Short Q-ary WOM Codes with Hot/Cold Write Differentiation

Yuval Cassuto
Technion – Israel Institute of Technology
Electrical Engineering Department
ycassuto@ee.technion.ac.il

Eitan Yaakobi
California Institute of Technology
EE Department
Univ. of California, San Diego
ECE Department
yaakobi@caltech.edu

***Abstract*— We construct new WOM codes with practical design considerations. First the problem of 2 cell q-ary WOM codes is addressed with a construction that uses lattice tilings. The resulting codes for arbitrary numbers of input bits are shown to be within a small additive constant from the capacity. Then we introduce a new model of WOM codes that support data bits with different update requirements. Differentiation between frequently written (hot) bits and rarely written (cold) ones allows a large number of re-writes while leveling the wear between the hot and cold input bits.**

I. Introduction

Storage media that are constrained to change their physical stored levels in one direction have inspired a significant body of work to allow unconstrained writes to such media. The first work in that area introduced re-write codes for write-once memories (WOM) [8]. In the WOM model, k input bits are written t times to n physical cells with q levels, where the cell levels cannot decrease between writes. The WOM model has received significant attention recently thanks to its applicability to the ubiquitous flash storage technology [3], [6]. Alongside the principal WOM model, other interesting re-write models have been proposed and studied with considerable success [4], [5], [10]. The starting point of this work are observations we make on the challenges of WOM codes with respect to their usage in realistic storage environments.

1) Redundancy-efficient WOM constructions often have a different k for each of the t writes. This property is hard to accommodate in practice.
2) Long WOM codes with little structure mean exponentially growing decoding complexities.
3) The generalization of binary WOM codes to q-ary cells is not well established yet.
4) The known models assume that all user bits have the same access characteristics, and therefore may be wasteful in redundancy.

These challenges are the main motivators to the current work, which addresses the challenges above as follows.

1) A fixed k number of bits in each of the t writes is sought by all constructions.
2) A small number (e.g. 2) of q-ary cells are used for the codes.
3) The codes distinguish between "hot" and "cold" bits in the number of updates they allow (hot bits are updated frequently, cold bits are updated rarely).

More concretely, in Section II we study WOM codes with $n = 2$ q-ary cells that use lattice tilings to obtain large numbers of writes. For a general k these codes are shown to be within an additive constant from the WOM capacity. We note that tilings have been proposed for use in re-write codes, but not for the fixed k WOM model [7]. Then in Section III we introduce a new model of WOM codes that support data bits with different update requirements. Frequently updated bits are assigned to *hot* input bits that can be updated multiple times. Rarely updated bits are assigned to *cold* input bits that are allowed to be written only once, but at any time in the write sequence. Several constructions with different parameters show that differentiation between hot and cold bits can significantly improve the re-write capabilities of the code.

II. WOM Codes with Two Cells

As a preparation to discuss WOM codes with two cells ($n = 2$), we start with the simple case of re-writing using one cell ($n = 1$). When there is only one cell, any new value of the k input bits has to result in a distinct level increment of the cell between 0 and $2^k - 1$. Therefore, it is clear that the number of writes that can be guaranteed with a single cell is $t = \lfloor (q-1)/(2^k-1) \rfloor$, and no greater. The special case of $k = 1$ gives $t = q - 1$ writes by incrementing the level by 1 each time the input bit changes $0 \to 1$ or $1 \to 0$ [4]. Since the case of $n = 1$ is completely characterized, we move to discuss the case of $n = 2$. With $n = 2$ cells, the WOM problem becomes interesting as early as $k = 3$, a case we study next.

A. Storing 3 bits in 2 cells

In an $n = 2$ code, the physical content of the memory is described by a pair $(c_1, c_2) \in \{0, \ldots, q-1\}^2$ of cell levels. The information content is represented by an integer number $v \in \{0, 1, \ldots, 2^k - 1\}$ or $\{0, 1, \ldots, 7\}$ for $k = 3$. A mapping between integers and k-bit vectors is implicitly assumed. Reading information is then performed by a function $\psi(c_1, c_2)$, where $\psi : \{0, \ldots, q-1\}^2 \to \{0, 1, \ldots, 7\}$. Writing k bits to the physical cells is specified as a function μ of the current cell contents and the new information integer. Thus

