Short Q-Ary Fixed-Rate WOM Codes for Guaranteed Rewrites and With Hot/Cold Write Differentiation

Yuval Cassuto, *Member, IEEE*, and Eitan Yaakobi, *Member, IEEE*

***Abstract*— To the body of works on rewrite codes for constrained memories, we add a comprehensive study in a direction that is especially relevant to practical storage. The subject of this paper is codes for the q-ary extension of the write-once memories model, with input sizes that are fixed throughout the write sequence. Seven code constructions are given with guarantees on the number of writes they can support. For the parameters addressed by the constructions, we also prove upper bounds on the number of writes, which prove the optimality of three of the constructions. We concentrate on codes with short block lengths to keep the complexity of decoding and updates within the feasibility of practical implementation. Even with these short blocks the constructed codes are shown to be within a small additive constant from capacity for an arbitrarily large number of input bits. Part of the study addresses a new rewrite model where some of the input bits can be updated multiple times in a write sequence (hot bits), while other are updated at most once (cold bits). We refer to this new model as hot/cold rewrite codes. It is shown that adding cold bits to a rewrite code has a negligible effect on the total number of writes, while adding an important feature of leveling the physical wear of memory cells between hot and cold input data.**

***Index Terms*— Re-write codes, WOM codes, lattice tiling, multi-level memories, flash storage, hot/cold.**

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) [22]. In the WOM model, k input bits are written t times to n physical cells, where the cell levels cannot decrease between writes. The WOM model has received a significant attention recently thanks to its applicability to the ubiquitous flash storage technology [8], [27]. Alongside the principal WOM model, other interesting re-write models have been proposed and studied with considerable success [14], [16], [17], [26]. The starting point of this work are observations we make on the challenges of the WOM model with respect to its 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 a real storage device expecting fixed-width read/write access.
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 number of input bits in each of the t writes is sought by all constructions.

2,3) A small number (e.g. $2, 3$) of q-ary cells are used by codes that exist for arbitrary q. The decoding and encoding complexities exhibit polynomial (quadratic/cubic) growth in q.

4) We propose constructions for codes that distinguish between "hot" and "cold" bits in the number of updates they allow (hot bits are updated frequently, cold bits are updated rarely).

In the q-ary write-once memory (WOM)[1] model, we are given physical memory cells that can be at one of q ordered discrete levels $\{0, \dots, q-1\}$, with the restriction to only change their levels in the upward direction [6]. Since we clearly wish to reuse the same memory cells for multiple writes, we need a way to allow unrestricted information updates on these restricted cells. One way to achieve that is by employing a WOM *code* [22]. A q-ary WOM code is defined with design parameters n and k, both of which are positive integers. The parameter k specifies the number

Manuscript received September 19, 2012; accepted January 13, 2014. Date of publication April 18, 2014; date of current version June 12, 2014. Y. Cassuto was supported in part by the European Union Marie Curie CIG Grant, in part by the Intel Center for Computing Intelligence, and in part by the Israeli Ministry of Science and Technology. E. Yaakobi was supported in part by the ISEF Foundation and in part by the Lester Deutsch Fellowship. This paper was presented at the 2012 IEEE International Symposium on Information Theory.

Y. Cassuto is with the Department of Electrical Engineering, Technion–Israel Institute of Technology, Haifa 32000, Israel (e-mail: ycassuto@ee.technion.ac.il).

E. Yaakobi is with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125 USA (e-mail: yaakobi@caltech.edu).

Communicated by J.-P. Tillich, Associate Editor for Coding Theory.

Digital Object Identifier 10.1109/TIT.2014.2318722

[1]Note that WOM is a misnomer for non-binary codes, because the physical cells are no longer limited to be written only once.

of input information bits at each write. The parameter n is the number of q-ary memory cells used in the WOM code block. The objective of the WOM code is to allow as many unrestricted k-bit writes as can be guaranteed without violating the restrictions of the physical cells. If a WOM code can guarantee t generations of such writes, we call it an (n, k, t) q-ary WOM code. Once an (n, k, t) code exhausts its t writes, the n cells can not be further reused without an external erase operation, which is not an explicit part of the WOM model (but does happen in practical storage media such as flash memory). The way we specify the WOM code is through a pair of functions: the *decoding* and *update* functions. We define the decoding function as $\psi : \{0, \ldots, q-1\}^n \to \{0, \ldots, M-1\}$, which maps the current levels of the n cells to one of the $M = 2^k$ possible values of the input in the most recent write. In the paper we will use k and M interchangeably, assuming the relation $M = 2^k$ between them. The update function is defined as $\mu : \{0, \ldots, q-1\}^n \times \{0, \ldots, M-1\} \to \{0, \ldots, q-1\}^n$, specifying how the cell levels need to change as a function of the current cell levels and the new information value at the input (here again the input is taken from a set of $M = 2^k$ possible values). For the code to be a legal WOM code, the update function μ must satisfy the following requirements.

1) Consistency: $\psi(\mu(c_1, \ldots, c_n; v)) = v$.
2) Adherence: If $(c'_1, \ldots, c'_n) = \mu(c_1, \ldots, c_n; v)$, then $c_i \leqslant c'_i \leqslant q-1$ for all $i = 1, \ldots, n$.
3) Completeness: $\mu(c_1, \ldots, c_n; v)$ is defined and consistent for every $v \in \{0, \ldots, M-1\}$.

In case the cell levels cannot change their values to a vector $\mu(c_1, \ldots, c_n; v)$ that satisfies these three requirements, we assume that the update function μ returns a special symbol indicating that the memory cannot accommodate more writes, and needs to be erased.

The WOM model specified above is not the most general that has been studied. The simplest generalization, used in the Rivest-Shamir original WOM paper [22], is when the number of possible input values M is not necessarily an integer power of 2. Another important generalization is *variable-rate* WOM codes, where the number of input bits is not constant for all write generations. For this commonly assumed model, the single k (or M) parameter above needs to be replaced by a vector $(k_1, \ldots, k_t)$ (or $(M_1, \ldots, M_t)$) of input sizes for the t write generations. The *sum-rate*, $\mathcal{R}_{\text{sum}}$, of a t write WOM code, which is specified by the input sizes $(M_1, \ldots, M_t)$, is defined to be the ratio between the total number of bits written to the memory and the number of cells, i.e.

$$\mathcal{R}_{\text{sum}} = \frac{\sum_{i=1}^{t} \log_2 M_i}{n}.$$

In [12], Heegard showed that the sum-rate of any binary t write WOM code is at most $\log_2(t+1)$, and later, Fu and Han Vinck [7] showed that for q levels, the sum-rate is at most

$$\log_2 \binom{q+t-1}{t}. \tag{1}$$

Note that these bounds were given to the case where the input sizes on all writes are not necessarily the same. In fact, these bounds are achievable in the sense that it was proved that random coding achieves capacity and in order to achieve the bounds, the input sizes must be all different. An upper bound on the sum-rate in case that the input sizes are the same was studied by Heegard [12] for the binary case, where he gave a recursive formula to calculate such a bound. However, a similar upper bound for q levels is far from being solved, where the only progress was recently given in [8] for the two-write case. In this paper we will also use the normalized sum-rate, denoted $\mathcal{R}_{\text{write}}$, which is defined as

$$\mathcal{R}_{\text{write}} = \frac{\mathcal{R}_{\text{sum}}}{t}.$$

While the capacity of WOM codes is well studied, finding efficient code constructions which achieve sum-rates close to the capacity still remains a challenge. For years, the binary case attracted most of the attention, starting with the early works from almost three decades ago, e.g. [5], [10], and [21], and the more recent ones, e.g. [2], [24], [26], and [27]. In the last few years, with the advent of multi-level flash memories, the interest in finding code constructions for non-binary WOM codes has significantly increased. Huang et al. gave such codes based on error-correcting codes [13]. More code constructions were found in [9]. Kurkoski [19] gave a first analysis of the potential construction of two-write non-binary WOM code using two cells. This idea was extended in [1] by Bhatia et al. to form specific WOM code constructions for arbitrary number of writes and two cells. Yet another extension for two writes and multiple cells was reported by Kurkoski [18]. Another family of non-binary WOM code constructions was recently reported by Haymaker and Kelley in [11]. Recently, Shpilka [23] proposed a capacity achieving construction for both binary and non-binary WOM codes. However the code length of these construction is at least in the order of $(t/\epsilon)^{t/\epsilon}$ (t is the number of writes and ϵ is the difference from the capacity), which is a significant drawback for any implementation of these codes. From a practical standpoint, the variable-rate (non-fixed input size) WOM model used in the great majority of the prior work is far less attractive for the usage of the code in realistic storage devices. That is because the access between a storage device and its hosting system is specified as an interface with a fixed number of bits. Therefore, in this paper we limit ourselves to the more practical WOM model with fixed input sizes M. For this model we give constructions with guaranteed numbers of writes. Although the fixed-input WOM model is more challenging than the general problem, for three of our proposed constructions we are able to show optimality for every alphabet size q.

Besides the contribution of fixed-input codes with re-write guarantees, another important contribution of the paper is code constructions for a whole new re-write model called *hot/cold* re-writes. This model allows to jointly store on the same physical cells bits with different update characteristics. Input bits that are frequently accessed are called *hot* bits, and input bits that are rarely updated are called *cold* bits. In the new hot/cold re-write model we construct codes that allow multiple updates of the hot input bits and a single update of the cold input bits. The motivation for joint hot/cold storage comes from the need to level the wear of physical cells in non-volatile storage media, which is achieved by hot/cold re-write codes even in the presence of extreme write imbalances across input data bits. We note that the problem of hot and cold data was addressed also in [15] in the context of the data movement problem. There, the motivation was to move data to the same physical blocks based on its hot/cold characterization.

The rest of the paper is organized as follows. In Section II, we study WOM codes with $n = 2$ q-ary cells that use lattice tilings to obtain arbitrary numbers of guaranteed writes. For a general k these codes are shown to be within an additive constant from the WOM (variable-input) capacity. The main idea behind tiling-based codes is to use the lattice tiling to define the decoding function, and then build on the algebraic structure of the lattice in finding update functions with guaranteed number of writes for any input sequence. We note that tilings have been previously proposed for use in re-write codes [20], but not for the fixed-k WOM model. While providing structured codes for a large variety of parameters, the tiling-based constructions are not provably optimal. Therefore, in Section III we consider the important special case of $k = 3$ and construct a 2-cell code that gives $t = \lceil 2(q-1)/3 \rceil - 1$ writes, exactly matching the upper bound we prove in Section II-D. In Section IV, we extend the tiling-based construction to three cells. The availability of 3-cell codes adds flexibility to the choice of a desired tradeoff between re-write capabilities and redundancy. The 3-cell constructions also expose the structure of general tiling-based codes, and therefore are an important step on the way to codes for general n. In Section V, we introduce a new re-write model that supports input bits with different update requirements. The code is specified with two types of input bits. Hot bits are allowed to be re-written multiple times, and cold bits are allowed to be written at most once. The objective of the hot/cold constructions is to allow as many writes of the hot bits, and one write of the cold bits. An important requirement is that the write of cold bits can be performed *anywhere 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. Tiling-Based 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 $1 \to 0$ [14].

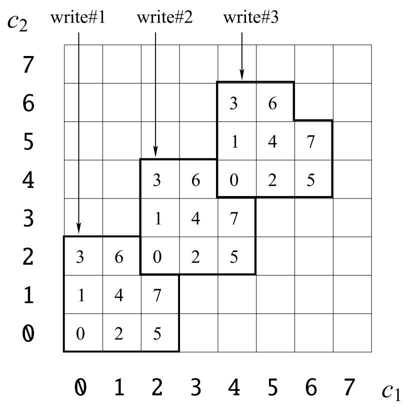


Fig. 1. A WOM code that stores $k = 3$ bits in $n = 2$ cells with $t = \lfloor (q-1)/2 \rfloor$ writes.

