High Sum-Rate Three-Write and Non-Binary WOM Codes

Eitan Yaakobi[*†], **Amir Shpilka**[‡]

[*]Dept. of Electrical Engineering
California Institute of Technology
Pasadena, CA 91125, U.S.A.

[†]Dept. of Electrical and Computer Engineering
University of California, San Diego
La Jolla, CA 92093, U.S.A.

[‡]Dept. of Computer Science
Technion – Israel Institute of Technology
Haifa, Israel 32000

***Abstract*—Write-once memory (*WOM*) is a storage medium with memory elements, called *cells*, which can take on q levels. Each cell is initially in level 0 and can only increase its level. A *t-write WOM code* is a coding scheme which allows one to store t messages to the WOM such that on consecutive writes every cell's level does not decrease. The *sum-rate* of the WOM code, which is the ratio between the total amount of information written in the t writes and the number of memory cells, is bounded by $\log_2(t+1)$.**

Our main contribution in this work is a construction of binary three-write WOM codes with sum-rate approaching 1.885 for sufficiently large number of cells, while the upper bound is 2. This improves upon a recent construction of sum-rate 1.809. We also give constructions of non-binary WOM codes which give better sum-rate than the currently best known ones.

I. Introduction

Write-once memories (WOM) were first introduced in 1982 by Rivest and Shamir [11]. Their motivation came from storage medium like punch cards and optical disks. These media are comprised of storage elements, which we call *cells*. The most distinguishable property of these cells is their asymmetry programming attribute. In the binary version, it is only allowed to irreversibly program each cell from value zero to value one. If a cell can accommodate more than two levels, then on each programming operation, it is only possible to increase the cell level. A WOM code is a coding scheme which allows one to store information and reuse the WOM more than once, while preserving the property that cells are only allowed to increase their levels on each write.

The most famous example of a WOM code is the one given by Rivest and Shamir for storage of two bits twice using only three cells [11] (Table I). In their work, they also analyzed the bounds on the amount of information possible to store in a WOM and presented more codes. Since then, more constructions were given in 1980's and 1990's, e.g., [2], [10] as well as capacity analysis, e.g., [4], [7].

TABLE I
A WOM code Example

Data bits	First write	Second write (if data changes)
00	000	111
10	100	011
01	010	101
11	001	110

A renewed interest in WOM codes came along in the past five years as a result of the tremendous research work on coding for flash memories. Flash memory is another example of a WOM where its cells are charged with electrons and thus represent multiple levels [1]. Increasing a cell level is fast and easy; however, in order to decrease its level, its entire containing block of cells has to be erased. This does not only affect the writing speed of the flash memory but also significantly reduces its lifetime [1].

Assume t messages are stored to the memory, consisting of n cells. On the i-th write, $1 \leqslant i \leqslant t$, the message size is M_i. The *rate* on the i-th write is defined to be $\mathcal{R}_i = \frac{\log_2 M_i}{n}$, and the *sum-rate* is $\mathcal{R}_{\text{sum}} = \sum_{i=1}^{t} \mathcal{R}_i$. The capacity region of a t-write WOM is the set of all achievable rate tuples. For the binary case, the capacity region was found in [4], [7], [11]. It was also proved that the maximum achievable sum-rate for a binary WOM code with t writes is $\log_2(t+1)$. These results were generalized in [4] for non-binary WOM and the maximum sum-rate for WOM with cells of q levels is $\log_2 \binom{t+q-1}{t}$.

The main goal in designing a WOM code is to achieve high sum-rate, while decreasing its encoding and decoding complexities. In [15], motivated by the constructions in [2] and [13], a capacity achieving two-write WOM code construction was presented, and in [9], all previous results for more than two writes were improved. Recently, in [12], yet another two-write capacity achieving construction was given. This construction is more attractive than the one in [15] as it achieves a better block length as well as efficient encoding and decoding complexity.

For three-write WOM codes, the best sum-rate that can be found according to [15] was 1.66 while the upper bound is 2. The work in [12] showed another scheme which improved the sum-rate to be 1.809. Our main contribution in this paper is to use ideas similar to those in [12] in order to give a construction of three-write WOM code with sum-rate approaching 1.885 for sufficiently large number of cells.

While the binary case received most of the attention in the literature, only a little is known for non-binary WOM codes. This problem was already proposed by Rivest and Shamir in their pioneering work [11]. The information theory limits under constrained sources were analyzed in [3] and the capacity region was studied in [4]. However, first WOM code constructions, based on error-correcting codes, were given by Huang et al. in [8], only a year ago. These results were then improved in [6]. In [5], Gabrys and Dolecek gave a construction of two-write WOM code for $q = 3$ and improved some of the bounds on the sum-rate in case the rates on each write are the same. We show in this work how to improve the currently best known constructions for two-write non-binary WOM codes. We also report on more efficient constructions for multiple writes, however with no details due to the lack of space.

