Coding for Unreliable Flash Memory Cells

Ryan Gabrys, *Student Member, IEEE*, Frederic Sala, *Student Member, IEEE*, and Lara Dolecek, *Senior Member, IEEE*

***Abstract*—In this work, the model introduced by Gabrys is extended to account for the presence of unreliable memory cells. Leveraging data analysis on errors taking place in a TLC Flash device, we show that memory cells can be broadly categorized into reliable and unreliable cells, where the latter are much more likely to be in error. Our approach programs unreliable cells only in a limited capacity. In particular, we suggest a coding scheme, using generalized tensor product codes, that programs the unreliable cells only at certain voltage levels that are less likely to result in errors. We present simulation results illustrating an improvement of up to a half order of magnitude in page error rates compared to existing codes.**

***Index Terms*—Block codes, error correction, flash memories cells.**

I. Introduction

FLASH memories are among the most promising and popular data storage devices today. Increases in storage density in Flash have resulted in decreased reliability. Many new devices have bit error rates exceeding 10^{-4}.

In this work, we study triple-level cell (TLC) Flash memories, which store information in cells with eight discrete voltage levels represented as a triple-bit word. We refer to the first, second, and third bits in the word as the most-, center-, and least- significant bits (MSB, CSB, LSB), respectively. The mapping between levels and words is given in Table I.

In [1], coding schemes were developed based on data recorded from production TLC Flash chips. Errors were measured from sixteen Flash blocks evenly divided across two planes. Testing was performed by erasing a block followed by writing and reading back random data and checking for errors on the first of every 100 program/erase (P/E) cycles. On the other 99 cycles, the block was simply erased and every page was programmed to simulate the aging of the device.

Based on this data, [1] and [6] observed that errors largely affect only a single bit in each triple-bit word. Traditional, frequently-used BCH codes cannot take advantage of such error patterns. Instead, [1] and [6] proposed generalized tensor product codes ([4]), yielding some of the best-known Flash codes. However, [1] noted the existence of other prominent error patterns not exploited by the proposed codes. Additional characterization of Flash error patterns can be found in [2].

Manuscript received February 28, 2014; revised July 24, 2014; accepted July 25, 2014. Date of publication July 31, 2014; date of current version September 8, 2014. This work was supported in part by NSF Grants CAREER CCF-1150212 and CCF-1029030, Intel Corporation, and the UC-Mexus Program. The associate editor coordinating the review of this paper and approving it for publication was M. F. Flanagan.

The authors are with the University of California at Los Angeles, Los Angeles, CA 90095 USA (e-mail: rgabrys@ucla.edu; fredsala@ucla.edu; dolecek@ee.ucla.edu).

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/LCOMM.2014.2344677

TABLE I
Mapping Voltage Levels to Triple-Bit Words

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

Our goal is to revisit the data from [1] and suggest modifications to the generalized tensor product codes in [1] and [6] to take into account these additional error patterns. In particular, we found that Flash cells can be broadly divided into two categories: 1) reliable and 2) unreliable cells, where the vast majority of the errors (across the entire device) are a result of the unreliable cells experiencing errors. From our data, we found that only 62659 of the 134217728 cells (i.e., 0.0467%) experienced more than 50 errors across the lifetime (that we measured) of the device. This implies that roughly 10^{-4} of the cells accounted for over ten percent of the errors.

Motivated by these trends, we more closely inspected the unreliable cells and found that errors occur most frequently within these unreliable cells when the voltage levels (the programmed levels) were set to be above voltage level 3. This insight motivates the coding scheme proposed in this paper.

Similarly to the work in [3], our model assumes knowledge of the unreliable cells only at the time of encoding. Our codes have the benefit that the unreliable cell locations are not needed for decoding. Thus, this information can be stored in a temporary buffer (when performing the encoding process) and later discarded once encoding is complete. As will be described later in Section IV, programming unreliable cells using our proposed codes has the advantage that it requires fewer bits of parity than naively storing the locations of the unreliable cells in memory, thereby offering a potential advantage over traditional coding schemes.

There are several practical methods for determining the unreliable cells at the encoder side. One such scheme is to examine the number of rounds of charge injection necessary during the write process. Unreliable cells require many more rounds. Existing algorithms used to determine optimal charge injection strategies, such as [7], can be applied to this context. Another technique, which we adopt in this work and use for our simulations, is to characterize an unreliable cell as a cell that experienced an error in the previous P/E cycle. We note that a minor drawback of this scheme is that every write requires an additional read since the locations of the cells which were in error are necessary during the encoding process.