Since the case of $n = 1$ is completely characterized, we move to discuss the case of $n = 2$. With $n = 2$ cells, it is easy to verify that for $k = 2$ the maximum guaranteed number of writes is $t = q-1$ and is achieved when assigning a bit to each cell. Hence, the WOM problem becomes interesting starting at $k = 3$, a case we study next.

A. Storing 3 Bits in 2 Cells: First Attempt

Recall from Section I that 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 $v \in \{0, 1, \ldots, 2^k - 1\}$; in the case of $k = 3$, $v \in \{0, 1, \ldots, 7\}$. A mapping between integers and $k = 3$-bit vectors is implicitly assumed. Reading information is then performed by the decoding function $\psi(c_1, c_2)$, where $\psi : \{0, \ldots, q-1\}^2 \to \{0, 1, \ldots, 7\}$. Writing $k = 3$ bits to the physical cells is specified by the update function μ taking the current cell contents and the new information value v'. Thus

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

Such decoding and update functions for $k = 3$ are specified in Figure 1. The numbers inside the array cells stand for information values v in $\{0, 1, \ldots, 7\}$. The coordinates marked at the exterior of the array represent cell levels. The horizontal coordinate is c_1 and the vertical one is c_2. The decoding function $\psi(c_1, c_2)$ is specified by the content of the (c_1, c_2) position in the array. An update 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 3. Then a value $v' = 7$ is written by moving the cells to levels $(c_1', c_2') = (4, 3)$.

Each shape of area $M = 8$ in Figure 1 specifies the range of possible cell levels (c_1', c_2') after a given write generation. Since the shape for write i has both $c_1 \leqslant 2i$ and $c_2 \leqslant 2i$, Figure 1 specifies an $n = 2$, $k = 3$ q-ary 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. 2-Cell Codes by 2-Dimensional Lattice Tilings: $k = 3$

To get more writes from the 2-dimensional (c_1, c_2) plane, the tiling-based approach takes the following steps.

1) Tile the entire (c_1, c_2) 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.

To make the forthcoming code constructions clearer, we review relevant fundamentals of lattice tilings for the special case of $n = 2$ and $k = 3$.

Two-dimensional lattice tiling. $k = 3$.

Let the shape 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 [25]. 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 *basis* 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 basis. The lattice T generated by G, together with the shape S form a perfect lattice tiling, i.e., copies of S, the centers of which are placed on the lattice points defined by T, are disjoint, and fill the entire two-dimensional plane. The fact that the translations of S fill the plane is seen by calculating the size of the lattice $|\det(G)| = 8$, and noting that it equals the size of the shape S. Given a perfect lattice tiling, labeling the integer points of the two-dimensional plane is done by assigning the numbers between 0 and 7 of S to their respective locations in copies of S translated by T.

Based on the above lattice tiling, we specify the following code construction, which outperforms the code of Figure 1 in the number of writes for a given q.

Construction 1: For any q, we define a 2-cell, $k = 3$ code as follows.

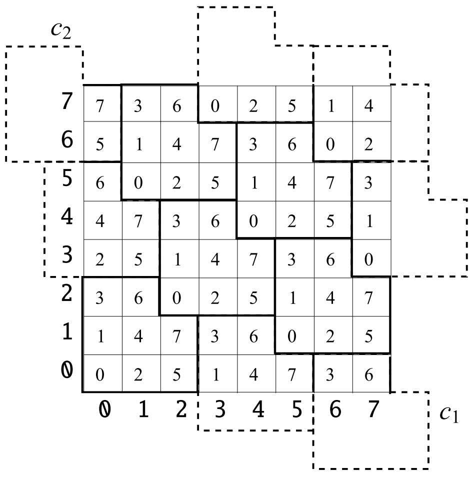


Fig. 2. Example of lattice-tiling based Construction 1 for $q = 8$.

1. Decoding Function: The decoding function $\psi(c_1, c_2)$ is obtained by a lattice tiling of the two-dimensional cell-level range $\{(c_1, c_2) \mid 0 \leqslant c_1, c_2 \leqslant q - 1\}$ by the shape

3	*6*	
1	*4*	*7*
0	*2*	*5*

using the generator matrix

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

2. Update Function: Given current levels (c_1, c_2) and input value m, find the the new cell levels (c'_1, c'_2) that satisfy the following conditions.

1) $c'_1 \geqslant c_1, c'_2 \geqslant c_2$.
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.

An example of a code obtained by Construction 1 is given in Figure 2, for $q = 8$. A central step toward the establishment of the number of writes guaranteed by Construction 1 is made in Lemma 1 below. To help in the proof of Lemma 1 and in subsequent results, we give the following definition.

Definition 1: For an update sequence of a 2-cell code, define the pair $(\delta_{i,1}, \delta_{i,2})$ as the cell-level increments in write i of cell 1 and cell 2, respectively, for $i = 1, 2, \ldots, t$. The final cell-level pair at the end of the update sequence is

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

At a high level, the algebraic structure of the lattice allows to maintain balance between c_1 and c_2 increments, such that an update sequence with $\delta_{i,1}$ repeatedly larger than $\delta_{i,2}$ (or vice versa) will be replaced with a more balanced one equivalent to it in terms of the information bits.

Lemma 1: If $q = 8$, then Construction 1 guarantees four writes.

Proof: Let us consider the four writes of this code. We need to argue that for any update sequence the update function will reach a final level pair (c_1, c_2) that satisfies $c_1, c_2 \leqslant 7$. 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 $\delta_{i,1} = 2$ for all $1 \leqslant i \leqslant 4$, or $\delta_{i,2} = 2$ for all $1 \leqslant i \leqslant 4$. 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 (fourth) 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

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

■

Now we prove the guaranteed write count of Construction 1 in the following theorem.

Theorem 2: Construction 1 guarantees $t = \lfloor 4(q - 1)/7 \rfloor$ writes.

Proof: When 7 divides $q - 1$, we can divide the write sequence to periods $j = 1, \ldots, (q - 1)/7$, each starting from level pair $c_1 = c_2 = 7(j - 1)$ and ending at level pairs $c_1, c_2 \leqslant 7j$. Lemma 1 guarantees that a level increase of 7 in each of c_1, c_2 within a period is sufficient to support four writes in each period. Hence the total number of writes is $4(q - 1)/7$. When dividing $(q - 1)$ by 7 results in a non-zero residue r, the theorem requires that in the last (partial) period we will be able to write $t_r = \lfloor 4r/7 \rfloor$ times with increment of up to r levels in each of c_1, c_2. These values of t_r are listed in the table

r	1	2	3	4	5	6
t_r	0	1	1	2	2	3

,

and all of these r, t_r combinations are achievable using the diagonal stacking construction of Figure 1. ■

The result from Theorem 2 is that the algebraic structure of the lattice tiling helped improving the write count from $t = \lfloor (q - 1)/2 \rfloor$ in Figure 1 to $t = \lfloor 4(q - 1)/7 \rfloor$ in Construction 1.

C. 2-Cell Codes by 2-Dimensional Lattice Tilings: General k

We now generalize the construction from the previous subsection to $k > 3$. For that we use a generalization of the two-dimensional corner shape from Construction 1, 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

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

An n-dimensional generalization of $C(a, b)$ is used in [3] for a different application.

Proposition 3: For all $a > b > 0$, a lattice tiling with the shape $C(a, b)$ is given by the vectors:

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

Proof: Elementary, see [3]. ■

The lattice equivalence proved in the following lemma will be later used to prove the number of writes of the generalized 2-cell lattice-based construction.

Lemma 4: The points $(a - 1, a - b - 1)$ and $(a - b - 1, 2a - b - 1)$ are equivalent (contain the same element) with respect to the lattice $\boldsymbol{v}_1 = (a - b, a - b), \boldsymbol{v}_2 = (a, -b)$.

Proof: Note that

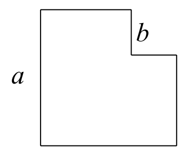
$$(a - 1, a - b - 1) - (a - b - 1, 2a - b - 1) = (b, -a) = \boldsymbol{v}_2 - \boldsymbol{v}_1.$$

■

For any parameters a and b such that $a > b$ the following is a generalization of Construction 1.

Construction 2: For any q and integer parameters a,b such that $a > b$, we define a 2-cell, $M = a^2 - b^2$ code as follows.

1. Decoding Function: The decoding function $\psi(c_1, c_2)$ is obtained by a lattice tiling of the two-dimensional cell-level range $\{(x, y) \mid 0 \leqslant x, y \leqslant q - 1\}$ by the shape



using the generator matrix

$$G = \begin{pmatrix} a - b & a - b \\ a & -b \end{pmatrix}.$$

2. Update Function: Same as in Construction 1.

Proposition 5: If $c \triangleq a/b$ is a positive integer, and $q = c(a - 1) + a - b$, then Construction 2 guarantees $t = c + 1$ writes.

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

For $1 \leqslant i \leqslant t$, let $(\delta_{i,1}, \delta_{i,2})$ be the level-increase in the two cells. According to the shape of the corner we know that $\min\{\delta_{i,1}, \delta_{i,2}\} \leqslant a - b - 1$ and $\max\{\delta_{i,1}, \delta_{i,2}\} \leqslant a - 1$.

Suppose that there exists i such that $\delta_{i,2} \leqslant a - b - 1$, then the maximum level of cell 2 after t writes is at most

$$a - b - 1 + (t - 1)(a - 1) = (a - b - 1) + c(a - 1) = q - 1.$$

The same applies to cell 1 from symmetry.

From the above we only need to consider the case where for all $1 \leqslant i \leqslant t$, $\delta_{i,1} \geqslant a - b$, or for all $1 \leqslant i \leqslant t$, $\delta_{i,2} \geqslant a - b$. Without loss of generality, assume that for all $1 \leqslant i \leqslant t$, $\delta_{i,1} \geqslant a - b$; then from $\min\{\delta_{i,1}, \delta_{i,2}\} \leqslant a - b - 1$ we know that $\delta_{i,2} \leqslant a - b - 1$ for all $1 \leqslant i \leqslant t$. Assume that the increase at every write is the worst case $(\delta_{i,1}, \delta_{i,2}) = (a - 1, a - b - 1)$, then at the last write, we can increase the cell levels by $(\delta_{i,1}, \delta_{i,2}) = (a - b - 1, 2a - b - 1)$, which by

Lemma 4 is equivalent to $(\delta_{i,1}, \delta_{i,2}) = (a-1, a-b-1)$ from the lattice perspective (thus the decoding function gives the desired value.). Therefore, the level of cell 1 is at most

$$(t-1)(a-1) + (a-b-1) = c(a-1) + a - b - 1 = q - 1,$$

and the level of cell 2 is at most

$$\begin{aligned}(t-1)(a-b-1) + 2a - b - 1 &= c(a-b-1) + 2a - b - 1\\ = c(a-1) - \underbrace{cb}_{=a} + 2a - b - 1 &= c(a-1) + a - b - 1 = q - 1,\end{aligned}$$

where the equality under the brace is from the definition of c. This proves that the specified code guarantees $c+1$ writes on q-ary cells. ■

Note that similarly to Construction 1, it is possible to use any number of periods, each adding $c+1$ writes and $c(a-1) + a - b$ levels.

