Buffer Coding for Asymmetric Multi-Level Memory

Vasken Bohossian
Electrical Engineering Department
California Institute of Technology
Pasadena, CA 91125
vincent@paradise.caltech.edu

Anxiao (Andrew) Jiang
Computer Science Department
Texas A & M University
College Station, TX 77843-3112
ajiang@cs.tamu.edu

Jehoshua Bruck
Electrical Engineering Department
California Institute of Technology
Pasadena, CA 91125
bruck@paradise.caltech.edu

***Abstract*— Certain storage media such as flash memories use write-asymmetric, multi-level storage elements. In such media, data is stored in a multi-level memory cell the contents of which can only be increased, or reset. The reset operation is expensive and should be delayed as much as possible. Mathematically, we consider the problem of writing a binary sequence into write-asymmetric q-ary cells, while recording the last r bits written. We want to maximize t, the number of possible writes, before a reset is needed. We introduce the term Buffer Code, to describe the solution to this problem. A buffer code is a code that remembers the r most recent values of a variable. We present the construction of a single-cell ($n = 1$) buffer code that can store a binary ($l = 2$) variable with $t = \lfloor \frac{q}{2^{r-1}} \rfloor + r - 2$ and a universal upper bound to the number of rewrites that a single-cell buffer code can have: $t \leq \lfloor \frac{q-1}{l^r-1} \rfloor \cdot r + \lfloor \log_l \{[(q-1) \ mod \ (l^r-1)]+1\} \rfloor$. We also show a binary buffer code with arbitrary n, q, r, namely, the code uses n q-ary cells to remember the r most recent values of one binary variable. The code can rewrite the variable $t = (q-1)(n-2r+1)+r-1$ times, which is asymptotically optimal in q and n. We then extend the code construction for the case $r = 2$, and obtain a code that can rewrite the variable $t = (q-1)(n-2)+1$ times. When $q = 2$, the code is strictly optimal.**

I. Introduction

We study asymmetric q-ary storage cells the content of which can only be increased or erased (set to 0). The erase operation is expensive and should be delayed as much as possible. This model arises, in practice, in the context of flash memories and similar storage devices that use an isolated charge in order to record data [1]. Different processes (e.g. tunneling vs. hot electron injection) are used to increase or decrease the charge, giving rise to the asymmetry. Without a scheme such as the one presented in this paper, the process of updating stored data requires large blocks of memory to be reset and rewritten, even if only a small fraction of that data needs to be updated. We consider the case of a set of q-ary cell used to record a sequence of data bits, while storing the last r bits. Our goal is to maximize t, the number of writes possible, before the cells needs to be erased. Similar schemes have been considered in the context of WOM codes, introduced by Rivest and Shamir [8] and extensively studied [2] [3] [4] [5] [6] [7] [8] [9]. WOM codes mainly address the case $q = 2$: binary write-asymmetric cells. In this paper we consider arbitrary q. We also introduce the notion of storing multiple consecutive values of a variable.

Definition 1 (q-ary storage cell):
A q-ary cell contains an integer value between 0 and $q-1$.

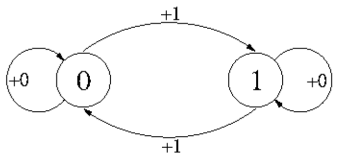

Fig. 1. State machine for Example 1. It describes a $(1, q, 1, q-1)$ storage scheme used to write a binary sequence into a q-ary cell, while storing only the last bit written. The maximum number of writes is $q-1$ corresponding to the worst case scenario of an alternating sequence of 1s and 0s.

When writing a new value into a q-ary cell one must increase its content, or if that is impossible, set it to 0.

In practice, q-ary cells can be found in flash memories and similar storage media [1].

Definition 2 ((n, q, r, t) Buffer Code):
A scheme that allows a sequence of bits to be written into n q-ary cells. At any point of the writing sequence, the last r bits written can be recovered. The code supports at most t writes, before the cells need to be reset.

Recording the last r values of a sequence is useful in practice for the implementation of certain data structures such as stacks, also in the context of memory pages for which the state of the RAM is saved to disk at different points in time. The above definition can be generalized to the case where l-ary variables – instead of bits, where $l = 2$ – are written and recovered.

In the case of a single cell (i.e. $n = 1$) the buffer code can be represented by a table, or in mathematical terms, by a surjective mapping f_r from the set of integers $\{0, .., q-1\}$ to the set of binary vectors of size r, $\{0, 1\}^r$.