The rest of the paper is organized as follows. In Section II, we briefly review the definitions of WOM codes and state the results which we will use in our constructions. In Section III, we show our main result of a three-write WOM code con-

978-1-4673-2579-0/12/$31.00 ©2012 IEEE

struction with sum-rate 1.885. In Section IV, we present our constructions for non-binary WOM codes. Finally, Section V concludes the paper.

II. Definitions and Basic Properties

In this work, the cells can have two or more (q) levels. Initially, all cells are in level zero. On each write, it is only possible to increase the levels of each cell. A vector $\boldsymbol{x} = (x_1, \ldots, x_n) \in \{0, \ldots, q-1\}^n$ will be called a *cell-state vector*. For two cell-state vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, we say that $\boldsymbol{x} \leqslant \boldsymbol{y}$ if $x_i \leqslant y_i$ for all $1 \leqslant i \leqslant n$. For $j \leqslant i$, we use the notation $[j:i]$ to define the set of integers $\{j, \ldots, i\}$. If x represents a bit value then its complement is $\overline{x} = 1 - x$, and for a binary vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, $\overline{\boldsymbol{x}} = (\overline{x}_1, \ldots, \overline{x}_n)$. For any map $f : A \to B$, $Im(f)$ is the image of the map f.

Definition. *An $[n, t; M_1, \ldots, M_t]_q$ t-write WOM code is a coding scheme comprising of n q-ary cells and is defined by t pairs of encoding and decoding maps $(\mathcal{E}_i, \mathcal{D}_i)$, for $1 \leqslant i \leqslant t$. The encoding map $\mathcal{E}_i$ is defined by*

$$\mathcal{E}_i : [1 : M_i] \times Im(\mathcal{E}_{i-1}) \to \{0, \ldots, q-1\}^n,$$

where, by definition, $Im(\mathcal{E}_0) = \{(0, \ldots, 0)\}$, such that $\mathcal{E}_i(m, \boldsymbol{c}) \geqslant \boldsymbol{c}$ for all $(m, \boldsymbol{c}) \in [1 : M_i] \times Im(\mathcal{E}_{i-1})$. Similarly, the decoding map $\mathcal{D}_i$ is defined by

$$\mathcal{D}_i : Im(\mathcal{E}_i) \to [1 : M_i],$$

such that for all $(m, \boldsymbol{c}) \in [1 : M_i] \times Im(\mathcal{E}_{i-1})$, $\mathcal{D}_i(\mathcal{E}_i(m, \boldsymbol{c})) = m$. The rate on the i-th write is defined by $\mathcal{R}_i = \frac{\log_2 M_i}{n}$, and the sum-rate is $\mathcal{R}_{sum} = \sum_{i=1}^{t} \mathcal{R}_i$.

We assume that the write number is known at every write as this does not affect the achievable rates; for more details, see [15].

In [4], [7], the capacity region of a binary t-write WOM was found and $\log_2(t+1)$ was proved to be the maximum sum-rate. In [4], it was shown that the maximum sum-rate for q levels is $\log_2 \binom{q+t-1}{t}$. It was also shown that all rate-tuples in the capacity region are achievable. However, the problem of finding efficient code construction still remains a challenge.

For binary two-write WOM, the best sum-rate reported in the literature is 1.4928 [15] and a code construction which is capacity achieving was given. Recently, the work in [12] shows another capacity achieving construction which its code length and encoding and decoding complexities are significantly better. The capacity region of a two-write WOM is given by

$$C_2 = \left\{(\mathcal{R}_1, \mathcal{R}_2) | \mathcal{R}_1 \leqslant h(p), \mathcal{R}_2 \leqslant (1-p), 0 \leqslant p \leqslant 1/2\right\},$$

and from [12] we have the following theorem.

Theorem 1. [12] *For any $0 < p < 1$ and $\epsilon > 0$, there exists a WOM code $\mathcal{C}_p$ with rates $(h(p) - \epsilon, 1 - p - \epsilon)$.*

The capacity achieving constructions in [15] and in [12] use similar ideas. If the WOM has n cells, then on the first write approximately $h(p)n$ bits are stored in the memory in a way that at most pn cells are programmed. Then, on the second write, it is shown how to store $(1-p)n$ bits on the remaining $(1-p)n$ cells. However, while the construction in [15] restricts on the first write which of the pn cells can be programmed, in [12] any of the pn cells can be programmed. This useful property can be used in showing the next lemma, which can be used to construct multiple-write WOM codes.

