On the Capacity and Programming of Flash Memories

Anxiao (Andrew) Jiang, *Member, IEEE*, Hao Li, and Jehoshua Bruck, *Fellow, IEEE*

***Abstract*—Flash memories are currently the most widely used type of nonvolatile memories. A flash memory consists of floating-gate cells as its storage elements, where the charge level stored in a cell is used to represent data. Compared to magnetic recording and optical recording, flash memories have the unique property that the cells are programmed using an iterative procedure that monotonically shifts each cell's charge level upward toward its target value. In this paper, we model the cell as a monotonic storage channel, and explore its capacity and optimal programming. We present two optimal programming algorithms based on a few different noise models and optimization objectives.**

***Index Terms*—Capacity, data storage, flash memory.**

I. Introduction

THERE are two unique properties that distinguish flash memories from traditional communication and storage channels: *iterative programming* and *monotonic change of cell state*. To write data into a flash memory cell, which is its basic storage element, the cell is programmed *iteratively* via multiple rounds of programming, until its state reaches the target value. That is different from communication channels, where the transmitted signals are no longer refined. It is also different from magnetic or optical recording, where data are typically written into the magnetic or optical disk in just one round of programming. The more the rounds of programming are used, the better the flash memory cell state can be controlled. On the other side, when a flash memory cell is programmed, its state—which corresponds to the amount of charge stored in the cell—can only change *monotonically* toward higher charge levels. These two properties make flash memories a special storage channel. It is interesting to study the capacity of the channel, and see how to program it for optimal performance.

We briefly review the flash memory technology for better understanding of its properties. Flash memories are currently the most widely used type of nonvolatile electronic memories (NVMs) due to their physical endurance and high performance [3]. In a flash memory, floating-gate cells are organized as blocks, usually 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 (e.g., electrons) 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. In current products, both single-level cells (SLCs), where $q = 2$, and multilevel cells (MLCs), where $q > 2$ (e.g., $q = 4$ or 8), are used. To increase the storage capacity, MLCs with more levels are being actively developed.

Flash memories have a prominent feature for their 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 must 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 increment in the cell level usually differs from the targeted increment. So, the overinjection of charge into cells is a major concern for programming, because the only way to correct overinjection is to erase the block and reprogram it. The cost of block erasure/reprogramming is prohibitively high [8], since it is not only very time consuming and energy consuming, but also decreases the lifetime of flash memories. (A flash memory block can only endure 10^4 to 10^5 erasures.) So 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], [7], [16]. Initially, the cell is at the base level. Then 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 overshooting is sufficiently small. The cell level shifts upward monotonically with each round, until it becomes close enough to the target value. An illustration of the cell-programming process is shown in Fig. 1.

The flash memory cell presents an interesting and new channel model, which we shall call monotonic storage channel. The model has several important elements. As a storage channel, the cell can be repeatedly programmed to make its level approach a target value. However, its level can only increase, which makes an overshooting error uncorrectable. In each round of programming, the actual increment of the cell level often differs from the targeted increment due to the noisy programming process. Since the cell level can be measured after every round of programming, which is feedback information, the programming algorithm should be adaptive. That is, how to set the targeted increment for the cell level in the $(i+1)$th round should depend on the cell level measured after the ith round. Every round of programming has a cost, which, for flash memories, is the time delay used to program a cell and measure

Manuscript received November 15, 2009; revised October 16, 2011; accepted October 18, 2011. Date of current version February 29, 2012. The material in this paper was presented in part at the International Symposium on Information Theory and Its Applications, Auckland, New Zealand, December 2008, and at the IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing, Victoria, BC, Canada, August 2009. This work was supported in part by NSF CAREER Award CCF-0747415 and in part by NSF Grant ECCS-0802107.

A. (A.) Jiang is with the Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843-3112 USA (e-mail: ajiang@cse.tamu.edu).

H. Li was with the Department of Computer Science and Engineering, Texas A&M University, College Station, TX 77843-3112 USA. He is now with Google, Inc., Kirkland, WA 98033 USA (e-mail: haoli@google.com).

J. Bruck is with the Department of Electrical Engineering, California Institute of Technology, Pasadena, CA 91125 USA (e-mail: bruck@caltech.edu).

Communicated by U. Mitra, Associate Editor At Large.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIT.2011.2177755

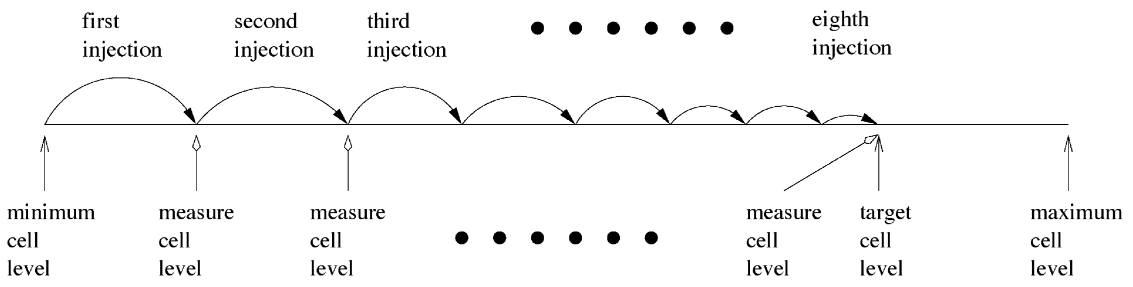

Fig. 1. Programming a cell using multiple rounds of charge injection and cell-level measurement.

its level. The programming procedure ends when either the number of rounds reaches its maximum allowed value, or when the cell level reaches the target level. The fundamental problem for monotonic storage channels is to find the optimal tradeoff between the programming precision—namely, the difference between the final cell level and the target level—and the total programming cost.

In flash memories, the time delay for writing is usually bounded to ensure high write speed. In this paper, we assume the number of rounds of charge injection to be given, and optimize the programming precision. The key is to find an algorithm that achieves the best performance by adaptively selecting programming actions based on the changing cell level. We present two optimal programming algorithms based on a few different programming noise models and different measures of the programming precision. The optimal programming precision is closely related to the storage capacity of the flash memory cells.

The rest of this paper is organized as follows. In Section II, we give an overview of related work. In Section III, we study the zero-error capacity, and present an optimal cell-programming algorithm that achieves it. In Section IV, we present a cell-programming algorithm that optimizes the expected programming precision. In Section V, we show the conclusions.

II. Overview of Related Work

The monotonic property of flash memories is caused by the high cost of block erasures [3]. It has motivated the study of *rewriting codes* for flash memories, where data can be modified by only increasing cell levels. A rewriting code builds a one-to-many mapping from the data to the cells' states, so that the data can be changed repeatedly without a block erasure. The rewriting codes include write-once-memory (WOM) codes for storing individual variables [4], [6], [17], floating codes for the joint coding of multiple variables [5], [8], [9], [21], buffer codes for buffering recent values in a data stream [2], and the rewriting codes in [11] that generalize the aforementioned data models. In these rewriting codes, every cell has q discrete levels, and the cell levels can only increase. It is assumed error free to change a cell from a lower discrete level to a higher discrete level. This is different from the monotonic storage channel model here, because here we focus on the programming noise, and study the programming algorithms that can minimize the impact of the programming noise.

A data representation scheme called rank modulation has been proposed in [13] and [14]. It assumes that the cell levels can only monotonically increase, and it uses the relative order of cell levels—instead of their absolute values—to represent data. This way, the danger of charge overinjection associated with fixed discrete cell levels can be reduced. Rewriting codes were also studied for the rank modulation scheme [13].

In addition to flash memories [10], the cell-programming problem for phase-change memories has also been studied in recent years [15]. Similar to flash memories, the phase-change memory cells can also be programmed multiple times to achieve higher accuracy. Thus, in [15], the authors appropriately named them rewritable storage channels. However, unlike flash memories, phase-change memories can change cell levels in both directions, and do not have block erasures. So, the channel does not have the monotonic property, and the programming algorithm can assume that the cell always starts with the same initial state in each round [15]. In comparison, in this study, we focus on the monotonic storage channel, where the cell level changes monotonically and requires an adaptive programming method.

The Z-channel is a well-known asymmetric channel, where the noise changes symbols in a monotonic direction [18], [19]. Error-correcting codes for correcting such asymmetric errors have been studied [20]. Compared to the Z-channel, the noise in the monotonic storage channel model appears during the cell-programming process instead of after it (i.e., after the data are written).

III. Zero-Error Capacity and Optimal Programming

In this section, we study the zero-error capacity of flash memory cells, and present an optimal programming algorithm.

A. Cell-Programming Model

The values that a flash memory cell's level can take can be seen as a continuous range $[0, \mathcal{L})$, where the maximum cell level $\mathcal{L}$ is determined by the physics of flash memories [3]. Normally, this range is divided into ℓ regions:

$$[0, a_1),\ [a_1, a_2), \ldots, [a_{\ell-2}, a_{\ell-1}),\ [a_{\ell-1}, \mathcal{L})$$

where the cell levels in each region represent a data symbol.[1] Let $a_0 = 0$ and $a_\ell = \mathcal{L}$. A cell with ℓ regions can represent a symbol from an alphabet $\{1, 2, \ldots, \ell\}$, and therefore can store up to $log_2 \ell$ bits.

[1]The inclusion and exclusion of the two boundary values of a region 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.

We assume that the initial level of a cell is 0. To write a symbol $i \in \{1, 2, \ldots, \ell\}$ into a cell, multiple rounds of charge injection will be used to shift the cell level from 0 monotonically into the region $[a_{i-1}, a_i)$. In this section, we measure the cost of programming by the number of rounds of charge injection that are used. That is, every round of charge injection has cost one, which in practice represents the average time delay associated with the charge injection and the following measurement of the actual cell level [3]. We assume that at most r rounds of charge injection can be used to program a cell, and are interested in the storage capacity given this cost constraint.

In this section, we focus on zero-error programming: given that a cell can be programmed using at most r rounds of charge injection, we want to make sure that for $i = 1, 2, \ldots, \ell$, the symbol i can be written into the cell with guaranteed success. Since charge injection is a noisy process, this implies that ℓ must be a finite number, and the ℓ regions—$[0, a_1)$, $[a_1, a_2), \ldots, [a_{\ell-2}, a_{\ell-1})$, $[a_{\ell-1}, \mathcal{L})$—must also be appropriately chosen. Given this zero-error programming requirement, we call the maximum value of $\log_2 \ell$ that can be achieved the *zero-error capacity* of the flash memory cells.

We assume that the flash memory circuit has a programming resolution Δ. In each round of charge injection, the circuit aims to increase a cell's level by $i\Delta$, for some integer i. This targeted cell-level increment $i\Delta$ is controlled by the discrete adjustment of the programming voltage or current [3]. Due to the programming noise, the actual increment of the cell level will be close to, but different from, $i\Delta$.

It is not easy to accurately model the programming noise. The distribution of the noise varies among cells [3]. In this paper, we assume that when the targeted cell-level increment is $i\Delta$, the actual increment of the cell level is randomly distributed in the range

$$[i\Delta(1-\epsilon),\ i\Delta(1+\delta))$$

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

To find the zero-error capacity, we need to solve the following optimization problem. Its input parameters are: 1) $\mathcal{L}$, the maximum cell level; 2) r, the maximum number of rounds of programming; 3) Δ, the programming resolution; 4) ϵ and δ, the noise parameters. The output includes: 1) the maximum value that ℓ can take for zero-error programming; 2) how to optimally divide the cell-level range $[0,\ \mathcal{L})$ into the intervals $[0, a_1)$, $[a_1, a_2), \ldots, [a_{\ell-2}, a_{\ell-1})$, $[a_{\ell-1}, \mathcal{L})$ so that zero-error programming can be realized.