We will introduce the error model, motivated by the data described above, and revisit results from [1] and [6] in Section II. In Section III, we give a code construction for this model, which is applicable to TLC Flash. In Section IV, the proposed codes are compared against alternative codes.

II. Model and Preliminaries

We seek to develop a model based on the observations described in Section I. We begin with some notation.

Definition 1: We call a linear code over $GF(q)$ of length n and dimension k correcting up to t errors an $[n,k,t]_q$ code.

The codes considered here are designed for Flash cells with 8 levels. Each cell is represented by a 3-bit vector. Thus, the information stored within n TLC Flash cells is represented by a length-$3n$ binary vector where bits $3(i-1)+1,\dots,3i$ represent the information in the i-th cell for $1 \le i \le n$. The following definition captures the dominant error patterns, as observed in [1] and [6]. For a vector $\boldsymbol{x} \in GF(q)^n$ let $wt(\boldsymbol{x})$ denote the number of non-zero entries in $\boldsymbol{x}$.

Definition 2: Let n,t_1,t_2 be positive integers with $t_1+t_2 < n$. Then, a vector $\boldsymbol{e} = (\boldsymbol{e}_1,\boldsymbol{e}_2,\dots,\boldsymbol{e}_n)$ over $(GF(2)^3)^n$ is called a $[t_1,t_2]$-**bit-error-vector** if the following holds:

1) $|\{i : \boldsymbol{e}_i \ne \boldsymbol{0}\}| \le t_1+t_2$,
2) $|\{i : wt(\boldsymbol{e}_i) > 1\}| \le t_2$.

We refer to a linear code over $GF(2)$ of length $3n$ and dimension k that can correct any $[t_1,t_2]$-bit-error-vector as a $[3n,k,t_1,t_2]_2$-**bit-error-correcting code**.

Next, we take a slight detour from the TLC setting and describe codes for memories experiencing stuck-at errors, which will form a building block for our proposed codes. Consider such a memory device which stores the data $\boldsymbol{x} = (x_1,\dots,x_m) \in GF(2)^m$. For every position i, $1 \le i \le m$, there are three possible states for x_i: Either 1) x_i is stuck at 0, 2) x_i is stuck at 1, or 3) x_i can freely store a 0 or a 1. We therefore represent stuck-at errors using a ternary vector as follows. Let $\boldsymbol{s} = (s_1,\dots,s_m) \in GF(3)^m$ be a vector that represents stuck-at errors in the memory. For every $1 \le i \le m$, if $s_i = 2$, then position i has not experienced a stuck-at error and otherwise, if $s_i < 2$, then position i is stuck at value s_i.

We model the behavior of stuck-at memory using the map $\circ : GF(2)^m \times GF(3)^m \to GF(2)^m$. Suppose $\boldsymbol{a},\boldsymbol{b} \in GF(2)^m, \boldsymbol{s} \in GF(3)^m$ and $\boldsymbol{b} = \boldsymbol{a} \circ \boldsymbol{s} = (b_1,\dots,b_m)$. For $1 \le i \le m$, if $s_i < 2$, then $b_i = s_i$, and $b_i = a_i$ otherwise. Let $P_\ell \subseteq GF(3)^m$ be the set of all vectors $\boldsymbol{s} = (s_1,\dots,s_m) \in P_\ell$ such that $|\{i : s_i < 2\}| \le \ell$. We define the following class of codes, which is capable of recovering from ℓ stuck-at errors:

Definition 3: For positive integers m, k, t, and ℓ with k, t, $\ell < m$, an $[m,k,t,\ell]_2$ binary code $\mathcal{C}_S$ is a linear code of length m and dimension k over $GF(2)$ which is specified by an encoding map $\mathcal{E}_{\mathcal{C}_S} : \mathbb{Z}_{2^k} \times GF(3)^m \to GF(2)^m$ and a decoding map $\mathcal{D}_{\mathcal{C}_S} : GF(2)^m \to \mathbb{Z}_{2^k}$ such that

1) for any $\boldsymbol{s} \in P_\ell$ and any message $h \in \mathbb{Z}_{2^k}$, $\mathcal{E}_{\mathcal{C}_S}(h,\boldsymbol{s}) \circ \boldsymbol{s} = \mathcal{E}_{\mathcal{C}_S}(h,\boldsymbol{s})$, and
2) for any $\boldsymbol{e} \in GF(2)^m$ where $wt(\boldsymbol{e}) \le t$, $\mathcal{D}_{\mathcal{C}_S}(\mathcal{E}_{\mathcal{C}_S}(h,\boldsymbol{s}) + \boldsymbol{e}) = h$.