$$(c_1', c_2') = \mu(c_1, c_2, v')$$

Such read and write functions for $k = 3$ are specified in Figure 1. The numbers inside the matrix stand for information integers in $\{0, 1, \ldots, 7\}$. The coordinates marked at the exterior of the matrix represent cell levels. The horizontal coordinate is c_1 and the vertical one is c_2. The reading function $\psi(c_1, c_2)$ is simply the content of the (c_1, c_2) position of the matrix. A write function $\mu(c_1, c_2, v')$ can be obtained from Figure 1 by defining (c_1', c_2') to be the nearest position that contains the number v', such that $c_1' \geqslant c_1$ and $c_2' \geqslant c_2$. For example, suppose the current cell levels are $(c_1, c_2) = (0, 2)$, storing the integer 4. Then a value $v' = 7$ is written by moving the

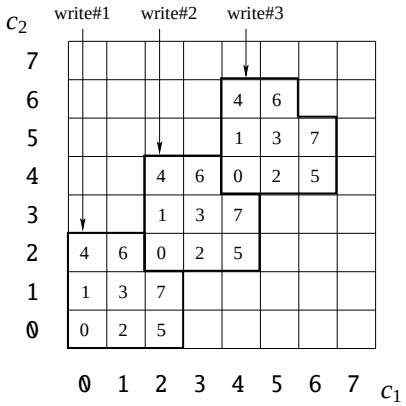

Figure 1. A code that stores 3 bits in 2 cells with $\lfloor (q-1)/2 \rfloor$ writes.

cells to levels $(c'_1, c'_2) = (4, 3)$. Each polygon of area 8 in Figure 1 specifies the range of possible cell levels (c'_1, c'_2) after a given write generation. Since the polygon for write i has both $c_1 \leqslant 2i$ and $c_2 \leqslant 2i$, Figure 1 specifies an $n = 2$, $k = 3$ WOM code with $t = \lfloor (q-1)/2 \rfloor$.

The code specified in Figure 1 provides re-write guarantees by stacking 2-dimensional shapes along the main diagonal of the (c_1, c_2) plane. The rest of the plane outside the diagonal stack remains unused. We thus raise the question of whether a better WOM code can be obtained by utilizing these remaining cell states. The construction to follow in the next sub-section answers this question to the affirmative.

B. $n = 2$ codes by 2-dimensional tilings

To get more writes from the 2-dimensional (c_1, c_2) plane we take the following steps:

1) Tile the plane with the same basic shape from Figure 1.
2) Specify update functions that traverse the tiling in a way that a certain number of writes is guaranteed for any sequence of input-value updates.

1. Lattice tiling for $k = 3$

Let the polygon of area 8 used in Figure 1 be defined formally as

$$S = \{(x, y) \mid 0 \leqslant x, y \leqslant 2\} \setminus \{(2, 2)\}.$$

Also define the center of S as the point $(0, 0)$. A tiling of $\mathbb{Z}^2$ by S is a pair (S, T), where T is a set of locations where centers of S copies are placed, such that the copies are disjoint and cover the entire $\mathbb{Z}^2$ plane [9]. A particularly convenient way to obtain T is by using a *lattice*, in which case (S, T) is called a *lattice tiling*. T is a lattice if its points can be written as

$$T = \{u_1 \boldsymbol{v}_1 + u_2 \boldsymbol{v}_2 \ : u_1, u_2 \in \mathbb{Z}\},$$

where $\{\boldsymbol{v}_1, \boldsymbol{v}_2\}$ are linearly independent vectors in $\mathbb{R}^2$, which are called the *base* for T. In other words, T is the set of linear combinations of $\{\boldsymbol{v}_1, \boldsymbol{v}_2\}$ with integer coefficients. The particular lattice we use to tile S is generated by the vectors $\boldsymbol{v}_1 = (2, 2)$ and $\boldsymbol{v}_2 = (3, -1)$, that is, its generator matrix is

$$G = \begin{pmatrix} 2 & 2 \\ 3 & -1 \end{pmatrix}.$$

Observe that the $\boldsymbol{v}_1 = (2, 2)$ vector is exactly the one used in the diagonal stacking of Figure 1. The vector $\boldsymbol{v}_2 = (3, -1)$ is now added to the base to form a complete tiling. The size of this lattice is $|\det(G)| = 8$, and the numbers between 0 and 7 of S are assigned to their respective locations in copies of S translated by T. A truncated version of the infinite tiling T is shown in Figure 2.

