Multidimensional Flash Codes

Eitan Yaakobi, Alexander Vardy, Paul H. Siegel, and Jack K. Wolf
University of California, San Diego
La Jolla, CA 92093 – 0401, USA
Emails: eyaakobi@ucsd.edu, avardy@ucsd.edu, psiegel@ucsd.edu, jwolf@ucsd.edu

***Abstract*—Flash memory is a non-volatile computer memory comprised of blocks of cells, wherein each cell can take on q different levels corresponding to the number of electrons it contains. Increasing the cell level is easy; however, reducing a cell level forces all the other cells in the same block to be erased. This erasing operation is undesirable and therefore has to be used as infrequently as possible. We consider the problem of designing codes for this purpose, where k bits are stored using a block of n cells with q levels each. The goal is to maximize the number of bit writes before an erase operation is required. We present an efficient construction of codes that can store an arbitrary number of bits. Our construction can be viewed as an extension to multiple dimensions of the earlier work of Jiang and Bruck, where single-dimensional codes that can store only 2 bits were proposed.**

I. Introduction

Flash memories are, by far, the most important type of non-volatile computer memory in use today. They are employed widely in mobile, embedded, and mass-storage applications, and the growth in this sector continues at a staggering pace.

A flash memory consists of an array of floating-gate *cells*, organized into *blocks* (a typical block comprises 2^{17} to 2^{20} cells). Hot-electron injection [16] is used to inject electrons into a cell, where they become trapped. The Fowler-Nordheim tunneling [19] mechanism (field emission) can be used to remove electrons from an entire block of cells, thereby discharging them. The level or "state" of a cell is a function of the amount of charge (electrons) trapped within it. Historically, flash cells have been designed to store only two values (one bit); however, *multilevel flash cells* are actively being developed and are already in use in some devices [2], [7]. In multilevel flash cells, voltage is quantized to q discrete threshold values, say $0, 1, \ldots, q-1$. The parameter q can range from $q = 2$ (the conventional two-state case) up to $q = 256$.

The most conspicuous property of flash storage is its inherent asymmetry between cell programming (charge placement) and cell erasing (charge removal). While adding charge to a single cell is a fast and simple operation, removing charge from a cell is very difficult. In fact, flash memories *do not allow* a single cell to be erased — rather *only entire blocks* (comprising up to 2^{20} cells) can be erased. Thus, a single-cell erase operation requires the cumbersome process of copying an entire block to a temporary location, erasing it, and then re-programming all the cells except one. Moreover, since over-programming (raising the charge of a cell above its intended level) can only be corrected by a block erasure, in practice a conservative procedure is used for programming a cell. Charge is injected into the cell over numerous rounds; after every round, the charge level is measured and the next-round injection is configured, so that the charge gradually approaches its desired level. All this is extremely costly in time and energy.

Codes designed to address this problem were first introduced in [10], [11], [12], and are called *floating codes*. Here we address these codes slightly differently under the name of *flash codes*. Flash codes are a sweeping generalization of write-once memory codes [4], [6], [18], designed to maximize the number of times information can be rewritten before block erasures are required. In a nutshell, the idea is to use n q-level cells to store $k < n\log_2 q$ bits, thereby storing less bits than possible (the rate of the flash code is $k/(n\log_2 q)$). The bits are represented in a clever way to guarantee that every sequence of up to t writes (of a single bit) does not lead to any of the n cells exceeding its maximum value $q-1$. Recently, several more papers have appeared [1], [3], [5], [8], [9], [13], [14], [15], [17] describing more problems in this model.

Let us begin by giving a precise definition of *flash codes*. An insightful way to do so is in terms of a pair of graphs and a pair of mappings between these graphs. The first graph is the familiar hypercube $\mathcal{H}_k$. The vertices of $\mathcal{H}_k$, called the *variable vectors*, are the 2^k binary vectors of length k, with two such vertices $\alpha, \beta \in \mathbb{F}_2^k$ being adjacent iff $\mathrm{d_H}(\alpha,\beta) = 1$ ($\mathrm{d_H}(\alpha,\beta)$ denotes the Hamming distance between α and β). This graph constitutes the state transition diagram of the k information bits. A single-bit write operation corresponds to the traversal of an edge in $\mathcal{H}_k$, and a sequence of t writes is a *walk of length* t in $\mathcal{H}_k$. To describe the second graph, set $\mathcal{A}_q = \{0, 1, \ldots, q-1\}$, and think of $\mathcal{A}_q$ as a subset of the integers. Now consider the *directed* graph $\mathcal{G}_n$ whose vertices are the q^n vectors of length n over $\mathcal{A}_q$, and are called the *cell state vectors*. There is a directed edge from $\boldsymbol{x} \in \mathcal{A}_q^n$ to $\boldsymbol{y} \in \mathcal{A}_q^n$ in $\mathcal{G}_n$ iff $\mathrm{d_H}(\boldsymbol{x},\boldsymbol{y}) = 1$ and in the single position i where $\boldsymbol{x}$ and $\boldsymbol{y}$ differ, we have $y_i = x_i + 1$. The graph $\mathcal{G}_n$ is the state transition diagram of n flash memory cells. Observe that there is a path from $\boldsymbol{x}$ to $\boldsymbol{y}$ in $\mathcal{G}_n$ iff $y_i \geq x_i$ for all $i = 1, 2, \ldots, n$, which reflects the condition that the charge of memory cells can only increase. The graphs $\mathcal{H}_k$ and $\mathcal{G}_n$ are illustrated in Figure 1 for the case $k = 3$, $n = 2$, and $q = 8$.

