New rewriting schemes for MLC flash memory

Luoming Zhang
Laboratoire d'algorithmique (ALGO)
EPFL, CH-1015 Lausanne, Switzerland
Email: luoming.zhang@epfl.ch

***Abstract*—MLC flash memory is a non-volatile memory, in which the cells can be injected with different numbers of electrons. The unique characteristics of asymmetric writing/erasing operation and limit write/erase cycles of flash memory make the erasing operation expensive in terms of the lifetime of flash memory. We present a new rewriting scheme to increase the rewrites between two consecutive erasing operations by making better usage of the cells' charge level. The scheme can get better device lifetime without expensive modifications to the common flash memory architecture.**

I. Introduction

Flash memory is a non-volatile memory that can be both electrically programmed and electrically erased. The unique characteristics of flash memory [1], such as low-latency IOPS and low power consumption, lead to the increasing application in both consumer and enterprise systems. However, the limit endurance of flash memory, which is typically only 10^4 for Multiple Level Cells (MLC) [2], is one of the main obstacles on the way for its adoption to all the storage architectures. Moreover, the lifetime of flash memory is greatly reduced because of the inherent working mechanism. According to Micron's datasheet, the newest product RealSSD C400 with a volume size of 128GB actually only has device lifetime (i.e. maximum rewrite information) of 72TB [3]. This is almost a reduction by a factor of 15 given that the write/erase cycles is 10^4. Therefore, it is a hot topic on how to extend the lifetime of MLC flash memory.

Many works have been done on improving the lifetime of flash memory based Solid State Device (SSD). One of the main directions is on the system level using hybrid storage architecture with HDD and SDD. Soundararajan *et al.* [4] used HDD as write cache for SDD which could save up to 50% writes to SDD so as to indrectly prolong the lifetime of SSD. Another main direction is trying to balance the writes to SSD on block levels, called wear leveling [5]. This technique operates in the layer called Flash Translation Layer (FTL) in flash memory, which functions as an indirect mapping between the Logical Block Address (LBA) and Physical Block Address (PBA). Based on the logged information in FTL, the write to the device is pointed to the less frequent written blocks, so as to level all the writes on all the blocks in flash memory. The lifetime of flash memory could be effectively extended, by jointly considering the workloads and garbage collecting policies [6]–[9]. However, these schemes still have rather high redundancy. As Hu *et al.* [10] analyzed, the optimized schemes may still suffer write amplification as high as three in some cases.

Besides the research on the system and block level, some researchers work on the data representation in flash memory. Inspired by Rivest and Shamir [11], they modeled some memories as Write-Once Memory (WOM), which means the basic storage unit could only transit to "1" state from "0" state but not vice versa. The corresponding WOM codes use a group of these basic units to represent variable values (i.e. data) so that the variables are rewritable, though limit rewrites. Researchers are working on maximizing the number of variables and rewrites using constant number of basic storage units [12]–[17]. Recently, Jiang *et al.* [18] developed floating codes, which not only generalize the WOM codes, but also greatly increase the rewrites. We are going to address the problem by a so-called water-filling scheme at the page level.

In the following of this paper, some foundations of flash memory will be given in section II, besides a simple example of the new scheme. And in section III, we will formalize the scheme. Three cases will be discussed. At last, performance will be evaluated by comparing our scheme with others in section IV.

II. Fundamentals and a simple construction

In this section, we first introduce some fundamental operations in flash memory. Later, the motivation of our new scheme is given. Before concluding this section, we will give a simple example to show the concept of our new scheme on how to address the problems that initially motivated us.

A. Fundamental operations in Flash memory

In flash memory, the basic storage unit is the floating-gate transistor [1], [2], which is also called cell. The cell stores information by trapping charge (i.e. electrons) in the floating gate, i.e. writing. The charge level, which is measured by discrete threshold (i.e. reading), can only increase monotomicly before the whole charge get released (i.e. erasing). The flash cells are arranged in matrix-like structures, which are called blocks. Each bocks is composed of several pages, which are the elementary writing/reading units. Each page, usually storing 2kBytes data, is extended with additional spare bytes, called metadata. These metadata keeps tracks of the page's running status. In order to keep the low latency property and increase the cell density, the writing and erasing operation are asymmetric, which means that writing is based on pages while erasing is based on blocks. Considering the limit endurance, it brings great challenges using MLC flash memory to design

SSD. For instance, in the primitive operation of updating the data stored in one cell, it is necessary to erase the whole block and then rewrite all the data back on the block except the target cell. This operation will not only induce huge inefficiency in writing/erasing operations, but also greatly shorten the lifetime of other cells in the same block. There are various ways that have been done to increase the operation efficiency in order to fully leverage the capacity of flash memory. In this paper, we are going to exploit the capacity of the MLC cells to represent more data so as to extend the lifetime of flash memory.

