Utilization of SECDED for Soft Error and Variation-Induced Defect Tolerance in Caches

Luong D. Hung, Hidetsugu Irie, Masahiro Goshima, Shuichi Sakai
Graduate School of Information Science and Technology, The University of Tokyo
Hongo 7-3-1, Bunkyo, Tokyo 113-8656, Japan
{hung,ern,goshima,sakai}@mtl.t.u-tokyo.ac.jp

Abstract

Combination of SECDED with a redundancy technique can effectively tolerate a high variation-induced defect rate in future processes. However, while a defective cell in a block can be repaired by SECDED, the block becomes vulnerable to soft errors. This paper proposes a technique to deal with the degraded resilience against soft errors. Only clean data can be stored in defective blocks of a cache. This constraint is enforced through selective write-through mechanism. An error occurring in a defective block can be detected and the correct data can be obtained from the lower level caches.

1 Introduction

SRAM designs are confronted with two serious problems: soft errors and variation-induced defects. Radiation-induced soft errors cause bit flips in memory caches [4]. Soft error rate (SER) per bit of SRAM is expected to stay steady over process generations [3][1]. However, device scaling allows the number of bits that can be integrated on a chip to increase exponentially, rising the total SER rapidly. Tolerance against soft errors is highly required. To provide high reliability, Error Correcting Code (ECC)–particularly, Single Error Correction Double Error Detection Hamming code (SECDED)– is widely employed to detect and correct soft errors in SRAMs. Moreover, interleaving of multiple ECC codewords can effectively tolerate spatial multi-bit errors [9].

Process variation causes spreads in the electrical characteristics of devices. Particularly, fluctuations in the number and locations of dopant atoms in the channel region of a transistor can result in a large within-die threshold voltage (V_{th}) variation in scaled processes [8][13]. Variation effect is pronounced in SRAMs where minimum-geometry transistors are used. Large variation can lead to a cell failure. Mechanisms of various types of cell failures (e.g., read, write, and access failures) and dependency of failure probabilities to V_{th} variation have been investigated [2][6]. The results indicate that the defect rate of a SRAM cell can reach 10^{-3} or even higher in future processes.

Redundancy techniques have been widely used to mitigate manufacturing defects. Spare elements are included in the design and are used to replace defective elements. Coarse-grain redundancy (e.g., spare rows/columns) cannot tolerate high defect densities. Fine-grain redundancy (e.g., spare words, bytes) [10][5] leads to efficient usage of redundancy resources but significantly increases the complexity of repair circuitry. For instance, assuming the defect rate of 10^{-3}, a 512KB SRAM will have roughly 5K defective words. The Built-In Self Repair (BISR) and decoder circuitry supporting the replacement of such a high number of defective words is complex. Particularly, the Content-Addressable Memory, which is usually used by BISR to store the addresses of the defective words, becomes excessively large and degrades the access latency. Stand-alone redundancy techniques therefore are not scalable defect-tolerance solutions.

Combination of a redundancy technique with ECC can tolerate high defect rates [11][12]. A hardware block containing a single defective can be repaired by ECC. At much lower probability, a block may contain multiple defective cells exceeding the error detection/correction capability of ECC. Only such a block is replaced by a redundancy element. With such a combined approach, a small number of redundancy elements can be sufficient.

It is cost-effective if the same ECC resource can be used for both defect and soft error tolerance. However, while a defective cell in a block can be tolerated by SECDED, the block becomes vulnerable to soft errors. A soft errors occurring in the block could be left uncorrectable. While the error tolerability can be enhanced by using ECC codes that are more powerful than SECDED, these codes incur significant area and latency overheads. Table 1 shows

978-3-9810801-2-4/DATE07 © 2007 EDAA

Table 1. Number of check bits for SECDED and BCH DEC

Information-bit length	32	64	128
SECDED	7	8	9
BCH DEC	12	14	16

the number of check bits required to implement SECDED and Bose-Chaudhury-Hocquenghem Double-Error Correction (BCH DEC) at various information bit lengths. DEC almost doubles the number of check bits as compared with SECDED. Moreover, while SECDED's encoding/decoding circuitry can be constructed as XOR trees and is quite fast, the powerful codes including BCH DEC are typically cyclic codes employing multi-bit Linear Feedback Shift Registers (LFSR) which introduce inordinate delay and are unsuitable for being implemented in SRAMs requiring fast access. Previous ECC/redundancy combined work assumes SECDED and therefore improves defect tolerance at the expense of degraded soft error tolerance [11][12].

