On Enhancing Data Integrity with Low-cost Retention-Refillable Programming Scheme

On Enhancing Data Integrity with Low-cost Retention-Refillable Programming Scheme

Kun-Chi Chiang
National Tsing Hua University
Hsinchu, Taiwan
sailfish0814@gapp.nthu.edu.tw

Yung-Chun Li
monixs1112@gmail.com
Macronix international Co., LTD.
Hsinchu, Taiwan

Wei-Chen Wang
wweichen@mit.edu
Massachusetts Institute of Technology
Cambridge, Massachusetts, U.S.A.

Wei-Kuan Shih
wshih@cs.nthu.edu.tw
National Tsing Hua University
Hsinchu, Taiwan

ABSTRACT

The retention error has become one of the most challenging reliability issues of flash memory due to the shrinking of the technology nodes. To enhance data integrity by resolving the retention error issues for 3D MLC flash memory devices (e.g., SSDs and SD cards), many excellent works that exploited in-place reprogramming and data refreshing concepts have been proposed in recent years. However, these approaches could result in additional issues, such as programming disturbance and performance overhead (e.g., unavoidable data refreshing and a larger amount of program and verify shots). This work is motivated by the need to explore a low-cost solution for resolving retention error issues without incurring negative impacts caused by conventional refresh-based and in-place reprogramming approaches. As a result, this work exploits the characteristics of the cell's V_t distribution and proposes the novel concept of "retention-refilling" to enhance data integrity. With such an idea, a retention-refillable programming scheme is proposed to improve flash reliability and mitigate performance overheads by trading data refreshing with retention-refilling. The capability of the proposed scheme is evaluated by a series of experiments, for which we have very encouraging results.

KEYWORDS

Retention-refilling, retention error, data integrity, program disturbance.

ACM Reference Format:
Kun-Chi Chiang, Yung-Chun Li, Wei-Chen Wang, and Wei-Kuan Shih. 2024. On Enhancing Data Integrity with Low-cost Retention-Refillable Programming Scheme. In *The 39th ACM/SIGAPP Symposium on Applied Computing (SAC '24), April 8–12, 2024, Avila, Spain.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3605098.3635905

1 INTRODUCTION

Flash memory has been widely adopted in recent years due to the request for high-speed computation and mass storage. The 3D multi-level-cell flash memory, including MLC, TLC, and QLC, is the favored choice for cost-effective and reliable data storage. Despite the dominance of 3D MLC flash chips, maintaining data integrity remains a significant challenge. To enhance data integrity and address reliability issues in flash devices, wear leveling is considered the primary method for evenly distributing erase operations among flash memory blocks [4, 7, 27]. Flash memory faces three major reliability concerns: endurance, retention, and disturbance. All of these concerns can result in a significant number of bit errors, especially as flash memory processes shrink or when used in environments with elevated operating temperatures, such as data centers or automotive electronics [23, 24]. The objective of wear leveling is to prevent any flash memory cells from being erased too many times before those flash memory cells can no longer store any data reliably. However, it is reported that retention errors might become the dominant errors in NAND flash memory [5, 33]. Unfortunately, wear leveling could not effectively resolve retention errors and guarantee data integrity because an even distribution of erases over blocks could not resolve the leaking of electrons from a memory cell. Although many excellent works are proposed to resolve the retention error issues with the in-place reprogramming and data-relocation approaches, they might incur additional issues, such as the programming disturbance issue and the large amount of live-page copying overhead. There is still a lack of a promising solution to simultaneously solve the reliability and performance issues, and this observation motivates this work to explore a new programming scheme that can efficiently resolve retention error issues by refilling charges into storage layer with very limited costs.

A flash memory storage device typically comprises multiple NAND flash memory chips. Each chip is divided into blocks, and each block contains fixed pages. Pages consist of a data area (to store user data) and a spare area (to store housekeeping data). Pages of a block are usually written from the first one to the last one sequentially. Due to the "write-once property", data cannot be updated unless their belonging blocks are erased. Therefore, an "out-of-place update" policy is used for writing to-be-updated data to selected free pages. As a result, up-to-date pages are called *valid pages*, while out-of-date pages are *invalid pages*. A flash translation layer (FTL) manages the mapping between logical block addresses (LBAs) and physical block addresses (PBAs) and handles garbage collection to reclaim space from invalid pages[1–3]. Since each block can only endure a limited number of program/erase operations, wear leveling evenly distributes erases to extend the flash memory chip lifespan.

Many studies have shown that the Program/Erase(P/E) cycle of flash memory could be significantly reduced by relaxing the retention time of pages [5, 19]. However, wear leveling could not directly address the retention error problem. With the considerations of the

SAC '24, April 8–12, 2024, Avila, Spain

ACM ISBN 979-8-4007-0243-3/24/04…$15.00
https://doi.org/10.1145/3605098.3635905

lengthy erase time and also the reliability concerns, researchers had proposed various garbage collection policies to reclaim the space of invalid pages to minimize the impact of performance and reduce the number of live-page copyings, e.g., [13, 17]. Thus, many efforts have been made to explore efficient solutions for resolving the retention error issues in recent years. They can be categorized into four types. (1) Error correction code (ECC) approaches: since the raw bit error rate (RBER) of NAND flash memories become worse and worse while 3D stacking and multi-level cell technology are adopted. The BCH and LDPC [20, 28] were widely adopted to correct the error bits. However, the data is corrupted while error bits exceed the correction ability. On the other hand, the data with lower reliability might incur longer latency while resolving the error bits [11, 25]. (2) Refresh-based approaches: some works explore efficient refresh-based solutions to resolve the retention error issues [18, 22, 26]. However, the refresh-based solution can result in significant live-page-copying overhead and waste the lifetime of the flash block. (3) Read voltage adjustment approaches: the retention issues induced by electronic leakage that leads to the left shift of cells' V_t distribution. To read the correct data from the left-shifted V_t distribution, some researchers proposed adjusting the read voltage dynamically to resolve retention error [12, 24]. However, these schemes result in a great number of read and retry operations are needed to find the optimized reference voltage, and thus generate additional overheads. (4) Re-programming approaches: some researchers propose to periodically correct retention error by in-place reprogramming retention-erred cells [6, 31]. It adopts the full and partial in-place self-programming (ISPP) procedure to push the left-shift V_t distribution back to their desired window. However, the disturbance and electronic injection rate should be a concern because the programming bias should be suitable for each data state.

