International Symposium on Information Theory and its Applications, ISITA2008
Auckland, New Zealand, 7-10, December, 2008

On The Capacity of Flash Memories

Anxiao (Andrew) Jiang[†] and Jehoshua Bruck[‡]

[†] Computer Science Department
Texas A&M University
College Station, TX, 77843, U.S.A.
E-mail: ajiang@cs.tamu.edu

[‡] Electrical Engineering Department
California Institute of Technology
Pasadena, CA, 91125, U.S.A.
E-mail: bruck@caltech.edu

Abstract

Flash memories are the most widely used type of non-volatile electronic memories. Compared to magnetic recording and optical recording, flash memories have the unique property that their cell levels, which represent data, are programmed using an iterative procedure that monotonically shifts each cell level upward toward its target value. In this paper, we study the capacity of flash memories to store data. We explore the relationship among their capacity, programming precision and programming time. The study is focused on the capacity of single cells, and an optimal programming algorithm is presented.

1. INTRODUCTION

Flash memories are the most widely used type of non-volatile electronic memories (NVMs) due to their physical endurance and high performance [2]. In a flash memory, floating-gate cells are organized as blocks, with 10^4 to 10^6 cells per block. The memory stores data in the cells by appropriately setting the cells' levels, where a cell's level can be increased or decreased by injecting charge into the cell or removing charge from the cell. In current flash memories, discrete cell levels are chosen to represent data. Therefore, a cell with q levels can store $\log_2 q$ bits. Most current products use single-level cells, where $q = 2$. To increase data density, multi-level cells, where $q > 2$, are being actively explored [2].

Flash memories have a prominent feature of cell programming. Although a cell's level can be efficiently increased via charge injection, to decrease it, the whole block must be erased (i.e., all cell levels be lowered to the minimum value) and then reprogrammed. On the other hand, the precision of charge injection is limited, which means that the actual increase in cell level usually differs from the aimed increase. Due to the prohibitively high cost of block erasure/reprogramming [5], in the current technology, a cell is programmed using a conservative approach: the memory circuit iteratively injects charge into the cell and then measures the level [1, 3, 4]. Initially, the cell is at level 0, which is the minimum value of the cell level. In each round, some charge is injected into the cell, and the new cell level is measured. The injection is small enough so that the risk of over-shooting is sufficiently small. The cell level shifts upward monotonically with each round, until it reaches the target value.

In this paper, we study the capacity of flash memories for storing data. We focus on the capacity of single flash cells. Let $\mathcal{A}$ denote the maximum value that a cell level can take. The actual cell level can be any number in the continuous range $[0, \mathcal{A}]$. Divide $[0, \mathcal{A}]$ into ℓ intervals:

$$[0, a_1), [a_1, a_2), \cdots, [a_{\ell-2}, a_{\ell-1}), [a_{\ell-1}, \mathcal{A}],$$

and map them to ℓ information symbols: $\{1, 2, \cdots \ell\}$. The intervals are set in a way such that to store the symbol $i \in \{1, 2, \cdots, \ell\}$ in the cell, we can shift the cell level from 0 to the i-th interval (via a cell-programming method) with guaranteed success. The *capacity* is defined as the maximum amount of information that can be stored in a cell with this approach. That is, the capacity equals $\log \ell$, where the value ℓ is maximized under the flash memory's constraints on cell programming.

We study the relationship among the capacity of a cell, the programming precision, and the programming time. Different from other conventional storage media (e.g., magnetic and optical recording), writing in flash memories is a multi-stage process because it takes multiple rounds and the programming strategy is adaptively chosen after every round based on the actual cell level. The number of rounds used to program a cell is a strong indicator of the programming time. In this work, we present a solution that derives the capacity of flash cells. We also present a cell-programming algorithm that achieves the capacity.

The paper is organized as follows. In Section 2, a programming model for flash cells is introduced. In Section 3, the capacity of a flash cell is derived. In Section 4, an efficient programming algorithm that achieves the capacity is obtained. In Section 5, the conclusions are presented.

This work was supported in part by NSF CAREER Award CCF-0747415, and by NSF grant ECCS-0802107.

2. Cell Programming Model

Let $[0, \mathcal{A}]$ denote the continuous range of values that a cell level can take. Given a positive integer k, let $[k]$ denote the set $\{1, 2, \cdots, k\}$. Divide $[0, \mathcal{A}]$ into ℓ intervals

$$\{\pi_i | i \in [\ell]\},$$

where $\pi_1 = [0, a_1)$, $\pi_i = [a_{i-1}, a_i)$ for $2 \leq i \leq \ell - 1$, and $\pi_\ell = [a_{\ell-1}, \mathcal{A}]$.[1] These intervals are mapped to symbols in the alphabet $[\ell]$. $\forall\ i \in [\ell]$, if a cell stores the information symbol i, its cell level should be in the interval π_i.