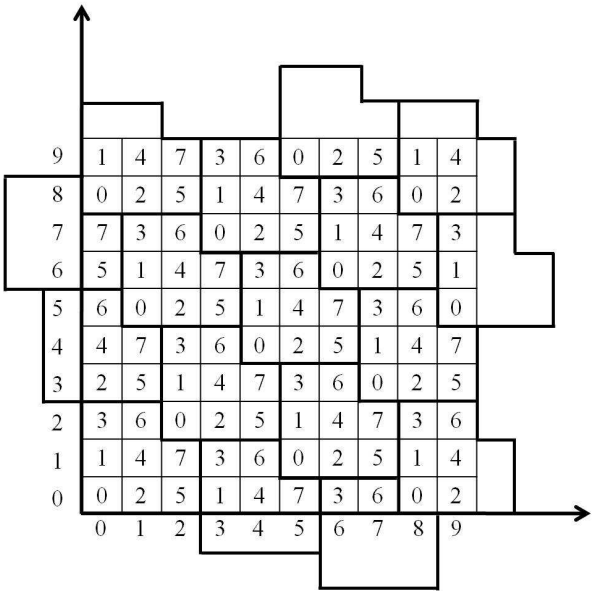

Figure 2. The tiling T used for an improved $n = 2$, $k = 3$ WOM code.

2. Decoding and update functions

According to the lattice tiling shown in Figure 2, every entry (c_1, c_2) in the two-dimensional array is assigned with a number $\psi(c_1, c_2) \in \{0, \ldots, 7\}$. This gives us the decoding procedure of the code. For the update procedure, given the current memory state (c_1, c_2) and an integer m in the range $0 \leqslant m \leqslant 7$ that represents the new bit values, the new memory state (c'_1, c'_2) satisfies the following conditions:

1) $(c'_1, c'_2) \geqslant (c_1, c_2)$.
2) $\psi(c'_1, c'_2) = m$.
3) (c'_1, c'_2) minimizes the value of $\max\{c''_1, c''_2\}$ among all the points (c''_1, c''_2) that satisfy conditions 1 and 2.

Lemma 1. *If $q = 8$, then the code guarantees four writes.*

Proof: Let us consider the four writes of this code, and let $(\delta_{i,1}, \delta_{i,2})$ be the cell-level increments in each write for $i = 1, 2, 3, 4$ such that the final cell level is

$$(c_1, c_2) = \sum_{i=1}^{4} (\delta_{i,1}, \delta_{i,2}).$$

If there exists $1 \leqslant i \leqslant 4$ such that $\delta_{i,1} < 2$ then $c_1 \leqslant 7$ and similarly for c_2. Hence we only need to consider the case where for all $1 \leqslant i \leqslant 4$ $\delta_{i,1} = 2$ or for all $1 \leqslant i \leqslant 4$ $\delta_{i,2} = 2$. From the tile shape, increments of $\delta_{i,1} = \delta_{i,2} = 2$ are never needed, so at every write at most one of $\delta_{i,1}$ and $\delta_{i,2}$ equals 2. Assume without loss of generality that for all $1 \leqslant i \leqslant 4$ $\delta_{i,1} = 2$, then $\delta_{i,2} \leqslant 1$. Consider the last write, note that instead of the increment vector $(\delta_{4,1}, \delta_{4,2}) = (2, \delta_{4,2})$, we could use the increment vector $(1, \delta_{4,2} + 3)$ without violating the decoding rule. This is true because

$$(2, \delta_{4,2}) - (1, \delta_{4,2} + 3) = (1, -3) = \boldsymbol{v}_2 - \boldsymbol{v}_1.$$

and since $\boldsymbol{v}_2 - \boldsymbol{v}_1$ is a lattice point, the values of the points $(2, \delta_{4,2})$ and $(1, \delta_{4,2} + 3)$ are the same. Hence the final cell level will be in this case

$$(c_1, c_2) = (2, \delta_{1,2}) + (2, \delta_{2,2}) + (2, \delta_{3,2}) + (1, \delta_{4,2} + 3) \leqslant (7, 7).$$

■

In general, if there are q levels, then the code guarantees $t = \lfloor 4(q-1)/7 \rfloor$, which is better than the $t = \lfloor (q-1)/2 \rfloor$ of Figure 1.