In this paper, we propose a technique to tolerate variation-induced defects while preserving high resilience against soft errors. Only clean data are allowed to be stored in the defective (but still usable) blocks of a cache. This constraint is enforced through a selective write-through mechanism called *assurance update*. Soft errors cannot cause integrity problem in these blocks because SECDED is still able to detect the errors and the correct data can always be obtained from the lower level caches. The number of assurance updates can be effectively reduced by maintaining data dirtiness at SECDED block level and performing data swapping between blocks.

The rest of this paper is organized as follows. Section 2 describes the assumed cache structure and classification of blocks based on the degree of defectiveness. Section 3 explains the mechanism of assurance update. Section 4 shows an exemplified cache structure. Section 5 presents evaluation. Finally, Section 6 concludes the paper.

2 Cache Structure and Block Classification

We assume that a cache consists of several subarrays. A subarray has several rows. Each row is comprised of several SECDED blocks. Each subarray has its own decoder, Built-In Self Test (BIST), BISR, and some spare rows. BIST detects the defective cells in the subarray.

SECDED blocks are classified based on the number of defective cells they contain. A block that does not have any defective cell is called a *good* block (g-block). A block having a single defective cell is called a *tolerable* block (t-block). A block having more than one defective cells is called a *bad* block (b-block). The row containing at least one b-block is a bad row. BISR replaces the bad rows with redundancy rows. Each block is associated with a *g-bit.* The g-bit of a block is set (or reset) if the block is a g-block (or t-block). Defect analyzing and setting of g-bits are performed by BIST.

We assume that multiple blocks in the same row are interleaved to disperse the error bits of a spatial multi-bit error; a block contains as much as one error bit. We therefore consider how to deal with a single-bit error in a block. If the block is a g-block, SECDED can be exclusively used for soft error tolerance. Therefore, the error is detectable and correctable. However, if the block is a t-block, a part of SECDED capability is used for repairing the defective cell, leaving the reduced capability for soft error tolerance. The error in this case is detectable, but *uncorrectable*. If the corrupted data are clean, error detection is sufficient since the correct data can be obtained from the lower level caches. Data integrity problem occurs if the corrupted data are dirty data which have no backup elsewhere. Writeback caches are therefore susceptible to such a problem.

Given the defect rate λ, the probabilities that a block of BS bits is a g-block, t-block, and b-block are respectively P_{g-blk}, P_{t-blk}, and P_{b-blk}, and are given by

$$P_{g-blk} = (1-\lambda)^{BS} \quad (1)$$

$$P_{t-blk} = BS(1-\lambda)^{BS-1}\lambda \quad (2)$$

$$P_{b-blk} = 1 - P_{g-blk} - P_{t-blk} \quad (3)$$

Figure 1 shows the distribution of three types of blocks at different defect rates. The block size is 72 bits (64 information bits and 8 check bits). With high defect rate, after the b-blocks are replaced by redundancy elements, there is a significant proportion of t-blocks that is vulnerable to soft errors. How to overcome such a vulnerability is described in the next section.

