ReveNAND: A Fast-Drift-Aware Resilient 3D NAND Flash Design

MUSTAFA M. SHIHAB, University of Texas at Dallas
 JIE ZHANG and MYOUNGSOO JUNG, Yonsei University
 MAHMUT KANDEMIR, Pennsylvania State University

The paradigm shift from planar (two dimensional (2D)) to vertical (three-dimensional (3D)) models has placed the NAND flash technology on the verge of a design evolution that can handle the demands of next-generation storage applications. However, it also introduces challenges that may obstruct the realization of such 3D NAND flash. Specifically, we observed that the fast threshold drift (fast-drift) in a charge-trap flash-based 3D NAND cell can make it lose a critical fraction of the stored charge relatively soon after programming and generate errors.

In this work, we first present an elastic read reference (V_{Ref}) scheme (ERR) for reducing such errors in ReveNAND—our fast-drift aware 3D NAND design. To address the inherent limitation of the adaptive V_{Ref} , we introduce a new intra-block page organization (hitch-hike) that can enable stronger error correction for the error-prone pages. In addition, we propose a novel reinforcement-learning-based smart data refill scheme (iRefill) to counter the impact of fast-drift with minimum performance and hardware overhead. Finally, we present the first analytic model to characterize fast-drift and evaluate its system-level impact. Our results show that, compared to conventional 3D NAND design, our ReveNAND can reduce fast-drift errors by 87%, on average, and can lower the ECC latency and energy overheads by 13× and 10×, respectively.

CCS Concepts: • **Hardware** → **Non-volatile memory**; **Memory and dense storage**; **Failure recovery, maintenance and self-repair**; • **Information systems** → **Flash memory**;

Additional Key Words and Phrases: Fast detrapping, threshold drift, adaptive read, reinforcement learning

ACM Reference format:

Mustafa M. Shihab, Jie Zhang, Myoungsoo Jung, and Mahmut Kandemir. 2018. ReveNAND: A Fast-Drift-Aware Resilient 3D NAND Flash Design. *ACM Trans. Archit. Code Optim.* 15, 2, Article 17 (April 2018), 26 pages.

<https://doi.org/10.1145/3184744>

1 INTRODUCTION

Since the mid-1990w, NAND flash has changed the perception of data storage, thanks to its diverse and successful incarnations as the preferred storage medium for most computing domains, from

This work was supported in part by NRF grants 2015M3C4A7065645 and 2016R1C1B2015312, DOE grant DE-AC02-05CH11231, MSIP grant IITP-2017-2017-0-01015, and MemRay grant 2015-11-1731. M. Kandemir's research was partly supported by NSF grants 1302557, 1213052, 1439021, 1302225, 1629129, 1526750, and 1629915 and a grant from Intel.

Authors' addresses: M. M. Shihab, Department of Electrical and Computer Engineering, University of Texas at Dallas; email: mustafa.shihab@utdallas.edu; J. Zhang and M. Jung (corresponding author), School of Integrated Technology, Yonsei University; emails: {jie, m.jung}@yonsei.ac.kr; M. Kandemir, Department of CSE, Pennsylvania State University; email: kandemir@cse.psu.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 ACM 1544-3566/2018/04-ART17 \$15.00

<https://doi.org/10.1145/3184744>

Fig. 1. Impact of fast-drift in 3D NAND flash.

low-power mobile devices to high-performance computing [23, 32, 38]. However, scalability has become a critical limitation that can impede further progress of the two-dimensional (2D) NAND flash designs. Specifically, considering “cell-level” issues such as insufficient number of electrons in the substrate, excessive cell-to-cell interference, and prohibitively expensive fabrication process, the general consensus is that 2D NAND cannot scale beyond the 1z nm process under reasonable assumptions [12]. To overcome these limitations, several designs have been proposed to stack flash cells in the “vertical direction” and expand storage capacity by constructing a three-dimensional (3D) NAND flash array [15, 17, 26, 39].

However, with very limited publicly available information, 3D NAND is still an uncharted territory that presents a set of unique challenges not observed in 2D NAND flash. One of the key differences between 2D NAND and 3D NAND is that 3D NAND replaces conventional *floating-gate-based flash* (FG-flash) cells with *charge-trap-based flash* (CT-flash) cells as the storage core for simplifying the vertical fabrication process [33]. Specifically, the conductive polysilicon *charge-storage layer* (CSL) requires a layer of insulator oxide between the adjacent FG-flash cells to prevent unwanted charge loss [21]. However, in 3D NAND fabrication, such *etching* of the CSL and horizontal *deposition* of oxide in each cell is practically impossible. In contrast, CT-flash cells do not require such isolation due to the non-conductive silicon nitride CSL, which in turn makes 3D NAND fabrication considerably less complicated.

Unfortunately, we observed that CT-flash-based 3D NAND cells can lose a critical fraction of the stored charge relatively soon after a program, undermining both reliability and performance. Figure 1(a) presents the impact of such retention errors in terms of bit error rates (BER) in 3D NAND (CT-flash) normalized to that in 2D NAND (FG-flash). In this figure, the 2D NAND curve represents the well-known trend for FG-Flash devices [35], while the 3D NAND data are derived using our proposed analytic model for similar process technology and parameters and verified against available data [7, 9, 16]. One can notice from the figure that while 2D NAND starts to suffer from high BER only near the end of its retention period (~ 10 years), 3D NAND can experience around 70% of the peak BER only months after a program. This happens because 3D NAND shows a sharp drift in threshold voltage relatively soon after a program due to fast-detrapping of charges, referred to as *fast-drift* [7, 22, 29]. A natural response to such higher error rate could be to employ a stronger error-correcting code (ECC) scheme for 3D NAND. Unfortunately, the energy and latency overheads of such mechanism would be overwhelming, as the ECC overheads increase super-linearly with error rate. Figure 1(b) shows the latency and energy overheads for a 3D NAND system employing a low-density parity-check (LDPC) [10, 19] scheme and considering

Fig. 2. Repeated charge-refills can minimize fast-drift but introduces unreasonable overheads.

the additional errors that 3D NAND can experience due to fast-drift. One can observe from the projections that the ECC latency and energy are approximately $16\times$ and $12\times$ higher, respectively, compared to 2D NAND. Therefore, while 3D NAND can suffer from severe reliability problems without effective measures against fast-drift, a naive, brute-force attempt to correct the errors can instead hurt the system performance.

In addition, Lue et al. [27] demonstrated that repeated in-place programming on CT-flash cells can refill the depleted charge and gradually diminish the impact of fast-drift. As shown in Figure 2(a), our array-level simulation results confirm that three extra charge-refill operations after a write can indeed slow-down fast-drift sufficiently to ensure storage-class data retention. Unfortunately, we observed that such refill operations exceedingly amplify the overheads of each program operation, as shown in Figure 2(b). Specifically, in a triple-level cell (TLC) NAND flash, the program latency and energy can increase by up to $9\times$ and $15\times$, respectively. Therefore, naively scheduling refill operations in 3D NAND can render it unattractive for high-performance and low-power applications.

In this work, we first propose our *elastic read reference scheme (ERR)*, which can dynamically adjust the V_{Ref} to correctly read data from a fast-drift affected 3D NAND. We want to emphasize that ERR is a low-level corrective strategy to reduce errors and is orthogonal to system-level reliability solutions. We then introduce *hitch-hike*, a modified page organization for 3D NAND blocks, that utilizes dedicated pages to store error correction bits for the pages that are expected to face the highest number of errors. Third, we propose a novel reinforcement-learning-based *intelligent charge-refill scheme (iRefill)* to diminish the impact of fast-drift with a minimum number of refill operations. Finally, to the best of our knowledge, we present the first analytic model for fast-drift in CT-flash that uncovers the impact of fast-drift on the 3D NAND flash reliability and leads to the solutions that can secure reliable, storage-class retention for future 3D NAND flash devices. Our main contributions can be summarized as follows:

- *Investigating a fast-drift aware V_{Ref} mechanism.* We observe that fast-drift can pull down the threshold voltage (V_{Th}) below the read reference voltage (V_{Ref}) and increase the error rate in 3D NAND significantly. To counter this, our proposed ERR scheme predicts the amount of fast-drift using our novel analytic model and adjusts the V_{Ref} at runtime to compensate for the V_{Th} drift.
- *Exploiting page organization to support stronger ECC.* We note that it is difficult to provide sufficiently strong ECC for the whole 3D NAND device, as storing the error correction bits

Fig. 3. Floating-Gate and Charge-Trap NAND flash cell structure (a), 3D NAND design (b), and detailed view of a 3D NAND string (c).

for such ECC may not be feasible in existing designs. We propose a scheme, called *hitch-hike*, that assigns a small fraction of the pages in a block for storing the long error correction bits on behalf of the error-prone pages and can significantly increase reliability of 3D NAND.

- *Utilizing reinforcement-learning for charge-refill.* Our observations indicate that while charge-refill can slow down fast-drift, it can also take a significant toll on 3D NAND performance by increasing write¹ overheads. Our proposed *iRefill* strategy addresses this problem by tracking fast-drift at a block-level granularity and by learning to schedule the minimum number of refills to ensure reliable data retention. By using our custom reinforcement learning algorithm, errors can be minimized while maintaining performance.
- *Modeling and impact analysis of fast-drift.* We present a detailed analytic model for fast-drift in 3D NAND based on a set of critical design parameters (e.g., threshold voltage) and environmental factors (e.g., temperature) and explore our model with varying values of these parameters. Our proposed model is the first that can predict the magnitude of fast-drift for a chosen technology and configuration for both 2D and 3D CT-flash-based NAND, and as such it can be used as a baseline for future work in this field.