We assume the flash memory circuit to have a programming resolution Δ. In a round of programming, the circuit aims to increase a cell's level by $i\Delta$, for some integer i. This aimed cell-level increase, $i\Delta$, is controlled by the discrete adjustment of the programming voltage or current. Due to noise in the charge-injection process, the actual increase of the cell level deviates from the aimed value. The programming noise can be characterized by a probability density function:

$$f_{x_0, i\Delta} : [\kappa_{low}(x_0, i\Delta), \kappa_{up}(x_0, i\Delta)) \to \mathbb{R}^+,$$

where κ_{low} and κ_{up} denote the minimum value and the maximum value of the noise, respectively, when the initial cell level is x_0 and the aimed increase of the cell level is $i\Delta$. Both κ_{low} and κ_{up} are functions of x_0 and $i\Delta$. When the programming noise is x, the actual cell level after programming becomes $x_0 + i\Delta + x$.

It is not easy to accurately model the programming noise. The distribution $f_{x_0, i\Delta}$ varies among cells [2]. In this paper, we assume the following constraints:

$$\kappa_{low} = i\Delta(1 - \epsilon),\ \kappa_{up} = i\Delta(1 + \delta),$$

for some parameters $\epsilon \in (0, 1)$ and $\delta > 0$. This model has the property that the actual increase of the cell level is always positive, and the higher the aimed cell-level increase is, the larger the noise can be. By setting ϵ and δ sufficiently large, we can make sure that the actual noise is contained in the region $[\kappa_{low}(x_0, i\Delta), \kappa_{up}(x_0, i\Delta)) = [i\Delta(1 - \epsilon), i\Delta(1 + \delta))$. The model can be refined if more knowledge on the programming noise is known.

A flash cell is programmed using multiple rounds, where the cell level is increased in each round. The number of rounds of programming is a strong indicator of the total time used for programming. Let's use r to denote the maximum number of rounds allowed to be used for programming a cell. Then the storage capacity of a cell is not only a function of the programming noise, but also the parameter r, because the more rounds of programming are used, the smaller the intervals $\pi_1, \cdots, \pi_\ell$ can be. There is a trade-off between the programming time and the capacity $\log \ell$. A programming process can be represented by r integers $z_1, z_2, \cdots, z_r$, where in the i-th round ($i \in [r]$), the memory circuit aims to increase the cell level by $z_i\Delta$. (Before programming, the cell level is zero. After programming, the cell level falls in the interval π_j if the information symbol to store is $j \in [\ell]$. Here z_i can be zero because once the cell level enters the interval π_j, the programming ends.) Each integer z_i is set adaptively based on the actual cell level before the i-th round starts.[2]

Definition 1 *Let $\mathcal{U}(\theta, x, i)$ denote the minimum real number that satisfies the following constraint: there is a programming strategy that can guarantee to shift the cell level from x into the range*

$$[\theta, \mathcal{U}(\theta, x, i))$$

using at most i rounds of programming.

$\mathcal{U}(\theta, x, i)$ is a monotonic function of θ: $\forall\ \theta_1 < \theta_2$, $\mathcal{U}(\theta_1, x, i) \leq \mathcal{U}(\theta_2, x, i)$. Therefore, to maximize the storage capacity, which is equivalent to maximizing ℓ, the cell-level intervals should be set as follows:

$$a_1 = \Delta(1 - \epsilon);\ \forall\ i \in [\ell - 1] \setminus \{1\},\ a_i = \mathcal{U}(a_{i-1}, 0, r).$$

Since the minimum aimed increase of the cell level is Δ, it is simple to see that $\pi_2 = [\Delta(1 - \epsilon), \Delta(1 + \delta))$.

Example 1 *Let $r = 1$. Consider the i-th cell-level interval $\pi_i = [a_{i-1}, a_i) = [a_{i-1}, \mathcal{U}(a_{i-1}, 0, 1))$, where $i \in [\ell - 1] \setminus \{1\}$. To shift the cell level from 0 into the range $[a_{i-1}, \mathcal{U}(a_{i-1}, 0, 1))$ with just one round of programming, the aimed increase of the cell level $j\Delta$ should satisfy the conditions*

$$j\Delta(1 - \epsilon) \geq a_{i-1},\ j\Delta(1 + \delta) \leq \mathcal{U}(a_{i-1}, 0, 1).$$

The minimum value for j is $\lceil \frac{a_{i-1}}{\Delta(1-\epsilon)} \rceil$. So

$$\mathcal{U}(a_{i-1}, 0, 1) = \lceil \frac{a_{i-1}}{\Delta(1 - \epsilon)} \rceil \Delta(1 + \delta).$$

