Using Lifetime-Aware Progressive Programming to Improve SLC NAND Flash Memory Write Endurance

Guiqiang Dong, *Student Member, IEEE*, Yangyang Pan, *Student Member, IEEE*, and Tong Zhang, *Senior Member, IEEE*

***Abstract*— This paper advocates a lifetime-aware progressive programming concept to improve single-level per cell NAND flash memory write endurance. NAND flash memory program/erase (P/E) cycling gradually degrades memory cell storage noise margin, and sufficiently strong fault tolerance must be used to ensure the memory P/E cycling endurance. As a result, the relatively large cell storage noise margin in early memory lifetime is essentially wasted in conventional design practice. This paper proposes to always fully utilize the available cell storage noise margin by adaptively adjusting the number of storage levels per cell, and progressively use these levels to realize multiple 1-bit programming operations between two consecutive erase operations. This simple progressive programming design concept is realized by two different implementation strategies, which are discussed and compared in detail. On the basis of an approximate NAND flash memory device model, we carried out simulations to quantitatively evaluate this design concept. The results show that it can improve the write endurance by 35.9% and in the meanwhile improve the average programming speed by 12% without sacrificing read speed.**

***Index Terms*— NAND flash memory, P/E cycling endurance, progressive programming, single-level per cell (SLC).**

I. Introduction

THE steady bit cost reduction of NAND flash memory now makes it economically viable to implement solid-state drive (SSD) using NAND flash memory. Nevertheless, continuous technology scaling meanwhile degrades the program/erase (P/E) cycling endurance of NAND flash memory [1]. Mainstream NAND flash memory can store either 1 bit or 2 bits per memory cell, which are referred to single-level per cell (SLC) and multilevel per cell (MLC), respectively. Compared to its MLC counterpart, SLC NAND flash memory has much higher P/E cycling endurance at the penalty of higher cost. Although MLC NAND flash memory completely dominates the consumer and low-end computing market, the write-intensive nature of high-end applications demands the use of SLC NAND flash memory, e.g., SSDs built upon either only SLC NAND flash memory or hybrid SLC/MLC NAND flash memory [2]. Therefore, to enable high-end applications to fully exploit the cost benefit of technology scaling, it is highly desirable to develop techniques to effectively offset the impact of technology scaling on SLC NAND flash memory P/E cycling endurance.

This paper presents a simple progressive programming concept that allows SLC memory to sustain more writes. NAND flash memory P/E cycling causes memory cell wear-out, which manifests as gradual memory cell operational noise margin degradation, leading to cycling endurance limit. Memory manufacturers must fabricate enough number of redundant memory cells to tolerate the worst case noise margin at the end of memory P/E cycling lifetime. Clearly, the relatively larger noise margin at the early lifetime of SLC memory is more than enough to store two levels i.e., conventional SLC memory essentially wastes the large noise margin during its early lifetime. This leads to the simple idea of this work: according to memory wear-out condition, we adaptively adjust the number of storage levels per cell and progressively use these more-than-two-level storage capacity to accommodate more than one 1-bit programming operations between two consecutive erase operations. Effective endurance is defined as the total number of 1-bit programming operations that one memory cell can survive. We can expect that progressive programming SLC can achieve higher effective endurance than conventional SLC memory.

This simple progressive programming SLC design concept can be implemented using two different strategies. The first implementation strategy is called constant-shift progressive programming, which always use only two active storage levels to represent logic 0 and 1, and the active storage levels always shift upward by one level during each 1-bit programming. The other implementation strategy is called fixed-position progressive programming, where all the storage levels alternatively represent logic 0 and 1, i.e., each storage level associates with a fixed logic (either logic 1 or 0). Intuitively, one may expect that progressive programming SLC memory using these two similar and straightforward implementation strategies should behave similarly with similar performance metrics. Nevertheless, we show that this intuition is wrong, and the constant-shift progressive programming can achieve

Manuscript received July 22, 2011; revised November 14, 2011 and February 21, 2012; accepted May 18, 2013. Date of publication July 3, 2013; date of current version May 20, 2014.

G. Dong is with Skyera, Inc., San Jose, CA 95131 USA (e-mail: dongguiqiang@gmail.com).

Y. Pan is with Fusion-io, San Jose, CA 95134-1922 USA (e-mail: yyangpan@gmail.com).

T. Zhang is with the Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, NY 12180 USA (e-mail: tzhang@ecse.rpi.edu).

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/TVLSI.2013.2267753

higher effective endurance and realize higher programming and read speed. This is mainly because these two different implementation strategies cause different operational noise characteristics and employ different programming and read procedures.

This paper elaborates on these two progressive programming implementation strategies, and discusses their essential difference and its implications on effectiveness endurance, programming, and read speed. We also discuss the corresponding implementation overhead. To further quantitatively demonstrate the effectiveness, based on extensive open literature from device research community, we first developed an approximate mathematical memory device model that captures major storage distortion sources. Using this device model, we carried out extensive Monte Carlo simulations to quantitatively study and compare these two different implementation strategies and conventional SLC memory. Results show that constant-shift and fixed-position progressive programming SLC memory can improve effective endurance by 35.9% and 29.4%, respectively. Compared to its fixed-position counterpart, constant-shift progressive programming achieves higher programming speed and higher read speed. Compared to conventional SLC memory, constant-shift progressive programming can improve the average programming speed by 12% and maintain the same read speed.

II. Background

To achieve sufficient memory cell operational noise margin, NAND flash memory programming must achieve a tight memory cell threshold voltage control, which is typically realized using incremental step-pulse programming (ISPP) [3], [4]. However, the noise margin can be seriously degraded in practice, mainly due to P/E cycling effects and cell-to-cell interference, which will be discussed in the remainder of this section.

A. Effects of P/E Cycling

Flash memory P/E cycling causes damage to the tunnel oxide of floating gate transistors in the form of charge traps in the oxide and interface states [5]–[7], which results in memory cell threshold voltage shift and fluctuation and hence degrades memory device noise margin. Let N denotes the number of P/E cycles that memory cells have gone through and ΔN_{trap} denotes the density growth of either interface or oxide traps. We can approximately quantify the relation between interface/oxide traps generation and P/E cycles as

$$\Delta N_{\text{trap}} = A \cdot N^a \tag{1}$$

where A is a constant factor fitted from measurements. Such a power-law relationship is explained by the widely accepted reaction–diffusion model (R–D) in negative bias temperature instability [8], [9] and the scattering-induced diffusion model [10]. Those gradually accumulated traps result in two major types of noises.

1) Electrons capture and emission events at charge trap sites near the interface developed over P/E cycling directly result in memory cell threshold voltage fluctuation, which is referred to as random telegraph noise (RTN) [11], [12].
2) Interface state trap recovery and electron detrapping [10], [13] gradually reduce memory cell threshold voltage, leading to the data retention limitation. This is referred to as data retention noise.

As the significance of these noises grows with the trap density and trap density grows with P/E cycling, NAND flash memory cell noise margin monotonically degrades with P/E cycling. This leads to the NAND flash memory P/E cycling endurance limit beyond which memory cell noise margin degradation can no longer be accommodated by the memory system fault tolerance capability.

B. Cell-to-Cell Interference

In NAND flash memory, the threshold voltage shift of one floating gate transistor can influence the threshold voltage of its neighboring floating gate transistors through parasitic capacitance-coupling effect [14]. This is referred to as cell-to-cell interference, which has been well-recognized as the one of major noise sources in NAND flash memory [15]–[17]. Threshold voltage shift of a victim cell caused by cell-to-cell interference is estimated as [14]

$$F = \sum_k (\Delta V_t^{(k)} \cdot \gamma^{(k)}) \tag{2}$$

where $\Delta V_t^{(k)}$ represents the threshold voltage shift of one interfering cell which is programmed after the victim cell, and $\gamma^{(k)}$ is coupling ratio.

