# Dynamic Partitioning to Mitigate Stuck-at Faults in Emerging Memories

Jiangwei Zhang*†, Donald Kline, Jr †, Liang Fang*, Rami Melhem‡, and Alex K. Jones†
State Key Laboratory of High Performance Computing, College of Computer, National University of Defense Technology, China*
Department of Electrical and Computer Engineering, University of Pittsburgh, USA †
Department of Computer Science, University of Pittsburgh, USA ‡
Email: jiz148@pitt.edu, dek61@pitt.edu, lfang@nudt.edu.cn, melhem@cs.pitt.edu, akjones@pitt.edu

***Abstract*—Emerging non-volatile memories have many advantages over conventional memory. Unfortunately, many are susceptible to write endurance challenges, resulting in stuck-at faults. Existing mitigation methods statically partition and invert data within a block containing such faults (partition-and-flip) to ensure data is written to match stuck-at cells such that they may remain in service. Unfortunately, these schemes have limited fault tolerance capabilities and require the assumption that their auxiliary bits are fault free. We propose a *dynamic* partitioning scheme that improves the number of tolerated stuck-at faults and simultaneously protects auxiliary bits. Dynamic partitioning can significantly improve the fault tolerance over existing static partitioning approaches with an equal number of auxiliary bits. Moreover, it can often still improve fault tolerance while reducing the number of auxiliary bits. Compared to flip-N-write and Aegis, a leading mitigation scheme, dynamic partitioning can achieve 7-72% and 5-53× lower write error rates, respectively, for the same capacity overhead with a stuck-at-fault rate of $10^{-3}$.**

***Index Terms*—Emerging Memories, Reliability, Stuck-at Faults, and Dynamic PartitioninEmerging Memories, Reliability, Stuck-at Faults, and Dynamic Partitioning**

## I. Introduction

DRAM and flash scaling is greatly hindered by physical limitations, among which limited charge is a major issue that lowers the retention time and makes the charge sensing unreliable [1], [2]. Emerging memories such as phase change memory (PCM) and memristors (RRAM) with great scalability, high density, and non-volatility are promising to replace DRAM and flash as main memory or secondary storage [3]–[6]. However, before PCM or RRAM becomes a mature alternative, one of their crucial shortcomings, limited write endurance, must be conquered [4]. A typical PCM cell can sustain $10^8$ to $10^9$ writes before it becomes stuck at '0' or '1'. While each individual cell has an optimal RESET current, the minimal current to perform the full reset operation, each optimal RESET current deviates from the group average due to process variation. Over-programming, which can be exacerbated by the variation in optimal RESET current, is a major cause of early failure of weak cells leading to permanent stuck-at faults.

When a cell is stuck at a value, the value still can be read, but can not be changed. When performing a write operation on a stuck-at bit, if the written value is the same as the stuck-at value (stuck-at-right, SA-R), an error does not occur, but if the two values are opposite (stuck-at-wrong, SA-W), there will be an error. In this case, the written bit can be stored as its inverse, and be marked as such using a flag (auxiliary) bit. In this way, whatever the stuck value is, this bit can still be stored. Partition-and-flip (PAF) schemes [7]–[9] are approaches that utilize this characteristic of stuck-at faults. They first use an efficient method to partition SA-Rs and SA-Ws into different groups, and then invert groups with any SA-W, setting the flag bits appropriately. Among the representative PAF schemes, flip-N-write [7] is the simplest but it is insufficient to protect memory with high fault rates, while Aegis [9] has more complex partitioning methods but is very efficient at fault mitigation.

Encoding and correction (EAC) schemes represent another approach that can be used to mitigate stuck-at faults; the methods which comprise this approach include error correction coding (ECC) [10], coset encoding [11], and PRES [12]. ECC is typically employed to protect memory against transient or bus-related faults, which are rare relative to stuck-at faults in PCM or RRAM. Another correction scheme, error correction pointers (ECP) [13] uses a pointer to address the stuck-at bit and an extra bit to replicate the data within its protected data block. Though ECP is effective to tolerate stuck-at faults, the high capacity overhead is always a burden for memory, compared to PAF schemes [9].

In this paper, we propose a partition method that dynamically changes the grouping of bits to mitigate more stuck-at faults in a data block. It protects the auxiliary bits and improves the effectiveness of the existing PAF schemes by generating more efficient configurations. The paper makes the following contributions:

1) It proposes a dynamic partition scheme to tolerate stuck-at faults which can be applied to several existing partitioning schemes to enhance their effectiveness
2) It describes applications of the dynamic partition scheme to flip-N-write and Aegis, showing the great adaptability of the proposed scheme
3) It provides a characterization of the recovery from stuck-at faults of the dynamic schemes and conducts an extensive study, illustrating the significant improvements brought by the dynamic partitioning strategy

978-1-5386-3093-8/17/$31.00 ©2017 IEEE

## II. Background

Error Correction Codes (ECC) [10] are general approaches that are used to protect memories from transient faults, but can also be applied to correct stuck-at faults. Among these ECC schemes, the (72, 64) Hamming Coding scheme is the most frequently used. When it is applied in a scenario where it is used to correct stuck-at faults, it can recover one SA-W and tolerate any SA-R within those 72 bits. When the fault rate of the memory is below a threshold of $10^{-6}$, this scheme is sufficient for fault recovery [14], but when the rate exceeds the threshold, this scheme is not adequate. PCM and RRAM face the problem of device variations, causing the endurance of weak cells to be much lower than the average level, leading to their early failures. Moreover, the auxiliary bits in the ECC schemes are written more frequently than data bits, so ECC is not suitable for protecting stuck-at faults in emerging memories like PCM and RRAM.

The error correction pointers (ECP) [13] scheme uses several bits as a pointer to record the address of a faulty bit and an extra bit to store the data as a replacement. In addition to these bits, ECP also needs one bit to mark whether all of the pointers are in use. In a 512-bit data block, to protect $N$ faulty bits, ECP needs $N\times10$+1 bits to ensure protection. However, this calculation is based on the assumption that the auxiliary bits are not faulty. A SA-W fault in a pointer of ECP requires two fault-free pointers to fix the error: one to point to the original intended location of the first pointer, and the other to point to the location the first pointer mistakenly overwrote. While this provides some level of protection to the ECP bits, it is usually insufficient with few pointers and/or for large error rates.