Define $b_1, b_2, \cdots, b_{\ell-2}$ as $b_1 = 1$, $b_2 = \lceil \frac{1+\delta}{1-\epsilon} \rceil$, $b_3 = \lceil \lceil \frac{1+\delta}{1-\epsilon} \rceil \frac{1+\delta}{1-\epsilon} \rceil$, $b_4 = \lceil \lceil \lceil \frac{1+\delta}{1-\epsilon} \rceil \frac{1+\delta}{1-\epsilon} \rceil \frac{1+\delta}{1-\epsilon} \rceil$, $\cdots$ Then, the cell-level intervals that maximize the storage capacity are:

$$\pi_1 = [0, \Delta(1 - \epsilon)),\ \pi_2 = [\Delta(1 - \epsilon), b_1\Delta(1 + \delta)),$$
$$\pi_i = [b_{i-2}\Delta(1 + \delta), b_{i-1}\Delta(1 + \delta)) \text{ for } i \in [\ell - 1] \setminus \{1, 2\},$$
$$\pi_\ell = [b_{\ell-2}\Delta(1 + \delta), \mathcal{A}].$$

[1] The inclusion and exclusion of the two boundary values are chosen for mathematical convenience, and are easy to deal with in practice. The same holds for the inclusion and exclusion of boundary values for other notations in the paper.

[2] As described before, a cell level cannot be greater than $\mathcal{A}$. For simplicity, we assume that the flash memory circuit can change the cell level to $\mathcal{A}$ with just one round of programming by using sufficiently strong charge injection. In the mathematical model of this paper, it means to set z_1 sufficiently large.

Knowing how to compute $\mathcal{U}(\theta, x, i)$ helps compute the storage capacity. That is the focus of the next section. It is also necessary to know how to program a cell given the cell-level intervals. A cell-programming algorithm is a function that, given the current cell level x, the target cell-level interval π_i and the number of remaining rounds of programming j, the aimed increase of the cell level in the next round of programming:

$$g : [0, \mathcal{A}] \times \{\pi_i | i \in [\ell]\} \times [r] \to \{0, \Delta, 2\Delta, \cdots\}.$$

Example 2 *Let $\mathcal{A} = 10$, $\Delta = 0.5$, $\epsilon = 0.3$, $\delta = 0.5$ and $r = 4$. A storage scheme is shown in Fig. 1(1) that divides $[0, \mathcal{A}]$ into $\ell = 12$ cell-level intervals. As an illustration, when the information symbol to store is 7 (namely, the target cell-level interval is $[3.75, 4.55)$), the cell-programming algorithm is shown in Fig. 1(2). For example, if the current cell level is in the region $[2.3, 3.05)$, the aimed cell-level increase in the next round of programming is 2Δ. (The cell-level region $(0, 2.1)$ is not shown in Fig. 1(2) because the cell level will not enter it.) It is noticeable that this programming algorithm does not depend on the number of programming rounds that remain. Both the storage scheme and the programming algorithm are derived based on the results to be presented next and, in fact, optimize the storage capacity.*

(1)

Cell level	[0,0.35)	[0.35,0.75)	[0.75,1.5)	[1.5,2.25)
Symbol	1	2	3	4
Cell level	[2.25,3)	[3,3.75)	[3.75,4.55)	[4.55,5.35)
Symbol	5	6	7	8
Cell level	[5.35,6.5)	[6.5,7.65)	[7.65,8.8)	[8.8,10]
Symbol	9	10	11	12

(2)

The symbol to store: 7				
Current cell level	0	[2.1,2.3)	[2.3,3.05)	[3.05, 3.75)
Aimed increase in the next round	6Δ	3Δ	2Δ	Δ

Figure 1: Storage in a flash-memory cell with $\mathcal{A} = 10$, $\Delta = 0.5$, $\epsilon = 0.3$, $\delta = 0.5$ and $r = 4$. (1) Mapping cell-level intervals to information symbols. (2) Programming algorithm.

3. Capacity of Single Flash Memory Cell

In this section, we show how to compute the storage capacity of a flash cell. First, we study the properties of the function $\mathcal{U}(\theta, x, i)$.