C. $n = 2$ tiling codes for general k

We now generalize the construction from the previous subsection to general k. For that we use the two-dimensional corner shape, which is formalized as follows. Let a and b be positive integers such that $a > b$, then the two-dimensional corner $C(a, b)$ is given by the set

$$C(a, b) = \{(x, y) \mid 0 \leqslant x, y \leqslant a - 1\} \setminus \{(x, y) \mid b \leqslant x, y \leqslant a - 1\}.$$

Proposition 2. *(proof omitted) For all $a > b > 0$, a lattice tiling to the shape $C(a, b)$ is given by the vectors:*

$$\boldsymbol{v}_1 = (b, b), \quad \boldsymbol{v}_2 = (a, b - a).$$

Proposition 3. *The points $(a - 1, b - 1)$ and $(b - 1, a + b - 1)$ are equivalent (contain the same element).*

Proof: Note that

$$(a - 1, b - 1) - (b - 1, a + b - 1) = (a - b, -a) = \boldsymbol{v}_2 - \boldsymbol{v}_1.$$

■

Lemma 4. *Assume $a/(a - b) = c$ is a positive integer. Then there exists a q-ary WOM code, $q = c(a - 1) + b$ that writes a symbol of size $M = a^2 - (a - b)^2$, $t = c + 1$ times.*

Proof: The proof is essentially the same as that of Lemma 1, which is a special case $a = 3$, $b = 2$.

■

Suppose for an odd k we choose $a = 1.5 \cdot 2^{\frac{k-1}{2}}$, $b = 2^{\frac{k-1}{2}}$, $(c = 3)$, where k is an odd integer, then the number of messages (the tile size) is

$$M = a^2 - (a - b)^2 = (2a - b) \cdot b = 2 \cdot 2^{\frac{k-1}{2}} \cdot 2^{\frac{k-1}{2}} = 2^k.$$

The number of levels is

$$q = 3(a - 1) + b = 5.5 \cdot 2^{\frac{k-1}{2}} - 3$$

and by Lemma 4 this gives $t = 4$ writes. The total rate of this code (sum rate of the 4 writes) is $R = 4 \cdot k/2 = 2k$, where the denominator 2 is the number of cells used by the code. An upper bound on the rate for this q and $t = 4$ is [2]:

$$\begin{aligned} &\log_2 \binom{q+3}{4} < \log_2 \left(\frac{(5.5 \cdot 2^{\frac{k-1}{2}})^4}{24} \right) \\ &= 4 \cdot \frac{k-1}{2} + 4 \cdot \log_2 5.5 - \log_2 24 \\ &= 2k - 2 + 9.8377 - 4.585 = 2k + 3.2527. \end{aligned}$$

Hence, with only two cells we can already achieve a WOM code which is within at most an additive constant 3.2527 of the capacity. We note further that this bound is only known to be achievable with non equal-rate writes, while in our codes the rate is identical in all 4 writes.

III. Joint Storage of "Hot" and "Cold" Bits

We now move to study a new type of WOM codes, where part of the input bits are allowed to be written multiple times, and another part are only allowed a single write. The former are called *hot* bits and the latter are called *cold* bits. The motivation for this model comes from the need of solid-state storage devices to level the wear between frequently and rarely written data blocks, which requires to jointly store them on the same physical cells.

To understand the model, we start with a simple example. In Figure 3 we show the level transition diagram of a single cell code for one hot and one cold bit. The stored information bits appear at the bottom of the figure, the right of which (underlined) is the hot bit. Solid arrow lines represent changes in the hot bit, and dashed arrow lines represent re-writing the same value for the hot bit.

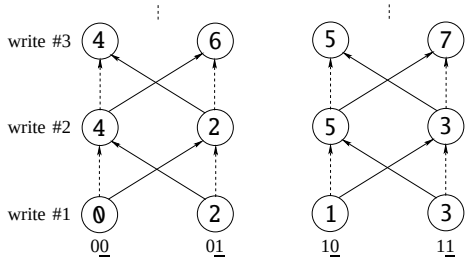

Figure 3. Code for hot+cold bit in 1 cell with $\lfloor q/2 \rfloor$ total writes.

