Let $I = \{i : (\boldsymbol{s}_i', \boldsymbol{s}_i'') \neq (\mathbf{0}, \mathbf{0})\}$.
7) Let $\boldsymbol{y}''$ satisfy $\boldsymbol{y}_i'' = \boldsymbol{y}_i$ if $i \in I$ and $\boldsymbol{y}_i'' = \boldsymbol{y}_i'$ if $i \notin I$.
8) $\mathcal{D}_3(H_3 \cdot (H_1'' \cdot \boldsymbol{y}_1''^T, \ldots, H_1'' \cdot \boldsymbol{y}_n''^T)^T) = (\boldsymbol{s}_1^1, \ldots, \boldsymbol{s}_n^1)$.
9) $\widehat{\boldsymbol{e}} = (\widehat{\boldsymbol{e}}_1, \ldots, \widehat{\boldsymbol{e}}_n)$ where $\widehat{\boldsymbol{e}}_i = \widehat{\boldsymbol{e}}_i^*$ if $i \notin I$ and otherwise $\widehat{\boldsymbol{e}}_i = \mathcal{D}_1(\boldsymbol{s}_i^0, \boldsymbol{s}_i^1)$.

Theorem 1: Assume that $\boldsymbol{y} = \boldsymbol{c} + \boldsymbol{e}$, where $\boldsymbol{c} \in \mathcal{C}_B$ and $\boldsymbol{e} \in (GF(2)^m)^n$ is a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error vector. The decoder output satisfies $\mathcal{D}_B(\boldsymbol{y}) = \widehat{\boldsymbol{e}} = \boldsymbol{e}$.

Proof: According to the definition of the code $\mathcal{C}_B$, $(H_1' \cdot \boldsymbol{c}_1^T, \ldots, H_1' \cdot \boldsymbol{c}_n^T) \in \mathcal{C}_2$ and

$$(H_1' \cdot \boldsymbol{y}_1^T, \ldots, H_1' \cdot \boldsymbol{y}_n^T) = (H_1' \cdot \boldsymbol{c}_1^T, \ldots, H_1' \cdot \boldsymbol{c}_n^T) + (H_1' \cdot \boldsymbol{e}_1^T, \ldots, H_1' \cdot \boldsymbol{e}_n^T).$$

The vector $(H_1' \cdot \boldsymbol{e}_1^T, \ldots, H_1' \cdot \boldsymbol{e}_n^T)$ has weight at most $t_1 + t_2$. Since the code $\mathcal{C}_2$ can correct $t_1 + t_2$ errors, we get that $(\boldsymbol{s}_1^0, \ldots, \boldsymbol{s}_n^0) = (H_1' \cdot \boldsymbol{e}_1^T, \ldots, H_1' \cdot \boldsymbol{e}_n^T)$ after step 1.

At step 2, since for every $1 \leq i \leq n$, $H_1' \cdot \boldsymbol{y}_i^T = H_1' \cdot \boldsymbol{c}_i^T + H_1' \cdot \boldsymbol{e}_i^T$, if $wt(\boldsymbol{e}_i) \leq \ell_1$

$$\widehat{\boldsymbol{e}}_i^* = \mathcal{D}_1'(\boldsymbol{s}_i^0) = \boldsymbol{e}_i$$

as $\mathcal{C}_1'$ corrects ℓ_1 errors. However, if the weight of $\boldsymbol{e}_i$ is between $\ell_1 + 1$ and ℓ_2, then $\widehat{\boldsymbol{e}}_i^* = \mathcal{D}_1(\boldsymbol{s}_i^0) \neq \boldsymbol{e}_i$. This observation results from the fact that the decoder for $\mathcal{C}_1'$ can only return a cell-error vector of weight at most ℓ_1. In particular, we get that $wt(\widehat{\boldsymbol{e}}^*) \leq t_2$ and for all $1 \leq i \leq n$, $wt(\widehat{\boldsymbol{e}}_i^*) \leq \ell_1 + \ell_2$. Thus, at the end of step 3, $\boldsymbol{y}'$ contains no cell errors of weight less than ℓ_1 and all the remaining (at most t_2) cell errors have weight at most $\ell_1 + \ell_2$.

Steps 4 and 5 compute the syndrome using $\boldsymbol{y}'$ as input. Since the minimum distance of the code $\mathcal{C}_1$ is $2\ell_2+1 > \ell_1+\ell_2$, we get that for all $1 \leq i \leq n$, if a miscorrection occurred, then $\widehat{\boldsymbol{e}}_i^*$ is not a codeword in $\mathcal{C}_1$. In such case, $(\boldsymbol{s}_i', \boldsymbol{s}_i'') \neq (\boldsymbol{0}, \boldsymbol{0})$. In step 6, the set I is the set of all i, $1 \leq i \leq n$, such that $\ell_1 < wt(\boldsymbol{e}_i) \leq \ell_2$. In step 7, the word $\boldsymbol{y}''$ is the word $\boldsymbol{y}$ after removing all cell errors of weight at most ℓ_1.

In step 8, the remaining portion of the syndrome is recovered for all cell errors with more than ℓ_2 bits in error. Finally, in step 9, for every cell error at position i, $1 \leq i \leq n$, if ℓ_1 or fewer bit errors occurred, then $\widehat{\boldsymbol{e}}_i$ is the ith component of the cell-error vector $\widehat{\boldsymbol{e}}^*$. If more than ℓ_1 bit errors occurred, the decoder $\mathcal{D}_1$ is used. Since $\mathcal{C}_1$ can correct ℓ_2 errors and the syndrome $H_1 \cdot \boldsymbol{e}_i{}^T = (\boldsymbol{s}_i^0, \boldsymbol{s}_i^1)$ is known for all cell errors with more than ℓ_1 bits in error, these errors are corrected as well. ■

The following example illustrates the decoding process for the preceding construction.

Example 3: Suppose H_1', H_1'', H_2, H_3, and H_B are as in Example 2. By construction, $\mathcal{C}_B$ is a $[1, 1; 1, 3]_{2^3}$-bit-error-correcting code. Let α be a primitive element of $GF(4)$ with field element representations: $\alpha^0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \alpha^1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \alpha^2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, and $0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$.

Suppose the all-zero codeword is transmitted and the following vector $\boldsymbol{y}$ is received:

$$\boldsymbol{y} = \boldsymbol{e} = (110\ 100\ \mathbf{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}\ )$$

where $\boldsymbol{e}_1$ has two bits in error and $\boldsymbol{e}_2$ has a single bit in error. Here, $\mathbf{0}$ is a shorthand for $(0\ 0\ 0\ )$.

In this case, the output of $\mathcal{D}_2$ at step 1 is $(\boldsymbol{s}_1^0, \ldots, \boldsymbol{s}_n^0) = (\ \alpha^2\ \alpha^0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ )$.

The decoder $\mathcal{D}_1'$ at step 2 outputs

$$\widehat{\boldsymbol{e}}^* = (001\ 100\ \mathbf{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}\ ).$$

Notice that for position 2, the output is correct since only a single bit error occurred, whereas for position 1, the decoder $\mathcal{D}_1'$ miscorrects since two bits were in error. Now, at step 3, the decoder outputs

$$\boldsymbol{y}' = (111\ \mathbf{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}\ ).$$

The output of $\mathcal{D}_2$ at step 4 is clearly the all-zero vector. Now, the output of $\mathcal{D}_3$ at step 5 is $(\boldsymbol{s}_1'', \ldots, \boldsymbol{s}_n'') = (1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ )$ so that an error at position 1 has been detected. Thus, it is known that an error of magnitude greater than 1 has occurred in position 1 (step 6).