Unlike a (conventional) random error-correcting code, the codes from Definition 3 do not necessarily have a one-to-one mapping of messages to codewords. We refer to the set $\{1,2,\dots,n\}$ as $[n]$. The following class of codes, suitable for TLC Flash memory, is capable of correcting any $[t_1,t_2]$-bit-error-vector while ensuring that all cells at indices given by set $\mathcal{I}'$, with $|\mathcal{I}'| \le l$ map to voltage level 3 or below.

Definition 4: For positive integers n,k,t_1,t_2,ℓ with $\ell,t_1,t_2 < n$, a $[3n,k,t_1,t_2,\ell]_2$**dynamic bit-error-correcting code** $\mathcal{C}$ is a linear code of length $3n$ and dimension k over $GF(2)$ specified by an encoding map $\mathcal{E}_{\mathcal{C}} : \mathbb{Z}_{2^k} \times \{\mathcal{I} : \mathcal{I} \subset [n], |\mathcal{I}| \le \ell\} \to (GF(2)^3)^n$ and a decoding map $\mathcal{D}_{\mathcal{C}} : (GF(2)^3)^n \to \mathbb{Z}_{2^k}$ where for any $h \in \mathbb{Z}_{2^k}$ and any $\mathcal{I}' \subset [n]$ with $|\mathcal{I}'| \le \ell$,

1) the codeword $\boldsymbol{c} = \mathcal{E}_{\mathcal{C}}(h,\mathcal{I}') = (\boldsymbol{c}_1,\boldsymbol{c}_2,\dots,\boldsymbol{c}_n)$ is such that for any $i \in \mathcal{I}'$, the voltage level of $\boldsymbol{c}_i$ is at most 3,
2) for any $[t_1,t_2]$-bit-error-vector $\boldsymbol{e} \in (GF(2)^3)^n$, $\mathcal{D}_{\mathcal{C}}(\mathcal{E}_{\mathcal{C}}(h,\mathcal{I}') + \boldsymbol{e}) = h$.

III. Code Construction

In this section, we describe a $[3n,k,t_1,t_2,\ell]_2$ dynamic bit-error-correcting code $\mathcal{C}$ in terms of its parity check matrix. Our codes rely on tensor product operations, which allow us to build parity check matrices for codes capable of correcting bit-error-vectors. Let the symbol $\otimes$ denote the tensor product operation. Suppose α is a primitive element of $GF(4)$. We use the invertible map $\Gamma : GF(4) \to GF(2)^2$ where $\Gamma(\alpha) = (0,1)^T$, $\Gamma(\alpha^2) = (1,1)^T$, $\Gamma(\alpha^3) = (1,0)^T$, $\Gamma(0) = (0,0)^T$. Γ is applied to a matrix A by applying it to each element in A.

Construction: Let

$$H_1 = (\alpha\ \alpha^2\ \alpha^3),\ H_2 = (1\ 1\ 1) \tag{1}$$

so that $H_1 \in GF(4)^{1\times 3}, H_2 \in GF(2)^{1\times 3}$. Suppose that

1) H_3 is a parity check matrix for an $[n,k_3,t_1+t_2]_4$ code $\mathcal{C}_3$ according to Definition 1.
2) H_4 is a parity check matrix for an $[n,k_4,t_2,\ell]_2$ code $\mathcal{C}_4$ according to Definition 3.