It is simple to see that when $x \geq \theta$, $\mathcal{U}(\theta, x, i) = x$. When $x < \theta$, we get

$$\mathcal{U}(\theta, x, 1) = x + \lceil \frac{\theta - x}{\Delta(1-\epsilon)} \rceil \cdot \Delta(1+\delta)$$

because to shift the cell level from x to some value above θ with just one programming round, the aimed increase in the cell level should be at least $\lceil \frac{\theta - x}{\Delta(1-\epsilon)} \rceil \cdot \Delta$. When $x < \theta$ and $i \geq 2$, we have the following recursion:

$$\mathcal{U}(\theta, x, i) =$$

$$\min_{j \in [\lceil \frac{\theta - x}{\Delta(1-\epsilon)} \rceil - 1]} \quad \max_{s \in [x + j\Delta(1-\epsilon), x + j\Delta(1+\delta))} \mathcal{U}(\theta, s, i-1).$$

This recursion holds because to change the cell level from x to some value above θ, the aimed cell-level increase in the next round can be set as $j\Delta$ with $j \in [\lceil \frac{\theta - x}{\Delta(1-\epsilon)} \rceil - 1]$; after the next round, the actual cell level (which is denoted by s in the recursion) can be any value in $[x + j\Delta(1-\epsilon), x + j\Delta(1+\delta))$, and $i-1$ more rounds can be used to program the cell.

The above recursive formula, however, cannot be directly used to compute $\mathcal{U}(\theta, x, i)$ effectively, because here s can take infinitely many different values. Therefore, more properties of $\mathcal{U}(\theta, x, i)$ need to be learned. It is not hard to show that $\mathcal{U}(\theta, x, i)$ is neither a monotonic nor a continuous function of x. However, it is piece-wise monotonic and continuous in x, as Lemma 1 below shows.

Definition 2 *For $j \in \mathbb{Z}$, define t_j as*

$$t_j = \theta - j\Delta(1-\epsilon).$$

$\forall\, x \in \mathbb{R}$, let x^- denote the limit:

$$\lim_{\varepsilon \to 0, \varepsilon > 0} x - \varepsilon.$$

Let $\mathcal{U}(\theta, x^-, i)$ denote the limit

$$\lim_{\varepsilon \to 0, \varepsilon > 0} \mathcal{U}(\theta, x - \varepsilon, i).$$

$\forall\, x, y \in \mathbb{R}$, we say "$x$ is above y" if $x \geq y$.

Lemma 1 *Let j be a non-negative integer. (1) $\forall\ x \in [t_{j+1}, t_j)$, $\mathcal{U}(\theta, x, i)$ is continuous and non-decreasing in x; (2) $\forall\ x \in [t_{j+1}, \theta)$, $\mathcal{U}(\theta, x, i) \leq \mathcal{U}(\theta, t_{j+1}^-, i)$.*

Proof: First, consider the case $i = 1$. $\forall\ x \in [t_{j+1}, t_j)$, $\mathcal{U}(\theta, x, 1) = x + \lceil \frac{\theta - x}{\Delta(1-\epsilon)} \rceil \cdot \Delta(1+\delta) = x + (j+1)\Delta(1+\delta)$, which is continuous and non-decreasing in x. $\forall\ x \in [t_{j+1}, \theta)$, without loss of generality (WLOG), we can assume $x \in [t_{k+1}, t_k)$ for some $0 \leq k \leq j$. Then, $\mathcal{U}(\theta, x, 1) = x + (k+1)\Delta(1+\delta) = t_{k+1} + (k+1)\Delta(1-\epsilon) + (k+1)\Delta(\epsilon+\delta) + (x - t_{k+1}) = \theta + (k+1)\Delta(\epsilon+\delta) + (x - t_{k+1}) \leq \theta + (j+1)\Delta(\epsilon+\delta) + \Delta(1-\epsilon) \leq \theta + (j+1)\Delta(\epsilon+\delta) + \Delta(1+\delta) = t_{j+1}^- + (j+2)\Delta(1+\delta) = \mathcal{U}(\theta, t_{j+1}^-, 1)$. So both properties are true when $i = 1$.

In the following, let us assume $i \geq 2$.

The proof is by induction on j. When $j = 0$ and $x \in [t_1, t_0)$, $\mathcal{U}(\theta, x, i) = x + \Delta(1+\delta)$, so Property (1) is true. When $j = 0$ and $x \in [t_1, \theta)$, $\mathcal{U}(\theta, x, i) = x + \Delta(1+\delta) \le \theta^- + \Delta(1+\delta) = t_0^- + \Delta(1+\delta) = \mathcal{U}(\theta, t_0^-, 1) \le \max_{s \in [t_0^-, t_0^- + \Delta(\epsilon+\delta))} \mathcal{U}(\theta, s, i-1) = \mathcal{U}(\theta, t_1^-, i)$, so Property (2) is also true. This serves as the base case of the induction. Now consider the inductive step.

We first consider Property (1). To shift the cell level from x to some value above θ, the aimed increase of the cell level in the next round will be $k\Delta$ for some $k \in [\lceil \frac{\theta - x}{\Delta(1-\epsilon)} \rceil - 1] = [j]$. Define $S_k(x)$ as $S_k(x) = [x + k\Delta(1-\epsilon), x + k\Delta(1+\delta))$, and define $f_k(x)$ as $f_k(x) = \max_{s \in S_k(x)} \mathcal{U}(\theta, s, i-1)$. Clearly, $\mathcal{U}(\theta, x, i) = \min_{k \in [j]} f_k(x)$.