Restricting the cold bit to up to one write allows a total number of writes that equals $t = \lfloor q/2 \rfloor$ (including the one cold write). This is better than the standard $n = 1$, $k = 2$ WOM code that gives only $t = \lfloor (q - 1)/3 \rfloor$ writes. It also gives one more write than an $n = 1$, $k = 2$ floating code [4]. Note that the t writes of the hot and cold bit can be performed in any order (moving to the right part of the diagram after a cold write can be done at any level). The following lemma states that the code of Figure 3 is optimal.

Lemma 5. *(proof omitted) Any one-cell code that stores a single hot and a single cold bit guarantees at most $\lfloor q/2 \rfloor$ writes.*

A. Two-cell hot+cold bit storage

We now detail a two-cell scheme to jointly store one hot and one cold bit. We start with specifying the decoding rule pictorially in Figure 4. As in the two-cell schemes of the previous section, the coordinates at the exterior of the matrix represent physical-cell levels, and the integers within the matrix are the information content of the stored bits: 0 stands for 0̲0, 1 stands for 0̲1, 2 stands for 1̲0 and 3 stands for 1̲1. The underlined bit is the hot bit that can change multiple times in the write sequence. The non-underlined bit is the cold bit that can be written once, at any point of the write sequence. To support unrestricted re-writing of the hot bit, the following transitions must be possible without decrease in physical-cell levels:

$$0 \to \{0, 1\}, \quad , 1 \to \{0, 1\}, \quad , 2 \to \{2, 3\}, \quad , 3 \to \{2, 3\},$$

In addition, a single transition of the form $0 \to 2$ or $1 \to 3$ must be supported.

The formal specification of the decoding and update rules are now given. Suppose the hot bit is denoted b_1 and the cold

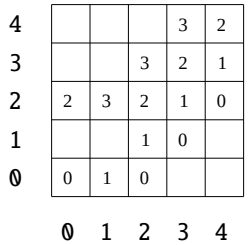

Figure 4. Code for 1 hot bit and 1 cold bit in 2 cells with $t = 2q-3$ writes, at most 1 of which is a cold-bit write.

bit is denoted b_2. The cold bit b_2 is written at most once and the goal is allow as many total writes using $n = 2$ cells. The levels of the two cells are denoted by c_1 and c_2.

The decoding rule $\mathcal{D}(c_1, c_2) = (b_1, b_2)$, previously described in Figure 4, now follows:

1) $\mathcal{D}(0,0) = (0,0)$.
2) For all $(c_1, c_2) \neq (0,0)$,
   a) $b_1 = (c_1 + c_2) (\mathrm{mod} 2)$,
   b) If $c_1 > c_2$ then $b_2 = 0$ and if $c_1 \leqslant c_2$ then $b_2 = 1$.

The encoding/update rule, $\mathcal{E}(c_1, c_2, i) = (c_1', c_2')$ is applied as follows: the current memory state is (c_1, c_2) and the bit index to be changed is $i \in \{1, 2\}$. We assume here that the second bit changes at most once and when a bit is rewritten, its value changes (otherwise, there is no need to change the memory state). Furthermore, $c_1' \geqslant c_1$, $c_2' \geqslant c_2$. The following rules constitute the encoder procedure:

1) If $i = 2$, then $(c_1', c_2') = (c_1, c_2 + 2)$.
2) If $i = 1$, then apply the following rules:
   a) If $c_1 = c_2 = 0$, then $(c_1', c_2') = (1, 0)$.
   b) If $c_1 = c_2 > 0$, then $(c_1', c_2') = (c_1, c_2 + 1)$.
   c) If $c_1 = c_2 + 2$, then $(c_1', c_2') = (c_1, c_2 + 1)$.
   d) If $c_1 = c_2 + 1$, then $(c_1', c_2') = (c_1 + 1, c_2)$.
   e) If $c_2 > c_1$, then $(c_1', c_2') = (c_1 + 1, c_2)$.

The rules become clearer when consulting Figure 4. The 5 cases in item 2 above correspond to the following, respectively.

a) 0 at the lower left corner changing to 1 by moving right.
b) 2 changing to 3 by going up one level.
c) 0 changing to 1 by going up one level.
d) 1 changing to 0 by going right one level.
e) 3 changing to 2 (or 2 changing to 3) by going right one level.