To obtain a k-bit WOM code using Construction 2, we need to choose a corner shape with size that equals $M = 2^k$. An option for an arbitrary odd number $k \geqslant 3$ is to choose $a = 3 \cdot 2^{\frac{k-3}{2}}$, $b = 2^{\frac{k-3}{2}}$, ($c = 3$), and the resulting tile size is

$$M = a^2 - b^2 = 9 \cdot 2^{k-3} - 2^{k-3} = 8 \cdot 2^{k-3} = 2^k.$$

The number of levels is

$$q = 3(a-1) + a - b = 11 \cdot 2^{\frac{k-3}{2}} - 3, \qquad (2)$$

and by Proposition 5 this gives $t = c + 1 = 4$ writes. If we compare this code to a code that stacks corner shapes diagonally (as shown in Figure 1 for $k = 3$), we get an advantage of about 10% because the latter requires

$$q' = t(a-1) + 1 = 4 \cdot (3 \cdot 2^{\frac{k-3}{2}} - 1) + 1 = 12 \cdot 2^{\frac{k-3}{2}} - 3.$$

The total rate of this 4-write code (recall the definition of the sum rate from Section I) is $\mathcal{R} = 4 \cdot k/2 = 2k$, where the denominator 2 is the number of cells used by the code. To compare this rate to the best possible, we now substitute q from (2) and $t = 4$ in the q-ary WOM capacity expression (1), thus getting

$$\begin{aligned}\log_2\binom{q+3}{4} &< \log_2\left(\frac{(11 \cdot 2^{\frac{k-3}{2}})^4}{24}\right)\\ &= 4 \cdot \frac{k-3}{2} + 4 \cdot \log_2 11 - \log_2 24\\ &= 2k - 6 + 13.8377 - 4.585 = 2k + 3.2527.\end{aligned}$$

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 upper bound. When we normalize this gap by the number of writes $t = 4$, we get that the written rate per write per cell is at most 0.8132 bit smaller than the upper bound on capacity. We note further that this upper bound is only known to be achievable with variable-rate writes, so the true gap of Construction 2 to optimality is likely to be smaller, and is still an open question.

D. Upper Bounds on Constant-k WOM Codes

Known bounds on the sum-rate of q-ary WOM codes over multiple writes do not restrict a constant k in all write generations. For example, the authors of [7] derived an upper bound by counting for a single q-ary cell all possible level-increment sequences observed in t writes. This counting bound however does not at all mandate the amount of information that is input to the cell in individual writes, which in principle can be arbitrary and variable (indeed the codes in [7] that asymptotically meet this bound use variable rates across write generations). To fix this limitation of known bounds, and to have a tool to evaluate fixed-rate constructions, we now propose an alternative bound technique that does restrict the input sizes of individual writes. We focus here on bounds for 2-cell codes, but the same technique can be generalized to codes with $n > 2$ in a straightforward manner. The core idea is that having M possible input values in each write implies that at least one choice will require a level increment with at least a certain sum over the code cells.

Lemma 6: Given a 2-cell WOM code with M input values in every write. If $M > s(s+1)/2$ for some integer s, then for each write i there exists an input value such that

$$\delta_{i,1} + \delta_{i,2} \geqslant s.$$

Proof: The proof uses a simple area argument. Each of the M input values must result in a distinct update vector $(\delta_{i,1}, \delta_{i,2})$. The number of non-negative integer vectors that satisfy the sum constraint $\delta_{i,1} + \delta_{i,2} < s$ equals $s(s+1)/2$. Thus since $M > s(s+1)/2$, there must be at least one update vector that requires $\delta_{i,1} + \delta_{i,2} \geqslant s$. ■

Using Lemma 6 we can now state an upper bound on the number of writes of a q-ary WOM code with constant input size.

Proposition 7: Given a q-ary 2-cell WOM code with $M > s(s+1)/2$ for some integer s, the number of writes is bounded by

$$t \leqslant \left\lfloor \frac{2(q-1)}{s} \right\rfloor.$$

Proof: The sum of the levels of cell 1 and cell 2 at the end of the write sequence is at most $2(q-1)$. From Lemma 6, there exists a worst-case write sequence in which for every write i the sum $\delta_{i,1} + \delta_{i,2}$ is increased by at least s. Summing over t writes we get the desired bound. ■

When we substitute $s = 3$ in Proposition 7 we obtain the following corollary that gives an upper bound on the number of writes in a 2-cell code with $M > 6$.

Corollary 8: Given a q-ary 2-cell WOM code with $M > 6$, the number of writes is bounded by

$$t \leqslant \left\lfloor \frac{2(q-1)}{3} \right\rfloor.$$

Since $k = 3$ codes have $M = 8$, a strictly stronger requirement than $M > 6$, it is possible to tighten the bound of Corollary 8 by a little. The resulting upper bound in the following theorem turns out to be the tightest possible. It is shown to be achievable in Section III.

Theorem 9: Given a q-ary 2-cell WOM code with $M \geqslant 8$, the number of writes is bounded by

$$t \leqslant \left\lceil \frac{2(q-1)}{3} \right\rceil - 1.$$

Proof: When $q \equiv 0 \pmod 3$ or $q \equiv 2 \pmod 3$, the right-hand side in Corollary 8 and the right-hand side in the current theorem are identical, and therefore the statement is trivially true. So we only need to prove that when $q \equiv 1 \pmod 3$, t is *strictly* smaller than $\lfloor 2(q-1)/3 \rfloor$ at the right-hand side of Corollary 8. Let the first $t-1$ writes be any sequence of value updates $v_1, \ldots, v_{t-1}$, each of which resulting in $\delta_{i,1} + \delta_{i,2} = 3$. We now consider the t-th write. For $M \geqslant 8$ there are at least two distinct inputs – denoted v_t and v_t' – that result in increments $\delta_{t,1}$,$\delta_{t,2}$ with $\delta_{t,1} + \delta_{t,2} = 3$. Assume by contradiction that there exists a code with $3t = 2(q-1)$. Then after the t-th write the cell levels must satisfy $c_1 = c_2 = q-1$. But since there are two distinct update sequences $v_1, \ldots, v_{t-1}, v_t$ and $v_1, \ldots, v_{t-1}, v_t'$ with sum increments of 3 in each write and different final levels, a unique final level $c_1 = c_2 = q-1$ is a contradiction. ■

III. Optimal 2-Cell Codes for $k = 3$

In Section II we showed that lattice tiling is a useful building block for the construction of effective re-write codes. In those constructions, the algebraic structure of the lattice helped in specifying update functions with guaranteed numbers of writes for any update sequence. The main strength of the lattice-tiling construction technique is its generality, i.e., its ability to provide codes for different values of k and n (lattice-tiling codes for $n > 2$ are discussed in Section IV). However, to that end we do not know that lattice-tiling constructions give the best codes for a given set of parameters k, n and q. A reason to suspect that this may not be the case is the fact that the number of writes guaranteed by the lattice-tiling based Construction 1 follows a slower growth of $4(q-1)/7$ compared to the upper bound of $2(q-1)/3$ proved in Theorem 9. This gap raises the motivation to look for improvements over lattice-tiling constructions for important special cases of n and k. In the following we propose such a construction for 2-cell, $k = 3$ codes that is in fact shown to be optimal.

Construction 3: For any q, we define a 2-cell, $k = 3$ code as follows.

1. Decoding Function: The decoding function $\psi(c_1, c_2)$ is specified in two parts: the initial *part, specifying $\psi(c_1, c_2)$ only for levels $c_1 + c_2 \leqslant 6$, and the* periodic *part, specifying $\psi(c_1, c_2)$ for the remainder of the two-dimensional cell-level range $\{(c_1, c_2) \mid 0 \leqslant c_1, c_2 \leqslant q-1\}$.*

Initial part: For cell levels c_1, c_2 such that $c_1 + c_2 \leqslant 6$, the decoding function is defined as:

4		2	6		
3		1	3	4	
2	3	6	0	5	7
1	1	4	7	1	2
0	0	2	5		
	0	1	2	3	4

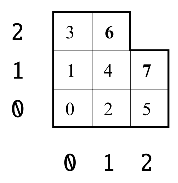
Periodic part: For $j = 1, 2, 3, \ldots$ the level pairs (c_1, c_2) such that $6j + 1 \leqslant c_1 + c_2 \leqslant 6j + 6$ are decoded as specified in:

$3j+$						
4	7	3	5	6		
3	2	0	6	1	4	
2	4	7	3	0	2	7
1		1	0	5	7	3
0			2	6	0	5
−1				4	1	6
	−1	0	1	2	3	4 $+3j$

In the decoding-function plot above, the level coordinates are given relative to the period j, i.e. $3j-1, 3j, \ldots, 3j+4$. For example, the level pair $(c_1, c_2) = (11, 8)$ is decoded as the value 4 because $(11, 8)$ can be written as $11 = 3j + 2$ and $8 = 3j - 1$ for $j = 3$. Finding the location pair $(3j+2, 3j-1)$ in the decoding figure gives the value 4. Note that the decoding function $\psi(c_1, c_2)$ specified above is not defined for some (c_1, c_2) pairs. This fact does not raise a problem as long as the update function never reaches an undefined level pair.

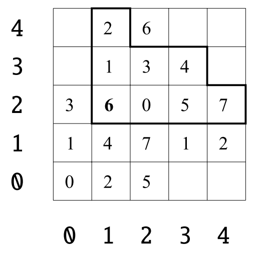
2. Update Function: As in the decoding function, the update function is specified in two parts: the initial part for the first two writes, and the periodic part for the remaining writes. The shapes with solid borders in the figures below specify the update function given the current memory state is the lower-left corner of the shape. Thus each such shape must contain every value in $\{0, \ldots, 7\}$ at least once.

Initial part: The first of the two writes of the initial part sets the cell levels according to

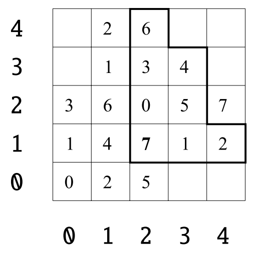


2	3	6	
1	1	4	7
0	0	2	5
	0	1	2

Before writing the second value, we can assume to start from one of two locations $(1, 2)$ or $(2, 1)$, at least one of which is accessible from any location of the first write. The update functions for each of the $(1, 2)$ and $(2, 1)$ starting locations are given (from left to right) in



4		2	6		
3		1	3	4	
2	3	6	0	5	7
1	1	4	7	1	2
0	0	2	5		
	0	1	2	3	4



4		2	6		
3		1	3	4	
2	3	6	0	5	7
1	1	4	7	1	2
0	0	2	5		
	0	1	2	3	4

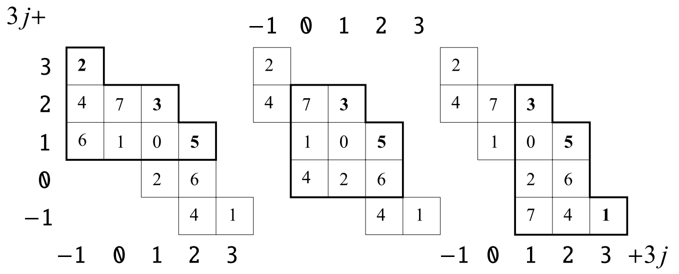
Periodic part: For the first write of the periodic part we consider three possibilities for the starting location (c_1, c_2), for every $j = 1, 2, 3 \ldots$:

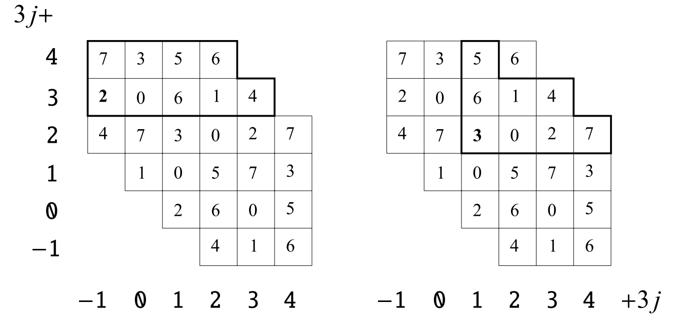
1) $(3j-1, 3j+1)$.
2) $(3j, 3j)$.
3) $(3j+1, 3j-1)$.