B. Motivation

In common flash memory, each MLC cell is erased before the next rewrite despite of cell's states. This is not efficient in term of using the charge level to represent data, because the cell may get erased even if the cell is charged to the lowest level. To clarify this point, take a specific 4-MLC device as a example. Each cell can store 2-bits information in every write/erase cycle using 4 charge levels. Meanwhile, there is possibility to double the stored information if there is a data scheme which could make use of the four charge levels such that each level can store 1 bit data. Various data representation schemes have different effects on the device lifetime.

Jiang *et al.* [18] designed floating codes by carefully changing the cells' charge levels when rewriting the variable values. Floating codes can achieve optimal rewrites between adjacent erasing operation so as to get better device lifetime. However, the mapping between the cell's states and the variable value is too complicate to implement.

Is there any simpler data representation scheme that can fully utilize the charge levels of the cell? To address the problem, we set up three goals for the new data representation scheme: fully utilizing the charge levels to represent data; little changes to the existing architecture of flash memory; customizable scheme in the upper layer.

C. A simple example

In order to achieve our goals, we propose a new data representation scheme. Here is the simple example of the scheme where a binary variable is represented by the charge levels of one cell. All the writing operation will be at the page level in our scheme. To simplify the description, we focus on cells in the following.

In the binary variable case, it's enough to use the difference of one charge level to represent the variable. Let's define the relatively low charge level to denote "0" while the high one to "1". And we define the rewrite generation to be how many writes since the latest erasing operation. In the scheme, we need the generation information, which is stored as the metadata in page, to assist the reading and writing operation. Meanwhile, the charge level will be be gradually pushed to the top from zero. When there is a write request, the generation information will be read out and converted to the highest charge level in the previous generation, let's say A_H. If the updating data is "0", the target charge level of the cell is equal to A_H; othervise, the target one is $A_H + 1$. When there is a reading request on the variable, we could get A_H in the same way as writing operation, which is used as the threshold to detect the cell's charge level. The measurement result is then mapped to the corresponding variable value. Once A_H touched the top, the generation information and cells will be erased before the next rewrite request. Fig. 1 shows the example where the write generation is 2, and A_H is equal to 2.

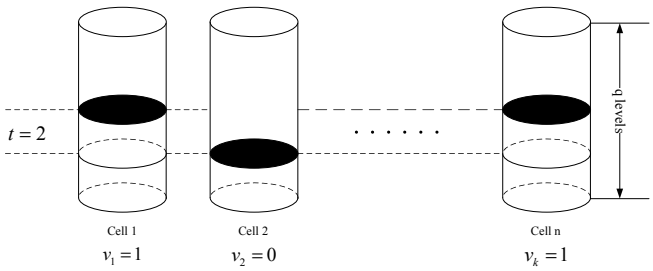

Fig. 1. A simple example of water-filling scheme. The rewrite generation is 2, the variable alphabet size is 2 , the cell's level is q, and using n cells.

From the example, it's easy to find that the new scheme is more efficient than the common scheme in terms of using charge level to represent data. Take the 4-MLC case in subsection II-B, the information stored in one cell is 3 bits, which brings 50% gain comparing to the common scheme. Since the cell's highest charge level could be arbitrary integer in the new scheme, i.e. no requirement to be power of 2, there is potential to have more gain, referring to III-D. Moreover, the changes to the existing architecture are trivial. And since the new scheme mainly works at the page layer, which is comparatively low, it's more flexible to customize the system at the upper layer. All these positive points show that the new scheme satisfies the goals mentioned in subsection II-B. The updating rule of cells' states in the new scheme is like filling the water into the barrels and the water's average level increase after each write so we name it water-filling scheme.

III. Formalization of the water-filling scheme

In this section, we will formalize the water-filling scheme. At first, we will introduce some notations and definitions to facilitate our description. Then we will describe how water-filling scheme works when the variable values are only stored in one cell. And later, we will extend the scheme to the more general case that the variable values are jointly stored in several cells. At last, we will discuss some issues on practical implementation about water-filling scheme.

A. Notations and Definitions

Notations and definitions in water-filling scheme will be given. Whenever possible, we will follow the same definitions in Jiang *et al.*'s work [19].