Lemma 2. *If there exists a t-write WOM code with n cells such that on the first $t-1$ writes at most pn cells are programmed, then it is possible to write on the last write $(1-p)n$ bits.*

Proof: We simply treat the first $t-1$ writes as the first write of the capacity achieving construction in [12]. Then, the last write is performed as the second write of the construction in [12] and thus it is possible to store $(1-p)n$ more bits. ■

Remark 1. In the forthcoming constructions, it may be easier to present the memory input $\boldsymbol{w}$ as a string in $M = \{0, \ldots, a-1\}^m$. If $\boldsymbol{w}$ can be any word in M then it represents $m \log_2 a$ bits. However, in many cases we will choose only words $\boldsymbol{w}$ in M such that the number of times each symbol $0 \leqslant \ell \leqslant a-1$ appears in $\boldsymbol{w}$ is fixed to be $p_\ell m$, where m is large enough and $\sum_{\ell=0}^{a-1} p_\ell = 1$. Then, we will assume that the message $\boldsymbol{w}$ represents $m \cdot \mathcal{H}(p_0, \ldots, p_{a-1})$ bits, where $\mathcal{H}$ is the entropy function. Even though this is not exactly accurate as there is a $O(\log m)$ factor that has to be subtracted, we will use this approximation in case m is a linear function of the number of cells n. Note that the degradation in the sum-rate is negligible as the block length goes to infinity. Furthermore, a mapping between all such words $\boldsymbol{w}$ and the number of bits these words can represent should be given as well, however this part is out of the scope of this paper.

III. Three-Write WOM Codes

In this section, we present our construction of three-write WOM codes. As we use similar ideas, let us first briefly review the three-write WOM code construction in [12].

The construction in [12] uses $n = 3m$ cells partitioned into m three-cell blocks. In every three-cell block, the Rivest Shamir construction is used such that on the first two writes, the input to every block is two bits, stored as shown in Table I. Alternatively, we can think about the input as a quaternary symbol. In general, if any two quaternary symbols are written to a three-cell block, then in the worst case, all three cells are programmed. However, if, for example, on the first write, the symbol zero is stored, then it is guaranteed that after the second write at most one cell is programmed. Furthermore, if a non-zero symbol is stored on the first and second write exactly two out of the three cells are programmed. The guiding principles in the construction in [12] are to program on the first two writes the least number of cells possible in order to leave a large number of cells for the third write. According to Lemma 2, if after the first two writes, at most pn cells are programmed, then it is possible to write $(1-p)n$ more bits.

The construction we present is similar but we use two-cell blocks instead of three, which we simply call *blocks*. We follow similar guidelines to [12] to decrease the worst case maximum number of programmed cells after the first two writes. We use $n = 2m + 1$ cells, partitioned into m blocks and one extra cell. We denote the m blocks by $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m)$, $1 \leqslant i \leqslant m$, $\boldsymbol{c}_i = (c_{i,1}, c_{i,2})$, and the extra cell is c_{2m+1}.

We describe in details the encoding and decoding maps for each of the three writes.

First write: write a message $\boldsymbol{w}_1 \in \{0,1,2\}^m$ with probabilities $p_0 = 0.5, p_1 = p_2 = 0.25$. Namely, $\boldsymbol{w}_1$ has $m/2$ zeros etc. For $1 \leqslant i \leqslant m$, the i-th ternary symbol $w_{1,i}$ is stored in the i-th block $\boldsymbol{c}_i$ according to the map $\boldsymbol{c}_i = \psi(w_{1,i})$, where

$$\psi(0) = (0,0), \psi(1) = (0,1), \psi(2) = (1,0).$$

The decoding of the message $\boldsymbol{w}_1$ is straightforward from this definition. According to Remark 1 and since the number of cells can be arbitrarily large, we conclude that the number of bits stored on the first write approaches $m \cdot \mathcal{H}(0.5, 0.25, 0.25) = 1.5m$.

Second write: the second write is performed in two steps. On the first step we store m bits and on the second step $m/4$ bits. Let $\boldsymbol{w}_2 \in \{0,1\}^m$ be a balanced binary vector (i.e. it has the same number of zeros and ones) and S be the set of blocks that were not programmed on the first write, i.e., $S = \{i \in [1:m] \mid \boldsymbol{c}_i = (0,0)\}$. Let

$I_0 = \{i \in [1:m] \mid w_{2,i} = 0\}, I_1 = \{i \in [1:m] \mid w_{2,i} = 1\}$.