Our experiments show that with our proposed counter-measures, ReveNAND can reduce errors from fast-drift by 87% on average. As a result, compared to the state of the art, it can lower the ECC latency and energy overheads by 13× and 10×, respectively.

The rest of the article is organized as follows: Section 2 summarizes the background concepts considered in this work. Sections 3 introduces our proposed counter-measures against fast-drift, and Section 5 describes their implementation details. Section 6 discusses results of our experiments. Section 8 encapsulates the prior studies related to this work. Finally, Section 9 concludes the article.

2 BACKGROUND

2.1 Floating-Gate vs. Charge-Trap Flash

The NAND flash cell is divided into multiple layers that are used for data storage and control purposes. Specifically, the *charge storage layer* (CSL) works as the storage core, while the control gate is used for managing cell operation (i.e., read, write, or idle). The tunnel-oxide and the buffer oxide are there to prevent charge leakage from the core to the channel. Figure 3(a) compares the cell structures for FG-flash and CT-flash. FG-flash has a polysilicon CSL. Since polysilicon is a conductor, any defect in the tunnel-oxide allows the stored charge to leak out. Consequently, the tunnel-oxide needs to be relatively thick, which in turn limits the scalability of FG-flash. On the other hand, CT-flash uses a non-conductive silicone nitride CSL (Figure 3(a)). This allows for better tolerance to oxide defects, as only a fraction of the stored charge can actually escape. Consequently,

¹In this work, we use the terms *program* and *write* interchangeably.

Fig. 4. 3D NAND fabrication process.

CT-flash can afford a much thinner tunnel-oxide and scale down to small process technologies [21]. In addition, the gate-oxide and tunnel-oxide are redesigned with high-K oxide and an oxide-nitride-oxide compound, respectively, to enhance efficiency and reduce program voltage [28].

2.2 3D NAND Flash

2.2.1 Vertically Stacked Structure. While multiple designs have been proposed for 3D NAND [17, 26, 39], the CT-flash-based terabit cell array transistor (TCAT) [15, 24, 33] has been the popular choice for real products. Figure 3(b) depicts the TCAT design, where flash cells are vertically fabricated in cylindrical shapes known as *strings*, and storage capacity can be increased by stacking more layers. At each layer, cells are organized into rows and columns, where the wordlines (WL) connect all the cells in a row *page*, and the bitlines (BL) connect all the cells in a column. The string select (SSL), drain select (DSL), and the ground select (GSL) lines connect to the peripheral network.

It is critical to make couple of observations from Figure 3(c). First, all the cells in a 3D NAND string share the continuous oxide-nitride-oxide structure among them, which allows the neighboring cells to modulate each other and can cause more charges to be shallow-trapped in 3D NAND (i.e., compared to the planar design). Second, because of the gate-spacer-gate design, individual cell gates in 3D NAND can have reduced control over the stored charges, resulting in lateral charge loss between cells [22, 29]. These two factors in combination render 3D NAND particularly vulnerable to fast-drift.

2.2.2 Fabrication Process and Use of CT-Flash. Compared to 2D NAND, the fabrication process is more complex in 3D NAND due to its vertical structure. Figure 4 shows the step-by-step process of fabricating 3D NAND [13]. First (❶), interleaved layers of oxide and polysilicon are horizontally deposited on the silicon substrate, and a hole is etched from the top oxide layer to the top of the substrate. The wall of the hole is first deposited with gate-oxide (❷) and then with a layer of silicon nitride (❸). Finally, the tunnel-oxide is deposited on the nitride layer, and the remaining space in the hole is filled with polysilicon channel (❹).

Most 3D NAND designs replace FG-flash with CT-flash because of this vertical fabrication process. Specifically, the FG-flash requires the CSLs of the adjacent cells to be kept isolated. However, the CSLs in 3D NAND are deposited vertically, like layers of paint, preceded and followed by two layers of oxide deposition (❷, ❸, ❹). Consequently, *any horizontal etching and deposition of materials at each layer of each string is impractical*. In contrast, the CSLs of CT-flash does *not* require such isolation, making it preferable for 3D NAND designs. However, using CT-flash also exposes the 3D NAND to fast-drift and loss of data.

2.3 Fast-Detrapping

The fast-detrapping problem comes with the use of CT-flash in 3D NAND. Since the CSL is an insulator, during a program operation, not all of the injected electrons are plunged deep inside it.

Fig. 5. The fast-detrapping phenomenon.

Table 1. ECB Overhead vs. Erroneous Bits Corrected

Bits corrected	Data size	ECB per page	Total ECB overhead for a 256-page block
1	4KB	16	512B
16		241	7.7KB
64		961	30.7KB
128		1,921	61.5KB
256		3,841	123KB
512		7,681	246KB

In fact, a large fraction of the electrons are shallowly trapped along the tunnel oxide-CSL boundary, as shown in Figure 5(a). These shallow-trapped electrons can escape the CSL soon after a program completes and cause the threshold voltage (V_{Th}) to drift [7, 27]. As shown in Figure 5(b), this V_{Th} drift can actually spread beyond the threshold reference voltage V_{Ref} and generate error.

2.4 Overhead of Error Correction Bits

Conventional ECC schemes used in memory/storage systems encode data bits with error correction bits (ECBs) to detect and correct a certain number of errors [11]. The number of ECBs depends primarily on the size of the data and the number of errors to be corrected. For large data, detecting and correcting a high number of errors results in a high ECB overhead. As NAND flash reads/writes in terms of pages, the ECC has to be applied to each page. Table 1 portrays the increase in ECB overhead for a typical 3D NAND with 4KB-pages and 256-page blocks. From our steady-state evaluations, 3D NAND encounters $3\times$ – $5\times$ more errors, compared to a modern 2D NAND flash. Consequently, 3D NAND designs should be able to correct 256–512 bits of error. Note that the large ECB overhead for correcting such high number of errors can be unfeasible to be accommodated as an external resource. However, in terms of pages, the ECB overhead for 512-bit correction is approximately 62 pages, which is only about 20% of a block's capacity. We later exploit this observation in our hitch-hike scheme.

2.5 Reinforcement Learning

Reinforcement learning (RL) is a machine-learning [8] strategy focusing on learning about an environment and improving over time to maximize some notion of reward [37]. Specifically, an autonomous RL agent interacts with its environment over discrete timesteps, senses the current state (S) of its environment, and executes an action (A) that produces a reward (R), as shown in Figure 6(a). The goal is to maximize the cumulative reward by learning an optimal policy for mapping states to actions, and revise a state-action-reward database (Figure 6(b)) [14, 30, 34]. The

Fig. 6. Reinforcement-learning process.

Fig. 7. Difference in V_{Th} drift behavior: 2D NAND vs. 3D NAND.

optimum policy is quantified with a parameter called the Q-value. The highest Q-value represents the optimum policy, and it is updated using the *State-Action-Reward-State-Action* (SARSA) rule [37],

$$Q(S_P, A_P) = (1 - \alpha)Q(S_P, A_P) + \alpha[R + \gamma Q(S_C, A_C)], \quad (1)$$

where (S_P, A_P) and (S_C, A_C) are the previous and current state and action pairs, respectively. Also, γ and α are constants for the convergence of rewards. In this work, we chose RL to develop our smart data refill scheme, because it can simultaneously learn and execute the “best” action based on its current knowledge.

3 COUNTER-MEASURES FOR FAST-DRIFT

Reliable memory operations depend on the flash cell’s capacity to retain the programmed V_{Th} , which can decrease if the cell loses charge from CSL. When V_{Th} drifts beyond the threshold reference voltage (V_{Ref}), a *read* fails to retrieve data correctly and generates an error. As shown in Figure 7, the stress from the program/erase cycles makes the conventional 2D NAND (FG-flash) cell gradually lose charge over its retention period. On the other hand, 3D NAND demonstrates a quick drop in V_{Th} relatively soon after a write/program operation. Consequently, although 2D NANDs need to focus on long-term reliability issues such as garbage-collection and wear-leveling, resolving fast-drift is an immediate concern for CT-flash-based 3D-NAND to avoid data loss. Furthermore, since this V_{Th} drop is caused by shallow-trapped electrons escaping the CSL of CT-flash, the existing retention improvement techniques (e.g., wear-leveling [6]) are *not* effective against fast-drift. In the following sections, we propose three novel measures to counter the fast V_{Th} drift and the resulting errors.

Fig. 8. Countering fast-drift with ERR.