Step 7 results in the following vector:

$$\boldsymbol{y}'' = (110\ \mathbf{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}\ )$$

so that the maximum magnitude of any of the errors is only 2. The output of $\mathcal{D}_3$ at step 8 is the all-zero vector of length 15. Thus, the last step of the decoding procedure produces

$$\widehat{\boldsymbol{e}} = (110\ 100\ \mathbf{0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}\ ).$$

We now quickly consider a special case where ℓ_2 takes on the maximum possible value.

1) Special Case of $\ell_2 = m$: In the case that $\ell_2 = m$, it is required that $\mathcal{C}_1$ can correct up to m errors. This would imply that a parity-check matrix H_1 for such a code is an $m \times m$ matrix with full rank. Thus, given any $[m, m - r', \ell_1]_2$ code $\mathcal{C}_1'$ with a parity matrix H_1', any choice of $m - k$ additional row vectors for H_1'' such that the resulting matrix $H = \begin{pmatrix} H_1' \\ H_1'' \end{pmatrix}$ has full rank can be used to create a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-correcting code $\mathcal{C}_B$. Note that in Construction B, it is required that the parity-check matrix H_1 can be decomposed into two parts whereby H_1' is a $[m, m-r', \ell_1]_2$ code. Hence, for $\ell_2 = m$, any $[m, m - r', \ell_1]_2$ code can be chosen for $\mathcal{C}_1'$.

In general, the decoder $\mathcal{D}_1'$ outputs $\mathbf{0}$ if the weight of the error vector output exceeded the error-correction capability of the code. In the current setup with $\ell_2 = m$, such a constraint becomes vacuous. As a result, the overall decoder can be simplified as follows.

1) $\mathcal{D}_2(H_2 \cdot (H_1' \cdot \boldsymbol{y}_1^T, \ldots, H_1' \cdot \boldsymbol{y}_n^T)^T) = (\boldsymbol{s}_1^0, \ldots, \boldsymbol{s}_n^0)$.
2) $\boldsymbol{y}' = \boldsymbol{y} + (\mathcal{D}_1'(\boldsymbol{s}_1^0), \ldots, \mathcal{D}_1'(\boldsymbol{s}_n^0))$.
3) $\mathcal{D}_3(H_3 \cdot (H_1'' \cdot \boldsymbol{y}_1'^T, \ldots, H_1'' \cdot \boldsymbol{y}_n'^T)^T) = (\boldsymbol{s}_1^1, \ldots, \boldsymbol{s}_n^1)$.
4) Let $I = \{i : \boldsymbol{s}_i^1 \neq 0\}$.
5) $\widehat{\boldsymbol{e}} = (\widehat{\boldsymbol{e}}_1, \ldots, \widehat{\boldsymbol{e}}_n)$ where $\widehat{\boldsymbol{e}}_i = \mathcal{D}_1'(\boldsymbol{s}_i^0)$ if $i \notin I$ and otherwise $\widehat{\boldsymbol{e}}_i = \mathcal{D}_1(\boldsymbol{s}_i^0, \boldsymbol{s}_i^1)$.

Note that since H_1 has full rank, any of the miscorrections introduced in step 3 are corrected in steps 4 and 5.

C. Some Extensions

Modifications 1 and 2 presented in this section are extensions of Construction B. The idea is to use a combination of codes whose abilities are to correct errors, correct erasures, and detect errors.

In Modification 1, the code $\mathcal{C}_1'$ in Construction B is modified such that it corrects ℓ_1 errors and detects when there are between $\ell_1 + 1$ and ℓ_2 errors. Accordingly, the code $\mathcal{C}_3$ in Construction B needs only to correct t_2 erasures instead of t_2 errors.

Modification 1: Let $\mathcal{C}_C$ be a code with the following modifications with respect to the code construction of $\mathcal{C}_B$.

1) The matrix H_1' now consists of the first r' rows of H_1 where
   a) H_1' is a parity-check matrix for an $[m, m-r', \ell_1]_2$-bit-error-correcting code $\mathcal{C}_1'$;

b) the minimum distance of $\mathcal{C}_1'$ is at least $\ell_1+\ell_2+1$, so the code can also detect an error vector of weight between ℓ_1+1 and ℓ_2.

2) H_3 is a parity-check matrix of an $[n,k_3,\lceil\frac{t_2}{2}\rceil]_{2^{r''}}$ code $\mathcal{C}_3$ that can correct at least t_2 erasures, and $r_3=n-k_3$.

Note that, as earlier, the matrix H_2 is a parity-check matrix for an $[n,k_2,t_1+t_2]_{2^{r'}}$ code $\mathcal{C}_2$, and $r_2=n-k_2$. The matrix H_1'' is an $r''\times m$ matrix consisting of the last r'' rows of H_1, where $r''=r-r'$.

As earlier, a parity-check matrix for $\mathcal{C}_C$ is $H_C=\begin{pmatrix}H_2\otimes H_1'\\ H_3\otimes H_1''\end{pmatrix}$. The decoders of the codes $\mathcal{C}_1'$ and $\mathcal{C}_3$ are changed, while the decoders for $\mathcal{C}_1''$ and $\mathcal{C}_2$ remain the same as in Construction B. The decoder $\mathcal{D}_1'$, in addition to correcting ℓ_1 errors, also detects if the number of errors is between ℓ_1+1 and ℓ_2. The decoder performs the mapping

$$\mathcal{D}_1':\{0,1\}^{r'}\to\{0,1\}^m\cup\{E\}$$

where the symbol E indicates a detected error of weight between ℓ_1+1 and ℓ_2. Note that now the decoder $\mathcal{D}_1'$ never miscorrects. The input to the decoder $\mathcal{D}_3$ is no longer a syndrome but a vector $\boldsymbol{x}=(\boldsymbol{x}_0,\ldots,\boldsymbol{x}_{n-1})$ with at most t_2 erasures so that

$$\mathcal{D}_3:(\{0,1\}^{r''}\cup ?)^n\to(\{0,1\}^{r''})^n$$

where ? is the erasure symbol. The output of the decoder $\mathcal{D}_C$ is the "erasure" vector $\boldsymbol{e}=(\boldsymbol{e}_1,\ldots,\boldsymbol{e}_n)$, where for every $\boldsymbol{e}_i\neq\boldsymbol{0}$, $\boldsymbol{e}_i$ is the correct value for $\boldsymbol{x}_i$. The decoder $\mathcal{D}_C:(\{0,1\}^m)^n\to(\{0,1\}^m)^n$ is summarized in the following. Recall that the input to $\mathcal{D}_C$ is $\boldsymbol{y}=\boldsymbol{c}+\boldsymbol{e}$.

1) $\mathcal{D}_2(H_2\cdot(H_1'\cdot\boldsymbol{y}_1^T,\ldots,H_1'\cdot\boldsymbol{y}_n^T)^T)=(\boldsymbol{s}_1^0,\ldots,\boldsymbol{s}_n^0)$.
2) $\widehat{\boldsymbol{e}}^*=(\mathcal{D}_1'(\boldsymbol{s}_1^0),\ldots,\mathcal{D}_1'(\boldsymbol{s}_n^0))$.
3) Let $I=\{i:\widehat{\boldsymbol{e}}_i^*=E\}$.
4) Let $\widehat{\boldsymbol{e}}'$ satisfy $\widehat{\boldsymbol{e}}_i'=\boldsymbol{0}$ if $i\in I$ and $\widehat{\boldsymbol{e}}_i'=\widehat{\boldsymbol{e}}_i^*$ if $i\notin I$.
5) $\boldsymbol{y}'=\boldsymbol{y}+\widehat{\boldsymbol{e}}'$.
6) Let $\boldsymbol{x}$ satisfy $\boldsymbol{x}_i=?$ if $i\in I$ and $\boldsymbol{x}_i=H_1''\cdot\boldsymbol{y}_i'^T$ if $i\notin I$.
7) $\mathcal{D}_3(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n)=(\boldsymbol{s}_1^1,\ldots,\boldsymbol{s}_n^1)$.
8) $\widehat{\boldsymbol{e}}=(\widehat{\boldsymbol{e}}_1,\ldots,\widehat{\boldsymbol{e}}_n)$ where $\widehat{\boldsymbol{e}}_i=\widehat{\boldsymbol{e}}_i^*$ if $i\notin I$ and otherwise $\widehat{\boldsymbol{e}}_i=\mathcal{D}_1(\boldsymbol{s}_i^0,\boldsymbol{s}_i^1+H_1''\cdot\boldsymbol{y}_i'^T)$.