Then, a parity check matrix for a $[3n,2k_3+k_4,t_1,t_2,\ell]_2$ dynamic bit-error-correcting code $\mathcal{C}$ of length n is given by

$$H = \begin{pmatrix} \Gamma(H_3 \otimes H_1) \\ H_4 \otimes H_2 \end{pmatrix}. \tag{2}$$

Example 1: Let $H_3 = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & \alpha \\ 0 & 1 & 0 & 0 & 0 & \alpha^2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & \alpha^2 \\ 0 & 0 & 0 & 0 & 1 & \alpha & \alpha \end{pmatrix} \in GF(4)^{5\times 7}$ be a parity check matrix for a $[7,2,2]_4$ code $\mathcal{C}_2$ and $H_4 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix} \in GF(2)^{3\times 7}$ be a parity check matrix for a $[7,3,1,1]_2$ code,[1] where α is a primitive element of $GF(4)$. Then from the previous construction, we have $H = \begin{pmatrix} \Gamma(H_3 & \otimes & (\alpha & \alpha^2 & \alpha^3)) \\ H_4 & \otimes & (1 & 1 & 1) \end{pmatrix}$. Since the elements in $GF(4)$ are represented as binary column vectors of length two (through the map Γ), $H \in GF(2)^{13\times 21}$ for a $[3\cdot 7, 2\cdot 2+3,1,1,1]_2$ dynamic bit-error-correcting code.

In the next subsections, we define the encoding map, $\mathcal{E}_{\mathcal{C}}$, and the decoding map, $\mathcal{D}_{\mathcal{C}}$, for a $[3n,2k_3+k_4,t_1,t_2,\ell]_2$ code.

[1]It can be easily verified that the codes corresponding to H_3 and H_4 have the desired error-correction ability.

A. Encoding Map

Suppose $\mathcal{C}$ is a $[3n, 2k_3 + k_4, t_1, t_2, \ell]_2$ dynamic bit-error-correcting code with the parity check matrix H in (2). If $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n) \in \mathcal{C}$ (each $\boldsymbol{x}_i$ is a binary vector of length 3 for $1 \le i \le n$) then two properties hold. In particular, in 1), the multiplications are taken over $GF(4)$, which is possible, as the elements of $\boldsymbol{x}_i$ are in $GF(2)$, a subfield of $GF(4)$.

1) $\boldsymbol{u} = (H_1 \cdot \boldsymbol{x}_1^T, H_1 \cdot \boldsymbol{x}_2^T, \ldots, H_1 \cdot \boldsymbol{x}_n^T) \in \mathcal{C}_3$, and
2) $\boldsymbol{z} = (H_2 \cdot \boldsymbol{x}_1^T, H_2 \cdot \boldsymbol{x}_2^T, \ldots, H_2 \cdot \boldsymbol{x}_n^T) \in \mathcal{C}_4$.

The next lemma is used to determine codebook sizes.

Lemma 1: Any codeword $\boldsymbol{x} \in \mathcal{C}$ can be uniquely mapped to $\boldsymbol{u} = (H_1 \cdot \boldsymbol{x}_1^T, H_1 \cdot \boldsymbol{x}_2^T, \ldots, H_1 \cdot \boldsymbol{x}_n^T) \in \mathcal{C}_3$ and $\boldsymbol{z} = (H_2 \cdot \boldsymbol{x}_1^T, H_2 \cdot \boldsymbol{x}_2^T, \ldots, H_2 \cdot \boldsymbol{x}_n^T) \in \mathcal{C}_4$.

From Lemma 1, we have that $\boldsymbol{x} \in \mathcal{C}$ is unique if the codewords $\boldsymbol{u} \in \mathcal{C}_3$ and $\boldsymbol{z} \in \mathcal{C}_4$ are fixed. The encoding procedure is therefore performed as follows. First a codeword $\boldsymbol{u} \in \mathcal{C}_3$ is chosen. Then, a codeword $\boldsymbol{z} \in \mathcal{C}_4$ is chosen. Recall that since $\mathcal{C}_4$ is a code that can recover from stuck-at errors, more than one codeword can be used to represent the same message. The idea is to choose the codeword $\boldsymbol{z} \in \mathcal{C}_4$ so that the unreliable cells have low voltages. We determine $\boldsymbol{u}$ and $\boldsymbol{z}$ and use them to compute $\boldsymbol{x}$, which is then stored in memory.

To concretely describe the choice of $\boldsymbol{z} \in \mathcal{C}_4$, we introduce the following map ϕ which maps 3 binary bits to a voltage level in $\{0, 1, \ldots, 7\}$ identically to the map in Table I,

$$\phi(000) = 5,\ \phi(001) = 4,\ \phi(100) = 2,\ \phi(010) = 6,$$
$$\phi(111) = 0,\ \phi(110) = 1,\ \phi(011) = 7,\ \phi(101) = 3.$$

The following claim is a result of the map ϕ and H_1 described in the construction.