If $|S \cap I_0| \geqslant \frac{|S|}{2}$ we let $\boldsymbol{w}'_2 = \boldsymbol{w}_2$, and otherwise $\boldsymbol{w}'_2 = \overline{\boldsymbol{w}}_2$ and we program the $(2m+1)$-th cell to be 1. Note that this choice of the vector $\boldsymbol{w}'_2$ guarantees that at least half of the blocks in S store bit-value zero of $\boldsymbol{w}'_2$.

For $1 \leqslant i \leqslant m$, $w'_{2,i}$ is stored in $\boldsymbol{c}_i$ according to its parity, and as follows: $\boldsymbol{c}'_i = \phi(\boldsymbol{c}_i, w'_{2,i})$, where

1) $\phi((0,0),0) = (0,0)$ and $\phi((0,0),1) = (0,1)$ or $(1,0)$. It will be explained in the second step which option to choose in the case $\boldsymbol{w}'_{2,i} = 1$.
2) $\phi((0,1),0) = (1,1)$, $\phi((0,1),1) = (0,1)$.
3) $\phi((1,0),0) = (1,1)$, $\phi((1,0),1) = (1,0)$.

Now we explain the second step of the second write. Note that if after the first write, the state of the i-th block is $\boldsymbol{c}_i = (0,0)$ and $w'_{2,i} = 1$ then according to the definition of ϕ there are two options to program the i-th block. However, since we cannot guarantee that indeed this scenario happens, it is not clear how to take advantage and store more information this way (e.g., by treating $(0,1)$ as 0 and $(1,0)$ as 1). We now explain how to modify this idea and store extra $m/4$ bits.

By abuse of notation, we say that the i-th block, $1 \leqslant i \leqslant m$, stores the information $[a_i, b_i] \in \{0,1,2\} \times \{0,1\}$ if $a_i = w_{1,i}$ and $b_i = w'_{2,i}$. According to the choice of the vectors $\boldsymbol{w}_1, \boldsymbol{w}'_2$, at least 1/4 of the blocks store the information $[0,0]$ and hence we can say that for some $0 \leqslant x \leqslant 1/4$,

1) $(1/4 + x)m$ blocks store $[0,0]$.
2) $(1/4 - x)m$ blocks store $[0,1]$.
3) $(1/2 - (1/4 + x))m = (1/4 - x)m$ blocks store $[1,0]$ or $[2,0]$.
4) $(1/2 - (1/4 - x))m = (1/4 + x)m$ blocks store $[1,1]$ or $[2,1]$.

We shall use $m/4$ among the $m/2$ blocks in S and store one more bit in each one of them, while guaranteeing that the number of available cells for the third write will be at least m. This requires the following steps:

1) We mark $m/4$ out of the $m/2$ blocks in S: xm blocks which store $[0,0]$ and their set of indices is denoted by S_0, and $(1/4 - x)m$ blocks which store $[0,1]$ and their set of indices is denoted by S_1. The other $3m/4$ blocks which to not belong to $S_0 \cup S_1$ are programmed according to the map ϕ. Each of the $m/4$ blocks in $S_0 \cup S_1$ has two options to be programmed, while preserving its parity value. A $[0,0]$ block can be programmed as $(0,0)$ or $(1,1)$ and a $[0,1]$ block as $(0,1)$ or $(1,0)$.
2) Each of the $3m/4$ blocks that do not belong to $S_0 \cup S_1$ is mapped to a bit, representing a cell value, according to its most-significant bit (MSB)
$$(0,0), (0,1) \to 0, (1,1), (1,0) \to 1.$$
The $m/4$ blocks in $S_0 \cup S_1$ correspond to $m/4$ available cells in an m-cell binary two-write WOM code, denoted by $\mathcal{C}_{3/4}$. Thus, we will write $m/4$ more bits according to the second write of $\mathcal{C}_{3/4}$. (The remaining $3m/4$ bits of the binary two-write WOM code are the MSBs that we just defined.)
3) Finally, we program the $m/4$ blocks in $S_0 \cup S_1$. If a block stores $[0,0]$ then it is programmed to be $(0,0)$ if its bit by the WOM code $\mathcal{C}_{3/4}$ is 0 and we program it to be $(1,1)$ if its bit by $\mathcal{C}_{3/4}$ is 1. Similarly, a block storing $[0,1]$ is programmed to be $(0,1)$ if its bit by $\mathcal{C}_{3/4}$ is 0 and it is programmed to be $(1,0)$ if its bit by $\mathcal{C}_{3/4}$ is 1.