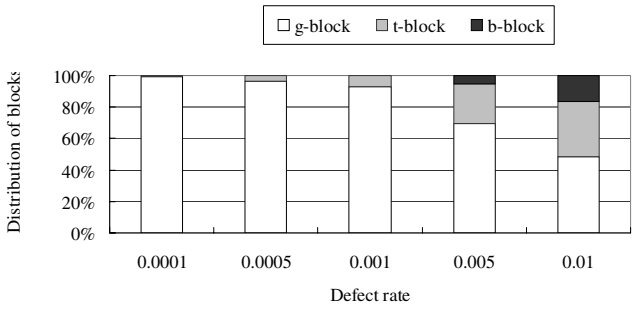

Figure 1. Distribution of blocks at different defect rates

3 Assurance Update

The data integrity problem caused by soft errors can be prevented by allowing only clean data to be stored in t-blocks. Whenever a t-block of a cache is updated, the updated data are also sent to the lower level cache. Such an update is referred to as an *assurance* update. An assurance update is also necessary if the update goes to a cache line which the block holding the tag of the cache line is a t-block. It is because if the tag is corrupted by a soft error, we cannot reclaim the correct address to which the cache line originally belonged.

Assurance updates increase the number of accesses to the lower level cache. Conventionally, there is a writeback buffer sitting between a cache and its lower level cache. Data updated to the lower level cache are temporarily stored in the writeback buffer and will be written back later when the bus is free. Thanks to writeback buffer, an assurance update can be mostly removed from the critical path of a cache access. However, if the writeback buffer is full, the access is stalled for the writeback buffer to retire its entries to make room for the coming data. Assurance updates also consume power. We next describe methods to efficiently perform assurance updates to limit their impacts on performance and power consumption.

Maintaining dirtiness at block level

In conventional caches, the dirtiness of data is maintained at cache line level; a cache line usually consists of multiple SECDED blocks. When a dirty cache line is written back, all the constituent blocks are written back no matter whether the blocks have been modified or not. By maintaining the dirtiness of data at the block level, unnecessary writebacks of clean blocks can be eliminated. Each block is associated with a *d-bit*. The d-bit of a block is set (or reset) if the block stores dirty (or clean) data. When a cache line is written back, only those blocks having their d-bits set are sent to the lower level cache. The d-bits are then reset to indicate that the data in the blocks are now clean. By reducing the number of blocks updated to the lower level cache, the probability of assurance update triggered by the lower level cache can also be reduced.

Data swapping between blocks

The clean data stored in a g-block are "over-protected" by SECDED since single error detection capability is sufficient in this case. On the other hand, the dirty data stored in a t-block are "under-protected" since an error occurring in the block drives the data into unrecoverable state. For other two combinations–clean data stored in t-block or dirty data stored in the g-block–the data are just sufficiently protected. The under-protected and over-protected data can be swapped to improve the protectiveness of the data as a whole. An assurance update can be avoided if the number of dirty blocks is no more than the number of t-blocks in the cache line. Figure 2 shows an example of data swapping. In Figure 2-a, since dirty data are stored to a t-block (the third block), an assurance update is required. By swapping data between the third and fourth blocks, the assurance update can be avoided, as shown in Figure 2-b.

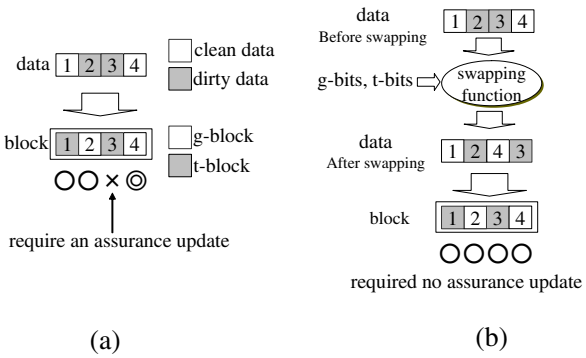

Figure 2. Example of data swapping

Overheads in terms of power and latency incur when clean data are read from g-blocks and stored to t-blocks in data swapping. However, we believe that performing a data swapping in a cache consumes less power than making an update request to the lower level cache which is typically many times bigger and slower. This is particularly true if the lower level cache is an off-chip memory.

Swapping is executed based on the information of t-bits and d-bits. There could be several ways to implement the swapping function. Instead of focusing on the detail implementation of swapping function, this paper concentrates on investigating the potential of data swapping in reducing the number of assurance updates.