This work aims at resolving retention error issues of 3D MLC flash memory [32]. Although lots of existing works had addressed the retention issues on the 2D MLC flash memory, there is still a lack of efficient way to guarantee data integrity with considering the live-page-copy overheads and disturbance effects on the 3D multi-level NAND flash memory cells. Therefore, we proposed a retention-refillable programming scheme to enhance data integrity and address retention errors. Our approach resolves reliability issues by skillfully altering the V_t distributions of retention-erred cells and improves data integrity with minimal overhead. Unlike previous work, we aim to significantly reduce the number of program shots required in the ISPP and delay the need for refresh operations to extend retention times for various data states. Our scheme considers the properties of different cell V_t states and various retention-error handling approaches to minimize negative impacts. We conducted a series of experiments to assess the effectiveness of our design. The results demonstrate that our retention-refillable programming scheme can extend device lifetime by up to 60% while it only incurs very limited performance overhead on the total execution time by about 10% under the different cases, compared to the in-place reprogramming approaches.

2 BACKGROUND AND MOTIVATION

2.1 Flash Basis and Retention Error

NAND flash memory functions by utilizing various threshold voltage (V_t) levels to encode stored data, achieved through the control of electrons trapped within the floating gate of flash memory cells. To meet the continuously growing need for storage capacity, most flash vendors have widely adopted the multi-level-cell (MLC) flash memory technique. In an $\text{MLC}_{\times N}$ flash memory cell, it stores N bits of data and can be programmed to one of 2^N threshold voltage levels. A typical MLC flash memory chip is composed of multiple MLC flash blocks, with each block containing N word lines (MLC flash pages), and each word line consisting of M memory cells. For instance, the MLC flash cell stores 2 bits of data, represented by 4 different V_t levels. Consequently, a MLC flash page contains data from two pages: an *upper page* and a *lower page*, each with M bits of data. To represent different data states in a MLC flash cell, the V_t distribution is adjusted to fit a specific V_t window by altering the number of electrons.

The narrowing of V_t windows in MLC flash cells increases their sensitivity to V_t variations. This sensitivity exposes them to several factors, including retention errors and read/write disturbances. Read and write disturbances occur when neighboring cells are repeatedly accessed or programmed, causing an overcharge of electrons in the cell. This shift in the V_t distribution of flash memory cells skews it to the right. Conversely, retention errors happen as charged electrons gradually leak from a cell over time, leading to a leftward shift in the V_t distribution. These shifted V_t distributions can result in incorrect data states during read operations, as depicted in Figure 1. While numerous effective error correction codes (ECC), programming, and wear-leveling methods can handle read or write disturbances, tackling retention errors remains challenging. In fact, there are reports that retention errors might become the predominant issue in NAND flash memory [5, 33].

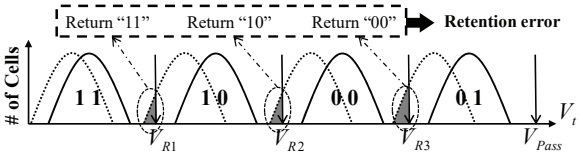
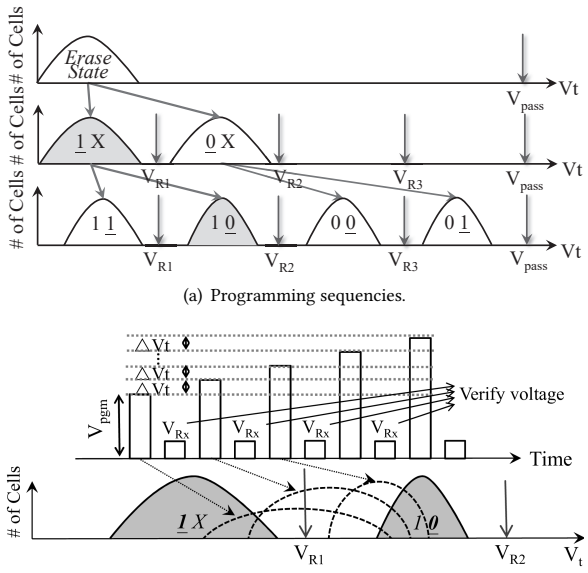


Figure 1: Fail on reading data stored on flash cell which expires its retention time.

2.2 Programming Methods

To program MLC flash memory cells into a limited V_t window precisely, a widely adopted technique is the coarse-fine programming scheme. Under the coarse-fine programming scheme, the programming process for the upper and lower pages of the same word line is divided into two stages, as depicted in Figure 2(a). The upper page's data is programmed into the selected word line in the first stage. Flash cells storing "1" (from the upper page) remain in state "1", while cells storing "0" are programmed to a higher threshold voltage state called "0". In the second stage, the lower page's data is programmed into the word line, pushing cells' V_t distributions into four different states, allowing each cell to represent two-bit data, one from the upper page and the other from the lower page. As shown in Figure 2(a), cells in state "1" are programmed to "10" if they store "0" from the lower page; otherwise, they become "11". Similar operations are applied to cells in state "0". Program verify voltages V_{R1}, V_{R2}, and V_{R3} are used to determine the lower bound of the V_t distribution for logical states "10", "00", and "01", respectively.

Over the course of a long time, several major optimizations were contributed for programming techniques to mitigate the challenges caused by the narrow V_t window. One notable approach used in NAND flash memory is *incremental step pulse programming*



(a) Programming sequencies.

(b) Incremental step pulse programming (ISPP) procedure.

Figure 2: Typical MLC programming method.