III. Lifetime-Aware Progressive Programming

From the earlier discussions, it is clear that the raw storage reliability of NAND flash memory cells gradually degrades with P/E cycling. During the early lifetime of memory cells (i.e., the P/E cycling number is relatively small), the oxide damage is relatively small, which leads to a relatively large memory cell noise margin and hence good raw storage reliability. Because the oxide damage scales up with the P/E cycling number in approximate power-law fashions, the raw storage reliability of memory cells gradually degrades as the P/E cycling number increases. Given the target P/E cycling endurance limit (e.g., 10k P/E cycling), each memory wordline must have enough redundant memory cells so that the corresponding error correction code (ECC) ensures the storage integrity as the P/E cycling reaches the endurance limit. As a result, NAND flash memory cells have more-than-enough noise margin for most of the time throughout the entire memory lifetime, especially at its early lifetime.

In this paper, we are interested in leveraging such noise margin dynamics to improve SLC NAND flash memory write endurance. The basic idea is very simple: if the present memory cell noise margin accommodates $m > 2$ storage levels per cell, we progressively utilize these multiple storage levels to enable multiple write-1-bit operations before we have to erase this cell. This is referred to as progressive programming, and each multiwrite-single-erase operation is

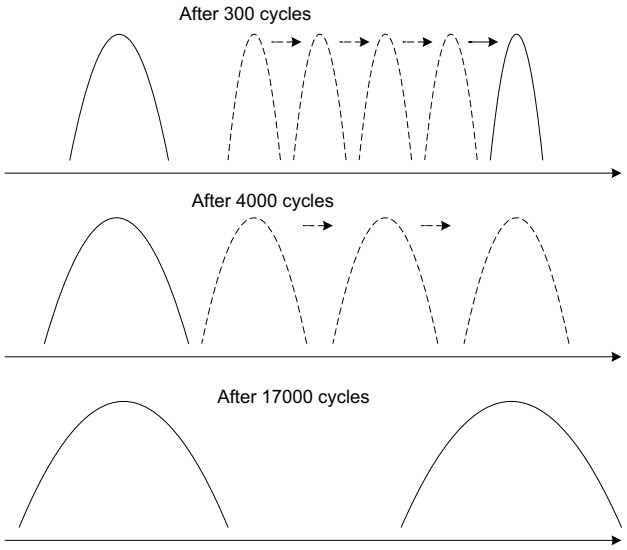


Fig. 1. Illustration of using memory cell noise margin dynamics to enable multiple storage levels per SLC NAND flash memory cell and hence progressive programming.

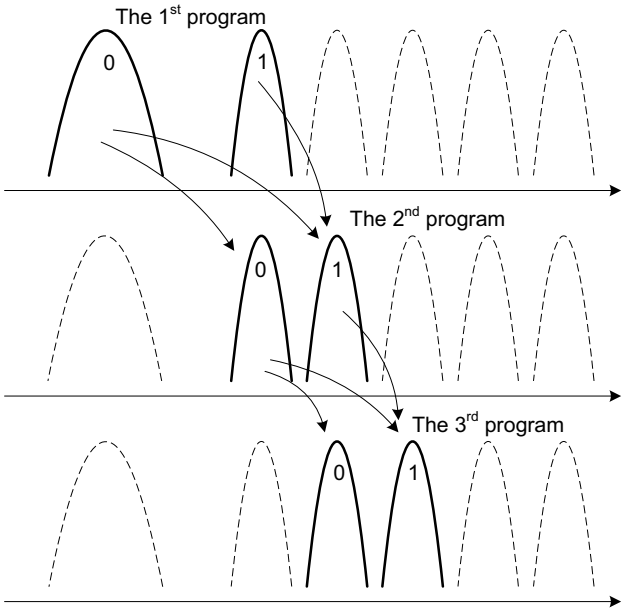


Fig. 2. Illustration of constant-shift progressive programming. The solid levels are active to represent logic 0 and 1, while those dashed levels are inactive.

called a super P/E cycle. Throughout the entire lifetime of SLC NAND flash memory cells, we can adaptively adjust the number of available storage levels per cell, which corresponds to different degree of progressive programming. This is shown in Fig. 1, where the same ISPP program step-voltage is used and the threshold voltage distribution becomes wider because of more significant P/E cycling effects. To gracefully exploit the gradual noise margin degradation over P/E cycling, the achievable number of storage levels per cell may not necessarily be the power of two and can be gradually reduced one by one as the number of P/E cycles increase. The cycling-induced damage of NAND flash memory cells is mainly due to the voltage applied across oxide (referred to as voltage stress) and electron tunneling current flowing through the oxide (see [18], [19]). Under the same overall memory cell threshold voltage window and the same ISPP program step-voltage, during each super cycle, progressive-programming SLC memory cells experience the same voltage stress and same electron tunneling current as their conventional SLC counterparts during each P/E cycle. This means that each super P/E cycle tends to induce similar device wear-out as each P/E cycle in conventional SLC memory. Therefore, it is reasonable to expect that progressive-programming SLC memory can sustain the same amount of super P/E cycles as the amount of P/E cycle in conventional SLC memory, which implies more write operations (and hence higher write endurance) compared to conventional SLC memory.

In practice, this simple progressive programming concept can be implemented using two different strategies. In the remainder of this section, we will describe these two different implementation strategies and compare them in terms of various memory performance metrics. The effectiveness of this progressive programming concept and difference of these two different implementation strategies will be quantitatively evaluated through simulations in Section IV.

A. Constant-Shift Progressive Programming

The first implementation strategy is called constant-shift progressive programming. As illustrated in Fig. 2, it always uses only two active storage levels to represent logic 0 and 1, and the active storage levels always shift upward by one level during each 1-bit programming, i.e., during the first 1-bit programming within each super cycle, the lowest two storage levels are active, representing logic 0 and 1. During the second 1-bit programming, the first storage level becomes inactive and the second and third storage levels become active instead and represent logic 0 and 1. This process repeats until the highest storage level is reached, after which the cell needs to be erased.

Fig. 3 shows the operational flow diagram of constant-shift progressive programming. During each 1-bit programming operation, based on the input bit and the number of 1-bit programming operations that are elapsed within the current super cycle, we can determine the target storage level. Meanwhile, we sense the memory cell threshold voltage to determine its present storage level. If the present storage level is not the target storage level, we apply the ISPP operations to move the memory cell threshold voltage into the target storage level. Because there are only two active storage levels associated with each 1-bit programming, the verify operation within each program–verify iteration in ISPP operation only involves two verify reference voltages, except in the first 1-bit programming operation, in which only one verify reference voltage is needed as conventional 1-bit programming. Hence, as illustrated in Fig. 4, each program–verify iteration contains only two verify pulses with two verify reference voltages.

B. Fixed-Position Progressive Programming

The second implementation strategy is called fixed-position progressive programming, where all the storage levels alter-

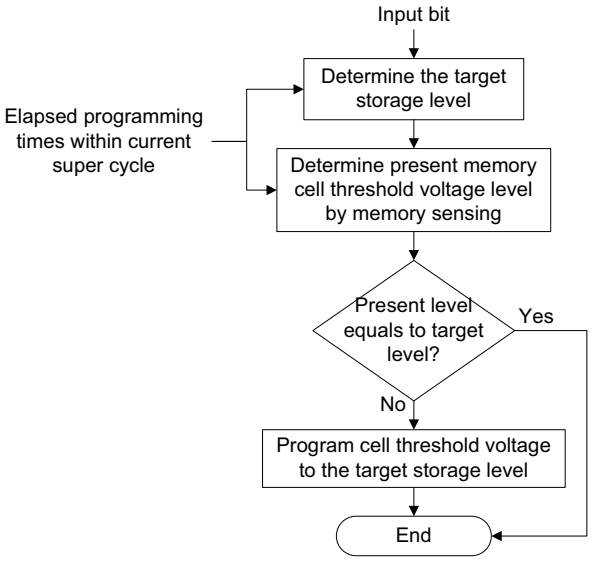


Fig. 3. Operational flow diagram of constant-shift progressive programming.