Lemma 2 establishes the correctness of the code construction by outlining the decoding procedure as previously done for Constructions A and B.

Lemma 2: The code $\mathcal{C}_C$ is an $[t_1,t_2;\ell_1,\ell_2]_{2^m}$-bit-error-correcting code.

Proof: Using the same arguments as in the proof of Theorem 1, if $wt(\boldsymbol{e}_i)\leq\ell_1$, then at step 2, we get that

$$\widehat{\boldsymbol{e}}_i^*=\mathcal{D}_1(\boldsymbol{s}_i^0)=\boldsymbol{e}_i.$$

However, if $\ell+1\leq wt(\boldsymbol{e}_i)\leq\ell_2$, then since $\mathcal{C}_1'$ can detect up to ℓ_2 errors

$$\widehat{\boldsymbol{e}}_i^*=\mathcal{D}_1'(\boldsymbol{s}_i^0)=E.$$

Thus, in step 3, we find all the locations of cell errors with weight greater than ℓ_1. In step 5, all the cell errors with ℓ_1 or less bits in error are corrected using the output from $\mathcal{D}_1'$ derived in step 2.

In step 6, for every cell error that has between ℓ_1+1 and ℓ_2 bits in error, $\boldsymbol{x}_i=?$, and by assumption there are at most t_2 of these. Since $\mathcal{C}_3$ is a code that can correct up to t_2 erasures, then every cell error with between ℓ_1+1 and ℓ_2 bits in error is corrected at step 7. Then, as in the proof of Theorem 1, each $\boldsymbol{e}_i$ can be recovered using either $\widehat{\boldsymbol{e}}_i^*$ or the decoder $\mathcal{D}_1$ with the syndrome $(\boldsymbol{s}_i^0,\boldsymbol{s}_i^1+H_1''\cdot\boldsymbol{y}_i'^T)$. ■

Yet another modification of Construction B is possible that may require less redundancy for the case where $t_2>t_1$. This modification is described in the following as Modification 2.

Modification 2: Let $\mathcal{C}_D$ be a code with these modifications with respect to the code construction of $\mathcal{C}_B$.

1) Let the matrix H_1' be a parity-check matrix of an ℓ_2-error-detecting code and consisting of the first r' rows of H_1.
2) The matrix H_3 is a parity-check matrix for a $[n,k_3,\lceil\frac{t_1+t_2}{2}\rceil]_{2^{r''}}$ code $\mathcal{C}_3$ that can correct up to t_1+t_2 erasures, and $r_3=n-k_3$.

As earlier, the matrix H_1'' is an $r''\times m$ matrix consisting of the last r'' rows of H_1, where $r''=r-r'$. The matrix H_2 is a parity-check matrix for an $[n,k_2,t_1+t_2]_{2^{r'}}$ code $\mathcal{C}_2$, and $r_2=n-k_2$.

The parity-check matrix is again the concatenation of constituent tensor products, as in (3). The encoders and decoders are the same as in Construction B except that the decoder $\mathcal{D}_1'$ can now only detect errors of weight at most ℓ_2. The decoding procedure is outlined in the following.

1) $\mathcal{D}_2(H_2\cdot(H_1'\cdot\boldsymbol{y}_1^T,\ldots,H_1'\cdot\boldsymbol{y}_n^T)^T)=(\boldsymbol{s}_1^0,\ldots,\boldsymbol{s}_n^0)$.
2) $\widehat{\boldsymbol{e}}^*=(\mathcal{D}_0'(\boldsymbol{s}_1^0),\ldots,\mathcal{D}_1'(\boldsymbol{s}_n^0))$.
3) Let $I=\{i:\widehat{\boldsymbol{e}}_i^*=E\}$.
4) Let $\boldsymbol{s}'$ satisfy $\boldsymbol{s}_i'=?$ if $i\in I$ and $s_i'=H_1''\cdot\boldsymbol{y}_i'^T$ if $i\notin I$.
5) $\mathcal{D}_3(\boldsymbol{s}_1',\ldots,\boldsymbol{s}_n')=(\boldsymbol{s}_1^1,\ldots,\boldsymbol{s}_n^1)$.
6) $\widehat{\boldsymbol{e}}=(\mathcal{D}_1(\boldsymbol{s}_1^0,\boldsymbol{s}_1^1+H_1''\cdot\boldsymbol{y}_1'^T),\ldots,\mathcal{D}_1(\boldsymbol{s}_n^0,\boldsymbol{s}_n^1+H_1''\cdot\boldsymbol{y}_n'^T))$.

The proof of the correctness is provided in Appendix. The main ideas are the same as earlier.

V. Code Analysis

In this section, we analyze the performance of Constructions A and B as well as Modifications 1 and 2 with respect to their redundancy. We begin by considering $[t,\ell]_{2^m}$-bit-error-correcting codes and show that under certain conditions, Construction A is perfect. We then analyze the minimum required redundancy of $[t_1,t_2;\ell_1,\ell_2]_{2^m}$-bit-error-correcting codes by counting the number of error vectors. Note that every $[t_1+t_2;\ell_2]_{2^m}$-bit-error-correcting code, given by Construction A, is a $[t_1,t_2;\ell_1,\ell_2]_{2^m}$-bit-error-correcting code as well. Hence, we will analyze under what conditions for the parameters $n,m,t_1,t_2,\ell_1,\ell_2$ Constructions A or B and Modifications 1 or 2 provide the least redundancy.

A nice property of Construction A is that if the codes $\mathcal{C}_1$ and $\mathcal{C}_2$ from Construction A are perfect, then the code $\mathcal{C}_A$ is perfect as well. For completeness, we provide the proof of this statement since we could not find it elsewhere.

Lemma 3: If the codes $\mathcal{C}_1$ and $\mathcal{C}_2$ from Construction A are perfect, then the code $\mathcal{C}_A$ is perfect as well.

Proof: The number of $[t;\ell]_{2^m}$-bit-error vectors of length n is given by

$$\sum_{k=0}^{t}\binom{n}{k}\left(\sum_{i=1}^{\ell}\binom{m}{i}\right)^k.$$

Since the code $\mathcal{C}_1$ is a perfect binary ℓ-error-correcting code, we get

$$2^{k_1}\cdot\sum_{i=0}^{\ell}\binom{m}{i}=2^m, \tag{4}$$

and similarly, since $\mathcal{C}_2$ is a perfect t-error-correcting code over $GF(2^{m-k_1})$, we get

$$(2^{m-k_1})^{k_2}\cdot\sum_{k=0}^{t}\binom{n}{k}(2^{m-k_1}-1)^k=(2^{m-k_1})^n. \tag{5}$$

From (4), we get $2^{m-k_1}-1=\sum_{i=1}^{\ell}\binom{m}{i}$, and thus, (5) becomes

$$(2^{m-k_1})^{k_2}\cdot\sum_{k=0}^{t}\binom{n}{k}\left(\sum_{i=1}^{\ell}\binom{m}{i}\right)^k=(2^{m-k_1})^n.$$

Thus,

$$\sum_{k=0}^{t}\binom{n}{k}\left(\sum_{i=1}^{\ell}\binom{m}{i}\right)^k=2^{(m-k_1)(n-k_2)}=2^{r_1r_2}.$$

Therefore, the code $\mathcal{C}_A$ is perfect as well. ■

The number of different $[t_1,t_2;\ell_1,\ell_2]_{2^m}$-bit-error vectors is given in the next lemma.