(ISPP) [15, 21]. The core concept involves repeatedly executing cycles that consist of *verify* and *program* phases. During the verify phase, cells that haven't reached the target V_t are identified. In the subsequent program phase, the program voltage is incrementally increased by a small, fixed amount called ΔV and used to program the cells identified in the verify phase. This programming strategy continues until all cells have achieved their desired states or the number of cycles exceeds a predefined limit, as depicted in Figure 2(b). Numerous ISPP variants have been extensively researched to balance performance and reliability. For instance, reducing ΔV can prevent over-programming and narrow the V_t distribution, improving MLC flash memory reliability. However, this approach may increase programming latency due to more ISPP cycles.

2.3 Research Motivation

Although many excellent works have addressed the retention error issues, there was a lack of solutions to simultaneously resolve the disturbance issues caused by in-place reprogram-based methods, system performance degradation by ECC decoding, and larger live-page copying overheads caused by data-refresh-based approaches. To take advantage of the efficiency of reprogram-based methods and the robustness of data-refresh-based approaches, our goal is to propose a new MLC programming scheme to avoid management overheads, such that the cell's retention capability can be effectively refilled without the data refreshes. Unlike existing reprogram-based methods that only adopt the ISPP method with a fixed program voltage to reprogram flash cells, our goal is to refill the retention capability of flash cells with the tailored program voltage according to the shifted V_t status of flash cells. To achieve this goal, the technical issues fall on (1) how to find out the tailored program voltage for flash cells with the different shifted V_t states to minimize performance overhead, (2) how to minimize the negative effect of program disturbance caused by the retention refill procedure, and (3) how to effectively reduce the data refresh overheads[1] caused by the processes of handling the retention errors. This thus motivates this work to explore a low-cost retention-refillable programming scheme to not only efficiently extend the retention time of flash cells but also eliminate the negative impacts caused by conventional reprogram-based and data-refresh-based approaches.

3 LOW-COST RETENTION-REFILLABLE PROGRAMMING SCHEME

3.1 Design Overview

This section presents an affordable retention-refillable programming scheme with dual goals: enhancing data integrity by reducing retention-induced error bits and optimizing management overhead by minimizing the need for data refreshes. The key idea is to restore left-shifted V_t distributions in cells to replenish retention capability and extend the observation period to identify hot/cold data, reducing unnecessary data refreshes. This significantly reduces the frequency of required data refreshes, improving management overhead efficiency.

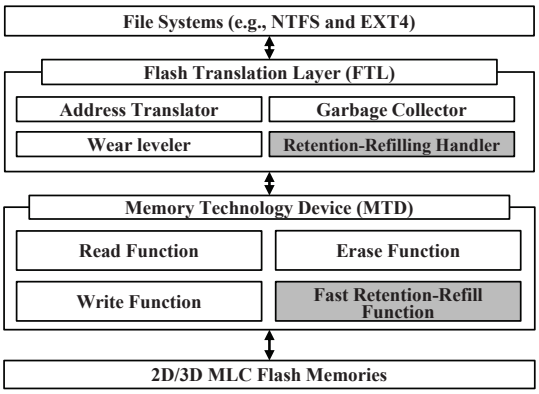


Figure 3: Architecture of the proposed retention-refillable program scheme.

To achieve this goal, we introduce a cost-effective retention-refillable programming scheme that leverages our innovative *fast retention-refill program function*. The corresponding system architecture can be seen in Figure 3. In the upcoming sections, we outline our design concept, which involves modifying the left-shifted V_t distributions of flash cells to restore retention capability (Section 3.2). We then delve into the design of the *fast retention-refill program function*, which integrates retention refilling with the programming procedure (Section 3.3). Finally, in Section 3.4, we present the design of the *retention-refilling handler*, demonstrating how these schemes seamlessly integrate into MLC flash memory management. For simplicity, we assume the use of a basic page-level FTL and focus on explaining the concepts and implementation details of the proposed retention-refillable programming scheme in the following sections.

3.2 Design Concept of Retention-Refillable Programming Scheme

As discussed in Section 2.1, NAND flash memory experiences retention errors over time due to the left-shifted V_t distribution of cells.

[1]Hereafter, the "data refresh" refers to the procedure of rewriting data of a flash page into another flash page to avoid the occurrence of retention errors on the corresponding page.

To effectively enhance the retention capability of cells that are at risk of encountering retention errors in the near future, our main idea is to refill the retention capability of these cells by strategically altering their left-shifted V_t states. In general, the ECC unit (such as BCH and LDPC) [20, 25] in the flash device can detect error bit locations and correct them within ECC's specified capacity. The guaranteed capability (i.e., the maximum correctable number of error bits) varies according to the requirements of each flash device. If the flash device incorporates the LDPC mechanism, we can apply our design to refill retention-affected cells when the decoding overhead increases, signifying a high number of iterations for hard/soft decisions. For the sake of simplicity, we assume BCH as the default ECC scheme in the following sections, with the capability to correct up to N bits of errors. Regardless of the ECC scheme adopted, the proposed retention-refillable programming scheme is invoked when the number of error bits exceeds a predefined threshold. When the count of retention error bits surpasses a predefined threshold of $\rho \times N$, our proposal suggests refilling the retention capability for these cells to address the retention error issues[2]. To achieve this, cells with erroneous V_t states must be reprogrammed to restore their incorrect V_t states to their original range. This approach aims to simultaneously correct the retention-affected bits, extend retention capability, and reduce the substantial data refresh overheads.

3.3 Fast Retention-Refill Function Design

The design of the proposed fast retention-refill function aims to efficiently restore the left-shifted V_t states of retention-affected cells and replenish their retention capability. To achieve this goal, a straightforward approach would involve applying the conventional ISPP procedure to a page. This would allow the retention-affected cells to be reprogrammed to their desired states through multiple programming and verification iterations. However, this reprogramming strategy may introduce disturbances to other cells on the same wordline, whose V_t states are already correct, potentially causing their V_t states to shift to the right. To expedite the retention-refilling process and reduce the risk of program disturbances, we adopt an approach inspired by the methods outlined in [14]. In this approach, we pre-select a specific program voltage (V_{pgm}) for each left-shifted V_t state and use this pre-selected program voltage to restore the retention capability of the affected cells. To clarify, when applying the proposed fast retention-refill function to a wordline for the recovery and refilling of retention-affected cells, *we exclusively employ the pre-selected program voltage (V_{pgm}) to reprogram these cells based on their current and desired V_t states*. This approach streamlines the programming process by eliminating the need to program a flash page with multiple fixed step voltages, ensuring the efficiency of retention refilling, and minimizing the risk of program disturbances affecting other cells on the same wordline.