Notation 1:

Let's assume we use a group of cells jointly represent a variable vector in the water-filling scheme. Given non-negative integer n, q, k, l, t and T, we define $W(n, q, k, l)$ to be the explicit water-filling scheme where n cells of q levels is used

to represent k variables of alphabet size l; While t denotes the write generation and T denotes the maximum rewrites, so $t \in \{1, 2, \cdots, T\}$.

Definition 1:
Water-filling scheme seems move a frameless window towards the top with all the cells' charge levels fall inside (see Fig. 1). To clearly describe this, we define the cell's status after t rewrites as below:

- **Absolute charge level:** cell's charge level, *abbr.* level;
- **Basis charge level:** the lowest level, *abbr.* basis level;
- **Relative charge level:** the absolute difference between the cell's charge level and basis level, *abbr.* relative level;
- **Windows height of charge level:** the maximum relative charge level, *abbr.* windows height.

After t writes, we use $(c_i)_t$, $(C_b)_t$, $(d_i)_t$ and $(\Delta)_t$ to denote the level, basis level, relative level and windows height of the ith cell respectively. Then we could have

$$\begin{aligned}(d_i)_t &= (c_i)_t - \min\left((c_j)_t : j \in \{1, 2, \cdots, n\}\right) \\ &= (c_i)_t - (C_b)_t \\ (\Delta)_t &= \max\left((d_i)_t : i \in \{1, 2, \cdots, n\}\right)\end{aligned}$$

Given $W(n, q, k, l)$, if we use v_i to denote the ith variable value, we will have the following vectors:

- $(\vec{c})_t = (c_1, c_2, \cdots, c_n)_t \in \{0, 1, \cdots, q-1\}^n$ denotes cells' absolute charge levels;
- $(\vec{d})_t = (d_1, d_2, \cdots, d_n)_t \in \{0, 1, \cdots, \Delta\}^n$ denotes cells' the relative charge levels;
- $(\vec{v})_t = (v_1, v_2, \cdots, v_k)_t \in \{0, 1, \cdots, l-1\}^k$ denotes the variables values.

Definition 2:
In water-filling scheme, the generation information is needed for the next rewrite. We use f_u and f_d to denote the writing function and the reading function respectively, which are defined as following:

$$f_u : (\vec{c})_{t-1} \times (\vec{v})_t \to (\vec{c})_t \tag{1}$$

$$f_d : (\vec{c})_t \to (\vec{v})_t \tag{2}$$

B. Single cell scheme

In the single cell scheme, water-filling scheme uses charge levels in one cell to represent variable values. As common scheme, charge levels are bijectively mapped to the variable set in each rewrite. The charge level is pushed towards the top charge level of the cell in the next rewrite. The same variable value in two consecutive rewrites will cause the charge level increase constantly, which equals windows height defined in subsection III-A.

Fig. 2 shows a mapping diagram between the charge levels and the variable values, where $n = 1$, $k = 1$ and $l = 4$. In the diagram, the number in the circle stands for the charge level, while the number beside the circle stands for the corresponding variable value. The cell states in the same rewrite generation is horizontally arranged in the diagram. We can find that the

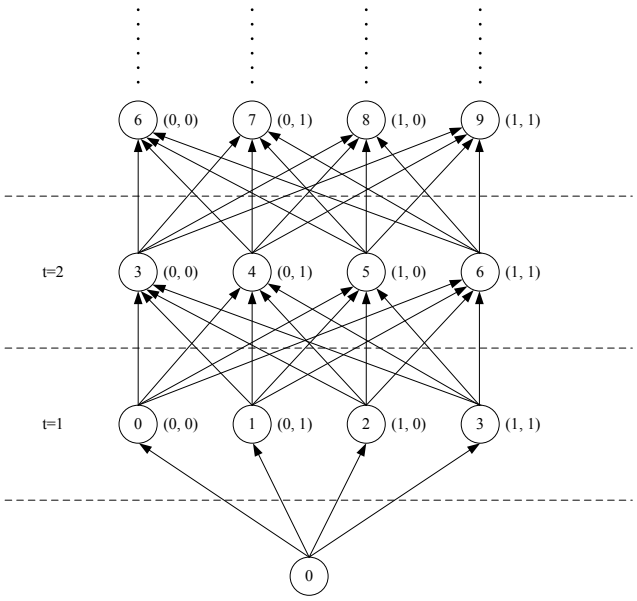

Fig. 2. The transition diagram for Water-filling scheme ($n = 1$, $k = 1$ and $l = 4$). The number inside the circle is the charge level; aside the circle is the corresponding binary value of the variable.

two adjacent rewrites share one charge level. This is one of the unique characteristics in water-filling scheme. By logging the generation information, the highest charge level in the previous rewrite could be used as the lowest charge level in the next rewrite without conflicts.