For $i = 1, 2, \ldots, r$, let $z_i\Delta$ denote the targeted cell-level increment of the ith round of programming, where $z_i \in \{0, 1, 2 \ldots\}$. To obtain the optimal cell-programming algorithm, we need to know how to choose $z_1, z_2, \ldots, z_r$ adaptively. The general problem is as follows. Suppose the target cell-level interval is $[a, a')$. (For example, to write the symbol $i \in \{1, 2, \ldots, \ell\}$, we have $[a, a') = [a_{i-1}, a_i)$.) And suppose the cell has been programmed for $r - j$ rounds—which means that there are still j rounds of programming left—and its current level is x. We need to find an integer $F(a, a', x, j) \in \{0, 1, 2, \ldots\}$ such that when we set the targeted cell-level increment of the next round to be $F(a, a', x, j)\Delta$, no matter what the actual cell-level increment will be inside the range $[F(a, a', x, j)\Delta(1-\epsilon),\ F(a, a', x, j)\Delta(1+\delta))$, there will still be a way to program the cell with the remaining $j-1$ rounds such that the final cell level will enter the target interval $[a, a')$. (It can be seen that the programming noise here can be considered as an adversary for the cell-programming algorithm.)

(a)

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

(b)

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

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

Let us show how to implement the cell-programming algorithm efficiently in flash memories. Instead of having the memory, compute $F(a_{i-1}, a_i, x, j)$ (for $1 \le i \le \ell$, $0 \le x < a_{i-1}$ and $1 \le j \le r$) during the write operation—which will slow down the write speed—and it will be better to store the values of $F(a_{i-1}, a_i, x, j)$ in a table for quick lookups. More specifically, the table should be small, which means that we should partition the range $[0, a_{i-1})$ into a small number of subranges for x, so that when x is in a subrange, regardless of its specific value, the value of $F(a_{i-1}, a_i, x, j)$ will be the same. This way, the cell-programming algorithm can be carried out at high speed. We illustrate it in the following example, where the data are computed based on the optimal cell-programming algorithm we will present. And our results will show that this approach is always feasible.

Example 1: Let $\mathcal{L} = 10$, $\Delta = 0.5$, $\epsilon = 0.3$, $\delta = 0.5$, and $r = 4$. A storage scheme is shown in Fig. 2(a) that divides $[0, \mathcal{L})$ 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. 2(b). For example, if the current cell level is in the region $[2.3, 3.05)$, the targeted cell-level increment in the next round of programming is 2Δ. (The cell-level region $(0, 2.1)$ is not shown in Fig. 2(b) 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. (This is due to the result that the optimal algorithm can take a greedy cell-programming strategy, which will be shown in Section III-C.) 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. □

B. Optimal Setting of Cell-Level Intervals

Consider how to set the cell-level intervals $[0,a_1),[a_1,a_2),\dots,[a_{\ell-2},a_{\ell-1}),[a_{\ell-1},\mathcal{L})$, so that ℓ can be maximized. Clearly, we can let $a_1=\Delta(1-\epsilon)$ and set the first interval as $[0,a_1)=[0,\Delta(1-\epsilon))$, because once a cell is programmed, its level will be at least $\Delta(1-\epsilon)$. Then, the optimal approach is to determine the values $a_2,a_3,\dots,a_{\ell-1}$ sequentially. Specifically, suppose that the value of a_i has been determined. To determine a_{i+1}, we make it as small as possible as long as there is a guarantee that a cell can be changed from level 0 to the interval $[a_i,a_{i+1})$ with at most r rounds of charge injection. This way, the size of the interval $[a_i,a_{i+1})$ is minimized. The value of ℓ can be determined this way: if the value of a_{i+1} found with the aforementioned approach is no less than $\mathcal{L}$, then we let $\ell=i+1$ and let $a_\ell=\mathcal{L}$.

The following notation, $\mathcal{U}(\theta,x,i)$, is useful based on the aforementioned analysis.

Definition 2: Let $\theta\in[0,\mathcal{L})$ and $x\in[0,\mathcal{L})$ be two cell levels, and let i be a positive integer. Let $\mathcal{U}(\theta,x,i)$ denote the minimum real number that satisfies the following constraint: there exists 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 charge injection. □

$\mathcal{U}(\theta,x,i)$ is a monotonic function of θ:

$$\forall\,\theta_1<\theta_2,\ \text{we have}\ \mathcal{U}(\theta_1,x,i)\le\mathcal{U}(\theta_2,x,i).$$

To maximize the storage capacity, which is equivalent to maximizing ℓ, the cell-level intervals should be set as follows:

$$\begin{aligned}a_1&=\Delta(1-\epsilon)\\ \forall\, i=2,3,\dots,\ell-1,\ a_i&=\mathcal{U}(a_{i-1},0,r).\end{aligned}$$

As the minimum targeted increment of the cell level is Δ, it is easy to see that the interval $[a_1,\ a_2)=[\Delta(1-\epsilon),\ \Delta(1+\delta))$.

Example 3: Let $r=1$—namely, only one round of charge injection can be used to program a cell—and consider how to set the cell-level intervals. Consider the ith cell-level interval

$$[a_{i-1},\ a_i)=[a_{i-1},\ \mathcal{U}(a_{i-1},0,1))$$

where $i\in\{2,3,\dots,\ell-1\}$. 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 charge injection, the targeted increment of the cell level $j\Delta$ should satisfy the two conditions

$$\begin{aligned}j\Delta(1-\epsilon)&\ge a_{i-1}\\ j\Delta(1+\delta)&\le\mathcal{U}(a_{i-1},0,1).\end{aligned}$$

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

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

Define $b_1,b_2,\dots,b_{\ell-2}$ as

$$\begin{aligned}b_1&=1\\ b_2&=\lceil\frac{1+\delta}{1-\epsilon}\rceil\\ b_3&=\lceil\lceil\frac{1+\delta}{1-\epsilon}\rceil\frac{1+\delta}{1-\epsilon}\rceil\\ b_4&=\lceil\lceil\lceil\frac{1+\delta}{1-\epsilon}\rceil\frac{1+\delta}{1-\epsilon}\rceil\frac{1+\delta}{1-\epsilon}\rceil.\\ &\cdots\end{aligned}$$

Then, the cell-level intervals that maximize the storage capacity are

$$\begin{aligned}&[0,\ a_1)=[0,\ \Delta(1-\epsilon))\\ &[a_1,\ a_2)=[\Delta(1-\epsilon),\ b_1\Delta(1+\delta))\\ &[a_{i-1},\ a_i)=[b_{i-2}\Delta(1+\delta),\ b_{i-1}\Delta(1+\delta))\\ &\text{for } i\in\{3,4,\dots,\ell-1\},\ \text{and}\\ &[a_{\ell-1},\ \mathcal{L})=[b_{\ell-2}\Delta(1+\delta),\ \mathcal{L}).\end{aligned}$$

□

For convenience, in the rest of this paper, for any positive integer k, let us define $[k]=\{1,2,\dots,k\}$.

C. Properties of the Function $\mathcal{U}(\theta,x,i)$

We have shown that to compute the zero-error capacity, the key is to compute the function $\mathcal{U}(\theta,x,i)$. In this section, we will present a recursive function useful for computing $\mathcal{U}(\theta,x,i)$. It is necessary to find properties of the function $\mathcal{U}(\theta,x,i)$ that can make the computation efficient. We will show that $\mathcal{U}(\theta,x,i)$ has three special properties.

1) It is piecewise continuous and nondecreasing in x. And the maximum value that $\mathcal{U}(\theta,x,i)$ attains in such a piece is also greater than or equal to its value in all the subsequent pieces.
2) For two cell levels x and y with $x<y$, if it is possible to shift the cell level from x to y through charge injection, then $\mathcal{U}(\theta,x,i)\ge\mathcal{U}(\theta,y,i)$. Namely, the value of $\mathcal{U}(\theta,x,i)$ decreases for a chain of cell levels as they get closer to the threshold θ, if it is possible to obtain the chain of cell levels through a sequence of charge injection.
3) For an optimal cell-programming algorithm, the targeted cell-level increment for each round of charge injection can be set in a greedy way by making the charge injection as large as possible, as long as it is guaranteed that the cell level after the charge injection will not exceed the target cell-level interval.

Those three properties will be used to derive the zero-error capacity and the optimal cell-programming algorithm.

We first derive the recursion for $\mathcal{U}(\theta,x,i)$. It is easy to see that when $x\ge\theta,\mathcal{U}(\theta,x,i)=x$.

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 round of charge injection, the targeted increment of 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:

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

This recursion holds because to change the cell level from x to some value above θ, the targeted cell-level increment of 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 aforementioned 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 in x. However, it is piecewise monotonic and continuous in x, as Lemma 5 shows.

Let us first divide the range of cell levels into pieces of length $\Delta(1-\epsilon)$, and use t_j, for $j \in \mathbb{Z}$, to denote the boundary values of those pieces.

Definition 4: For every $j \in \mathbb{Z}$, define t_j as

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

(Note that, here, we consider θ as a given constant.) □

For any $x, y \in \mathbb{R}$, if $x \geq y$, we will say "x is above y." We now present the piecewise property of $\mathcal{U}(\theta,x,i)$.

Lemma 5: Let j be a nonnegative integer. Then, we have the following.

1) $\forall\, x \in [t_{j+1}, t_j)$, the function $\mathcal{U}(\theta,x,i)$ is continuous and nondecreasing in x;
2) $\forall\, x \in [t_{j+1}, \theta)$,

$$\mathcal{U}(\theta,x,i) \leq \lim_{\nu \to 0^+} \mathcal{U}(\theta, t_{j+1}-\nu, i).$$

Proof: First, consider the case $i = 1$. For any $x \in [t_{j+1}, t_j)$, we have $\mathcal{U}(\theta,x,1) = x + \lceil \frac{\theta-x}{\Delta(1-\epsilon)} \rceil \cdot \Delta(1+\delta) = x + (j+1)\Delta(1+\delta)$, which is continuous and nondecreasing in x. For any $x \in [t_{j+1}, \theta)$, without loss of generality, we can assume $x \in [t_{k+1}, t_k)$ for some $0 \leq k \leq j$. Then, we have

$$\begin{aligned}
&\mathcal{U}(\theta,x,1)\\
&= x + (k+1)\Delta(1+\delta)\\
&= t_{k+1} + (k+1)\Delta(1-\epsilon) + (k+1)\Delta(\epsilon+\delta) + (x - t_{k+1})\\
&= \theta + (k+1)\Delta(\epsilon+\delta) + (x - t_{k+1})\\
&\leq \theta + (j+1)\Delta(\epsilon+\delta) + \Delta(1-\epsilon)\\
&\leq \theta + (j+1)\Delta(\epsilon+\delta) + \Delta(1+\delta)\\
&= \lim_{\nu\to 0^+}(t_{j+1}-\nu) + (j+2)\Delta(1+\delta)\\
&= \lim_{\nu\to 0^+}\mathcal{U}(\theta, t_{j+1}-\nu, 1).
\end{aligned}$$

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 $x \in [t_1, \theta)$, we have

$$\begin{aligned}
&\mathcal{U}(\theta,x,i)\\
&= x + \Delta(1+\delta)\\
&\leq \lim_{\nu\to 0^+}(\theta-\nu) + \Delta(1+\delta)\\
&= \lim_{\nu\to 0^+}(t_0-\nu) + \Delta(1+\delta)\\
&= \lim_{\nu\to 0^+}\mathcal{U}(\theta, t_0-\nu, 1)\\
&\leq \lim_{\nu\to 0^+}\max_{s\in[t_0-\nu, t_0-\nu+\Delta(\epsilon+\delta))}\mathcal{U}(\theta,s,i-1)\\
&= \lim_{\nu\to 0^+}\mathcal{U}(\theta, t_1-\nu, i).
\end{aligned}$$

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 targeted increment 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:

1) when $x \in [t_{j+1}, z_1)$, we have

$$S_k(x) \subseteq [t_{j+1-k}, t_{j-k});$$

2) when $x \in [z_1, z_2)$, we have $t_{j-k} \in S_k(x)$ and

$$\lim_{\nu\to 0^+}\mathcal{U}(\theta, t_{j-k}-\nu, i-1) > x + k\Delta(1+\delta);$$

3) when $x \in [z_2, t_j)$, we have $t_{j-k} \in S_k(x)$, but

$$\lim_{\nu\to 0^+}\mathcal{U}(\theta, t_{j-k}-\nu, i-1) \leq x + k\Delta(1+\delta).$$

The first or the third subregion above might be empty. Then, when $x \in [t_{j+1}, z_1)$, by the induction assumption

$$\begin{aligned}
f_k(x) &= \max_{s\in S_k(x)}\mathcal{U}(\theta,s,i-1)\\
&= \lim_{\nu\to 0^+}\mathcal{U}(\theta, (x+k\Delta(1+\delta))-\nu, i-1)
\end{aligned}$$

is continuous and nondecreasing in x. When $x \in [z_1, z_2)$, by the induction assumption, $f_k(x) = \max_{s \in S_k(x)} \mathcal{U}(\theta, s, i-1) = \lim_{\nu \to 0^+} \mathcal{U}(\theta, t_{j-k} - \nu, i-1)$ 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 nondecreasing in x. So, it is not hard to see that $f_k(x)$ is continuous and nondecreasing 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 nondecreasing functions of $x, \mathcal{U}(\theta, x, i)$ is continuous and nondecreasing in x. So, Property (1) is proved.