Some careful readers might question how to determine the pre-selected V_{pgm} for replenishing the retention capability of cells with various left-shifted V_t states. To gather this critical information, we conducted a series of offline profiling experiments using our FPGA-based flash memory evaluation platform, which allows precise adjustments of program voltages in engineering mode. We leveraged this platform to apply different program voltages, i.e., V_{pgm}, to all cells within the retention-erred pages, establishing the relationship between the shifted states and the applied V_{pgm} for different

[2]In our experimental setups, we configure the encoding/decoding unit of the adopted BCH ECC with 2K Bytes, 132 bits, and 80% for the maximum correctable error bits N and ρ, respectively.

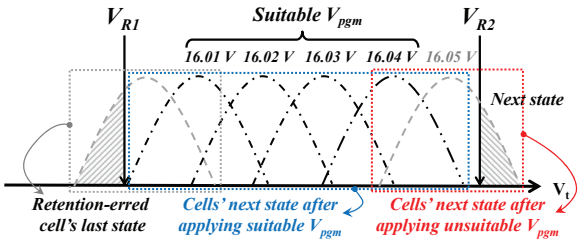


Figure 4: Process on evaluating suitable V_{pgm} for fast retention-refilling program function.

retention-affected conditions. Let's take, for instance, the retention-affected cells initially intended to have a V_t state within "10", as illustrated in Figure 4. We systematically applied various program voltages (V_{pgm}) within a specific range (e.g., ranging from 16.01V to 16.05V in our example) for one-shot programming on these affected cells and recorded the resulting V_t states. Subsequently, we analyzed the V_t states after applying the one-shot programming to identify the suitable V_{pgm} values capable of correctly restoring the right-shifted V_t state of cells initially targeted for "10". In Figure 4, for instance, the appropriate V_{pgm} values were found to be 16.01V, 16.02V, 16.03V, and 16.04V. By systematically exploring various configurations, we accumulated a repository of appropriate V_{pgm} values for cells under diverse retention-affected scenarios. This data allowed us to construct a lookup table, which specifies the necessary V_{pgm} settings. Consequently, our proposed fast retention-refill function can promptly choose an appropriate V_{pgm} to restore and replenish the retention capability of retention-affected cells, significantly reducing the number of required program cycles. It is essential to note that this scheme is also applicable to 3D TLC and QLC flash memories, offering substantial overhead reductions compared to refresh-based and in-place reprogramming methods. This is particularly beneficial as both 3D TLC and QLC flash memories necessitate a larger number of program cycles in conventional ISPP procedures.

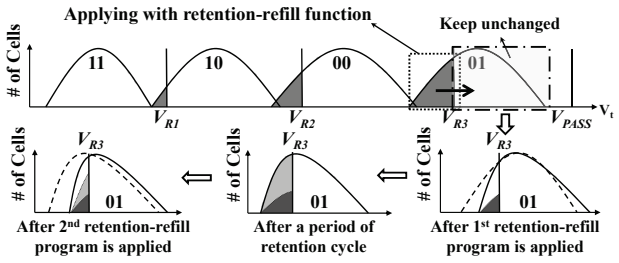


Figure 5: The retention-refillable programming method can result in the morphing of cell's V_t distribution.

In summary, the proposed fast retention-refilling function can refill the retention capability for flash cells. However, the number of times that retention-erred cells can be applied with the proposed fast retention-refill program function is limited. The retention-refill process will fail to refill the retention capability for the retention-erred cells while their refilling times exceed the guaranteed number. This is because the proposed fast retention-refill program function is only applied to the retention-erred cells on a wordline, and it cannot

push the left-shift V_t states of all the cells on the same wordline back to their target V_t window perfectly without completing the full ISPP procedure. This can be better explained with the example shown in Figure 5.

3.4 Retention-Refilling Handler Design

As we discussed in Section 3.1, the FTL layer requires a retention-refilling handler to monitor the status of flash pages and blocks, utilizing the retention-refilling functions as needed in various situations. The responsibilities of this retention-refilling handler are two-fold. Firstly, it must determine when to employ the proposed fast retention-refilling function to replenish the retention capability of cells. Secondly, it should minimize data refresh operations by using the proposed fast retention-refill function. This extension of retention time for less frequently accessed data helps reduce unnecessary data-refresh overhead. It's essential to mention that NAND flash programming often incorporates the randomizer technique [28], efficiently mitigating issues like the "11" data pattern and enhancing data reliability. In our approach, we select the first valid page of each block and then assess its retention error status using the ECC engine. If any page is found to have a retention error exceeding $\rho \times N$ bits, all remaining valid pages within the same block will undergo the retention-refilling management processes as outlined in Algorithm 1.

Algorithm 1: Retention-Refilling Handler Design

```
1 while Block k occurred retention error do
2     if k.Fast_refill_count is MAX then
3         Apply data-refresh procedure to copy valid data into a new block n;
4         k.Fast_refill_count =: 0
5     else
6         Perform Fast Retention-Refill Program Function on block k;
7         k.Fast_refill_count++;
```

Algorithm 1 outlines the management procedure of our retention-refilling handler (lines 1–7). To determine whether data inside a block should be employed retention refilling correction, a preliminary read operation is conducted on the first page of each block. Subsequently, we evaluate whether the number of error bits decoded by ECC reaches our predefined threshold. When the number of retention-erred bits in the examined block exceeds the predefined threshold, the retention-refilling handler evaluates the situation based on the accumulated retention error bits. It then determines whether to proceed with data refresh or retention refilling, forming its decision accordingly (lines 1–2). This handler monitors whether the fast retention-refilling function has been previously applied to the current page. If the $Fast\_refill\_counter$ for block k reaches the user-defined maximum, it indicates that the fast retention-refilling function has been used on this block multiple times. In such cases, we execute the data-refresh procedure to copy the valid data into a new block n, and then reset the counter (line 3–4). However, if the $Fast\_refill\_counter$ did not reach the maximum, the retention-refilling handler proceeds to apply the fast retention-refilling function to replenish the retention capability for retention-erred cells on each valid page within the examined block (line 6). After the fast retention-refilling function has been applied to the entire block, the $Fast\_refill\_counter$ of the current block is incremented (line 7). In our design, we utilize off-the-shelf hardware to implement the refilling process, with the associated design overheads primarily managed through software.