4 Example Cache Structure

Figure 3 shows a structure of a 512KB, four-way set-associative cache. The cache line size is 64B. The cache is constructed from a tag subarray and eight 64KB data subarrays. Each data subarray has 1024 rows, each row is comprised of eight 72b blocks. The tag subarray has 512 rows, each row is comprised of sixteen 44b blocks. A tag entry has a tag, some state bits (for LRU replacement and cache coherency), and g-bits and d-bits of all blocks belonging to the corresponding cache line. Tag access usually proceeds the data access in a typical cache. By storing the g-bits and d-bits of the blocks of the data array in the corresponding entry of the tag array, the decisions of 1) whether an assurance update is needed or not upon a cache write, and 2) which blocks are dirty that are needed to be written back upon a line eviction or an assurance update, can be determined at the end of the tag access. Accesses to clean blocks

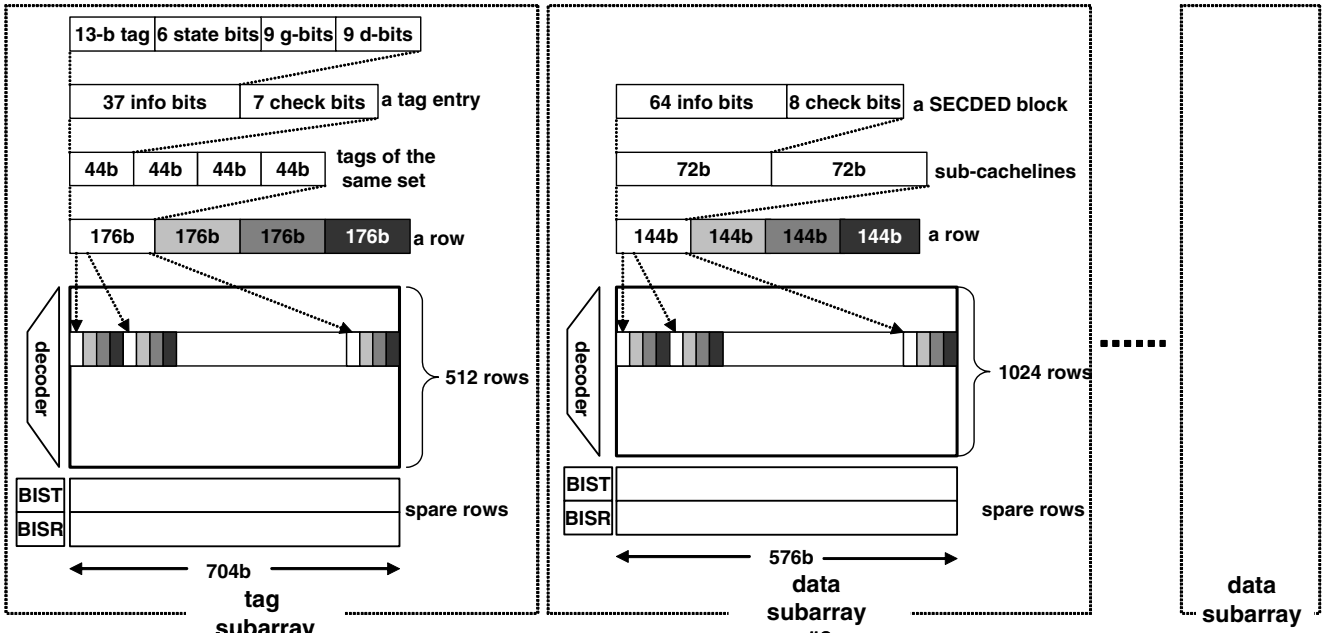

Figure 3. Structure of a 512KB cache

in a cache line can be skipped, thereby saving power consumption.

5 Evaluation

5.1 Evaluation Methodology

The simulated system is an out-of-order superscalar processor. Table 2 lists the configuration parameters. The processor has 16KB instruction and data caches, and a 512KB unified L2 cache. The data cache and L2 cache are write-back caches, each equipped with a four-entry writeback buffer.