Table 2. ERR Time-Bins and Corresponding Fast-Drift-Aware V_{Ref} Assignment

Time-bins for estimated fast-drift (days)	ERR assigned V_{Ref-FD} (V) (for 50°C)	$ y $ $= V_{Ref-FD} - V_{Ref-Initial}$
$t_{FD} < 0.01$	$ x = 0.425$	0
$0.01 < t_{FD} < 1$	$ x = 0.45$	0.025
$1 < t_{FD} < 100$	$ x = 0.475$	0.050
$t_{FD} > 100$	$ x = 0.5$	0.075

3.1 Elastic Read Reference

3.1.1 Overview. In practice, the flash controller executes a fixed V_{Ref} for the whole memory [3], which only considers the long-term V_{Th} drift of 2D NAND. Unfortunately, we observed that such fixed V_{Ref} scheme cannot adjust with the fast-drift to correctly read the victim pages. To address this, we propose a fast-drift-aware elastic read reference voltage (ERR), a lightweight low-level scheme for managing V_{Ref} . We observe that since the fast-drift varies with the elapsed time between writing a page and reading it, if the corresponding V_{Ref} is also adjusted proportionally, then we can correctly read the page affected by fast-drift. Motivated by this, we propose to maintain a set of V_{Ref} s, and dynamically assign a V_{Ref} to each page read, based on the time that page is last written. Figure 8 illustrates how our proposed ERR counters the fast-drift, and its operation can be explained as follows. First, the flash controller marks the time of a read request as the *read-time* (t_{RD}). Second, the controller then uses the time-stamp for the latest write on that page as the *write-time* (t_{WR}). Third, the effective elapsed time for fast-drift is *fast-drift time* (t_{FD}) = $t_{RD} - t_{WR}$. Fourth, the amount of fast-drift to be experienced by the target page is calculated using our analytic model (we present this in Section 5.1), and a suitable V_{Ref-FD} is assigned accordingly.

To efficiently execute ERR, we divide the retention period for all pages in the memory into five *bins* based on time passed after the last operation and assign a suitable V_{Ref-FD} for each bin. When selecting V_{Ref-FD} , we look for a minimum voltage level that can correct the maximum number of pages in its bin without conflicting with the adjacent memory state. Also, for the sake of simplicity, we have considered an operating temperature of 50°C. While this should be a good approximation for the typical range of operation for memory devices (27°C to 70°C), the ERR scheme can be easily modified to adjust with variable temperature. Finally, ERR does not need to consider aging/wear-out of NAND flash, as unlike long-term retention loss, fast-drift has not been observed to be affected by it. This is because, while conventional retention loss in NAND flash happens due to residual charge-traps built up from the repeated program-erase cycles, fast-drift occurs due to de-trapping of shallow-trapped charges in the charge-storage layer of CT-flash.

Table 2 summarizes the *time-bins* (in terms of days) used in our evaluation and the corresponding V_{Ref-FD} values expressed in terms of their absolute difference ($|x|$) from the initially programmed

threshold voltage ($V_{Th_Initial}$). Finally, the parameter $|y|$ is the difference between the ERR assigned V_{Ref-FD} and the $V_{Ref-Initial}$, where the latter represents the traditional static V_{Ref} value.

3.1.2 Novelty. We do NOT claim our proposed ERR scheme to be an general advancement over the prior work on adaptive read reference voltage. In fact, the novelty of our ERR scheme is rooted not in the idea of adjusting the read reference voltage but rather in its design, implementation, and unique capability to address the fast-drift problem in 3D NAND. Prior works have proposed variable read reference voltage techniques to address *conventional, long-term* V_{Th} drift in FG-flash-based 2D NAND [1, 2, 5]. For example, Cai et al. [1] proposed a retention optimized reading (ROR) scheme to reduce retention errors by periodically updating V_{Refs} , using an online pre-optimization process. As opposed to ROR and other such schemes, our ERR scheme addresses the *fast-drift* issue in CT-flash-based 3D NAND. In fact, we believe that the ERR and schemes such as ROR can be *co-implemented* to ensure superior performance and reliability in 3D NAND flash.

Also, Kim et al. [18] proposed a valley tracking scheme to find the optimum read reference voltage for 3D NAND. Unfortunately, such a scheme must incur the performance and energy penalty of three extra read operations, every time it needs to find the correct read reference voltage. In contrast, our ERR scheme avoids such overhead by calculating the amount of fast-drift using our proposed analytic model and selecting the proper V_{Ref} based on the time-bins set at design time.

3.1.3 Limitation. The adaptive process of revising V_{Ref} inherently imposes a limit on its correction capacity. Specifically, if ERR revises the V_{Ref} beyond a point, then it can generate new, self-inflicted errors. Figure 8(c) demonstrates a situation where fast-drift causes the V_{Th} distribution of state 2 to widen and overlap with that of state 1, and the V_{Ref} is revised to correctly read state 2 data. Regions R1 and R3 can be correctly read as state 1 and state 2, respectively. In the case of region R2, the revised V_{Ref} tries to compensate the impact of fast-drift and reads the data as state 2. However, some of the data in R2 actually belong to state 1; these data are read incorrectly and generate new errors. In such scenarios, the more ERR tries to shift V_{Ref} , the higher number of new errors will be generated from state 1. However, if we limit the V_{Ref} shift, errors from state 2 will go up. Therefore, we propose to supplement ERR with additional schemes to curb the effects of fast-drift.

3.2 Hitch-Hike: Risk-Based Prioritization

3.2.1 Overview. In Section 2.5, we observed that, while the ECB overhead for correcting a large number of errors in 3D NAND can be high, this overhead may be contained internally, in a reasonable number of flash pages. Also, detaching the physical mapping of data from the physical mapping of its associated ECC information can allow for more flexible error protection by reducing ECC overhead [40]. Inspired by these observations, we propose a scheme that can provide an alternate (and stronger) ECC to the most error-prone 3D NAND flash pages by placing their ECBs in a pool of predesignated free pages (hence the name hitch-hike).

In our hitch-hike scheme, as shown in Figure 9(a), most of the pages in a block are available for regular data storage and are encoded/decoded with the regular ECC module. However, the hitch-hike controller designates a small fraction of the pages as *custodian* pages and sets them aside for storing the ECB data. When a page retains data for a prolonged period and is expected to be vulnerable to fast-drift, the hitch-hike controller marks them as *client* pages. These critical client pages are then read by the memory controller in the background and are encoded using an augmented ECC codec. The ECB for this enhanced ECC encoding is stored in a custodian page. When a read is assigned for that client page, the controller accesses *both* the client page and its corresponding custodian page and decodes the data using the stored ECB.

Fig. 9. Our hitch-hike scheme: block organization (a) and ECB assignments to the custodian pages (b).

Fig. 10. An example showing how hitch-hike works.

3.2.2 Extending Hitch-Hike's Efficiency. The number of available custodian pages dictates how many error-prone pages can be handled by the stronger ECC scheme. For example, if 10% (as an example) of the pages in a 256-page block are assigned as custodian pages, then the hitch-hike scheme can serve only 25 error-ridden pages with the enhanced ECC. However, we can observe from Table 1 that even for a 512-bit error correction per page, the size of ECB is less than 1KB, while each page in 3D NAND is at least 4KB. Motivated by this, we propose to *split* the custodian pages into smaller segments and store the ECB data of multiple client pages in one custodian page. We use a simple pointer to inform the memory controller about the location of the required ECB data within a custodian page. When a client page is assigned a custodian, it gets a custodian page ID and an ECB pointer. The memory controller locates the custodian page using the page ID and utilizes the ECB pointer to move to the exact location of the ECB data. Figure 9(b) portrays an example scenario, where each 4KB custodian page is split into four segments that store one ECB data. For the earlier example of 512-bit correction using 10% custodian pages, the four-segment custodian pages can serve 100 clients, which is 40% of the whole block and can be adequate for most workloads.

3.2.3 Working Example. We present a simplified example of hitch-hike in Figure 10. In this example, we assume that (i) there are 10 storage (data) pages in a block, and only one custodian page is split into four ECB slots, and (ii) at the beginning, *page 1* has the oldest data and *page 10* has the newest. As shown in the figure, initially, pages 1 through 4 are the client pages, as they have the oldest data—most susceptible to fast-drift. Therefore, these pages are encoded with the enhanced ECC, and the ECBs are stored in the four slots of the custodian page. However, in the next cycle, pages 2 and 3 are re-programmed with new data, making them least vulnerable to fast-drift.

At this point, pages 5 and 6 move up in the queue as critical data. Consequently, the hitch-hike controller removes pages 2 and 3 from the client-page list and marks pages 5 and 6 as the new clients. Following this, the hitch-hike reads pages 5 and 6, encodes them with the augmented ECC, and writes the ECBs slots previously allocated to pages 2 and 3 with pages 5 and 6, respectively. For optimizing performance, since the hitch-hike controller has to wait for a block erase before overwriting the ECB slots previously used pages 2 and 3, the ECB for pages 5 and 6 are written in a custodian page with of empty slots (i.e., new or already erased).

3.2.4 Reliability and Overhead Considerations. It is worth noting that frequent writes on the custodian pages should not be a bottleneck for 3D NAND’s reliability. Initially, the custodian pages can have more writes, depending on the workload’s write-pattern and the number of ECBs per custodian page. However, we observed that this frequency gradually slows down, and the custodian pages keep holding the same ECBs for longer periods. In addition, for specific applications with frequent writes, we can reduce the number of ECBs per custodian page and/or increase the total number of custodian pages at design time.

Our hitch-hike scheme is designed to minimize overhead and any performance penalty. Specifically, not only read operations has the lowest overhead in NAND flash devices, but also they are processed in a queue that can allow the flash controller to locate the required custodian pages in advance and update them in the background—without incurring any significant performance penalty. Also, state-of-the-art flash controllers already records of page write times, and our design utilizes them to avoid extra overhead.

3.3 iRefill: Affordable Data Retention

As mentioned in Section 1, repeated write operations can reduce fast threshold (V_{Th}) drift in a CT-flash cell [27]. Specifically, if we perform in-place programming to re-write the same data in a cell, the lost charges can be “refilled” and the fraction of shallow-trapped charge is lowered, reducing fast-drift. We propose to exploit this observation for minimizing fast-drift in 3D NAND by scheduling regular “refill” operations for written pages on the memory.