The range $[t_{j+1}, t_j)$ can be split into three subregions $[t_{j+1}, z_1), [z_1, z_2), [z_2, t_j)$ such that: when $x \in [t_{j+1}, z_1)$, $S_k(x) \subseteq [t_{j+1-k}, t_{j-k})$; when $x \in [z_1, z_2)$, $t_{j-k} \in S_k(x)$, and $\mathcal{U}(\theta, t_{j-k}^-, i-1) > x + k\Delta(1+\delta)$; when $x \in [z_2, t_j)$, $t_{j-k} \in S_k(x)$, but $\mathcal{U}(\theta, t_{j-k}^-, i-1) \le x + k\Delta(1+\delta)$. (The first and the third subregions might be empty.) Then, when $x \in [t_{j+1}, z_1)$, by the induction assumption, $f_k(x) = \max_{s \in S_k(x)} \mathcal{U}(\theta, s, i-1) = \mathcal{U}(\theta, (x + k\Delta(1+\delta))^-, i-1)$ is continuous and non-decreasing in x. When $x \in [z_1, z_2)$, by the induction assumption, $f_k(x) = \max_{s \in S_k(x)} \mathcal{U}(\theta, s, i-1) = \mathcal{U}(\theta, t_{j-k}^-, i-1)$ remains constant as x increases. When $x \in [z_2, t_j)$, $f_k(x) = \max_{s \in S_k(x)} \mathcal{U}(\theta, s, i-1) = x + k\Delta(1+\delta)$ is continuous and non-decreasing in x. So it is not hard to see that $f_k(x)$ is continuous and non-decreasing in x for $x \in [t_{j+1}, t_j)$.

Since $\mathcal{U}(\theta, x, i) = \min_{k \in [j]} f_k(x)$ is the lower closure of j continuous and non-decreasing functions of x, $\mathcal{U}(\theta, x, i)$ is continuous and non-decreasing in x. So Property (1) is proved.

We now consider Property (2). Let k (here $k \le j$) denote the integer such that $\mathcal{U}(\theta, t_{j+1}^-, i) = \max_{s \in [t_{j+1}^- + k\Delta(1-\epsilon), t_{j+1}^- + k\Delta(1+\delta))} \mathcal{U}(\theta, s, i-1)$. We get $\mathcal{U}(\theta, t_{j+1}^-, i) = \max_{s \in [t_{j+1-k}^-, t_{j+1-k}^- + k\Delta(\epsilon+\delta))} \mathcal{U}(\theta, s, i-1) \ge \max_{s \in [t_{j-(k-1)}^-, t_{j-(k-1)}^- + (k-1)\Delta(\epsilon+\delta))} \mathcal{U}(\theta, s, i-1) \ge \mathcal{U}(\theta, t_j^-, i)$. By the induction assumption, $\mathcal{U}(\theta, x, i)$ is non-decreasing for $x \in [t_{j+1}, t_j)$, and $\mathcal{U}(\theta, t_j^-, i) \ge \mathcal{U}(\theta, x, i)$ for $x \in [t_j, \theta)$. So $\mathcal{U}(\theta, t_{j+1}^-, i) \ge \mathcal{U}(\theta, t_j^-, i) \ge \mathcal{U}(\theta, x, i)$ for $x \in [t_{j+1}, \theta)$. So Property (2) is proved. □

Lemma 2 *Suppose $x < y < \theta$, and suppose there exists an integer $j > 0$ such that*

$$y \in [x + j\Delta(1-\epsilon), x + j\Delta(1+\delta)).$$

Then it holds that

$$\mathcal{U}(\theta, x, i) \ge \mathcal{U}(\theta, y, i).$$

Furthermore, if $j \ge \lfloor \frac{\mathcal{U}(\theta, x, i) - x}{\Delta(1+\delta)} \rfloor$, then

$$\mathcal{U}(\theta, x, i) \ge \mathcal{U}(\theta, y, i-1).$$

Proof: Let S denote a programming algorithm that guarantees to shift the cell level from x into the range $[\theta, \mathcal{U}(\theta, x, i))$ by using at most i rounds of programming. With the algorithm S, if y is one of the possible values of the cell level after k rounds for some integer k, it is easy to see that the conclusions in this lemma are true. In the following, we assume that the cell level cannot be y after any number of rounds of programming with the algorithm S.