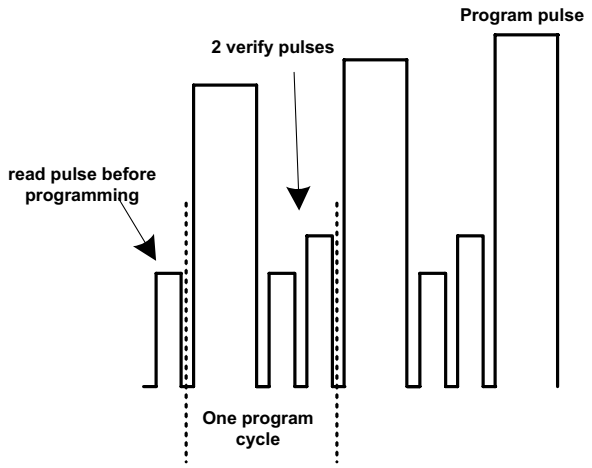


Fig. 4. Illustration of program process of 1-bit constant-shift progressive programming except the first programming within one super P/E cycle. Because only two levels are active, two verify pulses are needed in every program-and-verify iteration.

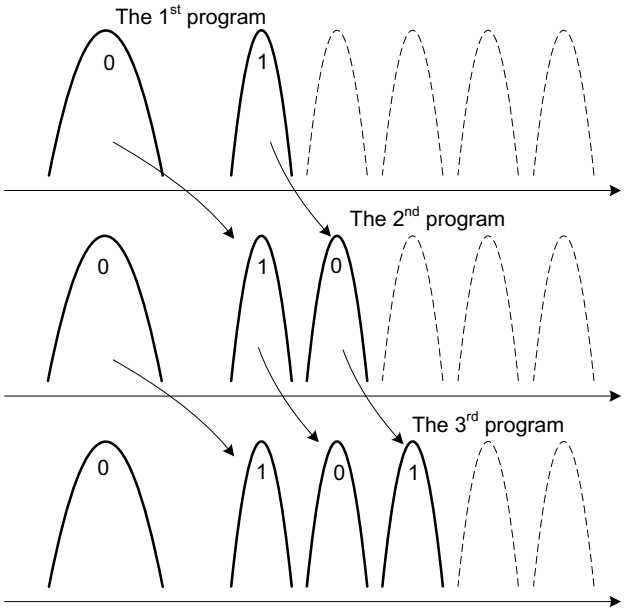


Fig. 5. Illustration of fixed-position progressive programming. Storage levels in solid lines are active levels.

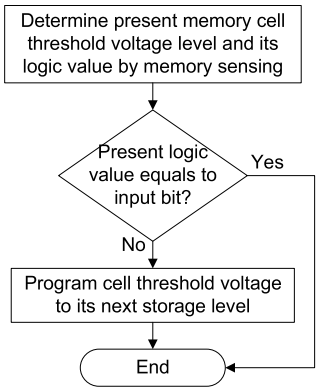


Fig. 6. Operational flow of fixed-position progressive programming.

natively represent logic 0 and 1, i.e., each storage level is always bound with the same fixed logic (either logic 1 or 0). Therefore, the number of active storage levels in fixed-position progressive programming monotonically increases with the write-1-bit operations, i.e., during the first 1-bit programming within each super cycle, the lowest two storage levels are active, representing logic 0 and 1. During the second 1-bit programming, the lowest three storage levels become active. This is shown in Fig. 5.

Fig. 6 shows the operational flow diagram of fixed-position progressive programming. During each 1-bit programming operation, we first sense the memory cell threshold voltage to determine its present storage level and hence its storage logic value (either 1 or 0). If the present storage logic value does not equal to the input bit, we apply the ISPP operations to move the memory cell threshold voltage into the target storage level. Suppose one memory cell can accommodate m storage levels. Because we need to move the threshold voltage of one memory cell only when its present storage value does not equal to the bit being programmed, the number of 1-bit programming operations that one memory cell can sustain during each super cycle is lower bounded by $m-1$. This is in contrast to the earlier constant-shift progressive programming, in which the number of 1-bit programming operations that one memory cell can sustain during each super cycle is exactly equal to $m-1$. However, as each page in NAND flash memory covers a large amount of cells (e.g., 512 B to 4 kB) and the number of 1-bit fixed-position progressive programming that each page can sustain during each super cycle is limited by the covered worst case cell, it is reasonable to expect that this number (almost) always equals $m-1$.

During the kth 1-bit fixed-position progressive programming within each super cycle, each one of the lowest $k+1$ storage levels will be used by some memory cells among the large number of memory cells in each page. Hence, the word-line voltage must sweep through all the k verify reference

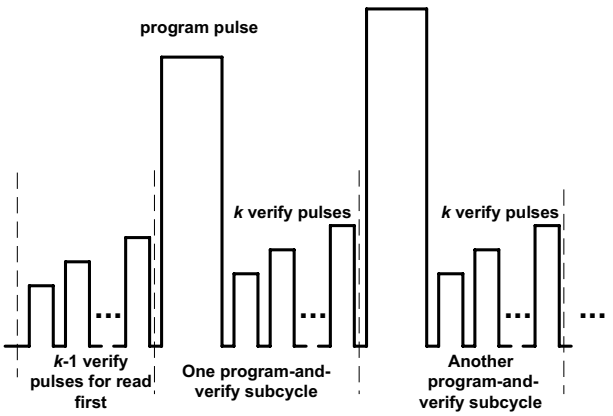


Fig. 7. Illustration for program process of the kth 1-bit fixed-position progressive programming within one super P/E cycle.

voltage during each program–verify cycle in ISPP operation, as illustrated in Fig. 7.

C. Discussions and Comparisons

This subsection discusses and compares the conventional SLC memory and progressive-programming SLC with the above-mentioned two different implementation strategies in terms of major memory system metrics. For a fair comparison, we assume that all the scenarios use the same ISPP programming step-voltage ΔV_{pp}, and have the same overall threshold voltage window (i.e., the highest storage level in progressive programming is the same as the high storage level in conventional SLC).

1) Programming Speed and Cell-to-Cell Interference: Both programming speed and cell-to-cell interference depend on the ratio of the largest memory cell threshold voltage shift during each programming operation over the programming step-voltage ΔV_{pp}. If ΔV_{pp} is fixed, as we reduce the largest memory cell threshold voltage shift, the memory programming speed increases and worst case cell-to-cell interference reduces. In conventional SLC memory, the largest memory cell threshold voltage shift is from the erased state (i.e., the lowest storage level) to the programmed state (i.e., the highest storage level). In constant-shift progressive programming, as shown in Fig. 2, the largest memory cell threshold voltage shift is from the erased state (i.e., the first storage level) to the third storage level, because the erased state tends to have a much wider distribution than the other programmed storage levels. In fixed-position progressive programming, as shown in Fig. 5, the largest memory cell threshold voltage shift is from the erased state to the second storage level. Clearly, both progressive-programming implementation strategies can reduce the largest memory cell threshold voltage shift, which leads to higher programming speed and less worst case cell-to-cell interference than conventional SLC.

We then further discuss the comparison between the two different progressive programming implementation strategies. Comparing Fig. 2 and Fig. 5, both implementation strategies have the same programming speed of the first 1-bit progressive programming within each super cycle. For the second 1-bit progressive programming, the fixed-position strategy has a higher programming speed because the constant-shift strategy results in a larger memory cell threshold voltage shift (i.e., from the erased state to the third storage level). Nevertheless, starting from the third 1-bit progressive programming, the constant-shift strategy tends to have a higher programming speed for two reasons.

1) In fixed-position progressive programming, during each 1-bit progressive programming, there are always some memory cells whose threshold voltage shifts from the erased state to the second storage level, leading to a relatively large shift. In constant-shift progressive programming, starting from the third 1-bit progressive programming, memory cell threshold voltage always shifts from the ith programmed level to the $(i+2)$th programmed level ($i > 1$), which may lead to a shift less than that of fixed-position progressive programming, as programmed states are narrower than the erased state.
2) In fixed-position progressive programming, the number of verify pulses during each program–verify iteration gradually increases, whereas constant-shift progressive programming always involves only two verify pulses except the first 1-bit constant-shift progressive programming.