SECDED is maintained for every 64 information bits in the L1 caches and L2 cache. We assume that all bad rows in the caches are replaced by redundancy rows. The probability that a block is a t-block or g-block is derived from Equation 1 and 2. The g-bits are randomly initialized at the beginning of the simulation based on such a probability. We consider defect tolerance only in SRAM caches and assume the memory to be fault-free.

The following target systems are evaluated:

- **Baseline**: The data cache and L2 cache do not perform assurance update
- **AsLine**: The caches perform assurance updates and maintain dirtiness at cache line level.
- **AsBlk**: The caches perform assurance updates and maintain dirtiness at block level.
- **AsBlkSwap**: The caches perform assurance updates at block level and data swapping.

Simulation is performed using SimpleScalar [7]. SPEC2000 benchmarks are used in the simulation. For each benchmark, we skip the first one billion instructions and simulate the next four billion instructions.

We assume that the soft error rate of an unprotected SRAM equal to 1.6 KFIT[1] per megabit [3], and soft errors follow an uniform distribution. Timestamps of accesses to the cache lines and blocks of the caches are recorded. Such information allows us to calculate the error rates.

5.2 Evaluation Results

Figure 4-a shows the normalized performance when the defect rate λ is equal to 0.005. **AsLine** degrades performance significantly for some applications (e.g., *lucas*, *gcc*). However, the performance degradation is reduced significantly for **AsLine** and becomes extremely small for **AsBlkSwap**. The same trend is also observed in Figure 4-b which shows the performance degradation averaged for all benchmarks as λ is varied.

Figure 5-a shows number of accesses from L2 cache to memory per one thousand instructions with λ equal to 0.005. The breakdowns of the accesses are also shown in the figure. The number of accesses to memory for **AsLine** increases considerably due to assurance updates for most

[1] One FIT (Failure In Time) corresponds to one failure per 10^9 hours

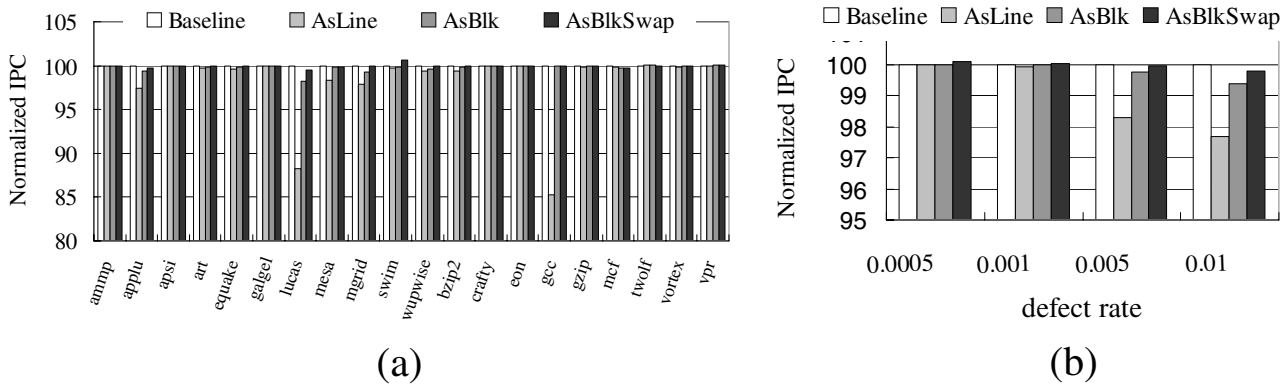

Figure 4. Normalized performance at λ=0.005 in (a) and at different λ in (b)