Let $z \in [x, y)$ be the number satisfying the following three properties: (1) With the algorithm S, z is one of the possible values of the cell level after a rounds, for some $a \in \{0, 1, \cdots, i-1\}$; (2) For some integer $b \ge 1$, $y \in [z + b\Delta(1-\epsilon), z + b\Delta(1+\delta))$; (3) Among all the numbers that satisfy the previous two properties, z is the greatest. (Note that z must exist because x itself satisfies the first two properties.)

Let c be the integer such that if the cell level is z after a rounds of programming, the algorithm S will set the aimed increase of the cell level to be $c\Delta$ in the next round. It is easy to see that $c > b$, because of the definition of z and the assumption that the cell level cannot be changed from x to y after any number of programming rounds with the algorithm S. So we get $\mathcal{U}(\theta, x, i) \ge \mathcal{U}(\theta, z, i-a) = \max_{s \in [z + c\Delta(1-\epsilon), z + c\Delta(1+\delta))} \mathcal{U}(\theta, s, i-a-1) \ge \max_{s \in [y + (c-b)\Delta(1-\epsilon), y + (c-b)\Delta(1+\delta))} \mathcal{U}(\theta, s, i-a-1) \ge \mathcal{U}(\theta, y, i-a) \ge \mathcal{U}(\theta, y, i)$.

If $j \ge \lfloor \frac{\mathcal{U}(\theta, x, i) - x}{\Delta(1+\delta)} \rfloor$, then $z > x$ (which means that $a \ge 1$). That is because if $z = x$, since $c > b = j$ here, it is possible for the cell level to be greater than or equal to $\mathcal{U}(\theta, x, i)$ after the first round of programming, which is a clear contradiction to the definition of algorithm S. Then by having $a \ge 1$ in the above analysis, we get $\mathcal{U}(\theta, x, i) \ge \mathcal{U}(\theta, y, i-1)$. □

Let us call a programming algorithm *optimal* if, when it is used to change a cell level from x to be above θ with i programming rounds, it guarantees that the final cell level will be less than $\mathcal{U}(\theta, x, i)$. The following theorem is useful for cell programming when the value of $\mathcal{U}(\theta, x, i)$ is known.

Theorem 1 *To change the cell level from x to be above θ with i rounds of programming, after each round, if the actual cell level is y and $y < \theta$ (naturally, $y \ge x$), an optimal algorithm for the next round is to set the aimed cell-level increase to be*

$$\lfloor \frac{\mathcal{U}(\theta, x, i) - y}{\Delta(1+\delta)} \rfloor \cdot \Delta.$$

Proof: Let η be an integer such that it is possible for the cell level to change from x to y after η rounds of programming using an optimal programming algorithm. Then

$\mathcal{U}(\theta, y, i-\eta) \le \mathcal{U}(\theta, x, i)$. Let $b = \lfloor \frac{\mathcal{U}(\theta,x,i)-y}{\Delta(1+\delta)} \rfloor \ge \lfloor \frac{\mathcal{U}(\theta,y,i-\eta)-y}{\Delta(1+\delta)} \rfloor$. Let $S = [y + b\Delta(1-\epsilon), y + b\Delta(1+\delta))$. $\forall\ s \in S$, if $s < \theta$, by Lemma 2, $\mathcal{U}(\theta, s, i-\eta-1) \le \mathcal{U}(\theta, y, i-\eta) \le \mathcal{U}(\theta, x, i)$; if $s \ge \theta, \mathcal{U}(\theta, s, i-\eta-1) = s < y + b\Delta(1+\delta) \le \mathcal{U}(\theta, x, i)$. So $\max_{s\in S} \mathcal{U}(\theta, s, i-\eta-1) \le \mathcal{U}(\theta, y, i-\eta) \le \mathcal{U}(\theta, x, i)$. So it is optimal to choose $b\Delta$ as the aimed increase of the cell level in the next round. □

When $x < \theta$, define τ as the non-negative integer such that

$$x \in [t_{\tau+1}, t_\tau).$$

In the following, we present a polynomial-time algorithm that computes $\mathcal{U}(\theta, x, i)$. We first show how to compute $\mathcal{U}(\theta, t_j^-, i)$, for $j \in \{0, 1, \cdots, \tau\}$. The algorithm is recursive. Its time complexity is $O(i\tau^2)$.