Claim 1: For any $\boldsymbol{x}_1 \in GF(2)^3$, $H_1 \cdot \boldsymbol{x}_1^T = H_1 \cdot (\boldsymbol{x}_1 + (111))^T$. Furthermore, if $\phi(\boldsymbol{x}_1) \ge 4$, then $\phi(\boldsymbol{x}_1 + (111)) \le 3$.

From the previous discussion, the code $\mathcal{C}$ can represent $2^{2k_3+k_4}$ messages where each message in $\mathcal{C}$ can be mapped to a codeword in $\mathcal{C}_3$ and a message in $\mathcal{C}_4$. We now present the encoding map $\mathcal{E}_{\mathcal{C}} : \mathbb{Z}_{2^{2k_3+k_4}} \times \{\mathcal{I} : \mathcal{I} \subset [n], |\mathcal{I}| \le \ell\} \to (GF(2)^3)^n$ for a $[3n, 2k_3 + k_4, t_1, t_2, \ell]_2$ code. We use the following auxiliary maps. Let $\mathcal{E}_{\mathcal{C}_3} : \mathbb{Z}_{2^{2k_3}} \to GF(4)^n$ be the encoding map for an $[n, k_3, t_1 + t_2]_4$ code $\mathcal{C}_3$ over $GF(4)$. Let $\mathcal{E}_{\mathcal{C}_4} : \mathbb{Z}_{2^{k_4}} \times GF(3)^n \to GF(2)^n$ be the encoding map for an $[n, k_4, t_2, \ell]_2$ code $\mathcal{C}_4$ according to Definition 3. Let ψ be the following invertible function from $GF(4)$ to $GF(2)^3$:

$$\psi(0) = 000,\ \psi(\alpha) = 011,\ \psi(\alpha^2) = 101,\ \psi(\alpha^3) = 110. \quad (3)$$

If the map ψ is applied to a vector of length n, then ψ is applied to each symbol in the vector resulting in a length-$3n$ binary vector. The input to the encoder $\mathcal{E}_{\mathcal{C}}$ is a message $m \in \mathbb{Z}_{2^{2k_3+k_4}}$ and a set $\mathcal{I} \subset [n]$ where $|\mathcal{I}| \le \ell$. The output is a codeword $\boldsymbol{x} \in \mathcal{C}$.

Step 4) applies the result in Claim 1 to ensure that unreliable cell voltage levels do not exceed 3.

Below, H refers to the matrix in (2).

Lemma 2: For any $\mathcal{I} \subset [n]$ where $|\mathcal{I}| \le \ell$ and $m \in \mathbb{Z}_{2^{2k_3+k_4}}$, let $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = \mathcal{E}_{\mathcal{C}}(m, \mathcal{I})$. Then, $H \cdot \boldsymbol{x}^T = \boldsymbol{0}$ and for any $i \in \mathcal{I}$, $\phi(\boldsymbol{x}_i) \le 3$.

We provide an example that illustrates the encoding process outlined in Fig. 1 using the code from Example 1.

Example 2: Let $\mathcal{C}$ be the $[3 \cdot 7, 2 \cdot 2 + 3, 1, 1, 1]_2$ code from Example 1. Suppose that $\mathcal{I} = \{4\}$ and $\boldsymbol{u} = (\alpha^3, 0, \alpha^2, \alpha, \alpha^3, 0, \alpha^2)$ at step 1) of the encoding. Then, at step 2) of the encoding we have $\boldsymbol{s} = (2, 2, 2, 1, 2, 2, 2)$ since $\phi(\psi(\boldsymbol{u}_4)) = 7$. Suppose at step 3) the output of $\mathcal{E}_{\mathcal{C}_4}$ is $\boldsymbol{z} = (0, 1, 0, 1, 0, 1, 0)$. Then at step 4) of the encoding, $\boldsymbol{x} = (110, 111, 101, 100, 110, 111, 101)$. Note that the fourth cell has a value of 100 so that according to Table I, the cell is programmed at voltage level 2, which does not exceed level 3, as we desired.

```
1) Let $\boldsymbol{u} = \mathcal{E}_{\mathcal{C}_3}(m \mod 2^{2k_3})$.
2) Define $\boldsymbol{s} = (s_1, \ldots, s_n) \in GF(3)^n$ as follows. For every $i \in \mathcal{I}$,
     a) If $\phi(\psi(\boldsymbol{u}_i)) \ge 4$, then $s_i = 1$, and
     b) If $\phi(\psi(\boldsymbol{u}_i)) < 4$, then $s_i = 0$.
   If $i \notin \mathcal{I}$, then $s_i = 2$.
3) Let $\boldsymbol{z} = \mathcal{E}_{\mathcal{C}_4}(\lfloor \frac{m}{2^{2k_3}} \rfloor, \boldsymbol{s})$.
4) Define $\boldsymbol{x} = \psi(\boldsymbol{u}) + \boldsymbol{z} \otimes (111)$.
```