Table 2. Parameters of Simulated Architecture

Processor Parameters	
Frequency	1 GHz
Functional Units	4 integer ALUs, 4 FP ALUs
	1 integer multiplier/divider
	1 FP multiplier/divider
LSQ size	32 instructions
RUU size	64 instructions
Issue Width	4 instructions/cycle
Memory Hierarchy Parameters	
L1 i-cache	16KB, direct-map, 32B line, 1 cycle latency
	72b SECDED block
L1 d-cache	16KB, 4-way, 32B line, 1 cycle latency, writeback
	72b SECDED block, 4-entry writeback buffer
L2 cache	512KB, unified, 4-way, 64B line, 6 cycle latency
	72b SECDED block, 4-entry writeback buffer
Memory	100 cycle latency

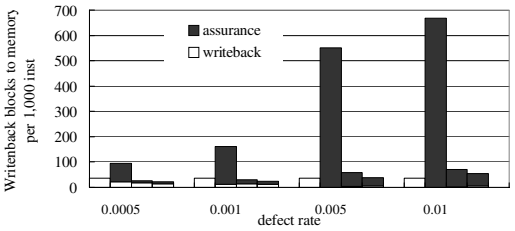

Figure 6. Number of written-back blocks to memory per 1,000 instructions

Table 3. Uncorrectable error rate of L2 caches

Cache	Uncorrectable Error Rate (FIT)
Baseline	631
Assurance update	3.84e-14

benchmarks. Since assurance updates can have the effect of early writing back the dirty cache lines, some normal writebacks (upon cache replacements) are converted to assurance updates which can be observed through the reduction in the number of writebacks for **AsLine** as compared with **baseline**. The number of assurance updates and writebacks are reduced greatly for **AsBlk**. Data swapping allows further significant reduction in the number of assurance updates. The same trend can also be observed when λ is varied, as shown in Figure 5-b. The reduction in the number of assurance updates reduces the stalled cycles caused by a full writeback buffer and results in very small performance degradation that we noticed earlier in Figure 4.

Figure 6 shows the number of written back blocks to memory per one thousand instruction averaged for all benchmarks as λ is varied. The figure clearly confirms the effectiveness of maintaining data dirtiness per block for eliminating unnecessary block updates and improving power efficiency.

Table 3 shows the uncorrectable error rate of the baseline L2 cache and the L2 cache performing assurance update. For the baseline cache, an uncorrectable error occurs if a strike occurs on a t-block of a dirty cache line and the cache line is read later. For the cache performing assurance update, an uncorrectable error occurs if two strikes occur on the same t-block between two consecutive accesses to the block. The error rate of the cache performing assurance update is many orders of magnitude lower than that of the baseline cache.

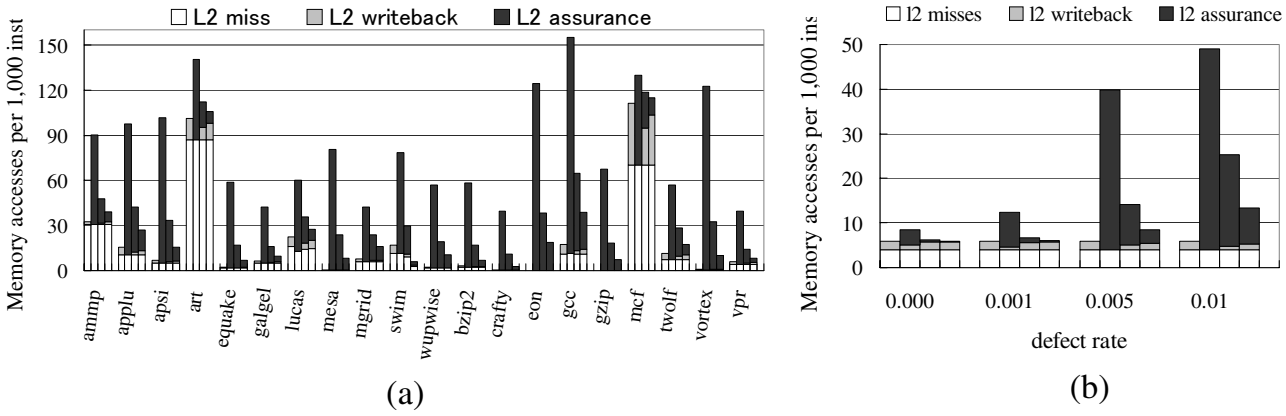

Figure 5. Breakdown of accesses from L2 cache to memory per 1,000 instructions at λ=0.005 in (a) and as λ is varied in (b)