3.3.1 In-Place Programming (ISPP). Typically, a NAND flash cell is only programmed when there is no charge in the CSL and the threshold voltage is at the lowest value. As a result, any re-programming operation must be preceded by an erase operation, which is expensive in terms of latency and power. However, the ISPP mechanism allows NAND flash cells to be programmed from a state with fewer charges to a state with more charges [36]. During ISPP, cells are programmed iteratively using a step-by-step program-and-verify method. Each programming step boosts the cell’s threshold voltage (V_{Th}), and then the following verify step senses that V_{Th} and compares it with the target value. This program-and-verify cycle continues until all the cells’ threshold voltages reach the target values. It should be noted that ISPP is not bi-directional; that is, it cannot be used to “take-out” charges from a NAND flash cell.

We have observed ISPP to be an excellent choice for treating 3D NAND flash cells suffering from fast-drift, as ISPP allows us to refill such cells by adding charges in a controlled manner. Also, even though prior work have reported potential error accumulation from ISPP when used for regular NAND flash programming [4], this does not apply for our refill scheme. This is because, even in the worst-case scenario, the number of refill operation will be merely a fraction of total number of writes and too small to accumulate errors of any significance.

3.3.2 Charge-Refill for Reducing Fast-Drift. We define a “refill” operation as reading a page while it is still valid and writing it back immediately. For a particular device and environmental configuration, the required number and frequency of refills mainly depend on the expected

retention period and the chosen refill strategy. In this work, we consider a storage-class retention of “ten years.” In addition, NAND flash cannot overwrite a page directly, and new data can only be written on a page after the previous data has been erased. Therefore, to implement our charge-refill technique, we need to modify the flash memory controller to support the “refill” operations. From a high level, we can schedule the refill operations either *concurrently* or *periodically*.

Concurrent charge-refill. In this scheme, a write operation is immediately followed by the required number of refill operations. For example, for 10-year data retention, each write should be followed by three subsequent refills. Since we cannot predict the required retention time for each data at the time of write, we must execute refill operations for every write and this can make the write overhead impractically high for the concurrent charge-refill.

Periodic charge-refill. In this case, we periodically schedule a single refill operation for a page only when it reaches the time threshold of losing data and repeat this process as needed. If a write/refill retains data for X months, and we need to retain the data for Y months, then that page will require $(\frac{Y}{X} - 1)$ refill operations after the initial write. With periodic refill, only data that need to be retained are refilled, which can reduce overhead of charge-refill. Interestingly, periodic refill can require more refills than concurrent refill if a large fraction of data needs to be retained for a long time. For example, let us assume a single write operation can retain data for one year. Now, if some data need to be retained for 10 years, periodic refilling will require 1 write + 8 refills, whereas concurrent refilling will only need 1 write + 3 immediate refills.

Note that both of these schemes can experience high write-overhead, making them unfit for many real-life applications (we discuss this further in Section 3.3.4). To address this, we propose *iRefill*—an efficient charge-refill scheme that can reduce fast-drift without hurting the performance of 3D NAND flash.

3.3.3 *iRefill*'s Scheduling Mechanism. *iRefill* utilizes the concept of reinforcement-learning (RL) to reduce the number of refills, which in turn can allow 3D NAND to attain storage-class retention with minimum overhead. The rate of fast-drift can fluctuate from cell to cell, as de-trapping of charges can be probabilistic to an extent. Also, for the same 3D NAND page, the fraction of cells requiring refill (i.e., BER) can vary with depending on the data pattern written, as only cells set to “0” have electrons injected into their CSL and are exposed to fast-drift. Unlike the static refill schemes, *iRefill* considers these situations and optimizes the refill operations.

Our *iRefill* scheduler is modeled as a RL problem, as described in Section 2.5. Specifically, the *iRefill* controller collects the state and reward information from the 3D NAND and assigns an action to the next state. The state functions (S) that our scheduler can acquire from the environment consist of the current refill count, the time elapsed since the last write/refill, and the current BER. The action function (A) includes either assigning a refill operation or continuing with the regular I/O operations. While the *immediate reward* (R) for *iRefill* is to maintain the BER permitted by the ECC scheme, the *long-term reward* is to minimize the refill frequency and maximize I/O throughput.

The scheduling mechanism employed by the *iRefill* scheduler is given as Algorithm 1. The scheduler operates based on a state-action table similar to the one shown in Figure 6(b). To reduce the state-space for the algorithm, the table is designed to record Q-values (rewards) at a block-level granularity, where each block is represented by the page holding the oldest data. Initially, all entries are initialized to the highest possible Q-value, using Equation (1) (line 3). The scheduler then randomly issues either a refill command or the scheduled I/O from the transaction queue (line 4) and calculates Q_p (line 5). For each block in the memory, a *refill-required* flag is enabled based on the *refill cycle* value. This flag is not disabled until all the pages in the block are refilled. At each *refill cycle*, *iRefill* issues the command selected in the previous cycle (line 7) and collects the

ALGORITHM 1: Our iRefill scheduling algorithm.

Data: A: Action (i.e., Command), S: State, R: Reward
Input: γ : Discount parameter, ϵ : Exploration parameter
Initialization
2 Refill cycle = 10 months; $\Delta T = 30$; Refill count = 0;
3 All Reward (**Q-values**) $\leftarrow \frac{1}{1-\gamma}$
4 $A \leftarrow$ select randomly: command from transaction queue or, refill
5 $Q_P \leftarrow$ get **Q-value** for current **S** and **C**
for Every "refill flag = high" signal **do**
7 Issue **A**, selected during the previous cycle
8 Collect immediate **reward** for the issued command
9 **if** $\text{rand}() < \epsilon$ **then**
10 Next **A** $\leftarrow$ random command (exploration)
11 **else**
12 Next **A** $\leftarrow$ command with the highest **Q-value** (exploitation)
13 **if** Next **A** = Refill **then**
14 Refill count $\leftarrow$ Refill count + 1;
15 Refill cycle $\leftarrow$ Refill cycle * 2;
16 $Q_{Sel} \leftarrow$ **Q-value** for the current **S** and **C**
17 Update_Q $\leftarrow$ SARSA update based on Q_P , **R**, Q_{Sel}
18 $Q_P \leftarrow Q_{Sel}$ // Set Q-value for next cycle

immediate reward (line 8). The next command is then selected based on the exploration parameter ϵ (lines 9 through 12) that keeps the RL algorithm dynamically tuned. ϵ is assigned a small value to ensure that the commands with the highest Q-value are mostly selected. If the next command is a refill operation, then the *refill count* value is incremented by one and the *refill cycle* value is doubled (lines 14 and 15). For each candidate command, the scheduler estimates the corresponding Q_{Sel} value from its Q-value table (line 16) and updates it using Equation (1) (line 17). Finally, the scheduler sets this value as Q_P for the next cycle. Note that we do *not* present iRefill as *the best* scheduler for 3D NANDs but as a scheme to efficiently exploit the characteristics of fast-drift and minimize errors in the CT-flash-based 3D NAND.

Intra-block pagewise refill process. Our iRefill schedules the refill operations at a block-level granularity to minimize the potential resource overheads. Specifically, the first page written in a block represents that block for fast-drift and retention considerations. As iRefill only requires a few refills spread over the whole life of data (we explain this in Section 5), the refill operations do *not* conflict with the regular I/Os, and refilling one block at a time does not have significant performance overhead. In addition, if regular I/O occurs while refilling a block, iRefill *interleaves* refill and the I/Os. This intra-block workflow is shown in Figure 11. When an idle block is flagged for refill, iRefill accesses it and refills the first page. Otherwise, it waits for the current I/O to complete. Once a page is refilled, iRefill checks for any I/Os scheduled for that block. If there is none, then iRefill accesses the next page. Else, the block is released for one I/O, and then iRefill refills the next page. This process continues until the last page in the block is refilled, and then iRefill moves on to the next block requiring refill. The advantage of this interleaving process is that even when refilling a block, regular I/Os can continue to execute, albeit at a slower rate.

Note that iRefill should not affect the endurance of 3D NAND, as endurance loss is mainly caused by the erase operations. Specifically, refills are in-place programming that do not require erases.

Fig. 11. iRefill workflow.

Fig. 12. Refill numbers in different schemes.

In addition, pages that are frequently written will not require refills, and pages only require refills if not written frequently.

3.3.4 Advantages of iRefill. The charge-refill mechanism can resemble legacy refresh operation in volatile memory such as dynamic random access memory (DRAM). Since periodic refresh with fixed interval has been used in DRAM for decades now, instinctively the same strategy may seem right for charge-refill in 3D NAND. On the other hand, concurrent charge-refill seems attractive in the sense that it requires little to no scheduling overhead. However, Figure 12 qualitatively portrays the comparative superiority of iRefill over such naive scheduling schemes. To highlight the advantage of iRefill, in this example we compare the three schemes at extremities. In terms of retention and number of refills, the most expensive situation is when a datum is written on 3D NAND flash at the start of its life, and it needs to be retained for its whole lifetime (in this work 10 years). We term these types of data as *cold*. On the other hand, we label data as *hot* if their required retention time is less than what one write can sustain with a refill (in this work 11 months). In this example, we have m number of hot data and n number of cold data, all to be written at time zero. However, it should be noted that both types of data are the same (i.e., $n = m$) as far as write overheads such as latency and power are concerned.

