On-Line Failure Detection and Confinement in Caches

Jaume Abella, Pedro Chaparro, Xavier Vera, Javier Carretero, Antonio González
Intel Barcelona Research Center
Intel Labs - Universitat Politècnica de Catalunya
{jaumex.abella, pedro.chaparro.monferrer, xavier.vera,
javierx.carretero.casado, antonio.gonzalez}@intel.com

Abstract

Technology scaling leads to burn-in phase out and increasing post-silicon test complexity, which increases in-the-field error rate due to both latent defects and actual errors. As a consequence, there is an increasing need for continuous on-line testing techniques to cope with hard errors in the field. Similarly, those techniques are needed for detecting soft errors in logic, whose error rate is expected to raise in future technologies.

Cache memories, which occupy most of the area of the chip, are typically protected with parity or ECC, but most of the wires as well as some combinational blocks remain unprotected against both soft and hard errors.

This paper presents a set of techniques to detect and confine hard and soft errors in cache memories in combination with parity/ECC at very low cost. By means of hard signatures in data rows and error tracking, faults can be detected, classified properly and confined for hardware reconfiguration.

1 Introduction

Burn-in becomes less effective due to the high degradation produced in otherwise well-functioning devices and its cost increases with technology scaling. Thermal runaway due to high leakage and longer test patterns due to the increased gate count diminish the effectiveness of burn-in [14]. Such effectiveness loss increases the likelihood of in-the-field failures due to rather small defects (latent defects) and errors in hard-to-test corner cases. Small latent defects become large enough during operation due to degradation and cause failures before the target lifetime [4, 12].

Such failures are likely to appear in highly integrated memories [3], and the failure rate is expected to worsen in future technologies. Thus, techniques to detect and correct failures in the field become a must. Moreover, soft error rates in logic are expected to increase in future technologies [11], which also calls for in-the-field testing mechanisms.

While off-line testing methods such as data patterns read/write can be adapted for on-line testing to some extent with some hardware support [19], their coverage is poor because many errors only arise at particular environmental situations (temperature, supply voltage, voltage droops, etc) and cannot be reproduced for periodic tests [1, 8]. Furthermore, soft errors in logic can be hardly detected by means of periodic tests. Thus, continuous on-line testing mechanisms will become mandatory.

Caches, which occupy most of the area of the chip, currently have some testing capabilities like ECC or parity that are especially suitable to detect soft and hard errors in the silicon of memory cells [6], both during off-line testing and in-the-field operation. However, errors in combinational logic (decoders, multiplexers) and wires (bridges or opens in wordlines and routing wires) are very likely to remain undetected. Thus, low-cost techniques to detect soft and hard errors during in-the-field operation are required to protect combinational parts as well as wires of the arrays of cache memories. Moreover, such techniques must distinguish soft and hard errors so we can confine hard errors to the smallest piece of hardware including the faulty circuit for hardware reconfiguration.

This paper proposes an on-line mechanism to detect and confine errors in all components of caches to achieve full coverage. Our mechanism complements parity/ECC by detecting errors in wordlines and logic. The key parts of our technique are as follows:

(i) Memory rows are extended with hardwired signatures to detect any row selection error.

(ii) Column decoders are replicated and signals between cache arrays are parity protected to detect errors in the remaining combinational blocks.

978-0-7695-3264-6/08 $25.00 © 2008 IEEE
DOI 10.1109/IOLTS.2008.15

(iii) Few small tables (around 100-bytes in total) are set up to track errors in such a way that frequent failures can be confined and repaired.

The rest of the paper is organized as follows. Section 2 describes our mechanism for failure detection and confinement. Section 3 evaluates the overhead for different caches. Section 4 reviews some related work. Finally, Section 5 draws the main conclusions of this work.

2 On-Line Failure Detection and Confinement in Cache Memories

This section presents our mechanism to detect errors in combinational blocks of caches. Our proposal allows detecting both hard and soft errors. Moreover, it classifies them and isolates defects for hardware reconfiguration. First, we introduce the targeted types of errors. Then, we describe our techniques for error detection in the different components of caches and isolation.