Partition-and-flip (PAF) schemes leverage the knowledge that the stuck-at faults are still readable, and can store the correct data as long as it aligns with the stuck-at value. Flip-N-write (FNW) [7] was originally developed for reducing energy by minimizing bit changes between a currently stored value and the new value intended to be written. FNW is amongst the simplest PAF schemes. While FNW has a straightforward and low overhead implementation, it can be ineffective when faults are clustered. Despite this limitation, many existing schemes have utilized FNW as a base operation for a more complicated partitioning scheme, including RDIS [15], SAFER [8], and Aegis [9].

RDIS transforms an one-dimensional (1D) data block into a two-dimensional (2D) matrix. Each row or each column might be inverted if there are stuck-at faults by using a flag bit to record the information of its inversion status. Because of the advantage of a 2D matrix putting each bit in two groups simultaneously, this scheme may tolerate more than one stuck-at fault which are in the same row or column. The shortcoming of this scheme is the large capacity overhead. For a 512-bit data block, it needs at least 46 flag bits, a 9% capacity overhead, even when the fault rate is not high.

SAFER and Aegis use relatively complex but efficient partitioning methods to partition stuck-at faults into different groups, which are then protected by one FNW flag bit per group. SAFER starts with a maximum number of available group partitions. Whenever a new fault occurs, the groups are repartitioned according to the XOR of the pointers to the fault locations in the data blocks, aiming to partition the stuck-at faults so that the maximum number of stuck-ats per group is one. If the number of faults is larger than the maximum number of groups, SAFER cannot always guarantee correction.

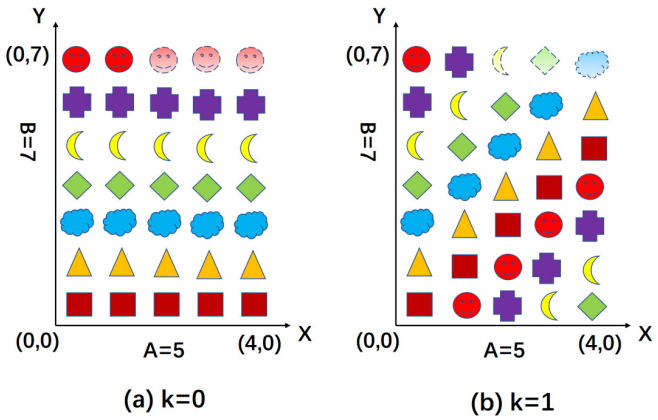


Fig. 1: An example of Aegis partitioning of a 32-bit block into $7 \times 5$ matrices with different slopes. Each bit is represented by a symbol except for the three dotted ones on the top right (not used). Different symbols represent different groups of the partitioning. In total, there are 7 configurations corresponding to 7 slopes, i.e. k ranges from 0 to 6.

Aegis interprets a 1D data block as a 2D matrix. Inspired by the principle that any two points in a line on a Cartesian plane determine the slope of the line, Aegis uses different slopes to generate different configurations (partitions) ensuring that all possible combinations of two bits which are in the same group of one configuration would not be in the same group of the other configurations. In Figure 1, as an example, we illustrate how to partition a 32-bit data block into $7 \times 5$ matrices according to Aegis. If there are $N$ stuck-at faults in a data block, Aegis guarantees the faults will be partitioned into different groups when there are at least $(N-1)N/2+1$ configurations [9]. Compared to SAFER, Aegis is superior because it creates a better distribution of stuck-at faults with equivalent capacity overhead.

## III. Dynamic Partitioning Scheme

Our dynamic partitioning scheme aims to distribute stuck-at faults to different groups by dynamically changing partition sizes and orientations within an existing PAF scheme to improve the fault-tolerating effectiveness without increasing the capacity overhead. Leveraging variable sized partitions can diversify the correction capability of existing partition based schemes. Dynamic partitioning is accomplished by splitting the auxiliary bits into two segments: the first segment, S1, specifies how many unique partitions are used; the second segment, S2, contains the auxiliary bits needed to flip each of the groups within the partition.

### A. Dynamic FNW

FNW is implementable with low capacity overhead, but it has only one configuration for a given partition size. If the dynamic partitioning strategy is applied to FNW, any one of a number of different partitions can be used to improve the scheme's effectiveness.

Dynamic partition-based FNW ($\text{FNW}_{\text{DY}}$) is a simple and effective application of our dynamic partitioning scheme. We first briefly describe the static FNW scheme. For a 512-bit data block with 10 auxiliary bits ($N = 10$), every 52 adjacent bits will be treated as being in the same group. Each group has a corresponding auxiliary bit, which indicates whether the entire group is flipped or not to attempt to avoid writing any SA-W in the block. Unfortunately, FNW cannot protect against stuck-at faults when there is at least one SA-R and one SA-W together in one group, or when there is one SA-W or SA-R bit in a group while the auxiliary bit for that group is stuck-at '0' or '1'.

Figure 2 is an example to demonstrate how $\text{FNW}_{\text{DY}}$ would protect a 24-bit data block of data assuming 10 auxiliary bits. Among the auxiliary bits, two bits in S1 are used to identify the number of groups used in the partition and the remaining eight bits in S2 control the inversion. The bits in S1 count down from the maximum number of partitions, in this case eight. Thus for the example, the data can be partitioned into eight, seven, six, and five groups. Note that partitions with 1, 2, 3 or 4 groups do not provide any benefit as a partition with $k$ groups cannot mitigate more faults in a data block than a partition with a multiple of $k$ groups.

The figure displays two group partitioning options '00' represents an 8-group partition (8-0) in the data block, while '10' represents a 6-group partition (8-2). In the 24-bit data block, the bits are labeled from 0 to 23. Dotted lines and solid lines correspond to boundaries of 8-group and 6-group partitions, respectively. In S2 of the auxiliary bits, 'X' indicates that the stuck-at faults within that data group cannot be corrected, while 'd' (don't care) means that the value of the auxiliary bit does not matter because the data group could be stored as original or inverted. In the data block, 'W' and 'R' represent SA-W and SA-R, respectively. The 8 sets of curved lines which connect two memory cells indicate the cases where the 6-group partition can tolerate a SA-W and SA-R, but the 8-group partition fails to correct them. For example, bits '15' and '16' (marked in purple and are SA-W and SA-R, respectively) can be tolerated by the 6-group partition with the fourth group inverted and the fifth group not inverted, but cannot be mitigated by the 8-group partition because these two bits are in the same group (the corresponding flag bit is marked as 'X'). Similarly, bits '2' and '3' can always be corrected by the 8-group partition, but cannot be corrected by the 6-group partition.