As shown in the figure, concurrent refill has to write, plus refill all $(n+m)$ data 3 times at once. This scheme suffers severely when there are no cold data. As we may not predict required retention of data at write time, even when $m = 0\%$ and $n = 100\%$, such a strategy has to incur the overhead of $4(n+m)$ ($= 8n$) writes. However, periodic refill only writes $(n+m)$ ($= 2n$) once at the beginning, but because of its fixed period, it has to refill m data 11 times over 10 years. While generally periodic refill may have a lower overhead than concurrent refill, it starts to suffer when the value of m starts to increase. For example, when $m = 100\%$ and $n = 0\%$ (i.e., no hot data), periodic refill incurs an overhead of $11n$ writes, which is much worse than concurrent refill.

Similar to periodic refill, our proposed iRefill also writes $(n+m)$ data, once at the beginning. However, unlike the fixed periodic refill scheme, the iRefill scheduler is designed to be aware of

Fig. 13. Modified 3D NAND flash software stack.

the slow-down of fast-drift with subsequent refill operations, and by keeping count of number of refills on a block, it can learn to reduce the number of refills. Specifically, iRefill only refills m number of cold data a total of three times at the 11th, 33rd (11+22), and 77th (33+44) months, respectively. In cases where there are no cold data (i.e., $m = 0\%$ and $n = 100\%$), iRefill only incurs the minimum number of $(n+m) (= 2n)$ writes. At the opposite extreme, when there are no hot data (i.e., $m = 100\%$ and $n = 0\%$), iRefill has an overhead of $4n$ writes.

It can be observed from this discussion that iRefill can resolve the weaknesses of concurrent and preodic refills by dynamically learning to optimize its schedule based on data characteristics. Therefore, with iRefill, 3D NAND can counter fast-drift and attain storage-class retention with minimum refill overhead and maximum efficiency.

Finally, our proposed ERR, hitch-hike, and iRefill schemes are specifically designed to counter the *fast-drift* phenomenon in 3D NAND flash and do NOT address the *long-term retention loss* that still exists in such devices. Therefore, our schemes are NOT replacements for the error correction/management mechanisms that 3D NAND has inherited from its 2D predecessor. In fact, such schemes should be orthogonally implemented with our proposed techniques to ensure a holistic reliability coverage for 3D NAND flash. For example, our schemes do not attempt to correct the errors from read and write operations—they are resolved by the conventional error checking and correction (ECC) mechanism as usual.

4 IMPLEMENTATION DETAILS

4.1 Hitch-Hike

The flash controller serves as a interface between the host system and the 3D NAND flash, providing the required communication platform. Implementing hitch-hike requires certain modifications in the existing flash controller design. This critical controller software is implemented in the embedded processor cores and is commonly known as the flash translation layer (FTL). From a software point of view, as shown in Figure 13, when a user application needs to access the flash memory, the operating system communicates with the FTL via the file system, which in turn connects it to the low-level NAND flash device. FTL can be broadly partitioned into two units: first, the flash abstraction layer (FAL) that translates *logical block address* to *physical page address* and

Table 3. Additional Block Data for Hitch-Hike

Block ID	Custodian Page	Busy Flag	Client Page	Linked Custodian Page	ECB Pointer
0	00000	1	0x02AB06C1	10101	10
	...	...	...	...	...
	11000	0	0x000004B0	01100	01
...	...	...	...	...	...
n	00000	0	0x004004C1	01010	00
	...	...	...	...	...
	11000	1	0x03d2E4C1	10001	11

carries out cache management and other background tasks and, second, the hardware adaption layer (HAL) that sends/receives low-level NAND instruction to/from the NAND flash and performs error correction and checking (ECC).

While the FAL's primary task is address translation, it also handles the tasks of wear leveling, bad-block management, and garbage collection. For a seamless integration, we fuse the hitch-hike controller within the FAL, as shown in Figure 13. In addition, we modify the HAL to incorporate two ECC codecs with regular and enhanced error correctibility. The hitch-hike controller utilizes the FAL to track writes in each block, maintains the block-level list of the custodian and client pages, and selects the appropriate ECC codec in the HAL for the page read operations.

Table 3 summarizes the blockwise information that the ReveNAND controller stores in its memory for executing the hitch-hike scheme. The memory controller maintains a list of all the custodian pages for each block and uses the busy flag to mark if a custodian page is free or not. The controller also retains a list of all the pages (in each block) that are currently in the client status and tracks which client page is *linked* to which custodian page. Finally, the ECB pointer is used locate a particular ECB in a custodian page.

4.2 iRefill

Implementation of iRefill in 3D NAND also requires certain modifications in the existing flash controller design. Following the same reasoning as the hitch-hike controller, we implement the iRefill scheduler as a FAL module. It operates based on the algorithm and work-flow described in Section 3.3. iRefill uses the HAL to collect state and reward information from the 3D NAND device and to send refill instructions. On the other hand, the FAL provides the iRefill scheduler with the refill and write statistics and ensures a smooth transition between the I/O and refill operations. In addition, similarly to the garbage collection module [25], the iRefill scheduler works as a background task to without compromising system performance.

Modifying the block-info table. For tasks like address translation, garbage collection, and bad-block management, the FTL stores a block-info table (BIT). As shown in Figure 14, the existing BITs contain six fields—block ID, erase flag, erase count, number of pages with valid data, sequence number for the block, and bad-block status flag. To facilitate the operation of our proposed iRefill scheme, we extend the BIT with four more fields. These fields are shaded in Figure 14. First, we need to store the time of the first page write in each block. Second, we need to store the refill count for the blocks. These two pieces of information are used by the iRefill scheduler to calculate fast-drift and the need for a refill operation. Third, the BIT needs to maintain the ID of the next page to be refilled in the block for cases when iRefill needs to release the block for I/O operation. Finally, we need to maintain the refill-required flag to mark whether a block needs refill or not. It should be noted that the information in BIT is stored in the DRAM buffer and is moved to the non-volatile flash storage before turning off the power.

Block ID	Erase Flag	Erase Count	Valid Page Count	Seq. Number	Bad-Block Status	Refill Count	First Write Time	Refill Page ID	Refill-Req Flag
1	False	4	11	3	False	2	10000370	0	False
2	False	9	5	8	False	1	10600234	53	True
3	True	6	0	-	False	1	21500710	7	True
4	False	0	9	12	False	0	25200860	0	False
5	True	3	0	-	False	3	40103004	214	True

Fig. 14. A snapshot of the modified block-info table.

Table 4. Memory Overhead for iRefill

3D NAND Flash: 1 TB Channels: 8, Chips: 8/channel Dies: 4/chip, Blocks: 2048/die, Pages (8KB): 256/block			
BIT Field	Bits/Block	Total Blocks	Total Overhead
Refill Count	2	524,288	1MB
First-Write Time	10	524,288	5MB
Next Refill Page ID	8	524,288	4MB
Refill Required Flag	1	524,288	0.5MB

Overhead estimation. Our iRefill design is entirely based on the embedded software in the FTL, and since the controller runs in the background, it should not impact the latency or power consumption significantly. In addition, as a refill operation is essentially a write operation, the modern flash controller hardware and the FTL has the required components to execute the refill instructions. Therefore, iRefill does *not* require any additional hardware. However, it requires a small amount of memory to store the four extra fields of information in the BIT. To estimate the memory overhead of iRefill in future 3D NAND, we calculated the individual overhead of the extra BIT fields for a 1TB 3D NAND flash device. The details of this device are given in Table 4. The refill count can assume values ranging from zero to three, and we therefore require 2 bits per block to represent it. Similarly, the time-stamp for the first page written on the block, the ID for the page to be refilled next, and the refill-required flag require 10, 8, and 1 bits per block, respectively. As one can observe from the table, the total extra memory overhead of iRefill amounts to less than 10MB only. Therefore, considering the relative sizes of the contemporary flash devices and their built-in DRAM buffers, iRefill’s memory overhead should be negligible.

5 EVALUATION

5.1 Fast-Drift: Modeling and Analysis

5.1.1 Analytic Model. The initiation and magnitude of fast-drift co-depend on certain design parameters and the underlying environmental conditions. While recent works have acknowledged the presence of fast-drive in CT-flash 3D NAND [9, 16, 22, 27, 29], to the best of our knowledge, Reference [7] remains the only publicly available work reporting the detailed behavior of fast-drift phenomenon in the CT-flash cells. Motivated by their empirical data, we analytically characterized the relation between fast-drift and the parameters that critical affect it. By considering all the parameters’ cumulative influence on fast-drift, and using the empirical data for obtaining the

corresponding fitting constants, we modeled fast-drift as

$$\Delta V_{Th}(t) = -[\log(t) + \alpha] \left[\frac{V_{Th,Init}}{\beta} + \theta \Delta T + \frac{\delta}{t_{buff-ox}} \right] \left[\frac{1}{R+1} \right], \quad (2)$$

where ΔV_{Th} is fast-drift, t is the elapsed time after a write, $V_{Th,Init}$ is the initially programmed V_{Th} , ΔT is the operating temperature – ideal room temperature (i.e., 20°C), $t_{buff-ox}$ is the buffer-oxide thickness, and R is the refill count. In addition, α , β , θ , and δ are the fitting constants with the values derived from Reference [7] as 5, 368, 2.1×10^{-4} , and 47.5×10^{-3} , respectively.