The following lemma will help proving the code properties.

Lemma 6.

1) When $b_2 = 0$, the hot bit b_1 can be updated by moving (c_1, c_2) to a state of the form $(x+1, x)$ or $(x+2, x)$.
2) When $b_2 = 1$, the hot bit b_1 can be updated by moving (c_1, c_2) to a state of the form (x, x) or $(x, x+1)$.

Proof: Immediate from Figure 4. ■

Now we prove the rewrite properties of the code.

Proposition 7. *The number of writes the code guarantees is $t = 2(q-1) - 1$ including the possible rewrite of the cold bit.*

Proof: From Lemma 6 part 1, if the cold bit never changes its value then writing stops when the memory state is $(q-1, q-2)$, after alternating +1 changes in c_1 and c_2. Thus, there are $q - 1 + q - 2 = 2(q-1) - 1$ writes. From Lemma 6 part 2, if the cold bit changes its value, then writing stops when the memory achieves state $(q-1, q-1)$, after $q - 1 + q - 3$ alternating +1 changes in c_1 and c_2 (for the hot bit) and a single +2 change in c_2 (for the cold bit). Thus, in that case too there are $q - 1 + q - 3 + 1 = 2(q-1) - 1$ writes. ■

Note that a trivial upper bound on the number of writes is $2(q-1)$. However, it is possible to show that the construction is strictly optimal.

Proposition 8. *Any code guarantees at most $2(q-1) - 1$ writes.*

Proof: Assume in the contrary that there exists a code which guarantees $2(q-1)$ writes. Let us consider $2(q-1) - 1$ writes where only the hot bit b_1 changes its value. Then, the memory state is either $(q-1, q-2)$ or $(q-2, q-1)$, and $b_1 = 1$. Without loss of generality, assume it is the first state. Therefore, the decoded value of the memory state $(q-1, q-2)$ is $(b_1, b_2) = (1, 0)$. On the following write, it is possible to change the two-bits value to either $(0,0)$ or $(1,1)$ but there is only one memory state that is accessible, $(q-1, q-1)$, which leads to a contradiction. ■

B. Multiple cold bits and a single hot bit

In this section we would like to extend the 2-cell hot+cold construction of the previous sub-section such that it will be possible to store multiple cold bits and a single hot bit. First, note that we can take k copies of the previous construction, that is, $2k$ cells. In every pair of cells, a single cold bit is stored and a single hot bit. Since we only need to store a single hot bit, its value is simply the sum of the k hot bits. Thus, it is possible to store a single hot bit and k cold bits in $n = 2k$ cells with $k(2q-3) = n(q-1) - k$ writes.

Next we show another example of such a construction with fewer cells $n = k + 1$. Let us denote the cells by $c_0, c_1, \ldots, c_k$. Our idea is similar to the one we just presented, with the cell c_0 acting as a mutual cell to all the other k cells. That is, every two cells of the form (c_0, c_i) for $1 \leqslant i \leqslant k$, generate a code of a single hot and a single cold bit. Let us now describe the encoding and decoding rules. We denote the hot bit by b_0 and the cold bits by $b_1, b_2, \ldots, b_k$. The decoding and encoding maps of the previous construction (Figure 4) are denoted by $\mathcal{D}(c, c'), \mathcal{E}(c, c', i)$, respectively. The new decoding map $\mathcal{D}^*(c_0, c_1, \ldots, c_k) = (b_0, b_1, \ldots, b_k)$ is applied as follows.

1) $b_0 = \sum_{i=0}^{k} c_i$.
2) For $1 \leqslant i \leqslant k$, $b_i = \mathcal{D}(c_0, c_i)_2$ (bit 2 of the decoded pair).

The new encoding map $\mathcal{E}^*(c_0, c_1, \ldots, c_k, s) = (c_0', c_1', \ldots, c_k')$, is applied as follows, where $0 \leqslant s \leqslant k$ and $(c_0, c_1, \ldots, c_k) \leqslant (c_0', c_1', \ldots, c_k')$. We distinguish between the two cases of whether the hot or a cold bit changes its value.