We now consider Property (2). Let k (here $k \le j$) denote the integer such that $\lim_{\nu \to 0^+} \mathcal{U}(\theta, t_{j+1} - \nu, i) =$

$$\lim_{\nu \to 0^+} \max_{s \in [t_{j+1} - \nu + k\Delta(1-\epsilon), t_{j+1} - \nu + k\Delta(1+\delta))} \mathcal{U}(\theta, s, i-1).$$

We get

$$\begin{aligned} &\lim_{\nu \to 0^+} \mathcal{U}(\theta, t_{j+1} - \nu, i) \\ &= \lim_{\nu \to 0^+} \max_{s \in [t_{j+1-k} - \nu, t_{j+1-k} - \nu + k\Delta(\epsilon+\delta))} \mathcal{U}(\theta, s, i-1) \\ &\ge \lim_{\nu \to 0^+} \max_{s \in [t_{j-(k-1)} - \nu, t_{j-(k-1)} - \nu + (k-1)\Delta(\epsilon+\delta))} \\ &\quad \mathcal{U}(\theta, s, i-1) \\ &\ge \lim_{\nu \to 0^+} \mathcal{U}(\theta, t_j - \nu, i). \end{aligned}$$

By the induction assumption, $\mathcal{U}(\theta, x, i)$ is nondecreasing for $x \in [t_{j+1}, t_j)$, and $\lim_{\nu \to 0^+} \mathcal{U}(\theta, t_j - \nu, i) \ge \mathcal{U}(\theta, x, i)$ for $x \in [t_j, \theta)$. So, for $x \in [t_{j+1}, \theta)$

$$\lim_{\nu \to 0^+} \mathcal{U}(\theta, t_{j+1} - \nu, i) \ge \lim_{\nu \to 0^+} \mathcal{U}(\theta, t_j - \nu, i) \ge \mathcal{U}(\theta, x, i)$$

So, Property (2) is proved. ■

The next lemma shows that for a sequence of cell levels that can be sequentially obtained through charge injection, the value of the function $\mathcal{U}(\theta, x, i)$ keeps decreasing. (In the lemma, x and y denote two such cell levels in the sequence.)

Lemma 6: 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 charge injection. With the algorithm S, if y is one of the possible values of the cell level after k rounds of charge injection 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 charge injection 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 $a \in \{0, 1, \ldots, i-1\}$.
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 charge injection, the algorithm S will set the targeted increment 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 rounds of charge injection with the algorithm S. So, we get

$$\begin{aligned} &\mathcal{U}(\theta, x, i) \\ &\ge \mathcal{U}(\theta, z, i-a) \\ &= \max\nolimits_{s \in [z + c\Delta(1-\epsilon), z + c\Delta(1+\delta))} \mathcal{U}(\theta, s, i-a-1) \\ &\ge \max\nolimits_{s \in [y + (c-b)\Delta(1-\epsilon), y + (c-b)\Delta(1+\delta))} \mathcal{U}(\theta, s, i-a-1) \\ &\ge \mathcal{U}(\theta, y, i-a) \\ &\ge \mathcal{U}(\theta, y, i). \end{aligned}$$

If $j \ge \lfloor \frac{\mathcal{U}(\theta,x,i) - x}{\Delta(1+\delta)} \rfloor$, then $z > x$ (which means that $a \ge 1$). This 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 charge injection, which is a clear contradiction to the definition of algorithm S. Then by having $a \ge 1$ in the aforementioned analysis, we get $\mathcal{U}(\theta, x, i) \ge \mathcal{U}(\theta, y, i-1)$. ■

We now show that once it is known how to compute the function $\mathcal{U}(\theta, x, i)$, the cell-programming problem can be simplified substantially. Let us call a cell-programming algorithm *optimal* if when it is used to change a cell level from x to any value above θ with i rounds of charge injection, the algorithm guarantees that the final cell level—the cell level after the i rounds of charge injection—will be less than $\mathcal{U}(\theta, x, i)$. The following theorem shows that if the value of $\mathcal{U}(\theta, x, i)$ is known, an optimal cell-programming algorithm can use a greedy approach: in each round of charge injection, it can make the targeted cell-level increment as large as possible, as long as it is guaranteed that the cell level after the charge injection will not exceed $\mathcal{U}(\theta, x, i)$.

Theorem 7: Suppose that our goal is to shift the cell level from x to any value above θ with i rounds of charge injection. Then, a cell-programming algorithm that takes the following greedy approach will be optimal: in each round of charge injection, if the cell level before this charge injection—which we denote by y—is less than θ (so we have $x \le y < \theta$), the algorithm sets the targeted cell-level increment for this round 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 charge injection using an optimal programming algorithm. Then

$$\mathcal{U}(\theta, y, i-\eta) \le \mathcal{U}(\theta, x, i).$$

Let

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

Let

$$D = [y + b\Delta(1-\epsilon), y + b\Delta(1+\delta))$$

$\forall\, s \in D$, if $s < \theta$, by Lemma 6

$$\mathcal{U}(\theta, s, i-\eta-1) \leq \mathcal{U}(\theta, y, i-\eta) \leq \mathcal{U}(\theta, x, i);$$

if $s \geq \theta$,

$$\mathcal{U}(\theta, s, i-\eta-1) = s < y + b\Delta(1+\delta) \leq \mathcal{U}(\theta, x, i).$$

So

$$\max_{s \in D} \mathcal{U}(\theta, s, i-\eta-1) \leq \mathcal{U}(\theta, y, i-\eta) \leq \mathcal{U}(\theta, x, i).$$

So, it is optimal to choose $b\Delta$ as the targeted cell-level increment in the next round. ■

D. Zero-Error Capacity of Flash Memory Cells

In this section, we will show how to compute the zero-error capacity of flash memory cells. First, let us show how to compute the value of $\mathcal{U}(\theta, x, i)$.

Let $x < \theta$ be a fixed number. Let τ denote the nonnegative integer such that

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

(Remember that by Lemma 5, the function $\mathcal{U}(\theta, x, i)$ is continuous and nondecreasing in x when x is in the range $[t_{\tau+1}, t_\tau)$.) In the following, we will present a polynomial-time algorithm that computes the value of $\mathcal{U}(\theta, x, i)$. We first show how to compute $\lim_{\nu \to 0^+} \mathcal{U}(\theta, t_j - \nu, i)$, for $j \in \{0, 1, \ldots, \tau\}$. The algorithm is recursive. Its time complexity is $O(i\tau^2)$.

Algorithm 8: Compute $\lim_{\nu \to 0^+} \mathcal{U}(\theta, t_j - \nu, i)$ by using the following two functions. (Here $j \in \{0, 1, \ldots, \tau\}$.) First,

$$\lim_{\nu \to 0^+} \mathcal{U}(\theta, t_j - \nu, 1) = t_j + (j+1)\Delta(1+\delta).$$

Second, if $i \geq 2$, then

$$\lim_{\nu \to 0^+} \mathcal{U}(\theta, t_j - \nu, i) = \min_{k \in [j]} \max\{(\lim_{\nu \to 0^+} \mathcal{U}(\theta, t_{j-k} - \nu, i-1)),\ t_j + k\Delta(1+\delta)\}.$$

□

Theorem 9: Algorithm 8 correctly computes

$$\lim_{\nu \to 0^+} \mathcal{U}(\theta, t_j - \nu, i).$$

Proof: Consider the case $i \geq 2$. Our goal is to shift the cell level from $\lim_{\nu \to 0^+} t_j - \nu$ to any value above θ. The targeted cell-level increment in the first round of charge injection can be $\Delta, 2\Delta, \ldots, j\Delta$. (There is no need to make the targeted cell-level increment be $(j+1)\Delta$ in an optimal algorithm.) When the targeted cell-level increment is $k\Delta$, after the first round, the cell level falls in the range $S_k =$

$$[(\lim_{\nu \to 0^+} t_j - \nu + k\Delta(1-\epsilon)),\ t_j + k\Delta(1+\delta)) = [(\lim_{\nu \to 0^+} t_{j-k} - \nu),\ t_j + k\Delta(1+\delta)).$$

By Lemma 5, we see that

$$\max_{s \in S_k} \mathcal{U}(\theta, s, i-1) = \max\{(\lim_{\nu \to 0^+} \mathcal{U}(\theta, t_{j-k} - \nu, i-1)),\ t_j + k\Delta(1+\delta)\}.$$

Since

$$\lim_{\nu \to 0^+} \mathcal{U}(\theta, t_j - \nu, i) = \min_{k \in [j]} \max_{s \in S_k} \mathcal{U}(\theta, s, i-1)$$

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 $\lim_{\nu \to 0^+} \mathcal{U}(\theta, t_j - \nu, i)$, the following algorithm has the time complexity $O(\tau)$.

Algorithm 10: 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.

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

$$(\lim_{\nu \to 0^+} t_{c_j} - \nu) \in [x + j\Delta(1-\epsilon), x + j\Delta(1+\delta)).$$

Let $C =$

$$\min_{j \in \{b, b+1, \ldots, \tau\}} \max\{(\lim_{\nu \to 0^+} \mathcal{U}(\theta, t_{c_j} - \nu, i-1)),\ x + j\Delta(1+\delta)\}$$

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

□

Theorem 11: Algorithm 10 correctly computes $\mathcal{U}(\theta, x, i)$.