An $(n,k)_q$ flash code $\mathbb{C}$ can now be specified in terms of two maps: a decoding map Δ and a transition map f. The decoding map $\Delta : \mathcal{A}_q^n \to \mathbb{F}_2^k$ simply indicates for each cell state vector $\boldsymbol{x} \in V(\mathcal{G}_n)$ the value of the variable vector associated with the corresponding cell state vector. This map can be, in principle, arbitrary, although it must be chosen

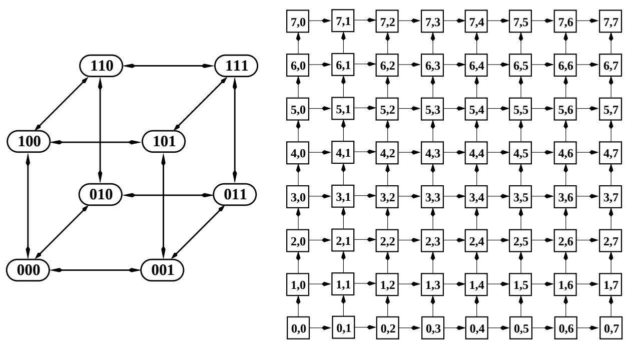


Fig. 1. State transition diagrams for $k = 3$ bits, and for $n = 2$ memory cells with $q = 8$ levels.

carefully in order to obtain flash codes with good performance. For each $\alpha \in \mathbb{F}_2^k$, let $\mathcal{G}_n(\alpha)$ denote the set of vertices $\boldsymbol{x} \in V(\mathcal{G}_n)$ such that $\Delta(\boldsymbol{x}) = \alpha$. With this, the transition map $f : E(\mathcal{H}_k) \times \mathcal{A}_q^n \to \mathcal{A}_q^n \cup \{\mathsf{E}\}$ can be described as follows. For every edge $(\alpha, \beta) \in E(\mathcal{H}_k)$ and every vertex $\boldsymbol{x} \in \mathcal{G}_n(\alpha)$, the value of $f(\alpha, \beta; \boldsymbol{x})$ is a vertex $\boldsymbol{y} \in \mathcal{G}_n(\beta)$ such that $y_i \geq x_i$ for all i (so that there exists a path from $\boldsymbol{x}$ to $\boldsymbol{y}$ in $\mathcal{G}_n$). Or, if no such vertex exists, $f(\alpha, \beta; \boldsymbol{x}) = \mathsf{E}$ indicating that a block erasure is required. It is clear from the definition that (Δ, f) map walks in $\mathcal{H}_k$ onto directed paths in $\mathcal{G}_n$, potentially terminating in the block-erasure symbol E.

Definition 1: A flash code $\mathbb{C}(\Delta, f)$ *guarantees t writes* if all walks of length t in $\mathcal{H}_k$, starting at the vertex $(0, 0, \ldots, 0)$, map onto valid paths in $\mathcal{G}_n$, never producing the symbol E.

The weight of a cell state vector $\boldsymbol{x} \in \mathcal{A}_q^n$ is defined to be $w_{\boldsymbol{x}} = \sum_{i=1}^{n} x_i$, and for convenience will be called the *cell state weight*. We note that at each write operation the increase in the cell state weight is at least one, and hence a trivial upper bound for the number of writes, t, is $n(q-1)$.

Definition 2: If a flash code guarantees at least t writes before erasing, then its *write deficiency* (or simply *deficiency*) is defined to be $\delta = n(q-1) - t$.

In [12], a code for storing two bits is presented. The code is constructed for arbitrary n and q and guarantees $t = (n-1)(q-1) + \left\lfloor \frac{q-1}{2} \right\rfloor$ writes. A general upper bound on t which holds for any k, ℓ, n, q (ℓ is the variable alphabet size, and usually $\ell = 2$) is presented as well, and assures that this two-bits construction is optimal.

Theorem 1: [12] For any code that uses n q-level cells and guarantees t writes before erasing, if $n \geq k(l-1) - 1$, then $t \leq (n - k(l-1) + 1) \cdot (q-1) + \left\lfloor \frac{(k(l-1)-1) \cdot (q-1)}{2} \right\rfloor$; if $n < k(l-1) - 1$, then $t \leq \left\lfloor \frac{n(q-1)}{2} \right\rfloor$.

This bound provides us also with a lower bound on the write deficiency of flash codes.

Corollary 1: For any code that uses n q-level cells and guarantees t writes before erasing, the code deficiency satisfies $\delta \geq (k(l-1) - 1) \cdot (q-1) - \left\lfloor \frac{(k(l-1)-1) \cdot (q-1)}{2} \right\rfloor$ if $n \geq k(l-1) - 1$, and $\delta \geq n(q-1) - \left\lfloor \frac{n(q-1)}{2} \right\rfloor$ if $n < k(l-1) - 1$. Furthermore, the bound in [12] implies that for n large enough the write deficiency of the code is not dependent on n.

Definition 3: Let q be a fixed number of cell levels, and k a fixed number of variables. A family of $(n_i, k)_q$ flash codes $\mathbb{C}_i$ (where $\lim_{i \to \infty} n_i = \infty$) that guarantees $t(n_i, q)$ writes is called *asymptotically optimal* if

$$\lim_{i \to \infty} \frac{t(n_i, q)}{n_i(q-1)} = 1.$$

Asymptotically optimal constructions are presented in [12] for storing two and three bits. These constructions are later enhanced and generalized for $3 \leq k \leq 6$ bits in [13]. Also, in [13] a new construction of codes, called *indexed codes*, is presented and supports the storage of an arbitrary number of variables. This construction, though shown to be asymptotically optimal, has deficiency that is dependent on the number of cells, n, and hence is still far from the lower bound on the deficiency.

The rest of the paper is organized as follows. In Section II we present another optimal construction for storing two bits. In Section III we demonstrate the basic idea of how to represent an arbitrary number of bits inside a multidimensional box. Our main construction is given in Section IV. We construct codes for efficient storage of bits such that the write deficiency does not depend on the number of cells. Finally, conclusions and an open problem are given in Section V.