Compared to constant-shift progressive programming, fixed-position progressive programming is subject to more severe cell-to-cell interference. In fixed-position progressive programming, if one memory cell stores the same value over several consecutive 1-bit programming operations, this cell will not be programmed (i.e., its threshold voltage should stay in the same level). As a result, the effects of cell-to-cell interference from its neighboring memory cells accumulate over those consecutive 1-bit programming operations, leading to gradually reduced noise margin. In the worst case, the victim cell stays in the erased state throughout the entire super cycle while the threshold voltages of all its neighboring cells always move up, and the overall cell-to-cell interference experienced by the victim cell is same as the conventional SLC memory. Therefore, large noise margin is required to accommodate such worst case cell-to-cell interference in fixed-position progressive programming. On the other hand, in constant-shift progressive programming, the threshold voltage of one memory cell at most stays only at the same storage level over two consecutive 1-bit programming operations, leading to less accumulated cell-to-cell interference effect. The worse case cell-to-cell interference in fixed-position progressive programming directly degrades certain memory system performance metrics such as endurance and retention.

2) Read Latency: Flash memory read latency is approximately proportional to the number of active storage levels per cell that must be distinguished during the read operation. To distinguish among s storage levels for all cells within one page, NAND flash memory has to carry out $s-1$ sensing iterations, and each sensing iteration targets at one sensing threshold between two adjacent active storage levels and involves bit-line/word-line charging and discharging operations. In conventional SLC and the constant-shift progressive

programming SLC flash memory, there are always only two active storage levels per cell, hence both have a similar read latency.

However, when using fixed-position progressive programming, the number of active storage levels monotonically increases within each super P/E cycle. As a result, the read latency also increases as more 1-bit progressive programming operations arecarried out within each super P/E cycle. Let the maximum number of storage levels per cell is m. In the worst case, after the last 1-bit progressive programming, we have to carry out $(m-1)$ sensing iterations to read the entire page, which results in $(m-1)$ times longer read latency than fixed-position progressive programming and conventional SLC memory. Clearly, the average read latency of 1-bit constant-shift progressive programming is $m-1/2$ times longer than the other two scenarios.

3) Implementation Overhead: Although the proposed progressive programming improves the effective endurance of SLC NAND flash memory, in the meanwhile, it incurs certain implementation overheads. First, NAND flash memory controller must keep record of sufficient run-time information in support of the progressive programming, leading to extra storage overhead. We note that the flash memory controller already needs to track the number of erase cycles for the purpose of wear-leveling and garbage collection [20], [21]. Therefore, as all the pages in each block must be programmed consecutively, we only need to keep record of: 1) the number of 1-bit progressive programming operations that are elapsed during present super P/E cycle and 2) the index of the page that is most recently programmed. As these run-time information is on a block-by-block basis, and the information associated with each block needs at most few bytes only, the total capacity of these run-time information is largely negligible. For example, let us consider a 16 GB SLC NAND flash memory chip with 4 kB page size, which contains 4000 blocks and each block contains 128 pages. Hence, we need 7 bits to record the index of the most recently programmed page within each block. Suppose the maximum number of 1-bit progressive programming operations within one super P/E cycle is represented with 3 bits. Then we need to store 10 extra bits for each block, resulting in only 5 kB data for all the 4000 blocks in total. Hence, the incurred extra storage overhead is not significant. In addition, as the controller can use embedded SRAM or an off-chip DRAM to store and update these 5 kB data during the run time, they will be written to NAND flash memory only when the storage system is powered off. During write operations, except the page data, the flash memory controller needs to transfer these 10-bit information and the number of erased operations undergone by the block to the flash memory. Compared to the large page data size that ranges from 512 B to 4 kB, the transfer of these extra few bytes do not incur noticeable data transfer overhead.

Second, progressive programming SLC NAND flash memory chips must be able to support run-time dynamic configuration of the number of storage levels per cell and the position of each storage level, which clearly complicates the memory peripheral circuit design. Because the memory still behaves as SLC, the page buffer size will not increase. During each programming operation, the 1-bit per cell page buffer contains the bit to be programmed into each cell, and the bit-line inhibition will be enabled once the verify cycle shows the match between current threshold voltage level and the bit to be programmed. During write operations, based on the received extra information sent from the controller as discussed earlier, the flash memory decides the number of allowable 1-bit write operations within each super cycle and the corresponding storage level locations. Regarding the peripheral circuits, on-chip charge pump and voltage regulator generates more different reference voltage values to support the different storage level locations. Because the maximum number of storage levels should not be too large (at most 5 or 6), the on-chip charge pump and voltage regulator should not be much more complicated than that in existing multibit per cell NAND flash memory. Hence, we expect that the impact on peripheral circuit complexity may not be significant.

IV. Simulations

For the purpose of quantitative evaluation, we develop a quantitative NAND flash memory device model that captures the major threshold voltage distortion sources described in Section II. On the basis of this model, we carry out simulations to evaluate the proposed progressive programming SLC memory design strategy and compare the above-mentioned two different implementation strategies.

A. NAND Flash Memory Device Model

1) Erase and Programming Operations: The threshold voltage of erased memory cells tends to have a wide Gaussian-like distribution [22]. Hence, we approximately model the threshold voltage distribution of erased state as

$$p_e(x) = \frac{1}{\sigma_e\sqrt{2\pi}} e^{-\frac{(x-\mu_e)^2}{2\sigma_e^2}} \tag{3}$$

where μ_e and σ_e are the mean and standard deviation of the erased state. During each program-and-verify cycle, the floating gate transistor threshold voltage is first boosted up to ΔV_{pp} and then compared to the verify reference voltage. If its threshold voltage is lower than the verify voltage, the program-and-verify recursion will continue, otherwise, the corresponding bit-line will be configured so that further programming of this cell is inhibited. At older technology nodes (e.g., 90-nm node), the threshold voltage of programmed states tends to have a uniform distribution with the width of ΔV_{pp} [12]. Nevertheless, for highly scaled technology nodes (e.g., 65 nm and below), the electron injection statistical spread [23] becomes significant, which tends to make the threshold voltage of programmed states approximately follows a Gaussian distribution. Hence, in this paper, we model the ideal distribution of each programmed state as

$$p_p(x) = \frac{1}{\sigma_p\sqrt{2\pi}} e^{-\frac{(x-\mu_p)^2}{2\sigma_p^2}} \tag{4}$$

where μ_p and σ_p are the mean and standard deviation of the programmed state. As discussed in Section II, such ideal

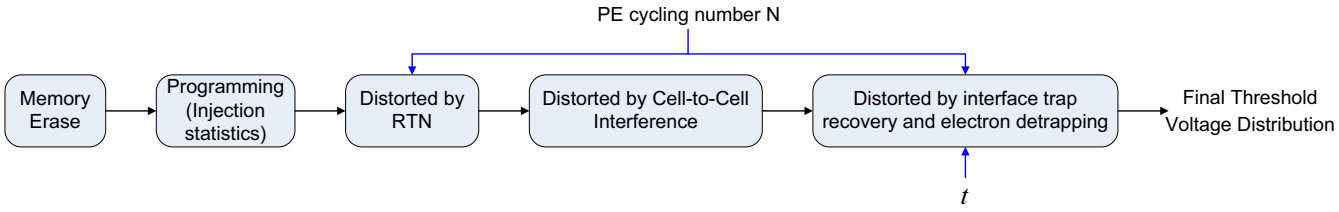


Fig. 8. Illustration of the approximate NAND flash memory device model to incorporate major threshold voltage distortion sources.

threshold voltage distribution is largely distorted mainly due to RTN, data retention, and cell-to-cell interference. Modeling of these noises is described as follows.

2) Random Telegraph Noise: RTN causes random fluctuation of memory cell threshold voltage, and we can model the probability density function $p_r(x)$ of RTN-induced threshold voltage fluctuation as a symmetric exponential function [12]

$$p_r(x) = \frac{1}{2\lambda_r} e^{-\frac{|x|}{\lambda_r}}. \tag{5}$$

As the significance of RTN is proportional to the interface state trap density, we set the mean of RTN, i.e., $\mu_{\text{RTN}} = \frac{1}{\lambda_r}$, approximately follows:

$$\mu_{\text{RTN}} = A_{\text{RTN}} \cdot N^{a_{IT}} \tag{6}$$

where a_{IT} is the exponent a in (1) for interface state traps.