These three starting locations will be updated according to the following three plots (from left to right):
For the second write of the periodic part we assume to start in one of the four locations

1) $(3j-1, 3j+3)$ *– bold* **2** *label above.*
2) $(3j+1, 3j+2)$ *– bold* **3** *label above.*
3) $(3j+3, 3j-1)$ *– bold* **1** *label above.*
4) $(3j+2, 3j+1)$ *– bold* **5** *label above.*

The update functions for each of these four starting locations are now specified. For 1) and 2) (from left to right) in the plot and for 3) and 4) (from left to right) in the plot





After the second write of the periodic part we return to the first write of the periodic part, but with j incremented by 1*. Note that all parts of the update function satisfy the three required properties from Section* I*.*

1) Consistency: The information labels $\{0, \ldots, 7\}$ *appearing in the update-function plots are identical to the information labels in the decoding-function plots for all* (c_1, c_2) *pairs.*
2) Adherence: Transitions specified by the different shapes in the update-function plots either leave a level unchanged or increment it. At the end of the periodic part, no cell exceeds level $q-1$ *as long as* $3j+4 \leqslant q-1$*.*
3) Completeness: In every shape in the update-function plots there is at least one instance of every information label $\{0, \ldots, 7\}$*.*

Theorem 10: Construction 3 *guarantees* $t = \lceil 2(q-1)/3 \rceil - 1$ *writes.*

Proof: Before counting the number of writes, an important preliminary step of the proof is to prove the continuity between the different parts of the update function of Construction 3. Continuity means that every possible end location of the update function in one write is considered as a start location for

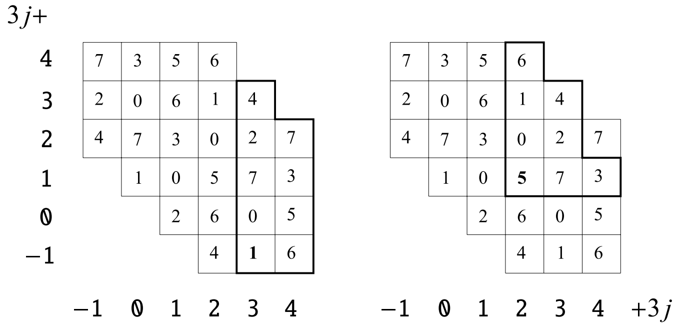


TABLE I
START LOCATIONS AND WORST-CASE END LOCATIONS OF CONSTRUCTION 3'S UPDATE FUNCTION

Step	Start locations	End locations
Initial write 1	$(0,0)$	$(1,2),(2,1)$
Initial write 2	$(1,2),(2,1)$	$(2,4),(3,3),(4,2)$
Periodic write 1	$(3j-1,3j+1),(3j,3j)$ $,(3j+1,3j-1)$	$(3j-1,3j+3),(3j+1,3j+2)$ $,(3j+3,3j-1),(3j+2,3j+1)$
Periodic write 2	$(3j-1,3j+3),(3j+1,3j+2)$ $,(3j+3,3j-1),(3j+2,3j+1)$	$(3j+2,3j+4),(3j+3,3j+3)$ $,(3j+4,3j+2)$

the following write. We demonstrate the continuity by Table I summarizing the four phases of the update function (initial write 1, initial write 2, periodic write 1, periodic write 2). For each phase, Table I lists the possible start locations specified in Construction 3, and the worst-case end locations after the write. By worst case we mean that at least one of the listed end locations is reachable from every possible end location by only incrementing levels or leaving them unchanged. In each row, we need to verify that every end location appears in the list of start locations of the next row. Similarly, the periodicity of the construction requires that the end locations of periodic write 2 have corresponding start locations in periodic write 1. Examining Table I, the continuities from initial write 1 to initial write 2 and from periodic write 1 to periodic write 2 are obvious (the right column of row initial write 1 is equal to the center column of row initial write 2, and same for the periodic-write rows). Continuity from initial write 2 to periodic write 1 is established by substituting $j = 1$ in the start locations of the latter. Finally, continuity between periodic write 2 back to periodic write 1 is obtained by substituting $j \leftarrow j+1$ in the start locations of the latter.
Now that Construction 3 is shown to be correct, we move to count the number of writes.

Case 1: $q \equiv 2 \pmod 3$ (This case corresponds to completing an integer number of full periods.). After periodic write 2 of period j, the worst-case cell level is $3j+4$. Writing j full periods is possible if $3j+4 \leqslant q-1$. Therefore, when $q \equiv 2 \pmod 3$ the number of guaranteed full periods is an integer $j^* = (q-5)/3$. Since each period has two writes, and the initial part contributes two more writes, in total we have

$$t = 2j^* + 2 = 2(q-2)/3.$$

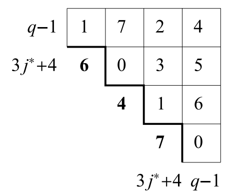


Fig. 3. Additional write after the j^*-th period with one extra level.

It is easy to verify that $2(q-2)/3 = \lceil 2(q-1)/3 \rceil - 1$ when $q \equiv 2 \pmod 3$.

Case 2: $q \equiv 0 \pmod 3$. In this case we complete $j^* = \lfloor (q-5)/3 \rfloor = (q-6)/3$ full periods, with level $q-1$ remaining unused. We claim that the update function can be extended in this case to have one more write beyond the j^* periods. This can be seen in Figure 3 showing that starting from the end of the j^*-th period, at one of the locations marked with bold **6**, **4** and **7**, any value in $\{0, \dots, 7\}$ can be reached with one extra level.

So overall for this case we have 2 writes of the initial part, $2j^*$ writes in the periodic part and 1 write afterward. These sum up to

$$t = 2j^* + 3 = 2(q-6)/3 + 3 = 2q/3 - 1.$$

It is easy to verify that $2q/3 - 1 = \lceil 2(q-1)/3 \rceil - 1$ when $q \equiv 0 \pmod 3$.

Case 3: $q \equiv 1 \pmod 3$. In this case, similarly to case 2, we can have $j^* = \lfloor (q-5)/3 \rfloor$ full periods, only now with one more unused level at the end. So it is clear that we can have at least as many writes as in case 2. In total

$$t = 2j^* + 3 = 2(q-7)/3 + 3 = 2(q-1)/3 - 1.$$

It is trivial that $2(q-1)/3 - 1 = \lceil 2(q-1)/3 \rceil - 1$ when $q \equiv 1 \pmod 3$. ■

Note that t in Theorem 10 matches the upper bound in Theorem 9, thus proving the optimality of Construction 3. The interesting implication from this optimality is that a weaker constraint used by the upper bound of Section II-D is actually equivalent, at least for the parameters in question, to a much stronger requirement from a real code. In the upper bound, the code was required to accommodate updates with only the *sum* $\delta_{i,1} + \delta_{i,2}$ being bounded from below by some constant. In a real code, however, one can imagine a worst-case/adversarial update sequence that is not only growing in the sum c_1+c_2, but also causes imbalance resulting in one of c_1, c_2 overflowing prematurely. The success of Construction 3 is therefore its ability to perfectly balance the increments in c_1 and c_2 in a t-write cycle, *for all update sequences*. While Construction 3 addresses the specific case of $k = 3$, the same balancing techniques with shapes of different sizes can yield codes for other values of k.

IV. Tiling-Based WOM Codes With Three Cells

As the constructions of Section II show, storing k bits in two cells can be done with good two-dimensional tilings with shapes of size 2^k, and specifying the encoding and decoding functions. Our objective in this section is to extend these constructions to three cells. The motivation is twofold. The first is to improve the storage efficiency and flexibility beyond 2-cell codes. The second is to study 3-cell codes as a gateway to understanding the general n-cell coding problem. Recall that in the 2-cell constructions we considered a two-dimensional array and if k bits were stored, shapes of size 2^k were positioned on this two-dimensional array such that

1) The shapes do not overlap.
2) The transition from one shape to another is restricted to only advance in the positive direction of each dimension.

An efficient code was obtained in Construction 2 by finding a shape that perfectly tiles the two-dimensional array – the corner shape. Going beyond two-dimensions, we use an extension of the two-dimensional corner shape for three and higher dimensions. This shape is known as the *n-dimensional chair*, and was recently studied in [3] for correcting asymmetric errors. Formally, an n-dimensional chair $\mathcal{S}_{n,\boldsymbol{a},\boldsymbol{b}} \subseteq \mathbb{R}^n$, where $\boldsymbol{a} = (a_1, a_2, \dots, a_n), \boldsymbol{b} = (b_1, b_2, \dots, b_n) \in \mathbb{R}^n$, $0 < b_i < a_i$ for each i, $1 \leqslant i \leqslant n$, is an n-dimensional $a_1 \times a_2 \times \cdots \times a_n$ box from which an n-dimensional $b_1 \times b_2 \times \cdots \times b_n$ box was removed from one of its corners.

$$\mathcal{S}_{n,\boldsymbol{a},\boldsymbol{b}} = \{(x_1, x_2, \dots, x_n) \in \mathbb{R}^n : 0 \leqslant x_i < a_i, \text{ and there exists a } j, 1 \leqslant j \leqslant n, \text{ such that } x_j < a_j - b_j\}.$$

In this work, we will only consider the special case of three dimensions where $\boldsymbol{a} = (a_1, a_2, a_3) = (a, a, a)$, $\boldsymbol{b} = (b_1, b_2, b_3) = (b, b, b) \in \mathbb{Z}^3$, for two positive integers a, b such that $0 < b < a$. Hence, as the special case of $\mathcal{S}_{n,\boldsymbol{a},\boldsymbol{b}}$ we define $\mathcal{S}_{3,a,b}$ to be the three-dimensional discrete shape with $\boldsymbol{a}$ and $\boldsymbol{b}$ replaced by fixed scalars a and b.

$$\mathcal{S}_{3,a,b} = \{(x_1, x_2, x_3) \in \mathbb{Z}^3 : 0 \leqslant x_i < a, \text{ and there exists a } j, 1 \leqslant j \leqslant 3, \text{ such that } x_j < a - b\}.$$

A construction of a lattice tiling Λ for the n-dimensional chair was recently studied in [3] and is given by the following generator matrix

$$G = \begin{pmatrix} a_1 & -b_2 & 0 & 0 & \cdots & 0 \\ 0 & a_2 & -b_3 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{n-2} & -b_{n-1} & 0 \\ 0 & 0 & \cdots & 0 & a_{n-1} & -b_n \\ -b_1 & 0 & \cdots & 0 & 0 & a_n \end{pmatrix}$$

Note that $|\det(G)| = \prod_{i=1}^n a_i - \prod_{i=1}^n b_i$, which is the volume of the n-dimensional chair.

For our special-case three-dimensional construction with equal a_i and b_i, the generator matrix is given by

$$G_3 = \begin{pmatrix} a & -b & 0 \\ 0 & a & -b \\ -b & 0 & a \end{pmatrix},$$

$|\det(G_3)| = a^3 - b^3$, and we let Λ_3 be the lattice generated by the matrix G_3. We denote the first, second, third row of the matrix G_3 by $\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3$, respectively. For the rest of this section we assume that $\frac{a}{b} = c$ is a positive integer greater than one.

Since the size of the three-dimensional chair is $M_3 = a^3 - b^3$, we assume that there is a mapping $\psi : \mathbb{Z}^3 \to \{0, 1, \ldots, M_3 - 1\}$, which simply assigns to each point $(i_1, i_2, i_3) \in \mathbb{Z}^3$ a label such that lattice translations of the chair shape preserve the labeling (these labels are called the coset numbers in the lattice terminology). This mapping defines the decoding function.

Let us now describe the idea how the update function works. Our main goal is to show that in every $T = \frac{c^3-1}{c-1} = 1+c+c^2$ writes, the maximum cell level increases by at most

$$\Delta_3 = T(a-1) - b$$

levels. To show this, we simply prove that on the first T writes the maximum cell level is at most Δ_3. Then, the same arguments are applied for the following cycles of T writes.

On each write, assume that the point $\boldsymbol{i} = (i_1, i_2, i_3)$ represents the current cell levels and a new symbol $m \in \{0, 1, \ldots, M_3 - 1\}$ is received. In general, by forming the three-dimensional chair $\mathcal{S}_{3,a,b}$ shifted by the vector $\boldsymbol{i}$, it is always possible to increase the cell levels by a vector $\boldsymbol{d} = (\delta_1, \delta_2, \delta_3)$ such that $\psi(\boldsymbol{i} + \boldsymbol{d}) = m$ and $\boldsymbol{d}$ is a point in the three-dimensional chair, $\mathcal{S}_{3,a,b}$. Hence, from the structure of $\mathcal{S}_{3,a,b}$, $0 \leqslant \delta_\ell \leqslant a - 1$ for all $1 \leqslant \ell \leqslant 3$, and there exists $1 \leqslant j \leqslant 3$ such that $\delta_j \leqslant a - b - 1$. This update rule will be called the *basic update rule* and the encoder will use it on most of its writes.