To summarize this discussion, we give the exact encoding map construction for the second write. We assume that the cell-state vector after the first write is $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m, c_{2m+1})$ and the input to the encoder is two vectors: a balanced vector $\boldsymbol{w}_2 \in \{0,1\}^m$ and a vector $\boldsymbol{s} \in \{0,1\}^{m/4}$. We use a binary two-write WOM code $\mathcal{C}_{3/4}$ of length m, given by the construction in [12] and stated in Theorem 1, with encoding map $\mathcal{E}_{3/4}$ and decoding map $\mathcal{D}_{3/4}$ for the second write. Yet another property we take from the two-write construction in [12] is the following. Assume k bits are to be stored on the second write and there are more than k cells which were not programmed on the first write. Then, it is possible to specify the encoder with a set S of k cells out of the non-programmed cells such that on the second write, only cells from S are allowed to be programmed. (in fact, this is the model for memories with defects). Hence, the encoding map $\mathcal{E}_{3/4}(\boldsymbol{c}, \boldsymbol{s}, S) = \boldsymbol{c}'$ receives as an input a binary cell-state vector $\boldsymbol{c}$ of weight at most $3m/4$, a message $\boldsymbol{s}$ of $m/4$ bits, and a set S of $m/4$ cells that can be programmed.

The steps to calculate the output $(\boldsymbol{c}'_1, \ldots, \boldsymbol{c}'_m, c'_{2m+1})$ of the encoding map are summarized as follows. The comments in parenthesis are just side properties on the values we calculate.

1) $S = \{i \in [1:m] \mid \boldsymbol{c}_i = (0,0)\}$, $(|S| = m/2)$.
2) $I_0 = \{i \in [1:m] \mid w_{2,i} = 0\}, I_1 = \{i \in [1:m] \mid w_{2,i} = 1\}$, $(|I_0| = |I_1| = m/2)$.
3) If $|S \cap I_0| \geqslant \frac{|S|}{2}$ then $\boldsymbol{w}'_2 = \boldsymbol{w}_2$ and $c'_{2m+1} = 0$; otherwise $\boldsymbol{w}'_2 = \overline{\boldsymbol{w}}_2$ and $c'_{2m+1} = 1$.
4) $S'_0 = \{i \in S \mid w'_{2,i} = 0\}, S_1 = \{i \in S \mid w'_{2,i} = 1\}$, $(|S'_0| + |S_1| = m/2, |S_1| \leqslant m/4 \leqslant |S'_0|)$.
5) Choose $S_0 \subseteq S'_0$ such that $|S_0| = m/4 - |S_1|$.
6) For all $i \in [1:m] \setminus (S_0 \cup S_1)$, $\boldsymbol{c}'_i = \phi(\boldsymbol{c}_i, w'_{2,i})$.
7) Let $\boldsymbol{c} = (c_1, \ldots, c_m)$ be $c_i = c_{i,1}$ for $i \in [1:m] \setminus (S_0 \cup S_1)$ and otherwise $c_i = 0$.
8) Let $\boldsymbol{c}' = \mathcal{E}_{3/4}(\boldsymbol{c}, \boldsymbol{s}, S_0 \cup S_1)$.
9) For $i \in S_0$, $\boldsymbol{c}'_i = (c'_i, c'_i)$ and for $i \in S_1$, $\boldsymbol{c}'_i = (c'_i, 1 - c'_i)$.

To complete the second write, we show how to decode the messages $\boldsymbol{w}_2$ and $\boldsymbol{s}$. Assume the memory state is $(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_m, c_{2m+1})$ and we decode the messages $\widehat{\boldsymbol{w}}_2$ and $\widehat{\boldsymbol{s}}$. The message $\widehat{\boldsymbol{s}}$ is decoded according to

$$\widehat{\boldsymbol{s}} = \mathcal{D}_{3/4}(c_{1,1}, c_{2,1}, \ldots, c_{m,1}),$$

and the message $\widehat{\boldsymbol{w}}_2$ is decoded according to

$\widehat{w}_2 = (c_{1,1} \oplus c_{1,2}, c_{2,1} \oplus c_{2,2}, \ldots, c_{m,1} \oplus c_{m,2}) \oplus c_{2m+1} \cdot \mathbf{1}$, where $\mathbf{1}$ denotes the all-ones vector.

Third write: We first calculate the minimum number of available cells for the third write. With the notation of the second write, note the following:

1) blocks storing $[0,0]$: there are $(1/4+x)m$ such blocks. In $m/4$ blocks, the two cells are not programmed and in the other xm blocks, the two cells may have been programmed.
2) blocks storing $[0,1]$: there are $(1/4-x)m$ such blocks and in all of them exactly one cell is programmed.
3) blocks storing $[1,0]$ or $[2,0]$: there are $(1/4-x)m$ such blocks and in all of them the two cells are programmed.
4) blocks storing $[1,1]$ or $[2,1]$: there are $(1/4+x)m$ such blocks and in all of them exactly one cell is programmed.