3) Retention Noise: Memory cell threshold voltage reduction during data retention is mainly due to interface state trap recovery and electron detrapping, especially for technology nodes below 90 nm. Because of their Poisson statistics nature [7], we can approximately model the threshold voltage reduction as a Gaussian distribution, i.e., $p_t(x) = \mathcal{N}(\mu_d, \sigma_d^2)$. Both mean and variation are proportional to the sum of interface state traps and oxide traps (i.e., $A_t \cdot N^{a_{IT}} + B_t \cdot N^{a_{OT}}$, where the first item and second items correspond to interface state traps and oxide traps, respectively), and also proportional to the logarithm of data retention time [7], [24].

In addition, the significance of threshold voltage reduction during data retention tends to be proportional to the initial threshold voltage magnitude also [25], i.e., the higher the initial threshold voltage, the faster is the memory cell threshold voltage to reduce. Hence, we set that the generated retention noise approximately scale $K_s(x - x_0)$, where x is the initial threshold voltage, and x_0 and K_s are constants.

4) Cell-to-Cell Interference: Different NAND flash memory bit-line structures lead to different modeling of cell-to-cell interference. In current design practice, there are two different bit-line structures, including conventional even/odd bit-line structure [26], [27] and all-bit-line structure [28], [29]. In even/odd bit-line structure, memory cells on one word-line are alternatively connected to even and odd bit-lines and they are programmed at different time. As a result, an even cell is mainly interfered by five neighboring cells and an odd cell is interfered by only three neighboring cells. Cells in all-bit-line structure suffers less cell-to-cell inference than even cells in odd/even structure, and the all-bit-line structure most effectively supports high-speed current sensing to improve the memory read and verify speed. Therefore, in this paper, we only consider SLC NAND flash memory with the all-bit-line structure.

To capture inevitable process variability, we set both the vertical coupling ratio γ_y and diagonal coupling ratio γ_{xy} as random variables with bounded Gaussian distribution

$$p_c(x) = \begin{cases} \frac{c_c}{\sigma_c\sqrt{2\pi}} \cdot e^{-\frac{(x-\mu_c)^2}{2\sigma_c^2}}, & \text{if } |x - \mu_c| \le w_c \\ 0, & \text{else} \end{cases} \tag{7}$$

where μ_c and σ_c are the mean and standard deviation, and c_c is chosen to ensure the integration of this bounded Gaussian distribution equals one.

5) Overall NAND Flash Model: On the basis of the earlier discussions, we can approximately model NAND flash memory device characteristics as shown in Fig. 8, based on which we can simulate memory cell threshold voltage distribution and hence obtain memory cell raw storage reliability. According to (3) and (4), we obtain the ideal threshold voltage distribution function $p_p(x)$ right after programming. Recall that $p_{pr}(x)$ denotes the RTN distribution function [see (5)], and let $p_{ar}(x)$ denotes the threshold voltage distribution after incorporating RTN, which is obtained by convoluting $p_p(x)$ and $p_r(x)$, i.e.,

$$p_{ar}(x) = p_p(x) \bigotimes p_r(x). \tag{8}$$

The cell-to-cell interference is further incorporated based on (2). Let p_{ac} denotes the threshold voltage distribution after incorporating cell-to-cell interference and retention noise distribution is denoted as $p_t(x)$. The final threshold voltage distribution p_f is obtained as

$$p_f(x) = p_{ac}(x) \bigotimes p_t(x). \tag{9}$$

The above-presented approximate mathematical model for simulating NAND flash memory cell threshold voltage is further demonstrated using the following example.

Example 1: Let us consider single-bit per cell NAND flash memory. We set normalized σ_e and μ_e of the erased state as 0.35 and 1.4, respectively. For programmed state, we set the normalized program step-voltage ΔV_{pp} as 0.2, and its deviation as 0.05, with mean as 4.3. According to [10], the exponent a in (1) for interface state and oxide traps generation is $a_{IT} = 0.62$ and $a_{OT} = 0.3$, respectively. For RTN, we set $A_{\text{RTN}} = 1.81 \times 10^{-4}$. Regarding retention noise, we set $\sigma_d = 0.3|\mu_d|$, and $A_t = 3.5 \times 10^{-5}$ and $B_t = 2.35 \times 10^{-4}$ (these parameters are chosen to match the measurement results [10] that show the ratio of threshold voltage reduction due to interface state trap recovery and electron detrapping is 0.7:0.3). Regarding the influence of initial threshold voltage on

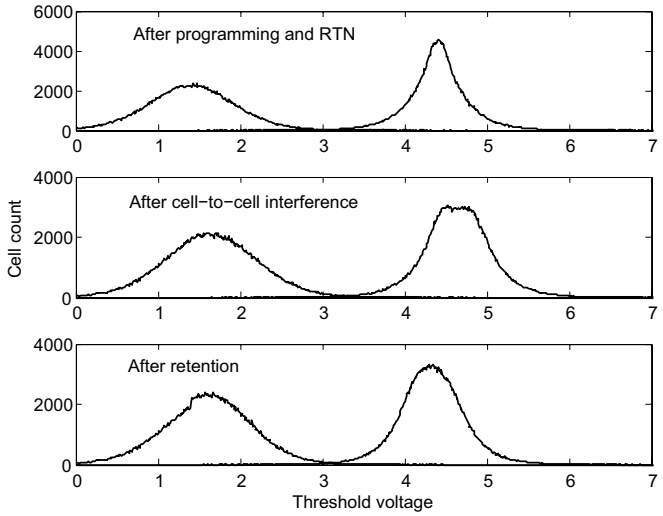


Fig. 9. Simulated results to show the effects of RTN, cell-to-cell interference, and retention on memory cell threshold voltage distribution under 10k P/E cycling with 10-year storage period.

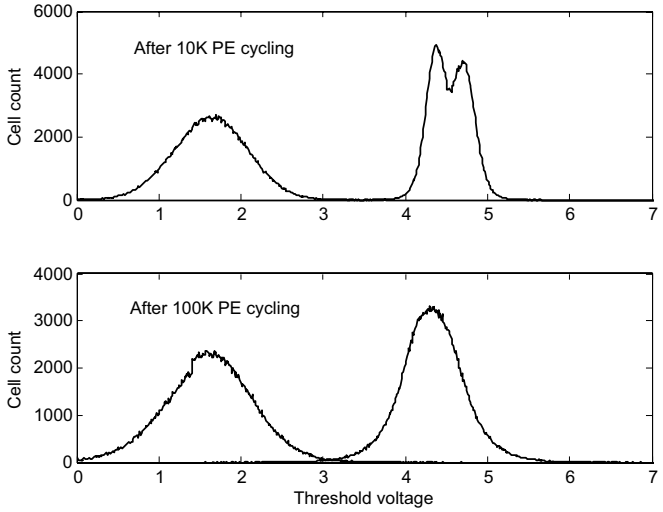


Fig. 10. Simulated threshold voltage distribution after 10 k P/E cycling with 10 years storage period and 100 k P/E cycling with 10-year storage period, which clearly shows the dynamics inherent in NAND flash memory characteristics.

retention noise, we set $x_0 = 1.4$ and $K_s = 0.333$. According to [16], [30], we set the means of γ_y and γ_{xy} as 0.12 and 0.009, respectively. For the modeling of coupling capacitance, we set $\omega_c = 0.1$ μ_c and $\sigma_c = 0.4$ μ_c. We carry out Monte Carlo simulations to obtain the cell threshold voltage distribution at different stages of the whole NAND flash model under 10k P/E cycling and after 10-year storage limit, as shown in Fig. 9. The final threshold voltage distributions under 100 P/E cycling with a one month storage period and 10k P/E cycling with 10-year storage period are both shown in Fig. 10. These results clearly show the dynamic characteristics of NAND flash memory.