Example 1 (a simple $(1, q, 1, q-1)$ buffer code):
We want to store a single bit of data into a q-ary cell, with the ability to write as many times as possible. In the description above, this corresponds to the case: $r = 1$, a single stored bit. The solution is to encode the single bit as the parity of the content of the cell. For example, let $q = 6$:

cell value	0	1	2	3	4	5
stored bit	0	1	0	1	0	1

Notice that if the bit to be stored is the same as the one already stored, the content of the cell does not need to be changed. The number of possible writes is $q-1$, corresponding to the worst case scenario of alternating 1 and 0 bits. Figure 1 shows the state machine used to write the content of the cell. Every transition in the input data corresponds to a $+1$ increment in the cell value.

In Section II we generalize the above example to $r > 1$. Section III shows and upper bound to t for single-cell buffer codes. Sections IV and V show two asymptotically optimal multi-cell buffer codes, one being strictly optimal when $q = 2$.

II. A SINGLE-CELL CONSTRUCTION

In this section we present a $(1, q, r, t)$ buffer code and show that it achieves:

$$t = \left\lfloor \frac{q}{2^{r-1}} \right\rfloor + r - 2$$

In other words, the code allows $\lfloor \frac{q}{2^{r-1}} \rfloor + r - 2$ bits to be written into a q-ary cell before it needs to be reset (see Definition 1). After every write, the last r bits written can be recovered.

Code Construction 1:
The $(1, q, r, t)$ buffer code is defined by a surjective mapping, f_r, from N to $\{0,1\}^r$. Defined by induction:

$$f_1(x) = x \bmod 2$$

$$f_{r+1}(x) = \begin{cases} (0, f_r(x)) & \text{, if } x \bmod 2^{r+1} < 2^r \\ (1, \overline{f_r(x)}) & \text{, otherwise} \end{cases}$$

Here follow two examples using the above code for cells of sizes 6 and 12. For each example we show a graphical representation of the writing (encoding) process, i.e. a state diagram that defines by how much one needs to increase the value of the cell, as a function of the current stored bits, and the new bit being written. We also show a table, used to read (decode) the content of the cell, i.e. the mapping f_r mentioned in Construction 1, above. Compare the tables of examples 1 through 3 to get an idea of the recursive definition of Construction 1.

Example 2 ($(1, 6, 2, 3)$ buffer code):
We want to store 2 bits of data into a 6-ary cell, with the ability to write 3 times, i.e. after every write, the last 2 bits are recorded, and can be recovered. Figure 2 shows how to increase the value of the cell, as a function of the bit being written. The starting state is 00. The following table shows how to recover the last 2 bits, at any point of the writing process.

cell value	0	1	2	3	4	5
stored bits	00	01	11	10	00	01

Notice that the above table can be generated by the state machine of Figure 2 by following the arrows labeled $+1$. The number of possible writes is 3, corresponding to the worst case scenario of alternating 0 and 1 bits.

Example 3 ($(1, 12, 3, 4)$ buffer code):
In this example we use a 12-ary cell to store the last 3 bits. Consequently we get a guaranteed minimum of 4 writes. Figure 3 shows the corresponding state machine. The function f_3 is shown in the table below.

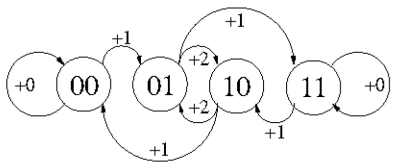

Fig. 2. State machine for Example 2. This diagram shows by how much one needs to increase the content of the cell, when writing a new bit. That amount is a function of the current stored bits (the start state), and the new bit to be written (the end state).

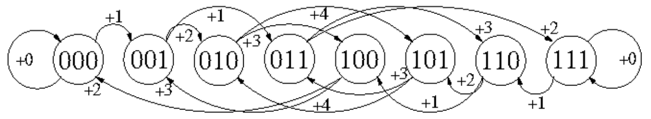

Fig. 3. State machine for Example 3. It implements a general $(1, q, 3, \lfloor \frac{q}{4} \rfloor + 1)$ buffer code. In Example 3, $q = 12$ resulting in $t = 4$.

value	0	1	2	3	4	5
bits	000	001	011	010	111	110
value	6	7	8	9	10	11
bits	100	101	000	001	011	010

Next, we show the main result of this section in the form of a theorem. Without loss of generality, it assumes $q \geq 2^r$.