In general, for a large block, using several bits to indicate the partition can be more beneficial than using those bits to have a smaller but fixed partition size with several extra groups. Note that the number of bits in S1 can be adjusted according to different data block sizes and overhead capacity. For example, for a 512-bit data block, when there are 21 overhead bits, 3 auxiliary bits can be used to record different configurations.

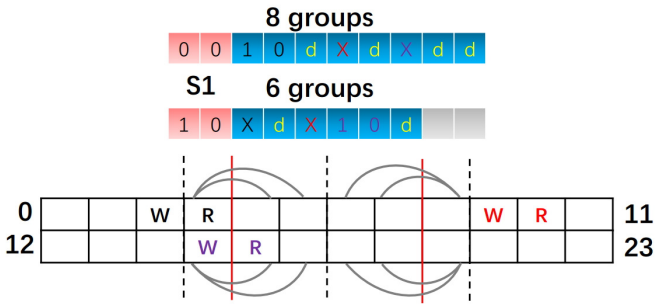


Fig. 2: Illustration of dynamic FNW protecting a 24-bit data block with 10 auxiliary bits, and cases where a smaller FNW partition size can outperform a larger one.

### B. Dynamic Aegis

The number of configurations available to Aegis is determined by the width of the 2D rectangle used to organize the bits in the data block. The partitioning of Aegis has two restrictions on this rectangle width: it needs to be a prime number, and this prime number must be greater than or equal to the length of the rectangle. Using widths with these characteristics ensures for Aegis that any two bits in the same group of one configuration will not be in the same group in another configuration. The dynamic partition strategy applied to Aegis does not limit itself to prime widths, using multiple combinations of prime and non-prime widths for a larger number of available partition configurations. Our dynamic partitioning strategy enhances the existing advantages of Aegis over competing partitioning schemes such as SAFER and RDIS.

Figure 3 illustrates how a 32-bit data block is partitioned into different configurations according to *Static* (Original) Aegis and *Dynamic* Aegis ($\text{Aegis}_{\text{DY}}$). For this block size, Aegis uses 10 auxiliary bits for fault correction, including 7 flag bits and 3 slope bits. For each slope, the data block is partitioned into a $7 \times 5$ matrix. The lengths (A) and the widths (B) of the matrices are marked, while the available slopes (k) are also shown. The numbers for the bits in the matrices represent their group identifiers (ID) for the slope shown in red and recorded in the slope bits ('000'). The bits labeled 'X' are bits which are not in the 32 bit data block.

In the case shown, $\text{Aegis}_{\text{DY}}$ uses 2 bits in S1 and the other 8 bits in S2. S2 is then partitioned into two subsegments, the flag bits and slope bits. The 2 bits in S1 represent the size of the matrix available, such as '10' for the $6 \times 6$ matrix. The number of slopes for each partition size is limited by the bits left over after removing the flag bits needed for the number of groups. For example, the $6 \times 6$ matrix has 2 bits available as slope bits (4 slopes), when it could have used 6 different slopes, and the $5 \times 7$ matrix has 3 bits available (5 slopes) limited by the number of groups (5) in the matrix. Because the group partitioning encoded in S1 ranges from 8 to 5, the number of

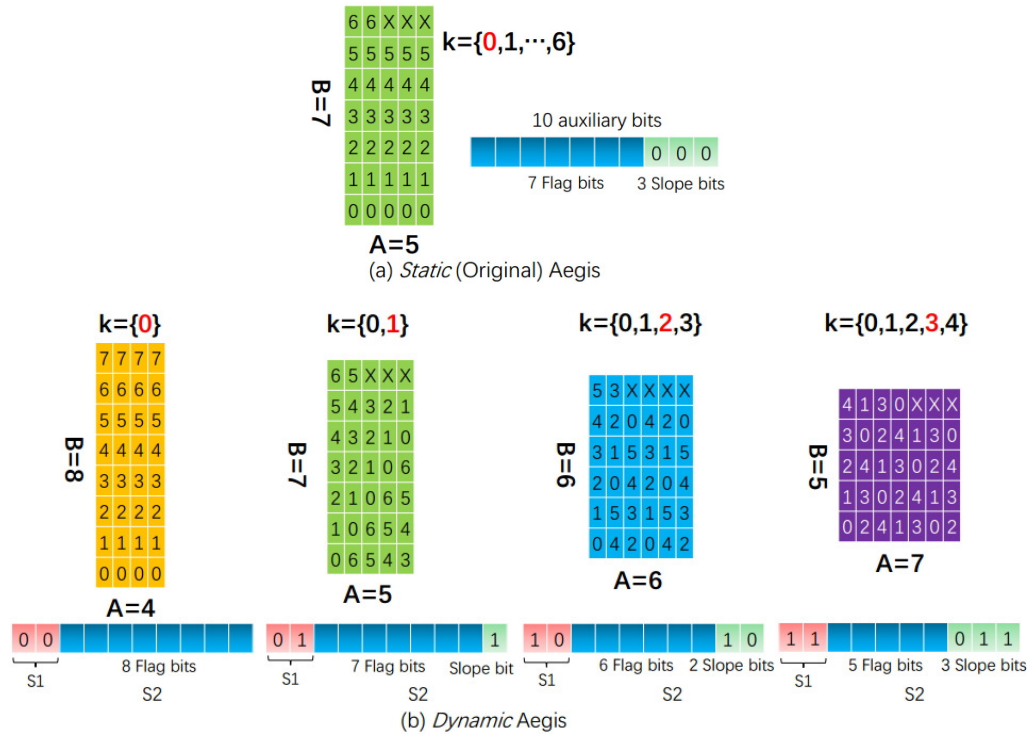


Fig. 3: Illustrating how to partition a 32-bit data block into different configurations according to (a) *Static* (Original) Aegis and (b) *Dynamic* Aegis.