Algorithm 1 *Compute $\mathcal{U}(\theta, t_j^-, i)$ by using the following two functions. (Here $j \in \{0, 1, \cdots, \tau\}$.) First,*

$$\mathcal{U}(\theta, t_j^-, 1) = t_j + (j+1)\Delta(1+\delta).$$

Second, if $i \ge 2$, then

$$\mathcal{U}(\theta, t_j^-, i) = \min_{k\in[j]} \max\{\mathcal{U}(\theta, t_{j-k}^-, i-1), t_j + k\Delta(1+\delta)\}.$$

Theorem 2 *Algorithm* 1 *correctly computes $\mathcal{U}(\theta, t_j^-, i)$.*

Proof: Consider the case $i \ge 2$. To change the cell level from t_j^- to be above θ, the aimed cell-level increase in the first round of programming can be $\Delta, 2\Delta, \cdots, j\Delta$. (There is no need to make the aimed increase be $(j+1)\Delta$ in an optimal algorithm.) When the aimed increase is $k\Delta$, after the first round, the cell level falls in the range $S_k = [t_j^- + k\Delta(1-\epsilon), t_j + k\Delta(1+\delta)) = [t_{j-k}^-, t_j + k\Delta(1+\delta))$. By Lemma 1, we see that $\max_{s\in S_k} \mathcal{U}(\theta, s, i-1) = \max\{\mathcal{U}(\theta, t_{j-k}^-, i-1), t_j + k\Delta(1+\delta)\}$. Since $\mathcal{U}(\theta, t_j^-, i) = \min_{k\in[j]} \max_{s\in S_k} \mathcal{U}(\theta, s, i-1)$, the lemma holds. □

We now show how to compute $\mathcal{U}(\theta, x, i)$, with $x < \theta$ and $x \in [t_{\tau+1}, t_\tau)$. Given the values of $\mathcal{U}(\theta, t_j^-, i)$'s, the following algorithm has time complexity $O(\tau)$.

Algorithm 2 *Compute $\mathcal{U}(\theta, x, i)$ in the following way. (Here $x < \theta$ and $x \in [t_{\tau+1}, t_\tau)$).*

(1) Let $B = x + (\tau+1)\Delta(1+\delta)$. If $i = 1$, then $\mathcal{U}(\theta, x, i) = B$.

(2) Let b denote the smallest integer such that $x + b\Delta(1+\delta) > \theta$. If $b = \tau+1$, then $\mathcal{U}(\theta, x, i) = B$; otherwise, go to the next step.

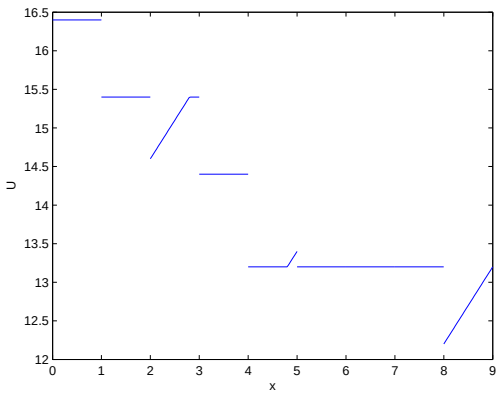

Figure 2: The horizontal axis is x, and the vertical axis is $\mathcal{U}(\theta, x, i)$. Here $\theta = 9$, $i = 4$, $\Delta = 2$, $\epsilon = 0.5$, $\delta = 1.1$.

(3) For $j \in \{b, b+1, \cdots, \tau\}$, Let c_j be the greatest integer such that

$$t_{c_j}^- \in [x + j\Delta(1-\epsilon), x + j\Delta(1+\delta)).$$

Let

$$C = \min_{j\in\{b,b+1,\cdots,\tau\}} \max\{\mathcal{U}(\theta, t_{c_j}^-, i-1), x + j\Delta(1+\delta)\}.$$

Then, $\mathcal{U}(\theta, x, i) = \min\{B, C\}$.

Theorem 3 *Algorithm* 2 *correctly computes $\mathcal{U}(\theta, x, i)$.*

Proof: Consider the programming algorithm that changes the cell level from x to be above θ. Let $a\Delta$ denote the aimed cell-level increase in the first round of programming using an optimal programming algorithm. By Theorem 1, a can be $\lfloor \frac{\mathcal{U}(\theta,x,i)-x}{\Delta(1+\delta)} \rfloor$, in which case we will have $x + a\Delta(1+\delta) > \theta$. (Otherwise, the cell level would be at most $(x + a\Delta(1+\delta))^- < \theta$ after the first round. Then at least one more round of programming would be needed to change the cell level to be above θ, and it would be possible for the cell level to be above $\mathcal{U}(\theta, x, i)$ after the second round, which would make the programming algorithm invalid.) So if at least two rounds are used, $a \in \{b, b+1, \cdots, \tau\}$; if only one round is used, $a = \tau+1$. The rest of the proof is similar to that of Theorem 2. □

An example of the function $\mathcal{U}(\theta, x, i)$ is shown in Fig. 2. We can see that it has the properties presented in Lemma 1 and Lemma 2.