4 EXPERIMENT RESULTS

4.1 Experiment Setups

In this section, we conducted a series of device-level and system-level experiments to evaluate the capability of our proposed fast retention-refilling programming scheme. For the device-level evaluation, we implemented the fast retention-refill program function design in 3D MLC flash chips [32] on an ALTERA Cyclone IV FPGA platform [9, 10]. The 3D MLC flash chip used for evaluation was placed on the FPGA-based platform, which allowed fine-grained adjustments for each pin of the flash chip in an engineering mode. We also integrated related schemes for reducing retention errors into this platform, enabling measurement of their timing delay, cells' V_t distribution, and bit error rate status.

For the system-level evaluation, a series of experiments were conducted to assess the effectiveness of our proposed approach, along with comparison to alternative methods, utilizing a flash-memory simulator [8, 10]. We simulated a 16GB MLC flash-memory device with block and page sizes set to 16MB and 16KB, respectively. To evaluate the performance of our design under varying system loads, we employed different initial data ratios within the simulated device, specifically 20%, 50%, and 80%. The proposed retention refilling scheme was configured to allow a maximum of 3 refilling cycles. We then used the results of these experiments to compare our approach with three others: (1) The baseline approach, which does not employ any strategy to mitigate retention errors. (2) The refresh-based approach, which implements a data-refresh policy to prevent retention errors. (3) The in-place reprogramming approach, utilizing the complete ISPP procedure to refill cells' retention capability once and perform data refresh to mitigate disturbance-induced data errors. These approaches are referred to as *W/O Refresh*, *Data Relocation*, and *In-Place ISPP*, respectively.

The simulator configurations were based on device-level experiments. In our setup, the three approaches, i.e., *W/O Refresh*, *Data relocation*, and *In-place ISPP*, utilized an equal number of program and verify shots, following the traditional program policy. The traditional programming procedure adopted one programming voltage with three verify voltage on target memory cell. This policy resulted in longer program latency, requiring 60 program shots and 180 verify shots for a 3D MLC flash page. In contrast, the proposed fast retention refilling only necessitated 3 program and 3 verify shots for retention refilling. To evaluate various benchmarks, real traces from Microsoft Research Cambridge, Postmark, and the SNIA IOTTA Repository were employed [16, 29, 30].

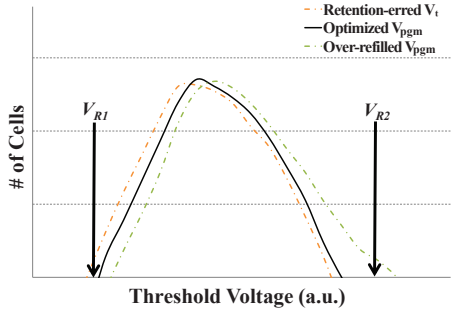


Figure 6: Examining process for the selection of optimized V_{pgm}.

4.2 Device Level Experiment Results

In Section 3.3, we explored the necessity of a lookup table to determine the ideal V_{pgm} for refilling cells' retention capability in different scenarios. Figure 6 illustrates how various pre-selected V_{pgm} values affect retention-erred cells with a desired state of "10" during the retention-refilling process. If the chosen program bias is excessively high, it can result in over-programming and subsequent over-refilling errors when using the fast retention-refilling function. To prevent these errors, the function selects an optimized program V_{pgm} for each state, ensuring that the refilled V_t distribution falls within the desired range. This fine-tuning enables the fast retention-refilling function to efficiently restore and refill the retention capacity of retention-erred cells on flash pages, as demonstrated in Figure 7.

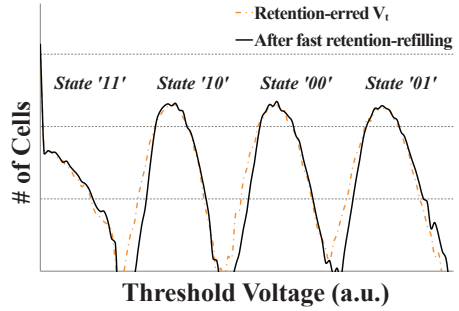


Figure 7: The corrected V_t states of retention-erred cells.

To further analyze the capability of the fast retention-refilling function, the corrected rate can be formulated with the following equation:

$$Corrected\ Rate = \frac{Corrected\ Bits}{Retention\ Erred\ Bits} \quad (1)$$

Referring to Equation 1, we present the corrected rates achieved by the fast retention-refilling function for each data state, as detailed in Table 1. To summarize, the corrected rates of the fast retention-refilling function range from 76.98% to 83.33%. Notably, the corrected rate for the state '01' is observed to be lower compared to the others. This discrepancy can be attributed to the fact that state '01' represents the highest V_t state, requiring a more robust electric field for electron refilling. However, the careful selection of V_{pgm} is pivotal in preventing excessive program disturbances in other remaining cells.

Table 1: Fast retention-refilling corrected rate.