Theorem 1:
The $(1, q, r, t)$ buffer code, defined in Construction 1 is such that:

$$t = \left\lfloor \frac{q}{2^{r-1}} \right\rfloor + r - 2$$

Proof: We need to show that any binary sequence s of length at most t will bring the value of the cell to at most $q - 1$. We first show that the worst case sequence, s_t, is an alternation of 1s and 0s, namely the sequence:

$$s_t = (1, 0, 1, 0, 1, 0 ...)$$

of length t. In the state machine of Figure 2, s_t corresponds to alternating states (01) and (10). Each state transition increases the value of the cell by 2, which is the maximum increase for this state machine ($r = 2$). Similarly, in the $r = 3$ state machine, shown in Figure 3, s_t corresponds to alternating states (010) and (101), increasing the value of the cell by 4, for each bit written. In the general case, as defined in Construction 1, any two consecutive entries of the table f_r, for which $f_r(x_0) = (0, 1, ..., 0, 1)$ and $f_r(x_1) = (1, 0, ..., 1, 0)$ are such that $|x_1 - x_0| = 2^{r-1}$, which happens to be the largest possible increment of the f_r state machine, therefore the sequence s_t is the worst case sequence. s_t corresponds to $r - 1$ initial writes to get from state $(0, 0, ..., 0, 0)$ to state $(0, 1, ..., 0, 1)$, followed by an additional write for each 2^{r-1} rows of the table f_r (the size of which depends on q). The $r - 1$ initial writes increase the cell's value respectively by $2^0, 2^1, \cdots, 2^{r-2}$, each of which is the maximum possible. Therefore $t = \lfloor \frac{q}{2^{r-1}} \rfloor + r - 2$ ■

III. Upper Bound for single-cell Buffer Codes

In this section, we present an upper bound to t for buffer codes with $n = 1$ and arbitrary l, q, r.

We first define some terms that will be used throughout the rest of this paper. For $1 \leq i \leq n$, we use a_i to denote the i-th cell. We use $(c_1, c_2, \cdots, c_n)$ – called the *cell state vector* – to denote the states of the n cells, where c_i $(0 \leq c_i \leq q-1)$ is the state of the i-th cell. For a variable, we use $(v_1, v_2, \cdots, v_r)$ – called the *variable vector* – to denote the variable's most recent r values $(0 \leq v_i \leq l-1$ for all $i)$, with v_r being the most recent value and v_1 being the oldest among the most recent r values.

By default, initially, all the cells are in the state 0, and $v_1, v_2, \cdots, v_r$ are all 0. For writing, we only need to consider those writes that do change the variable vector. (For example, if $r = 2$ and the current variable vector is $(v_1 = 1, v_2 = 1)$, then the writing operation that updates the variable to be '1' does not really change the variable vector. Consequently, we do not need to consider such a writing operation.) We define *the cell state vectors of the i-th generation* $(0 \leq i \leq t)$ to be the set of cell state vectors reachable after exactly i writing operations from the beginning.

Theorem 2: When $n = 1$, $t \leq \lfloor \frac{q-1}{l^r-1} \rfloor \cdot r + \lfloor \log_l \{[(q-1)$ mod $(l^r-1)] + 1\} \rfloor$.

Proof: Suppose that a buffer code guaranteeing t writes is given. Starting with a valid cell state vector, by performing r or fewer writes, the variable vector $(v_1, v_2, \cdots, v_r)$ can reach any of the l^r possible values. Those l^r variable values correspond to l^r different cell state vectors (possibly including the starting cell state vector). Therefore, there is a sequence of r consecutive writes that causes the cell's state to increase by at least $l^r - 1$.

We choose the first set of r writes, the second set of r writes, ..., the b-th set of r writes such that every such a set of r writes increases the cell's state by at least $l^r - 1$. Let b be as large as possible. After those br writes, select a set of y writes after which no more write can be performed. Let y be as small as possible. Clearly, $y < r$.