II. Another Optimal Construction for Two Bits

The construction presented in [12] for storing two bits using an arbitrary number of q-level cells is optimal. We present here another optimal construction which we believe is simpler. In this construction, the leftmost (rightmost) cells of the memory correspond to the first (second) bit. In writing, if we change the first (second) bit then we increase by one the leftmost (rightmost) cell having level less than $q-1$. We repeat this process until the cells coincide and then the only cell of level less than $q-1$ represents the two bits. In general, the cell state vector has the following form: $(q-1, \ldots, q-1, x_i, 0, \ldots, 0, x_j, q-1, \ldots, q-1)$, where $0 < x_i, x_j \leq q-1$. We present here the construction for odd values of q, but it is easily modified to handle even values as well.

Encoding: We consider the following cases:

1) There are at least two cells of level less than $q-1$. If we change the level of $v_1 (v_2)$ then we increase by one the leftmost (rightmost) cell having level less than $q-1$. If after this change there is only one cell of level less than $q-1$, then it has to represent two bits. We change its level such that its residue modulo 4 corresponds to the four possible variable vectors. If the cell level has residue modulo 4 equal to $0, 1, 2$, or 3, then the corresponding variable vector is $(0,0), (0,1), (1,0)$, or $(1,1)$, respectively.
2) There is only one cell of level less than $q-1$. In this case the cell represents both bits and we increase its level to the correct residue modulo 4 according to the new value of the variable vector.

Decoding: The equivalent cases are considered:

1) There are at least two cells of level less than $q-1$. Let $i_1(i_2)$ be the index of the leftmost (rightmost) cell having level less than $q-1$. Then, we decode
$$v_1 = x_{i_1}(\text{mod } 2), \quad v_2 = x_{i_2}(\text{mod } 2).$$
2) There is one cell of level less than $q-1$, and it is the i-th cell. Then we decode
$$v_1 = \lfloor (x_i(\text{mod } 4))/2 \rfloor, v_2 = (x_i(\text{mod } 4))(\text{mod } 2).$$
3) All cells have level $q-1$. We decode according to the previous rule with $x_i = q-1$.

For even values of q, the construction is very similar. Again, the last available cell represents two bits. However, now the value of the two bits, given by the last available cell, is their relative difference from the value of the two bits given by all other cells. We note that since q is even, a cell of level $q-1$ represents a bit of value 1 and not 0 as we had in the case of odd q. Furthermore, if we use the last available cell up to level $q-1$ then it will be impossible to distinguish which cell represents two bits in case all of them are at level $q-1$. Therefore, we use the last available cell only until level $q-2$. This construction is optimal as well.

Theorem 2: If there are n q-level cells, then the code described above guarantees at least $t = (n-1)(q-1) + \left\lfloor \frac{q-1}{2} \right\rfloor$ writes before erasing.

Proof: As long as there is more than one cell of level less than $q-1$, the cell state weight increases by one after each write. This may change only after at least $(n-1)(q-1)$ writes. Let us assume that there is only one available cell of level less than $q-1$ after $s = (n-1)(q-1)+j$ writes, where $j \geq 0$. Starting at this write, the different residues modulo 4 of this cell correspond to the four possible variable vectors. Therefore, at the s-th write, we also need to increase the level of the last available cell so it will correspond to the variable vector at the s-th write. For all succeeding writes, if we change the first bit then the cell level increases by two. If however we change the second bit then the increase in the cell level alternates between one and three. Hence, if there are m writes to the last available cell, then the cell level increases by at most $2m+1$. If the initial level of the last available cell, before it starts representing the two bits together and after updating its level to correspond to the variable vector at the s-th write, is x, then there are at least $\lfloor (q-1-x)/2 \rfloor$ more writes. Next, we consider all possible options for the values of j and the variable vector at the s-th write in order to calculate the number of guaranteed writes before erasing.

1) Suppose $j(\text{mod } 4) = 0$, the value of both bits is 0, and the level of the last available cell does not increase at the s-th write. Hence, there are at least $\lfloor (q-1-j)/2 \rfloor$ more writes and a total of at least $(n-1)(q-1)+j+\lfloor (q-1-j)/2 \rfloor \geq (n-1)(q-1)+\lfloor (q-1)/2 \rfloor$ writes.
2) Suppose $j(\text{mod } 4) = 1$, one of the bits has value 1 and the other one 0. If the variable vector is $(v_1, v_2) = (0,1)$ then at the s-th write the level of the last available cell does not increase and if it is $(v_1, v_2) = (1,0)$ then its level increases by one. There are at least $\lfloor (q-1-(j+1))/2 \rfloor$ more writes, where $j \geq 1$ and a total of at least $(n-1)(q-1)+j+\lfloor (q-2-j)/2 \rfloor \geq (n-1)(q-1)+\lfloor (q-1)/2 \rfloor$ writes.
3) Suppose $j(\text{mod } 4) = 2$, the value of both bits is 0 and we increase the level of the last available cell by two at the s-th write. Therefore, there are at least $\lfloor (q-1-(j+2))/2 \rfloor$ more writes, where $j \geq 2$ and a total of at least $(n-1)(q-1)+j+\lfloor (q-3-j)/2 \rfloor \geq (n-1)(q-1)+\lfloor (q-1)/2 \rfloor$ writes.
4) Suppose $j(\text{mod } 4) = 3$, one of the bits has value 1 and the other one 0. If the variable vector is $(v_1, v_2) = (0,1)$ then the level of the last available cell increases by two, and if it is $(v_1, v_2) = (1,0)$ then we increase by three the level of the last available cell at the s-th write. Thus, there are at least $\lfloor (q-1-(j+3))/2 \rfloor$ more writes, where $j \geq 3$ and a total of at least $(n-1)(q-1)+j+\lfloor (q-4-j)/2 \rfloor \geq (n-1)(q-1)+\lfloor (q-1)/2 \rfloor$ writes.