Note that on each write every cell increases its level by at most $a - 1$ levels and there is at least one cell whose level increase is at most $a-b-1$. In order to show that in every T writes the increase in the maximum cell level is at most Δ_3, we will show that in these T writes, *every cell* increases its level at least once by at most $a-b-1$.

We now describe formally the update and decoding functions of the 3-cell construction for a cycle of T writes.

Construction 4: For $q = \Delta_3 + 1$, we define a $(3, k = \log_2(M_3), T)$ WOM code as follows.

1. Decoding Function: The decoding function $\psi(c_1, c_2, c_3)$ is obtained by a lattice tiling of the three-dimensional chair $\mathcal{S}_{3,a,b}$ using the generator matrix G_3.

2. Update Function: On the j-th write, $1 \leqslant j \leqslant T$, we let $\boldsymbol{i}_j = (i_{j,1}, i_{j,2}, i_{j,3})$ be the cell-level vector at the beginning of the j-th write and $m_j \in \{0, 1, \ldots, M_3 - 1\}$ is the received message. On each write, the increment vector according to the basic update rule is denoted by $\boldsymbol{d}'_j = (\delta'_{j,1}, \delta'_{j,2}, \delta'_{j,3})$. Our goal is to find the actual increment vector to the cells, which we denote by $\boldsymbol{d}_j = (\delta_{j,1}, \delta_{j,2}, \delta_{j,3})$.

On the first c writes we simply apply the basic update rule, that is, $\boldsymbol{d}_j = \boldsymbol{d}'_j$ for $1 \leqslant j \leqslant c$. On the $(c+1)$-th write, we apply the following rules. For $\ell = 1, 2, 3$, we define

$$w_{c+1,\ell} = \left\lfloor \frac{(c+1)(a-1) - (i_{c+1,\ell} + \delta'_{c+1,\ell})}{b} \right\rfloor.$$

The goal of the variable $w_{c+1,\ell}$ is to indicate, in case that the basic update rule is applied, how far the cell level is from the maximum increase $(c+1)(a-1)$, normalized by b. The increment vector $\boldsymbol{d}_{c+1}$ is determined as follows.

1) If there are $\ell_1, \ell_2 \in \{1, 2, 3\}$, $\ell_1 \neq \ell_2$, and $w_{c+1,\ell_1}, w_{c+1,\ell_2} \geqslant 1$, then $\boldsymbol{d}_{c+1} = \boldsymbol{d}'_{c+1}$.

2) Otherwise, there exists $\ell \in \{1, 2, 3\}$ such $w_{c+1,\ell} \geqslant c+1$ and set $\boldsymbol{d}_{c+1} = \boldsymbol{d}'_{c+1} + \boldsymbol{g}_\ell$.

On the following $c^2 - 1$ writes, simply apply the basic update rule again, and so $\boldsymbol{d}_j = \boldsymbol{d}'_j$ for $c+2 \leqslant j \leqslant c^2 + c = T - 1$. On the the T-th write, we define, as in the $(c+1)$-th write for $\ell = 1, 2, 3$,

$$w_{T,\ell} = \left\lfloor \frac{T(a-1) - (i_{T,\ell} + \delta'_{T,\ell})}{b} \right\rfloor.$$

The increment vector $\boldsymbol{d}_T$ is determined as follows.

1) If for $\ell = 1, 2, 3$, $w_{T,\ell} > 0$, then $\boldsymbol{d}_T = \boldsymbol{d}'_T$.

2) Otherwise, there exists ℓ such $w_{T,\ell} = 0$. If $w_{T,\ell-1} \geqslant c+1$ then we set $\boldsymbol{d}_T = \boldsymbol{d}'_T + \boldsymbol{g}_{\ell-1}$, and else we set $\boldsymbol{d}_T = \boldsymbol{d}'_T + \boldsymbol{g}_{\ell-1} + \left(c - \left\lceil \frac{w_{T,\ell-1}-1}{c} \right\rceil\right) \boldsymbol{g}_{\ell-2}$.

The indices of the vectors $\boldsymbol{g}_1, \boldsymbol{g}_2, \boldsymbol{g}_3$ are calculated modulo 3 where the residues are 1, 2, or 3. For example, if $\ell = 1$ then $\boldsymbol{g}_{\ell-1} = \boldsymbol{g}_3$ and $\boldsymbol{g}_{\ell-2} = \boldsymbol{g}_2$.

The goal in the update function is to balance the increase of all cell levels. This is done in two stages. The first stage is on the $(c+1)$-th write, after which it is guaranteed that there are at least two cells whose levels are at most $(c+1)(a-1) - b$. This property is achieved by one of the options 1 and 2 listed above for the $(c+1)$-th write. In option 1, it is checked if there are already two cells with this property, in which case the basic update rule can be applied as is. Otherwise, in option 2 there must be (to be proved later) one cell whose level is small enough to allow a large increase in its level, while at the same time helping another cell gain at most $a - b - 1$ levels. The second stage is on the T-th write. On this write there are already two cells whose levels will be at most $\Delta_3 = T(a-1) - b$ if the basic increment vector $\boldsymbol{d}'_T$ is used. For the update function to be correct, we need this property for all three cells. In option 1 of the T-th write above indeed all three cells have this property, and we can use the basic update rule without change. Otherwise, in option 2 we assume the third cell does not satisfy this property. In that case we try to find an update vector that guarantees a small increment to the third cell, while keeping the other two still within the Δ_3 limit. The two cases in option 2 of the T-th write represent a direct compensation by a row-vector of G_3 in the former case, or by a linear combination of two row-vectors of G_3 in the latter case. We will prove the correction of these operations in the next lemmas.

Lemma 11: If the condition in option 1 of the $(c+1)$-th write does not hold, then the condition in option 2 holds. Furthermore, $\boldsymbol{d}_{c+1} \geqslant \boldsymbol{0}$.

Proof: If the condition in option 1 of the $(c+1)$-th write holds, then $\boldsymbol{d}_{c+1} = \boldsymbol{d}'_{c+1} \geqslant \boldsymbol{0}$ because we apply the basic update rule. If the condition in option 1 does not hold, let us show that there exists $\ell \in \{1, 2, 3\}$ such that $w_{c+1,\ell} \geqslant c+1$. Without loss of generality, assume that $w_{c+1,2} = w_{c+1,3} = 0$. This implies that none of the cells 2, 3 had an increment smaller than $a-b$ in the preceding writes or in the current write with the basic update rule. Since on every write the increment level of at least one of the cells is at most $a-b-1$, we get that for $1 \leqslant j \leqslant c$,

$$a-b \leqslant \delta_{j,2}, \delta_{j,3} \leqslant a-1, \quad \delta_{j,1} \leqslant a-b-1$$

and also

$$a-b \leqslant \delta'_{c+1,2}, \delta'_{c+1,3} \leqslant a-1, \quad \delta'_{c+1,1} \leqslant a-b-1.$$

Therefore,

$$i_{c+1,1} + \delta'_{c+1,1} = \sum_{j=1}^{c} \delta_{j,1} + \delta'_{c+1,1} \leqslant (c+1)(a-b-1) \quad (3)$$

and

$$w_{c+1,1} = \left\lfloor \frac{(c+1)(a-1) - (i_{c+1,1} + \delta'_{c+1,1})}{b} \right\rfloor \geqslant \frac{b(c+1)}{b} = c+1.$$

In particular, we have that $\delta'_{c+1,2} \geqslant a-b$ and thus $\delta_{c+1,2} = \delta'_{c+1,2} - b \geqslant a-2b \geqslant 0$. ■

Lemma 12: After the $(c+1)$-th write, the level of at least two cells is at most $(c+1)(a-1)-b$.

Proof: Assume the condition in option 1 of the $(c+1)$-th write holds, and let ℓ_1, ℓ_2 be such that $w_{c+1,\ell_1}, w_{c+1,\ell_2} \geqslant 1$. Then, for the ℓ_1-th cell we get

$$w_{c+1,\ell_1} = \left\lfloor \frac{(c+1)(a-1) - (i_{c+1,\ell_1} + \delta'_{c+1,\ell_1})}{b} \right\rfloor \geqslant 1,$$

or

$$(c+1)(a-1) - (i_{c+1,\ell_1} + \delta'_{c+1,\ell_1}) \geqslant b$$

and thus after the $(c+1)$-th write, the level of the ℓ_1-th cell, $i_{c+1,\ell_1} + \delta_{c+1,\ell_1}$, satisfies

$$i_{c+1,\ell_1} + \delta_{c+1,\ell_1} = i_{c+1,\ell_1} + \delta'_{c+1,\ell_1} \leqslant (c+1)(a-1) - b.$$

Similarly, we get $i_{c+1,\ell_2} + \delta_{c+1,\ell_2} \leqslant (c+1)(a-1) - b$. If the condition in option 1 does not hold, assume without loss of generality that $w_{c+1,2} = w_{c+1,3} = 0$ and $w_{c+1,1} \geqslant c+1$. The level of cell 1 after this write is given by $i_{c+1,1} + \delta'_{c+1,1} + a$. As in the proof of Lemma 11 for inequality (3) we get

$$\begin{aligned} i_{c+1,1} + \delta'_{c+1,1} + a &\leqslant (c+1)(a-b-1) + a \\ &= (c+1)(a-1) - bc - b + a = (c+1)(a-1) - b. \end{aligned}$$

For the level of cell 2, we simply get

$$i_{c+1,2} + \delta'_{c+1,2} - b \leqslant (c+1)(a-1) - b.$$

■

Lemma 13: On the T-th write, we have $\boldsymbol{d}_T \geqslant \boldsymbol{0}$.

Proof: As in the proof of Lemma 11, if the condition in option 1 holds, then $\boldsymbol{d}_T = \boldsymbol{d}'_T \geqslant \boldsymbol{0}$. Assume the condition in option 1 does not hold. According to Lemma 12, after the $(c+1)$-th write, the level of at least two of the cells is at most $(c+1)(a-1)-b$, and without loss of generality assume these are cells $1, 2$. Since $\delta_{j,1}, \delta_{j,2} \leqslant a-1$ for $c+2 \leqslant j \leqslant T-1$ and $\delta'_{T,1}, \delta'_{T,2} \leqslant a-1$, we have that $w_{T,1}, w_{T,2} \geqslant 1$. Therefore, if the condition in option 1 of the T-th write does not hold, then $w_{T,3} = 0$. Hence in every write between the $(c+2)$-th write and the T-th write, the increment level of either cell 1 or cell 2, according to the basic update rule, is at most $a-b-1$. Thus we get that

$$\begin{aligned} &w_{T,1} + w_{T,2} \\ &= \left\lfloor \frac{T(a-1) - (i_{T,1} + \delta'_{T,1})}{b} \right\rfloor + \left\lfloor \frac{T(a-1) - (i_{T,1} + \delta'_{T,1})}{b} \right\rfloor \\ &\geqslant 2 + c^2. \end{aligned}$$

In case $w_{T,2} \geqslant c+1$, we have

$$\boldsymbol{d}_T = \boldsymbol{d}'_T + \boldsymbol{g}_2 = (\delta'_{T,1}, \delta'_{T,2} + a, \delta'_{T,3} - b) \geqslant \boldsymbol{0},$$

since $\delta'_{T,3} \geqslant a-b$. The proof that $\boldsymbol{d}_T \geqslant \boldsymbol{0}$ in case that $w_{T,2} < c+1$ is also very similar. ■

Finally, we can prove the re-write properties of Construction 4 in the next theorem.