2.1 Targeted Errors

Some on-line testing techniques such as parity and ECC are in place for soft error detection, but they can also detect some defects [6]. Such techniques are limited to bit errors in memory cells as well as some faults in metal layers with limited cost in terms of power, area and delay. Unfortunately, many faults in combinational logic and metal layers of caches are left undetected. In particular, errors detected by parity/ECC correspond to soft errors and silicon defects in memory cells, defects affecting metal 1 in memory cells (in-cell connections), bitline defects, power/ground defects and sense amplifier defects. Any of those errors and defects is expected to show up in one or few bits of data, and codes such as parity and ECC are very likely to detect them.

The type of errors that may appear in a cache and are not covered by parity/ECC are as follows:

(A) **Soft errors, and silicon and metal defects in combinational logic.** Some of such faults may turn into a bit flip (e.g., sense amplifier fault), and hence, can be detected with parity/ECC, but some others may cause wrong data to be read/written (e.g., decoder fault) or a false hit/miss to be reported (e.g., address comparator fault). The case of reading/writing wrong data is very unlikely to be detected with parity/ECC. For instance, if a given byte and its parity bit are written in a wrong location the fault will not be detected because data is consistent from the parity point of view.

(B) **Wordline defects (metal 2).** Opens and bridges in wordlines may make either read/write wrong data or access a wrong location [9]. In the latter case, since such data is stored jointly with their parity/ECC bits, the fault will not be detected with parity/ECC logic.

(C) **Data and address routing.** Such wires may experience opens or bridges [9]. If parity/ECC is not available for data and address in those wires, faults will not be detected.

2.2 Error Detection

For the sake of illustration, we use the example in Figure 1 in the explanation of our techniques. Figure 1 shows the schematic for the data arrays of a 32KB cache arranged as 4 arrays of 256 rows and 288 columns (256 for data and 32 for parity bits). Grey boxes and straight lines stand for unmodified blocks and routing wires. Conversely, black boxes and dotted lines stand for new blocks and modified routing wires respectively.

The set of solutions that we propose for the different types of errors are as follows:

Type (A). Soft errors and silicon defects not protected by parity/ECC correspond basically to decoder errors (both row and column decoders as well as predecoders) and address comparator faults. Such blocks can be easily replicated and their outputs compared. The cost is very low because the original blocks are rather small and replicas are even smaller because they are not connected to large transistors feeding long wires (bitlines, wordlines and routing wires). Since our mechanism to protect wordlines also detects faults in the row decoder (see next bullet), such decoder does not have to be replicated in any of the arrays. Thus, only column decoders must be replicated in the data side as shown in Figure 1. Although not shown in the figure, note that an extra bit must be generated indicating whether both column decoders in the accessed array provided the same result.

Type (B). Wordline defects are detected adding a signature to each wordline. Such signature is a set of ROM cells that are read every time that a given wordline is activated. This way we can check whether the proper wordline was activated by comparing the signature and those address bits identifying the row to be accessed. There are two considerations to set up such signatures:

- Any wordline can be activated at any time due to a bridge with any other wire (e.g., power supply) so unique signatures are required for wordlines inside an array. For any array with 2^K rows, K bits are required per row to encode the row address (e.g., signature 0 for the first row, signature 1 for the second

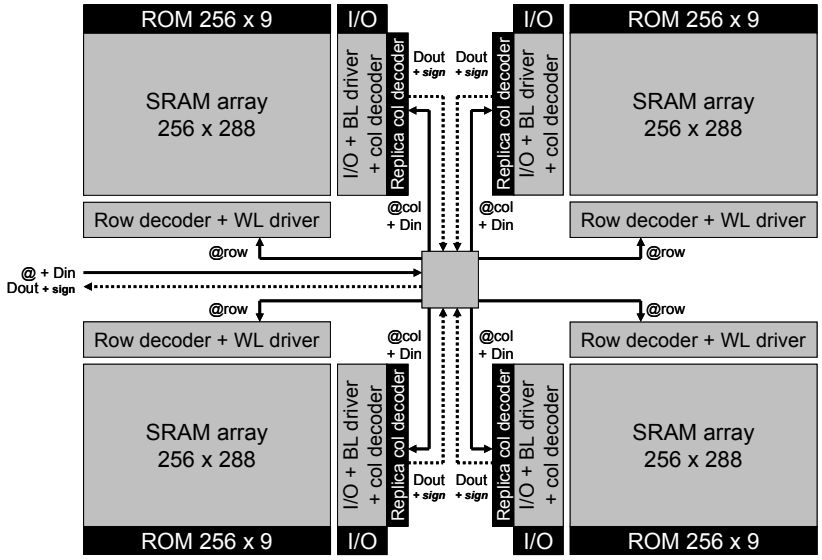

Figure 1. Diagram of the data side of a 32KB cache with parity protection extended with features for on-line error detection

row, and so on and so forth). The number of bits can be lower to reduce the overhead at the expense of some aliasing between different rows, which decreases the fault detection coverage.