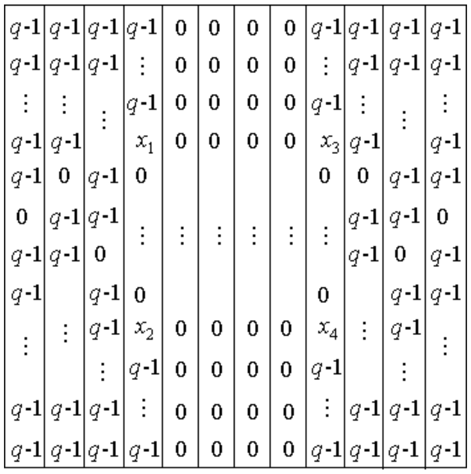
In any case, the guaranteed number of writes is $(n-1)(q-1) + \left\lfloor \frac{q-1}{2} \right\rfloor$. ■

III. Basic Multidimensional Construction

In this section we start the discussion of how to store an arbitrary number of bits. We demonstrate a basic construction for representing the bits inside a multidimensional box. The main drawback of this construction is its relatively high write deficiency that depends on the number of cells. In the next section we will show an alternative construction with a better deficiency.

Assuming we want to store four bits using n q-level cells. We represent the memory as a matrix of $n_1 \times n_2$ cells, where $n_1 n_2 = n$. In each column two bits are stored. The first and second bits are stored using the left columns. The leftmost column is used first, then the second leftmost and so on. Similarly, the third and fourth bits are stored using the right columns right-to-left. In each column we store the bits from the opposite directions as in the previous two-bits construction. However, in this case we don't use the last available cell to represent two bits, but leave it as a separation cell. Assuming we change the value of one of the first two bits, if it is possible to update this change in the current column that represents these bits, we do so. Otherwise, and if there is at least one more column for separation, we use the next column. An example of the memory state of this construction is demonstrated in Figure 2(a). The worst case scenario for the number of writes before erasing occurs when:

1) One column is used for separation.
2) Another column is only partially used and represents one write operation. That is, there was only one write to this column and still it is impossible to update the current write in this column.
3) In two other previous columns there is one more cell that is also only partially used and represents one write operation.


(a)

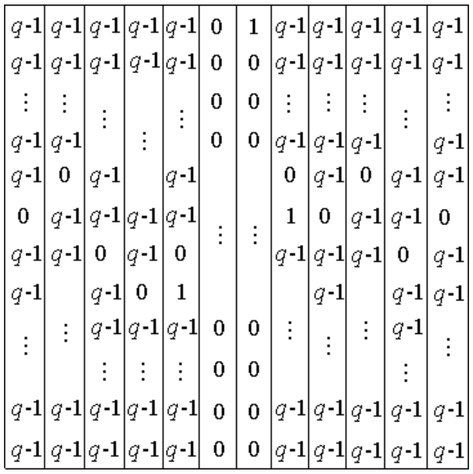

(b)

Fig. 2. Figure 2(a) describes an example of the memory state for the basic multidimensional construction, and Figure 2(b) demonstrates an example for the worst case scenario of the guaranteed number of writes when it is impossible to write the first bit.

Hence, the guaranteed number of writes is $(n_1-1)(n_2-1)(q-1)-((n_1-1)(q-1)-1)-2\left((q-1)-1\right)$, where $n_1n_2 = n$. Another example of the memory state that corresponds to the worst case scenario is given in Figure 2(b).

The generalization of this construction to three dimensions supports the storage of up to eight bits. Each plane stores four bits. The lower planes represent the first four bits and the upper planes represent the last four bits. In each plane we use the previous construction in order to represent four bits. We can use all the columns in each plane except for one that is left for separation between the two groups of two bits in this plane. Also, one more plane is used for separation between the two groups of four bits. The equivalent worst case scenario for the number of writes before erasing occurs as follows:

1) One plane is used for separation.
2) Another plane is partially used and represents only one write operation.
3) In two previous planes there is one more column that represents only one write and two more cells that represent only one write as well.

Therefore, the guaranteed number of writes is $(n_1-1)(n_2-1)(n_3-1)(q-1)-((n_1-1)(n_2-1)(q-1)-1)-2\left((n_1-1)(q-1)-1\right)-4((q-1)-1)$, where $n_1n_2n_3 = n$.

In general, using a D-dimensional box we can store 2^D bits, and we can show that the number of guaranteed writes is

$$\prod_{i=1}^{D}(n_i-1)(q-1)-\sum_{i=1}^{D-1}2^{i-1}\left(\prod_{j=1}^{D-i}(n_j-1)(q-1)-1\right)$$
$$-2^{D-1}\left((q-1)-1\right),$$

where $n_1n_2\cdots n_D = n$. It is also possible to show that this code is asymptotically optimal. However, its deficiency depends on n and therefore is not close enough to the lower bound on the deficiency. Next, we will show how to modify this construction in order to obtain a deficiency that is only dependent on the number of bits k and the number of cell levels q.

IV. Enhanced Multidimensional Construction

The last construction of flash codes demonstrates the idea of how to use a multidimensional box in order to represent multiple bits. Even though its asymptotical behavior is optimal, there is still a large gap between its write deficiency and the lower bound on the deficiency. The high deficiency mainly results from the separation cell in each column, the separation column in each plane, and in general from the separation hyperplane in each dimension.

The following construction of flash codes shows how to improve the deficiency. In order to reduce the deficiency from the extra separation hyperplane in the last dimension, the length of each dimension, besides the last one, should be as small as possible, for example we want to choose $n_i = 2, 3$ for $1 \le i \le D-1$. We also want to store the bits differently so that in each dimension it is possible to take advantage of all cell levels before using the next dimension. Another advantage of using small dimension lengths is that the rate of the code (defined as $k/(n\log_2 q)$) is enhanced as well. We show a construction where each dimension length, other than the last one, is two. If we want to store $k = 2^D$ binary bits then we show how to store 2^{D-1} of them inside $(D-1)$-dimensional boxes of size $n_1 \times n_2 \times \cdots \times n_{D-1} = 2\times 2\times\cdots\times 2$. Then, we use a D-dimensional box of size $n_1 \times n_2 \times \cdots \times n_D = 2\times 2\times\cdots\times 2\times n_D$, where $n_D \ge 3$, which consists of n_D $(D-1)$-dimensional boxes. For convenience, we call every multidimensional box, of any dimension, whose edges are of length 2, a *block*.