Theorem 14: On the first T writes, the increase in the maximum cell level is at most

$$\Delta_3 = T(a-1) - b.$$

Proof: We will show that after T writes, the level of every cell is at most $T(a-1)-b$. Assume the condition in option 1 of the T-th write holds. Then, for the ℓ-th cell, $\ell = 1, 2, 3$, we get that

$$w_{T,\ell} = \left\lfloor \frac{T(a-1) - (i_{T,\ell} + \delta'_{T,\ell})}{b} \right\rfloor \geqslant 1,$$

or

$$T(a-1) - (i_{T,\ell_1} + \delta'_{T,\ell_1}) \geqslant b$$

and hence the level of the ℓ-th cell satisfies

$$i_{T,\ell} + \delta_{T,\ell} = i_{T,\ell} + \delta'_{T,\ell} \leqslant T(a-1) - b.$$

Now, assume that the condition in option 1 of the T-th write does not hold, and without loss of generality assume that $w_{T,3} = 0$. According to Lemma 12 and as was shown also in the proof of Lemma 13, we necessarily have that $w_{T,1}, w_{T,2} \geqslant 1$ and $w_{T,1} + w_{T,2} \geqslant 2 + c^2$. We consider the two cases of this option. In case that $w_{T,2} \geqslant c+1$, then the levels of the three cells are given by

$$(i_{T,1} + \delta'_{T,1}, i_{T,2} + \delta'_{T,2} + a, i_{T,3} + \delta'_{T,1} - b).$$

According to $w_{T,1} \geqslant 1$, we have $i_{T,1} + \delta'_{T,1} \leqslant T(a-1) - b$. According to $w_{T,2} \geqslant c+1$, we have

$$i_{T,2} + \delta'_{T,2} + a \leqslant T(a-1) - (c+1)b + a = T(a-1) - b.$$

Lastly, we have that $i_{T,3} + \delta'_{T,1} - b \leqslant T(a-1) - b$.

The remaining case is when $w_{T,2} < c+1$, and so the levels of the three cells are given by

$$\begin{aligned} c_1 &= i_{T,1} + \delta'_{T,1} + \left(c - \left\lceil \frac{w_{T,2} - 1}{c} \right\rceil\right) a, \\ c_2 &= i_{T,2} + \delta'_{T,2} + a - \left(c - \left\lceil \frac{w_{T,2} - 1}{c} \right\rceil\right) b, \\ c_3 &= i_{T,3} + \delta'_{T,1} - b. \end{aligned}$$

TABLE II
NUMERICAL RESULTS OF THE 3-CELL CONSTRUCTION

a	b	c	T	Δ_3	M_3	$\mathcal{R}_{\text{write}}$	Upper Bound
2	1	2	7	6	7	$\frac{\log(7)}{3} = 0.93$	$\log\binom{13}{7}/7 = 1.53$
4	2	2	7	19	56	$\frac{\log(56)}{3} = 1.93$	$\log\binom{26}{7}/7 = 2.76$
6	3	2	7	32	189	$\frac{\log(189)}{3} = 2.52$	$\log\binom{39}{7}/7 = 3.41$
8	4	2	7	45	448	$\frac{\log(448)}{3} = 2.93$	$\log\binom{52}{7}/7 = 3.85$
10	5	2	7	58	875	$\frac{\log(875)}{3} = 3.25$	$\log\binom{65}{7}/7 = 4.19$
20	10	2	7	123	7000	$\frac{\log(7000)}{3} = 4.25$	$\log\binom{130}{7}/7 = 5.23$
6	2	3	13	63	208	$\frac{\log(208)}{3} = 2.56$	$\log\binom{76}{13}/13 = 3.62$

If $w_{T,2} = 1$ then since $w_{T,1} + w_{T,2} \geqslant 2 + c^2$, we have $w_{T,1} \geqslant c^2 + 1$ and thus

$$\begin{aligned} c_1 = i_{T,1} + \delta'_{T,1} + ca &\leqslant T(a-1) - (c^2+1)b + ca \\ &= T(a-1) - b, \\ c_2 = i_{T,2} + \delta'_{T,2} + a - cb &\leqslant T(a-1) - b + a - cb \\ &= T(a-1) - b, \\ c_3 = i_{T,3} + \delta'_{T,1} - b &\leqslant T(a-1) - b. \end{aligned}$$

If $2 \leqslant w_{T,2} < c+1$ then since $w_{T,1} + w_{T,2} \geqslant 2 + c^2$, we have $w_{T,1} \geqslant c^2 - c + 2$ and thus

$$\begin{aligned} c_1 = i_{T,1} + \delta'_{T,1} + (c-1)a &\leqslant T(a-1) - (c^2-c+2)b + (c-1)a \\ &= T(a-1) - 2b, \\ c_2 = i_{T,2} + \delta'_{T,2} + a - (c-1)b &\leqslant T(a-1) - 2b + a - cb + b \\ &= T(a-1) - b, \\ c_3 Z = i_{T,3} + \delta'_{T,1} - b &\leqslant T(a-1) - b. \end{aligned}$$

This completes the proof. ■

As a result of Theorem 14, we get that the sum-rate is given by

$$\mathcal{R}_{\text{sum}} = T \cdot \frac{q-1}{\Delta_3} \cdot \frac{\log(a^3-b^3)}{3} = \frac{T(q-1)}{T(a-1)-b} \cdot \frac{\log(a^3-b^3)}{3}.$$

When we normalize the sum-rate by the number of writes T, and substitute $T = 1 + c + c^2$ we get

$$\mathcal{R}_{\text{write}} = \frac{(q-1)}{(1+c+c^2)(a-1)-b} \cdot \frac{\log(a^3-b^3)}{3}.$$

Table II gives some numerical examples of the 3-cell Construction 4. The value of c is the ratio between a and b, i.e., $c = a/b$. For one cycle of T writes the number of levels, q, is $q = \Delta_3 + 1$, and M_3 is the number of possible values written on each write, which is the size of the three-dimensional chair, $M_3 = a^3 - b^3$. The write rates $\mathcal{R}_{\text{write}}$ listed in Table II are calculated as

$$\mathcal{R}_{\text{write}} = \frac{\log(M_3)}{3}.$$

Finally, the value $\log\binom{q+T-1}{T}/T$ in the last column is the rate upper bound of [7] normalized by the number of writes (and hence an upper bound on $\mathcal{R}_{\text{write}}$). Note that this upper bound is for the case where non-equal rate writes are allowed, and therefore may not be the true limit for fixed-rate codes like those of Construction 4. We further note that in general, the results are better when the number of writes is small. Thus, most of the results (with the exception of the last row) are given for $c = 2$. In this case, $a = 2b$, $T = 7$, $\Delta_3 = 7(2b-1) - b = 13b - 7$, and $M_3 = 7b^3$. Then the per-write rate is given by

$$\mathcal{R}_{\text{write}} = \frac{\log(M_3)}{3} \overset{c=2}{=} \frac{\log(7b^3)}{3} = \log b + 0.93,$$

while the upper bound is

$$\frac{1}{T}\log\binom{\Delta_3 + T}{T} \overset{c=2}{=} \frac{1}{7}\log\binom{13b}{7}.$$

Hence this upper bound is no greater than

$$\frac{1}{7}\log\binom{13b}{7} < \frac{1}{7}\log\left(\frac{(13b)^7}{7!}\right) = \log b + 1.94.$$

Therefore, the WOM codes we achieve for three cells and $c = 2$ are within at most an additive constant of 1.01 bits per write per cell from the upper bound. A similar analysis can be derived for other values of c, however the additive constant will be larger. This is demonstrated in the last row of Table II.

Another approach to evaluate our construction is by comparison to bounds on fixed-rate codes in the form of Theorem 9 (Section II), generalized to 3-cell codes. The resulting comparison indeed tightens the gaps somewhat, but we skip the exact numerical values to keep the presentation cleaner. In terms of comparing to known codes, unfortunately, the number of such constructions is limited, especially for a large number of writes. One example where we can compare the results is for the 7-write code at the top row of Table II, with $q = 7$ levels and a write rate of 0.93 (sum-rate of 6.55), while a known construction in [9] achieves a 7-write code with a smaller rate and only with $q = 8$ (i.e., one more level than in our construction).

V. 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, while 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 the best of our knowledge, this type of problem has not been addressed by coding before.

In the proposed hot/cold re-write model, we look for codes that provide the following features.

1) Store k_1 hot bits and k_2 cold bits on n cells.
2) Each cold bit can be written at most once.
3) The k_1 hot bits can be written (jointly) as many times as possible.
4) Cold-bit writes can be performed anywhere in the write sequence, and in any order.
5) Any bit can be read anytime.

In the remainder of the section we construct codes that address the above features of the hot/cold model. We note that hot/cold

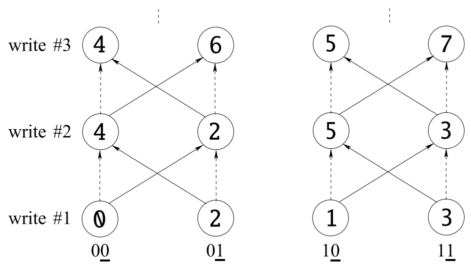
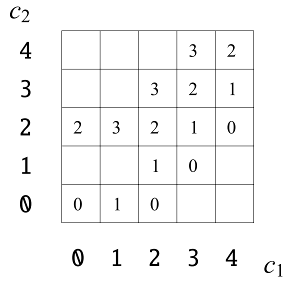


Fig. 4. Code for hot+cold bit in 1 cell with $\lfloor q/2 \rfloor$ total writes ($q = 8$ in the plot).



c_2 \ c_1	0	1	2	3	4
4				3	2
3			3	2	1
2	2	3	2	1	0
1			1	0	
0	0	1	0		

Fig. 5. Decoding function: 1-hot bit and 1-cold bit 2-cell code.

codes use a different counting of writes than WOM codes. We no longer have the notion of a write *generation*, but rather the rewrite capabilities are defined on subsets of the hot and cold stored bits.

A. One-Cell, 1-Hot Bit and 1-Cold Bit Storage

To understand the hot/cold model, we start with a simple example. In Figure 4 we show the level transition diagram of a 1-cell code for one hot bit and one cold bit. The stored information bits appear at the bottom of the figure, the right of which (underlined) is the hot bit. The integers inside the state nodes are the physical levels of the cell. Arrow lines specify level changes upon re-writes of the hot bit (solid lines represent hot-bit flips, and dashed lines represent re-writing of the same value to the hot bit). The left part of the diagram (even-numbered levels) is for a 0 value in the cold bit, and the right part (odd-numbered levels) is for a 1 value. When a 1 value is written to the cold bit, we move from the current state on the left to its "twin" state on the right side. This action flips the cold bit to 1 but leaves the hot bit unchanged. Once on the right part of Figure 4, the remaining writes of the hot bit are performed on that side, without reverting to the left part again in the write sequence (hence the restriction to have at most one write of the cold bit). Extending Figure 4 upward for a cell with q levels allows a total of $t = \lfloor q/2 \rfloor$ writes, including the one write of the cold bit. 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 [14] when q is even. Note that, as required by the hot/cold model, the t writes of the hot and cold bit can be performed in any order (moving to the right part of the diagram upon a cold write can be done at any level). The following proposition states that the code described in Figure 4 is optimal.

Proposition 15: Any 1-cell code that stores one hot bit and one cold bit guarantees at most $\lfloor q/2 \rfloor$ total writes.

Proof: Let a 1-hot and 1-cold bit code characterized by a decoding function $\mathcal{D} : \{0, 1, \ldots, q-1\} \to \{0, 1\}^2$, where for all $0 \leqslant c \leqslant q-1$, $\mathcal{D}(c) = (b_{c,0}, b_{c,1})$, and $b_{c,0}, b_{c,1}$ are the values of the cold, hot bit, respectively.
Let S_0, S_1 be the sets of all cell values that decode the cold bit to value 0, 1, respectively. That is,

$$S_0 = \{c \in \{0, 1, \ldots, q-1\} \ : \ b_{c,0} = 0\},$$
$$S_1 = \{c \in \{0, 1, \ldots, q-1\} \ : \ b_{c,0} = 1\}.$$

Assume first that q is even. If $|S_0| \leqslant |S_1|$ then $|S_0| \leqslant q/2$. Since $0 \in S_0$, we can have at most $q/2 - 1$ writes of the hot bit. That is $q/2$ including the cold-bit write. If $|S_1| < |S_0|$, then $|S_1| \leqslant q/2 - 1$. Now, suppose on the first write we update the cold bit and move to a level in S_1. Then since there are at most $q/2 - 2$ additional cell levels in S_1, the number of hot-bit writes is at most $q/2 - 2$. That is $q/2 - 1$ writes in total.

