CODING TECHNIQUES FOR ERROR CORRECTION AND REWRITING IN FLASH MEMORIES

A Thesis

by

SHOEB AHMED MOHAMMED

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2010

Major Subject: Electrical Engineering

CODING TECHNIQUES FOR ERROR CORRECTION AND REWRITING IN FLASH MEMORIES

A Thesis

by

SHOEB AHMED MOHAMMED

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Co-Chairs of Committee,	Anxiao Jiang
	Scott L. Miller
Committee Members,	Tie Liu
	Sebastian Hoyos
Head of Department,	Costas N. Georghiades

August 2010

Major Subject: Electrical Engineering

ABSTRACT

Coding Techniques for Error Correction

and Rewriting in Flash Memories. (August 2010)

Shoeb Ahmed Mohammed, B.Tech., Indian Institute of Technology Roorkee

Co–Chairs of Advisory Committee: Dr. Anxiao Jiang
Dr. Scott L. Miller

Flash memories have become the main type of non-volatile memories. They are widely used in mobile, embedded and mass-storage devices. Flash memories store data in floating-gate cells, where the amount of charge stored in cells – called cell levels – is used to represent data. To reduce the level of any cell, a whole cell block (about 10^6 cells) must be erased together and then reprogrammed. This operation, called block erasure, is very costly and brings significant challenges to cell programming and rewriting of data. To address these challenges, rank modulation and rewriting codes have been proposed for reliably storing and modifying data. However, for these new schemes, many problems still remain open.

In this work, we study error-correcting rank-modulation codes and rewriting codes for flash memories. For the rank modulation scheme, we study a family of one-error-correcting codes, and present efficient encoding and decoding algorithms. For rewriting, we study a family of linear write-once memory (WOM) codes, and present an effective algorithm for rewriting using the codes. We analyze the performance of our solutions for both schemes.

To My Parents and Brothers

ACKNOWLEDGMENTS

Firstly, I thank God Almighty for granting an opportunity to study at a prestigious university.

I would like to thank Dr. Anxiao Jiang for his valuable guidance through the course of this work, constant encouragement and discussions on various topics that helped me learn a lot. I also thank him for his patience and help in writing this thesis.

I will not forget the time spent with Hao Li and Yue Wang, members of Dr. Jiang's research group. I am grateful to all professors at Texas A&M University who have taught me during the course of graduate studies. I am also thankful to all my friends at Texas A&M University for making stay at College Station pleasant.

Finally, I deeply appreciate the constant support of my parents and relatives who were always there with words of encouragement and help.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Non-volatile memories (NVMs) are developing fast as a major storage technology. Among current NVMs, flash memories are by far the most widely used. Flash memories use floating-gates cells as their basic storage units, where the charge (e.g., electrons) stored in the cells represents data [4]. The charge can be injected into the cells using the hot-electron injection mechanism or the Fowler-Nordheim tunnelling mechanism, and be removed from the cells using the Fowler-Nordheim tunnelling mechanism. However, the charge injection and removal processes have different costs. This is because in a flash memory, the cells are organized as blocks, where every block consists of about 10^6 cells. Although it is relatively simple to inject charge into individual cells, to remove charge from any cell, the whole block of cells must be erased first (which means to remove all charge from all the cells in the block) before reprogrammed [4]. This *block erasure* property makes it very expensive to remove charge, and is very costly for the longevity, speed and efficiency of flash memories. (Note that a cell block can endure only about $10^4 \sim 10^5$ erasures.) For this reason, it is very beneficial to minimize and balance the number of block erasures. A well known technique, called *wear leveling*, uses the idea of shifting data among blocks to balance the erasures performed on different blocks, and is widely used in flash file systems [9].

The block erasure property brings serious challenges to cell programming and data rewriting in flash memories. The charge injection is a noisy process, where the injected charge usually deviates from the target value [4]. So to program a cell (which means to inject charge into the cell to reach a target value), it is common

This thesis follows the style of *IEEE Transactions on Information Theory.*

practice to use multiple rounds of charge injection. In each round, a small amount of charge is injected, and then the charge level in the cell is measure to see how close it is compared to the target level. This way, the charge level in the cell can cautiously approach the target level and avoid overshooting (which would cause the very expensive block erasure operation) [1]. This iterative programming procedure is time consuming, especially as the flash memories move toward multi-level cells (MLCs) with more levels and smaller cell sizes, both of which are important for increasing the storage capacity [4]. So a new data representation scheme, which fits the asymmetric property of flash memories and allows efficient cell programming, will be very beneficial. The block erasure property also makes it very challenging for rewriting (i.e., modifying) data, because even to change one bit, a whole block of cells may have to be erased, which is very costly. Therefore, it is desirable to find a coding scheme that can allow data to be rewritten without block erasures.

To meet these challenges, the *rank modulation* scheme [15, 17] and rewriting codes [3, 12] have been proposed for flash memories. In *rank modulation*, the relative order of the cells' charge levels – instead of their absolute values – is used to represent data [15, 17]. When programming cells, the cells of lower charge levels are programmed before cells of higher charge levels. Since only the order of the charge levels matters for representing data, when a cell is programmed, the only objective is to make its charge level be higher than some previous charge level. This eliminates the risk of charge overshooting, and allows more efficient programming of cells. After cells are programmed, the charge levels can be disturbed by various mechanisms, including charge leakage, read and write disturbs, etc. Many of these noise mechanisms change the charge levels in one direction [4]. Compared to the absolute values of charge levels, their relative order is more robust to asymmetric errors [15]. There has been a number of works studying rank modulation, including rewriting codes and Gray codes for rank modulation [15, 16, 23], error-correcting codes based on the Kendall tau distance [2, 17, 18], error-correcting codes based on the L_∞ distance [24], low-density parity-check (LDPC) codes for rank modulation [29], variations of rank modulation sequences [25, 26], etc.

Rewriting codes are also a new approach of representing data in flash memories [3, 12]. Instead of the conventional one-to-one mapping between the data and the cells' charge levels, a rewriting code builds a one-to-many mapping from the data to the cells' charge levels. This way, we can rewrite data by only increasing cell levels, not decreasing them, and therefore avoid the costly block erasure operation (until the cells' charge levels reach their highest values). Given the rate of the rewriting code, the objective is to maximize the number of rewrites that can be performed before the block erasure becomes necessary. Rewriting codes called write-once memory (WOM) codes were proposed by Rivest and Shamir in their seminal work [22], and have been studied in a number of subsequent papers [5, 6, 8, 10, 19, 21, 27]. In recent years, floating codes and buffer codes (which are generalizations of WOM codes) have been proposed for flash memories [3, 12]. Various code constructions have been presented, and the storage capacity of rewriting codes has been studied [7, 11, 13, 14, 20, 28].