Proof: Consider the programming algorithm that shifts the cell level from x to any value above θ. Let $a\Delta$ denote the targeted cell-level increment in the first round of charge injection using an optimal programming algorithm. By Theorem 7, a can be

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

in which case we will have

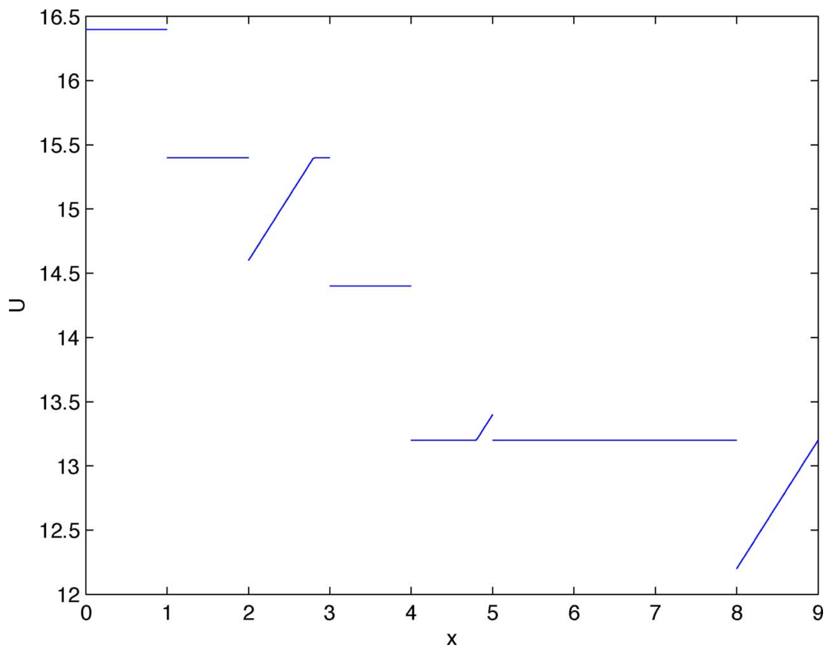

Fig. 3. Horizontal axis is x, and the vertical axis is $\mathcal{U}(\theta, x, i)$. Here, $\theta = 9, i = 4, \Delta = 2, \epsilon = 0.5, \delta = 1.1$.

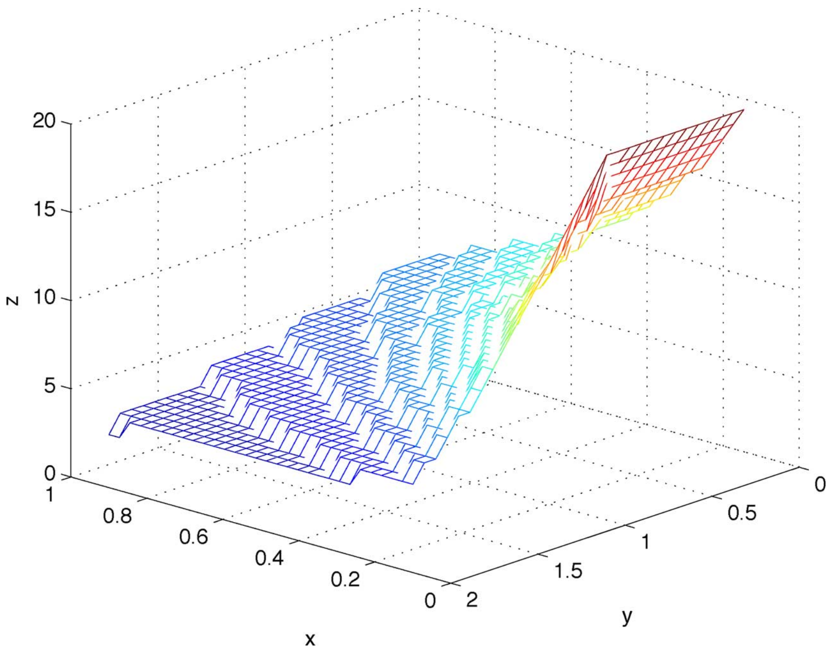

Fig. 4. x-axis is ϵ, the y-axis is δ, and the z-axis is the maximum value of ℓ for zero-error programming. Here, the parameters are $\mathcal{L} = 10, \Delta = 0.5$, and the maximum number of rounds of charge injection is $r = 5$.

$$x + a\Delta(1+\delta) > \theta.$$

(Otherwise, the cell level would be at most

$$\lim_{\nu \to 0^+} x + a\Delta(1+\delta) - \nu < \theta$$

after the first round. Then, at least one more round of programming would be needed to shift the cell level 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 of charge injection are used, $a \in \{b, b+1, \ldots, \tau\}$; if only one round is used, $a = \tau + 1$. The rest of the proof is similar to that of Theorem 9. ■

An example of the function $\mathcal{U}(\theta, x, i)$ is shown in Fig. 3. We can see that it has the properties presented in Lemmas 5 and 6 .

Having shown how to compute the function $\mathcal{U}(\theta, x, i)$, we can use the method presented in Section III-B to find the maximum value ℓ such that the cell-level range $[0, \mathcal{L})$ can be partitioned into ℓ intervals and mapped to ℓ information symbols. This gives us the zero-error capacity of flash memory cells. An example is illustrated in Fig. 4.

The following theorem considers the storage capacity when arbitrarily many rounds of charge injection can be used.

Theorem 12: When r is sufficiently large, $\ell \geq \lceil \frac{\mathcal{L}}{\Delta(1+\delta)} \rceil + 1$.

Proof: Suppose r is sufficiently large. To shift the cell level from 0 to above a positive number θ, we can set Δ as the targeted cell-level increment in every round of charge injection. This way, the final cell level will be smaller than $\theta + \Delta(1+\delta)$. With this programming method, the width of each cell-level interval mapped to a symbol is less than $\Delta(1+\delta)$. In addition, $a_2 = \Delta(1+\delta)$. So $\ell \geq \lceil \frac{\mathcal{L}}{\Delta(1+\delta)} \rceil + 1$. ■

E. Optimal Cell-Programming Algorithm

In this section, we present an algorithm that optimally programs a cell, given the optimal set of cell-level intervals

$$[0,\ a_1),\ [a_1,\ a_2), \ldots,\ [a_{\ell-2},\ ,a_{\ell-1}),\ [a_{\ell-1},\ \mathcal{L}).$$

Here, for $i = 2, 3, \ldots, \ell-1$, $a_i = \mathcal{U}(a_{i-1}, 0, r)$.

Algorithm 13: For any given $i \in [\ell]$, the following algorithm shifts the cell level from 0 into the interval $[a_{i-1},\ a_i)$ using at most r rounds of charge injection.

1) If $i = 1$, since the initial cell level—zero—is in $[0, a_1)$, no charge injection is necessary.
2) If $i \in \{2, 3, \ldots, \ell-1\}$, keep programming the cell with the following approach until the cell level is in the interval $[a_{i-1}, \mathcal{U}(a_{i-1}, 0, r))$: given the current cell level $x < a_{i-1}$, set the targeted cell-level increment in the next round of charge injection as
$$\lfloor \frac{\mathcal{U}(a_{i-1}, 0, r) - x}{\Delta(1+\delta)} \rfloor \Delta.$$
3) If $i = \ell$, use one round of charge injection to shift the cell level from 0 into the interval $[a_{\ell-1},\ \mathcal{L})$, where the targeted cell-level increment of this round is $\lceil \frac{a_{\ell-1}}{\Delta(1-\epsilon)} \rceil \cdot \Delta$. □

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

IV. Programming for Optimal Expected Precision

In this section, we study cell programming with optimal expected precision. The motivation is that to study the capacity of cell ensembles, which depends on the statistical precision of programming, it is necessary to understand how accurately the cells can be programmed. In contrast to the study in the previous section, here we need to consider the probability distribution of the programming noise.

We continue to use the programming model of the previous section. That is, when the targeted cell-level increment is $i\Delta$ during a round of charge injection, the actual cell-level increment will be randomly distributed in the range $[i\Delta(1-\epsilon),\ i\Delta(1+\delta))$. For simplicity, in this section, we assume that the actual cell-level increment has a uniform random distribution in this range. More practical programming-noise models can be studied in the future. And as before, $\mathcal{L}$ denotes the maximum cell level, r denotes the maximum number of rounds of charge injection that can be used to program a cell, and the initial level of the cell is zero.

Our objective is to program a cell to a target level $\theta \in [0, \mathcal{L})$, using at most r rounds of charge injection. Due to the programming noise, the final cell level will deviate from θ. There is a cost $C(x)$ associated with the final cell level x, whose value clearly should increase when x deviates further away from θ. In this paper, we consider two families of cost functions.

Definition 14 (Cost function $C(x)$ for MLC and Rank Modulation): In the MLC technology, the final cell level should be close to one of a set of discrete levels. It is appropriate to define $C(x)$ as

$$C(x) = |x - \theta|^p$$

for some positive integer p.

In the rank modulation technology [12]–[14], the objective of programming a cell is to shift the cell level *above* a certain value θ. It is appropriate to define $C(x)$ as

$$C(x) = \begin{cases} \infty & \text{, if } x < \theta \\ (x-\theta)^p & \text{, if } x \geq \theta \end{cases}$$

for some positive integer p. □

Let $z_1\Delta, z_2\Delta, \ldots, z_r\Delta$ denote the targeted cell-level increments in the r rounds of charge injection, and let $x_1, x_2, \ldots, x_r$ denote the actual cell level after each round. For $j = 1, 2, \ldots, r-1$, after the jth round of charge injection, the flash memory can measure x_j, and adaptively choose the targeted cell-level increment $z_{j+1}\Delta$ for the next round. The objective of the cell-programming problem is to find the adaptive strategy of selecting $z_1, z_2, \ldots, z_r$, such that the expected cost of the final cell level

$$E(C(x_r))$$

is minimized. (Here, $E(x)$ is the expectation of the random variable x.) In the following, we present an optimal programming algorithm.

A. Optimal Strategy for Adaptive Cell Programming

The cell-programming algorithm requires an adaptive strategy for selecting the targeted cell-level increments $z_1\Delta, z_2\Delta, \ldots, z_r\Delta$ for the r rounds of charge injection. To characterize the adaptive strategy, let us define two functions $\mathcal{A}(x; i)$ and $\alpha(x; i; j)$.

Definition 15 (Functions $\mathcal{A}(x; i)$ and $\alpha(x; i; j)$):

Let x be a real number, and let i, j be integers.

$\mathcal{A}(x; i)$ is defined to be the minimum achievable value of the expected cost of the final cell level, given the following.

1) The current cell level is $\theta + x$.
2) We can program the cell with i more rounds of charge injection.

$\alpha(x; i; j)$ is defined to be the minimum achievable value of the expected cost of the final cell level, given the following.

1) The current cell level is $\theta + x$.
2) We can program the cell with i more rounds of charge injection.
3) In the first round of the remaining i rounds of charge injection, we will choose the targeted cell-level increment to be $j\Delta$.

□

It is simple to see that

$$\mathcal{A}(x;i)=\min_{j=0,1,2\cdots}\alpha(x;i;j).$$

For the cell-programming problem, since the initial cell level is $0=\theta+(-\theta)$ and r rounds of charge injection can be used, the objective is to find a strategy that makes the final cell level's expected cost reach its minimum value

$$\mathcal{A}(-\theta;r).$$

During the programming process, given that the cell level is x_j after $j<r$ rounds of charge injection, the flash memory should adaptively choose $i_{j+1}\Delta$ as the targeted cell-level increment of the $(j+1)$th round, such that i_{j+1} minimizes the value of $\alpha(x_j-\theta;\ r-j;\ i_{j+1})$.

B. Computing the Initial Value of $\mathcal{A}(x;i)$

The cost function we consider is for MLC or rank modulation, which is shown in Definition 14. As we have seen, to find the optimal cell-programming algorithm, it is helpful to know how to compute $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$. In this section, we first compute the initial values of $\mathcal{A}(x;i)$—in particular, the values of $\mathcal{A}(x;1)$—corresponding to the two cost functions.

1) When the Cost Function Is for MLC: The cost function for MLC is $C(x)=|x-\theta|^p$. For simplicity, we show how to compute $\mathcal{A}(x;1)$ when $p=2$. The other values of p can be dealt with similarly. Note that when there is just one round of charge injection, the only decision to make is to choose the targeted cell-level increment of that round, which we denote by $j\Delta$ in the following equations.