Now, assume that q is odd. If $S_0 \leqslant (q-1)/2$ then as before, the number of hot-bit writes is at most $(q-1)/2 - 1$, and the total at most $(q-1)/2 = \lfloor q/2 \rfloor$. Otherwise, $S_0 \geqslant (q+1)/2$ and $S_1 \leqslant (q-1)/2$ and the number of hot-bit writes after a cold-bit write is at most $(q-1)/2 - 1$. Again adding to a total of $(q-1)/2 = \lfloor q/2 \rfloor$. ■

B. Two-Cell, 1-Hot Bit and 1-Cold Bit Storage

Moving beyond 1-cell codes given in the preceding subsection, we now consider hot/cold codes that use multiple cells. This will allow better utilization of the memory cells by codes that make the cost of storing the cold bits negligible (in the code of Figure 4 the existence of the cold bit decreased the number of writes to the hot bit by a significant factor 2, compared to a 1-cell code with one hot bit.).

The first construction is now given. As in the 2-cell codes of earlier sections, the decoding function will be specified by a two-dimensional array. The coordinates at the exterior of the array represent physical-cell levels, and the integers within the array are the information values of the stored bits. In particular, for Construction 5 below the information value 0 stands for 0<u>0</u>, 1 stands for 0<u>1</u>, 2 stands for 1<u>0</u> and 3 stands for 1<u>1</u>. The underlined least-significant bit is the hot bit, and the most-significant bit is the cold bit. Note that for re-writing of the hot bit the code needs to support the transitions

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

In addition, for the write of the cold bit the code should support a single transition of the form $0 \to 2$ or $1 \to 3$.

Construction 5: For any q, we define a 2-cell, 1-hot bit and 1-cold bit code as follows.

1. Decoding Function: Shown pictorially in Figure 5.

Formally, denote the value of the hot bit by b_0 and the value of the cold bit by b_1. The levels of the two cells are denoted by c_1 and c_2. The decoding function $\mathcal{D}(c_1, c_2) = (b_0, b_1)$ (as specified in Figure 5) is given as:

1) $\mathcal{D}(0,0) = (0,0)$.

2) For all $(c_1, c_2) \neq (0,0)$,

a) $b_1 = (c_1 + c_2) \bmod 2$.

b) If $c_1 > c_2$ then $b_0 = 0$ and if $c_1 \leqslant c_2$ then $b_0 = 1$.

2. Update Function: The update function is denoted by $\mathcal{E}(c_1, c_2, i) = (c_1', c_2')$. The current memory state is (c_1, c_2) and the bit index to be changed is $i \in \{0, 1\}$. We assume here that b_0 changes at most once, and when a bit is written, 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 update function.

1) If $i = 0$ (cold-bit write), then $(c_1', c_2') = (c_1, c_2 + 2)$.

2) If $i = 1$ (hot-bit write), 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 5. The five 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 properties of Construction 5.

Lemma 16: the following hold throughout the update sequence of Construction 5.

1) When $b_0 = 0$, 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 $b_0 = 1$, 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 5. ■

Now we prove the re-write properties of Construction 5.

Proposition 17: The number of writes guaranteed by Construction 5 is $t = 2(q-1) - 1$, including the possible rewrite of the cold bit.

Proof: From Lemma 16 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 16 part 2, if the cold bit changes its value, then writing stops when the memory hits 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 18: Any 2-cell code for 1-hot bit and 1-cold bit guarantees at most $2(q-1)-1$ writes.

Proof: Assume to the contrary that there exists a code that 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 option. Therefore, the decoded value of the memory state $(q-1, q-2)$ is $(b_0, b_1) = (0, 1)$. On the following write, both a cold-bit write and a hot-bit write are legal, so the two-bit value may change to either $(0,0)$ or $(1,1)$. But there is only one memory state $(q-1, q-1)$ that is accessible from $(q-1, q-2)$, which leads to a contradiction. ■

Proposition 17 proves that a 2-cell code for one hot bit and one cold bit can give $2(q-1)-1$ guaranteed total writes. When the same two cells are used to re-write only a hot bit (without the cold bit), the number of writes is $2(q-1)$, just one more than in the 1-hot, 1-cold setting. Consequently, the addition of a cold bit to the code has a negligible effect on the total number of writes. It is thus a very attractive property that adding cold bits to re-write codes comes with no additional storage cost, and while maintaining essentially the same re-write properties.

It is also academically interesting to compare the 1-hot, 1-cold Construction 5 to 2-cell floating codes with 2 input bits introduced in [14]. The common property of the two codes is that each write is an update of one of two stored bit. The difference is that floating codes do not distinguish between hot and cold bits, and allow any sequence of t 1-bit updates. The number of writes guaranteed by 2-cell floating codes is $\lfloor 3(q-1)/2 \rfloor$. As a result, Construction 5 shows that by designating one of the input bits as a cold bit, it is possible to increase the number of writes by 33%.

C. Multiple Cold Bits and a Single Hot Bit

In this sub-section we would like to extend the 2-cell hot+cold Construction 5 of the previous sub-section such that it will be possible to store multiple cold bits and a single hot bit. Such codes address the common case in practice where the hot bits represent a small fraction of the total storage space.

A simple idea for multiple cold bits code is to take k copies of Construction 5, that is, using $2k$ cells in total. In every pair of cells, a single cold bit and a single hot bit are stored. Since we only need to store a single hot bit, we take its value to be the sum modulo 2 of the k hot bits of the individual copies. Thus, it is possible to store a single hot bit and k cold bits in $n = 2k$ cells with $k(2q-3) = n(q-1)-k$ total writes. Despite the simplicity of this idea, using two cells for every cold bit is too wasteful. Next we show a construction for k cold bits and 1 hot bit using as few as $n = k+1$ cells. Let us denote the cell levels by $c_0, c_1, \ldots, c_k$. The construction builds on the idea just presented, but instead of using k copies of a 2-cell code, we use the cell c_0 as a shared 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. We denote the hot bit by b_k and the cold bits by $b_0, b_1, \ldots, b_{k-1}$. The decoding and update functions of Construction 5 will be used in the following construction, and are denoted therein by $\mathcal{D}(r, s)$ and $\mathcal{E}(r, s, m)$, respectively.

Construction 6: For any q and k, we define a $(k+1)$-cell, 1-hot bit and k-cold bits code as follows.

1. Decoding Function: *The decoding function $\mathcal{D}^*(c_0, c_1, \ldots, c_k) = (b_0, b_1, \ldots, b_k)$ is defined as follows.*

1) $b_k = (\sum_{i=0}^{k} c_i) \bmod 2$.
2) *For* $0 \leqslant i \leqslant k-1$, $b_i = \mathcal{D}(c_0, c_{i+1})_0$ *(the first bit of the decoded pair cf. the decoding function of Construction 5).*

2. Update Function: *The update function $\mathcal{E}^*(c_0, c_1, \ldots, c_k, m) = (c'_0, c'_1, \ldots, c'_k)$, is defined as follows, where $0 \leqslant m \leqslant k$ is the index of the updated bit, 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) *$m \neq k$ (cold): then $(c'_0, c'_{m+1}) = \mathcal{E}(c_0, c_{m+1}, 1) = (c_0, c_{m+1}+2)$.*
2) *$m = k$ (hot): if there exists $1 \leqslant i \leqslant k$ such that $\mathcal{E}(c_0, c_i, 0) = (c_0, c_i+1)$, then set $c'_i = c_i+1$ and $c'_j = c_j$ for all $j \neq i$ in the range $0 \leqslant j \leqslant k$. Otherwise (that is, for all $1 \leqslant i \leqslant k$, $\mathcal{E}(c_0, c_i, 0) = (c_0+1, c_i)$), 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 Construction 6 works for $k = 4$ ($n = 5$), and $q = 5$. Thus, we store a single hot bit b_0 and four cold bits b_1, b_2, b_3, b_4 in five 5-ary cells c_0, c_1, c_2, c_3, c_4.*

Updated Bit #	*Cell Levels* $(c_0, c_1, c_2, c_3, c_4)$	*Bit Values* $(b_0, b_1, b_2, b_3, b_4)$
	(0, 0, 0, 0, 0)	(0, 0, 0, 0, 0)
2	(0, 0, 0, 2, 0)	(0, 0, 1, 0, 0)
0	(0, 2, 0, 2, 0)	(1, 0, 1, 0, 0)
4	(1, 2, 0, 2, 0)	(1, 0, 1, 0, 1)
4	(2, 2, 0, 2, 0)	(1, 0, 1, 0, 0)
4	(2, 3, 0, 2, 0)	(1, 0, 1, 0, 1)
4	(2, 3, 1, 2, 0)	(1, 0, 1, 0, 0)
4	(2, 3, 1, 3, 0)	(1, 0, 1, 0, 1)
4	(2, 3, 1, 3, 1)	(1, 0, 1, 0, 0)
3	(2, 3, 1, 3, 3)	(1, 0, 1, 1, 0)
4	(3, 3, 1, 3, 3)	(1, 0, 1, 1, 1)
4	(3, 3, 2, 3, 3)	(1, 0, 1, 1, 0)
4	(3, 4, 2, 3, 3)	(1, 0, 1, 1, 1)
4	(3, 4, 2, 4, 3)	(1, 0, 1, 1, 0)
1	(3, 4, 4, 4, 3)	(1, 1, 1, 1, 0)
4	(3, 4, 4, 4, 4)	(1, 1, 1, 1, 1)
4	(4, 4, 4, 4, 4)	(1, 1, 1, 1, 0)

The table shows a sequence of bit updates (left column) and the resulting updated cell levels (center column). The values of all 5 bits after the write are shown on the right column.

The correctness of Construction 6 is proved in the following two lemmas.

Lemma 19: *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*

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

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. Now we need to show that the cold bits are not affected. The correctness of the update function comes from the property that throughout the write sequence $\mathcal{D}(c_0, c_{i+1})_0 = b_i$. This is clearly true for the initial state $(c_0, \ldots, c_k) = (0, \ldots, 0)$, and we now show that it is kept as an invariant property after every update step.

When the hot bit is updated by applying the update function $\mathcal{E}(c_0, c_i, 0) = (c_0, c_i+1)$ for some i, then necessarily the value of all the cold bits besides the $(i-1)$-th bit do not change. However, the value of the $(i-1)$-th bit does not change either, because the pair (c_0, c_i) is updated according to $\mathcal{E}$, thus maintaining the invariant property. Alternatively, when the hot bit is updated by incrementing $c_0 \to c_0+1$, all pairs (c_0, c_i) are affected, and we need to show that all cold bits b_i remain unchanged. This is guaranteed by the fact that this update rule is only chosen when $\mathcal{E}(c_0, c_i, 0) = (c_0+1, c_i)$ for all i, and thus applying this update will maintain the invariant property for all pairs (c_0, c_i) simultaneously. ■

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

Lemma 20: *If $\mathcal{D}^*(c_0, c_1, \ldots, c_k) = (b_0, b_1, \ldots, b_k)$ and the m-th bit, $0 \leqslant m \leqslant k-1$, changes its value (for the first time) then*

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

Proof: First note that when a cold bit changes its value, then the sum of the cell levels increases by two, and from the modulo 2 addition the value of the hot bit b_k does not change. Since only c_{m+1} is updated, all cold bits other than b_m remain unchanged. Finally, to prove that b_m changes to 1 we note that before the write $c_{m+1} \geqslant c_0 - 2$ from Lemma 16 and the invariant property proved in Lemma 19, and thus $c'_{m+1} = c_{m+1} + 2 \geqslant c_0$. Hence, from the definition of $\mathcal{D}$ the value of the cold bit b_m is 1. ■

The number of writes of Construction 6 is proved in the next theorem.