By analogy, the principles of the singel cell scheme $W(1, q, k, l)$ can be deduced as below:

1) *The generation information are recorded and updated after every rewrite and reset to zero once the page gets erased;*
2) *Consecutive two rewrites share charge level in the cell, so the windows height* Δ *is* $l^k - 1$*;*
3) *In each rewrite, the variables are mapped to the same relative charge level;*
4) *The basis charge level is* $\Delta \times (t-1)$.

The relation between the charge level and the variable value is equally converted to the one between the relative level and the variable value once the generation information is known. Given the mapping function and the demapping function between the relative level and the variable value to be $m(\cdot)$ and $m^{-1}(\cdot)$ respectively, the explicit writing and reading function are as below:

$$(c_i)_t = f_u\left((v_i)_t, t\right) = \Delta \times (t-1) + m\left((v_i)_t\right) \tag{3}$$

$$(v_i)_t = f_d\left((d_i)_t\right) = m^{-1}\left((d_i)_t\right) \tag{4}$$

Here is the corresponding description for the writing and reading operation:

Writing operation: (i) The rewrite generation is read out to calculate the basis charge level for the following write; (ii) Check whether the page should be erase or not by comparing the maximum charge

level of the cell with the sum of the new basis level and the windows height. If the maximum charge level is smaller, then the page is erased. Otherwise, the new variable values are mapped to the corresponding relative value and the rewrite generation is updated; (iii) The final step is programming the cell to the target charge level, which is the sum of the new basis charge level and the relative charge level;

Reading operation: (i) The rewrite generation is read out to calculate the basis level; (ii) The detecting thresholds is determined by the basis level and the windows height; (iii) The relative charge level of the cell is detected, which is demapped to the corresponding variable value.

In the single cell scheme, the maximum rewrites T is:

$$T = \left\lfloor \frac{q-1}{\Delta} \right\rfloor = \left\lfloor \frac{q-1}{l^k - 1} \right\rfloor \tag{5}$$

We found T is exponentially decreasing when the number of variables increases. To solve this problem, it's necessary to jointly use multiple cells to represent data.

C. Multiple cells scheme

In the multiple cells scheme, variable values are jointly stored in a group of cells. The variable values are bijectively mapped to the permutation set of charge levels in a group of cells. Each write, the mapping between the relative levels and the variable values are the same. This is similar to the single cell case, even though the mapping function between the relative charge levels and variable values becomes more complex. Considering the intercell interference [20] and the structure of flash memory, the windows heights in all cells are equally chosen to be Δ. Let's use $M(\cdot)$ denote the bijective mapping function from the vector of relative charge levels to variable values and $M^{-1}(\cdot)$ denote the demapping function. Given $W(n, q, k, l)$, the writing and reading function in the multiple cells case is as below:

$$\begin{aligned}(c_1, c_2, \cdots, c_n)_t &= f_u\left((v_1, v_2 \cdots, v_k)_t, t\right) \\ &= \Delta \times (t-1) + M\left((v_1, v_2 \cdots, v_k)_t\right)\end{aligned} \tag{6}$$

$$\begin{aligned}(v_1, v_2, \cdots, v_k)_t &= f_d\left((d_1, d_2, \cdots, d_n)_t\right) \\ &= M^{-1}\left((d_1, d_2, \cdots, d_n)_t\right)\end{aligned} \tag{7}$$

Each cell shares one charge level in the two adjacent writes in order to represent data, so the number of shared levels is n for $W(n, q, k, l)$. Given the windows height Δ, the size of permutation set of charge levels is $(\Delta+1)^n$ in every write, which should be no less than the possibilities of variable values to be represented, i.e. $(\Delta+1)^n \geq l^k$. Under such writing and reading mechanism, the maximum rewrites T is as below:

$$T = \left\lfloor \frac{q-1}{\lceil l^{k/n} \rceil - 1} \right\rfloor \tag{8}$$

From equation (8), it's not easy to find that the maximum rewrites increase exponentially with the number of cells used in the data representation scheme. By logging the rewrite generation and using multiple cells, it becomes more efficient on the usage of charge levels for data representation.