Since the maximum cell level is $q-1$, $b \leq \lfloor \frac{q-1}{l^r-1} \rfloor$. Note that $\lfloor \log_l \{[(q-1)$ mod $(l^r-1)] + 1\} \rfloor < r$. If $b < \lfloor \frac{q-1}{l^r-1} \rfloor$, then $t \leq br + y < \lfloor \frac{q-1}{l^r-1} \rfloor \cdot r \leq \lfloor \frac{q-1}{l^r-1} \rfloor \cdot r + \lfloor \log_l \{[(q-1)$ mod $(l^r-1)] + 1\} \rfloor$. Now consider the case that $b = \lfloor \frac{q-1}{l^r-1} \rfloor$. In that case, the last y writes increase the cell's level by at most $(q-1)$ mod (l^r-1). As y or fewer writes lead the variable value to l^y possible values, with the same analysis as before, we get $l^y - 1 \leq (q-1)$ mod (l^r-1). So $y \leq \lfloor \log_l \{[(q-1)$ mod $(l^r-1)]+1\} \rfloor$. So again, $t \leq br + y \leq \lfloor \frac{q-1}{l^r-1} \rfloor \cdot r + \lfloor \log_l \{[(q-1)$ mod $(l^r-1)] + 1\} \rfloor$. So the theorem holds. ■

IV. Asymptotically Optimal Buffer Code for $l = 2$ and General n, q, r

In this section, we present a buffer code for $l = 2$ and general n, q, r where $n \geq 2r$. That is, the code uses n q-ary cells to store the most recent r values of one binary variable (a bit). In lots of electronic memories (e.g., flash memories), the 16 bits of a word are stored separately in 16 parallel blocks, using the same address. So the writing operation for a word becomes a write for a single bit in each block. For this reason, it is of particular interest to study the storage of binary variables.

The code we present in this section achieves $t = (q-1)(n-2r+1)+r-1$. Note that a buffer code can write a variable no more than $(q-1)n-1$ times. Therefore, the code presented here achieves a t value asymptotically optimal in both q and n.

A. Construction of The Code

For the buffer code, we first present its construction for the special case $q = 2$. We then naturally extend the code construction for arbitrary q.

Code Construction 2: Buffer code for $l = 2$ and general n, q, r, $n \geq 2r$

- *Mapping cell state vectors to variable vectors:* By *valid cell state vector*, we mean a cell state vector that can be reached by some writing operations. Every valid cell state vector $(c_1, c_2, \cdots, c_n)$ of this code satisfies the following property: For $i = 1, 2, \cdots, n-r$, for any cell state vector of the i-th generation, there are exactly i cells in the state 1 and $n-i$ cells in the state 0; what's more, all those i cells in the state 1 belong to the set $\{a_1, a_2, \cdots, a_{i+r}\}$ (namely, the first $i+r$ cells).
  Clearly, a valid cell state vector $(c_1, c_2, \cdots, c_n)$ is in the $(\sum_{i=1}^{n} c_i)$-th generation.
  A valid cell state vector $(c_1, c_2, \cdots, c_n)$ in the i-th generation is mapped to the variable vector $(v_1, v_2, \cdots, v_r)$ as follows: For $j = 1, 2, \cdots, r$, $v_j = c_{i+j}$.
- *Writing:* The code enables $n-r$ writing operations. Let's say that the current cell state vector $(c_1, c_2, \cdots, c_n)$ is $(x_1, x_2, \cdots, x_n)$ and it is in the i-th generation. $(0 \leq i < n-r.)$ Say that the next writing operation is to change the variable's value to y. (By default, only the writing operations that change the variable vector are considered. It means that if the current variable state is $(0, 0, \cdots, 0)$ or $(1, 1, \cdots, 1)$, then y cannot be 0 or 1, respectively.) Then, if $y = 0$, find an integer $j \leq i+1$ such that $x_j = 0$, and change c_j – the state of the j-th cell – to be 1; if $y = 1$, then change c_{i+r+1} from 0 to 1.

The following is an example of the code.

Example 4: Let $l = 2, n = 9, q = 2$, and $r = 3$. If the $n - r = 6$ writing operations change the variable vector as $(0,0,0) \rightarrow (0,0,1) \rightarrow (0,1,1) \rightarrow (1,1,0) \rightarrow (1,0,0) \rightarrow (0,0,1) \rightarrow (0,1,0)$, then the cell state vector changes as $(0,0,0,0,0,0,0,0,0) \rightarrow (0,0,0,1,0,0,0,0,0) \rightarrow (0,0,0,1,1,0,0,0,0) \rightarrow (0,0,1,1,1,0,0,0,0) \rightarrow (0,1,1,1,1,0,0,0,0) \rightarrow (0,1,1,1,1,0,0,1,0) \rightarrow (0,1,1,1,1,1,0,1,0)$. We can see that given a cell state vector, recovering the variable vector is very simple: just read the $(w+1)$-th, $(w+2)$-th, $\cdots$, $(w+r)$-th entries in the cell state vector, where w is the number of 1's in the vector.