- Given that any wordline may always be low due to an open or a short with any other wire (e.g., ground), all bitlines may be interpreted in a given direction depending on the bias of latches and sense amplifiers, and hence, bits may become all ones or all zeros (e.g., in case of an open all bits are ”0). Thus, an extra bit set to the opposite value of such bias is required for each wordline to identify this particular situation (e.g., the bit is set to ”1). If there is an open, such bit will have the wrong value (e.g., ”0) and the fault will be detected.

As we can see in the figure, SRAM arrays are extended with some ROM cells and the corresponding logic to read such cells. Note that row decoders do not have to be replicated because signatures read from ROM cells protect both wordlines and row decoders. The signature is compared against the row address (8 out of the 9 signature bits) and the expected value for the fixed bit. Such comparison is not shown in the figure, but a small 9-bit comparator should be included for each SRAM array as well as a 1-bit signal reporting whether the signature and the address matched.

Type (C). Data and address routing from/to the tag and data arrays need parity protection. Such protection should be in place for any cache protected with parity (L1 caches) or ECC (L2/L3 caches), but if it is not the case, we propose to include it. Both data and address routing wires are parity protected in the example so no extra protection is required.

2.3 Error Confinement

Once an error is detected, it is crucial to identify whether it was caused by a soft error or a defect. In case of having a defect, we would like to disable the minimum amount of hardware (e.g., a single cache line) in such a way that the processor keeps operating and performance impact is minimal. For instance, if we detect a defect in a wordline, only those cache lines using the faulty wordline should be disabled. To do this we will include few small tables to track errors at array, array row, array column, datum, decoder, comparator and routing level. The different structures are described in Table 1.

Since keeping track of errors in all blocks would require large storage and errors are expected to happen seldom, tables for error tracking will be small (e.g., 8 entries each) with least recently used (LRU) replacement. Whenever an error is detected, it may be detected in the decoders, in the comparators, in the routing or in the arrays. The corresponding tables are updated either inserting the new information of the faulty access or incrementing the proper error counter if the entry exists. Whenever a block is faulty its counter saturates rapidly because errors show up in bursts and can be either deactivated or repaired. From time to time (e.g., every 1 billion cycles) error counters are either shifted right or

Table 1. Tables for error confinement

Error location	Fields of the table	When it is updated
Array	Port, Array, #errors (X_1+X_2+X_3 bits)	Parity/ECC of data reports an error, the signature is wrong, or the column decoder and its replica provide different outputs
Array row	Port, Array, Row, #errors (X_1+X_2 bits)	Parity/ECC of data reports an error, the signature is wrong, or the column decoder and its replica provide different outputs
Array column	Port, Array, Column, #errors (X_1+X_2 bits)	Parity/ECC of data reports an error, the signature is wrong, or the column decoder and its replica provide different outputs
Datum	Port, Array, Row, Column, #errors (X_1 bits)	Parity/ECC of data reports an error, the signature is wrong, or the column decoder and its replica provide different outputs
Decoder	Port, Array, #errors (X_1+X_2 bits)	Column decoder and its replica provide different outputs
Comparator	Port, Cache way, #errors (Y bits)	Address comparator and its replica provide different outputs
Routing	Port, Array, #errors (Z bits)	Parity/ECC of routing wires reports an error

reset to get rid of faults tracked due to soft errors. Soft errors are relatively infrequent so even if some errors are reported due to strikes, neither they will be enough to saturate any counter, nor they will always happen in the same block. Thus, soft errors will not cause the deactivation of any operating block.

Note that the sizes of the counters must meet some constraints to ensure that fine-grain errors are not considered coarse-grain errors. For instance, many errors in a given array column will saturate the counter for such column, but will not be enough to saturate the corresponding counter for the array, which needs more errors to saturate. Similarly, errors in a column will distribute across different rows and data, and hence, row and datum counters will not saturate. In fact, row and datum tables are very likely to evict entries because errors will happen at different locations from their point of view, whereas such errors will happen in the same location from the column point of view. All those constraints are met by ensuring that errors in a given datum (counters of X_1 bits) saturate neither array column and row counters (X_1+$X_2 > X_1$) nor the array counter (X_1+X_2+$X_3 > X_1$), and errors in a given column or row do not saturate the array counter (X_1+X_2+$X_3 > X_1$+X_2). Note that X_1, X_2, X_3, Y and Z are greater than zero to track errors and meet those constraints.