Theorem 21: *Construction 6 guarantees $t = n(q-1) - k$ total writes.*

Proof: Each cold-bit write adds 2 to a cell $i \in \{1, \ldots, k\}$. Throughout the write sequence each cell $i \in \{1, \ldots, k\}$ is incremented by 2 at most once. From the invariant property that $\mathcal{E}^*(c_0, c_1, \ldots, c_k, 0)$ is either $(c_0, c_1, \ldots, c_i+1, \ldots, c_k)$ for some i or $(c_0+1, c_1, \ldots, c_k)$, each of the remaining $q-3$ increments of each cell $i \in \{1, \ldots, k\}$ can be used for hot writes, as well as all $q-1$ increments of c_0. In total the number of hot writes is thus $k(q-3)+q-1$, and adding the k cold-bit writes we get

$$k + k(q-3) + q - 1 = (k+1)(q-1) - k = n(q-1) - k$$

total writes. ■

D. Two Hot Bits and One Cold Bit

In this part we extend the 2-cell 1-hot and 1-cold bit Construction 5 to storing 2 hot bits and 1 cold bit in two cells. According to the hot/cold model, the two hot bits are updated (together, as a group) as many times as possible, while the cold bit is updated up to once, and anywhere in the write sequence. The 3-bit information value (b_0, b_1, b_2) stored in the

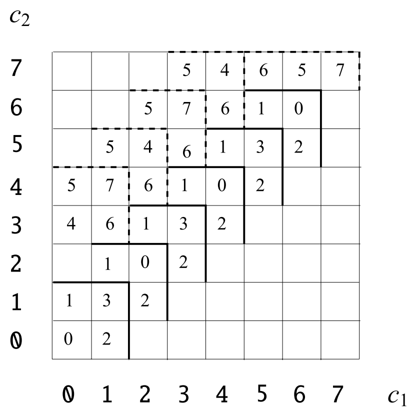


Fig. 6. Decoding function: 2-hot bits and 1-cold bit 2-cell code.

memory will be represented by an integer from $\{0, \ldots, 7\}$. The hot bits b_1, b_2 are the two least significant bits in the binary representation of this integer, and the cold bit b_0 is the most significant bit (hence the cold bit is set to 1 when the value is $\geqslant 4$ and to 0 otherwise). We now present the code construction.

Construction 7: For any q, we define a 2-cell, 2-hot bits and 1-cold bit code as follows.

1. Decoding Function: $\psi(c_1, c_2)$ specified pictorially in Figure 6.

2. Update Function: Given current levels (c_1, c_2) and input value m, find the the new cell levels (c_1', c_2') that satisfy the following conditions.

1) *$c_1' \geqslant c_1, c_2' \geqslant c_2$.*
2) *$\psi(c_1', c_2') = m$ as defined in Figure 6.*
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.*

The update function of Construction 7 is the natural one given the decoding function of Figure 6. In each update of the hot bits, we go diagonally up and to the right to the nearest value equal to the input. In the update of the cold bit, we go vertically upward (and if necessary, as happens after the later hot writes, also to the right) to the value that equals the current value plus 4. To verify the number of writes of Construction 7, we see that the j-th write of the hot bits progresses the levels not beyond the j-th frontier marked in Figure 6 by right-angled solid/dashed lines. A cold write brings us within the corresponding dashed frontier vertically above. The only exception of the regular square shaped frontiers happens at the last write, where the dashed frontier becomes a rectangle. Altogether it is not hard to see that the following result applies.

Proposition 22: Construction 7 guarantees $q-3$ writes of the two hot bits and one write of the cold bit, or $q-2$ writes of the hot bits.

As previous hot/cold constructions, Construction 7 also shows that the cost of adding a cold bit to a re-write code in terms of the number of writes is insignificant ($q-3$ or $q-2$ writes vs. $q-1$ when the code has just the hot bits). In the following we show that Construction 7 achieves an optimal number of writes.

Proposition 23: Any code that guarantees t writes of two hot bits and one write of a cold bit, or $t+1$ writes of the two hot bits must have $t \leqslant q-3$.

Proof: The number of writes when storing only two bits in two cells is at most $q-1$. Hence, we already have that $t \leqslant q-2$. Assume to the contrary that $t = q-2$. Let us consider a write sequence where on the first $q-2$ writes only the two hot bits are updated and on every write we choose an update of the two hot bits that increases the sum of the levels in the two cells by at least 2. Hence, after these t writes, the sum of levels is at least $2(q-2)$, and the memory state can only be one of the following: $(q-2, q-2), (q-1, q-3), (q-3, q-1), (q-2, q-1), (q-1, q-2), (q-1, q-1)$. On the $(t+1)$-th write, we can either write the two bits again or write the cold bit. Thus, from the current memory state there should be at least four different memory states that are reachable. However, from none of these six memory states it is possible to reach four more different memory states, which leads to contradiction. Hence, $t \leqslant q-3$. ■

VI. Conclusion

The present paper advances our constructive knowledge on short re-write codes with fixed input sizes, a model we believe is the most applicable to practical multi-level flash storage. Easy to implement constructions for general alphabet sizes q are given with precise guarantees on the number of writes. Many of the constructions are shown to be optimal, and others close to optimality. In the hot/cold model we show, in a series of optimal constructions, that adding cold bits to re-write codes incurs little cost, and thus joint hot/cold storage is an attractive solution to achieving wear leveling between input bits of different update characteristics. Continuing the progress started here, important research directions lie ahead. Lattice-tiling codes need to be extended to dimensions beyond 3, and their gaps to optimality need to be further studied and reduced. As far as hot/cold codes are concerned, a more complete theory of constructions and limits ought to be developed on the foundations presented here.

Acknowledgement

The authors wish to thank Tuvi Etzion for valuable discussions, and the anonymous reviewers for improving the clarity of the results.

References

[1] A. Bhatia, A. Iyengar, and P. H. Siegel, "Multilevel 2-cell t-write codes," in *Proc. IEEE Information Theory Workshop*, Lausanne, Switzerland, Sep. 2012, pp. 247–251.

[2] D. Burshtein and A. Strugatski, "Polar write once memory codes," in *Proc. IEEE Int. Symp. Information Theory*, Cambridge, MA, USA, Jul. 2012, pp. 1972–1976.

[3] S. Buzaglo and T. Etzion, "Tilings with n-dimensional chairs and their applications to WOM codes," *IEEE Trans. Inf. Theory*, vol. 59, no. 3, pp. 1573–1582, Mar. 2013.

[4] Y. Cassuto and E. Yaakobi, "Short Q-ary WOM codes with hot/cold write differentiation," in *Proc. IEEE Int. Symp. Information Theory*, Cambridge, MA, USA, Jul. 2012, pp. 1396–1400.

[5] G. D. Cohen, P. Godlewski, and F. Merkx, "Linear binary code for write-once memories," *IEEE Trans. Inf. Theory*, vol. 32, no. 5, pp. 697–700, Oct. 1986.

[6] A. Fiat and A. Shamir, "Generalized write-once memories," *IEEE Trans. Inf. Theory*, vol. 30, no. 3, pp. 470–480, May 1984.

[7] 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 Trans. Inf. Theory*, vol. 45, no. 1, pp. 308–313, Jan. 1999.

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

[9] R. Gabrys *et al.*, "Non-binary WOM-codes for multilevel flash memories," in *Proc. IEEE Information Theory Workshop*, Paraty, Brazil, Oct. 2011, pp. 40–44.

[10] P. Godlewski, "WOM-codes construits à partir des codes de Hamming," *Discrete Math.*, vol. 65, no. 3, pp. 237–243, Jul. 1987.

[11] K. Haymaker and C. A. Kelley, "Geometric WOM codes and coding strategies for multilevel flash memories," *Design Codes Cryptography*, vol. 70, pp. 91–104, Jan. 2014.

[12] C. Heegard, "On the capacity of permanent memory," *IEEE Trans. Inf. Theory*, vol. 31, no. 1, pp. 34–42, Jan. 1985.

[13] Q. Huang, S. Lin, and K. A. S. Abdel-Ghaffar, "Error-correcting codes for flash coding," *IEEE Trans. Inf. Theory*, vol. 57, no. 9, pp. 6097–6108, Sep. 2011.

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

[15] A. Jiang *et al.*, "Storage coding for wear leveling in flash memories," *IEEE Trans. Inf. Theory*, vol. 56, no. 10, pp. 5290–5299, Oct. 2010.

[16] A. Jiang, M. Langberg, M. Schwartz, and J. Bruck, "Trajectory codes for flash memory," *IEEE Trans. Inf. Theory*, vol. 59, no. 7, pp. 4530–4541, Jul. 2013.

[17] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for flash memories," *IEEE Trans. Inf. Theory*, vol. 55, no. 6, pp. 2659–2673, Jun. 2009.

[18] B. Kurkoski, "Lattice-based WOM codebooks that allow two writes," in *Proc. Int. Symp. Information Theory and Its Applications*, Honolulu, HI, USA, Oct. 2012.

[19] B. Kurkoski, "Notes on a lattice-based WOM construction," in *Proc. 34th Symp. Information Theory and Its Applications*, Iwate, Japan, Nov./Dec. 2011, pp. 520–524.

[20] B. Kurkoski, "Rewriting codes for flash memories based upon lattices, and an example using the E8 lattice," in *Proc. IEEE Globecom, ACTEMT Workshop*, Dec. 2010, pp. 1–5.

[21] F. Merkx, "Womcodes constructed with projective geometries," *Traitement Signal*, vol. 1, no. 2, pp. 227–231, 1984.

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

[23] A. Shpilka. (2012, Sep.). *Capacity achieving multiwrite WOM codes*. arXiv:1209.1128 [Online]. Available: http://arxiv.org/abs/1209.1128

[24] A. Shpilka, "New constructions of WOM codes using the Wozencraft ensemble," in *Proc. Latin American Symp. Theoretical Informatics*, Arequipa, Peru, Apr. 2012.

[25] S. Stein and S. Szabo, *Algebra and Tiling*. Washington, DC, USA: Math. Assoc. Amer., 1994.

[26] Y. Wu and A. Jiang, "Position modulation code for rewriting write-once memories," *IEEE Trans. Inf. Theory*, vol. 57, no. 6, pp. 3692–3697, Jun. 2011.

[27] E. Yaakobi, S. Kayser, P. H. Siegel, A. Vardy, and J. K. Wolf, "Codes for write-once memories," *IEEE Trans. Inf. Theory*, vol. 58, no. 9, pp. 5985–5999, Sep. 2012.

Yuval Cassuto (S'02–M'08) is a faculty member at the Department of Electrical Engineering, Technion–Israel Institute of Technology. His research interests lie at the intersection of the theoretical information sciences and the engineering of practical computing and storage systems.

During 2010–2011 he has been a Scientist at EPFL, the Swiss Federal Institute of Technology in Lausanne.

From 2008 to 2010 he was a Research Staff Member at Hitachi Global Storage Technologies, San Jose Research Center.

He received the B.Sc degree in Electrical Engineering, summa cum laude, from the Technion, Israel Institute of Technology, in 2001, and the MS and Ph.D degrees in Electrical Engineering from the California Institute of Technology, in 2004 and 2008, respectively.

From 2000 to 2002, he was with Qualcomm, Israel R&D Center, where he worked on modeling, design and analysis in wireless communications.

Dr. Cassuto has won the 2010 Best Student Paper Award in data storage from the IEEE Communications Society, as well as the 2001 Texas Instruments DSP and Analog Challenge $100,000$ prize.

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

He is currently a postdoctoral researcher in electrical engineering at the California Institute of Technology, Pasadena and he is also affiliated with the Center for Magnetic Recording Research at the University of California, San Diego. His research interests include information and coding theory with applications to non-volatile memories, associative memories, data storage and retrieval, and voting theory. He received the Marconi Society Young Scholar in 2009 and the Intel Ph.D. Fellowship in 2010-2011.