1) $s \neq 0$ (cold): then $(c_0', c_s') = \mathcal{E}(c_0, c_s, 2) = (c_0, c_s + 2)$.
2) $s = 0$ (hot): if there exists $1 \leqslant i \leqslant k$ such that $\mathcal{E}(c_0, c_i, 1) = (c_0, c_i + 1)$ then set $c_j' = c_j$ for $0 \leqslant j \leqslant k$ and $j \neq i$, and $c_i' = c_i + 1$. Otherwise (that is, for all $1 \leqslant i \leqslant k$, $\mathcal{E}(c_0, c_i, 1) = (c_0 + 1, c_i)$), set $(c_0', c_1', \ldots, c_k') = (c_0 + 1, c_1, \ldots, c_k)$.

Let us show an example of this construction.

Example 1. *In this example, we show how the construction works for $k = 4, q = 5$. Thus, we store a single hot bit b_0 and*

four cold bits b_1, b_2, b_3, b_4 in five cells c_0, c_1, c_2, c_3, c_4.

Written Bit	*Memory State* $(c_0, c_1, c_2, c_3, c_4)$	*Bits State* $(b_0, b_1, b_2, b_3, b_4)$
	(0, 0, 0, 0, 0)	(0, 0, 0, 0, 0)
3	(0, 0, 0, 2, 0)	(0, 0, 0, 1, 0)
1	(0, 2, 0, 2, 0)	(0, 1, 0, 1, 0)
0	(1, 2, 0, 2, 0)	(1, 1, 0, 1, 0)
0	(2, 2, 0, 2, 0)	(0, 1, 0, 1, 0)
0	(2, 3, 0, 2, 0)	(1, 1, 0, 1, 0)
0	(2, 3, 1, 2, 0)	(0, 1, 0, 1, 0)
0	(2, 3, 1, 3, 0)	(1, 1, 0, 1, 0)
0	(2, 3, 1, 3, 1)	(0, 1, 0, 1, 0)
4	(2, 3, 1, 3, 3)	(0, 1, 0, 1, 1)
0	(3, 3, 1, 3, 3)	(1, 1, 0, 1, 1)
0	(3, 3, 2, 3, 3)	(0, 1, 0, 1, 1)
0	(3, 4, 2, 3, 3)	(1, 1, 0, 1, 1)
0	(3, 4, 2, 4, 3)	(0, 1, 0, 1, 1)
2	(3, 4, 4, 4, 3)	(0, 1, 1, 1, 1)
0	(3, 4, 4, 4, 4)	(1, 1, 1, 1, 1)
0	(4, 4, 4, 4, 4)	(0, 1, 1, 1, 1)

The correctness of this construction is proved in the following two lemmas.

Lemma 9. *If $\mathcal{D}^*(c_0, c_1, \ldots, c_k) = (b_0, b_1, \ldots, b_k)$ and the hot bit changes its value then*

$$\mathcal{D}^*(\mathcal{E}^*(c_0, c_1, \ldots, c_k, 0)) = (\overline{b_0}, b_1, \ldots, b_k).$$

Proof: If the hot bit changes its value then exactly one cell increases by one level. Therefore, the value of the hot bit is flipped. When applying the encoding map $\mathcal{E}(c_0, c_i, 1)$ for all $1 \leqslant i \leqslant k$, if there exists i such that $\mathcal{E}(c_0, c_i, 1) = (c_0, c_i + 1)$, then necessarily the value of all the cold bits besides the i-th bit do not change. However, the value of the i-th bit does not change either because $\mathcal{D}(c_0, c_i + 1)_2 = b_i$. If there does not exist i such that $\mathcal{E}(c_0, c_i, 1) = (c_0, c_i + 1)$, then for all i, $\mathcal{E}(c_0, c_i, 1) = (c_0 + 1, c_i)$. In this case, the cell c_0 increases by one level and thus for all i we get $\mathcal{D}(c_0 + 1, c_i)_2 = b_i$. That is, the values of all cold bits remain the same in this case too. ■

For the correctness of the cold writes we have the following.

Lemma 10. *(proof omitted) If $\mathcal{D}^*(c_0, c_1, \ldots, c_k) = (b_0, b_1, \ldots, b_k)$ and the s-th bit, $1 \leqslant s \leqslant k$, changes its value (for the first time) then*

$$\mathcal{D}^*(\mathcal{E}^*(c_0, c_1, \ldots, c_k, s)) = (b_0, b_1, \ldots, b_{s-1}, \overline{b_s}, b_{s+1}, \ldots, b_k).$$

The number of writes of this code is proved in the next theorem.