In this work, we study error-correcting rank-modulation codes and linear rewriting codes. Error-correcting codes (ECCs) are important for the reliable storage of data. And for the rank modulation scheme, a family of one-error-correcting codes has been proposed in [17, 18], whose size is provably at least half of the optimal size. The codes are based on the Kendall tau distance. However, no efficient encoding and decoding algorithms have been shown for the code. We present an efficient encoding algorithm that uses the concatenation of codewords, and show its error-correction performance via simulations.

We also study an important family of rewriting codes called *linear WOM codes.*

It is a generalization of the linear WOM code proposed by Rivest and Shamir in their work [22]. Although the code is known to have asymptotically optimal rewriting performance when the number of cells is large [22], it has not been shown how to efficiently rewrite data based on the linear code in general. We present a pseudo-polynomial-time algorithm for rewriting data using the linear WOM codes. The algorithm locally minimizes the number of cells programmed for each rewrite.

CHAPTER II

ERROR-CORRECTING RANK-MODULATION CODES

In this chapter, we study error-correcting rank-modulation codes. We first introduce the basic concepts of rank modulation codes, and focus on an asymptotically optimal error-correcting code that corrects one error. We then study its properties and present an efficient encoding algorithm based on codeword concatenation. We evaluate its error-correction performance through simulations.

A. Rank Modulation Codes

The rank modulation scheme was proposed in [15, 17]. Consider n flash memory cells. For $i = 1, 2, \ldots, n$, the charge level of the i-th cell – which we shall call *cell level* – is denoted by $c_i \in \mathbb{R}$. The *ranks* of the n cells are a permutation of $\{1, 2, \ldots, n\}$. If the permutation is $[a_1, a_2, \cdots, a_n]$, then

$$c_{a_1} > c_{a_2} > \cdots > c_{a_n}.$$

(It is assumed that no two cells have exactly the same level. Since here the cell levels are real numbers, this is essentially always true in practice.) We say that the a_1-th cell has the highest rank, and the a_n-th cell has the lowest rank.

A *rank modulation scheme* uses the ranks induced by the n cell levels to represent data. Let $\mathcal{S}_n$ denote the set of permutations of $\{1, 2, \ldots, n\}$, and let q denote the size of an alphabet $Q = \{0, 1, \ldots, q-1\}$. When the n cells store the data of alphabet size q, we have a decoding function

$$D \, : \, \mathcal{S}_n \to Q$$

that maps every permutation (which are the ranks induced by the n cell levels

$c_1, \ldots, c_n$) to some value in the alphabet Q.

Definition 1.. *Given a permutation, an* adjacent transposition *is the local exchange of two numbers in the permutation. That is, an adjacent transposition changes a permutation*

$$[a_1, \cdots, a_{i-1}, a_i, a_{i+1}, a_{i+2}, \cdots, a_n] \in \mathcal{S}_n$$

to a permutation

$$[a_1, \cdots, a_{i-1}, a_{i+1}, a_i, a_{i+2}, \cdots, a_n]$$

for some $i \in \{1, 2, \ldots, n-1\}$.

Given two permutations $A, B \in \mathcal{S}_n$, the Kendall tau distance *between A and B, $d(A, B)$, is the minimum number of adjacent transpositions needed to change A into B (and vice versa).*

Example 2.. *Let $n = 5$, $A = [2, 1, 3, 4, 5]$ and $B = [3, 2, 1, 5, 4]$, then $d(A, B) = d(B, A) = 3$, because we can change A into B with the minimum number of adjacent transpositions as $[2, 1, 3, 4, 5] \to [2, 3, 1, 4, 5] \to [3, 2, 1, 4, 5] \to [3, 2, 1, 5, 4]$.*

When we use the Kendall tau distance to measure errors, we consider an error as an adjacent transposition (caused by the error). An error-correcting rank-modulation code that can correct t errors is a code $\mathcal{C} \subseteq \mathcal{S}_n$ such that for any two codewords (i.e., permutations) $A, B \in \mathcal{C}$, we have $d(A, B) \geq 2t + 1$. That is, the minimum Kendall tau distance of the code is $2t + 1$.

Example 3.. *The following two codes were presented in [17]. $\mathcal{C} = \{[1, 2, 3], [3, 2, 1]\}$ is a one-error-correcting rank-modulation code of length $n = 3$ and size $|\mathcal{C}| = 2$. And $\mathcal{C} = \{[1, 2, 4, 3], [3, 1, 4, 2], [3, 2, 4, 1], [4, 1, 3, 2], [4, 2, 3, 1]\}$ is a one-error-correcting rank-modulation code of length $n = 4$ and size $|\mathcal{C}| = 5$.*

In [17, 18], a one-error-correcting rank-modulation code of asymptotically optimal size has been presented. The code construction is based on mapping permutations to nodes in a $(n-1)$-dimensional linear array.

Definition 4.. *Given a permutation $A = [a_1, a_2, \ldots, a_n] \in \mathcal{S}_n$, the* coordinates *of A,*

$$X_A = (x_1, x_2, \ldots, x_{n-1}),$$

are defined as follows: for $i = 1, \ldots, n-1$, let $z \in \{1, 2, \ldots, n\}$ denote the integer such that $a_z = i+1$, then

$$x_i = \left|\{j \mid z < j \leq n, a_j \leq i\}\right|.$$

Clearly, for $i = 1, \ldots, n-1$, we have $x_i \in \{0, 1, \ldots, i\}$.

Clearly, this is a one-one correspondence between the set of permutations and their coordinates. To determine i^{th} coordinate, we count symbols less than $(i+1)$ that are to its right in the given permutation.

Example 5.. *Let $n = 6$, and let $A = [1, 2, 3, 4, 5, 6]$, $B = [6, 5, 4, 3, 2, 1]$, $C = [2, 4, 6, 1, 5, 3]$. Then the coordinates of A, B, C are $X_A = (0, 0, 0, 0, 0)$, $X_B = (1, 2, 3, 4, 5)$, $X_C = (1, 0, 2, 1, 3)$, respectively.*

The one-error-correcting rank-modulation code presented in [17, 18] is as follows. It is also proved in [17, 18] that this is a one-error-correcting code. However, we do not yet have specific constructions for codes that can correct more than one error.

Construction 6. (One-error-correcting code)

Let C_1, C_2 denote two rank-modulation codes constructed as follows. Let A be a general permutation whose coordinates are $(x_1, x_2, \ldots, x_{n-1})$. Then A is a codeword *in C_1 if and only if the following equation is satisfied:*

$$\sum_{i=1}^{n-1} i x_i \equiv 0 \mod (2n-1).$$

And A is a codeword in C_2 if and only if the following equation is satisfied:

$$\sum_{i=1}^{n-2} i x_i + (n-1)\cdot(-x_{n-1}) \equiv 0 \mod (2n-1).$$

Between C_1 and C_2, choose the code with more codewords as the error-correcting code **C**.