We now extend the above code from $q = 2$ to arbitrary q. The code uses the cells "layer by layer." Specifically, when $q > 2$, for the first $n - r$ writes, we use the cells as if $q = 2$. That is, the cells use only the two states 0 and 1. Then, let's say that the $(n-r+1)$-th writing operation changes the variable vector to $(v_1 = z_1, v_2 = z_2, \cdots, v_r = z_r)$. The $(n-r+1)$-th writing operation is carried out as follows: first, every cell raises its state to 1, and we map this cell state vector – $(1, 1, \cdots, 1)$ – to the variable vector $(0, 0, \cdots, 0)$; from then on, treat the cell state 1 (respectively, cell state 2) as the old cell state 0 (respectively, cell state 1), including the way cell state vectors are mapped to variable vectors and the way writing operations are performed; perform r successive writing operations, where the i-th writing operation $(1 \le i \le r)$ changes the variable to z_i. At this moment, the cell state vector corresponds to the variable vector $(v_1 = z_1, v_2 = z_2, \cdots, v_r = z_r)$. Then the cells use the two levels – level 1 and level 2 – to perform more writes. Totally $(n - 2r + 1)$ writes can be performed by using the two states 1 and 2, after which the cells use the states 2 and 3 for writing in the same way, and so on.

For example, assume that $l = 2, n = 9, q = 4, r = 3$. If the current cell state vector is $(0, 1, 1, 1, 1, 1, 0, 1, 0)$ (which is in the $(n - r)$-th generation and corresponds to the variable vector $(0, 1, 0)$) and the next three writing operations change the variable to 1, 0 and 1 successively, then the cell state vector changes as $(0, 1, 1, 1, 1, 1, 0, 1, 0) \to (1, 1, 2, 2, 1, 2, 1, 1, 1) \to (1, 2, 2, 2, 1, 2, 1, 1, 1) \to (1, 2, 2, 2, 1, 2, 1, 2, 1)$.

B. Analysis of The Code

Theorem 3: The buffer code presented in Code Construction 2 is correct.

Proof: First, assume $q = 2$. To prove the correctness of the code construction, we use induction to prove the following *assertion*: For $i = 1, 2, \cdots, n - r$, the i-th writing operation leads the cells to a valid cell state vector that correctly corresponds to the new variable vector.

Consider the case $i = 1$. The first writing operation has only one possibility: to change the variable to 1. By Code Construction 2, the cell state $(c_1, c_2, \cdots, c_n)$ becomes as follows: $c_{r+1} = 1$, and $c_j = 0$ for all $j \neq r+1$. That cell state is valid and corresponds to the variable vector $(0, 0, \cdots, 0, 1)$. So the *assertion* holds when $i = 1$. That serves as the base case.

Assume that the *assertion* holds for all $i < p$, where $p \le n - r$. Now consider the case $i = p$. Say that the p-th writing operation changes the variable to y, where $y = 0$ or 1. By the induction assumption, after the $(p - 1)$-th write, $p - 1$ cells are in the state 1, and they all belong to the first $p - 1 + r$ cells (namely, cells $a_1, a_2, \cdots, a_{p-1+r}$); therefore, among the first p cells, at least one of them is in state 0. If $y = 0$, the p-th write changes such a cell in state 0 to state 1, so the number of cells in state 1 becomes p; if $y = 1$, the $(p + r)$-th cell is changed from 0 to 1, so the number of cells in state 1 also becomes p. Clearly, after the p-th write, all those cells in state 1 are among the first $p + r$ cells. Therefore, the cell state vector after the p-th write is *valid*. Say that after the $(p - 1)$-th write, the cell state vector is $(c_1, c_2, \cdots, c_n)$. Its corresponding variable vector is simply $(v_1 = c_p, v_2 = c_{p+1}, \cdots, v_r = c_{p+r-1})$. After the p-th write, the state of the $(p+r)$-th cell becomes y, so the corresponding variable vector is $(v_1 = c_{p+1}, v_2 = c_{p+2}, \cdots, v_{r-1} = c_{p+r-1}, v_r = y)$, which is the *correct variable vector*. So the assertion holds when $i = p$. This completes the induction. Therefore, the theorem holds when $q = 2$.