The construction is recursive. We first present how to store two bits using two-cell blocks. Then, we use this construction in order to store four bits using two-dimensional four-cell blocks of size 2×2. Using the four-bits construction, it is possible to store eight bits in a three-dimensional eight-cell blocks of size $2\times 2\times 2$. In general, the construction for storing 2^{i-1} bits in $(i-1)$-dimensional blocks, where $i \ge 3$, is utilized in order to store 2^i bits in i-dimensional blocks of 2^i cells each. We show in detail the basic constructions for storing two and four bits as these constructions are the building blocks for the arbitrary recursive construction. An analysis of the construction deficiency is given as well and it is shown that the write deficiency order is $O(k^2q)$.

Like the two-bits construction, this construction is presented for odd values of q, and it is possible to modify it in order to support even values. However, the deficiency is larger when q is even.

A. Two-Bits Construction

Our point of departure for these codes is a basic construction for storing two bits using blocks of two cells.

Encoding:

1) As long as the number of writes in the block is no greater than $q-1$:
   a) If the first bit is changed then the left cell is raised by one.
   b) If the second bit is changed then the right cell is raised by one.
2) Starting at the q-th write, it may happen that for some cell state vectors of the block, it is possible to write only one of the bits. In this case, if the other bit is changed then a new block is used.
   a) If the cell state vector of the block is of the form $(q-1, x)$, where $x < q-1$, then only the first bit can be written to this block, and the level of the second cell is raised by one.
   b) If the cell state vector of the block is of the form $(x, q-1)$, where $x < q-1$, then only the second bit can be written at the next write, and the level of the first cell is raised by one.
   c) If the cell state vector of the block is of the form (x_1, x_2), where $x_1 < q-1, x_2 < q-1$, then both bits can be written at the next step. If the first (second) bit is changed then the second (first) cell is raised by one.

Decoding:

1) If the cell state vector is of the form (x_1, x_2), where $0 \le x_1, x_2 \le q-1$, and $x_1 + x_2 \le q-1$ then the variable vector is
$$(v_1, v_2) = (x_1 (\text{mod } 2), x_2 (\text{mod } 2)).$$
2) If the cell state vector is of the form (x_1, x_2), where $0 \le x_1, x_2 \le q-1$, and $x_1 + x_2 > q-1$ then the variable vector is
$$(v_1, v_2) = (x_2 (\text{mod } 2), x_1 (\text{mod } 2)).$$

An example of this construction for $q = 5$ is given in Figure 3. We note that as long as the number of writes is no greater than $q-1$ then it is possible to write both bits. Only at the q-th write might it happen that writing will continue in the next block.

By abuse of terminology we use the following definitions for a block of any size:

1) A block is called *empty* if all its cells are at level zero.
2) A block is called *full* if all its cells are at level $q-1$.

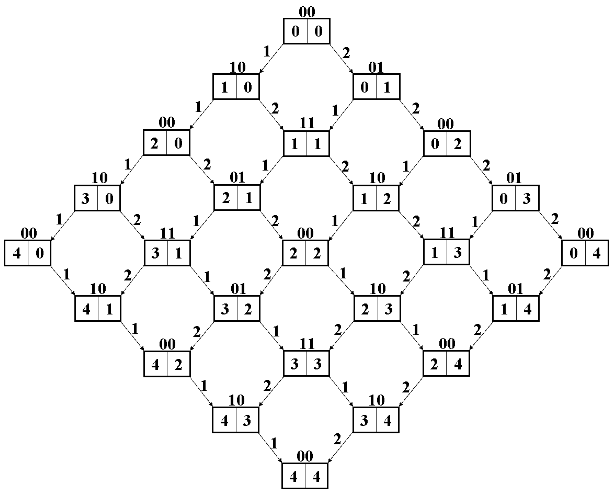


Fig. 3. State diagram for flash codes storing two bits in a block of two 5-level cells. The numbers in each block, above the block, and next to each edge represent the cell state vector, variable vector, and the written bit, respectively.

3) A block is called *active* if it is neither empty nor full.

Lemma 1: For the two-bits construction, at any write operation, there are at most two active blocks and at most
$$A_1 = (q-1) + 2(q-1) - 1 = 3(q-1) - 1$$
levels that are not used in these two blocks.

Proof: If a new block is used then the previous block uses at least $q-1$ levels, and there are no more active blocks. Therefore, at most $q-1$ levels are not used in the previous block and $2(q-1)-1$ in the new block. ■

B. Four-Bits Construction

Next, the four-bits case is considered. We use blocks of four cells. Each block is divided into two sub-blocks of two cells each, and each such sub-block is a column that stores two bits according to the previous construction. The block can either store the first and second bits together or the third and fourth bits together, i.e., it is impossible to store all four bits together in the same block. If the block stores the first and second bits then the sub-blocks are written left-to-right, and if the block stores the third and fourth bits then the sub-blocks are written right-to-left.

$$1,2 \rightarrow \begin{array}{|c||c|}\hline * & * \\ * & * \\ \hline\end{array} \leftarrow 3,4.$$

In this construction we have another distinction between storing the first and second bits and the third and fourth bits. In each sub-block we represent two bits according to the previous construction, but the order in which the bits are written in the sub-blocks is changed. If the block represents the first and second bits then both sub-blocks represent the bits in the same way:

$$\begin{array}{|c||c|}\hline 1 & 1 \\ 2 & 2 \\ \hline\end{array}$$

However, if the block represents the third and fourth bits then the representation order of the bits in the two sub-blocks is changed as follows:

4	3
3	4

Encoding:

1) The block can either represent the bits $1, 2$ or the bits $3, 4$. In each sub-block (column) we use the previous construction in order to represent two bits.
2) If the block represents the bits $1, 2$, then we write the two sub-blocks left-to-right, and the bits are stored similarly in the two sub-blocks.
3) If the block represents the bits $3, 4$, then we write the two sub-blocks right-to-left. For the right sub-block we represent the two bits where the third (fourth) bit is considered to be the first (second) bit in the two bits construction,
$$3 \leftrightarrow 1, 4 \leftrightarrow 2.$$
However, for the left sub-block we change the order of the bits, i.e., the third (fourth) bit is considered to be the second (first) bit in the two-bits construction,
$$3 \leftrightarrow 2, 4 \leftrightarrow 1.$$
4) If the block represents the bits $1, 2$ $(3, 4)$ then the right (left) sub-block cannot be full before the left (right) sub-block is full.