Having known how to compute the function $\mathcal{U}(\theta, x, i)$, we can use the method presented in Section2 to find the maximum value ℓ such that the cell-level range $[0, \mathcal{A}]$ can be partitioned into ℓ intervals and mapped to ℓ information

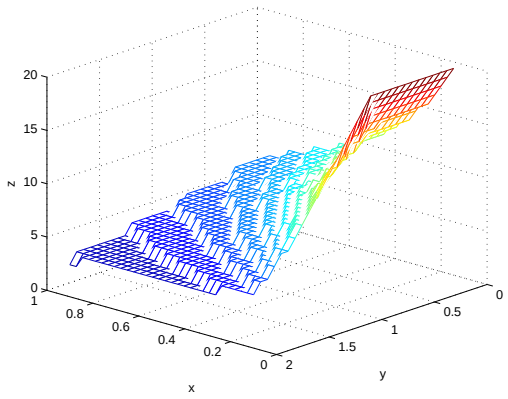

Figure 3: Storage capacity of a single flash cell. The x-axis is ϵ, the y-axis is δ, the z-axis is the maximum value of ℓ. Here the parameters are $\mathcal{A} = 10$, $\Delta = 0.5$, and the maximum number of rounds of cell-programming is $r = 5$.

symbols. This gives the storage capacity of a flash cell. An example of the storage capacity is illustrated in Fig. 3.

The following theorem considers the storage capacity when an arbitrarily large number of programming rounds can be used.

Theorem 4 *When r is sufficiently large, $\ell \geq \lceil \frac{\mathcal{A}}{\Delta(1+\delta)} \rceil + 1$.*

Proof: Suppose r is sufficiently large. To change the cell level from 0 to be above a positive number θ, we can set Δ as the aimed cell-level increase in every round of programming. This way, the final cell level is smaller than $\theta + \Delta(1+\delta)$. With this programming method, the width of each cell-level interval is less than $\Delta(1+\delta)$. In addition, the total width of π_1 and π_2 is $\Delta(1+\delta)$. So $\ell \geq \lceil \frac{\mathcal{A}}{\Delta(1+\delta)} \rceil + 1$.

4. Cell Programming Algorithm

In this section, we present an algorithm that optimally programs a cell, given the optimal set of cell-level intervals $\pi_1, \pi_2, \cdots, \pi_\ell$. Here $\pi_1 = [0, a_1)$, $\pi_i = [a_{i-1}, \mathcal{U}(a_{i-1}, 0, r))$ for $i \in [\ell - 1] \setminus \{1\}$, $\pi_\ell = [a_{\ell-1}, \mathcal{A}]$.

Algorithm 3 *$\forall\, i \in [\ell]$, the following algorithm shifts the cell level from 0 into π_i using at most r rounds of programming.*

(1) If $i = 1$, since the initial cell level, zero, is in $\pi_1 = [0, a_1)$, no programming is necessary.

(2) If $i \in [\ell - 1] \setminus \{1\}$, keep programming with the following approach until the cell level is in $\pi_i = [a_{i-1}, \mathcal{U}(a_{i-1}, 0, r))$: given the current cell level $x < a_{i-1}$, set the aimed cell-level increase in the next round of programming as

$$\lfloor \frac{\mathcal{U}(a_{i-1}, 0, r) - x}{\Delta(1+\delta)} \rfloor \Delta.$$

(3) If $i = \ell$, by our previous assumption, one round of programming that uses sufficiently strong charge injection can shift the cell level into π_ℓ.

Based on the previous analysis, it is easy to verify the correctness of the above programming algorithm.

5. Concluding Remarks

In this paper, we study the storage capacity of a flash memory cell under a specific cell-programming model. A method for computing the capacity is presented, and an optimal cell-programming algorithm is derived. As the future work, we are interested in fully characterizing the capacity of cell ensembles, and in considering errors caused by various disturb mechanisms and the retention problem [2].

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 *Proceedings of the IEEE International Symposium on Circuits and Systems*, 2005, pp. 2148–2151.

[2] P. Cappelletti, C. Golla, P. Olivo and E. Zanoni (*Ed.*), *Flash Memories*, Kluwer Academic Publishers, 1st Edition, 1999.

[3] M. Grossi, M. Lanzoni and B. Ricco, "Program Schemes for Multilevel Flash Memories," in *Proceedings of the IEEE*, vol. 91, no. 4, 2003, pp. 594–601.

[4] H. N. et al., "A 144-Mb, Eight-level NAND Flash Memory with Optimized Pulsewidth Programming," *IEEE J. Solid-State Circuits*, vol. 35, no. 5, pp. 682–690, 2000.

[5] A. Jiang, V. Bohossian and J. Bruck, "Floating Codes for Joint Information Storage in Write Asymmetric Memories," in *Proceedings of the IEEE International Symposium on Information Theory*, 2007, pp. 1166–1170.