6 Conclusions

Tolerating soft errors for high reliability and tolerating variation-induced defects for yield improvement are highly required in advanced SRAM designs. The proposed technique can satisfy both the requirements. Combination of SECDED with a redundancy technique can tolerate a high defect rate. Enforcement of assurance update mechanism aided by the newly added g-bits and d-bits efficiently eliminates the uncorrectable soft errors occurring in defective (but still usable) blocks. The overheads in terms of performance and power consumption incurred by assurance updates are fairly low even when the defect rate is as high as 10^{-2}.

Acknowledgement

This research is partially supported by Grant-in-Aid for Fundamental Scientific Research B(2) #13480077, B(2) #16300013 from the Ministry of Education, Culture, Sports, Science and Technology Japan, a CREST project of the Japan Science and Technology Corporation, and by a 21st century COE project of the Japan Society for the Promotion of Science.

References

[1] International Technology Roadmap for Semiconductor. http://public.itrs.net, 2005.

[2] A. Agarwal, B. C. Paul, S. Mukhopadhyay, and K. Roy. Process Variation in Embedded Memories: Failure Analysis and Variation Aware Architecture. *IEEE Journal on Solid State Circuits*, 40(9):1804–1814, 2005.

[3] R. Baumann. Soft Errors in Advanced Computer System. *IEEE Design and Test of Computers*, 22(3):258–266, 2005.

[4] R. C. Baumann. Soft Errors in Advanced Semiconductor Devices–Part I: Three Radiation Sources. *IEEE Transactions on Device and Materials Reliability*, 1(1):17–22, 2001.

[5] A. Benso, S. Chiusano, G. D. Natale, P. Prinetto, and M. L. Bodoni. A family of self-repair SRAM cores. In *Proc. IEEE International On-Line Testing Workshop*, pages 214–218, 2000.

[6] A. Bhavnagarwala, S. Kosonocky, C. Radens, K. Stawiasz, R. Mann, Q. Ye, and K. Chin. Fluctuation Limits & Scaling Oppoturnities for CMOS SRAM Cells. In *Proc. IEEE International Electron Devices Meeting*, pages 659–662, 2005.

[7] D. Burger and T. Austin. The SimpleScalar Tool Set. Technical Report CS-TR-1997-1342, University of Wisconsin-Madison, 1997.

[8] D. J. Frank, Y. Taur, M. Ieong, and Hon. Monte Carlo Modelling of Threshold Variation due to Dopant Fluctuations. In *Proc. IEEE International Electron Devices Meeting*, pages 93–94, 1999.

[9] K. Osada, Y. Saitoh, and K. Ishibashi. 16.7-fA/Cell Tunnel-Leakage-Suppressed 16-Mb SRAM for Handling Cosmic-Ray-Induced Multierrors. *IEEE Journal on Solid State Circuits*, 38(11):1952–1957, 2003.

[10] V. Schober, S. Paul, and O. Picot. Memory Built-in Self-Repair using Redundant words. In *Proc. IEEE International Test Conference*, pages 995–1001, 2001.

[11] C. H. Stapper and H.-S. Lee. Synergistic Fault-Tolerance for Memory Chips. *IEEE Transactions on Computers*, 41(9):1078–1088, 1992.

[12] C.-L. Su and Y.-T. Yeh. An Integrated ECC and Redundancy Repair Scheme for Memory Reliability Enhancement. In *Proc. IEEE International Test Conference*, pages 81–89, 2005.

[13] X. Tang, V. K. De, and J. D. Meindl. Intrinsic MOSFET Parameter Fluctuations Due to Random Dopant Placement. In *Proc. IEEE International Electron Devices Meeting*, pages 369–376, 1997.