Memory State	State '10'	State '00'	State '01'
Correct Rate	83.33%	83.11%	76.98%

4.3 System Level Experiment Results

4.3.1 Performance Results. In this section, a series of experiments were conducted to assess the performance of the proposed scheme at the system level. For the evaluated approaches, each page in the flash device can be refilled using the proposed fast retention-refilling function. Two crucial metrics, namely, the execution time for read/write requests and the time spent on retention-error management (including retention-refilling, in-place ISPP, and data relocation), are assessed and labeled as "R/W Execution" and "Data-Refresh", respectively. Figure 8 illustrates the performance results across different initial data ratios within the evaluated flash devices for all compared approaches under various benchmarks, with a focus on total execution time. All results are normalized against those of the *W/O refresh* approach, which takes no action to address retention error issues. The *W/O refresh* approach relies solely on natural data updates to refresh the retention capability, resulting in increased read/write execution time due to additional latency introduced by garbage collection. Conversely, the execution time for *Data relocation* can be significantly reduced. This is because data relocation consistently transfers retention-erred data to new data pages. When free space is insufficient, the device performs garbage collection to free up space. However, it's noteworthy that the data-refresh overhead of the *Data relocation* approach is notably high due to the frequent initiation of the data-refresh procedure to prevent data from succumbing to retention errors. On the other hand, when compared to the *In-place ISPP* or *Data relocation* schemes, our proposed retention refilling scheme effectively curtails data-refresh overheads without imposing substantial overhead on read/write execution times.

4.3.2 Overhead Analysis. To better understand the effectiveness of the evaluated schemes, let's delve into key metrics related to management overhead, which includes the number of triggered data refreshes, total block erase counts, and write amplification. These metrics are benchmarked in various configurations with different initial data ratios. Figure 9 provides insights into the total data refresh operations performed during retention error handling. As previously mentioned in Section 4.3.1, the *W/O refresh* scheme avoids triggering data-refresh procedures to prevent data retention errors, resulting in zero data refresh across all cases. Notably, both the *Retention-Refilling* and *In-place ISPP* schemes significantly reduce data-refresh instances when compared to the *Data relocation* approach. This reduction is due to these three approaches employing fine-grained data-refresh techniques for addressing retention-erred pages, eliminating the need for strategies involving a larger number of live-page copies. Consequently, the number of data refreshes is reduced by 20% to 81% under both the *Retention-Refilling* and *In-place ISPP* schemes when compared to the *Data relocation* scheme. It's worth highlighting that the *Retention-Refilling* scheme particularly excels in scenarios with a higher volume of reads but fewer writes, such as the Web+SQL trace, substantially reducing data-refresh instances. These results in data-refresh frequencies also impact another critical system overhead metric: write amplification, as shown in Figure 10. The *Retention-Refilling* scheme outperforms other approaches, with experimental results indicating write amplification ranging from 49% to 93% compared to alternative methods. Meanwhile, we also present the total erase count for the evaluated approaches across different benchmarks in Figure 11. Remarkably, the *Retention-Refilling* scheme effectively delays the need for executing data refreshing, limiting the total erase count to a maximum of 86% compared to the *W/O refresh* scheme.

4.3.3 Lifetime. Figure 12 presents the lifetime results regarding the first failure time, where lifetime is defined as the total execution time until any block in the simulated flash device wears out. All the results are normalized to the baseline *W/O refresh* scheme. Notably, the *Data relocation* scheme quickly depletes the limited program/erase cycles due to its frequent data-refresh procedure

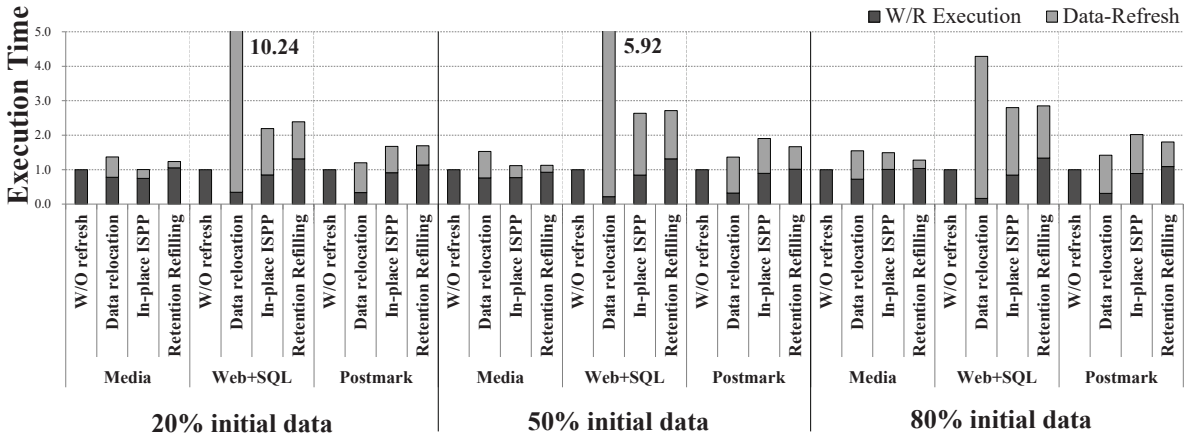


Figure 8: Execution time.

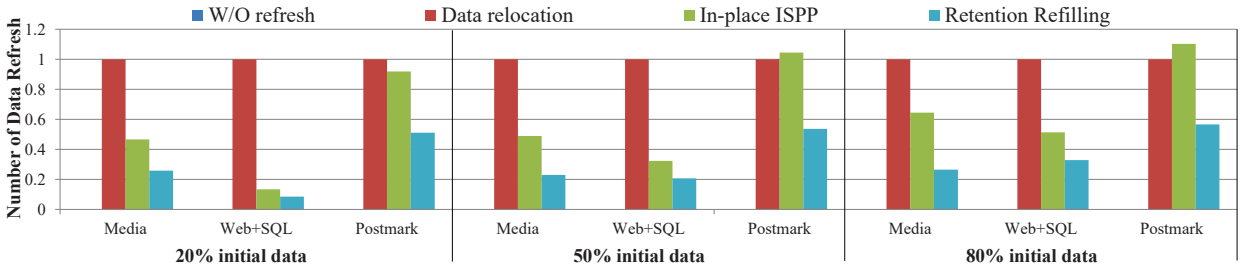


Figure 9: Number of data-refreshes.

to prevent retention errors. In contrast, the proposed *Retention-Refilling* scheme significantly extends device lifetime by effectively refilling the retention capability for cold data (as discussed in Section 3.4), slowing down the rate of program/erase cycle exhaustion. Compared to the *W/O refresh* scheme, the device lifetime can be extended by up to 60% under the evaluated benchmarks.