Once a block is considered to be faulty it will be disabled, which can be done using hardware fuses to permanently invalidate the block, or storing fault information in non-volatile memory (e.g., in the BIOS) to be read at boot time. In fact, redundant hardware not used at fabrication may be available and can be used to replace the faulty block. Nevertheless, how the faulty block is disabled or replaced is out of the scope of this paper.

3 Evaluation

This section evaluates our set of techniques for two different cache configurations resembling an L1 and an L2 cache. We describe the evaluation methodology and present results for both cache configurations as well as an example of the operation of error confinement tables.

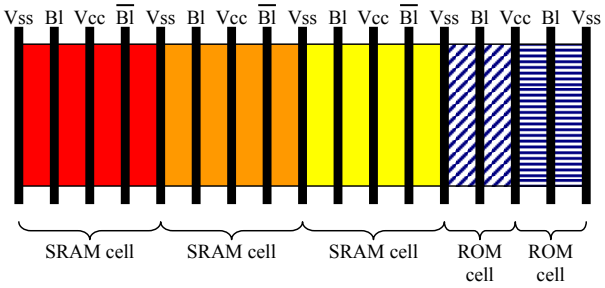

Figure 2. *Vcc*, *Vss* and bitline layout for SRAM and ROM cells

3.1 Evaluation Methodology

The evaluation has been done using CACTI 4.0 [24], which is a timing, power and area model for cache-like structures. Replicated components have been incorporated in the power and area calculation. Since CACTI does not model ROM cells needed for wordline signatures, we model them as SRAM cells. We have incorporated ROM cells based on regular SRAM cell designs [3, 13], where bitlines are interleaved with *Vcc* and *Vss* lines as shown in Figure 2. ROM cells require a single bitline given that they are connected directly to *Vcc*/*Vss*, and hence their read speed is very high (differential signaling is not required). ROM cells require much smaller area than conventional 6-T SRAM cells [3, 13]. Therefore, bitlines, *Vcc* and *Vss* wires could be laid out in a more compressed manner for ROM cells than for SRAM cells. However, to keep regularity, which is important for yield [18], we assume the same wire layout as for the rest of the array (see Figure 2). Such assumption is detrimental for our approach because increases the area devoted to the ROM cells of wordline signatures, but it is the most realistic assumption.

Table 2. Configuration of L1 tables for error confinement

Table	Field size						Total bits
	Port	Array	Row	Col	Way	#errors	(8 entries)
Array	1	3	-	-	-	8	96
Array row	1	3	8	-	-	6	144
Array column	1	3	-	2	-	6	96
Datum	1	3	8	2	-	5	152
Decoder	1	3	-	-	-	6	80
Comparator	1	-	-	-	2	4	56
Routing	1	3	-	-	-	4	64
							688 bits

Table 3. Configuration of L2 tables for error confinement

Table	Field size						Total bits
	Port	Array	Row	Col	Way	#errors	(8 entries)
Array	0	4	-	-	-	8	96
Array row	0	4	10	-	-	6	160
Array column	0	4	-	5	-	6	120
Datum	0	4	10	5	-	5	192
Decoder	0	4	-	-	-	6	80
Comparator	0	-	-	-	2	4	48
Routing	0	4	-	-	-	4	64
							760 bits

3.2 Results

We have evaluated our mechanism for two different caches implemented with 65nm technology: (i) a parity protected L1 data cache 32KB 4-way 64 bytes/line, with 1 read and 1 write ports, arranged as 4 data arrays of 256x288 bits each and 4 tag arrays of 64x204 bits each, and (ii) an ECC protected L2 cache 2MB 4-way 64 bytes/line, with 1 read/write port, arranged as 8 data arrays of 1024x2304 bits and 4 tag arrays of 512x180 bits. Error confinement table configurations are reported in Tables 2 and 3. As shown, the L2 cache does not require any bit to identify the port where errors happen because there is a single port. Column errors are tracked at 8-byte level for both caches (the largest size for a single datum). The number of bits of the L1 cache confinement tables is less than 0.3% that of the L1 cache itself. In the case of the L2 cache such ratio is 0.005%. Overall, the overhead of confinement tables is negligible.