When $p=2$, the minimum expected cost of the final cell level is

$$\begin{aligned}&\mathcal{A}(x;1)\\ &=\min\nolimits_{j=0,1,2\cdots}\quad \alpha(x;1;j)\\ &=\min\{\quad \alpha(x;1;0)\\ &\quad\min\nolimits_{j=1,2,3\cdots}\quad \int_{\theta+x+j\Delta(1-\epsilon)}^{\theta+x+j\Delta(1+\delta)}\frac{1}{j\Delta(\epsilon+\delta)}\cdot|y-\theta|^2dy\}\\ &=\min\{\quad x^2,\\ &\quad\min\nolimits_{j=1,2,3\cdots}\quad \frac{1}{j\Delta(\epsilon+\delta)}\int_{x+j\Delta(1-\epsilon)}^{x+j\Delta(1+\delta)}y^2dy\}\\ &=\min\nolimits_{j=0,1,2\cdots}\quad x^2+j\Delta(2+\delta-\epsilon)x+\frac{1}{3}j^2\Delta^2(3+\\ &\qquad 3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2).\end{aligned}$$

To see which value of j minimizes the aforementioned equation, define

$$f(j)=x^2+j\Delta(2+\delta-\epsilon)x+\frac{1}{3}j^2\Delta^2(3+3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2).$$

Since $0<\epsilon<1$ and $\delta>0$, we have

$$3+3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2>0.$$

So, $f(j)$ is convex. By setting $\frac{df(j)}{dj}=0$, we find that $f(j)$ is minimized when

$$j=\frac{-3x(2+\delta-\epsilon)}{2\Delta(3+3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2)}$$

(assuming that j does not have to be an integer). We can see that the aforementioned value for j is positive if and only if x is negative. Since j actually needs to be a nonnegative integer, we find that to minimize $f(j)$, j should take the following value of j^*:

$$j^*=\begin{cases}\lceil\frac{-3(2+\delta-\epsilon)x}{2\Delta(3+3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2)}-0.5\rceil & \text{, if } x<0\\ 0 & \text{, if } x\ge 0.\end{cases}$$

We can now compute the value of $\mathcal{A}(x;1)$ as follows. Let $\gamma=\frac{2\Delta(3+3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2)}{3(2+\delta-\epsilon)}$. Then, when $x<0$, $j^*=\lceil\frac{-x}{\gamma}-0.5\rceil$. So, for $i=1,2,\ldots,\lceil\frac{\theta}{\gamma}-1.5\rceil$, when

$$x\in[-(i+0.5)\gamma,-(i-0.5)\gamma)$$

we have $j^*=i$ and

$$\begin{aligned}\mathcal{A}(x;1)=&\,x^2+i\Delta(2+\delta-\epsilon)x\\ &+\tfrac{1}{3}i^2\Delta^2(3+3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2).\end{aligned}$$

Similarly, when $x\in[-0.5\gamma,0)$, we have $j^*=0$ and

$$\mathcal{A}(x;1)=x^2.$$

When $x\in[-\theta,-(\lceil\frac{\theta}{\gamma}-0.5\rceil-0.5)\gamma)$, we have $j^*=\lceil\frac{\theta}{\gamma}-0.5\rceil$ and

$$\begin{aligned}\mathcal{A}(x;1)=&\,x^2+\lceil\tfrac{\theta}{\gamma}-0.5\rceil\Delta(2+\delta-\epsilon)x+\\ &\tfrac{1}{3}\lceil\tfrac{\theta}{\gamma}-0.5\rceil^2\Delta^2(3+3\delta-3\epsilon+\delta^2-\delta\epsilon+\epsilon^2).\end{aligned}$$

When $x\ge 0$, we have $j^*=0$ and

$$\mathcal{A}(x;1)=x^2.$$

Therefore, we can partition the domain of x, $[-\theta,\infty)$, into $\lceil\frac{\theta}{\gamma}+0.5\rceil$ regions, while in each region $\mathcal{A}(x;1)$ is a degree-2 polynomial. So, $\mathcal{A}(x;1)$ is *piecewise polynomial.*

It is not hard to see that when $p\neq 2$, $\mathcal{A}(x;1)$ is also piecewise polynomial. For simplicity, we skip the details.

2) When the Cost Function Is for Rank Modulation: The cost function for rank modulation is $C(x)=\infty$ if $x<\theta$ and $C(x)=(x-\theta)^p$ if $x\ge\theta$. For simplicity, we show how to compute $\mathcal{A}(x;1)$ when $p=1$. The other values of p can be dealt with similarly.

We have $\mathcal{A}(x;1)=\min_{j=0,1,2\ldots}\alpha(x;1;j)$. The value of j that minimizes $\alpha(x;1;j)$ is the minimum integer that satisfies the constraint $\theta+x+j\Delta(1-\epsilon)\ge\theta$, which is $j=\lceil\frac{-x}{\Delta(1-\epsilon)}\rceil$. So, if $x<0$, we have

$$\begin{aligned}\mathcal{A}(x;1)&=\int_{\theta+x+\lceil\frac{-x}{\Delta(1-\epsilon)}\rceil\Delta(1-\epsilon)}^{\theta+x+\lceil\frac{-x}{\Delta(1-\epsilon)}\rceil\Delta(1+\delta)}\frac{1}{\lceil\frac{-x}{\Delta(1-\epsilon)}\rceil\Delta(\epsilon+\delta)}\cdot(y-\theta)dy\\ &=x+\lceil\tfrac{-x}{\Delta(1-\epsilon)}\rceil\Delta(1+\tfrac{\delta-\epsilon}{2}).\end{aligned}$$

If $x \geq 0$, clearly $\mathcal{A}(x;1) = x$.

So, for $i = 1,2,\ldots,\lceil\frac{\theta}{\Delta(1-\epsilon)}\rceil - 1$, when $x \in [-i\Delta(1-\epsilon), -(i-1)\Delta(1-\epsilon))$, we get

$$\mathcal{A}(x;1) = x + i\Delta(1 + \frac{\delta-\epsilon}{2}).$$

When $x \in [-\theta, -(\lceil\frac{\theta}{\Delta(1-\epsilon)}\rceil - 1)\Delta(1-\epsilon))$, we get

$$\mathcal{A}(x;1) = x + \lceil\frac{\theta}{\Delta(1-\epsilon)}\rceil\Delta(1 + \frac{\delta-\epsilon}{2}).$$

When $x \geq 0$, we get

$$\mathcal{A}(x;1) = x.$$

So, we can partition the domain of x, $[-\theta,\infty)$, into $\lceil\frac{\theta}{\Delta(1-\epsilon)}\rceil + 1$ regions, while in each region, $\mathcal{A}(x;1)$ is a linear function. So, $\mathcal{A}(x;1)$ is *piecewise polynomial*.

It is not hard to see that when $p \neq 1$, $\mathcal{A}(x;1)$ is also piecewise polynomial. For simplicity, we skip the details.

C. Computing the Functions $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$

In this section, we show how to compute the functions $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$, where $i \geq 2$. We first present a recursive relation for the two functions, which is useful for their computation. To make the computation efficient, we will use the property that both $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$ are piecewise polynomial in x, for any positive integer i. The results will be used to derive the optimal cell-programming algorithm later.

When $i \geq 2$, we have the equation

$$\mathcal{A}(x;i) = \min_{j=0,1,2\cdots} \alpha(x;i;j)$$

And since the programming noise is assumed to be uniformly distributed over a range of size $j\Delta(\delta+\epsilon)$ when the targeted cell-level increment of the next round of charge injection is $j\Delta$, we have the recursion

$$\alpha(x;i;j) = \int_{x+j\Delta(1-\epsilon)}^{x+j\Delta(1+\delta)} \frac{\mathcal{A}(y;i-1)}{j\Delta(\delta+\epsilon)} dy$$

for $j \geq 1$. (By default, we have $\alpha(x;i;0) = \mathcal{A}(x;i-1)$.)

To use the aforementioned recursion to effectively compute the values of $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$, we need to know more properties of the two functions. In this section, we use the property that they are both piecewise polynomial. (Note that this property of being piecewise polynomial has been proved for $\mathcal{A}(x;1)$. It will be shown that it holds for $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$ with $i \geq 2$, too.)

For convenience of expression, let us first define some notations. Given integers i,j, let $p_{i,j}$ be the integer such that the function $\alpha(x;i;j)$ is a polynomial in x in each of $p_{i,j}+1$ subdomains of x. And let

$$b_{i,j}(-1)\ ,\ b_{i,j}(0)\ ,\ b_{i,j}(1)\ ,\ \cdots\ ,\ b_{i,j}(p_{i,j})$$

denote the boundary values of those $p_{i,j}+1$ subdomains. More specifically, given integers i,j, let $p_{i,j}$ and $b_{i,j}(-1), b_{i,j}(0), b_{i,j}(1), \ldots, b_{i,j}(p_{i,j})$ be the numbers with the following properties.

1) $b_{i,j}(-1) > b_{i,j}(0) > b_{i,j}(1) > b_{i,j}(2) > \cdots > b_{i,j}(p_{i,j})$.
2) $b_{i,j}(-1) = \infty$, $b_{i,j}(0) = 0$, $b_{i,j}(p_{i,j}) = -\theta$.
3) for $k = 0,1,\ldots,p_{i,j}$, the function $\alpha(x;i;j)$ is a polynomial in x when $x \in [b_{i,j}(k), b_{i,j}(k-1))$.

We define similar notations for $\mathcal{A}(x;i)$. Given an integer $i \geq 1$, let q_i be the integer such that the function $\mathcal{A}(x;i)$ is a polynomial in x in each of q_i+1 subdomains of x. And let

$$B_i(-1)\ ,\ B_i(0)\ ,\ B_i(1)\ ,\ \cdots\ ,\ B_i(q_i)$$

denote the boundary values of those q_i+1 subdomains. More specifically, given an integer $i \geq 1$, let q_i and $B_i(-1), B_i(0), B_i(1), \ldots, B_i(q_i)$ be the numbers with the following properties.

1) $B_i(-1) > B_i(0) > B_i(1) > \cdots > B_i(q_i)$.
2) $B_i(-1) = \infty$, $B_i(0) = 0$, $B_i(q_i) = -\theta$.
3) For $k = 0,1,\ldots,q_i$, the function $\mathcal{A}(x;i)$ is a polynomial in x when $x \in [B_i(k), B_i(k-1))$.

In the following, we show how to compute $\alpha(x;i;j)$ and $\mathcal{A}(x;i)$ with $i \geq 2$, respectively.

1) Computing $\alpha(x;i;j)$ With $i \geq 2$: We first show how to compute $\alpha(x;i;j)$ with $i \geq 2$.

Given a real number $x \in [-\theta,\infty)$ and an integer $i \geq 1$, we call the unique integer $j \in \{0,1,\ldots,q_i\}$ such that

$$x \in [B_i(j)\ ,\ B_i(j-1))$$

the "*i-index of x*", and denote it by

$$index(i;x).$$

The i-index of x states which "polynomial piece" of the function $\mathcal{A}(x;i)$ the input variable x is in. Note that $index(i;x)$ decreases as x increases. Let us use $\mathcal{I}(i;x;j)$ to denote the set of (i)-indices of the real numbers in the interval

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

We get

$$\begin{aligned}\mathcal{I}(i;x;j) = \{\quad & index(i;x+j\Delta(1-\epsilon));\\ & index(i;x+j\Delta(1-\epsilon)) - 1;\\ & index(i;x+j\Delta(1-\epsilon)) - 2;\\ & \cdots\\ & \lim_{\nu\to 0^+} index(i;x+j\Delta(1+\delta)-\nu)\quad \}.\end{aligned}$$

The last element in the aforementioned set is a limit because the interval $[x+j\Delta(1-\epsilon)\ ,\ x+j\Delta(1+\delta))$ does not contain the boundary value $x+j\Delta(1+\delta)$.