D. Hybrid cells scheme

The practical implementation issues are addressed by hybrid cells scheme. In the common flash memory, cell's charge level q should be equal to $2^p\,(p \in \mathbb{N})$ in order to store p bits information. From subsection III-B and III-C, it's easy to know that arbitrary charge level is allowed as long as the size of permutation set of charge level is no smaller than the possibility of variables. However, the size of permutation set may not be the exact multiples to the possibilities of variables. For example, in the case of $n = 2, q = 6, k = 3$ and $l = 2$, the top charge level is wasted if only the multiple cells scheme is used. In such a case, the flexibily of water-filling allow us to "recycle" the remaining charge level by using the hybrid cells solution. In the hybrid cells case, two or more explicit schemes are used for different writes between two adjacent erasing operations. Back to the above example, we could use $W(2, 6, 3, 2)$ for the first two writes and $W(1, 6, 1, 2)$ for third write. The average information bits stored in one cell is 4 bits, i.e. 33% gain. By using the hybrid cells scheme, it's possible to adequately use all the charge levels in cells to extend the device lifetime.

IV. Performance evaluation

In flash memory based devices, the device lifetime and the input/output latency are two main concerns. We are going to evaluate the water-filling scheme from three aspests: Device lifetime, Writing efficiency and Reading complexity. Meanwhile, the tradeoff between the device lifetime and device's volume size is also analyzed. At last, the potential application of the scheme is discussed.

- **Device lifetime**
  As defined in [3], Device lifetime means how long the device can reliably store information under fixed workload. It could equally translate into how many information bis that can reliably store on the device. The lifetime is effected by the data representation scheme and several other system factors, such as the wear leveling policies. We could formalize the device lifetime L_{device} as below:

$$L_{device} = \alpha \times N_{cell} \times \bar{E}_{cell} \times L_{cell} \tag{9}$$

  Where α stands for the system factor except data representation; N_{cell} stands for the device's cell number; $\bar{E}_{cell}$ stands for the average endurance of cells and L_{cell} stands for the cell lifetime, which means how many information bits that one cell could store during a write/erase cycle. Simply, L_{cell} determines the device lifetime if we just concern about the efficiency of data representaiton scheme. Referting to section III, the cell lifetime in $W(n, q, k, 2)$ is determined as below:

$$L_{cell} = T \times \frac{k}{n} = \left\lfloor \frac{q-1}{\lceil l^{k/n} \rceil - 1} \right\rfloor \times \frac{k}{n} \tag{10}$$

Given $W(1,q,2,2)$, the cell lifetime is $\lfloor \frac{q-1}{3} \rfloor$ bits. Using floating codes, we could get $\lfloor \frac{q-1}{2} \rfloor$ rewrites. Since every rewrite only change one variable, the corresponding cell lifetime is $\lfloor \frac{q-1}{4} \rfloor$ bits. So $W(1,q,2,2)$ is even more optimal than floating codes.

Analogously, we could find the cell lifetime for $W(2,q,3,2)$ is $\left\lfloor \frac{3(q-1)}{2} \right\rfloor$ bits; while the one for floating codes is also $\left\lfloor \frac{3(q-1)}{2} \right\rfloor$ bits. So $W(2,q,3,2)$ is also optimal scheme.

By enumeration, we could find the cell lifetime for water-filling scheme and floating codes is the same or even better if hybrid cells scheme is used in practice. So the water-filling scheme is optimal in terms of device lifetime.

- **Writing efficiency**
  Flash memory is programmed by means of the Incremental Step Pulse Programming (ISPP) [20]. The higher target charge level of the cell is, the more voltage steps it needs to reach the target which leads to longer programming time. So in the page range, the programming time is determined by the cell with highest target level in the common flash memory. In water-filling scheme, the programming time is determined by the windows height, which is usually lower than the highest charge level that the cells could reach. So the programming of the page could be faster (more efficient) than the common flash memory by using water-filling scheme.
- **Reading complexity**
  In water-filling scheme, the number of detection threshold is less since reading only needs cell's relative level given the rewrite generation. Moreover, compared to common flash memory or floating codes, where the absolute level is needed, the bit width that needed to represent the charge level becomes shorter in the water-filling scheme. This reduces on the routing complexity. We could have faster reading speed by using water-filling scheme.
- **Tradeoff analysis**
  As seen from the water-filling scheme, the absolute information store in one cell per write is reduced even though the cell lifetime increases. The scheme offers a way to make tradeoff between the device's volume size and the device lifetime. Meanwhile, the scheme implicitly offers another way to lever the writes at the page level, which leads potential applications such as RAID-like SSD.