Since a permutation of length n has $n-1$ neighboring permutations at Kendall tau distance one, by the sphere packing bound, the size of a one-error-correcting rank-modulation code is at most

$$\frac{n!}{n} = (n-1)!.$$

So the following theorem shows that for the code of Construction 6, its size is at least half of the optimal size. It has been proved in [17].

Theorem 7.. *The rank-modulation code built in Construction 6 has a minimum size of*

$$\frac{(n-1)!}{2}.$$

B. Size of the One-error-correcting Code

It is interesting to understand how to use the one-error-correcting code of Construction 6 for encoding and decoding of information. Although the code is known to be nearly optimal, it has not been shown how to encode and decode data using the code. Note that it is not efficient to simply build a table mapping the codewords to data because when n is large, this approach is both space- and time-consuming. Our objective is to find an efficient algorithm to encode data using the code. For this purpose, we first need to understand the size of the code.

Let C_1 and C_2 denote the two codes described in Construction 6. That is, C_1 consists of all the permutations of length n whose coordinates $(x_1, x_2, \ldots, x_{n-1})$ satisfy the construction

$$\sum_{i=1}^{n-1} ix_i \equiv 0 \mod (2n-1),$$

and C_2 consists of all the permutations of length n whose coordinates $(x_1, x_2, \ldots, x_{n-1})$ satisfy the construction

$$\sum_{i=1}^{n-2} ix_i + (n-1) \cdot (-x_{n-1}) \equiv 0 \mod (2n-1).$$

Then the code of Construction 6, $\mathbf{C}$, has size

$$|\mathbf{C}| = \max\{C_1, C_2\}.$$

Since $\gcd(n-1, 2n-1) = 1$, there exists a unique solution $y \in \{0, 1, \ldots, 2n-2\}$ to the equation

$$(n-1)y \equiv r \pmod{2n-1}$$

for any $r \in \mathbb{Z}$. In particular, when $r = 1$, we get $y = 2n-3$ because $(n-1)(2n-3) = (n-2)(2n-1) + 1$.

Given $x_i \in \{0, 1, \ldots, i\}$ for $i = 1, 2, \ldots, n-2$, let us define r as

$$r = \left(\sum_{i=1}^{n-2} ix_i \pmod{2n-1} \right).$$

Then the solution $y \in \{0, 1, \ldots, 2n-2\}$ to the equation

$$\sum_{i=1}^{n-2} ix_i + (n-1)y \equiv 0 \pmod{2n-1}$$

is

$$\begin{aligned} y &\equiv -(2n-3)r \\ &\equiv 2r \pmod{2n-1} \end{aligned}$$

Therefore, a permutation with the coordinates $(x_1, x_2, \ldots, x_{n-1})$ is in C_1 if and only if

$$0 \leq (2r \pmod{2n-1}) < n$$

and

$$x_{n-1} = (2r \pmod{2n-1}).$$

Note that the condition $0 \leq (2r \pmod{2n-1}) < n$ is equivalent to the condition

$$0 \leq r < \left\lceil \frac{n}{2} \right\rceil \quad \text{or} \quad n \leq r < n + \left\lfloor \frac{n}{2} \right\rfloor .$$

Similarly, a permutation with the coordinates $(x_1, x_2, \ldots, x_{n-1})$ is in C_2 if and only if

$$0 \leq (-2r \pmod{2n-1}) < n$$

and

$$x_{n-1} = (-2r \pmod{2n-1}).$$

Note that the condition $0 \leq (-2r \pmod{2n-1}) < n$ is equivalent to the condition

$$r = 0 \quad \text{or} \quad \left\lceil \frac{n}{2} \right\rceil \leq r < n \quad \text{or} \quad n + \left\lfloor \frac{n}{2} \right\rfloor \leq r < 2n-1.$$

We present an algorithm that computes the size of the code $\mathbf{C}$. Note that for every combination of the first $n-2$ coordinates $(x_1, \ldots, x_{n-2})$, there is a unique permutation with coordinates $(x_1, \ldots, x_{n-2}, x_{n-1})$ that either belongs to C_1 or C_2 (or both if $r = 0$). The algorithm checks such combinations and obtains the values of $|C_1|$ and $|C_2|$, by which we get $|\mathbf{C}|$. Algorithm 8 has time complexity $O((n-1)!)$.

.

Algorithm 8 COMPUTE CODE SIZE

```
1: $n \leftarrow$ number of cells, $|C_1| \leftarrow 0$, $|C_2| \leftarrow 0$
2: for $(x_1, x_2, \ldots, x_{n-2}) \in \{0,1\} \times \{0,1,2\} \times \cdots \times \{0,1,\ldots,n-2\}$ do
3:     $r \leftarrow \left(\sum_{i=1}^{n-2} ix_i \pmod{2n-1}\right)$
4:     if $0 \le r < \lceil \frac{n}{2} \rceil$ or $n \le r < n + \lfloor \frac{n}{2} \rfloor$ then
5:         $|C_1| \leftarrow |C_1| + 1$
6:     end if
7:     if $r = 0$ or $\lceil \frac{n}{2} \rceil \le r < n$ or $n + \lfloor \frac{n}{2} \rfloor \le r < 2n - 1$ then
8:         $|C_2| \leftarrow |C_2| + 1$
9:     end if
10: end for
11: $|\mathbf{C}| \leftarrow \max\{|C_1|, |C_2|\}$
12: Output $|C_1|, |C_2|$ and $|\mathbf{C}|$
```

The results for $3 \le n \le 11$ are shown in Table I. (It is noticeable that the values of $|C_1|$ and $|C_2|$ are very close.) The table also shows the comparison between the code size $|\mathbf{C}|$ and its lower bound $\frac{(n-1)!}{2}$.

C. Efficient Encoding Algorithm

We present an efficient encoding algorithm based on codeword concatenation. It stores

$$\log_2\left(\frac{(n-1)!}{2}\right)$$

information bits per codeword (of length n) on average. Note that this performance matches the lower bound, $\frac{(n-1)!}{2}$, of the code size shown in Theorem 7.