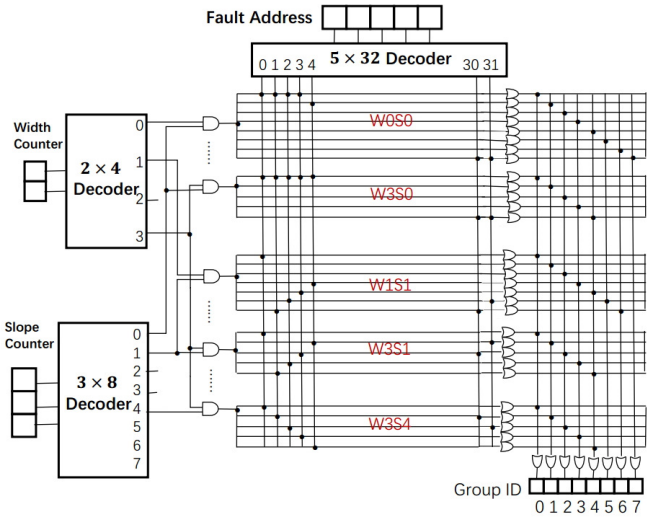


Fig. 4: An implementation logic to know the ID of a stuck-at fault at a specific data in S1 and an available slope for Aegis$_{DY}$. A $49 \times 32$ size ROM and a $49 \times 7$ ROM are used as lookup tables.

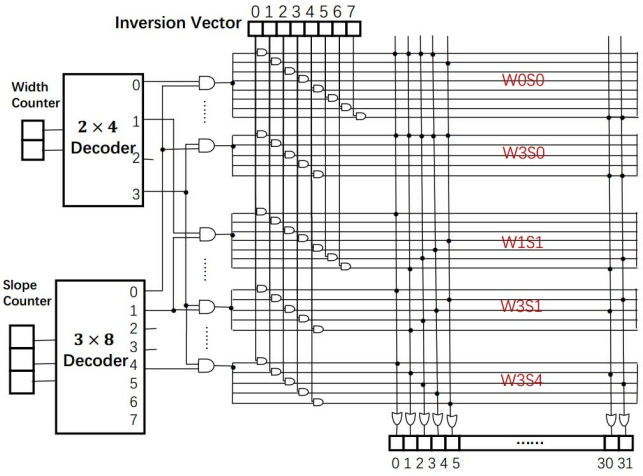


Fig. 5: An implementation logic to know which bits among a 32-bit data block should be written in their inverted forms. A $49 \times 32$ size ROM is used as a lookup table.

available configurations equals 1+2+4+5=12, as compared to Aegis, which only has 7 configurations.

In Figure 4, we show encoding logic for a 32-bit data block to know the ID of a stuck-at fault at a specific width in S1 and an available slope for Aegis$_{DY}$. For each stuck-at fault, we can get the corresponding ID. If there is any ID collision of SA-R and SA-W, we increment the slope counter. If the slope surpasses the maximum available value at the data in S1 (see Figure 3), we increment the width counter and reset the slope. In this manner, we search the possible data patterns in S1 and their corresponding available slopes until we find a configuration where no collision happens, which tolerates all the stuck-at faults in the block. The worst case encoding includes the traversal of all possible configurations, but in practice for lower error rates, this process needs to be repeated very few times. In Figure 5, we also display the decoding

logic to know which bits among a 32-bit data block should be written in their inverted forms. Note that decoding is more efficient than encoding because it only needs to perform one lookup. In each implementation, we use a ROM lookup table to record the relationship between the input and the output information. The ROM capacity overhead is discussed as part of the following section.

## IV. Evaluation

To validate our proposed dynamic partitioning strategy, we evaluate its effectiveness for both $\text{FNW}_{\text{DY}}$ and $\text{Aegis}_{\text{DY}}$, by comparing with their static counterparts of FNW and Aegis and baselines of ECP and ECC. Our experiments consider stuck-at fault rates of $10^{-3}$ and $10^{-4}$ with different auxiliary bits. Additionally, we provide a sensitivity study of stuck-at faults rates of $10^{-5}$ and $10^{-6}$.

### A. Experimental Methodology

We developed a PIN-based simulator [16] and implemented ECC, ECP, FNW, $\text{FNW}_{\text{DY}}$, Aegis, and $\text{Aegis}_{\text{DY}}$ to evaluate their tolerance to stuck-at faults. The PIN simulator evaluates main memory writes by encoding or partitioning the data block and the auxiliary bits, and recording a fault if the value of any fault bit being written is opposite to its stuck-at value. To model the stuck-at faults, a fault map, including fault bits stuck at '0' or '1', is developed by using Bayesian distribution to mimic the impact of process variation and includes spatial correlation of faults [17], [18]. In particular for this work we followed the model described in [17] to generate maps of weak cells for a 4GB PCM.

*1) Model Implementation Details:* In our evaluation, stuck-at faults can be tolerated in a data block for each of the schemes as follows:

1) For a Hamming code based error correction (ECC-$1_{64}$), one SA-W and any number of SA-Rs can be tolerated in each data block or its parity bits.
2) For $\text{ECP}_N$, $N$ SA-Ws can be tolerated in the data block assuming no SA-Ws compromise the auxiliary bits.
3) For FNW, there is no group that has both SA-Ws and SA-Rs in the group's data and its corresponding flag (encoding) bit.
4) For $\text{FNW}_{\text{DY}}$, there is a configuration that no group of the configuration has both SA-Ws and SA-Rs in the group data and its corresponding flag bit. The auxiliary bits (in S1) to record the group partitioning may not have any SA-Ws.
5) For Aegis, there is a configuration that no group of the configuration has both SA-Ws and SA-Rs in the group data and its corresponding flag bit. The slope bits may not have any SA-Ws.
6) For $\text{Aegis}_{\text{DY}}$, there is a configuration that no group has both SA-Ws and SA-Rs in the group data and its corresponding flag bit. The auxiliary bits in S1 and the slope bits in S2 may not have any SA-Ws.

The data block size examined in the study was 512 bits. For FNW, $\text{FNW}_{\text{DY}}$, and $\text{Aegis}_{\text{DY}}$, we used 10, 15, 21, 28, and 36 overhead bits per data block to tolerate stuck-at faults. These overhead ratios are 1.96%, 2.93%, 4.10%, 5.47%, and 7.03%, respectively. Aegis required a minimum of 23-bit encoding overhead for a 512-bit block to guarantee that each possible slope would have valid partitioning as discussed in section III-B. For fewer encoding bits, this correction can not be guaranteed as the group size is not sufficiently large to guarantee independent slopes, however, it is still effective. For comparison, we relax this requirement and also allow Aegis to use 10, 15, and 21 auxiliary bits per data block, generated in the same manner as described in [9].