One can observe from Equation (2) that a high $V_{Th,Init}$ energizes the shallow-trapped electrons and linearly increases the fast-drift rate. In addition, the operating temperature (T) also has a linear relationship with fast-drift. Fast-drift changes inversely with the buffer-oxide thickness ($t_{buff-ox}$), as the buffer-oxide helps the CSL to achieve better non-conductivity. Finally, refill-count also has an inverse relationship with fast-drift, as higher number of refills gradually reduces fast-drift.

While Equation (2) characterizes fast-drift for 2D NAND cells, the impact of fast-drift can be more critical for 3D NAND. Fast-drift is directly dependent on the number of shallow-trapped charges at the nitride-oxide border, and because of its shared oxide and CSL, 3D NAND can allow a significantly higher number of electrons to be shallow-trapped. More importantly, the shared surface area in 3D-NAND increases with the additional stacked-layers, as these layers result in more cells in strings and all cells in a 3D-NAND string share a common channel, CSL, and tunnel-oxide (Figure 3).

A 3D NAND flash cell's retention is affected most by the inclusion of an immediate neighbor (layer), and it can be reasonably conjectured that the future 3D NAND flash memories with many stacked layers will have many shallow-trapped electrons available for detrapping and will in turn face a worse case of fast-drift compared to 2D NAND flash. Specifically, adding layers in a 3D-NAND string can be considered as *linear* stacking of rings/slices of critical surface that can host shallow-trapped charges. In addition, prior works have reported that for a fixed programming voltage, fast-drift increases linearly [9, 29]. This is because, in a 3D NAND string, the amount of fast-drift experienced by a layer increases *only with addition of a neighboring layer and is independent of other layers*. Carefully considering these factors, we assume a linear correlation between fast-drift and the stacked layers in our proposed model. For a $P\%$ increase in fast-drift for each stacked layer in 3D NAND, the total fast-drift in a 3D NAND design with n layers can be expressed as

$$\Delta V_{Th-3D} = \left(1 + \frac{P}{100}(n-1) \right) \Delta V_{Th-Cell}, \quad (3)$$

where $\Delta V_{Th-Cell}$ is the fast-drift for a single CT-flash cell. Equation (3) indicates that the stacked design can significantly aggravate errors from fast-drift in 3D NAND. It is worth noting that while our model is based on the TCAT design, it can be applied to any CT-flash-based 3D-NANDs by using the value of “ $P\%$ ” as the knob to reflect the varying impact of each stacked layer. We plan to explore this in the future when other 3D-NAND products are realized.

5.1.2 Analysis. First, Figure 15(a) confirms that fast-drift increases linearly with operating temperature, where the change in fast-drift is demonstrated for a single-level cell (SLC) 3D NAND cell initially programmed at $V_{Th} = 4V$. It should be noted that, unlike typical “Arrhenius”-induced retention loss, fast-drift is caused by de-trapping of the shallow-trapped electrons and is less sensitive to temperature change. The temperature sensitivity of the long-term retention loss in 3D NAND is *not* a consideration in modeling fast-drift. Second, as shown in Figure 15(b), multi-level cell (MLC) flash exhibits higher fast-drift than SLC. This is expected, as storing 2 bits/cell requires programming to higher V_{Th} s. Also, as different MLC states are programmed to different V_{Th} , the fast-drift is non-uniform across them—the state programmed with the highest V_{Th} suffers from the largest

Fig. 15. Characteristic analyses of fast-drift for critical environmental condition and design parameters.

amount of drift. Third, as we scale the process technology from 90nm to 10nm, the thickness of the buffer-oxide is approximately reduced from 7nm to 1nm (Figure 15(c)) [7]. From Equation (2), fast-drift increases with the decreasing buffer-oxide thickness. This is reflected in Figure 15(c) where we can observe that fast-drift increases linearly when scaling to smaller process nodes. Finally, the potential impact on fast-drift for having many stacked layers, as in 3D NAND, is projected in Figure 15(d), where the fast-drift values are normalized to that of a single CT-flash cell. Also, we consider a conservative 1% increase in fast-drift for each stacked layer. It has been predicted that, for reaching a terabit capacity, 3D NAND design will require stacking 90 layers [15]. One can observe from the figure that, in such cases, the fast-drift can almost double compared to a single 2D NAND cell—even when all other parameters are kept constant. Possible higher impact from stacked layers can reduce the reliability of future high-density 3D NAND flash even more.

5.2 Evaluation Setup

We designed an in-house simulator based on the fast-drift analytic model presented in the previous section and simulated the raw BER (RBER) for a 256GB 3D NAND flash fabricated in 40nm process technology. We considered a maximum operating temperature of 70°C. We calculated the RBER for different configurations running a wide range of real-life workload traces [31]. Using the RBER values, we then calculated corresponding ECC latency and energy consumption values for a 2.0 bit soft-decision LDPC scheme [19].

Workloads. We used workloads from a range of enterprise, online transaction processing, and scientific applications that often employ flash storage devices. These are data center workloads [31] from the public trace repository, with varying number of write operations. The details of the workloads are summarized in Table 5.

Evaluated configurations. For a detailed impact analysis of fast-drift and our proposed counter-measures, we evaluated six different configurations of 3D NAND:

Table 5. Workloads Used in Our Evaluations

Type	Name	Variation	Number of Writes ($\times 10^3$)
Enterprise (ENT)	Mail File Server	cfs0	135
		cfs1	130
		cfs2	135
		cfs3	132
	Hardware Monitor	hm0	2,575
		hm1	28
	Media Server	mds0	1,423
		mds1	73
	MSN File Server	msnfs0	1,467
		msnfs1	2,100
		msnfs4	500
OLTP	Financial	fin1	4,099
		fin2	653

Fig. 16. Sensitivity study for hitch-hike (data normalized to “0” custodian page): Improvement in error correction gradually slows down with higher number of custodian pages (a). However, ECC latency (b) and ECC energy (c) overheads continue to increase.

- Baseline: 3D NAND with fixed V_{Ref} , and no charge refill feature.
- ERR: 3D NAND with ERR scheme, but no charge refill mechanism.
- HitchHike: 3D NAND with fixed V_{Ref} , but employs hitch-hike.
- HitchHike+ERR: 3D NAND with both ERR and hitch-hike schemes.
- iRefill: 3D NAND with fixed V_{Ref} , but employs iRefill.
- iRefill+ERR: 3D NAND with both ERR and iRefill mechanisms.

Note that we do not evaluate HitchHike+iRefill, as it is an unrewarding overkill to concurrently implement hitch-hike and iRefill.

5.3 Hitch-Hike: Sensitivity Study

We evaluated hitch-hike for a range of custodian pages using three real-life workloads from Table 5 that represent three levels of fast-drift in 3D NAND. *hm0* has frequent writes and retains few data for a long time, reducing the impact of fast-drift. In contrast, *msnfs4* has few writes, which can lead to many pages suffering from fast-drift. Finally, *mds0* has a more balanced write/retention ratio.

Error correction improvement. More custodian pages generally allows for more fast-drift errors to be corrected. As shown in Figure 16(a), with high levels of fast-drift, *msnfs4* benefits the most from increasing the number of custodian pages, whereas the improvement is lowest for *hm0*. However, we also observe that the improvement in error correction seems to saturate eventually for all three workloads. We believe this is because every workload encounters a fixed number of

Fig. 17. Charge-refills increase the number of writes.

steady-state errors from fast-drift, and as the number of available custodian pages is increased, the error correction count for the additional custodian pages starts to diminish.

Escalating ECC latency and energy. While hitch-hike benefits from the availability of more custodian pages, Figure 16(b) and (c) shows that the ECC latency and energy quickly escalate to an unreasonable level to diminish that gain. Considering that a higher number of pages allocated to custodian duty also reduces the storage capacity of ReveNAND, we chose 25 custodian pages (i.e., 10%) as a Pareto optimal point for our proposed hitch-hike scheme. Note that Figure 15(b) and (c) only shows the trend between ECC latency and energy versus the number of custodian pages—it does not represent any hard limits. Any optimal point can be chosen by the 3D NAND architect (e.g., we chose 25 pages per block).

5.4 iRefill: Feasibility Study

In this section, we quantitatively validate the advantages of iRefill over the concurrent and periodic refill schemes by comparing their write overheads for 13 real-life workloads.

Figure 17 plots the total number writes (i.e., writes + refills) on 3D NAND for each workload for four different configurations: no refill, concurrent refill, periodic refill with 11 month fixed interval, and iRefill. We carried out this experiment for three *assumed* scenarios to represent the possible range of data retention requirement—with 100% cold data (= data with a 10-year retention requirement), 50% cold and 50% hot data (= data with a less-than-11-month retention requirement), and no cold data. Since the overheads of the read operation preceding a refill is minimal compared to the actual refill operation, we have not shown them explicitly. With 100% cold data, on average, our workloads have 411K writes, and this increases to 1645K and 4525K for the concurrent and the periodic refill schemes, respectively. iRefill performs better than both those schemes with an average of 1234K writes. However, this improvement is significantly more critical for the workloads with higher number of writes. For fin1, the number of writes increases from 4099K with no refill to 16396K and 45089K respectively when using the concurrent refill and the periodic refill. iRefill, on the other hand, needs only 12297K write operations. A similar trend can be seen for the 50% mix, with the expected reduction in writes for periodic refill as it needs to refill only half of the writes for a 10-year retention. In this case, the average number of writes with no refill, concurrent

Fig. 18. BER analysis.