Decoding:

1) For every active block we first determine, according to the encoding rules, whether it represents the bits $1, 2$ or $3, 4$:
   a) If the right (left) sub-block is empty then the block represents the bits $1, 2$ $(3, 4)$.
   b) If the left (right) sub-block is full then the block represents the bits $1, 2$ $(3, 4)$.
   c) If both sub-blocks are active then according to the decoding procedure of the two-bits construction we can decide for each sub-block whether it is in a state that enables to write both bits, only the first bit or only the second bit.
      i) If the right (left) sub-block is in a state that enables to write both bits then the block represents the bits $1, 2$ $(3, 4)$.
      ii) Assuming the states of both sub-blocks enable to write only one bit. If both bits from the two sub-blocks are first bits or both bits are second bits then the block represents the bits $3, 4$. Otherwise, it represents the bits $1, 2$.
2) If the block represents the bits $1, 2$ or $3, 4$, we can decode their values from the two sub-blocks using the decoding procedure of the two-bits construction.
3) The value of each bit is the XOR of its values from all blocks.

An example of this construction for $q = 3$ is given in Figure 4.

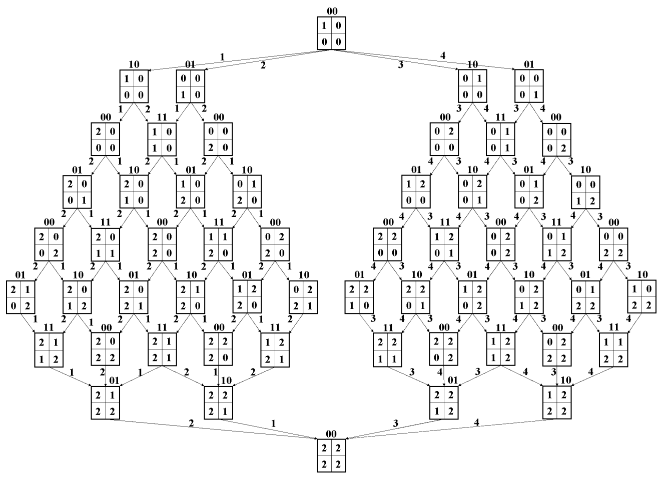


Fig. 4. State diagram for flash codes storing four bits in a four-cell block. The numbers in each block, above the block, and next to each edge represent the cell state vector, variable vector, and the written bit, respectively

Lemma 2: For the four-bits construction, after any write operation, there are at most four active blocks and at most
$$A_2 = 2\left((q-1) + 1 + 4(q-1) - 1\right) = 10(q-1)$$
levels that are not used in these four blocks.

Proof: For each pair of bits there are at most two active blocks, a new block and a previous one that could not be full before starting the new block. In the previous block at most $(q-1)+1$ levels are not used, corresponding to the case that the first sub-block does not use $(q-1)$ levels and as a result the other sub-block does not use one level. For example, for the block representing the first and second bits, this block is of the form:

$q-1$	$q-2$
0	$q-1$

In the new block at most $4(q-1)-1$ levels are not used. Therefore, there are at most four active blocks and at most
$$A_2 = 2 \cdot \left((q-1) + 1 + 4(q-1) - 1\right) = 10(q-1)$$
levels that are not used in these blocks. ■

C. Construction for Arbitrary Number of Bits

We are now ready to present the general construction of flash codes storing an arbitrary number of bits. First, we briefly describe how to represent eight bits and then give the general construction.

In order to store eight bits we use a block of eight cells which is a three dimensional box of size $2 \times 2 \times 2$. In fact, the block consists of two sub-blocks of four cells each that can be considered as two concatenated sub-blocks of size 2×2.

*	*	*	*
*	*	*	*

Each block can either represent the bits $1, \ldots, 4$ or $5, \ldots, 8$ according to the following order:

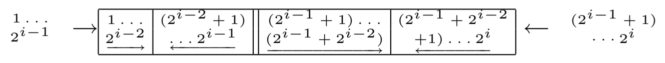
$$1, \ldots, 4 \rightarrow \begin{array}{|c|c||c|c|} \hline \overset{1,2}{\longrightarrow} & \overset{3,4}{\longleftarrow} & \overset{5,6}{\longrightarrow} & \overset{7,8}{\longleftarrow} \\ * & * & * & * \\ \hline \end{array} \leftarrow 5, \ldots, 8.$$

The bits $1, \ldots, 4$ $(5, \ldots, 8)$ write the two 2×2 sub-blocks left-to-right (right-to-left). Each sub-block represents four bits according to the four-bits construction. More rules are used to decide whether the code represents the bits $1, \ldots, 4$ or $5, \ldots, 8$, and are described in detail below for the arbitrary case. By abuse of terminology, a block of four cells having a cell state weight which is greater than $3(q-1)-2$ will be called *almost full.*

For representing 2^i bits, we assume that there is a construction for storing 2^{i-1} bits in blocks of 2^{i-1} cells. We use blocks of 2^i cells that consist of two sub-blocks of 2^{i-1} cells. We assume that for the 2^{i-1}-bits construction there are at most 2^{i-1} active blocks and at most A_{i-1} levels that are not used in these blocks.