Table I. Size of One-error-correcting Code

n	$\lvert C_1\rvert$	$\lvert C_2\rvert$	$\lvert \mathbf{C}\rvert$	$(n-1)!/2$
3	2	1	2	1
4	4	3	4	3
5	14	13	14	12
6	66	66	66	60
7	388	388	388	360
8	2688	2688	2688	2520
9	21346	21345	21346	20160
10	190990	190989	190990	181440
11	1900800	1900800	1900800	1814400

Let $n \geq 3$ and $m \geq 1$ be positive integers. Let

$$Q = \{0, 1, \ldots, \frac{(n-1)!}{2} - 1\}$$

be an alphabet of size $q = \frac{(n-1)!}{2}$. For every $v \in Q$, let

$$f(v) = (x_2(v), x_3(v), \ldots, x_{n-2}(v))$$

be the factoradic representation of v in the set

$$\{0, 1, 2\} \times \{0, 1, 2, 3\} \times \cdots \times \{0, 1, \ldots, n-2\}.$$

Specifically, we have

$$x_2(v) = (v \mod 3)\,;$$

and for $i = 3, \ldots, n-2$, we have

$$x_i(v) = \left(\left\lfloor \frac{v}{3 \times 4 \times \cdots \times i} \right\rfloor \mod (i+1) \right).$$

We store a sequence of m variables from the alphabet Q:

$$v_1, v_2, \ldots, v_m$$

in $mn + 3$ cells as follows. Let

$$A_1, A_2, \ldots, A_m$$

denote m codewords from the one-error-correcting code C of Construction 6. Here the code C has length n, and the values of $A_1, \ldots, A_m$ will be determined by $v_1, \ldots, v_m$. For $i = 1, \ldots, m$, let

$$X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,n-1})$$

denote the coordinates of the codeword A_i. (Note that no two permutations have the same coordinates, so the coordinates uniquely determine the corresponding permutation.) And we let

$$A_0 \in \{[1,2,3],[3,2,1]\}$$

denote a permutation of length 3. (Note that {[1,2,3],[3,2,1]} is a one-error-correcting code of length 3, because $d([1,2,3],[3,2,1]) = 3$.) Among the $mn+3$ cells, the ranks of the first 3 cells become the permutation A_0. We partition the remaining mn cells into m cell groups, where every cell group has n cells. The cell levels of these m separate cell groups induce the permutations $A_1, A_2, \ldots, A_m$.

We now show how to encode the data

$$(v_1, v_2, \ldots, v_m) \in Q^m$$

into the permutations

$$(A_0, A_1, A_2, \ldots, A_m).$$

(That is, given the data $(v_1, v_2, \ldots, v_m)$, we show how to compute $(A_0, A_1, A_2, \ldots, A_m)$.) For $i = 1, \ldots, m$, we will decide if A_i is a codeword of C_1 or C_2 adaptively. Let

$$s_i \in \{0, 1\}$$

be a variable such that if we choose the code C_1 for A_i, then $s_i = 0$; if we choose the code C_2 for A_i, then $s_i = 1$. And for convenience of presentation, let $s_{m+1} = 0$.

The encoding is as follows. We compute $X_m, X_{m-1}, \ldots, X_1$ sequentially. To compute $X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,n-1})$ (where $1 \le i \le m$), let

$$x_{i,1} = s_{i+1},$$

$$(x_{i,2}, x_{i,3}, \ldots, x_{i,n-2}) = f(v_i),$$
$$r = \left(\sum_{j=1}^{n-2} i x_{i,j} \pmod{2n-1} \right).$$

If $0 \le r < \lceil \frac{n}{2} \rceil$ or $n \le r < n + \lfloor \frac{n}{2} \rfloor$, then

$$x_{i,n-1} = (2r \mod (2n-1)) \quad \text{and} \quad s_i = 0;$$

otherwise,

$$x_{i,n-1} = (-2r \mod (2n-1)) \quad \text{and} \quad s_i = 1.$$

Given $X_1, \ldots, X_m$, it is simple to get the permutations $A_1, \ldots, A_m$. As the final step, if $s_1 = 0$, let $A_0 = [1, 2, 3]$; if $s_1 = 1$, let $A_0 = [3, 2, 1]$.

Given the uncorrupted codewords $A_1, A_2, \ldots, A_m$ (or their coordinates $X_1, X_2, \ldots, X_m$), it is simple to recover the data $(v_1, v_2, \ldots, v_m) \in Q^m$: For $i = 1, 2, \ldots, m$, we have

$$\begin{aligned} v_i &= f^{-1}\left((x_{i,2}, x_{i,3}, \ldots, x_{i,n-2})\right) \\ &= x_{i,2} + 3 \cdot x_{i,3} + 3 \cdot 4 \cdot x_{i,4} + \cdots + (\textstyle\prod_{j=3}^{n-2} j) \cdot x_{i,n-2} \end{aligned}$$

We now discuss how to decode information when the codewords are corrupted by errors. The following theorem proves that if each codeword contains at most one error, we can correctly recover all the data.

Theorem8.. *Let $A_0, A_1, \ldots, A_m$ be the codewords corresponding to the data $(v_1, \ldots, v_m) \in Q^m$. Let $A'_0, A'_1, \ldots, A'_m$ be the received noisy codewords. If*

$$d(A_i, A'_i) \leq 1$$

for $i = 0, 1, \ldots, m$, then given the noisy codewords $A'_0, A'_1, \ldots, A'_m$, we can correctly recover all the data $(v_1, \ldots, v_m)$.

Proof. We show how to recover $v_1, \ldots, v_m$ sequentially. First, since $d([1, 2, 3], [3, 2, 1]) = 3$, we can correctly recover s_1. If $s_1 = 0$ (respectively, $s_1 = 1$), we know A_1 should be a codeword of the code C_1 (respectively, code C_2), and then from A'_1 we can recover A_1 (because A'_1 contains at most one error). From A_1 we can get its coordinates $X_1 = (x_{1,1}, x_{1,2}, \ldots, x_{1,n-1})$. Then we get

$$\begin{aligned} v_1 &= f^{-1}\left((x_{1,2}, x_{1,3}, \ldots, x_{1,n-2})\right) \\ &= x_{1,2} + 3 \cdot x_{1,3} + 3 \cdot 4 \cdot x_{1,4} + \cdots + \left(\textstyle\prod_{j=3}^{n-2} j\right) \cdot x_{1,n-2} \end{aligned}$$

and

$$s_2 = x_{1,1}.$$

In the same way, we can sequentially recover $v_2, s_3, v_3, s_4, \ldots, s_m, v_m$. More specifically, for $i = 2, \ldots, m$, with s_i and A'_i we can get A_i and X_i. Then we get

$$\begin{aligned} v_i &= f^{-1}\left((x_{i,2}, x_{i,3}, \ldots, x_{i,n-2})\right) \\ &= x_{i,2} + 3 \cdot x_{i,3} + 3 \cdot 4 \cdot x_{i,4} + \cdots + \left(\prod_{j=3}^{n-2} j\right) \cdot x_{i,n-2} \end{aligned}$$

and $s_{i+1} = x_{i,1}$. So the theorem holds. $\square$

When $m \to \infty$, the rate of this coding scheme is

$$\lim_{m\to\infty} \frac{m \log_2\left((n-1)!/2\right)}{mn+3} = \frac{1}{n} \cdot \log_2 \frac{(n-1)!}{2}$$

bits per cell.