Fig. 19. Normalized ECC latency improvement.

refill, periodic refill, and iRefill are 206K, 1645K, 2262K, and 617K, respectively. When there are no hot data to be retained, both the periodic refill and iRefill perform on par with the no-refill case. However, the concurrent refill still suffers from a higher write overhead. For example, no refill, periodic refill, and iRefill execute 2575K writes for hm0, while the concurrent refill performs 10,300K writes.

Note that while the results for the concurrent and periodic refill schemes vary with the data retention requirement, our iRefill out-performs both of the schemes for all three workloads tested.

5.5 Reliability and Performance Evaluation

5.5.1 BER Improvement. Figure 18 shows the level of reliability attained by the different configurations in terms of RBER, where the values are *normalized* to Baseline. As shown in the figure, ERR attains an average RBER improvement of 26% over Baseline, while the peak improvement of 57% is attained for *msnfs4*. We believe that this improvement of ERR comes from its ability to adjust V_{Ref} with the fast-drift level and correct more errors for each read. We then observe that HitchHike and HitchHike+ERR do not show any BER improvements over Baseline and ERR, respectively. This is because the hitch-hike scheme is not designed to reduce error but to correct more of them. In comparison, by reducing the fast-drift, iRefill attains a significant improvement of 78% over the baseline, on average. However, iRefill+ERR demonstrates the optimum reliability rating with an average improvement of 87% and a maximum of 92% for *msnfs4*. In our opinion, the combined impact of reducing fast-drift through iRefill, and correcting more errors with ERR, allows iRefill+ERR to achieve such excellent reliability.

5.5.2 ECC Latency Analysis. While a strong error correction scheme is required for reliable operation, ECC operations can add significant latency overhead to the memory. Since the ECC latency overhead is proportional to the number of errors experienced by the system, reducing BER can significantly lower the ECC latency. Figure 19 shows how ECC latency changes when using the five evaluated configurations, normalized to Baseline. As shown in the figure, ERR improves the ECC latency by up to 7 $\times$ for *msnfs4* and by 4 $\times$ on average. This latency reduction comes from a comparatively low BER due to more accurate data read by ERR. We can observe that the use of

Fig. 20. Normalized ECC energy improvement.

the enhanced ECC engine in HitchHike and HitchHike+ERR results in an average increase in ECC latency by $0.6\times$ and $0.3\times$, respectively, compared to the baseline. However, by reducing fast-drift, iRefill attains a much better average ECC latency improvement of $9\times$. Finally, iRefill+ ERR, with the lowest number of error bits to correct among all the configurations, produces the highest ($13\times$) latency improvement over the baseline.

5.5.3 ECC Energy Analysis. Stronger ECC schemes also consume large amounts of energy for the encoding and decoding operations [1]. As shown in Figure 20, ERR can reduce the energy overhead for ECC by $3\times$ compared to Baseline, on average. As expected, compared to the baseline, the stronger ECC engine increases the average ECC energy of HitchHike and HitchHike+ERR by $0.5\times$ and $0.2\times$, respectively. In comparison, iRefill improves the ECC energy efficiency by $6\times$, on average. However, the combined effort of ERR and iRefill allows iRefill+ERR to deliver a maximum ECC energy improvement of $31\times$ for *msnfs4*, and $10\times$ on average.

To conclude, one can observe from these evaluation results that with the conventional controller design, emerging 3D NAND flash devices can suffer critical reliability and performance issues. While individually our proposed ERR, Hitch-hike, and iRefill schemes can reduce the problem, combining them can allow 3D NAND flash to meet specific reliability and performance goals.

5.6 Limitations

To the best of our knowledge, the analytic model presented in this work is the first of its kind—at least in the public domain. Consequently, while we have used prior work as the basis of our proposed model and compared matched the results, we are yet to make a direct verification against another model. Also, due to unavailability of 3D NAND chips (i.e., not the packaged SSDs, since they come with built-in proprietary counter-measures), we are yet to compare our results with empirical data. However, we encourage the interested reader to go over [7, 9, 16, 21, 22, 27–29], as they can provide further details regarding how each component of our model impacts fast-drift.

6 RELATED WORK

3D NAND, fast-drift, charge-refill. Several prior works [17, 26, 39] proposed 3D NAND designs, while the CT-flash-based one has been the popular choice among real products [15, 33]. While they provide valuable perspectives, they do not provide any information data on fast-drift related problems in 3D NAND. However, References [7] and [27] have presented critical insight on the characteristics of fast-drift and charge refill, respectively. However, both of these works are based on the empirical data from a CT-flash cell, which does not *not* represent the system-level impact on 3D NAND accurately. Finally, References [9, 16, 22, 29] acknowledged the presence of fast-drift in 3D NAND products, but unfortunately they do not form the necessary link with the cell-level information on fast-drift. Compared to these prior works, we present a complete analytic model

that can evaluate fast-drift's impact on 3D NAND, both at the cell level and at the system level. We also propose a set of novel schemes to counter and alleviate the effects of fast-drift.

Adaptive read reference. A prior work [4] proposed to reduce errors in 2D NAND by refreshing the cells periodically/adaptively and preserve their relative V_{Ref} value, while another [2] proposed to assign V_{Ref} -based one the cells' location, order of programming, and content/value. In a different approach, Reference [5] proposed to assign a V_{Ref} to the target cell by taking into consideration the stored value of its immediate neighbors. Finally, Reference [1] proposed a scheme to reduce retention errors by periodically updating V_{Refs} , using an online pre-optimization process. While these prior works achieve varying degree of success in curbing the amount of error from conventional retention loss in FG-flash-based 2D NAND, unlike our ERR scheme, they are not equipped to address fast-drift in CT-flash-based 3D NAND. We believe that such schemes should be implemented together with ERR in future 3D NAND flash to ensure best possible protection against both fast and long-term loss of retention.

Annealing, tracking, and coding fast-drift. Kim et al. [18] presented a two-step annealing process to remove shallow-trapped holes during erase operations. They also proposed to track the optimum read reference voltage for 3D NAND using three read operations. Another prior work [20] proposed to tackle fast-drift by enhancing the data encoding process based on side-information. While the annealing process does not treat shallow-trapped electrons, it can still be used orthogonally with our ERR scheme. In addition, our ERR scheme can avoid any overhead from additional read operations or, complicated encoding, by directly evaluating fast-drift using our proposed analytic model and selecting the correct V_{Ref} based on the time-bins set at design time.

Flexible ECC. Yoon et al. [40] proposed to lower the ECC energy overhead of dynamic read only memory (DRAM) by using a virtualized mapping for ECC information and separating it from the physically mapped data. While our hitch-hike shares such idea of reducing the overhead of reliable execution by using non-uniform protection, the two methods are very different both in terms of technology (volatile DRAM vs. non-volatile 3D NAND) and implementation. Also, the hitch-hike introduces a novel mechanism that optimizes efficiency with the concept of split custodian pages.

7 CONCLUSIONS

In this work, we first proposed *ERR*, a fast-drift aware adaptive V_{Ref} scheme that reduces read errors. We then presented the *hitch-hike* scheme that exploits intra-block page organization to support enhanced ECC for critical pages. We then proposed the novel reinforcement-learning-based *iRefill* scheme to replenish lost charges and minimize the impact of fast-drift. Finally, we presented an analytic model for fast-drift in CT-flash and used it to explore the reliability and ECC overheads fast-drift may impose on emerging 3D NAND flash memories. Our evaluations show that the proposed *iRefill* and *ERR* schemes together can reduce the errors from fast-drift by 87%, on average, and significantly improve both ECC latency and ECC energy.

REFERENCES

- [1] Yu Cai, Yixin Luo, Erich F. Haratsch, Ken Mai, and Onur Mutlu. 2015. Data retention in MLC NAND flash memory: Characterization, optimization, and recovery. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*.
- [2] Yu Cai, Onur Mutlu, Erich F. Haratsch, and Ken Mai. 2013. Program interference in MLC NAND flash memory: Characterization, modeling, and mitigation. In *Proceedings of the IEEE International Conference on Computer Design*.
- [3] Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Crista, Osman S. Unsal, and Ken Mai. 2013. Error analysis and retention aware error management for NAND flash memory. *Intel Technol. J.* 17, 1 (2013), 140.
- [4] Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Adrian Crista, Osman S. Unsal, and Ken Mai. 2012. Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime. In *Proceedings of the IEEE International Conference on Computer Design*.