When $q > 2$, the code uses the cell levels in a simple "layer by layer" way, which is clearly also correct. ■

The number of writes t guaranteed by the buffer code can be directly derived from Code Construction 2. Thus we have the following conclusion.

Theorem 4: For the buffer code presented in Code Construction 2, $t = (q - 1)(n - 2r + 1) + r - 1$.

V. Enhanced Buffer Code for $l = 2, r = 2$ and General n, q

The code presented in Code Construction 2 has a t that is asymptotically optimal in n, q. When $r = 2$, it gives $t = (q - 1)(n - 3) + 1$. In this section, we present a better code with $t = (q - 1)(n - 1)$. In particular, when $q = 2$, this code is strictly optimal.

We first present the new code construction for the case $q = 2$, and analyze its properties. The construction is then extended for general q using the "layer-by-layer" approach.

A. Optimal Buffer Code for $q = 2$

The new buffer code enhances Code Construction 2. When $q = 2$, it has $t = n - 1$. So the code allows $n - 1$ writing operations.

The new code uses the same method as Code Construction 2 to map cell state vectors of the 1st, 2nd, $\cdots$, $(n - 2)$-th generations to variable vectors. It adds the following specification to Code Construction 2 to handle the first $n - 2$ writing operations:

- *Writing:* Let's say that the current cell state vector $(c_1, c_2, \cdots, c_n)$ is $(x_1, x_2, \cdots, x_n)$ and it is in the i-th generation, where $0 \le i < n - 2$. (The corresponding variable vector is $(v_1 = x_{i+1}, v_2 = x_{i+2})$.) Say that the next writing operation is to change the variable's value to y. The write is performed as follows:
  1) If $y = 0$ and $(x_{i+1}, x_{i+2}) = (0, 1)$, then change c_{i+1} – the state of the $(i + 1)$-th cell – to 1.
  2) If $y = 0$ and $(x_{i+1}, x_{i+2}) = (1, 0)$, then find the integer $j \le i$ such that $x_j = 0$, and change c_j to 1.
  3) If $y = 0$ and $(x_{i+1}, x_{i+2}) = (1, 1)$, then find the integer $j \le i$ such that "$x_j = 0$ and $(i + 3) - j$ is an even integer", and change c_j to 1.
  4) If $y = 1$, change c_{i+3} from 0 to 1.

The mapping from the cell state vectors in the $(n - 1)$-th generation to the variable vectors is as follows:

- *Mapping from cell state vectors to variable vectors:* Every valid cell state vector in the $(n - 1)$-th generation satisfies this property: Among the n cells, $n - 1$ of them are in state 1 and one of them is in state 0.

Given a valid cell state vector in the $(n-1)$-th generation, let's say that a_i – the i-th cell – is the unique cell in state 0. The cell state vector is mapped to the variable vector (v_1, v_2) in the following way:

1) If $i \leq n-2$ and $n-i$ is even, then $(v_1, v_2) = (1, 0)$.
2) If $i \leq n-2$ and $n-i$ is odd, then $(v_1, v_2) = (0, 0)$.
3) If $i = n-1$, then $(v_1, v_2) = (1, 1)$.
4) If $i = n$, then $(v_1, v_2) = (0, 1)$.

The $(n-1)$-th writing operation is performed in the following way:

- *The $(n-1)$-th write:* Let's say that after the $(n-2)$-th write, the cell state vector $(c_1, c_2, \cdots, c_n)$ is $(x_1, x_2, \cdots, x_n)$. (The corresponding variable vector is $(v_1 = x_{n-1}, v_2 = x_n)$.) Say that the $(n-1)$-th write is to change the variable's value to y. It is performed as follows:
    1) If "$y = 0$ and $(x_{n-1}, x_n) = (0, 1)$" or "$y = 1$ and $(x_{n-1}, x_n) = (0, 0)$," then change c_{n-1} from 0 to 1.
    2) If $y = 0$ and $(x_{n-1}, x_n) = (1, 0)$, then change c_n from 0 to 1.
    3) If $y = 0$ and $(x_{n-1}, x_n) = (1, 1)$, then let $j \leq n-2$ be the integer such that "$x_j = 0$ and $n-j$ is odd", and change c_j from 0 to 1.
    4) If $y = 1$ and $(x_{n-1}, x_n) = (0, 1)$ or $(1, 0)$, then let $j \leq n-2$ be the integer such that $x_j = 0$, and change c_j from 0 to 1.