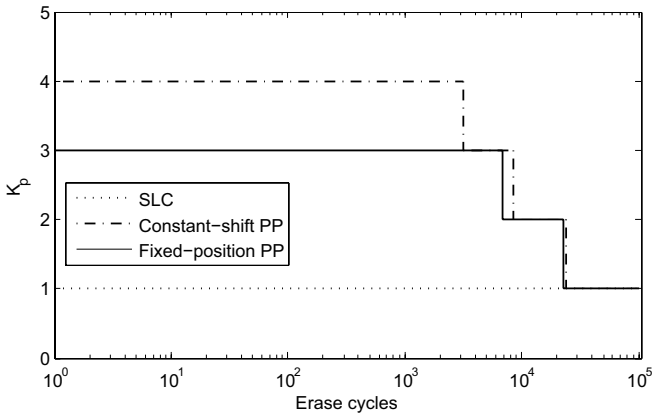


Fig. 11. The allowable number K_p of 1-bit progressive programming operations within one super P/E cycle for two progressive programming schemes under various erase cycles.

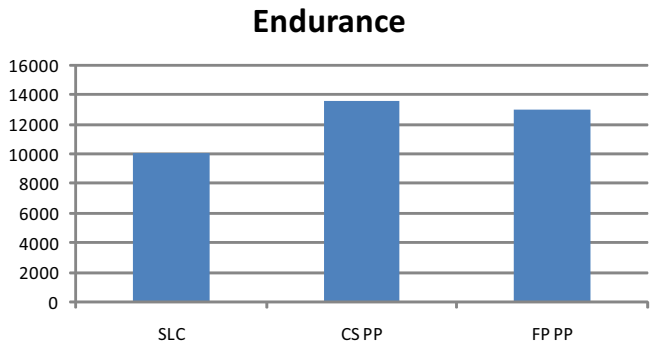


Fig. 12. Effective endurance comparison of conventional SLC, 1-bit constant-shift progressive programming and 1-bit fixed-position progressive programming.

B. Simulation Results

We use the approximate NAND flash memory device model described earlier with the same parameters used in Example 4.1 in the following simulations. In addition, we assume that the controller or flash memory chip employs the simple postcompensation [31] technique to mitigate the effect of cell-to-cell interference. To ensure a fair comparison, we assume that all the scenarios use the same programming step-voltage and have the same overall threshold voltage window. We set the page size as 4 kB and the target page failure rate as 10^{-15} after ECC decoding, and set the target P/E cycling endurance as 100 k with the data retention of 10 years. On the basis of the simulations, a binary BCH code with the code rate of 94% could achieve the target page error rate. Under such memory system configuration, we carry out extensive Monte Carlo simulations to estimate the allowable number of storage levels per cell under various P/E cycling conditions in support of progressive programming. As discussed earlier, the two different progressive programming implementation strategies subject to different worst case cell-to-cell interference. Hence, even after the use of postcompensation for mitigating cell-to-cell interference, these two different implementation strategies subject to different noise characteristics under the same P/E cycling condition, which leads to different allowable number

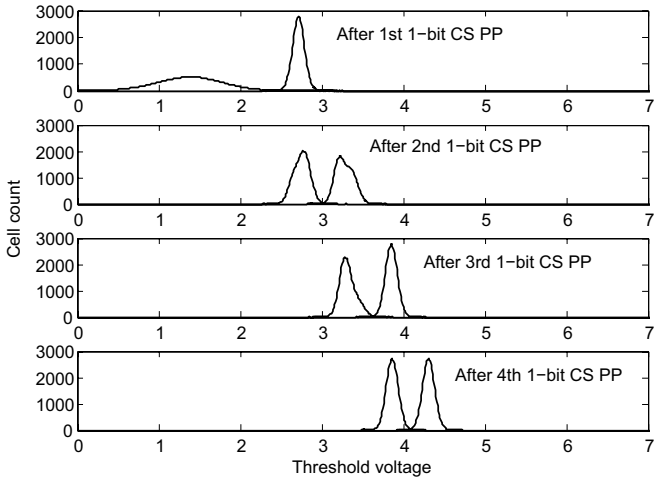


Fig. 13. Simulated threshold voltage distributions over four consecutive constant-shift progressive programming operations within one super P/E cycle, when the allowable number of storage levels is five.

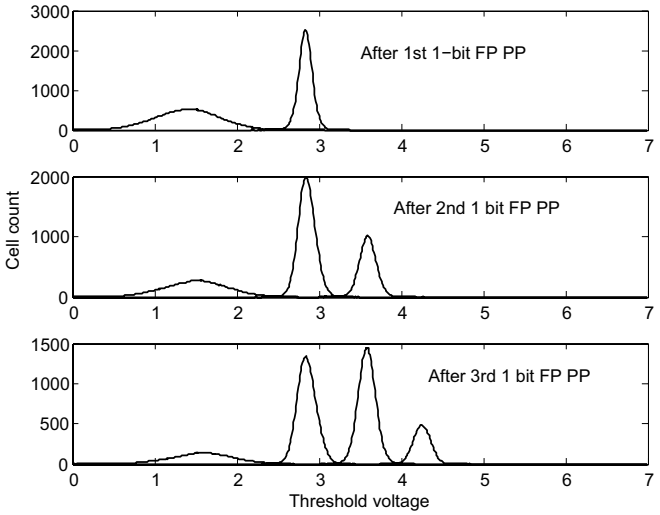


Fig. 14. Simulated threshold voltage distributions over three consecutive fixed-position progressive programming operations within one super P/E cycle, when the allowable number of storage levels is four.

of storage levels per cell. Let N_e denote the number of erase cycles that are elapsed, and K_p denote the allowable number of 1-bit progressive programming operations within one super P/E cycle. On the basis of our simulation, for constant-shift progressive programming implementation strategy, we have

$$K_p = \begin{cases} 4, & \text{if } N_e \leq 3,200 \\ 3, & \text{if } 3,200 < N_e \leq 8,500 \\ 2, & \text{if } 8500 < N_e \leq 24,200 \\ 1, & \text{if } 24,200 < N_e \leq 100,000. \end{cases}$$

For fixed-position progressive programming implementation strategy, we have

$$K_p = \begin{cases} 3, & \text{if } N_e \leq 6,900 \\ 2, & \text{if } 6,900 < N_e \leq 22,500 \\ 1, & \text{if } 22,500 < N_e \leq 100,000. \end{cases}$$

Fig. 11 shows how the value of K_p changes with the number of erase cycles under these two different progressive programming implementation strategies. Fig. 12 further illustrates the comparison among conventional SLC and these two progressive programming implementation strategies with respect to effective endurance. It shows that the constant-shift and fixed-position progressive programming improve the effective endurance by 35.9% and 29.4%, respectively.

To further illustrate the difference between constant-shift and fixed-position progressive programming implementation strategies, Fig. 13 shows the simulated threshold voltage distributions (after the use of post-compensation for cell-to-cell interference mitigation) over four consecutive constant-shift progressive programming operations within one super P/E cycle, where the allowable number of storage levels is five. Fig. 14 shows simulated threshold voltage distributions (after the use of postcompensation for cell-to-cell interference mitigation) over three consecutive fixed-position progressive programming operations within one super P/E cycle, where the allowable number of storage levels is four.

On the basis of the simulation results, we further estimate and compare programming speed and read speed of the three scenarios. According to [3], [32], [33], the program pulse and

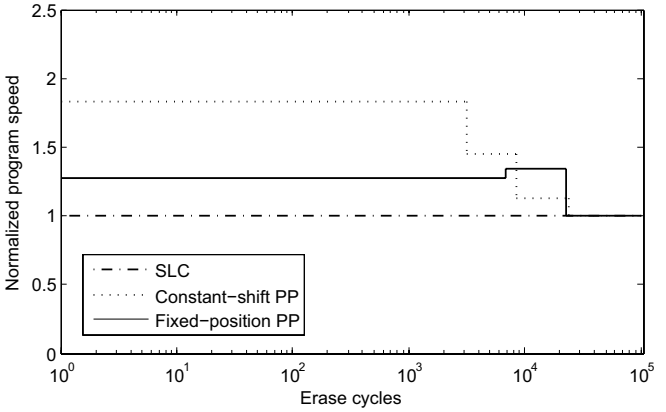


Fig. 15. Programming speed comparison of conventional SLC, constant-shift progressive programming SLC, and fixed-position progressive programming SLC under various erase cycles. The programming speed of conventional SLC is normalized as one.