Table 4 shows the overheads (hardware to detect and confine errors) in terms of power and area for both caches. For the sake of illustration we also report the cost of memory cells for parity/ECC (assuming an overhead of 8 bits per 64 bits of data). As shown, our mechanism raises the coverage to full coverage for hard errors at low cost. The area overhead for the 32KB L1 cache is 3.6% and corresponds mainly to the wordline signatures. Such overhead is lower for the large L2 cache because its data arrays are larger, and thus, the relative cost of ROM cells is lower. If wordline partitioning [10, 23] was used the area cost would raise, but in any reasonable configuration the area overhead would remain well below 5%. Nevertheless, the area cost can be reduced by reducing the size of the signatures at the expense of some aliasing as explained before.

Table 4. Overhead of our technique and parity/ECC in terms of area and power

Cache size	Our approach		Parity/ECC	
	Area	Power	Area	Power
32KB L1	3.6%	3.0%	11.3%	11.0%
2MB L2	1.3%	4.4%	12.4%	10.1%

It can be observed that the cost of parity/ECC is significantly higher than that of our set of techniques. This gives a point of reference for the overheads of our mechanism and shows that it is reasonable including the extra hardware for soft and hard error detection and confinement.

3.3 Example of Error Confinement Operation

In order to illustrate the operation of error confinement tables we show an example corresponding to the L1 cache. In the example we emulate the behavior of our scheme in the presence of a defect affecting the whole *row 37* in *array 1*. Consider the case where 64 consecutive accesses to *row 37* in *array 1* are performed (other accesses to other arrays or other rows in *array 1* may occur in-between but no errors are detected). Since each datum is 72-bits wide (64 bits for data and 8 for parity), we distinguish only 4 columns per array. The following distribution of errors is obtained:

- *Array 1, row 37, column 1:* 25 errors.
- *Array 1, row 37, column 2:* 19 errors.
- *Array 1, row 37, column 3:* 11 errors.
- *Array 1, row 37, column 4:* 9 errors.

This would lead to the following list of errors tracked in the different tables:

- **Array Table**
  - *Array 1:* 64 errors (saturates at 256).
- **Array Row Table**
  - *Array 1, row 37:* 64 errors (saturates at 64).
- **Array Column Table**
  - *Array 1, column 1:* 25 errors (saturates at 64).
  - *Array 1, column 2:* 19 errors (saturates at 64).
  - *Array 1, column 3:* 11 errors (saturates at 64).
  - *Array 1, column 4:* 9 errors (saturates at 64).
- **Array Datum Table**
  - *Array 1, row 37, column 1:* 25 errors (saturates at 32).

- *Array 1, row 37, column 2:* 19 errors (saturates at 32).
- *Array 1, row 37, column 3:* 11 errors (saturates at 32).
- *Array 1, row 37, column 4:* 9 errors (saturates at 32).

As shown, the only counter saturating would be the one corresponding to the *row 37* in *array 1*. Note that 32 or more accesses out of the 64 total accesses might occur in the same datum (e.g., *array 1, row 37, column 1*). In this case the counter for such datum would saturate first and the datum would be deactivated. However, further accesses to the *row 37* would either disable the remaining data of this row one by one, or the full row. At the end, the whole row would be disabled, as desired.

4 Related Work

Mechanisms for off-line testing at fabrication have been proposed in the past [26]. However, such mechanisms rely on some Automated Test Equipment (ATE) to inject the proper inputs to perform the tests. Off-line mechanisms have been extended to perform tests without any ATE [19], but they are still focused on off-line testing at fabrication to prevent faulty processors to be shipped. Some works have shown that slightly defective processors can be also shipped for yield increase [16, 25]. Although off-line testing mechanisms can be used for on-line testing, they may miss a significant number of faults that may show up intermittently. Since defects are expected to produce faults progressively, they will manifest only under certain environmental conditions (temperature, voltage, noise, crosstalk, etc.) and continuous on-line testing will be required to detect them timely.

Caches have some on-line testing features like parity or ECC [6], so many errors can be detected, but errors in wordlines, decoders, comparators and other circuits are not detected.