Encoding:

1) The block can either represent the bits $1, \ldots, 2^{i-1}$ or the bits $(2^{i-1}+1), \ldots, 2^i$ according to the following order:



$1 \ldots 2^{i-1}$
$1 \ldots 2^{i-2}$
$(2^{i-2}+1) \ldots 2^{i-1}$
$(2^{i-1}+1) \ldots (2^{i-1}+2^{i-2})$
$(2^{i-1}+2^{i-2}+1) \ldots 2^i$
$(2^{i-1}+1) \cdots 2^i$

   In each sub-block, 2^{i-1} bits are represented according to the recursive construction.
2) Assuming the block represents the bits $1, \ldots, 2^{i-1}$,
   a) The sub-blocks are written left-to-right.
   b) The 2^{i-1} bits are stored similarly in the two sub-blocks using the recursive 2^{i-1}-bits construction.
   c) Each four-cell block located in the right sub-block can be almost full only if one of the four-cell blocks in the left sub-block has cell state weight greater than the cell state weight of the four-cell block in the right sub-block.
3) Assuming the block represents the bits $(2^{i-1}+1), \ldots, 2^i$,
   a) The sub-blocks are written right-to-left.
   b) The 2^{i-1} bits are also stored using the recursive 2^{i-1}-bits construction. For the right sub-block we represent the 2^{i-1} bits $(2^{i-1}+1), \ldots, 2^i$ as if they were the bits $1, \ldots, 2^{i-1}$, where the $(2^{i-1}+j)$ -th bit, $1 \leq j \leq 2^{i-1}$ is considered to be the j-th bit. However, for the left sub-block we change the order of the bits. The $(2^{i-1}+j)$-th bit is considered to be the $(2^{i-2}+j)$-th bit for $1 \leq j \leq 2^{i-2}$, and the $(2^{i-1}+j)$-th bit is considered as the $(j-2^{i-2})$-th bit for $2^{i-2}+1 \leq j \leq 2^{i-1}$.
   c) Each four-cell block located in the left sub-block can be almost full only if one of the four-cell blocks in the right sub-block has cell state weight greater than the cell state weight of the four-cell block in the left sub-block.

Decoding:

1) For every active block we first determine, according to the encoding rules, which group of 2^{i-1} bits it represents.
   a) If the right (left) sub-block is empty then the block represents the bits $1, \ldots, 2^{i-1}$ $((2^{i-1}+1), \ldots, 2^i)$.
   b) If the left (right) sub-block is full then the block represents the bits $1, \ldots, 2^{i-1}$ $((2^{i-1}+1), \ldots, 2^i)$.
   c) If both sub-blocks are active and all their four-cell blocks are not almost full then according to the decoding procedure of the 2^{i-1}-bits construction we can decode for each sub-block whether it represents the 2^{i-2} bits $1, \ldots, 2^{i-2}$ or $(2^{i-2}+1), \ldots, 2^{i-1}$.
      i) If the left and right sub-blocks represent different groups of 2^{i-2} bits then the block represents the bits $1 \cdots 2^{i-1}$.
      ii) If the left and right sub-blocks represent the same groups of 2^{i-2} bits then the block represents the bits $(2^{i-1}+1), \ldots, 2^i$.
   d) Assume both sub-blocks are active and one of them has a four-cells block that is almost full. If the four-cell block of maximum cell state weight in the left sub block is greater than the four-cell block of maximum cell state weight in the right sub-block then the block represents the bits $1, \ldots, 2^{i-1}$, and otherwise it represents the bits $(2^{i-1}+1), \ldots, 2^i$.
2) If the block represents the bits $1, \ldots, 2^{i-1}$ or $(2^{i-1}+1), \ldots, 2^i$, we can decode their value from the two sub-blocks using the decoding procedure of the 2^{i-1}-bits construction.
3) The value of each bit is the XOR of its values from all blocks.

Lemma 3: For flash codes storing 2^i bits, after any write operation, there are at most 2^i active blocks and at most

$$A_i = 2A_{i-1} + \frac{q+1}{8}4^i + \frac{3q-5}{4}2^i$$

levels that are not used in these blocks.

Proof: Each of the blocks that represents the bits $1, \ldots, 2^{i-1}$ can be considered as a pair of sub-blocks containing 2^{i-1} cells, such that each sub-block represents the bits $1, \ldots, 2^{i-1}$. According to the recursive construction at most 2^{i-1} sub-blocks are active and at most A_{i-1} levels are not used in these sub-blocks. If all these sub-blocks happen to be the left ones in their containing blocks then there are 2^{i-1} more sub-blocks that are active, which are the corresponding right sub-blocks in each block. At most one of them may be empty and for the others, at most $(q-1)+2$ levels from each block of four cells are not used. Hence, in these sub-blocks at most

$$\begin{aligned} B_i &= (2^{i-1}-1)\frac{2^{i-1}}{4}((q-1)+2) + 2^{i-1}(q-1) \\ &= \frac{q+1}{16}4^i + \frac{3q-5}{8}2^i. \end{aligned}$$

levels are not used. The same analysis is applied to the blocks that represent the bits $(2^{i-1}+1), \ldots, 2^i$. Therefore, for all the bits, there are at most 2^i active blocks and at most

$$A_i = 2A_{i-1} + 2B_i = 2A_{i-1} + \frac{q+1}{8}4^i + \frac{3q-5}{4}2^i$$

levels that are not used in these blocks. ■

D. Deficiency Analysis

Lemma 4: For $i \geq 2$ we have

$$A_i = \frac{q+1}{4}4^i + \frac{3q-5}{4}i2^i - 2^i.$$

Proof: We prove the correctness of the expression for A_i by induction. According to Lemma 2, we have $A_2 = 10(q-1)$ which is also given by this expression. Assuming $A_{i-1} = \frac{q+1}{4}4^{i-1} + \frac{3q-5}{4}(i-1)2^{i-1} - 2^{i-1}$, for $i \geq 3$, then according to Lemma 3

$$\begin{aligned} A_i &= 2A_{i-1} + \frac{q+1}{8}4^i + \frac{3q-5}{4}2^i \\ &= 2\left(\frac{q+1}{4}4^{i-1} + \frac{3q-5}{4}(i-1)2^{i-1} - 2^{i-1}\right) \\ &+ \frac{q+1}{8}4^i + \frac{3q-5}{4}2^i = \frac{q+1}{4}4^i + \frac{3q-5}{4}i2^i - 2^i. \end{aligned}$$