For comparison we provide results for ECC-$1_{64}$, which requires 64 bits per data block, with an additional eight parity bits for one-bit error correction and two-bit error detection, corresponding to an overhead ratio of 12.5%[1]. $\text{ECP}_N$ requires $N\times10$+1 bits per data block. We compare our proposed schemes with $\text{ECP}_N$ such that ECP requires the minimum auxiliary bits that exceeds the auxiliary bits of our scheme (e.g., a 15-bit encoding would compare to $\text{ECP}_2$ that requires 21 bits).

In our experiments we perform two kinds of evaluations. First, we evaluate the memory accesses for the PARSEC benchmark suite for different fault maps. The entire benchmark suite is executed and error rate is determined by accesses with uncorrectable errors compared to total accesses. Second, we calculate what we call a "true random error rate" which we define as the error rate if a perfect distribution of all possible values were applied to each location of a fault map through an exhaustive search. For each fault rate, we use the average error rates for five fault maps to evaluate and compare the effectiveness of the different fault recovery schemes.

*2) Hardware Implementation of $\text{Aegis}_{DY}$:* To provide an ISO-overhead comparison for dynamic encoding for Aegis, we implemented the encoding and decoding look-up tables (ROMs) for Aegis and $\text{Aegis}_{\text{DY}}$ in 45nm CMOS technology shown in Table I. This was generated by running the ROM lookup table designs in Synopsis Design Compiler targeting a 45nm FreePDK [19]. $\text{Aegis}_{\text{DY}}$ contains more configuration options for identical bit overhead, and thus requires additional look-up table area. The encoding latency reported is for evaluating one data partitioning option; while encoding for Aegis or $\text{Aegis}_{\text{DY}}$ multiple encoding options may need to be explored to find a successful encoding which partitions SA-Rs and SA-Ws into separate groups. However, encoding is not on the critical path, and for lower error rates it is rare to require many encoding attempts.

Table I also enumerates the area and latency comparisons for decoding. For both 28 and 36 auxiliary bits, $\text{Aegis}_{\text{DY}}$ has a significant increase in latency over Aegis, while 15 and 21 auxiliary bits in $\text{Aegis}_{\text{DY}}$ has approximately the same delay as 28 and 36 bits for Aegis, respectively. In the following evaluation, we will demonstrate that even for these

[1]We also considered ECC-$1_{256}$ to achieve a similar overhead (auxiliary bits) compared to Aegis and FNW. However, ECC performed so poorly and it was more appropriate to compare with the more common (64,72) ECC at higher overhead.

TABLE I: Area and latency comparisons between Aegis and Aegis$_{DY}$ ROMs for encoding and decoding (Minimum Area Implementation).

| | | *ENCODING* | | | *DECODING* | |
|---|---|---|---|---|---|---|
| | *Aux* | *Opt* | $Area_{um^2}$ | $Lat_{ns}$ | $Area_{um^2}$ | $Lat_{ns}$ |
| Aegis | 10 | 7 | $1.33{\cdot}10^4$ | 1.87 | $1.31{\cdot}10^4$ | 1.34 |
| | 15 | 11 | $4.37{\cdot}10^4$ | 2.56 | $4.20{\cdot}10^4$ | 1.73 |
| | 21 | 17 | $1.10{\cdot}10^5$ | 2.90 | $1.06{\cdot}10^5$ | 1.91 |
| | 28 | 23 | $1.69{\cdot}10^5$ | 3.12 | $1.61{\cdot}10^5$ | 1.98 |
| | 36 | 31 | $2.29{\cdot}10^5$ | 4.51 | $2.13{\cdot}10^5$ | 2.57 |
| Aegis$_{DY}$ | 10 | 12 | $3.26{\cdot}10^4$ | 2.59 | $3.17{\cdot}10^4$ | 1.85 |
| | 15 | 41 | $1.16{\cdot}10^5$ | 2.91 | $1.12{\cdot}10^5$ | 1.97 |
| | 21 | 65 | $2.66{\cdot}10^5$ | 4.32 | $2.57{\cdot}10^5$ | 2.77 |
| | 28 | 88 | $4.02{\cdot}10^5$ | 3.72 | $3.77{\cdot}10^5$ | 2.72 |
| | 36 | 112 | $8.78{\cdot}10^5$ | 7.11 | $8.25{\cdot}10^5$ | 3.94 |

iso-performance comparisons, Aegis$_{DY}$ can achieve improved reliability compared to Aegis.

### B. Benchmark Evaluation

In this section, we evaluate the effectiveness of the error mitigation strategies described in Section IV-A1 for the PARSEC benchmark suite. We obtain the error rates for initial stuck-at-fault rates of $10^{-3}$ and $10^{-4}$ shown in Figure 6 and 7, respectively. In each figure the fault mitigation schemes are compared to a baseline of ECC-$1_{64}$ shown with a green line.

For the $10^{-3}$ initial stuck-at-fault rate (Figure 6) all the schemes outperform ECC-$1_{64}$ even though it requires the largest capacity overhead. While FNW does not obviously improve over ECC-$1_{64}$, FNW$_{DY}$ does provide improvements over *static* FNW for 10 and 15 auxiliary bits cases with larger improvement margins for 28 and 36 auxiliary bits, amounting to a 64% and 90% improvement, respectively. Recalling that (apart from 21 auxiliary bits), ECP$_N$ uses slightly more auxiliary bits than the other schemes, ECP$_N$ outperforms both ECC-$1_{64}$ and FNW$_{DY}$ but is far inferior to Aegis, which achieves more than a $20\times$ improvement over the next leading candidate. However, our dynamic partitioning, Aegis$_{DY}$, far outstrips Aegis. With 10 and 15 auxiliary bits, Aegis$_{DY}$ achieves $5\times$ and $372\times$ lower error rates than its static counterpart, respectively. When there are 21 auxiliary bits, Aegis$_{DY}$ completely recovers from all the stuck-at faults in the fault maps, while static Aegis still has a higher than $10^{-5}$ error rate. They both achieve perfect protection when there are 28 and 36 auxiliary bits.