Fig. 1. Encoding map $\mathcal{E}_{\mathcal{C}}$.

```
1) Let $m_1 = \mathcal{D}_{\mathcal{C}_3}(H_1 \cdot \boldsymbol{y}_1^T, \ldots, H_1 \cdot \boldsymbol{y}_n^T)$.
2) Let $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n) = \mathcal{E}_{\mathcal{C}_3}(m_1) + (H_1 \cdot \boldsymbol{y}_1^T, \ldots, H_1 \cdot \boldsymbol{y}_n^T)$ and $\hat{\boldsymbol{e}} = (\mathcal{D}_{\mathcal{C}_1}(\boldsymbol{v}_1), \ldots, \mathcal{D}_{\mathcal{C}_1}(\boldsymbol{v}_n))$.
3) Define $\boldsymbol{y}' = \boldsymbol{y} + \hat{\boldsymbol{e}}$.
4) Let $m_2 = \mathcal{D}_{\mathcal{C}_4}(H_2 \cdot \boldsymbol{y}_1'^T, \ldots, H_2 \cdot \boldsymbol{y}_n'^T)$.
5) Define $\hat{m} = 2^{2k_3} \cdot m_2 + m_1$.
```

Fig. 2: Decoding map $\mathcal{D}_{\mathcal{C}}$.

B. Decoding Map

We now describe the decoding procedure for the code $\mathcal{C}$ in the Construction. Let $\mathcal{C}_1$ be a binary code with the parity check matrix H_1, where H_1 is as defined in the Construction. Notice that $\mathcal{C}_1$ is a binary single error-correcting code.

Let $\mathcal{D}_{\mathcal{C}_1} : GF(4) \to GF(2)^3$ be the decoder for the code $\mathcal{C}_1$. The input to the 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. Let $\mathcal{D}_{\mathcal{C}_3} : \{GF(4)\}^n \to \mathbb{Z}_{2^{2k_3}}$ be the decoder for $\mathcal{C}_3$ and $\mathcal{E}_{\mathcal{C}_3}$ be the encoder for $\mathcal{C}_3$, introduced in the previous subsection. Then, for any $\boldsymbol{e} \in GF(4)^n$ with $wt(\boldsymbol{e}) \le t_1 + t_2$ and any $h \in \mathbb{Z}_{2^{2k_3}}$, we define $\mathcal{D}_{\mathcal{C}_3}$ so that $\mathcal{D}_{\mathcal{C}_3}(\mathcal{E}_{\mathcal{C}_3}(h) + \boldsymbol{e}) = h$. Let $\mathcal{D}_{\mathcal{C}_4} : GF(2)^n \to \mathbb{Z}_{2^{k_4}}$ be the decoder for the code $\mathcal{C}_4$ where the input and output are as defined in Definition 3.