Lemma 4: Let m be a positive integer and $0<\ell_1<\ell_2\le m$, $t_1,t_2>0$. Then, the number of different $[t_1,t_2;\ell_1,\ell_2]_{2^m}$-bit-error vectors of length $n\ge t_1+t_2$ is given by

$$V=\sum_{i=0}^{t_2}\binom{n}{i}\left(\sum_{k=\ell_1+1}^{\ell_2}\binom{m}{k}\right)^i\cdot\sum_{j=0}^{t_1+t_2-i}\binom{n-i}{j}\left(\sum_{p=1}^{\ell_1}\binom{m}{p}\right)^j.$$

Proof: Let i be the number of cells that have between ℓ_1+1 and ℓ_2 bits in error. The total number of such errors is t_2. Then, for any value of i, $0\le i\le t_2$, there are $\binom{n}{i}$ choices of positions for such errors to occur. There are $\sum_{k=\ell_1+1}^{\ell_2}\binom{m}{k}$ distinct cell errors that have more than ℓ_1 bits in error but at most ℓ_2 bits in error, for each such symbol error.

Similarly, let j be the number of cells that have between 1 and ℓ_1 bits in error. Then, for any value of $j\le t_1+t_2-i$, there are $\binom{n-i}{j}$ possible choices of cells where errors can occur. Furthermore, when an error occurs there are $\sum_{p=1}^{\ell_1}\binom{m}{p}$ distinct cell errors that have between 1 and ℓ_1 bits in error. ■

Since V denotes the volume of the error vectors in a $[t_1,t_2;\ell_1,\ell_2]_{2^m}$ bit-error-correcting codes, $r_{\min}=\lceil\log V\rceil$ is the minimal redundancy in bits. Here, and in the rest of the paper, all logarithms are in base 2. In general, the redundancy of Constructions A and B and Modifications 1 and 2 depends on the choice of constituent codes. However, to simplify the analysis we assume in this section that both n and m are relatively large so that $\mathcal{H}(\frac{t}{n})$ can be reasonably approximated as $t\log n$, where $\mathcal{H}$ denotes the binary entropy function. Thus, all the analysis in this section will be performed using such approximations.

The goal of the following is to identify regimes where one code construction does better than the other. From Lemma 4, we approximate $r_{\min}$ as

$$r_{\min}\approx(t_1+t_2)\log n+(t_1\ell_1+t_2\ell_2)\log m.$$

The redundancy of the code $\mathcal{C}_B$ in Construction B using optimal choices redundancy wise for matrices H_1, H_2, and H_3 is

$$\begin{aligned}r_B&\approx(t_1+t_2)(\log n+r')+t_2(\log n+r'')\\&\approx(t_1+2t_2)\log n+(t_1+t_2)\ell_1\log m+t_2(\ell_2-\ell_1)\log m\\&=(t_1+2t_2)\log n+(t_1\ell_1+t_2\ell_2)\log m\end{aligned}$$

where $r'=\ell_1\log m$ and $r''=r-r'\approx t_2(\ell_2-\ell_1)\log m$ from Construction B.

Using the approximations for the redundancy of Construction B given previously, we now consider the difference between $r_{\min}$ and the redundancy of the code specified by Construction B. This difference is approximately

$$r_B-r_{\min}\approx t_2\log n.$$

Another alternative to constructing a $[t_1,t_2;\ell_1,\ell_2]_{2^m}$-bit-error-correcting code is by using a $[t_1+t_2;\ell_2]_{2^m}$-bit-error-correcting code $\mathcal{C}_A$ from Construction A. The redundancy of $\mathcal{C}_A$ is approximately

$$r_A\approx(t_1+t_2)(\log n+\ell_2\log m).$$

Therefore, the code $\mathcal{C}_B$ outperforms the code $\mathcal{C}_A$ redundancy wise approximately when $r_B\lesssim r_A$, that is,

$$(t_1+2t_2)\log n+(t_1\ell_1+t_2\ell_2)\log m\lesssim(t_1+t_2)(\log n+\ell_2\log m)$$

or

$$\frac{\log n}{\log m}\lesssim\frac{t_1}{t_2}(\ell_2-\ell_1).$$

Thus, Construction B requires less redundancy than Construction A whenever the ratio $\frac{t_1}{t_2}(\ell_2-\ell_1)$ is large.

Now, we consider the redundancy of a code given by Modification 1. Since a t_2 erasure-correcting code of length-n over $2^{r''}$ can correct at least $\lfloor\frac{t_2}{2}\rfloor$ errors, the redundancy can be approximated as $\frac{t_2}{2}(\log n+r'')$ bits. Then, the redundancy of $\mathcal{C}_C$ is

$$r_C\approx(t_1+t_2)(\log n+r')+\frac{t_2}{2}(\log n+r'').$$

Since a binary code that can correct up to ℓ_1 errors and detect up to ℓ_2 errors can correct at least $\ell_1+\lfloor\frac{\ell_2-\ell_1}{2}\rfloor$ errors, r' and r'' are approximated as $\frac{\ell_1+\ell_2}{2}\log m$ and $\frac{\ell_2-\ell_1}{2}\log m$, respectively. Then, we have

$$r_C\approx(t_1+\frac{3}{2}t_2)\log n+\frac{1}{4}\left(2t_1(\ell_1+\ell_2)+t_2(\ell_1+3\ell_2)\right)\log m.$$

TABLE IV
COMPARISON OF CONDITIONS FOR LEAST REDUNDANCIES FOR $\mathcal{C}_A, \mathcal{C}_B$, AND $\mathcal{C}_C$

Code	Condition on $\frac{\log n}{\log m}$
$\mathcal{C}_A$	$> (\frac{t_1}{t_2} + \frac{1}{2})(\ell_2 - \ell_1)$
$\mathcal{C}_B$	$< (\frac{t_1}{t_2} - \frac{1}{2})(\ell_2 - \ell_1)$
$\mathcal{C}_C$	$< (\frac{t_1}{t_2} + \frac{1}{2})(\ell_2 - \ell_1)$ and $> (\frac{t_1}{t_2} - \frac{1}{2})(\ell_2 - \ell_1)$

There are now two alternatives to consider. If Construction A is used to create a $[t_1 + t_2; \ell_2]_{2^m}$ code $\mathcal{C}_A$, then $r_A \gtrsim r_C$ approximately when

$$\frac{\log n}{\log m} \lesssim (\ell_2 - \ell_1)\left(\frac{t_1}{t_2} + \frac{1}{2}\right).$$

If Construction B is used to create a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$ code $\mathcal{C}_B$, then $r_B \gtrsim r_C$ approximately when

$$\frac{\log n}{\log m} \gtrsim (\ell_2 - \ell_1)\left(\frac{t_1}{t_2} - \frac{1}{2}\right).$$

Thus, Modification 1 offers the least redundancy amongst Constructions A and B and Modification 1 whenever the two previous inequalities hold. It will be shown in Example 4 that this modification also does well in the special case where $t_1 = t_2$ and $\ell_2 = \ell_1 + 1$ for any n, m. In general, the code with the least redundancy from amongst the choices $\mathcal{C}_A, \mathcal{C}_B$, and $\mathcal{C}_C$ depends on the ratio $\frac{\log n}{\log m}$. Table IV shows under what conditions for $\frac{\log n}{\log m}$, $\mathcal{C}_A, \mathcal{C}_B$, or $\mathcal{C}_C$ requires the least redundancy.