Example 5: Let $l = 2, n = 6, q = 2, r = 2$. If the $n-1 = 5$ writing operations change the variable vector as $(0, 0) \to (0, 1) \to (1, 0) \to (0, 1) \to (1, 1) \to (1, 0)$, then the cell state vector changes as $(0, 0, 0, 0, 0, 0) \to (0, 0, 1, 0, 0, 0) \to (0, 1, 1, 0, 0, 0) \to (0, 1, 1, 0, 1, 0) \to (0, 1, 1, 0, 1, 1) \to (1, 1, 1, 0, 1, 1)$.

B. Analysis of The Code

The new code has a special structural property, as the following lemma shows.

Lemma 1: For the new code constructed in this section, for $i = 0, 1, \cdots, n-2$, let $(c_1, c_2, \cdots, c_n)$ be a valid cell state vector in the i-th generation. By the code construction, among the first $i+2$ cells – $a_1, a_2, \cdots, a_{i+2}$ – exactly two of them are in the state 0. Let a_p and a_q be those two cells. Then, between p and q, one is odd and the other is even.

Proof: The proof is by induction on i. When $i = 0$, $c_1 = c_2 = 0$, so p=1 and $q = 2$. So the lemma holds when $i = 0$. This serves as the base case.

Assume that when $i < z \leq n-2$, the lemma holds. Now consider the case $i = z$. The proof for this induction step is a straightforward check using the rule on writing in the code construction. For example, consider the following case: after $z-1$ writes, the states of a_z and a_{z+1} are 0 and 1, respectively, and the z-th write changes the variable to 0. In this case, the code construction changes the state of a_z to 1. By the induction assumption, after $z-1$ writes, there is a cell a_j $(j \leq z+1)$ whose state is 0 such that between j and z, one is odd and one is even. After the z-th write, both a_j and a_{z+2} are in the state 0, so we can let $p = j$ and $q = z+2$; then between p and q, one is odd and the other is even; so the lemma holds. All the other cases can be checked similarly; for simplicity, we skip the details. That completes the induction. So the lemma holds for all $0 \leq i \leq n-2$. ■

Theorem 5: The new code constructed in this section is correct. And it has $t = n-1$.

Proof: It is easy to verify that the new code deals with the first $n-2$ writes and the 0-th, 1st, $\cdots$, $(n-2)$-th generations of cell state vectors in the same way as Code Construction 2 does, except that the $n-2$ writes are performed in a more specific way. For succinctness, we omit the details of this simple verification. Now consider the $(n-1)$-th write. Based on Lemma 1, any cell state vector in the $(n-2)$-th generation has exactly two cells a_p, a_q whose states are 0, while between p and q one is odd and the other is even. By using this observation, and by the way the code construction performs the $(n-1)$-th write and maps the $(n-1)$-th generation of cell state vectors to variable vectors, we can easily use a case by case verification to see that the $(n-1)$-th write always leads the cells to a valid cell state vector that corresponds to the correct variable vector. So the code is correct. It directly follows from the code construction that $t = n-1$. ■

The above code construction and analysis are for $q = 2$. When $q \geq 2$, we can use the cells "level by level" in the same way as the code in Section IV does. For such a code, t becomes $(q-1)(n-2)+1$.

Acknowledgements

This work was supported in part by the Lee Center for Advanced Networking at the California Institute of Technology.

References

[1] P. Capelletti, C. Golla, P. Olivio and E. Zanoni, (Ed.), *Flash Memories*, Kluwer Academic Publishers, 1st edition, 1999.

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

[3] 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 Transactions on Information Theory*, vol. 45, no. 1, pp 308-313, 1999.

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

[5] C. Heegard, "On the capacity of permanent memory," *IEEE Transactions on Information Theory*, vol. IT-31, pp 34-42, January 1985.

[6] A. V. Kuznetsov and A. J. H. Vinck, "On the the general defective channel with informed encoder and capacities of some constrained memories," *IEEE Transactions on Information Theory*, vol. 40, no. 6, pp 1866-1871, November 1994.

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

[8] R.L. Rivest and A. Shamir, "How to reuse a 'write-once' memory," *Information and Control*, 55:1-19, 1982.

[9] J. K. Wolf, A. D. Wyner, J. Ziv and J. Korner, "Coding for a write-once memory," *AT&T Bell Labs. Tech. J.*, vol. 63, no. 6, pp. 1089-1112, 1984.