D. Decoding Algorithm

We consider the decoding of codewords when there are errors. Let $A'_0, A'_1, \ldots, A'_m$ denote the received noisy codewords. The decoding objective is to recover the stored data $v_1, \ldots, v_m$.

We will do decoding for $v_1, v_2, \ldots, v_m$ sequentially. Let $s'_1, \ldots, s'_m, s'_{m+1}$ denote the decoded values of $s_1, \ldots, s_m, s_{m+1}$. Let $v'_1, \ldots, v'_m$ denote the decoded values of $v_1, \ldots, v_m$. As the initial step, we compute s'_1 as follows: If

$$d(A'_0, [1,2,3]) \leq 1$$

(which means $A'_0 = [1,2,3]$, $[2,1,3]$ or $[1,3,2]$), let $s'_1 = 0$; otherwise (which means $d(A'_0, [3,2,1]) \leq 1$), let $s'_1 = 1$.

Now for $i = 1, 2, \ldots, m$, assuming that the value of s'_i has been computed, we show how to compute v'_i and s'_{i+1}. To compute v'_i, if $s'_i = 0$, we let B_i be the codeword in code C_1 whose Kendall tau distance to A'_i is the smallest. (B_i can be found through exhaustive search, where we gradually increase the Kendall tau distance. Since C_1 is a one-ECC with high codeword density, this search complexity is usually small.) Let

$Y_i = (y_{i,1}, y_{i,2}, \ldots, y_{i,n-1})$ be the coordinates of B_i. Let

$$\begin{aligned} v'_i &= f^{-1}\left((y_{i,2}, y_{i,3}, \ldots, y_{i,n-2})\right) \\ &= y_{i,2} + 3 \cdot y_{i,3} + 3 \cdot 4 \cdot y_{i,4} + \cdots + (\textstyle\prod_{j=3}^{n-2} j) \cdot y_{i,n-2} \end{aligned}$$

be the decoded value for v_1.

To compute s'_{i+1}, we notice that $s_{i+1} = x_{i,1}$, whose value depends only on the relative order of the first cell and second cell in the i-th cell group: If the first cell has a higher rank (i.e., higher cell level) than the second cell, then $x_{i,1} = 0$; otherwise, $x_{i,1} = 1$. Typically, the greater the gap between the ranks of these two cells, the less likely that errors will change their relative order. To reduce the potential error propagation when we decode the chain of codewords sequentially, we compute s'_{i+1} as follows: If $d(A'_i, B_i) \leq 1$, let $s'_{i+1} = y_{i,1}$; otherwise, let $X'_i = (x'_{i,1}, x'_{i,2}, \ldots, x'_{i,n-1})$ denote the coordinates of A'_i, and let $s'_{i+1} = x'_{i,1}$.

It can be seen that as long as s_i is decoded correctly, the decoding error of the codeword A_i will not be propagated to the decoding of the codeword A_{i+1}.

E. Performance Evaluation

We use simulations to evaluate the performance of the coding scheme. We approximate the noise in cell levels by the Gaussian distribution. Specifically, if a cell has the i-th lowest rank in its group, then its level is $(i-1)+\epsilon$, where ϵ is i.i.d. Gaussian noise with zero mean and variance σ^2. Here the noise ϵ models both the imprecise programming of the cell level and the disturbances to the cell level after programming.

We uniformly randomly generate the data $(v_1, v_2, \ldots, v_m) \in Q^m$. Let $(v'_1, v'_2, \ldots, v'_m)$ denote the decoded data. We define the *Symbol Error Rate* as

$$\frac{|\{i \mid 1 \leq i \leq m,\ v_i \neq v'_i\}|}{m}.$$

Table II tabulates symbol error rate (SER) for different values of 'n' (number of cells) and σ^2 (noise power, normalized). Information symbols (to be encoded/decoded) were chosen uniformly randomly from the set $\{0, 1, \cdots, \frac{(n-1)!}{2}\}$. The length of information sequence, the value 'm', is 10^4 for $n = 4$ and 10^5 for $n = 6, 8$. This data is also plotted in Fig 1, where x-axis is noise power and y-axis is observed SER.

Table II. Rank Modulation Codes: Symbol Error Rate

$n \backslash \sigma^2$	0.1	0.2	0.3	0.4	0.5	0.6	0.7
4	0	0.0024	0.0121	0.0261	0.0367	0.0603	0.0772
6	0.0006	0.0159	0.0559	0.1078	0.1674	0.2199	0.2769
8	0.0011	0.0373	0.1213	0.2197	0.3057	0.3847	0.4790