5 CONCLUSION

This paper introduces an innovative retention-refilling scheme designed to enhance data integrity by efficiently addressing the issue of left-shifted V_t distribution in retention-erred cells. We employ a fast retention-refill program function to effectively restore the retention capability of cells with such left-shifted V_t distribution. To seamlessly integrate this retention-refillable programming scheme into the existing Flash Translation Layer (FTL) with minimal overhead, we present a comprehensive management design for the retention-refilling module and block allocation/management policies. Extensive experiments confirm the effectiveness of our design, demonstrating its ability to extend device lifetime by up to 75%. Furthermore, our scheme generally incurs similar or lower performance overhead in terms of total execution time compared to in-place reprogramming in most scenarios.

REFERENCES

[1] 1998. Flash-memory Translation Layer for NAND flash (NFTL). *M-Systems* (1998).
[2] G. Aayush, K. Youngjae, and U. Bhuvan. 2009. DFTL: A Flash Translation Layer Employing Demand-based Selective Caching of Page-level Address Mappings. *SIGARCH Comput. Archit. News* (March 2009).
[3] A. Ban. 1995. Flash File System. US Patent 5,404,485. In *M-Systems*.
[4] A. Ban. 2018. Wear Leveling of Static Areas in Flash Memory. US Patent 6,732,221. *M-systems* (2018).
[5] Y. Cai, E. F. Haratsch, O. Mutlu, and K. Mai. 2012. Error patterns in MLC NAND flash memory: Measurement, characterization, and analysis. In *Proceedings of the Conference on Design, Automation and Test in Europe*.
[6] Y. Cai, G. Yalcin, O. Mutlu, E. F. Haratsch, A. Cristal, O. S. Unsal, and K. Mai. 2012. Flash correct-and-refresh: Retention-aware error management for increased flash memory lifetime. In *2012 IEEE 30th International Conference on Computer Design (ICCD)*.
[7] Y. H. Chang, J. W. Hsieh, and T. W. Kuo. 2010. Improving Flash Wear-Leveling by Proactively Moving Static Data. *Computers, IEEE Transactions on* (Jan 2010).
[8] Y. M. Chang, Y. H. Chang, T. W. Kuo, H. P. Li, and Y. C. Li. 2013. A Disturb-alleviation Scheme for 3D Flash Memory. In *Proceedings of the International Conference on Computer-Aided Design (ICCAD '13)*.
[9] Y. M. Chang, Y. C. Li, P. H. Lin, H. P. Li, and Y. H. Chang. 2016. Realizing Erase-free SLC Flash Memory with Rewritable Programming Design. In *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*.
[10] Tseng-Yi Chen, Yuan-Hao Chang, Yuan-Hung Kuan, and Yu-Ming Chang. 2017. VirtualGC: Enabling erase-free garbage collection to upgrade the performance of rewritable SLC NAND flash memory. In *Proceedings of the 54th Annual Design Automation Conference 2017*. ACM.
[11] Yajuan Du, Qiao Li, Liang Shi, Deqing Zou, Hai Jin, and Chun Jason Xue. 2017. Reducing LDPC soft sensing latency by lightweight data refresh for flash read performance improvement. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*.
[12] Z. Fan, G. Cai, G. Han, W. Liu, and Y. Fang. 2019. Cell-State-Distribution-Assisted Threshold Voltage Detector for NAND Flash Memory. *IEEE Communications Letters* (April 2019).
[13] L. Han, Y. Ryu, and K. Yim. 2006. CATA: A Garbage Collection Scheme for Flash Memory File Systems. In *International Conference on Ubiquitous Intelligence and Computing*.
[14] C. C. Ho, Y. C. Li, Y. H. Chang, and Y. M. Chang. 2018. Achieving Defect-free Multilevel 3D Flash Memories with One-shot Program Design. In *Proceedings of the 55th Annual Design Automation Conference (DAC '18)*.
[15] C. C. Ho, Y. C. Li, P. H. Lin, W. C. Wang, and Y. H. Chang. 2018. A Stride-Away Programming Scheme to Resolve Crash Recoverability and Data Readability

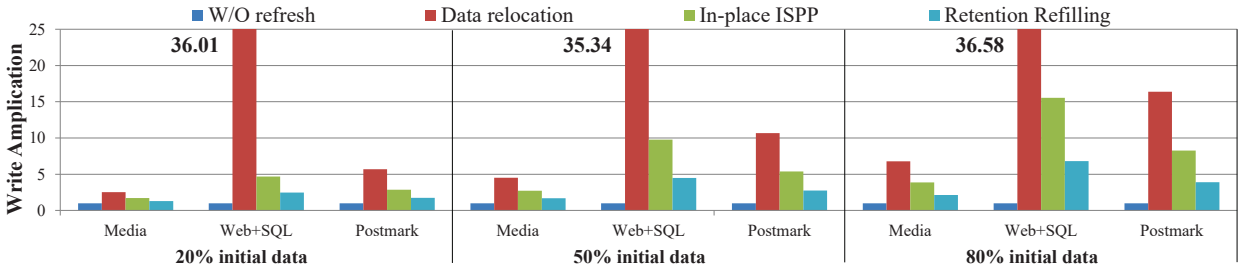


Figure 10: Write amplification.