Theorem 11. *The code guarantees $t = n(q-1) - k$ writes.*

Proof: Since every pair of cells of the form (c_0, c_s) constitutes an independent hot-cold bit code, it is possible to write $2(q-1) - 1$ times. However, since the cell c_0 is a mutual cell to all these k codes, we get that the number of write is

$$q - 1 + k(q-2) = (k+1)(q-1) - k = n(q-1) - k.$$

■

C. Two hot bits, one cold bit

In this part we extend the 2-cell hot+cold construction of sub-section III-A to storing 2 hot bits and 1 cold bit. As it turns out, the penalty (in the number of writes t) for adding a cold bit to the cells remains negligible even when the number of hot bits is doubled from 1 to 2. Due to lack of space we will only include the pictorial specification of the code in Figure 5. The key feature to see in Figure 5 is that changing the cold bit (MSB) is done by moving upward from the solid stack of squares to the dashed one. Updates of the hot bits are done in a diagonal fashion. The re-write capabilities of the code are

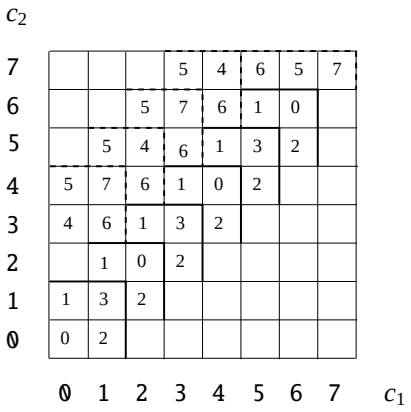

Figure 5. Code to store 2 hot bits and 1 cold bit in 2 cells with $t_h = q - 3$ writes of the 2 hot bits and 1 write of the cold bit.

summarized in the following proposition.

Proposition 12. *(proof omitted) The specified code supports $q-3$ writes of the hot-bit pair in addition to a single write of the cold bit. The write of the cold bit can be anywhere in the write sequence.*

The fact that the pair of hot bits can be written $q - 3$ times, very close to the $q - 1$ writes possible in the absence of the cold-bit write, means that the penalty we suffer to level the wear between the hot and cold bits is insignificant.

References

[1] A. Fiat and A. Shamir, "Generalized write-once memories," *IEEE Transactions on Information Theory*, vol. 30, pp. 470–480, 1984.

[2] F. Fu and A. H. Vinck, "On the capacity of generalized write once memory with state transitions described by an arbitrary directed acyclic graph," *IEEE Transactions on Information Theory*, vol. 45, no. 1, pp. 308–313, 1999.

[3] R. Gabrys and L. Dolecek, "Characterizing capacity achieving write once memory codes for multilevel flash memories," *Proc. IEEE Int. Symp. Inform. Theory*, pp. 2484–2488, St. Petersburg, Russia, August 2011.

[4] A. Jiang, V. Bohossian, and J. Bruck, "Rewriting codes for joint information storage in flash memories," *IEEE Transactions on Information Theory*, vol. 56, no. 10, pp. 5300–5313, 2010.

[5] A. Jiang, M. Langberg, M. Schwartz, and J. Bruck, "Trajectory codes for flash memory," *IEEE Transactions on Information Theory*, vol. submitted, available at arXiv.org.

[6] S. Kayser, E. Yaakobi, P.H. Siegel, A. Vardy, and J.K. Wolf, "Multiple-write WOM-codes," *Proc. 48-th Annual Allerton Conference on Communication, Control and Computing*, Monticello, IL, September 2010.

[7] B. Kurkoski, "Rewriting codes for flash memories based upon lattices, and an example using the E8 lattice," in *Proc. IEEE Globecom 2010, ACTEMT Workshop*, 2010.

[8] R. L. Rivest and A. Shamir, "How to reuse a write-once memory," *Information and Control*, vol. 55, no. 1, pp. 1–19, 1982.

[9] S. Stein and S. Szabo, *Algebra and Tiling*. The Mathematical Association of America, 1994.

[10] Y. Wu and A. Jiang, "Position modulation code for rewriting write-once memories," *IEEE Transactions on Information Theory*, vol. 57, no. 6, pp. 3692–3697, June 2011.