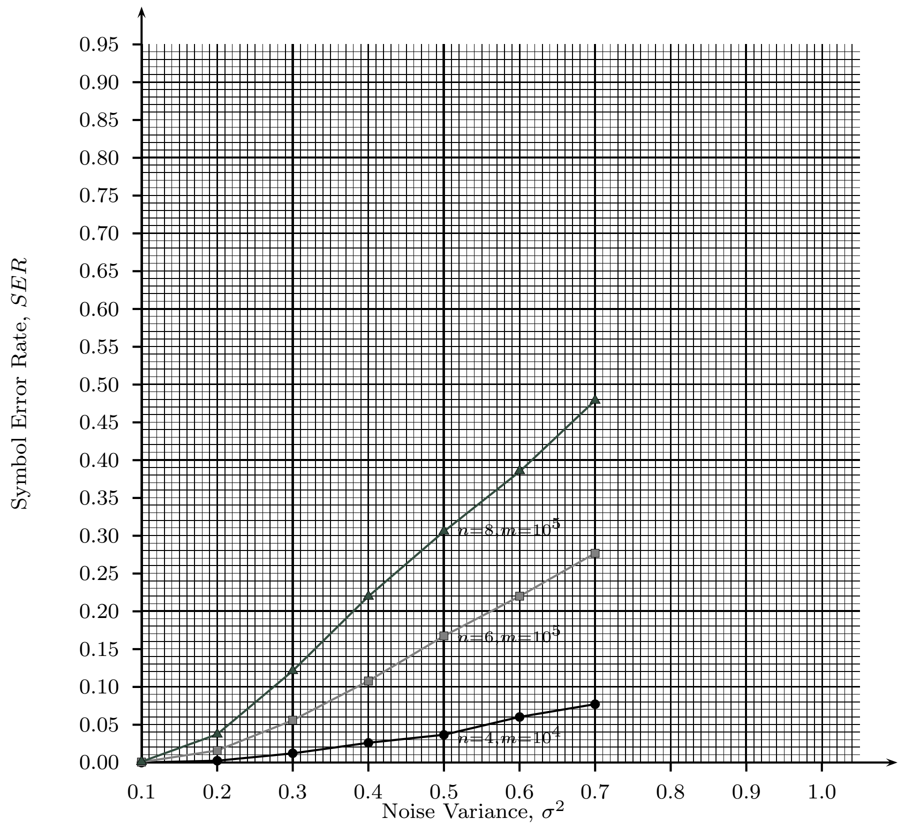

Fig. 1. Rank Modulation Codes: Symbol Error Rate

For comparison purposes, we also simulated the performance of BCH codes. The flash memory was assumed to be a multi level cell memory, with $n = 4$ and $n = 8$ levels. For $n = 4$, data is encoded with a narrow sense (511,484) 3-error-correcting binary BCH code and for $n = 8$, we used (255,231) 3-error-correcting binary BCH code. Data symbols were generated uniformly randomly from an alphabet of size four and eight for $n = 4$ and $n = 8$ respectively. These symbols are converted to binary format, encoded with appropriate BCH code and stored in multi-level flash memory. The noise we generate to simulate disturbances in flash memory is such that it always increases cell charge levels (specifically, we take absolute value of a gaussian random variable with zero mean and specified variance). This will approximate asymmetric nature of noise in flash memory.

As can be observed from Table III, the performance of BCH codes under the conditions we discussed is poorer compared to rank modulation codes in Table II.

Table III. BCH Codes: Symbol Error Rate

$n\backslash\sigma^2$	0.1	0.2	0.3	0.4	0.5	0.6
4	0.8671	0.8732	0.8952	0.8651	0.9022	0.9066
8	0.8531	0.8677	0.8908	0.9036	0.9086	0.9148

CHAPTER III

LINEAR REWRITING CODES

In this chapter, we study linear rewriting codes for flash memories. We first introduce rewriting codes and a linear write-once memory (WOM) code presented by Rivest and Shamir [22], and generalize its definition. We then present a pseudo-polynomial-time rewriting algorithm that locally minimizes the number of programmed cells for each rewrite.

A. Introduction to Rewriting Codes

A *single-level cell (SLC)* in a flash memory has two possible levels: 0 and 1. Without the block erasure, an SLC can change from level 0 to level 1, but not from level 1 to level 0. We consider rewriting codes that store data in n single-level cells and enable the data to be rewritten (i.e., modified) multiple times without the block erasure. More specifically, we assume that with each rewrite, the data can change from its current value to any other value in its alphabet (i.e., unconstrained rewriting). This is the write-once memory (WOM) model proposed by Rivest and Shamir in their seminal paper [22]. (Examples of codes for constrained rewriting include floating codes, buffer codes, etc. [3, 12, 14].)

Consider n single-level cells, whose levels are denoted by

$$c_1, c_2, \ldots, c_n.$$

For $i = 1, \ldots, n$, we have $c_i \in \{0, 1\}$. Let

$$L = \{0, 1, \ldots, \ell - 1\}$$

be an alphabet of size ℓ. A WOM code stores data from the alphabet L in the cells.

It has a *decoding function*

$$F_d \ : \ \{0,1\}^n \to L$$

and an *update function*

$$F_u \ : \ \{0,1\}^n \times L \to \{0,1\}^n$$

explained as follows. When the cell levels are $\vec{c} = (c_1, c_2, \ldots, c_n) \in \{0,1\}^n$, the stored data is $F_d(\vec{c}) \in L$. When the cell levels are currently $\vec{c} = (c_1, c_2, \ldots, c_n)$ and we need to rewrite the data as $s \in L$, the code will increase the cell levels to $F_u(\vec{c}, s) \in \{0,1\}^n$. (Naturally, we require $F_d(F_u(\vec{c}, s)) = s$. Also, if $F_u(\vec{c}, s) = (c'_1, c'_2, \ldots, c'_n)$, then $c'_i \geq c_i$ for $i = 1, \ldots, n$.)

We assume that initially (i.e., before rewriting), all the cell levels are 0. Let t denote the number of rewrites that are guaranteed to succeed, regardless of what the sequence of rewrites are (namely, the worst-case performance). A WOM code that maximizes t is called optimal.

There has been a number of papers on the capacity of WOM codes and some code constructions [5, 6, 8, 10, 19, 21, 27]. We introduce a *linear code* proposed by Rivest and Shamir, which is proved to have asymptotically optimal rewriting performance [22].

Construction 9. LINEAR WOM CODE

Let $\ell = n + 1$. For the linear WOM code, the cell levels $\vec{c} = (c_1, c_2, \ldots, c_n) \in \{0,1\}^n$ represent the data

$$F_d(\vec{c}) = \left(\sum_{i=1}^{n} i c_i \mod \ell \right).$$

Example10.. *Let $\ell = 9$ and $n = 8$. The linear WOM code has $F_d(\vec{c}) = \left(\sum_{i=1}^{8} i c_i \mod 9\right)$. When the sequence of rewrites change the data as $0 \to 6 \to 2 \to 7 \to 8$, the cell levels $\vec{c} = (c_1, \ldots, c_8)$ can change as $(0,0,0,0,0,0,0,0) \to (0,0,0,0,0,1,0,0) \to$*

$(0,0,0,0,1,1,0,0) \to (1,0,0,1,1,1,0,0) \to (1,1,0,1,1,1,0,1)$.

For the linear WOM code in Construction 9, it can be shown that as long as there are more than $\ell/2$ cells at level 0, the next rewrite can be realized by changing at most two cells from level 0 to level 1; therefore the code supports at least $\ell/4$ rewrites [22]. Since $n = \ell - 1$ and every rewrite has to change the level of at least one cell, the rewriting performance of the code is asymptotically optimal in n.

B. Generalized Linear WOM Codes and Rewriting Algorithm

Our interest in linear WOM codes comes from the fact that linear codes have regular structures and often enable more tractable analysis. However, for the linear WOM code in Construction 9, no algorithm has been presented on how to use it for rewriting (other than increasing as few cell levels as possible for every rewrite based on brute-force search). Note that if a rewrite requires i cell levels to be increased, the time complexity of the brute-force search (for finding those i cell levels) will be $O(n^i)$, which is exponential in i. For the linear WOM code, once more than $\ell/2$ cell have been changed to level 1, the number of cell levels to change for a rewrite can be large, for which the brute-force method of rewriting becomes very time consuming. This motivates us to study efficient rewriting algorithms for the linear WOM code.