Therefore, the number of cells which are not programmed after the first two writes is at least

$$m/4 \cdot 2 + (1/4-x)m \cdot 1 + (1/4+x)m \cdot 1 = m,$$

and according to Lemma 2 it is possible to store m more bits.

To conclude, the sum-rate of the construction approaches

$$\frac{3m/2 + (m+m/4) + m}{2m+1} = 1.875 - \frac{1.875}{2m+1}.$$

As the value of m is sufficiently large, we conclude that the sum-rate approaches 1.875.

A small improvement is achieved in case the probability p_0 on the first write is not 0.5. Assume that $\boldsymbol{w}_1$ is distributed according to $p_0 = p, p_1 = p_2 = (1-p)/2$. Thus, the rate on the first write approaches $(h(p)+(1-p))/2$. The construction is similar with the following modification on the number of blocks of each type on the second write:

1) $p/2+x$ of the blocks store $[0,0]$.
2) $p/2-x$ of the blocks store $[0,1]$.
3) $1/2-(p/2+x) = (1-p)/2-x$ of the blocks store $[1,0]$ or $[2,0]$.
4) $1/2-(p/2-x) = (1-p)/2+x$ of the blocks store $[1,1]$ or $[2,1]$.

As before, m bits are stored on the first step of the second write. By marking xm blocks which store $[0,0]$ and $(p/2-x)m$ which store $[0,1]$, $xm+(p/2-x)m = pm/2$ more bits are stored using another two-write WOM code $\mathcal{C}_{1-p/2}$. Thus, the rate on the second write approaches $(1+p/2)/2$. The remaining number of cells for the third write is given by:

1) At least $pm/2$ blocks store $[0,0]$ and their two cells are not programmed. They contribute at least pm cells.
2) $(p/2-x)m$ blocks store $[0,1]$ and only one of their cells is programmed. They contribute $(p/2-x)m$ cells.
3) $((1-p)/2+x)m$ blocks store $[1,1]$ or $[2,1]$ and just one of their cells is programmed. They contribute $((1-p)/2+x)m$ cells.

Hence, there are $pm+(p/2-x)m+((1-p)/2+x)m=(1/2+p)m$ available cells for the third write. Thus, for large number of cell, the sum-rate approaches

$$\frac{(h(p)+1-p)+(1+p/2)+(p+1/2)}{2}.$$

This term is maximized for $p=\frac{\sqrt{2}}{1+\sqrt{2}}$ and its value is 1.885. We summarize this in the following Theorem.

Theorem 3. *For any $\epsilon > 0$, there exists a three-write WOM code of sum-rate at least $1.885-\epsilon$.*

IV. Improved construction for large alphabets

In this section, we give new constructions of two-write WOM codes over larger alphabets. What we show next is an approach for achieving sum-rate of roughly $\log(q^2/3) + O(\log(q)/q)$. The term $O(\log(q)/q)$ represents our improvement over existing constructions and note that the theoretical upper bound in this case is $\log_2 \binom{q+1}{2}$ [4]. This is done in several steps. First, we start with a simple construction. Then, we show improvements for this construction using an idea of [12]. Finally, we embed these ideas together with a construction in [6] in order to achieve the currently best known sum-rate. The effectiveness of this construction is demonstrated by new construction of codes for alphabets of size 8 and 16, which beat the current state of the art. Due to lack of space we will only be able to sketch our ideas for the case $q=8$ and report on the results for $q=16$.

Let us first start with a simple construction, reported also in [6]. We use it as a building step for our construction.

A simple construction: Assume q is even. On the first write we store an arbitrary word $\boldsymbol{w}_1 \in \{0,\ldots,q/2\}^n$ and on the second write a word $\boldsymbol{w}_2 \in \{0,q/2+1,\ldots,q-1\}^n$. For $1 \leqslant i \leqslant n$, if $w_{2,i} \neq 0$ then we write $w_{2,i}$ in the i-th cell, and otherwise $q/2$. Since we wrote $q/2$ only when $\boldsymbol{w}_2$ is zero, we will be able to recover $\boldsymbol{w}_2$ easily. Clearly, the sum-rate of this construction is $\log(q/2+1)+\log(q/2) = \log(q(q+2)/4)$.

First idea, renaming the symbols: Note that if $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ were chosen uniformly at random, from their respective domains, then we would have $\Theta(1/q)$ coordinates in which it holds that $\boldsymbol{w}_1$ is smaller than, say, $q/2-q/\log(q)$, and that $\boldsymbol{w}_2$ is zero. A much better sum-rate would be achieved if we could store more information on these memory cells. For example, we could hope to encode there roughly $O(n/q)$ symbols in an alphabet of size $O(q/\log(q))$. If we could do that, then the rate on the second write will be $\log(q/2)+O((1/q)\log(q))$, as we wanted.