V. Conclusion

In the paper, we proposed a new rewriting scheme which could better use the cells' charge level to represent data. By jointly increasing the charge levels of a group of cells, flash memory could get approximately optimal rewrites, which implicitly extend the device lifetime. The average charge levels of the cells could reach as high as possible before the cells get erased. So the new scheme is pretty efficient in using the charge levels to represent data. Meanwhile, by logging the rewrite generation, the writing and reading operation could become faster than common flash memory schemes.

Acknowledgment

This work was supported by Grant 228021-ECCSciEng of the European Research Council.

References

[1] R. Bez, E. Camerlenghi, A. Modelli, and A. Visconti, "Introduction to flash memory," *Proceedings of the IEEE*, vol. 91, no. 4, pp. 489–502, 2003.

[2] M. Sanvido, F. R. Chu, A. Kulkarni, and R. Selinger, "Nand flash memory and its role in storage architectures," *Proceedings of the IEEE*, vol. 96, no. 11, pp. 1864–1874, 2008.

[3] Micron, "C400 1.8-inch sata nand flash ssd." [Online]. Available: http://www.micron.com/get-document/?documentId=6419

[4] G. Soundararajan, V. Prabhakaran, M. Balakrishnan, and T. Wobber, "Extending ssd lifetimes with disk-based write caches," in *Proceedings of the 8th USENIX conference on File and storage technologies*, ser. FAST'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 8–8. [Online]. Available: http://portal.acm.org/citation.cfm?id=1855511.1855519

[5] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Comput. Surv.*, vol. 37, no. 2, pp. 138–163, 2005, 1089735.

[6] L. Chang, T. Kuo, and S. Lo, "Real-time garbage collection for flash-memory storage systems of real-time embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 3, no. 4, pp. 837–863, 2004.

[7] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for ssd performance," in *USENIX 2008 Annual Technical Conference on Annual Technical Conference*. Berkeley, CA, USA: USENIX Association, 2008, pp. 57–70. [Online]. Available: http://portal.acm.org/citation.cfm?id=1404014.1404019

[8] F. Chen, T. Luo, and X. Zhang, "Caftl: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives," in *Proceedings of the 9th USENIX conference on File and storage technologies*, 2011.

[9] Y. Pan, G. Dong, and T. Zhang, "Exploiting memory device wear-out dynamics to improve nand flash memory system performance," in *Proceedings of the 9th USENIX conference on File and storage technologies*, 2011.

[10] X.-Y. Hu, E. Eleftheriou, R. Haas, I. Iliadis, and R. Pletka, "Write amplification analysis in flash-based solid state drives," in *Proceedings of SYSTOR 2009: The Israeli Experimental Systems Conference*, ser. SYSTOR '09. New York, NY, USA: ACM, 2009, pp. 10:1–10:9. [Online]. Available: http://doi.acm.org/10.1145/1534530.1534544

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

[12] J. Wolf, A. Wyner, J. Ziv, and J. Korner, "Coding for a write-once memory," *AT&T Bell Laboratories technical journal*, vol. 63, no. 6, pp. 1089–1112, 1984.

[13] A. Fiat and A. Shamir, "Generalized 'write-once' memories," *Information Theory, IEEE Transactions on*, vol. 30, no. 3, pp. 470–480, 1984.

[14] F. Merkx, "Womcodes constructed with projective geometries," *TS Traitement du signal*, vol. 1, pp. 227–231, 1984.

[15] G. Cohen, P. Godlewski, and F. Merkx, "Linear binary code for write-once memories (corresp.)," *Information Theory, IEEE Transactions on*, vol. 32, no. 5, pp. 697–700, 1986.

[16] S. Kayser, E. Yaakobi, P. Siegel, A. Vardy, and J. Wolf, "Multiple-write wom-codes," *Proc. 48th Annual Allerton Conferenece on Communications, Control and Computing*, 2010.

[17] E. Yaakobi, S. Kayser, P. H. Siegel, A. Vardy, and J. K. Wolf, "Efficient two-write wom-codes," in *Information Theory Workshop (ITW), IEEE*, 2010, pp. 1–5.

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

[19] A. Jiang and J. Bruck, "Data representation for flash memories," in *Data Storage*. In-Tech Publisher, 2010.

[20] R. Micheloni, L. Crippa, and A. Marelli, *Inside NAND flash memories*. Springer Verlag, 2010.