$\mathcal{C}_D$ is omitted from the aforementioned table because, as we will see, the optimal choice among $\mathcal{C}_A, \mathcal{C}_B, \mathcal{C}_C$, and $\mathcal{C}_D$ no longer depends on the same constants. We now consider the redundancy of $\mathcal{C}_D$ discussed in Modification 2. A $(t_1 + t_2)$ error-correcting code of length-n over $2^{r'}$ requires approximately $(t_1 + t_2)\log n + (t_1 + t_2)r'$ bits. The redundancy of a $(t_1 + t_2)$ erasure-correcting code over $2^{r''}$ can be approximated as requiring $\frac{t_1+t_2}{2}\log n + \frac{t_1+t_2}{2}r''$ bits. Furthermore, a binary code that can detect up to ℓ_2 errors is approximated as requiring $\frac{\ell_2}{2}\log m$ bits of redundancy. Note that now both r' and r'' are approximately $\frac{\ell_2}{2}\log m$. Then, the redundancy of $\mathcal{C}_D$ is approximately

$$\begin{aligned} r_D &\approx (t_1 + t_2)(\log n + r') + \frac{t_1 + t_2}{2}(\log n + r'') \\ &= \frac{3(t_1 + t_2)}{2}\log n + \frac{3(t_1 + t_2)}{4}\ell_2 \log m. \end{aligned}$$

Then, r_D is approximately less than r_A when

$$\frac{\log n}{\log m} \lesssim \frac{\ell_2}{2}.$$

The redundancy of Construction B is $(t_1+2t_2)\log n+(t_1\ell_1+t_2\ell_2)\log m$ so that $\mathcal{C}_D$ requires approximately less redundancy than $\mathcal{C}_B$ when

$$\frac{\log n}{\log m} \gtrsim \frac{t_1(3\ell_2 - 4\ell_1) - t_2\ell_2}{2(t_2 - t_1)}.$$

The redundancy of $\mathcal{C}_C$ is $(t_1 + \frac{3}{2}t_2)\log n + \frac{1}{4}(2t_1(\ell_1 + \ell_2) + t_2(\ell_1 + 3\ell_2))\log m$. Thus, using these approximations, $\mathcal{C}_D$ requires less redundancy than $\mathcal{C}_C$ whenever

$$\frac{\log n}{\log m} \lesssim \frac{t_1(2\ell_1 - \ell_2) + t_2\ell_1}{2t_1}.$$

The following example illustrates a special case where Modification 1 is nearly optimal.

Example 4: Let $\ell_1 = 1, \ell_2 = 2$, and $t_1 = t_2 = 1$. Suppose $\tilde{\mathcal{C}}$ is a $[15, 7, 2]_2$ code with a parity-check matrix $\tilde{H}$ and odd minimum distance. Now, we append an extra parity bit to every codeword in $\tilde{\mathcal{C}}$ to get the code $\mathcal{C}_1$. The minimum distance of the code $\tilde{\mathcal{C}}$ is increased by 1 and the resulting $\mathcal{C}_1$ code is a $[16, 7, 2]_2$ code. Suppose H_1 is a parity-check matrix of $\mathcal{C}_1$. Let H_1' be the top $r' = 5$ rows of H_1, and let H_1'' be the remaining (bottom) $r'' = 4$ rows of H. Then, H_1' is a parity-check matrix for a $[16, 11, 1]_2$ code. Note that this is an extended single error-correcting Hamming code with minimum distance 4 and $r' = 5$.

Let $\mathcal{C}_2$ be a double error-correcting code over $GF(2)^5$ of length n with parity-check matrix H_2, and let $\mathcal{C}_3$ be a single erasure-error-correcting code over $GF(2)^4$ of length n with a parity-check matrix H_3. Using the approach in Modification 1, and the matrices H_1', H_1'', H_2, and H_3 as the constituents in this construction, we construct a $[1, 1; 1, 2]_{2^{16}}$-bit error-correcting code.

The redundancy can be analyzed as follows. Since $\mathcal{C}_2$ is a double error-correcting code over $GF(2)^5$, it requires approximately $2\,(\log(n) + 5)$ bits of redundancy. Since a single erasure-correcting code $\mathcal{C}_3$ can be created using only a single additional parity symbol, it follows that the redundancy of $\mathcal{C}_3$ is approximately four bits. Thus, the overall redundancy of the code in the example is $2\log(n) + 14$ bits. Note that from Lemmas 4 and 5, the optimal redundancy is approximately$(t_1+t_2)\log n+t_1\ell_1\log m + t_2\ell_2\log m = 2\log n + 3\log 16 = 2\log n + 12$. The constructed code is thus nearly optimal.

In summary, Construction B offers the least redundancy amongst the four code choices whenever $\frac{t_1}{t_2}$ is large and $\ell_2 - \ell_1$ is large. Modification 2 has the least redundancy whenever $t_2 \gg t_1$. However, if $t_1 \gg t_2$ and $(\ell_2 - \ell_1)\left(\frac{t_1}{t_2} - \frac{1}{2}\right) \leq \frac{\log n}{\log m} \leq (\ell_2 - \ell_1)\left(\frac{t_1}{t_2} + \frac{1}{2}\right)$, then Modification 1 has approximately less redundancy than the other three code options. Construction A requires approximately the least redundancy for large $\frac{\log n}{\log m}$.

VI. PERFORMANCE AND RESULTS

In this section, the performance of various linear error-correcting codes with guaranteed error-correction capability is evaluated for a TLC Flash device. The goal of the simulations was to evaluate the ability of different error-correcting codes to delay the onset of errors for as long as possible. We compared five different types of codes:

1) binary codes with the same error correction capability for the LSB, CSB, and MSB pages;

TABLE V

Code	Rate
$[2^{12}, 3531, 47]_2$	0.86
$[2^{13}, 7242, 73]_2$	0.88
$[2^{14}, 14591, 128]_2$	0.89

TABLE VI

LSB Code	CSB Code	MSB Code	rate
$[2^{12}, 3351, 62]_2$	$[2^{12}, 3339, 63]_2$	$[2^{12}, 3915, 15]_2$	0.86
$[2^{13}, 6904, 99]_2$	$[2^{13}, 6891, 100]_2$	$[2^{13}, 7931, 20]_2$	0.88
$[2^{14}, 13905, 177]_2$	$[2^{14}, 13863, 180]_2$	$[2^{14}, 15963, 30]_2$	0.89

TABLE VII

CSB, LSB Code	MSB Code	Rate
$[2^{12}, 3338, 84]_4$	$[2^{12}, 3915, 15]_2$	0.86
$[2^{13}, 6882, 125]_4$	$[2^{13}, 7931, 20]_2$	0.89
$[2^{14}, 13862, 240]_4$	$[2^{14}, 15963, 30]_2$	0.89

TABLE VIII

Code	Rate
$[2^{12}, 3534, 80]_8$	0.86
$[2^{13}, 7244, 120]_8$	0.88
$[2^{14}, 14593, 205]_8$	0.89

TABLE IX

Codes	Rate
$[2^{12}, 3302, 88]_4, [2^{12}, 4011, 7]_2$	0.86
$[2^{13}, 6847, 128]_4, [2^{13}, 8087, 8]_2$	0.89
$[2^{14}, 13757, 250]_4, [2^{14}, 16271, 8]_2$	0.89

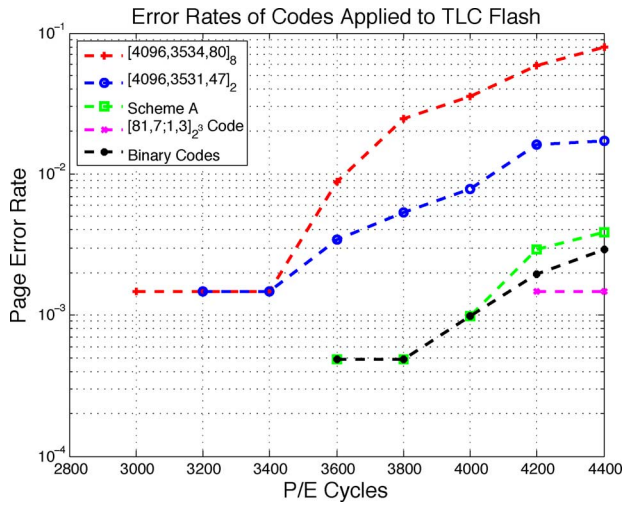

Fig. 2. Page error rates of codes applied to TLC Flash for code lengths 512 B.