There are several problems of course. The first is that $\boldsymbol{w}_1$ does not necessarily have many "small" coordinates. E.g., it could be the constant $q/2$ word. Similarly, it can be the case that $\boldsymbol{w}_2$ has no zeros. An even bigger concern is how to make sure that there are indeed $O(1/q)$ coordinates in which $\boldsymbol{w}_1$ is small and $\boldsymbol{w}_2$ is zero. The idea of [12] for overcoming these issues is to simply rename the symbols so that we will have many coordinates with the required properties. This is done by adding a few more coordinates to the code (when n is large this has a tiny effect on the sum-rate) in which we write the new representations of the symbols. To better understand this let us consider Example 1 below.

Example 1. Let us demonstrate this idea for a two-write WOM code for $q=8$. We let the length of the code be $N=n+3$.

First write: we get a word $\boldsymbol{w}_1 \in \{0,1,2,3,4\}^n$. Let $\alpha \leqslant \beta \in \{0,\ldots,4\}$ be the two most common values appearing in $\boldsymbol{w}_1$. Define a new word $\boldsymbol{w}'_1$ as follows. Whenever $w_{1,i}=\alpha$ we set $w'_{1,i}=0$ and when $w_{1,i}=0$ we write $w'_{1,i}=\alpha$. In the same way we replace the value 1 with β. The rest of the coordinates are unchanged. We now write $\boldsymbol{w}'_1$ to the first n memory cells as is. In order to recover $\boldsymbol{w}_1$ we write α in the $(n+1)$-th cell and β in the $(n+2)$-th cell.

Second write: Let $\boldsymbol{w}_2 \in \{0,5,6,7\}^n$ and $\boldsymbol{w}_3 \in \{1,2,3\}^{n/10}$ be the input words to be written. Let the set of "small" coordinates of $\boldsymbol{w}'_1$ be $I' = \{i \mid w'_{1,i} \leqslant 1\}$. Note that $|I'| \geqslant (2/5) \cdot n$. Let $\gamma \in \{0,5,6,7\}$ be the most common symbol appearing in $(w_2)_{I'}$. I.e., we only consider the coordinates I' and among them we check which value was the most popular in $\boldsymbol{w}_2$. (We break ties arbitrarily.) Let $I'' = \{i \mid i \in I' \text{ and } w_{2,i} = \gamma\}$. It is clear that $|I''| \geqslant (1/4) \cdot |I'| \geqslant n/10$. Let I be the first $n/10$ coordinates of I''. We now write γ on the N-th memory cell and define a new word $\boldsymbol{w}'_2$ as follows. Whenever $\boldsymbol{w}_2$ had zero we change it to γ and whenever it had γ we change it to zero. We now write the rest of the memory cells as follows. If $w'_{2,i} \neq 0$ then we write its value to the i-th memory cell. If $w'_{2,i} = 0$ and $i \notin I$ then we write the value 4 in the i-th cell. If $i \in I$ and it is the j-th element in I (according to the order $0 < 1 < 2 < \cdots < n/10$) then we write the value $w_{3,j}$ in the i-th memory cell (alternatively, we can think of $\boldsymbol{w}_3$ as $\boldsymbol{w}_3 \in \{1,2,3\}^I$ and write its i-th coordinate). Notice that for $i \in I$, $w_{3,i} \geqslant 1 \geqslant w'_{1,i}$ and so this is a "legal" write.

Note that to decode the word $\boldsymbol{w}_2$, one needs to read all cell levels, while treating levels 1, 2, 3, and 4 to be zero and converting the γ symbol. The word $\boldsymbol{w}_3$ is decoded according to the first $n/10$ cells of level less than 4.

The number of bits that we stored on the first write is $\log_2 5 \cdot n$ and the number of bits stored on the second write is $2n + (\log_2 3) \cdot n/10$. Thus, the sum-rate is $(\log_2 5 + 2 + \log_2 3/10) \cdot (n/N) = (\log_2 5 + 2 + \log_2 3/10) \cdot (1 - \frac{3}{n+3})$.

The construction in Example 1 can be extended for arbitrary q and will be called *Construction 1*. We state its sum-rate result in a slightly informal manner.

Theorem 4. *Construction 1 described above achieves sum-rate* $2\log(q/2) + O(\log(q)/q)$.