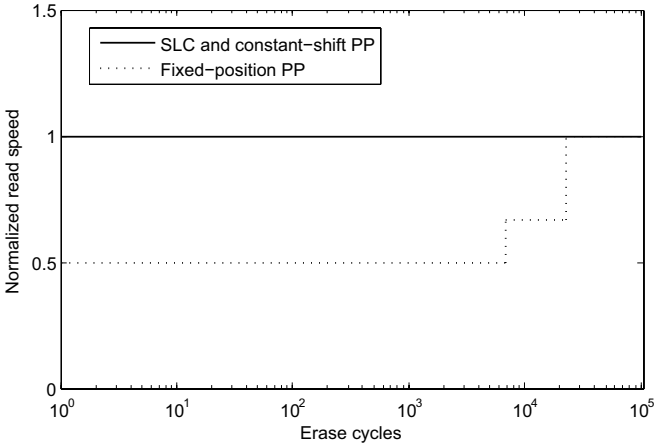


Fig. 16. Read speed comparison of conventional SLC, constant-shift progressive programming SLC, and fixed-position progressive programming SLC under various erase cycles. The read speed of conventional SLC is normalized as one.

verify pulse are set to last 20 and 8μs, respectively. Fig. 15 shows the programming speed comparison under various erase cycles, where the programming speed of conventional SLC is normalized as one and remains unchanged over the entire memory lifetime. The ratio of overall average programming speed of conventional SLC, constant-shift scheme, and fixed-position scheme is 1:1.12:1.10.

Fig. 16 shows the read speed comparison. As discussed in Section III-C2, conventional SLC and constant-shift progressive programming SLC have the same read speed over the entire memory lifetime, which is normalized as one in Fig. 16. Fixed-position progressive programming has lower read speed than the other two. The ratio of the average read speed of these three scenarios is 1:1:0.78. The above-mentioned simulations results suggest that progressive programming improves SLC memory endurance and meanwhile increase programming speed, and the constant-shift progressive programming implementation strategy should be preferred over its fixed-position counterpart.

V. Conclusion

This paper presented a simple design approach for improving SLC NAND flash memory effective endurance. As memory P/E cycling increased, NAND flash memory cell storage noise margin and hence raw storage reliability accordingly degraded. To ensure the system storage integrity, sufficiently strong memory fault tolerance must be employed to handle worst case cell storage noise margin at the end of memory P/E cycling lifetime. This made the cell storage noise margins at the early memory P/E cycling lifetime essentially under-utilized. This paper presented a progressive programming design concept to trade such under-utilized cell storage noise margin to improve effective endurance. We further present and compare two strategies for practically implementing this simple progressive programming design concept. On the basis of a flash memory device model, we carry out extensive simulations to evaluate the effectiveness of this progressive programming design approach and compare these two different implementation strategies. Results suggested that this simple progressive programming design approach is an attractive option to improve SLC NAND flash memory endurance and meanwhile improve programming speed.

References

[1] Y. Koh, "Nand flash scaling beyond 20 nm," in *Proc. IEEE Int. Memory Workshop*, May 2009, pp. 1–3.

[2] L. Chang, "Hybrid solid-state disks: Combining heterogeneous NAND flash in large SSDs," in *Proc. Asia South Pacific Design Auto. Conf.*, 2008, pp. 428–433.

[3] K.-D. Suh, B.-H. Suh, Y.-H. Um, J.-K. Kim, Y.-J. Choi, Y.-N. Koh, S.-S. Lee, S.-C. Kwon, B.-S. Choi, J.-S. Yum, J.-H. Choi, J.-R. Kim, and H.-K. Lim, "A 3.3 V 32 Mb NAND flash memory with incremental step pulse programming scheme," *IEEE J. Solid-State Circuits*, vol. 30, no. 11, pp. 1149–1156, Nov. 1995.

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

[5] P. Olivo, B. Ricco, and E. Sangiorgi, "High-field-induced voltage-dependent oxide charge," *Appl. Phys. Lett.*, vol. 48, no. 17, pp. 1135–1137, 1986.

[6] P. Cappelletti, R. Bez, D. Cantarelli, and L. Fratin, "Failure mechanisms of flash cell in program/erase cycling," in *IEDM Tech. Dig.*, Dec. 1994, pp. 291–294.

[7] N. Mielke, H. Belgal, I. Kalastirsky, P. Kalavade, A. Kurtz, Q. Meng, N. Righos, and J. Wu, "Flash EEPROM threshold instabilities due to charge trapping during program/erase cycling," *IEEE Trans. Device Mater. Rel.*, vol. 4, no. 3, pp. 335–344, Sep. 2004.

[8] J. B. Yang, T. P. Chen, S. S. Tan, and L. Chan, "Analytical reaction-diffusion model and the modeling of nitrogen-enhanced negative bias temperature instability," *Appl. Phys. Lett.*, vol. 88, no. 17, pp. 172109-1–172109-3, 2006.

[9] S. Ogawa and N. Shiono, "Generalized diffusion-reaction model for the low-field charge-buildup instability at the Si-SiO_2 interface," *Phys. Rev. B*, vol. 51, no. 7, pp. 4218–4230, 1995.

[10] H. Yang, H. Kim, S.-I. Park, J. Kim, S.-H. Lee, J.-K. Choi, D. Hwang, C. Kim, M. Park, K.-H. Lee, Y.-K. Park, J. K. Shin, and J.-T. Kong, "Reliability issues and models of sub-90 nm NAND flash memory cells," in *Proc. 8th Int. Conf. Solid-State Integr. Circuit Technol.*, Oct. 2006, pp. 760–762.

[11] K. Fukuda, Y. Shimizu, K. Amemiya, M. Kamoshida, and C. Hu, "Random telegraph noise in flash memories—Model and technology scaling," in *Proc. IEEE IEDM*, Dec. 2007, pp. 169–172.

[12] C. Compagnoni, M. Ghidotti, A. Lacaita, A. Spinelli, and A. Visconti, "Random telegraph noise effect on the programmed threshold-voltage distribution of flash memories," *IEEE Electron Device Lett.*, vol. 30, no. 9, pp. 984–986, Sep. 2009.

[13] N. Mielke, H. Belgal, A. Fazio, Q. Meng, and N. Righos, "Recovery effects in the distributed cycling of flash memories," in *Proc. IEEE Int. Rel. Phys. Symp.*, Mar. 2006, pp. 29–35.

[14] J.-D. Lee, S.-H. Hur, and J.-D. Choi, "Effects of floating-gate interference on NAND flash memory cell operation," *IEEE Electron Device Lett.*, vol. 23, no. 5, pp. 264–266, May 2002.

[15] K. Kinam, "Future memory technology: Challenges and opportunities," in *Proc. Int. Symp. VLSI Technol., Syst. Appl.*, Apr. 2008, pp. 5–9.

[16] K. Prall, "Scaling non-volatile memory below 30 nm," in *Proc. IEEE 2nd Non-Volatile Semicond. Memory Workshop*, Aug. 2007, pp. 5–10.

[17] H. Liu, S. Groothuis, C. Mouli, J. Li, K. Parat, and T. Krishnamohan, "3D simulation study of cell-cell interference in advanced NAND flash memory," in *Proc. IEEE Workshop Microelectron. Electron Devices*, Apr. 2009, pp. 1–3.

[18] D. Dumin, S. Mopuri, S. Vanchinathan, R. Scott, R. Subramoniam, and T. Lewis, "High field related thin oxide wearout and breakdown," *IEEE Trans. Electron Devices*, vol. 42, no. 4, pp. 760–772, Apr. 1995.

[19] J.-D. Lee, J.-H. Choi, D. Park, and K. Kim, "Degradation of tunnel oxide by FN current stress and its effects on data retention characteristics of 90 nm NAND flash memory cells," in *Proc. IEEE Int. Rel. Phys. Symp.*, Mar.–Apr. 2003, pp. 497–501.