The following code construction generalizes the linear WOM code.

Construction 11. GENERALIZED LINEAR WOM CODE

Let $b_1, b_2, \ldots, b_n$ be n integer parameters in the set $\{1, 2, \ldots, \ell - 1\}$. For the generalized linear WOM code, the cell levels $\vec{c} = (c_1, c_2, \ldots, c_n) \in \{0,1\}^n$ represent the data

$$F_d(\vec{c}) = \left(\sum_{i=1}^{n} b_i c_i \mod \ell \right).$$

For every rewrite, we would like to increase as few cell levels from 0 to 1 as possible. This local optimization problem can be formulated as the *minimum cost rewriting problem* below. It is not difficult to see that in this problem, $x_1, \ldots, x_m$ represent the coefficients of those cells whose levels are 0 before the rewrite, and Δ represents the difference between the new data after the rewrite and the old data before the rewrite (modulo ℓ).

Definition 12.. Minimum cost rewriting problem

Let $x_1, x_2, \ldots, x_m$ and Δ be $m+1$ integer parameters in the set $\{1, 2, \ldots, \ell-1\}$. Find a set $S \subseteq \{1, 2, \ldots, m\}$ of minimum cardinality such that

$$\sum_{i \in S} x_i \equiv \Delta \mod \ell.$$

The above problem is NP hard because the NP-complete *subset-sum problem* can be reduced to it. In the following, we present a pseudo-polynomial time dynamic programming algorithm to solve it. Its time complexity is $O(m\ell)$.

For every set $S \subseteq \{1, 2, \ldots, m\}$, we associate it with a cost $c(S)$:

$$c(S) = \begin{cases} |S|, & \text{if } \sum_{i \in S} x_i \equiv \Delta \mod \ell \\ \infty, & \text{otherwise} \end{cases}$$

It is simple to see that for the minimum cost rewriting problem, the objective is to find the set S of the minimum cost.

For $i \in \{1, 2, \ldots, m\}$ and $s \in \{0, 1, \ldots, \ell-1\}$, we define $Q(i, s)$ as follows:

- If there does not exist a subset $S \subseteq \{1, 2, \ldots, i\}$ such that

$$\sum_{j \in S} x_j \equiv s \mod \ell,$$

then let $Q(i, s) = \infty$;

- Otherwise, let S_{min} denote the subset of $\{1, 2, \ldots, i\}$ with the minimum cardinality such that
$$\sum_{j \in S_{min}} x_j \equiv s \mod \ell,$$
and let $Q(i, s) = |S_{min}|$.

By default, for the empty set $\emptyset$, we have $\sum_{j \in \emptyset} x_j = 0$.

Initially, we set
$$Q(i, 0) = 0$$
for all $i \in \{1, 2, \ldots, m\}$, set
$$Q(1, x_1) = 1,$$
and set
$$Q(1, s) = \infty$$
for $s \in \{1, 2, \ldots, \ell - 1\} \setminus \{x_1\}$. Then, for $i = 2, 3, \ldots, m$ and $s \in \{1, 2, \ldots, \ell - 1\}$, we have the following recursion:
$$\begin{aligned} & Q(i, s) \\ = \; & \min\{\; Q(i-1, s),\; 1 + Q(i-1, s - x_i \mod \ell)\} \end{aligned}$$

Clearly, for the minimum cost rewriting problem, the optimal solution – which we denote by S_{opt} – has cost $c(S_{opt}) = Q(m, \Delta)$.

When $Q(m, \Delta) \neq \infty$, we can compute S_{opt} using algorithm 13.

C. Performance Evaluation

We have conducted extensive simulations to evaluate the rewriting performance of the codes. Table IV tabulates average rewriting performance of generalized linear WOM codes. These data were obtained from simulations for different values of (l, n) (refer Construction 11). For every pair (l, n), the b_i's (refer construction:8) were chosen

Algorithm 13 COMPUTE S_{opt}

```
1: $S_{opt} \leftarrow \emptyset$
2: for $i = m, m-1, \ldots, 2$ do
3:     if $Q(i-1, T) > Q(i, T)$ then
4:         $S_{opt} \leftarrow S_{opt} \cup \{x_i\}$ and $T \leftarrow (T - x_i \mod \ell)$
5:     end if
6: end for
7: if $T = x_1$ then
8:     $S_{opt} \leftarrow S_{opt} \cup \{x_1\}$
9: end if
10: Output $S_{opt}$
```

uniformly randomly from the set $\{1, \cdots, l-1\}$. Average rewriting performance of these codes for different information sequences, chosen randomly, was then recorded. The information sequences were generated so that every next data symbol required at least one rewrite. The variance among the number of rewrites supported by each pair (l, n), for the same information sequences, is also tabulated in Table V.

Table VI also tabulates average rewriting performance of generalized linear WOM codes for different values of (l, n)(refer Construction 11). Unlike the simulations for Table IV, the b_i's are prime numbers from the set $\{0, \cdots, l-1\}$. Specifically, each prime in the set $\{0, \cdots, l-1\}$ was distributed among b_i's equally on average. The information sequences were generated randomly and each next data symbol in the information sequence required at least one rewrite. The variance among the number of rewrites supported by each pair (l, n), for the same information sequences, is also tabulated in Table VII. As can be observed from Table VI, the average rewriting performance for this choice of coefficients (b_i's) is comparatively poorer.

Table IV. Linear Rewriting Codes: Average Number of Rewrites

$l \diagdown n$	50	100	150	200
64	26.14	62.80	103.83	146
128	22.63	54.01	89.81	127.75
256	20.25	48.56	79.41	111.92
512	17.85	44.19	72.67	101.85
1024	15.53	40.28	66.67	93.97

Table V. Linear Rewriting Codes: Variance

$l \diagdown n$	50	100	150	200
64	3.0913	8.3838	14.5264	23.4747
128	2.0940	5.4847	11.0847	14.2096
256	1.4621	3.2590	6.7494	7.3067
512	1.3409	2.5191	3.6375	6.9571
1024	1.2415	2.1430	1.9809	3.9688

Table VI. Linear Rewriting Codes with Prime Coefficients: Average Number of Rewrites

$l \diagdown n$	50	100	150	200
64	22.04	45.83	68.65	92.08
128	20.62	43.31	66.39	88.38
256	18.86	41.00	64.34	86.34
512	16.23	39.59	62.08	82.68
1024	14.56	36.08	58.38	81.06

Table VII. Linear Rewriting Codes with Prime Coefficients: Variance

$l \diagdown n$	50	100	150	200
64	3.1384	9.6011	9.6675	17.094
128	1.9756	5.6539	8.5379	16.076
256	2.1604	4.1600	6.0444	10.084
512	1.3771	3.5419	6.0336	8.5376
1024	1.2664	2.8536	5.8356	5.3564

CHAPTER IV

CONCLUSIONS

Flash memories have become by far the most widely used non-volatile memories. Due to their unique properties – notably the block erasure property – cell programming and data rewriting have been two important research areas. In this work, we have studied the error-correcting rank-modulation codes and linear rewriting codes. We have presented efficient algorithms for encoding and decoding of such codes. There are still many open problems in these two areas. In particular, they include the design of high-rate rank-modulation codes that can correct multiple errors, and rewriting codes with encoding-decoding algorithms of polynomial time complexity.