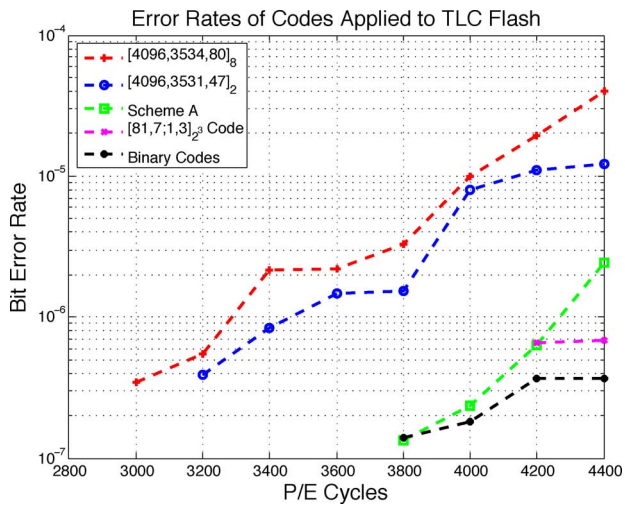

Fig. 3. BERs of codes applied to TLC Flash for code lengths 512 B.

2) binary codes with different error correction capability for the LSB, CSB, and MSB pages;
3) nonbinary codes over $GF(4)$ applied to the CSB and LSB sharing the same physical cells and binary codes applied to the MSB;
4) nonbinary codes over $GF(8)$ which correct errors in a group of LSB, CSB, and MSB pages sharing the same physical cells;
5) graded bit-error-correcting codes.

These codes were constructed for lengths of 4096, 8192, and 16 384 bits, and for rates between approximately 0.86 and 0.89. All the codes we used in our constructions are based on binary and nonbinary BCH codes. The codes listed in the following were created using the same techniques as in the MinT [16] database (see also [14]).

The specifications of the parameters for all different types of codes are summarized in the next tables.

1) Binary codes with the same error correction capability for the LSB, CSB, and MSB pages.
2) Binary codes (labeled "Binary Codes" in figures) with different error correction capability for the LSB, CSB, and MSB pages.
3) Scheme A—A nonbinary code over $GF(4)$ applied to the CSB and LSB pages sharing the same physical cells and a binary code applied to the MSB page.
4) Nonbinary codes over $GF(8)$ that correct errors in a group of LSB, CSB, and MSB pages sharing the same physical cells.
5) Graded bit-error-correcting codes applied to the LSB, CSB, and MSB pages. The following table lists the constituent codes that comprise each code depicted in the figures that follow. More specifically, the $[81, 7; 1, 3]_{2^3}$ code (shown in Figs. 2 and 3) is the result of applying Construction B to the two constituents in the first row of the table. The $[120, 8; 1, 3]_{2^3}$ (shown in Figs. 4 and 5) code is the result of applying Construction B to the two constituents from the second row of the table. The $[242, 8; 1, 3]_{2^3}$ (shown in Figs. 6 and 7) code is the result of applying Construction B to the two constituents in the third row.

We assume that all the constructed codes in our simulations have efficient encoders in terms of their time complexity as they are based on BCH codes. Hence, the complexity of the resulting graded bit-error-correcting code is on the same order as the complexity of a BCH code.

For each of the codes we used, if its error correction capability is t errors and there are at most t errors, then they are all corrected. Otherwise, we assume that the decoder would detect the error and will leave the information word unchanged.

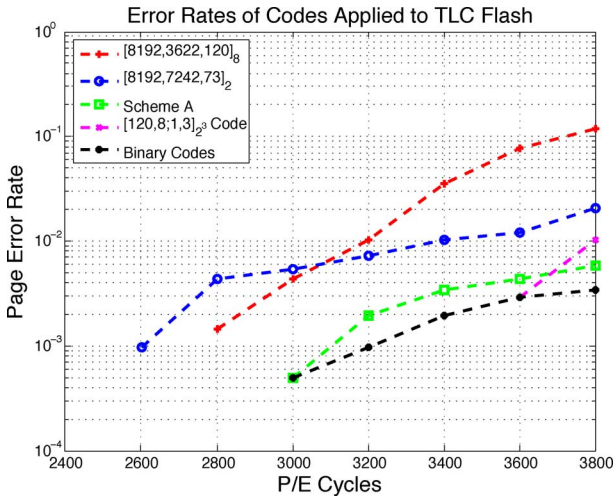

Fig. 4. Page error rates of codes applied to TLC Flash for code lengths 1 KB.

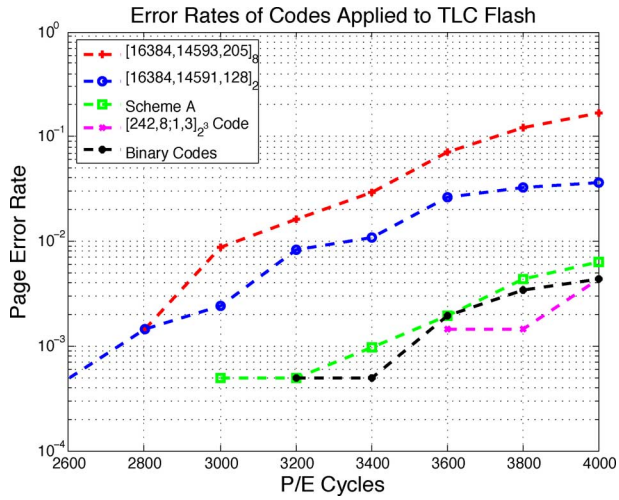

Fig. 6. Page error rates of codes applied to TLC Flash for code lengths 2 KB.

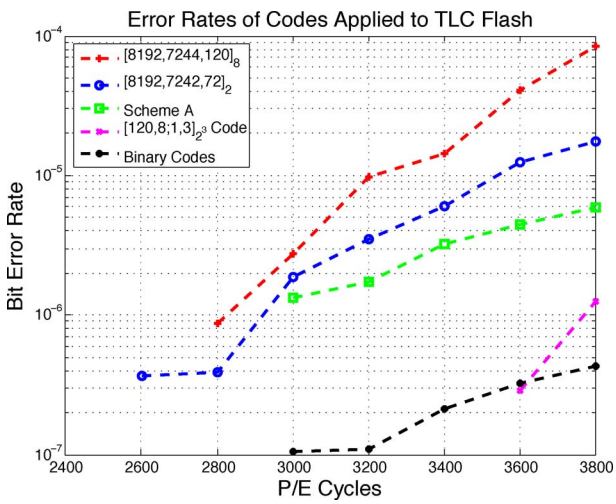

Fig. 5. BERs of codes applied to TLC Flash for code lengths 1 KB.