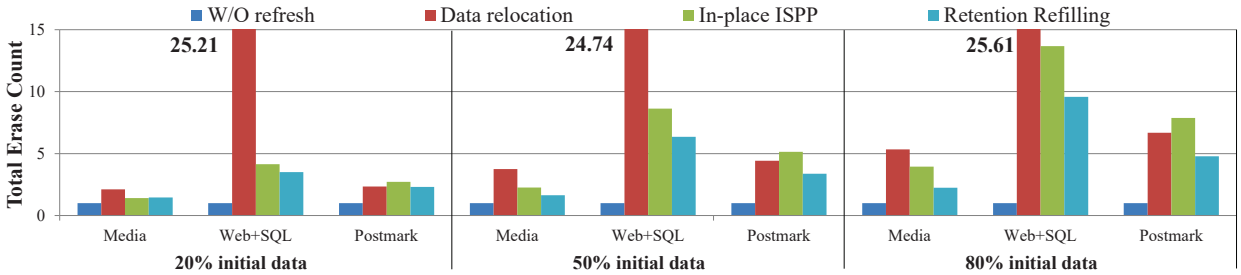


Figure 11: Number of total erase count.

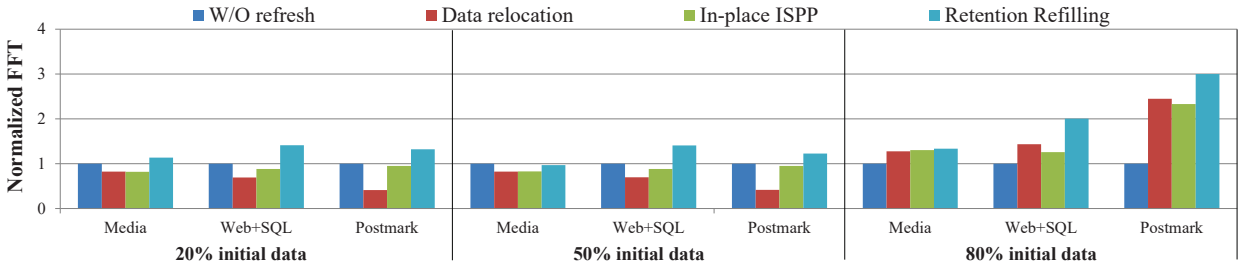


Figure 12: Lifetime.

Issues of Multi-Level-Cell Flash Memory.
[16] J. Katcher. 1997. Postmark: A new file system benchmark. (January 1997).
[17] A. Kawaguchi, S. Nishioka, and H. Motoda. 1995. A Flash-memory Based File System. In *Proceedings of the USENIX 1995 Technical Conference Proceedings (TCON'95)*.
[18] P. Li, Y. Zhang, D. Yin, and P. Xie. 2021. An Efficient Refresh Strategy of Flash Memory via High Delay Blocks in LDPC. In *2021 6th International Conference on Integrated Circuits and Microsystems (ICICM)*.
[19] R. S. Liu, C. L. Yang, and W. Wu. 2012. Optimizing NAND flash-based SSDs via retention relaxation. *Target* (2012).
[20] W. Liu, J. Rho, and W. Sung. 2006. Low-Power High-Throughput BCH Error Correction VLSI Design for Multi-Level Cell NAND Flash Memories. In *2006 IEEE Workshop on Signal Processing Systems Design and Implementation*.
[21] H. T. Lue, T. H. Hsu, S. Y. Wang, E. K. Lai, K. Y. Hsieh, R. Liu, and C. Y. Lu. 2008. study of incremental step pulse programming (ISPP) and STI edge effect of BE-SONOS NAND Flash. In *2008 IEEE International Reliability Physics Symposium*.
[22] Y. Luo, Y. Cai, S. Ghose, J. Choi, and O. Mutlu. 2015. WARM: Improving NAND flash memory lifetime with write-hotness aware retention management. In *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*.
[23] Yixin Luo, Saugata Ghose, Yu Cai, Erich F. Haratsch, and Onur Mutlu. 2018. HeatWatch: Improving 3D NAND Flash Memory Device Reliability by Exploiting Self-Recovery and Temperature Awareness. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.
[24] Y. Luo, S. Ghose, Y. Cai, E. F. Haratsch, and O. Mutlu. 2018. Improving 3D NAND Flash Memory Lifetime by Tolerating Early Retention Loss and Process Variation. In *Abstracts of the 2018 ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS '18)*.
[25] Y. Lv, S. Liang, L. Luo, C. Li, C. Xue, and H.-M. Sha. 2022. Tail Latency Optimization for LDPC-Based High-Density and Low-Cost Flash Memory Devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022).
[26] Y. Lv, L. Shi, Q. Li, C. Gao, C. J. Xue, and E. Sha. 2019. Optimizing Tail Latency of LDPC based Flash Memory Storage Systems Via Smart Refresh. In *2019 IEEE International Conference on Networking, Architecture and Storage (NAS)*.
[27] M. Murugan and D. H. C. Du. 2011. Rejuvenator: A Static Wear Leveling Algorithm for NAND Flash Memory with Minimized Overhead. In *MSST*.
[28] H. Qin, Y. Zhao, D. Feng, J. Liu, and W. Tong. 2020. CeSR + Assisted LDPC: A Holistic Strategy to Improve MLC NAND Flash Reliability. *IEEE Access* (April 2020).
[29] Exchange Trace. 2010. SNIA IOTTA Repository.
[30] A. Traeger, E. Zadok, N. Joukov, and C. P. Wright. 2008. A Nine Year Study of File System and Storage Benchmarking. *Trans. Storage* (May 2008).
[31] W. Wang, T. Xie, A. Khoueir, and Y. Kim. 2015. Reducing MLC flash memory retention errors through Programming Initial Step Only. In *2015 31st Symposium on Mass Storage Systems and Technologies (MSST)*.
[32] C. Wu, H. Lue, T. Hsu, C. Hsieh, W. Chen, P. Du, C. Chiu, and C. Lu. 2016. Device Characteristics of Single-Gate Vertical Channel (SGVC) 3D NAND Flash Architecture. In *2016 IEEE 8th International Memory Workshop (IMW)*.
[33] C. Yu, Y. Gulay, M. Onur, H. Erich F., C. Adrian, U. Osman S., and M. Ken. 2013. ERROR ANALYSIS AND RETENTION-AWARE ERROR MANAGEMENT FOR NAND FLASH MEMORY. *Intel Technology Journal* (2013).