[20] A. B. Aroya and S. Toledo, "Competitive analysis of flash-memory algorithms," in *Proc. Annu. Eur. Symp.*, 2006, pp. 100–111.

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

[22] K. Takeuchi, T. Tanaka, and H. Nakamura, "A double-level-V_{th} select gate array architecture for multilevel NAND flash memories," *IEEE J. Solid-State Circuits*, vol. 31, no. 4, pp. 602–609, Apr. 1996.

[23] C. Compagnoni, A. Spinelli, R. Gusmeroli, A. Lacaita, S. Beltrami, A. Ghetti, and A. Visconti, "First evidence for injection statistics accuracy limitations in NAND flash constant-current Fowler-Nordheim programming," in *Proc. IEEE IEDM*, Dec. 2007, pp. 165–168.

[24] C. M. Compagnoni, C. Miccoli, R. Mottadelli., S. Beltrami, M. Ghidotti., A. L. Lacaita, A. S. Spinelli, and A. Visconti, "Investigation of the threshold voltage instability after distributed cycling in nanoscale nand flash memory arrays," in *Proc. IEEE Int. Rel. Phys. Symp.*, May 2010, pp. 604–610.

[25] J. Lee, J. Choi, D. Park, K. Kim, R. Center, S. Co, and S. Gyunggi-Do, "Effects of interface trap generation and annihilation on the data retention characteristics of flash memory cells," *IEEE Trans. Device Mater. Rel.*, vol. 4, no. 1, pp. 110–117, Mar. 2004.

[26] K. Taueuchi, Y. Kameda, S. Fujimura, H. Otake, K. Hosono, H. Shiga, Y. Watanabe, T. Futatsuyama, Y. Shindo, M. Kojima, M. Iwai, M. Shirakawa, M. Ichige, K. Hatakeyama, S. Tanaka, T. Kamei, J. Y. Fu, A. Cernea, Y. Li, M. Higashitani, G. Hemink, S. Sato, K. Oowada, S.-C. Lee, N. Hayashida, J. Wan, J. Lutze, S. Tsao, M. Mofidi, K. Sakurai, N. Tokiwa, H. Waki, Y. Nozawa, K. Kanazawa, and S. Ohshima, "A 56-nm CMOS 99-mm^2 8-Gb multi-level NAND flash memory with 10-mb/s program throughput," *IEEE J. Solid-State Circuits*, vol. 42, no. 1, pp. 219–232, Jan. 2007.

[27] K.-T. Park, M. Kang, D. Kim, S.-W. Hwang, B. Y. Choi, Y.-T. Lee, C. Kim, and K. Kim, "A zeroing cell-to-cell interference page architecture with temporary LSB storing and parallel MSB program scheme for MLC NAND flash memories," *IEEE J. Solid-State Circuits*, vol. 43, no. 4, pp. 919–928, Apr. 2008.

[28] Y. Li, S. Lee, Y. Fong, F. Pan, T.-C. Kuo, J. Park, T. Samaddar, H. Nguyen, M. Mui, K. Htoo, T. Kamei, M. Higashitani, E. Yero, G. Kwon, P. Kliza, J. Wan, T. Kaneko, H. Maejima, H. Shiga, M. Hamada, N. Fujita, K. Kanebako, E. Tarn, A. Koh, I. Lu, C. Kuo, T. Pham, J. Huynh, Q. Nguyen, H. Chibvongodze, M. Watanabe, K. Oowada, G. Shah, B. Woo, R. Gao, J. Chan, J. Lan, P. Hong, L. Peng, D. Das, D. Ghosh, V. Kalluru, S. Kulkarni, R. Cernea, S. Huynh, D. Pantelakis, C.-M. Wang, and K. Quader, "A 16 Gb 3b/cell NAND flash memory in 56 nm with 8MB/s write rate," in *IEEE Int. Solid-State Circuits Conf., Dig. Tech. Papers*, Feb. 2008, pp. 506–632.

[29] R.-A. Cerna, L. Pham, F. Moogat, S. Chan, B. Le, Y. Li, S. Tsao, T.-Y. Tseng, K. Nguyen, J. Li, J. Hu, J. H. Yuh, C. Hsu, F. Zhang, T. Kamei, H. Nasu, P. Kliza, K. Htoo, J. Lutze, Y. Dong, M. Higashitani, J. Yang, H.-S. Lin, V. Sakhamuri, A. Li, F. Pan, S. Yadala, S. Taigor, K. Pradhan, J. Lan, J. Chan, T. Abe, Y. Fukuda, H. Mukai, K. Kawakami, C. Liang, T. Ip, S.-F. Chang, J. Lakshmipathi, S. Huynh, D. Pantelakis, M. Mofidi, and K. Quader, "A 34 MB/s MLC write throughput 16 Gb NAND with all bit line architecture on 56 nm technology," *IEEE J. Solid-State Circuits*, vol. 44, no. 1, pp. 186–194, Jan. 2009.

[30] N. Shibata, H. Maejima, K. Isobe, K. Iwasa, M. Nakagawa, M. Fujiu, T. Shimizu, M. Honma, S. Hoshi, T. Kawaai, K. Kanebako, S. Yoshikawa, H. Tabata, A. Inoue, T. Takahashi, T. Shano, Y. Komatsu, K. Nagaba, M. Kosakai, N. Motohashi, K. Kanazawa, K. Imamiya, and H. Nakai, "A 70 nm 16 Gb 16-level-cell NAND flash memory," in *Proc. IEEE Symp. VLSI Circuits*, Jun. 2007, pp. 190–191.

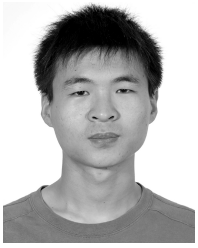
[31] G. Dong, S. Li, and T. Zhang, "Using data postcompensation and predistortion to tolerate cell-to-cell interference in MLC NAND flash memory," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 10, pp. 2718–2728, Oct. 2010.

[32] 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.

[33] C. Lee, S. Lee, S. Ahn, J. Lee, W. Park, Y. Cho, C. Jang, C. Yang, S. Chung, I. Yun, B. Joo, B. Jeong, J. Kim, J. Kwon, H. Jin, Y. Noh, J. Ha, M. Sung, D. Choi, S. Kim, J. Choi, T. Jeon, H. Park, J.-S. Yang, and Y.-H. Koh, "A 32-Gb MLC NAND flash memory with Vth endurance enhancing schemes in 32 nm CMOS," *IEEE J. Solid-State Circuits*, vol. 46, no. 1, pp. 97–106, Jan. 2011.

Guiqiang Dong (S'09) received the B.S. and M.S. degrees from the University of Science and Technology of China, Hefei, China, in 2004 and 2008, respectively, and the Ph.D. degree from the Electrical, Computer and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, NY, USA, in 2012.

He is currently a Chief Channel Architect with Skyera, Inc., San Jose, CA, USA. He has discovered a novel method by successfully building a software tool to rapidly estimate the error floor of low-density parity-check codes. His current research interests include coding theory, NAND Flash memory, error correction code and signal processing application for NAND Flash SSD, and firmware design for NAND Flash SSD.



Yangyang Pan (S'12) received the B.S. degree in electrical engineering from Zhejiang University, Zhejiang, China, in 2007, and the Ph.D. degree from the Electrical, Computer and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, NY, USA, in 2012.

He is currently a Non-Volatile Memory Technologist with Fusion-io, San Jose, CA, USA. His current research interests include architecture for high performance storage systems and signal processing for SSD systems.

Tong Zhang (M'02–SM'08) received the B.S. and M.S. degrees in electrical engineering from Xi'an Jiaotong University, Xi'an, China, in 1995 and 1998, respectively, and the the Ph.D. degree in electrical engineering from the University of Minnesota, Minneapolis, MN, USA, in 2002.

He is currently a Professor with the Electrical, Computer and Systems Engineering Department, Rensselaer Polytechnic Institute, Troy, NY, USA. His current research interests include circuits and systems for memory and data storage, computing, and signal processing.

Dr. Zhang has served an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS: PART II—BRIEF PAPERS and the IEEE TRANSACTIONS ON SIGNAL PROCESSING.