Let us define the set $S_{i,j}$ (for $i \geq 2$) as

$$\begin{aligned}S_{i,j} = \{\quad & s \in (-\theta,0)|\text{ either } s = B_{i-1}(k) - j\Delta(1-\epsilon)\\ & \text{for some } 0 \leq k \leq q_{i-1}-1, \text{ or}\\ & s = B_{i-1}(k) - j\Delta(1+\delta)\\ & \text{for some } 0 \leq k \leq q_{i-1}-1\}.\end{aligned}$$

The next lemma shows that the numbers in $S_{i,j}$—which are cell levels—partition the domain of x into subdomains that have the following property: for two cells of two arbitrary cell levels x_1, x_2 in the same subdomain, if we program both of them with the same targeted cell-level increment $j\Delta$, the two sets of possible levels of the two cells after the charge injection will have the same set of $(i-1)$-indices.

Lemma 16: We denote the $|S_{i,j}|$ numbers in the set $S_{i,j}$ by

$$s_1, s_2, \ldots, s_{|S_{i,j}|}$$

such that $s_1 > s_2 > \cdots > s_{|S_{i,j}|}$. Also, let $s_0 = 0$ and $s_{|S_{i,j}|+1} = -\theta$. Then, for $k = 1, 2, \ldots, |S_{i,j}|+1$, for any two numbers x_1, x_2 in the interval (s_k, s_{k-1}),

$$\mathcal{I}(i-1; x_1; j) = \mathcal{I}(i-1; x_2; j)$$

.

Proof: Without loss of generality, assume that $x_1 < x_2$. We just need to prove that (1)

$$index(i-1; x_1+j\Delta(1-\epsilon)) = index(i-1; x_2+j\Delta(1-\epsilon))$$

and (2)

$$\begin{aligned}&\lim_{\nu\to 0^+} index(i-1; x_1 + j\Delta(1+\delta) - \nu)\\ &= \lim_{\nu\to 0^+} index(i-1; x_2 + j\Delta(1+\delta) - \nu).\end{aligned}$$

Let us prove condition (1) by contradiction. Assume that $index(i-1; x_1+j\Delta(1-\epsilon)) \neq index(i-1; x_2+j\Delta(1-\epsilon))$. Then, there must be some $B_{i-1}(k')$ such that

$$x_1 + j\Delta(1-\epsilon) < B_{i-1}(k') \leq x_2 + j\Delta(1-\epsilon).$$

So

$$x_1 < B_{i-1}(k') - j\Delta(1-\epsilon) \leq x_2.$$

Since

$$B_{i-1}(k') - j\Delta(1-\epsilon) \in S_{i,j} = \{s_1, s_2, \ldots, s_{|S_{i,j}|}\}$$

x_1 and x_2 cannot be in the same interval (s_k, s_{k-1}). That is a contradiction. So $index(i-1; x_1 + j\Delta(1-\epsilon)) = index(i-1; x_2 + j\Delta(1-\epsilon))$.

Condition (2) can be proved similarly. For simplicity, we skip the details. ■

The following theorem shows that the numbers in $S_{i,j}$ partition the domain of the cell level x into subdomains, such that in every subdomain, the function $\alpha(x;i;j)$ is a polynomial in x. Furthermore, it shows how to compute the algebraic expression of the function $\alpha(x;i;j)$ efficiently.

Theorem 17: We denote the $|S_{i,j}|$ numbers in the set $S_{i,j}$ by

$$s_1, s_2, \ldots, s_{|S_{i,j}|}$$

such that $s_1 > s_2 > \cdots > s_{|S_{i,j}|}$. Also, let $s_0 = 0$ and $s_{|S_{i,j}|+1} = -\theta$. Then, for $k = 1, 2, \ldots, |S_{i,j}|+1$, the function $\alpha(x;i;j)$ is a polynomial in x for $x \in (s_k, s_{k-1})$. Furthermore, it can be computed as follows. Let

$$u = \lim_{\nu\to 0^+} index(i-1; s_k + j\Delta(1-\epsilon) + \nu)$$

and let

$$v = \lim_{\nu\to 0^+} index(i-1; s_{k-1} + j\Delta(1+\delta) - \nu).$$

Then

$$\begin{aligned}\alpha(x;i;j) = &\textstyle\int_{x+j\Delta(1-\epsilon)}^{B_{i-1}(u-1)} \frac{\mathcal{A}(y;i-1)}{j\Delta(\epsilon+\delta)}\,dy +\\ &\textstyle\sum_{k=v+1}^{u-1} \int_{B_{i-1}(k)}^{B_{i-1}(k-1)} \frac{\mathcal{A}(y;i-1)}{j\Delta(\epsilon+\delta)}\,dy +\\ &\textstyle\int_{B_{i-1}(v)}^{x+j\Delta(1+\delta)} \frac{\mathcal{A}(y;i-1)}{j\Delta(\epsilon+\delta)}\,dy.\end{aligned}$$

Proof: We know that

$$\alpha(x;i;j) = \int_{x+j\Delta(1-\epsilon)}^{x+j\Delta(1+\delta)} \frac{\mathcal{A}(y;i-1)}{j\Delta(\epsilon+\delta)} dy.$$

Since $x \in (s_k, s_{k-1})$, by Lemma 16, we have

$$\begin{aligned}&index(i-1; x + j\Delta(1-\epsilon))\\ &= \lim_{\nu\to 0^+} index(i-1; s_k + j\Delta(1-\epsilon) + \nu)\\ &= u\end{aligned}$$

and

$$\begin{aligned}&\lim_{\nu\to 0^+} index(i-1; x + j\Delta(1+\delta) - \nu)\\ &= \lim_{\nu\to 0^+} index(i-1; s_{k-1} + j\Delta(1+\delta) - \nu)\\ &= v.\end{aligned}$$

So in the aforementioned integration, we can partition the domain of y into smaller intervals, in each of which the function $\mathcal{A}(y;i-1)$ is a polynomial in y. So, the way to compute $\alpha(x;i;j)$ in this theorem is correct.

$\mathcal{A}(y;i-1)$ is a polynomial in y for $y \in [B_{i-1}(u), B_{i-1}(u-1)) \supseteq [x + j\Delta(1-\epsilon), B_{i-1}(u-1))$ and for $y \in [B_{i-1}(v), B_{i-1}(v-1)) \supseteq [B_{i-1}(v), x + j\Delta(1+\delta))$. Also note that the value of $\sum_{k=v+1}^{u-1} \int_{B_{i-1}(k)}^{B_{i-1}(k-1)} \frac{\mathcal{A}(y;i-1)}{j\Delta(\epsilon+\delta)}\,dy$ is independent of $x \in (s_k, s_{k-1})$. Since polynomials are closed under integration and summation, we get that $\alpha(x;i;j)$ is a polynomial in x for $x \in (s_k, s_{k-1})$. ■

The aforementioned theorem shows that $\alpha(x;i;j)$ is an integration of $\mathcal{A}(x;i-1)$. It is easy to see that if $\mathcal{A}(x;i-1)$ is a piecewise polynomial function of degree d, then $\alpha(x;i;j)$ is a piecewise polynomial function of degree at most $d+1$. As we will see, $\mathcal{A}(x;i)$ is also a piecewise polynomial of degree at most $d+1$.

The following corollary is a refinement of Theorem 17. It shows that the piecewise-polynomial property of the function $\alpha(x;i;j)$ can be extended to include all the boundary values of the subdomains of x.

Corollary 18: We denote the $|S_{i,j}|$ numbers in the set $S_{i,j}$ by $s_1, s_2, \ldots, s_{|S_{i,j}|}$ such that $s_1 > s_2 > \cdots > s_{|S_{i,j}|}$. Also, let $s_{-1} = \infty$, $s_0 = 0$ and $s_{|S_{i,j}|+1} = -\theta$. Then, for $k = 0, 1, 2, \ldots, |S_{i,j}|+1$, the function $\alpha(x;i;j)$ is a polynomial in x for $x \in [s_k, s_{k-1})$.

Proof: Since the integration of a finite function is a continuous function, we get $\alpha(s_k;i;j) = \lim_{\nu\to 0^+} \alpha(s_k+\nu;i;j)$. With Theorem 17, it is not hard to see that the conclusion holds. ■

With the algorithm in Theorem 17, we can partition the domain of x, $[-\theta, \infty)$, into the subdomains

$$[-\theta, s_{|S_{i,j}|}), [s_{|S_{i,j}|}, s_{|S_{i,j}|-1}), \ldots, [s_1, 0), [0, \infty)$$

and compute the polynomial $\alpha(x;i;j)$ for each subdomain. (We comment that two polynomials in adjacent sub-domains may have the same algebraic expression. When that happens, we will merge the two adjacent subdomains into one. This way, the overall algebraic expression of $\alpha(x;i;j)$ can be simplified; and when we use it to compute $\mathcal{A}(x;i)$, the computation can be significantly more efficient in practice.)

2) Computing $\mathcal{A}(x;i)$ With $i \geq 2$: In the previous section, we have shown how to compute $\mathcal{A}(x;1)$. We now show how to compute $\mathcal{A}(x;i)$ for $i \geq 2$.

It is easy to see that when $j \geq \lceil \frac{-x}{\Delta(1-\epsilon)} \rceil$, we have

$$\alpha(x;i;j) \geq \alpha(x;i;\lceil \frac{-x}{\Delta(1-\epsilon)} \rceil)$$

because setting the targeted cell-level increment too high will only increase the expected cost of the final cell level. So when $i \geq 2$, we have

$$\mathcal{A}(x;i) = \min_{j=0}^{\lceil \frac{-x}{\Delta(1-\epsilon)} \rceil} \alpha(x;i;j).$$

We first use the algorithm in Theorem 17 to compute the functions

$$\alpha(x;i;0)\ ,\ \alpha(x;i;1), \ldots, \alpha(x;i;\lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil).$$

(Note that when $x \in [-\theta, 0)$, $\lceil \frac{-x}{\Delta(1-\epsilon)} \rceil \leq \lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil$.) Let the set $S_{i,j}$ be as defined before. And denote the $|S_{i,j}|$ numbers in the set $S_{i,j}$ by

$$s_1^{i,j}, s_2^{i,j}, \ldots, s_{|S_{i,j}|}^{i,j}$$

such that $0 > s_1^{i,j} > s_2^{i,j} > \cdots > s_{|S_{i,j}|}^{i,j} > -\theta$. We have shown that the function $\alpha(x;i;j)$ is a polynomial in x for x in each of the following subdomains

$$[-\theta, s_{|S_{i,j}|}^{i,j})\ ,\ [s_{|S_{i,j}|}^{i,j}\ ,\ s_{|S_{i,j}|-1}^{i,j})\ ,\ \cdots\ ,\ [s_1^{i,j}, 0)\ ,\ [0, \infty).$$

Given the integer $i \geq 2$, let us define the set P as

$$P = \bigcup_{j=0}^{\lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil} \{s_1^{i,j}, s_2^{i,j}, \ldots, s_{|S_{i,j}|}^{i,j}\}.$$

Let us alternatively denote the elements in P by

$$p_1, p_2, \ldots, p_{|P|}$$

such that

$$p_1 > p_2 > \cdots > p_{|P|}.$$

Also let $p_{-1} = \infty$, $p_0 = 0$ and $p_{|P|+1} = -\theta$. Then, we get the next lemma, which naturally follows from Corollary 18.

Lemma 19: For $k = 0, 1, 2, \ldots, |P| + 1$, the function $\alpha(x;i;j)$ is a polynomial in x for $x \in [p_k, p_{k-1})$. (Here, $i \geq 2$ and $0 \leq j \leq \lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil$.)