For an initial stuck-at-fault rate of $10^{-4}$ (Figure 7), we see a similar trend, as the $10^{-3}$ case, except that ECC-$1_{64}$ is more effective than FNW, and FNW$_{DY}$ with a minimum of 28 bits is necessary to achieve a better result. Unsurprisingly, Aegis and Aegis$_{DY}$ achieve perfect protection with fewer auxiliary bits, while only *static* Aegis with 10 auxiliary bits sees any faults, but still achieving an uncorrectable error rate of $10^{-5}$. While for the same number of auxiliary bits, clearly Aegis$_{DY}$ provides better protection than *static* Aegis, a method to distinguish

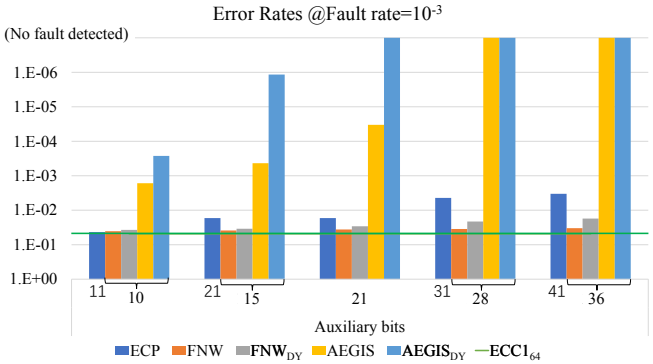


Fig. 6: Comparisons of different recovery schemes on the PARSEC benchmarks for a $10^{-3}$ fault rate.

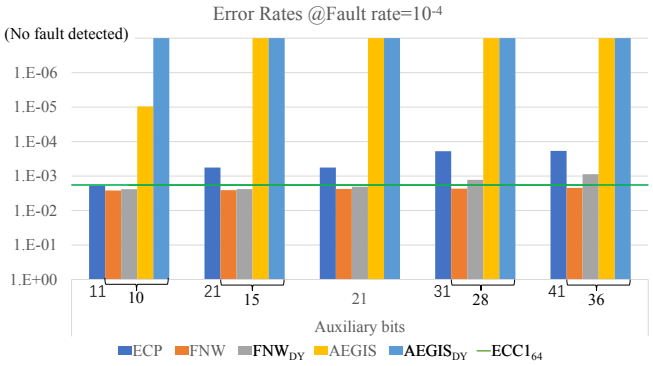


Fig. 7: Comparisons of different recovery schemes on the PARSEC benchmarks for a $10^{-4}$ fault rate.

lower uncorrectable error rates is necessary. We make this comparison with a probabilistic study in the next section.

### C. Probabilistic Evaluation

In this evaluation, we simulate all possible data patterns in every row of the generated memory model and discover the "true error rates" of a perfectly even data distribution and to distinguish lower error rates than exhibited through benchmark evaluation. Using true error rates, first, we compare the effectiveness of the error correction schemes with ISO auxiliary bits. Second, we compare the effectiveness of the *dynamic* schemes with their counterparts for an ISO performance comparison. The ISO performance comparison also provides an advantage as it requires a lower auxiliary bit storage overhead for Aegis$_{DY}$.

*1) ISO Auxiliary Bits Comparison:* Figure 8 and 9 show the comparisons of the various recovery schemes at the initial fault rates of $10^{-3}$ and $10^{-4}$, respectively. FNW$_{DY}$ protected 68% and 72% more data patterns than *static* FNW with 28 auxiliary bits at the two fault rates, while the improvements are 66% and 70% with 36 auxiliary bits demonstrating the value of the dynamic partitioning strategy. For Aegis, dynamic partitioning is even more striking, with Aegis$_{DY}$ obtaining $14\times$ and $47\times$ lower error rates, respectively, at the two fault rates for 28 auxiliary bits, and $21\times$ and $120\times$ lower error rates, respectively, for 36 auxiliary bits.

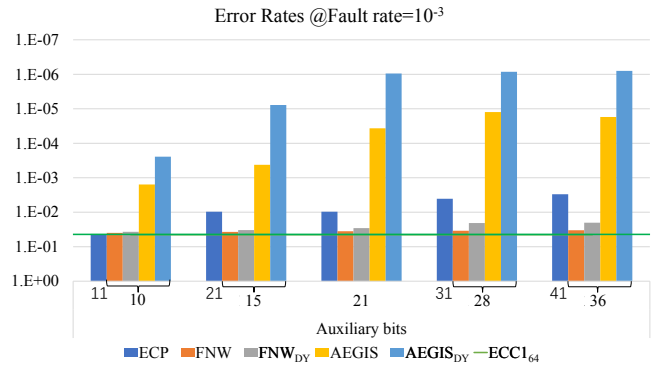


Fig. 8: Comparisons of different recovery schemes at fault rate $10^{-3}$ using ISO auxiliary bits.

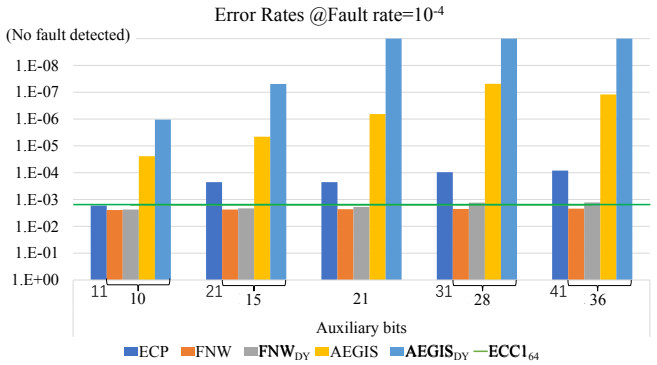


Fig. 9: Comparisons of different recovery schemes at fault rate $10^{-4}$ using ISO auxiliary bits.

In Figures 8 and 9, the correction capabilities shown for Aegis with 28 auxiliary bits are larger than those for Aegis with 36 auxiliary bits, which is counterintuitive. The source of this problem is faults in the auxiliary bits: when the same simulation was run with the auxiliary bits never able to have faults, the results were identical to those shown in [9]. Further, Aegis has a fundamental limitation with weaknesses in auxiliary bits: bits which are always in the same *number* group (e.g. the first bit of each group) can never be fixed if the auxiliary bit for that group has a stuck-at fault which prevents that group from being fixed. Moving from 28 to 36 adds 8 more groups, and thus contributes to this error. Note that for Aegis$_{\text{DY}}$, this does not occur except the bottom left bit.