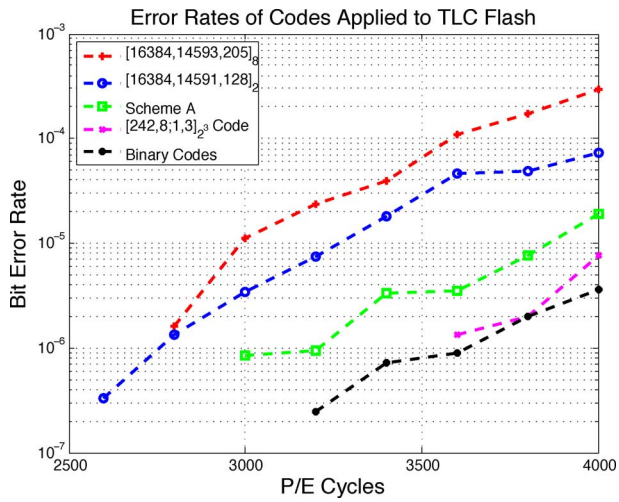

Fig. 7. BERs of codes applied to TLC flash for code lengths 2 KB.

For the codes that read multiple pages at the same time, it is assumed that if an error occurs in the decoder, then a page error occurs across all the pages from which the symbol was read. This means that when a symbol error occurs under a 2-bit alphabet (i.e., where the symbol was read using the information from two pages), both pages are considered to err. Similarly, when a symbol error occurs under a 3-bit alphabet, it is assumed that three pages are in error. Likewise, whenever a symbol error occurs, it is assumed that all the bits comprising the symbol are in error.

In Figs. 2 and 3, we show the page error rate and the BER, respectively, for codes of length 4096. Figs. 4 and 5 display the page error rate and the BER for codes of length 8192. In Figs. 6 and 7, the page error rate and the BER is plotted for codes of length 16 384, respectively. We note that in each performance comparison plots, the code lengths are the same.

As can be seen in Figs. 2–7 in the following, the codes over $GF(8)$ and the basic binary codes (where the same binary code is applied to the LSB, CSB, and MSB) consistently have the highest error rates both at a bit level and a page level. The $GF(8)$ code is inefficient since it can correct any type of cell error despite the fact that when a cell error occurs there is usually only one bit in error. The basic binary code is clearly inefficient since it does not take advantage of the difference in error rate between the MSB, CSB, and LSB. The remaining codes have lower error rates because they capitalize on these error profile characteristics.

Among the codes tested, the graded-bit-error correcting codes delayed the onset of errors the longest. Scheme A performed slightly worse than the method of applying different binary codes to each bit line. Part of the reason for this trend is that the binary codes used were simply better codes in the sense that they are closer to the sphere-packing bound than the nonbinary codes tested in this setup. Despite this advantage, the graded bit-error codes still outperformed all the binary codes tested.

Remark 4: We quickly remark on the potential performance gain offered by optimal codes (i.e., codes that meet the sphere-packing bound). To see the potential performance gain by using optimal codes (codes that are close to the sphere-packing bound), we ran simulations with the following codes:

1) $[89, 8; 1, 3]_{2^3}$ of length 4096;
2) $[150, 8; 1, 3]_{2^3}$ of length 8192; and
3) $[280, 12; 1, 3]_{2^3}$ of length 16 384.

Using the same approximations as in Section V, these codes have rates of around 0.88, 0.90, and 0.90, respectively. Using the first code, no errors appeared in the tested device until cycle 4700. The second code delayed the onset of errors until cycle 4400 and the third code delayed errors until cycle 4600. Thus, the device lifetime can be further extended if even better constituent codes were to be discovered and used.

While the simulations support the benefits of the proposed approach, it should also be noted that the proposed coding scheme will require a change in the current architecture of flash memories. Currently, in order to reduce the interferences among pages, it is required to write and read each one of the MSB, CSB, and LSB pages independently. However, independent coding of pages that share the same physical cells does not take advantage of the correlation between errors. Our goal in this paper is to demonstrate this correlation among errors and to propose suitable coding schemes that take the advantage of these correlations. The coding gain is demonstrated in the analysis of the codes in Section V and by simulation in this section. Our coding scheme only requires the three pages to be written together. This can be easily accomplished in case big files are written or by extending the logical page size. We are also aware that this change might decrease the reading and writing speed of the memory, and thus, we suggest to use this coding scheme only toward the end of the lifetime of the memory.

We quickly note that once errors were observed, the BER of our graded bit-error-correcting codes scheme was in some cases inferior to the BER of the "Binary Codes" scheme. This property results from the joint coding of three pages in our scheme. In particular, once the codes fail to correct the errors, more bits and pages were affected (and therefore more errors occurred). Furthermore, the binary codes are close to the sphere-packing bound, while the nonbinary codes we used in our scheme were far from optimal. In fact, Remark 4 shows that if optimal nonbinary codes were to be used as constituents in the graded-bit-error correcting code design, then even better results could be achieved.

In this section, it was demonstrated that leveraging graded bit-error-correcting codes delays the onset of high error rates associated with multilevel Flash devices. Because graded bit-error-correcting codes are designed specifically to account for the asymmetric nature of the observed errors, these codes can prolong the lifetime of multilevel Flash memory devices.

VII. Conclusion

In this work, data from a TLC Flash device demonstrated that when errors occur within a Flash cell, the vast majority of such errors only affect one of the three bits of information. This observation was used to motivate a new error-correction model for Flash memory. New error-correcting code constructions based upon generalized tensor product codes were provided. The proposed codes were analytically and empirically shown to offer a potentially valuable component for future coding schemes in the context of Flash memory.

Appendix
Proof of Correctness of Modification 2

Lemma 5: The code $\mathcal{C}_D$ is a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error-correcting code.

Proof: Suppose $\boldsymbol{y} = \boldsymbol{c}+\boldsymbol{e}$ is the input to decoder $\mathcal{D}_D$ where $\boldsymbol{c} \in \mathcal{C}_D$ and $\boldsymbol{e} \in (GF(2)^m)^n$ is a $[t_1, t_2; \ell_1, \ell_2]_{2^m}$-bit-error vector. Then, since $\mathcal{C}_2$ can correct $t_1 + t_2$ errors

$$(\boldsymbol{s}_1^0, \ldots, \boldsymbol{s}_n^0) = (H_1' \cdot \boldsymbol{e}_1^T, \ldots, H_1' \cdot \boldsymbol{e}_n^T)$$

where $\boldsymbol{e} = (\boldsymbol{e}_1, \ldots, \boldsymbol{e}_n)$. If $0 < wt(\boldsymbol{e}_i) \leq \ell_2$

$$\widehat{\boldsymbol{e}}_i^* = \mathcal{D}_1'(\boldsymbol{s}_i^1) = E$$

since $\mathcal{D}_1'$ can detect up to ℓ_2 errors. Thus, in step 3, the locations of all the cell errors are noted. In step 5, the locations of these errors are passed to $\mathcal{D}_3$ and since $\mathcal{D}_3$ can correct $t_1 + t_2$ erasures, the remaining portion of the syndrome $\boldsymbol{s}_i^1$ is recovered. In step 6, these two pieces are combined and given to $\mathcal{D}_1$ which can correct up to ℓ_2 errors so that every cell error can now be corrected. ■

Acknowledgment

The authors would like to thank Ms. Laura Grupp for her help with the data collection. The authors thank the anonymous reviewers for their expert comments that have help us to significantly improve the results section. We also thank Associate Editor, Prof. Olgica Milenkovic for a timely handling of our paper.