- [5] Yu Cai, Gulay Yalcin, Onur Mutlu, Erich F. Haratsch, Osman Unsal, Adrian Cristal, and Ken Mai. 2014. Neighbor-cell assisted error correction for MLC NAND flash memories. *ACM SIGMETRICS Performance Evaluation Review* 42, 491–504.
- [6] Li Pin Chang. 2007. On efficient wear leveling for large scale flash memory storage systems. In *Proceedings of the ACM Symposium on Applied Computing*.
- [7] Chih-Ping Chen, Hang-Ting Lue, Chih-Chang Hsieh, Kuo-Pin Chang, Kuang-Yeu Hsieh, and Chih-Yuan Lu. 2010. Study of fast initial charge loss and its impact on the programmed states Vt distribution of charge-trapping NAND Flash. In *Proceedings of the IEEE International Electron Devices Meeting*.
- [8] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. 2014. Dadiannao: A machine-learning supercomputer. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*.
- [9] Bongsik Choi, Sang Hyun Jang, Jinsu Yoon, Juhee Lee, Minsu Jeon, Yongwoo Lee, Jungmin Han, Jieun Lee, Dong Myong Kim, Dae Hwan Kim, et al. 2016. Comprehensive evaluation of early retention (fast charge loss within a few seconds) characteristics in tube-type 3-D NAND Flash memory. In *Proceedings of the IEEE Symposium on VLSI Technology*.
- [10] Robert Gallager. 1962. Low-density parity-check codes. *IRE Trans. Inf. Theory* 8, 1 (1962), 21–28.
- [11] Seong-Lyong Gong, Minsoo Rhu, Jungrae Kim, Jinsuk Chung, and Mattan Erez. 2015. CLEAN-ECC: High reliability ECC for adaptive granularity memory system. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture*.
- [12] Laura M. Grupp, John D. Davis, and Steven Swanson. 2012. The bleak future of NAND flash memory. In *Proceedings of the USENIX Conference on File and Storage Technologies*.
- [13] Jim Handy. 2013. An Alternative Kind of Vertical 3D NAND String. Retrieved from <http://goo.gl/cp8mmV>.
- [14] Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana. 2008. Self optimizing memory controllers: A reinforcement learning approach. *ACM SIGARCH Comput. Arch. News* 36, 39–50.
- [15] Jaehoon Jang, Han-Soo Kim, Wonseok Cho, Hoosung Cho, Jinho Kim, Sun Il Shim, Jae-Hun Jeong, Byoung-Keun Son, Dong Woo Kim, Jae-Joo Shim, et al. 2009. Vertical cell array using TCAT (Terabit Cell Array Transistor) technology for ultra high density NAND flash memory. In *Proceedings of the IEEE Symposium on VLSI Technology*.
- [16] Ho-Jung Kang, Nagyong Choi, Sung-Min Joe, Ji-Hyun Seo, Eunseok Choi, Sung-Kye Park, Byung-Gook Park, and Jong-Ho Lee. 2015. Comprehensive analysis of retention characteristics in 3-D NAND flash memory cells with tube-type poly-Si channel structure. In *Proceedings of the IEEE Symposium on VLSI Technology*.
- [17] Ryota Katsumata, Masaru Kito, Yoshiaki Fukuzumi, Masaru Kido, Hiroyasu Tanaka, Yosuke Komori, Megumi Ishiduki, Junya Matsunami, Tomoko Fujiwara, Yuzo Nagata, et al. 2009. Pipe-shaped BiCS flash memory with 16 stacked layers and multi-level-cell operation for ultra high density storage devices. In *Proceedings of the IEEE Symposium on VLSI Technology*.
- [18] Chulbum Kim, Ji-Ho Cho, Woopyo Jeong, Il-Han Park, Hyun-Wook Park, Doo-Hyun Kim, Daewoon Kang, Sunghoon Lee, Ji-Sang Lee, Wontae Kim, et al. 2017. A 512Gb 3b/cell 64-stacked WL 3D V-NAND flash memory. In *Proceedings of the IEEE International Solid-State Circuits Conference*.
- [19] Jonghong Kim and Wonyong Sung. 2012. Low-energy error correction of NAND flash memory through soft-decision decoding. *EURASIP J. Adv. Sign. Process.* 2012, 195.
- [20] Y. Kim, R. Mateescu, S.-H. Song, Z. Bandic, and B. V. K. Vijaya Kumar. 2015. Coding scheme for 3D vertical flash memory. In *Proceedings of the IEEE International Conference on Communications*.
- [21] Chang Hyun Lee, Kyung In Choi, Myoung Kwan Cho, Yun Heub Song, Kyu Charn Park, and Kinam Kim. 2003. A novel SONOS structure of SiO/sub 2/SiN/Al/sub2/O/sub3/with TaN metal gate for multi-gigabit flash memories. In *Proceedings of the IEEE International Electron Devices Meeting*.
- [22] Xinkai Li, Zongliang Huo, Lei Jin, Yan Wang, Jing Liu, Dandan Jiang, Xiaonan Yang, and Ming Liu. 2014. Investigation of charge loss mechanisms in 3D TANOS cylindrical junction-less charge trapping memory. In *Proceedings of the IEEE International Conference on Solid-State and Integrated Circuit Technology*.
- [23] Yan Li and Khandker N. Quader. 2013. NAND flash memory: Challenges and opportunities. *IEEE Comput.* 46, 8 (2013), 23–29.
- [24] J. W. Lim et al. 2015. A 128Gb 3b/cell V-NAND flash memory with 1Gb/s I/O rate. In *Proceedings of the IEEE International Solid-State Circuits Conference*.
- [25] Seung-Ho Lim and Kyu-Ho Park. 2006. An efficient NAND flash file system for flash memory storage. *IEEE Trans. Comput.* 55, 7 (2006), 906–912.
- [26] H.-T. Lue, T.-H. Hsu, Y.-H. Hsiao, S. P. Hong, M. T. Wu, F. H. Hsu, N. Z. Lien, S.-Y. Wang, J.-Y. Hsieh, L.-W. Yang, et al. 2010. A highly scalable 8-layer 3D vertical-gate (VG) TFT NAND flash using junction-free buried channel BE-SONOS device. In *Proceedings of the Symposium on VLSI Technology*.

- [27] Hang-Ting Lue, Yen-Hao Shih, Kuang Yeu Hsieh, Rich Liu, and Chih-Yuan Lu. 2005. Novel soft erase and re-fill methods for a P/sup+/-poly gate nitride-trapping non-volatile memory device with excellent endurance and retention properties. In *Proceedings of the IEEE International Reliability Physics Symposium*.
- [28] Hang-Ting Lue, Szu-Yu Wang, Erh-Kun Lai, Yen-Hao Shih, Sheng-Chih Lai, Ling-Wu Yang, Kuang-Chao Chen, Joseph Ku, Kuang-Yeu Hsieh, Rich Liu, et al. 2005. BE-SONOS: A bandgap engineered SONOS with excellent performance and reliability. In *Proceedings of the IEEE International Electron Devices Meeting*.
- [29] A. Maconi, A. Arreghini, C. Monzio Compagnoni, A. S. Spinelli, J. Van Houdt, A. L. Lacaita, et al. 2012. Comprehensive investigation of the impact of lateral charge migration on retention performance of planar and 3D SONOS devices. *Solid-State Electron.* 74 (2012), 64–70.
- [30] Janani Mukundan and Jose F. Martinez. 2012. MORSE: Multi objective reconfigurable self optimizing memory scheduler. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture*.
- [31] Dushyanth Narayanan, Eno Thereska, Austin Donnelly, Sameh Elnikety, and Antony Rowstron. 2009. Migrating server storage to SSDs: Analysis of tradeoffs. In *Proceedings of the ACM European Conference on Computer Systems*.
- [32] Michael Norman (Intel). 2012. Flash Technology in High-Performance Computing Accelerates Scientific Discovery. Retrieved from <http://goo.gl/VV8jsn>.
- [33] Ki-Tae Park, Sangwan Nam, Daehan Kim, Pansuk Kwak, Doosub Lee, Yoon-He Choi, Myung-Hoon Choi, Dong-Hun Kwak, Doo-Hyun Kim, Min-Su Kim, et al. 2014. Three-dimensional 128Gb MLC vertical NAND flash-memory with 24-WL stacked layers and 50MB/s high-speed programming. In *Proceedings of the IEEE International Solid-State Circuits Conference*.
- [34] Leeor Peled, Shie Mannor, Uri Weiser, and Yoav Etsion. 2015. Semantic locality and context-based prefetching using reinforcement learning. In *Proceedings of the ACM/IEEE International Symposium on Computer Architecture*.
- [35] Alessandro S. Spinelli, Christian Monzio Compagnoni, and Andrea L. Lacaita. 2017. Reliability of NAND flash memories: Planar cells and emerging issues in 3D devices. *MDPI Comput.* 6, 2 (2017), 16.
- [36] Kang-Deog Suh, Byung-Hoon Suh, Young-Ho Lim, Jin-Ki Kim, Young-Joon Choi, Yong-Nam Koh, Sung-Soo Lee, Suk-Chon Kwon, Byung-Soon Choi, Jin-Sun Yum, et al. 1995. A 3.3 V 32 Mb NAND flash memory with incremental step pulse programming scheme. *IEEE J. Solid-State Circ.* 30, 11 (1995), 1149–1156.
- [37] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.
- [38] Shuhei Tanakamaru, Masafumi Doi, and Ken Takeuchi. 2013. Unified solid-state-storage architecture with NAND flash memory and ReRAM that tolerates 32× higher BER for big-data applications. In *Proceedings of the IEEE International Solid-State Circuits Conference*.
- [39] SungJin Whang, KiHong Lee, DaeGyu Shin, BeomYong Kim, MinSoo Kim, JinHo Bin, JiHye Han, SungJun Kim, BoMi Lee, YoungKyun Jung, et al. 2010. Novel 3-dimensional dual control-gate with surrounding floating-gate (DC-SF) NAND flash cell for 1Tb file storage application. In *Proceedings of the IEEE International Electron Devices Meeting*.
- [40] Doe Hyun Yoon and Mattan Erez. 2010. Virtualized and flexible ECC for main memory. *ACM SIGARCH Comput. Arch. News* 38, 397–408.

Received June 2017; revised December 2017; accepted January 2018