With the aforementioned observation, we can easily compute the function $\mathcal{A}(x;i)$ for x in each subdomain $[p_k, p_{k-1})$, where $k = 0, 1, \ldots, |P| + 1$. That is because by the equation $\mathcal{A}(x;i) = \min_{j=0}^{\lceil \frac{-x}{\Delta(1-\epsilon)} \rceil} \alpha(x;i;j)$, $\mathcal{A}(x;i)$ is the minimum of at most $\lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil + 1$ known polynomials. The method of computation should be clear, so we skip its details. The only thing to note is that if the curves defined by these polynomials intersect, the subdomain $[p_k, p_{k-1})$ may need to be partitioned into more smaller intervals, such that in each smaller interval, $\mathcal{A}(x;i)$ is still a polynomial in x.

As mentioned before, after the aforementioned computation, if the polynomials for $\mathcal{A}(x;i)$ in adjacent subdomains happen to have the same algebraic expression, we merge the two subdomains into one for a more succinct representation.

D. Optimal Cell Programming Algorithm

In this section, we describe the cell-programming strategy that minimizes the expected cost of the final cell level. Recall that at most r rounds of charge injection can be used for programming a cell. We use the algorithm described before to compute the functions $\mathcal{A}(x;i)$ for $i = 1, 2, \ldots, r$, and compute the functions $\alpha(x;i;j)$ for $i = 1, 2, \ldots, r$ and $j = 0, 1, 2, \ldots, \lceil \frac{\theta}{\Delta(1-\epsilon)} \rceil$. These functions are then stored in the memory storage system, to be looked up during the actual cell-programming process.[2]

We now present the cell-programming algorithm. Recall that θ is the target cell level, and r rounds of charge injection can be used to move the cell level from 0 to close to θ. The task is to adaptively choose the targeted cell-level increment for each of the r rounds.

For $i = 1, 2, \ldots, r$, let x_i denote the actual cell level after the ith round of charge injection. Let $x_0 = 0$ denote the initial cell level. The objective of cell programming is to minimize the expectation of $C(x_r)$. The optimal cell-programming algorithm is as follows.

For $i = 0, 1, \ldots, r-1$, set the targeted cell-level increment in the $(i+1)$th round of charge injection to be $j^*\Delta$ such that

$$\alpha(x_i - \theta; r - i; j^*) = \mathcal{A}(x_i - \theta; r - i).$$

That is, in each round, the algorithm always chooses the targeted cell-level increment to be the one that, given the current cell level, minimizes the expected cost of the final cell level. Note again that the randomness in programming is due to the programming noise, and the programming strategy in each round is chosen adaptively based on that. We can see that the algorithm also minimizes the expectation of the overall cost (i.e., the cost of the r rounds in total), $C(x_r)$, due to the linearity of expectation for random variables (i.e., the programming noise in the r rounds). So, it is optimal.

It should be noted that once the functions $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$ are stored, it is very efficient to look them up for the actual programming of cells (specifically, to find the value j^* in the aforementioned programming algorithm). Let us now analyze the time complexity of computing these functions $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$, which are computed only once. For

[2]Since $\theta \leq \mathcal{L}$, in the aforementioned computation, we let $\theta = \mathcal{L}$. Functions $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$ computed this way can be used for any $\theta \leq \mathcal{L}$.

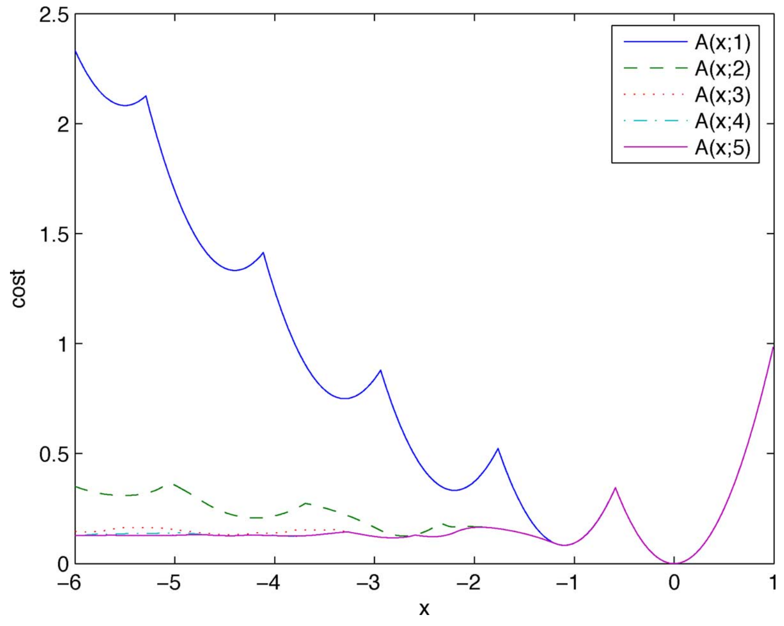

Fig. 5. Functions $\mathcal{A}(x;1)$, $\mathcal{A}(x;2)$, $\mathcal{A}(x;3)$, $\mathcal{A}(x;4)$, and $\mathcal{A}(x;5)$. Here, the cost function is for MLC, and $\Delta = 1, \epsilon = 0.4, \delta = 0.6$. Note that the value of $A(x;i)$ decreases with i.

simplicity, we use the cost function $C(x) = (x-\theta)^2$ for the MLC technology as an example, but the results can be easily extended for both general cost functions in Definition 14.

When $C(x) = (x-\theta)^2$, the function $\mathcal{A}(x;1)$ is a degree-2 polynomial of x in $O(\frac{\theta}{\Delta\delta})$ intervals. By induction (for simplicity, we only present the conclusion and skip the detailed analysis), for $i = 2,3,\ldots,r$ and $j = 0,1,\ldots,\lceil\frac{\theta}{\Delta(1-\epsilon)}\rceil$, the function $\alpha(x;i;j)$ is a degree-$(i+1)$ polynomial of x in $O(\frac{1-\epsilon}{\delta}(\frac{2\theta}{\Delta(1-\epsilon)})^{i-1}(\frac{\theta}{\Delta(1-\epsilon)})^{2(i-2)}i!)$ intervals; for $i = 2,3,\ldots,t$, the function $\mathcal{A}(x;i)$ is a degree-$(i+1)$ polynomial in $O(\frac{\theta}{\Delta\delta}(\frac{2\theta}{\Delta(1-\epsilon)})^{i-1}(\frac{\theta}{\Delta(1-\epsilon)})^{2(i-1)}(i+1)!)$ intervals. So, the overall time complexity of computing all the functions is $O(\frac{1-\epsilon}{\delta}(\frac{2\mathcal{L}^3}{\Delta^3(1-\epsilon)^3})^r(r+1)!)$. So, when the number of rounds of charge injection r is a constant (in practice r is small), Δ is not arbitrarily small, and ϵ is not arbitrarily close to 1, the complexity is upper bounded by a polynomial function of the parameters. We note that the aforementioned complexity is derived based on a very pessimistic analysis. The actual complexity is usually (significantly) lower.

E. Numerical Computation

We demonstrate the numerical computation of the functions $\mathcal{A}(x;i)$ and $\alpha(x;i;j)$. We consider two cases for the cost function: for MLC and for rank modulation (see Definition 14).

1) Multilevel Cells: For MLC, we set the cost function as

$$C(x) = (x-\theta)^2$$

and set the parameters as $\Delta = 1, \epsilon = 0.4, \delta = 0.6$.

The function $\mathcal{A}(x;i)$ is shown in Fig. 5, for $x \in [-6,1)$ and $i = 1,2,\ldots,5$. We can see that $\mathcal{A}(x;i)$ is piecewise polynomial, and it monotonically decreases when i increases (because more rounds of charge injection leads to more accurate programming). We can also see that $\mathcal{A}(x;i)$ converges quickly as i increases.

$\mathcal{A}(x;3)$	
[-6,-5.56)	$23.9 + 11.2x + 1.76x^2 + 0.0917x^3$
[-5.56,-5.49)	$16.4 + 8.37x + 1.43x^2 + 0.0815x^3$
[-5.49,-5.39)	$24.9 + 12.7x + 2.16x^2 + 0.122x^3$
[-5.39,-4.76)	$14.3 + 8.76x + 1.8x^2 + 0.122x^3$
[-4.76,-4.18)	$7.66 + 4.6x + 0.924x^2 + 0.0611x^3$
[-4.18,-4.16)	$15.5 + 10.6x + 2.42x^2 + 0.183x^3$
[-4.16,-3.79)	$8.84 + 5.8x + 1.27x^2 + 0.0917x^3$
[-3.79,-3.77)	$0.948 + 1.63x + 0.722x^2 + 0.0917x^3$
[-3.77,-3.56)	$1.55 + 0.771x + 0.107x^2$
[-3.56,-3.42)	$-6.75 - 6.21x - 1.85x^2 - 0.183x^3$
[-3.42,-3.39)	$-1.22 - 0.743x - 0.1x^2 - 1.49e-08x^3$
[-3.39,-2.59)	$5.91 + 5.57x + 1.76x^2 + 0.183x^3$
[-2.59,-2.19)	$7.1 + 7.92x + 2.97x^2 + 0.367x^3$
[-2.19,-1.82)	$1.84 + 3.1x + 1.87x^2 + 0.367x^3$
[-1.82,-1.19)	$-0.259 - 0.413x - 0.1x^2$
[-1.19,-0.588)	$1.29 + 2.2x + x^2$
[-0.588,1)	x^2

Fig. 6. Function $\mathcal{A}(x;3)$ for MLC.

$\alpha(x;3;3)$	
[-6,-5.99)	$-9.86 - 4.78x - 0.761x^2 - 0.0407x^3$
[-5.99,-5.49)	$16.4 + 8.37x + 1.43x^2 + 0.0815x^3$
[-5.49,-5.39)	$24.9 + 12.7x + 2.16x^2 + 0.122x^3$
[-5.39,-4.76)	$14.3 + 8.76x + 1.8x^2 + 0.122x^3$
[-4.76,-4.13)	$7.66 + 4.6x + 0.924x^2 + 0.0611x^3$
[-4.13,-3.99)	$5.06 + 2.29x + 0.267x^2 - 9.93e-09x^3$
[-3.99,-2.99)	$12.8 + 8.12x + 1.73x^2 + 0.122x^3$
[-2.99,-2.39)	$9.55 + 4.85x + 0.633x^2$
[-2.39,1)	$11.6 + 6.6x + x^2$

Fig. 7. Function $\alpha(x;3,3)$ for MLC.

As an example, we show the numerical functions of $\mathcal{A}(x;3)$ and $\alpha(x;3;3)$ in Figs. 6 and 7, respectively, for $x \in [-6,1)$. The left column of the table shows the domain for x, and the right column shows the polynomial ($\mathcal{A}(x;3)$ or $\alpha(x;3;3)$) in this domain.

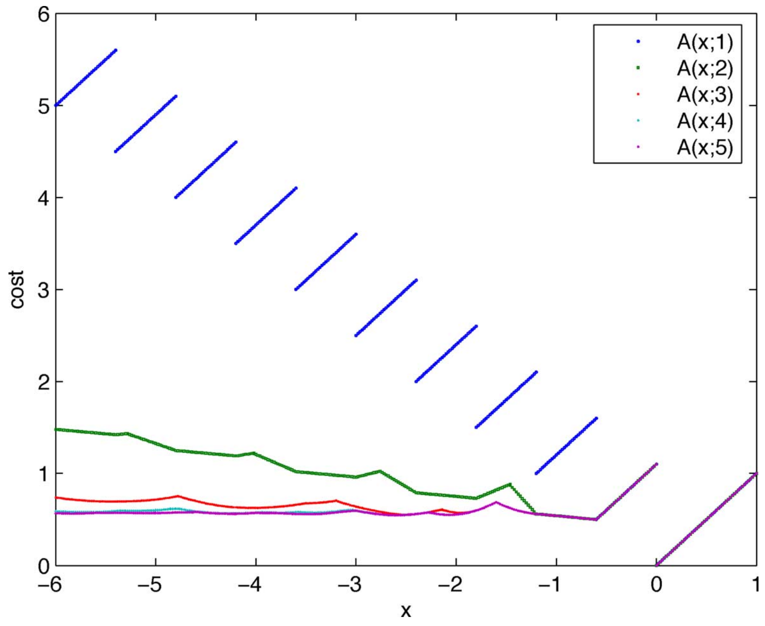

Fig. 8. Functions $\mathcal{A}(x;1), \mathcal{A}(x;2), \mathcal{A}(x;3), \mathcal{A}(x;4)$, and $\mathcal{A}(x;5)$. Here, the cost function is for rank modulation, and $\Delta = 1, \epsilon = 0.4, \delta = 0.6$. Note that the value of $A(x;i)$ decreases with i.