References

[1] H. Alhussien and J. Moon, "An iteratively decodable tensor product code with application to data storage," *IEEE J. Sel. Areas Commun.*, vol. 28, no. 2, pp. 228–240, Feb. 2010.
[2] Y. Cassuto, M. Schwartz, V. Bohossian, and J. Bruck, "Codes for asymmetric limited-magnitude errors with application to multi-level flash memories," *IEEE Trans. Inf. Theory*, vol. 56, no. 4, pp. 1582–1595, Apr. 2010.
[3] P. Chaichanavong and P. H. Siegel, "A tensor-product parity code for magnetic recording," *IEEE Trans. Magn.*, vol. 42, no. 2, pp. 350–352, Feb. 2006.
[4] P. Chaichanavong and P. H. Siegel, "Tensor-product parity codes: Combination with constrained codes and application to perpendicular recording," *IEEE Trans. Magn.*, vol. 42, no. 2, pp. 214–219, Feb. 2006.
[5] L. Dolecek, "Towards longer lifetime of emerging memory technologies using number theory," presented at the IEEE GLOBECOM Workshop Appl. Commun. Theory Emerg. Memory Technol., Miami, FL, Dec. 2010.
[6] N. Elarief and B. Bose, "Optimal, systematic q-ary codes correcting all asymmetric and symmetric errors of limited magnitude," *IEEE Trans. Inf. Theory*, vol. 56, no. 3, pp. 979–983, Mar. 2010.
[7] R. Gabrys, E. Yaakobi, L. Grupp, S. Swanson, and L. Dolecek, "Tackling intracell variability in TLC flash through tensor product codes," in *Proc. IEEE Int. Symp. Inf. Theory*, Boston, MA, Jul. 2012, pp. 1000–1004.
[8] L. M. Grupp, A. M. Caulfield, J. Coburn, S. Swanson, E. Yaakobi, P. H. Siegel, and J. K. Wolf, "Characterizing flash memory: Anomalies, observations, and applications," in *Proc. 42nd Annu. IEEE/ACM Int. Symp. Microarchitecture*, 2009, pp. 24–33.
[9] Q. Huang, S. Lin, and K. A. S. Abdel-Ghaffar, "Error-correcting codes for flash coding," *IEEE Trans. Inf. Theory*, vol. 57, no. 9, pp. 6097–6108, Apr. 2011.
[10] H. Imai and H. Fujiya, "Generalized tensor product codes," *IEEE Trans. Inf. Theory*, vol. 27, no. 2, pp. 181–187, Mar. 1981.
[11] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for flash memories," *IEEE Trans. Inf. Theory*, vol. 55, no. 6, pp. 2659–2673, Jun. 2009.

[12] T. Kløve and B. Bose, "Systematic, single limited magnitude error correcting codes for flash memories," *IEEE Trans. Inf. Theory*, vol. 57, no. 7, pp. 4477–4487, Jul. 2011.
[13] Y. Maeda and H. Kaneko, "Error control coding for multilevel cell flash memories using nonbinary low-density parity-check codes," in *Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Syst.*, Chicago, IL, Oct. 2009, pp. 367–375.
[14] Magma Computational Algebra System University of Sydney. Sydney, Australia, 2011 [Online]. Available: http://magma.maths.usyd.edu.au/magma
[15] R. Roth, *Introduction to Coding Theory*. Cambridge, U.K.: Cambridge Univ. Press, 2006.
[16] W. C. Schmid and R. Schürer, MinT, the online database for optimal parameters of (t, m, s)-nets (t, s)-sequences, orthogonal arrays, linear codes and OOAs University of Salzburg. Salzburg, Austria [Online]. Available: http://mint.sbg.ac.at/index.php
[17] M. Schwartz, "Quasi-cross lattice tilings with applications to flash memory," in *Proc. IEEE Int. Symp. Inf. Theory*, St. Petersburg, Russia, Aug. 2011, pp. 2133–2137.
[18] F. Sun, S. Devarajan, K. Rose, and T. Zhang, "Multilevel flash memory on-chip error correction based on trellis coded modulation," presented at the IEEE Int. Symp. Circuits Syst., Island of Kos, Greece, Sep. 2006.
[19] J. Wang, H. Shankar, and R. D. Wesel, "Soft information for LDPC decoding in flash:Mutual-information optimized quantization," presented at the IEEE GLOBECOM Conf., Houston, TX, Dec. 2011.
[20] J. K. Wolf, "An introduction to tensor product codes and applications to digital storage systems," in *Proc. IEEE Inf. Theory Workshop*, Punta del Este, Uruguay, Oct. 2006, pp. 6–10.
[21] J. K. Wolf, "Error-locating codes–A new concept in error control," *IEEE Trans. Inf. Theory*, vol. 9, no. 2, pp. 113–117, Apr. 1963.
[22] J. K. Wolf, "On codes derivable from the tensor product of check matrices," *IEEE Trans. Inf. Theory*, vol. 11, no. 2, pp. 281–284, Apr. 1965.
[23] E. Yaakobi, P. H. Siegel, A. Vardy, and J. K. Wolf, "On codes that correct asymmetric errors with graded magnitude distribution," in *Proc. IEEE Int. Symp. Inf. Theory*, St. Petersburg, Russia, Aug. 2011, pp. 1056–1060.
[24] E. Yaakobi, L. Grupp, P. H. Siegel, S. Swanson, and J. K. Wolf, "Characterization and error-correcting codes for TLC flash memories," in *Proc. IEEE Int. Conf. Comput., Netw. Commun.*, Maui, HI, Jan.–Feb. 2012, pp. 486–491.
[25] H. Zhou, A. Jiang, and J. Bruck, "Nonuniform codes for correcting asymmetric errors," in *Proc. IEEE Int. Symp. Inf. Theory*, St. Petersburg, Russia, Aug. 2011, pp. 1046–1050.

Ryan Gabrys received the B.S. degree in mathematics and computer science from the University of Illinois at Champaign-Urbana in 2005. In 2010 he received the Master of Engineering degree from the University of California at San Diego. In 2010, he was awarded the SMART scholarship through the Department of Defense, and since that time he has been pursuing a Ph.D. in electrical engineering at the University of California at Los Angeles. His research interests include coding theory and its applications to storage.

Eitan Yaakobi (S'07–M'12) received the B.A. degrees in computer science and mathematics, and the M.Sc. degree in computer science from the Technion—Israel Institute of Technology, Haifa, Israel, in 2005 and 2007, respectively, and the Ph.D. degree in electrical engineering from the University of California, San Diego, in 2011.

He is currently a postdoctoral researcher in electrical engineering at the California Institute of Technology, Pasadena. His research interests include coding theory, algebraic error-correction coding, and their applications for digital data storage and in particular for non-volatile memories.

Lara Dolecek (S'05–M'10) is an Assistant Professor with the Electrical Engineering Department at the University of California, Los Angeles (UCLA) where she is the director of the Laboratory for Robust Information Systems. She holds a B.S. (with honors), M.S. and Ph.D. degrees in Electrical Engineering and Computer Sciences, as well as an M.A. degree in Statistics, all from the University of California, Berkeley. She received the 2007 David J. Sakrison Memorial Prize for the most outstanding doctoral research in the Department of Electrical Engineering and Computer Sciences at UC Berkeley. Prior to joining UCLA, she was a postdoctoral researcher with the Laboratory for Information and Decision Systems at the Massachusetts Institute of Technology. She is a 2011 Hellman Fellow at UCLA. In 2012 she received NSF CAREER award and Okawa Research Grant award. She is an Associate Editor for IEEE Communication Letters and the lead guest editor for JSAC special issue on emerging data storage. She is also Data Storage Symposium co-chair for 2013 IEEE ICNC and has served a member of the technical program committee for over a dozen international conferences. Her research interests span coding and information theory, graphical models, statistical algorithms, and computational methods, with applications to emerging systems for data storage, processing, and communication.