To illustrate this advantage we compare the error rates of the static and dynamic schemes in Table II. In the table, *FR* is the initial fault rate, *Both* refers to faults corrected by both the static and dynamic schemes, *Dynamic* refers to faults only corrected by the dynamic scheme, *Static* refers to faults corrected only by the static scheme, and *Neither* refers to faults uncorrectable by either scheme. If the *Dynamic* column is much larger than the *Static* column, it illustrates superiority of the dynamic scheme. For FNW, the numbers for FNW$_{DY}$ are $21\times$ larger than *static* FNW, on average. For Aegis, Aegis$_{DY}$ is $64\times$ better than *static* Aegis at stuck-at-fault $10^{-3}$, while Aegis$_{DY}$ achieves perfect protection but Aegis still fails to protect few data patterns at stuck-at-fault $10^{-4}$.

TABLE II: Comparisons of FNW$_{\text{DY}}$ and Aegis$_{\text{DY}}$ with their static counterparts. 'none' means no fault detected for five 256MB fault maps.

| | *FR* | *Aux* | *Both* | *Dynamic* | *Static* | *Neither* |
|---|---|---|---|---|---|---|
| FNW | $10^{-3}$ | 28 | 0.96 | $1.48{\cdot}10^{-2}$ | $8.70{\cdot}10^{-4}$ | $1.97{\cdot}10^{-2}$ |
| | | 36 | 0.97 | $1.38{\cdot}10^{-2}$ | $5.03{\cdot}10^{-4}$ | $1.96{\cdot}10^{-2}$ |
| | $10^{-4}$ | 28 | 0.998 | $1.00{\cdot}10^{-3}$ | $6.00{\cdot}10^{-5}$ | $1.24{\cdot}10^{-3}$ |
| | | 36 | 0.998 | $9.37{\cdot}10^{-4}$ | $3.57{\cdot}10^{-5}$ | $1.24{\cdot}10^{-3}$ |
| Aegis | $10^{-3}$ | 28 | 0.999987 | $1.17{\cdot}10^{-5}$ | $7.16{\cdot}10^{-7}$ | $6.76{\cdot}10^{-7}$ |
| | | 36 | 0.999983 | $1.66{\cdot}10^{-5}$ | $1.45{\cdot}10^{-7}$ | $6.52{\cdot}10^{-7}$ |
| | $10^{-4}$ | 28 | $1\text{-}4.83{\cdot}10^{-8}$ | $4.83{\cdot}10^{-8}$ | none | none |
| | | 36 | $1\text{-}1.21{\cdot}10^{-7}$ | $1.21{\cdot}10^{-7}$ | none | none |

TABLE III: Improvement ratio in fault rate of Aegis$_{\text{DY}}$ over Aegis for different auxiliary bits at an initial stuck-at rate of $10^{-3}$. For example, 4.75 indicates the fault rate of Aegis divided by the fault rate of Aegis$_{\text{DY}}$ is 4.75, at the respective auxiliary bits.

| | Aegis *Aux* | | | | |
|---|---|---|---|---|---|
| Aegis$_{\text{DY}}$ *Aux* | *10* | *15* | *21* | *28* | *36* |
| 10 | 6.41 | ***1.71*** | ***0.15*** | ***0.05*** | 0.07 |
| 15 | — | 54.07 | ***4.75*** | ***1.60*** | 2.22 |
| 21 | — | — | 38.85 | 13.07 | ***18.17*** |
| 28 | — | — | — | 14.69 | ***20.41*** |
| 36 | — | — | — | — | 21.65 |

* Configurations within 10% decoding latency are shown in bold and italics

*2) ISO Performance Comparison:* Due to the marked superiority of the dynamic strategy, Aegis$_{\text{DY}}$ can achieve improved effectiveness over its static counterpart while maintaining performance (latency) with fewer auxiliary bits. In Table III, we show the improvement of Aegis$_{\text{DY}}$ over Aegis at the initial fault rate of $10^{-3}$ with equivalent or reduced auxiliary bits. We mark configurations within 10% of decoding latency overhead (see Table I) in bold and italics. For example, the error rate of Aegis$_{\text{DY}}$ with 15 auxiliary bits is as $1.60\times$ lower as the rate of Aegis with 28 auxiliary bits at equivalent decoding latency. FNW$_{\text{DY}}$ has a similar advantage over FNW, but the detailed comparison is not shown in the paper due to page limitations.

### D. Sensitivity Study for Lower Fault Rates

In this study, we expand the range of the fault rate of the memory model to lower initial fault rates of $10^{-5}$ and $10^{-6}$, shown in Figure 10 and 11, respectively. At both fault rates, FNW fails to improve of ECC-1$_{64}$, only equivalent to it for FNW$_{\text{DY}}$ with 28 and 36 auxiliary bits, and even worse comparing to ECP$_N$ as expectation. Aegis and Aegis$_{\text{DY}}$ continue to be dramatically more effective than the other schemes with at least two and three exponential orders improvement in uncorrectable error rates than ECP$_N$, respectively. At the fault rate $10^{-5}$, Aegis$_{\text{DY}}$ significantly outperforms static Aegis using 10 auxiliary bits with a $15\times$ improvement. When there are at least 15 auxiliary bits, Aegis$_{\text{DY}}$ achieves perfect protection, while Aegis achieves this goal using at least 28 auxiliary bits. At

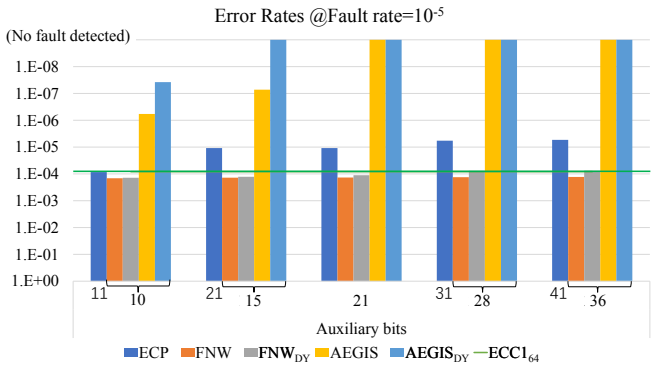


Fig. 10: Comparisons of different recovery schemes at fault rate $10^{-5}$ using ISO auxiliary bits.