■

Theorem 3: If the flash codes represent $k = 2^D$ bits then the deficiency δ_D satisfies

$$\delta_D = 2A_{D-1} + 1 = \frac{q+1}{8}k^2 + \frac{3q-5}{4}k\log k - \frac{3q-1}{4}k + 1.$$

Proof: In order to represent $k = 2^D$ bits a D-dimensional box of size $2 \times 2 \times \cdots \times 2 \times n_D$ is used. The multidimensional box can be considered as an array of n_D $(D-1)$-dimensional boxes, which we call them blocks. The first 2^{D-1} bits are represented using the blocks left-to-right and the last 2^{D-1} bits use the blocks right-to-left, and there is always one block for separation. In each $(D-1)$-dimensional box we use the construction to represent 2^{D-1} bits. Writing stops if we need to start using a new block but this is the last separation block. According to Lemma 3 there are at most 2^{D-1} active blocks for each group of 2^{D-1} bits and at most A_{D-1} levels that are not used in these blocks for each group of cells. Also, it is impossible to use the last separation block, and hence at most $2A_{D-1} + 1$ levels are not used in the worst case, where

$$\begin{aligned} &2A_{D-1} + 1 \\ &= 2\left(\frac{q+1}{4}4^{D-1} + \frac{3q-5}{4}(D-1)2^{D-1} - 2^{D-1}\right) + 1 \\ &= \frac{q+1}{8}4^D + \frac{3q-5}{4}D2^D - \frac{3q-1}{4}2^D + 1 \\ &= \frac{q+1}{8}k^2 + \frac{3q-5}{4}k\log k - \frac{3q-1}{4}k + 1. \end{aligned}$$

■

For even values of q, we consider every two cells of level q as one cell of level $q' = 2q-1$, and we can apply the construction for odd values of q. The code deficiency becomes $\frac{q}{4}k^2 + \frac{3q-4}{2}k\log k - \frac{3q-2}{2}k + 1$.

V. Conclusion

In [12], the problem of coding to minimize block erasures in flash memories was first presented. In this work we show an optimal construction of flash codes for storing two bits. We believe that our construction is simpler than an earlier optimal construction presented in [12]. Our main contribution is an efficient construction of codes that support the storage of any number of bits. We show that the order of the code deficiency is $O(k^2q)$, which is an improvement upon the equivalent construction in [13]. The upper bound in [12] on the guaranteed number of writes implies that the order of the lower bound on the deficiency is $O(kq)$. Therefore, there is a gap, which we believe can be reduced, between the write deficiency orders of our construction and the lower bound.

References

[1] V. Bohossian, A. Jiang, and J. Bruck, "Buffer coding for asymmetric multi-level memory," in *Proceedings IEEE International Symposium on Information Theory*, Nice, France, June 2007.

[2] P. Cappelletti, C. Golla, P. Olivo, and E. Zanoni (Editors), *Flash memories*, Boston: Kluwer Academic, 1999.

[3] Y. Cassuto, M. Schwartz, V. Bohossian, and J. Bruck, "Codes for multi-level flash memories: correcting asymmetric limited-magnitude errors," in *Proceedings IEEE International Symposium on Information Theory*, Nice, France, June 2007.

[4] G.D. Cohen, P. Godlewski, and F. Merkx, "Linear binary code for write-once memories," *IEEE Trans. Inform. Theory,* vol. 32, pp. 697-700, October 1986.

[5] B. Eitan and A. Roy, "Binary and multilevel flash cells," in Flash Memories, P. Cappelletti, C. Golla, P. Olivo, E. Zanoni Eds. Kluwer, pp. 91-152, 1999.

[6] A. Fiat and A. Shamir, "Generalized write-once memories," *IEEE Trans. Inform. Theory,* vol. 30, pp. 470–480, September 1984.

[7] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Computing Surveys*, vol. 37, no. 2, pp. 138–163, June 2005.

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

[9] M. Grossi, M. Lanzoni, and B. Ricco, "Program schemes for multilevel flash memories," in *Proceedings of the IEEE*, vol. 91, no. 4, pp. 594-601, April 2003.

[10] A. Jiang, "Information storage in flash memories with floating codes," in *Proceedings 45-th Annual Allerton Conference on Communication, Control and Computing*, Monticello, IL, September 2007.

[11] A. Jiang, "On the generalization of error-correcting WOM codes," in *Proceedings IEEE International Symposium on Information Theory*, Nice, France, June 2007.

[12] A. Jiang, V. Bohossian, and J. Bruck, "Floating codes for joint information storage in write asymmetric memories," in *Proceedings IEEE International Symposium on Information Theory*, Nice, France, June 2007.

[13] A. Jiang and J. Bruck, "Joint coding for flash memory storage," in *Proceedings IEEE International Symposium on Information Theory*, Toronto, Canada, July 2008.

[14] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for flash memories," in *Proceedings IEEE International Symposium on Information Theory*, Toronto, Canada, July 2008.

[15] A. Jiang, M. Schwartz, and J. Bruck, "Error-correcting codes for rank modulation," in *Proceedings IEEE International Symposium on Information Theory*, Toronto, Canada, July 2008.

[16] D. Kahng and S.M. Sze, "A floating-gate and its application to memory devices," *Bell Systems Tech. J.*, vol. 46, no. 4, pp. 1288-1295, 1967.

[17] M. Mitzenmacher, Z. Liu, and H. Finucane, "Designing floating codes for expected performance," in *Proceedings 46-th Annual Allerton Conference on Communication, Control and Computing*, Monticello, IL, September 2008.

[18] R.L. Rivest and A. Shamir, "How to reuse a write-once memory," *Information and Control*, vol. 55, nos. 1–3, pp. 1–19, December 1982.

[19] B. Van Zeghbroeck, *Principles of semiconductor devices*, e-book published online at *ece-www.colorado.edu/ bart/book*, 1997.