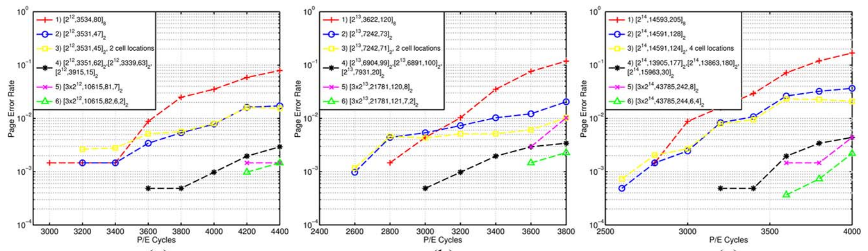
The decoder $\mathcal{D}_{\mathcal{C}} : (GF(2)^3)^n \to \mathbb{Z}_{2^{2k_3+k_4}}$ for the code $\mathcal{C}$ gets as an input a word of the form $\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{e}$, where $\boldsymbol{x} \in \mathcal{C}$ and $\boldsymbol{e} \in (GF(2)^3)^n$ is a $[t_1, t_2]$-bit-error-vector. The decoder outputs $\hat{m}$ which is an estimate of the message $m \in \mathbb{Z}_{2^{2k_3+k_4}}$ that was stored. The decoder $\mathcal{D}_{\mathcal{C}}$ performs the following sequence of steps, described in Fig. 2. We note that in step 1), the multiplications are performed over $GF(4)$, just as in the encoding map.

Lemma 3: For any $[t_1, t_2]$-bit-error-vector $\boldsymbol{e}$, $m \in \mathbb{Z}_{2^{2k_3+k_4}}$, and $\mathcal{I} \subset [n]$ where $|\mathcal{I}| \le \ell$, $\mathcal{D}_{\mathcal{C}}(\mathcal{E}_{\mathcal{C}}(m, \mathcal{I}) + \boldsymbol{e}) = m$.

The next theorem follows from Lemmas 2 and 3.

Theorem 1: The code $\mathcal{C}$ is a $[3n, 2k_3 + k_4, t_1, t_2, \ell]_2$ dynamic bit-error-correcting code.

Example 3: Let $\mathcal{C}$ be the $[3 \cdot 7, 2 \cdot 2 + 3, 1, 1, 1]_2$ code from Examples 1 and 2 and suppose that $\boldsymbol{x} = (110, 111, 101, 100, 110, 111, 101)$ is stored. Notice that $\boldsymbol{u} = (H_1 \cdot \boldsymbol{x}_1^T, \ldots, H_1 \cdot \boldsymbol{x}_7^T) = (\alpha^3, 0, \alpha^2, \alpha, \alpha^3, 0, \alpha^2) \in \mathcal{C}_3$ and $\boldsymbol{z} = (H_2 \cdot \boldsymbol{x}_1^T, \ldots, H_2 \cdot \boldsymbol{x}_7^T) = (0, 1, 0, 1, 0, 1, 0) \in \mathcal{C}_4$. Suppose the vector



(a) (b) (c)

Fig. 3. Page error rates (PERs) for codes of varying lengths. The code numbers correspond to the list in Section IV. (a) PER for length 4096 codes; (b) PER for length 8192 codes; (c) PER for length 16384 codes.

$\boldsymbol{y} = \boldsymbol{x} + (001, 000, 000, 011, 000, 000, 000) = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_7)$ was retrieved. Let $\boldsymbol{v}' = (H_1 \cdot \boldsymbol{y}_1^T, \ldots, H_1 \cdot \boldsymbol{y}_7^T) = (0, 0, \alpha^2, 0, \alpha^3, 0, \alpha^2)$. At step 1), since $\mathcal{C}_3$ is a double error-correcting code by design, the decoder for $\mathcal{C}_3$ can determine $\boldsymbol{u}$ from $\boldsymbol{v}'$. Then at step 2), in the decoding we have $\boldsymbol{v} = (\alpha^3, 0, 0, \alpha, 0, 0, 0)$ and $\hat{\boldsymbol{e}} = (001, 000, 000, 100, 000, 000, 000)$. At step 3), we have $\boldsymbol{y}' = (110, 111, 101, 011, 110, 111, 101)$. Let $\boldsymbol{v}'' = (H_2 \cdot \boldsymbol{y}_1'^T, \ldots, H_2 \cdot \boldsymbol{y}_7'^T) = (0, 1, 0, 0, 0, 1, 0)$. The decoder for $\mathcal{C}_4$ can determine $\boldsymbol{z}$ from $\boldsymbol{v}''$ to recover the stored message.

IV. Evaluation

Next we evaluate the potential reduction in the error rate of a Flash device by applying a dynamic bit-error-correcting code. We selected from the following classes of codes, created using the same techniques as in the MinT database [5].