$\mathcal{A}(x;3)$	
[-6,-5.4)	$4.76 + 1.5x + 0.138x^2$
[-5.4,-5.16)	$3.42 + 1x + 0.0917x^2$
[-5.16,-4.8)	$6.47 + 2.18x + 0.206x^2$
[-4.8,-4.77)	$4.89 + 1.52x + 0.138x^2$
[-4.77,-4.56)	$2.04 + 0.853x + 0.122x^2$
[-4.56,-4.2)	$5.22 + 2.25x + 0.275x^2$
[-4.2,-3.6)	$3.6 + 1.48x + 0.183x^2$
[-3.6,-3.5)	$2.41 + 0.817x + 0.0917x^2$
[-3.5,-3.2)	$4.08 + 1.94x + 0.275x^2$
[-3.2,-3)	$2.32 + 1.38x + 0.275x^2$
[-3,-2.66)	$1.08 + 0.56x + 0.138x^2$
[-2.66,-2.4)	$4.02 + 2.76x + 0.55x^2$
[-2.4,-2.14)	$2.43 + 1.44x + 0.275x^2$
[-2.14,-2.06)	$1.25 + 0.89x + 0.275x^2$
[-2.06,-1.8)	$4.77 + 4.3x + 1.1x^2$
[-1.8,-1.6)	$2.99 + 2.32x + 0.55x^2$
[-1.6,-1.21)	$1.23 + 1.22x + 0.55x^2$
[-1.21,-1.20)	$-0.88 - 1.2x$
[-1.20,-0.6)	$0.44 - 0.1x$
[-0.6,0)	$1.1 + x$
[0,1)	x

Fig. 9. Function $\mathcal{A}(x;3)$ for rank modulation.

2) Rank Modulation: For rank modulation, we set the cost function as

$$C(x) = \begin{cases} \infty & \text{, if } x < \theta \\ x - \theta & \text{, if } x \geq \theta \end{cases}$$

and set the parameters as $\Delta = 1, \epsilon = 0.4, \delta = 0.6$.

The function $\mathcal{A}(x;i)$ is shown in Fig. 8, for $x \in [-6,1)$ and $i = 1,2,\ldots,5$. Again, we see that $\mathcal{A}(x;i)$ is piecewise polynomial, it monotonically decreases with i, and it converges quickly with i. For illustration, we also show the numerical functions of $\mathcal{A}(x;3)$ and $\alpha(x;3;3)$ in Figs. 9 and 10, respectively, for $x \in [-6,1)$.

$\alpha(x;3;3)$	
[-6,-5.82)	$-1.99 - 0.76x - 0.0458x^2$
[-5.82,-5.4)	$1.64 + 0.487x + 0.0611x^2$
[-5.4,-4.8)	$5.21 + 1.81x + 0.183x^2$
[-4.8,-4.56)	$2.04 + 0.853x + 0.122x^2$
[-4.56,-4.2)	$5.22 + 2.25x + 0.275x^2$
[-4.2,-3.6)	$3.6 + 1.48x + 0.183x^2$
[-3.6,-3.26)	$2.41 + 0.817x + 0.0917x^2$
[-3.26,-3)	$5.35 + 2.61x + 0.367x^2$
[-3,-2.4)	$3.7 + 1.51x + 0.183x^2$
[-2.4,-1.8)	$2.64 + 0.633x$
[-1.8,1)	$3.3 + x$

Fig. 10. Function $\alpha(x;3,3)$ for rank modulation.

V. Conclusion

This paper studies the capacity and programming of flash memories. The cell programming is an iterative process that monotonically increases the charge level in the cell. This makes the flash memory cells a unique kind of storage media, which can be modeled, and generalized, by the monotonic storage channel. This paper presents an optimal programming algorithm that achieves the zero-error capacity. It also presents a programming algorithm that optimizes the expected cell-programming precision, under the MLC model and the rank modulation model, respectively. The results in this paper can be extended in several significant ways. First, more accurate programming noise models can be considered, which may depend on the physical quality of the flash memory cell, the interference introduced by parallel programming of cells, and the current level of the cell. Second, reliability requirements can be introduced so that after data are written, they can be reliably stored and retrieved despite the read disturbs, write disturbs, and other noise sources. A typical approach is to introduce sufficiently large gaps between adjacent cell levels and to use appropriate error-correcting codes. Third, the capacity of cell ensembles and its relationship with the programming cost is an interesting topic to study.

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 Int. Symp. Circuits Syst.*, Kobe, Japan, May 23–26, 2005, pp. 2148–2151.

[2] V. Bohossian, A. Jiang, and J. Bruck, "Buffer codes for asymmetric multi-level memory," in *Proc. IEEE Int. Symp. Inf. Theory*, Nice, France, Jun. 24–29, 2007, pp. 1186–1190.

[3] , P. Cappelletti, C. Golla, P. Olivo, and E. Zanoni, Eds., *Flash Memories*, 1st ed. Norwell, MA: Kluwer, 1999.

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

[5] H. Finucane, Z. Liu, and M. Mitzenmacher, "Designing floating codes for expected performance," in *Proc. 46th Annu. Allerton Conf. Commun., Control Comput.*, Monticello, IL, Sep. 23–26, 2008, pp. 1389–1396.

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

[7] M. Grossi, M. Lanzoni, and B. Ricco, "Program schemes for multilevel flash memories," *Proc. IEEE*, vol. 91, no. 4, pp. 594–601, Apr. 2003.

[8] A. Jiang, V. Bohossian, and J. Bruck, "Floating codes for joint information storage in write asymmetric memories," in *Proc. IEEE Int. Symp. Inf. Theory*, Nice, France, Jun. 24–29, 2007, pp. 1166–1170.

[9] A. Jiang and J. Bruck, "Joint coding for flash memory storage," in *Proc. IEEE Int. Symp. Inf. Theory*, Toronto, ON, Canada, Jul. 6–11, 2008, pp. 1741–1745.

[10] A. Jiang and J. Bruck, "On the capacity of flash memories," in *Proc. Int. Symp. Inf. Theory Appl.*, Dec. 2008, pp. 94–99.

[11] A. Jiang, M. Langberg, M. Schwartz, and J. Bruck, "Universal rewriting in constrained memories," in *Proc. IEEE Int. Symp. Inf. Theory*, Seoul, Korea, Jun./Jul. 2009, pp. 1219–1223.

[12] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for flash memories," in *Proc. IEEE Int. Symp. Inf. Theory*, Toronto, ON, Canada, Jul. 6–11, 2008, pp. 1731–1735.

[13] A. Jiang, R. Mateescu, M. Schwartz, and J. Bruck, "Rank modulation for flash memories," *IEEE Trans. Inf. Theory*, vol. 55, no. 6, pp. 2659–2673, Jun. 2009.

[14] A. Jiang, M. Schwartz, and J. Bruck, "Error-correcting codes for rank modulation," in *Proc. IEEE Int. Symp. Inf. Theory*, Toronto, ON, Canada, Jul. 6–11, 2008, pp. 1736–1740.

[15] L. A. Lastras-Montano, M. Franceschini, T. Mittelholzer, and M. Sharma, "Rewritable storage channels," in *Proc. Int. Symp. Inf. Theory Appl.*, Auckland, New Zealand, Dec. 7–10, 2008, pp. 106–111.

[16] H. Nobukata, S. Takagi, K. Hiraga, T. Ohgishi, M. Miyashita, K. Kamimura, S. Hiramatsu, K. Sakai, T. Ishida, H. Arakawa, M. Itoh, I. Naiki, and M. Noda, "A 144-Mb, eight-level NAND flash memory with optimized pulsewidth programming," *IEEE J. Solid-State Circuits*, vol. 35, no. 5, pp. 682–690, May 2000.

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

[18] L. A. Silverman, "On binary channels and their cascades," *IRE Trans. Inf. Theory*, vol. IT-1, pp. 19–27, Dec. 1955.

[19] L. G. Tallini, S. Al-Bassam, and B. Bose, "Feedback codes achieving the capacity of the Z-channel," *IEEE Trans. Inf. Theory*, vol. 54, no. 3, pp. 1357–1363, Mar. 2008.

[20] R. R. Varshamov, "Estimates of the number of signals in codes with correction of nonsymmetric errors," (in Russian) Transl.:Translation: Automation and Remote Control, vol. 25, pp. 1468–1469, 1965 *Avtomatika i Telemekhanika*, vol. 25, no. 11, pp. 1628–1629, 1964.

[21] E. Yaakobi, A. Vardy, P. H. Siegel, and J. Wolf, "Multidimensional flash codes," in *Proc. 46th Annu. Allerton Conf. Commun., Control Comput.*, Monticello, IL, Sep. 23–26, 2008, pp. 392–399.

Anxiao (Andrew) Jiang (S'00–M'05) received the B.S. degree in electronic engineering from Tsinghua University, Beijing, China in 1999, and the M.S. and Ph.D. degrees in electrical engineering from the California Institute of Technology, Pasadena, in 2000 and 2004, respectively.

He is currently an Associate Professor in the Computer Science and Engineering Department at Texas A&M University in College Station, Texas. His research interests include information theory, data storage, networks, and algorithm design.

Dr. Jiang is a recipient of the NSF CAREER Award in 2008 for his research on information theory for flash memories and a recipient of the 2009 IEEE Communications Society Best Paper Award in Signal Processing and Coding for Data Storage.

Hao Li received the B.S. degree in computer science from Tsinghua University, Beijing, China in 1998, the M.S. degree in computer science from Institute of Software, Chinese Academy of Sciences, Beijing, in 2001, and the Ph.D. degree in computer engineering from Texas A&M University, College Station, Texas in 2010. He is currently a software engineer at Google Inc. in Kirkland.

Jehoshua Bruck (S'86–M'89–SM'93–F'01) received the B.Sc. and M.Sc. degrees in Electrical Engineering from the Technion, Israel Institute of Technology, in 1982 and 1985, respectively and the Ph.D. degree in Electrical Engineering from Stanford University in 1989.

He is the Gordon and Betty Moore Professor of Computation and Neural Systems and Electrical Engineering at the California Institute of Technology. His research focuses on information theory and systems and the theory computation in biological networks. His extensive industrial experience includes working for IBM Research as well as cofounding and serving as a chairman of Rainfinity (acquired in 2005 by EMC) and XtremIO.

Dr. Bruck is a recipient of the Feynman Prize for Excellence in Teaching, a Sloan Research Fellowship, a National Science Foundation Young Investigator Award, an IBM Outstanding Innovation Award, and an IBM Outstanding Technical Achievement Award.

His papers were recognized in journals and conferences, including winning the 2009 IEEE Communications Society best paper award in Signal Processing and Coding for Data Storage for his paper on rank modulation for flash memories; the 2005 A. Schelkunoff Transactions prize paper award from the IEEE Antennas and Propagation society for his paper on signal propagation in wireless networks; and the 2003 best paper award in the 2003 Design Automation Conference for his paper on cyclic combinational circuits.