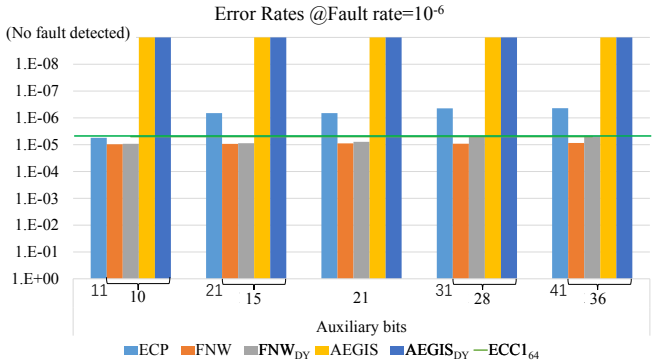


Fig. 11: Comparisons of different recovery schemes at fault rate $10^{-6}$ using ISO auxiliary bits.

the fault rate $10^{-6}$, Aegis and Aegis$_{\text{DY}}$ protect all possible accesses, while the other schemes fail to achieve this goal.

## V. Conclusion

Endurance limitations is a significant challenge for mass commercialization of several emerging non-volatile memories, including PCM and RRAM. This is especially problematic when more cells become potentially faulty due to technology scaling and resulting process variation. We presented our proposed dynamic partitioning approach to mitigate stuck-at faults in these emerging memories. Dynamic partitioning recovers more stuck-at faults in a data block by increasing the number of possible partitions, thus improving over static partition-and-flip schemes. Our results, including benchmark and probabilistic evaluations, show that dynamic partitioning significantly improves the effectiveness of FNW and Aegis over their static counterparts by generating more efficient configurations. Aegis has the shortcoming that when the first bit of a group and its corresponding flag bit are stuck at opposite values, this scheme cannot mitigate the faults for any slope. This problem might be solved by carefully rearranging the lookup table (ROM). Dynamic Aegis can also further improve its effectiveness by intelligently selecting slopes. For example, for the 6x6 grid shown in Figure 3(b), slope 3 overlaps significantly with slope 0, but it could be replaced with slope 5, which has much fewer overlaps with the preceding slopes.

As there are some relatively infrequent cases where the static scheme can correct faults uncorrectable in our dynamic approach, it is possible to incorporate a static scheme into our dynamic scheme by using one extra bit as a record in each data block. Our evaluation shows that the extra tolerance provided by the original static scheme is negligible compared to the uncorrectable error rate. However, this could be explored further in future work.

## VI. Acknowledgements

This work is supported by National Natural Science Foundation of China Grant No. 61332003 and US National Science Foundation Graduate Research Fellowship Grant No. 1247842.

## References

[1] K. Kim, "Technology for sub-50nm DRAM and NAND flash manufacturing," *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*, pp. 323–326, IEEE, 2005.

[2] S. Li, P. Wang, N. Xiao, G. Sun, and F. Liu, "SPMS: Strand based persistent memory system," *DATE*, pp. 622–625, IEEE, 2017.

[3] O. Zilberberg, S. Weiss, and S. Toledo, "Phase-change memory: An architectural perspective," *ACM Computing Surveys (CSUR)*, Vol. 45, No. 3, No. 3, p. 29, 2013.

[4] C. J. Xue, G. Sun, Y. Zhang, J. J. Yang, Y. Chen, and H. Li, "Emerging non-volatile memories: opportunities and challenges," *CODES+ISSS*, pp. 325–334, IEEE, 2011.

[5] H.-S. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, Vol. 98, No. 12, No. 12, pp. 2201–2227, 2010.

[6] H. Zhang, N. Xiao, F. Liu, and Z. Chen, "Leader: Accelerating ReRAM-based main memory by leveraging access latency discrepancy in crossbar arrays," *DATE*, pp. 756–761, 2016.

[7] S. Cho and H. Lee, "Flip-N-Write: A simple deterministic technique to improve PRAM write performance, energy and endurance," *MICRO*, pp. 347–357, IEEE, 2009.

[8] N. H. Seong, D. H. Woo, V. Srinivasan, J. A. Rivers, and H.-H. S. Lee, "SAFER: Stuck-at-fault error recovery for memories," *MICRO*, pp. 115–124, 2010.

[9] J. Fan, S. Jiang, J. Shu, Y. Zhang, and W. Zhen, "Aegis: Partitioning data block for efficient recovery of stuck-at-faults in phase change memory," *MICRO*, pp. 433–444, 2013.

[10] R. W. Hamming, "Error detecting and error correcting codes," *Bell Labs Technical Journal*, Vol. 29, No. 2, No. 2, pp. 147–160, 1950.

[11] A. N. Jacobvitz, R. Calderbank, and D. J. Sorin, "Coset coding to extend the lifetime of memory," *HPCA*, pp. 222–233, IEEE, 2013.

[12] S. M. Seyedzadeh, R. Maddah, D. Kline, A. K. Jones, and R. Melhem, "Improving bit flip reduction for biased and random data," *IEEE Transactions on Computers*, Vol. 65, pp. 3345–3356, 2016.

[13] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, "Use ECP, not ECC, for hard failures in resistive memories," *ACM SIGARCH Computer Architecture News*, Vol. 38, pp. 141–152, ACM, 2010.

[14] P. J. Nair, D.-H. Kim, and M. K. Qureshi, "ArchShield: Architectural framework for assisting DRAM scaling by tolerating high error rates," *ACM SIGARCH Computer Architecture News*, Vol. 41, pp. 72–83, ACM, 2013.

[15] R. Melhem, R. Maddah, and S. Cho, "RDIS: A Recursively Defined Invertible Set Scheme to Tolerate Multiple Stuck-at Faults in Resistive Memory," *DSN*, pp. 1–12, 2012.

[16] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," *Acm sigplan notices*, Vol. 40, pp. 190–200, ACM, 2005.

[17] Z. Al-Ars, *DRAM fault analysis and test generation*. TU Delft, Delft University of Technology, 2005.

[18] T. Yuan, S. Z. Ramadan, and S. J. Bae, "Yield prediction for integrated circuits manufacturing through hierarchical Bayesian modeling of spatial defects," *Transactions on Reliability 2011*.

[19] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, *et al.*, "FreePDK: An open-source variation-aware design kit," *MSE*, pp. 173–174, 2007.