1) non-binary codes over $GF(8)$ correcting errors in a group of LSB, CSB, and MSB pages sharing the same physical cells,
2) binary codes with the same error correction ability for the LSB, CSB, and the MSB pages,
3) binary codes that do not program into unreliable cells,
4) binary codes with different error correction ability for the LSB, CSB, and the MSB pages (respectively),
5) graded bit-error-correcting codes from [1], and
6) dynamic bit-error-correcting codes.

The third class of codes does not program into a number of unreliable cells. This number, given in the legend in Fig. 3, is selected to optimize performance. To account for the extra storage required by these codes (since the locations of the unreliable cells must be known at the decoder and thus be stored), the information rate of the code was reduced. For the purposes of the test, we assumed that a cell was unreliable if it was in error in the previous cycle.

Notice that if a length n bit-error-correcting code were to store the locations of t unreliable cells between the time of encoding and decoding then an additional $t \log_2(n)$ bits of metadata would be required. Under our setup, a dynamic bit-error-correcting code that can encode up to t unreliable cells would require only $(t/2) \log_2(n)$ bits of parity.

The evaluation was performed by programming at most ℓ unreliable cells at low voltage levels per data page. If there were more than ℓ unreliable cells, the first ℓ (lowest index in the page) were chosen. These cells were programmed to levels not exceeding 3 as guaranteed by the $[n, k, t_1, t_2, \ell]_{2^3}$ code. We chose values of ℓ of 2, 2, and 4 for code lengths 2^{12}, 2^{13}, and 2^{14}, respectively, so that the resulting code has the same number of parity bits as the codes it was tested against.

The dynamic bit-error-correcting codes were constructed by weakening one of the constituent codes for the graded bit-error-correcting code ([1]). However, the resulting codes can control the voltage levels of unreliable cells. The values for ℓ were determined by a computer search. As can be seen from Fig. 3, the dynamic bit-error-correcting codes had a half-order of magnitude improvement in error rates compared to the graded bit-error-correcting codes, which themselves offer an approximately 40% lifetime improvement in comparison to the widespread, frequently-used binary BCH codes of equal rate. In particular, the dynamic bit-error-correcting codes seem to offer the greatest advantage for longer block lengths, where it is essentially guaranteed that unreliable cells can be located and carefully programmed to (potentially) avoid cell errors.

V. Conclusion

We observed from TLC Flash chip data that many errors result from a small number of unreliable cells. Such cells produced errors more frequently when programmed to high voltages. We proposed dynamic bit-error-correcting codes to take this trend into account. Simulations show that these codes reduce the page error rates of the graded bit-error-correcting codes proposed in [1] by half an order of magnitude, outperforming other more conventional coding schemes.

Acknowledgment

The authors sincerely thank the editor Prof. Mark Flanagan and the anonymous reviewers for their expert comments that have helped improve the paper.

References

[1] R. Gabrys, E. Yaakobi, and L. Dolecek, "Graded bit-error-correcting codes with applications to Flash memory," *IEEE Trans. Inf. Theory*, vol. 59, no. 4, pp. 2315–2327, Apr. 2013.
[2] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai, "Error patterns in MLC NAND flash memory: Measurement, characterization, analysis," in *Proc. DATE*, Dresden, Germany, Mar. 2012, pp. 521–526.
[3] C. Heegard, "Partitioned linear block codes for computer memory with 'stuck-at' defects," *IEEE Trans. Info. Theory*, vol. 29, no. 6, pp. 831–842, Nov. 1983.
[4] H. Imai and H. Fujiya, "Generalized tensor product codes," *IEEE Trans. Info. Theory*, vol. 27, no. 2, pp. 181–187, Mar. 1981.
[5] W. C. Schmid and R. Schurer, "MinT, the online database for optimal parameters of (t, m, s)-nets, (t, s)-sequences, orthogonal arrays, linear codes and OOAs," Univ. Salzburg. [Online]. Available: http://mint.sbg.ac.at/index.php
[6] E. Yaakobi, L. Grupp, P. H. Siegel, S. Swanson, and J. K. Wolf, "Characterization and error-correcting codes for TLC flash memories," in *Proc. ICNC*, Maui, HI, USA, Jan.–Feb. 2012, pp. 486–491.
[7] A. Jiang and H. Li, "Optimized cell programming for Flash memories," in *Proc. IEEE PACRIM*, Aug. 2009, pp. 914–919.