Improvements of the new idea: We can further improve Construction 1 as follows. We demonstrate it using Example 1 for $q = 8$. Let $N = n + 1$. Let $2/5 \leqslant p \leqslant 1$ be some parameter. On the first write, we consider words $\boldsymbol{w}_1 \in \{0,1,2,3,4\}^n$ that have at least pn coordinates that are either zero or one. We also let $\boldsymbol{w}_3$ be a word in $\boldsymbol{w}_3 \in \{1,2,3\}^{pn/4}$. We repeat the construction as before except for the following changes. First, we do not need to write α and β now. This enables us to take $N = n + 1$. Secondly, we have that $|I''| \geqslant pn/4$ so we can let I be the first $pn/4$ coordinates of I''. This explains the change in the domain of $\boldsymbol{w}_3$. The rest of the construction remains the same. By expressing the sum-rate as a function of p and maximizing, we get that for $p = 2/(2 + 3^{3/4}) \approx 0.4672$ the sum-rate is 4.493.

Improvement using the construction of [6]: By combining our techniques with construction B of [6] we can achieve better sum-rates. Roughly, we get sum-rate $\log(q^2/3) + O(\log(q)/q)$.

To conclude this section, we summarize the sum-rate results for two writes over $q = 8$ and $q = 16$, using all the techniques we discussed and mentioned, in Table II. Note that the construction we took from [6] uses a two-write WOM code. The first number in the fifth column corresponds to the case where one takes the best known two-write WOM code from [15] of sum-rate 1.4928. The second number corresponds to the case where the WOM code achieves the maximum sum-rate $\log_2 3$.

TABLE II
SUM-RATE RESULTS FOR $q = 8$ AND $q = 16$

q	Simple construction	Renaming symbols	First improvement	Combining with [6]	Upper bound
8	4.3219	4.4804	4.493	4.7368/4.829	5.1699
16	6.1699	6.3084	6.314	6.466/6.5585	7.0875

Lastly, we report on more efficient multiple writes constructions. For three writes and to the case where the number of levels is $q = 8$ we achieved sum-rate 5.44, where the upper bound on the sum-rate is 6.39. For $q = 8$ and four writes, we achieved sum-rate 6.26, while the upper bound is 6.97.

V. Conclusion and Future Research

In this paper, we studied binary three-write and non-binary WOM codes. The upper bound on the sum-rate of binary three-write WOM code is 2 and the recently best known construction in [12] had sum-rate 1.809. We used similar techniques from the construction in [12] in order to achieve sum-rate 1.885. For future research, we believe that this construction can be used in order to improve the best known sum-rates for more than three writes. We also studied the problem of non-binary WOM codes. We gave a construction of two-write code which we then showed how to combine with a construction in [6] in order to achieve high sum-rate results.

VI. Acknowledgement

Research of A.S. was supported in part by the Israel Science Foundation (grant number 339/10). Research of E.Y, was supported by the ISEF Foundation, and the Lester Deutsch Fellowship. Part of this work was done while A.S. was visiting the Centre Interfacultaire Bernoulli at EPFL.

References

[1] P. Cappelletti, C. Golla, P. Olivo, and E. Zanoni, *Flash Memories*, Boston: Kluwer Academic, 1999.

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

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

[4] F. Fu and A.J. Han Vinck, "On the capacity of generalized write-once memory with state transitions described by an arbitrary directed acyclic graph," *IEEE Trans. Inform. Theory*, vol. 45, no. 1, pp. 308–313, 1999.

[5] R. Gabrys and L. Dolecek, "Characterizing capacity achieving write once memory codes for multilevel flash memoriess," *Proc. IEEE Int. Symp. Inform. Theory*, pp. 2517–2521, St. Petersburg, Russia, July-August 2011.

[6] R. Gabrys et al., "Non-binary WOM-codes for multilevel flash memories," *Proc. IEEE Inform. Theory Workshop*, Paraty, Brazil, Oct. 2011.

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

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

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

[10] F. Merkx, "Womcodes constructed with projective geometries," *Traitement du signal*, vol. 1, no. 2-2, pp. 227–231, 1984.

[11] R.L. Rivest and A. Shamir, "How to reuse a write-once memory," *Inform. and Contr.*, vol. 55, no. 1–3, pp. 1–19, December 1982.

[12] A. Shpilka, "New constructions of WOM codes using the Wozencraft ensemble," to appear in *Latin American Symp. on Theoretical Informatics*, Arequipa, Peru, April 2012.

[13] Y. Wu, "Low complexity codes for writing write-once memory twice," *Proc. IEEE Int. Symp. Inform. Theory*, Austin, Texas, June 2010.

[14] Y. Wu and A. Jiang, "Position modulation code for rewriting write-once memories," acceptted by *IEEE Trans. Inform. Theory*, October 2010.

[15] E. Yaakobi, S. Kayser, P.H. Siegel, A. Vardy, and J.K. Wolf, "Efficient two-write WOM-codes," *Proc. IEEE Inform. Theory Workshop*, Dublin, Ireland, August 2010.