REFERENCES

[1] A. Bandyopadhyay, G. Serrano and P. Hasler, “Programming analog computational memory elements to 0.2% accuracy over 3.5 decades using a predictive method,” in *Proc. IEEE International Symposium on Circuits and Systems*, Kobe, Japan, May 2005, pp. 2148-2151.

[2] A. Barg and A. Mazumdar, “Codes in permutations and error correction for rank modulation,” available as Arxiv preprint cs.IT/0908.4094, 2009.

[3] V. Bohossian, A. Jiang and J. Bruck, “Buffer codes for asymmetric multi-level memory,” in *Proc. IEEE International Symposium on Information Theory (ISIT2007)*, Nice, France, June 2007, pp. 1186-1190.

[4] P. Cappelletti, C. Golla, P. Olivo and E. Zanoni (*Eds.*), *Flash Memories*, 1st Edition, Amsterdam, The Netherlands: Kluwer, 1999.

[5] G. D. Cohen, P. Godlewski and F. Merkx, “ Linear binary code for write-once memories,” *IEEE Transactions on Information Theory*, vol. IT-32, no. 5, pp. 697-700, September 1986.

[6] A. Fiat and A. Shamir, “Generalized ‘write-once’ memories,” *IEEE Transaction on Information Theory*, vol. IT-30, no.3, pp. 470-480, May 1984.

[7] H. Finucane, Z. Liu and M. Mitzenmacher, “Designing floating codes for expected performance,” in *Proc. 46th Annual Allerton Conference on Communications, Control and Computing*, Monticello, Illinois, USA, September 2008, pp. 1389-1396

[8] 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, January 1999.

[9] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," *ACM Computing Surveys*, vol. 37, no. 2, pp. 138-163, June 2005.

[10] C. Heegard, "On the capacity of permanent memory," *IEEE Transactions on Information Theory*, vol. IT-31, no. 1, pp. 34-42, January 1985.

[11] A. Jiang, "On the generalization of error-correcting WOM codes," in *Proc. IEEE International Symposium on Information Theory (ISIT2007)*, Nice, France, June 2007, pp. 1391-1395.

[12] A. Jiang, V. Bohossian and J. Bruck, "Floating codes for joint information storage in write asymmetric memories," in *Proc. IEEE International Symposium on Information Theory (ISIT2007)*, Nice, France, June 2007, pp. 1166-1170.

[13] A. Jiang and J. Bruck, "Joint coding for flash memory storage," in *Proc. IEEE International Symposium on Information Theory (ISIT2008)*, Toronto, ON, Canada, July 2008, pp. 1741-1745

[14] A. Jiang, M. Langberg, M. Schwartz and J. Bruck, "Universal rewriting in constrained memories," in *Proc. IEEE International Symposium on Information Theory (ISIT2009)*, Seoul, Korea, June-July 2009, pp. 1219-1223

[15] A. Jiang, R. Mateescu, M. Schwartz and J. Bruck, "Rank modulation for flash memories," in *Proc. IEEE International Symposium on Information Theory (ISIT2008)*, Toronto, ON, Canada, July 2008, pp. 1731-1735.

[16] A. Jiang, R. Mateescu, M. Schwartz and J. Bruck, "Rank modulation for flash memories," *IEEE Transactions on Information Theory*, vol. 55, no. 6, pp. 2659-2673, June 2009.

[17] A. Jiang, M. Schwartz and J. Bruck, "Error-correcting codes for rank modulation," in *Proc. IEEE Internatinal Symposium on Information Theory (ISIT2008)*, Toronto, ON, Canada, July 2008, pp. 1736-1740.

[18] A. Jiang, M. Schwartz and J. Bruck, "Correcting charge-constrained errors in the rank modulation scheme," *IEEE Transactions on Information Theory*, vol. 56, no. 5, pp. 2112-2120, May 2010.

[19] A. V. Kuznetsov and A. J. H. Vinck, "On 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.

[20] H. Mahdavifar, P. H. Siegel, A. Vardy, J. K. Wolf and E. Yaakobi, "A nearly optimal construction of flash codes," in *Proc. IEEE International Symposium on Information Theory (ISIT2009)*, Seoul, Korea, June-July 2009, pp. 1239-1243.

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

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

[23] M. Schwartz, "Constant-weight Gray codes for local rank modulation," available as arXiv preprint arXiv:1002.1436, February 2010.

[24] I. Tamo and M. Schwartz, "Correcting limited-magnitude errors in the rank-modulation scheme," in *Proc. Information Theory and Applications Workshop*

(ITA), San Diego, CA, U.S.A., January 2010, pp. 1-2.

[25] Z. Wang and J. Bruck, "Partial rank modulation for flash memories," to appear in *Proc. IEEE International Symposium on Information Theory (ISIT2010)*, Austin, TX, USA, June 2010.

[26] Z. Wang, A. Jiang and J. Bruck, "On the capacity of bounded rank modulation for flash memories," in *Proc. IEEE International Symposium on Information Theory (ISIT2009)*, Seoul, Korea, June-July 2009, pp. 1234-1238.

[27] 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.

[28] E. Yaakobi, A. Vardy, P. H. Siegel and J. K. Wolf, "Multidimensional flash codes," in *Proc. 46th Annual Allerton Conference on Communications, Control and Computing*, Monticello, Illinois, USA, September 2008, pp. 392-399.

[29] F. Zhang, H. Pfister and A. Jiang, "LDPC codes for rank modulation in flash memories," to appear in *Proc. IEEE International Symposium on Information Theory (ISIT)*, Austin, TX, USA, June 2010.

VITA

Name:	Shoeb Ahmed Mohammed
Address:	Department of Electrical and Computer Engineering, Texas A&M University, 214 Zachry Engineering Center, College Station, TX 77843-3128
Email Address:	shoebahmed@hotmail.com
Education:	B.Tech., Electronics and Communication Engineering, Indian Institute of Technology Roorkee, 2008
	M.S., Electrical Engineering, Texas A&M University, 2010