Some mechanisms check the outputs of the program instead of using codes. Brute force mechanisms are used to perform such coarse-grain checking. Typically reexecution is employed to produce the outputs twice or more times, and compare them to identify wrong results. For instance, lockstep [5], DIVA [2] and redundant multithreading either in a single SMT core [20] or in separate cores [15] are examples of coarse-grain concurrent testing. Most of those techniques do not replicate cache accesses [2,5,15,20], and thus, those errors not detected by parity or ECC are neither detected by those reexecution mechanisms. Only some implementations of lockstep [5] detect such errors, but the cost is huge in power (more than 2X), area (two cores are required to execute a single program) and performance. Moreover, errors are not confined so further techniques are required to identify the faulty component.

Synthetic program execution has been proposed as a method to force some errors to show up [17, 22]. Synthetic programs are run on the core and their output is compared against the expected output. The main disadvantage of those techniques is the fact that many errors are missed (i.e., soft errors) even if synthetic program execution is performed at worst-than-real execution conditions [22] (i.e., overclocking the processor) and errors are not confined. Other techniques based on periodic tests have been proposed [7,21]. However, such techniques have the same disadvantage as synthetic program execution because soft errors and many hard errors showing up intermittently are missed. Thus, continuous dynamic testing is much more suitable to detect errors timely.

To the best of our knowledge our set of techniques is the first approach to detect errors and confine them for hardware reconfiguration at low cost. Errors are detected concurrently with execution, so our technique is useful for both post-silicon testing and in-the-field error detection. Moreover, error confinement features are key to debug and find error location.

5 Conclusions

New process generations constrain burn-in capabilities thus increasing in-the-field fault rates. Parity and ECC are effective to detect soft errors and defects in bitcells of cache-like structures dynamically, but many soft errors and defects in logic require further error detection mechanisms. Thus, techniques to detect different types of errors in cache memory logic are required as well as methods to confine errors for efficient hardware reconfiguration. In this paper we present solutions to protect those components of caches that parity and ECC leave unprotected. In particular, the contributions of this paper are as follows:

(i) Memory rows are extended with hardwired signatures to detect faults in wordlines and row decoders, which represent most of the devices left unprotected by parity/ECC.

(ii) The remaining combinational blocks (column decoders and comparators) are replicated and signals between cache arrays are parity protected.

(iii) By tracking the location and frequency of errors, we can classify them into soft and hard, and can confine defects so that the minimum amount of hardware needs to be reconfigured.

We show that any kind of error left undetected in caches by parity/ECC can be detected and confined with power and area overheads largely below those of parity or ECC for both L1 and L2 caches.

Acknowledgements

This work has been partially supported by the Spanish Ministry of Education and Science under grants TIN2004-03702 and TIN2007-61763, and Feder Funds. We would like to thank anonymous reviewers by their comments.

References

[1] R. Aitken. Nanometer technology effects on fault models for IC testing. *IEEE Computer*, 32(11):46–51, 1999.

[2] T. Austin. DIVA: a reliable substrate for deep submicron microarchitecture design. In *MICRO 32: Procs of the 32nd ACM/IEEE Int'l Symp. on Microarchitecture*, pages 196–207, 1999.

[3] P. Bai et al. A 65nm logic technology featuring 35nm gate lengths, enhanced channel strain, 8 Cu interconnect layers, low-k ILD and 0.57 /spl mu/m/sup 2/ SRAM cell. In *IEDM '04: Procs. of the IEEE Int'l Electron Devices Meeting*, pages 657–660, 2004.

[4] T. Barnett, A. Singh, and V. Nelson. Extending integrated-circuit yield-models to estimate early-life reliability. *IEEE Trans. on Reliability*, 52(3):296–300, 2003.

[5] W. Bartlett and L. Spainhower. Commercial fault tolerance: A tale of two systems. *IEEE Trans. on Dependable and Secure Computing*, 1(1):87–96, 2004.

[6] C. Chen and M. Hsiao. Error-correcting codes for semiconductor memory applications: A state of the art review. *IBM Journal of Research and Development*, 28(2):124–134, 1984.

[7] K. Constantinides, O. Mutlu, T. Austin, and V. Bertacco. Software-based online detection of hardware defects mechanisms, architectural support, and evaluation. In *MICRO 40: Procs. of the 40th ACM/IEEE Int'l Symp. on Microarchitecture*, pages 97–108, 2007.

[8] M. Favalli and C. Metra. Online testing approach for very deep-submicron ICs. *IEEE Design and Test*, 19(2):16–23, 2002.

[9] R. Fritzemeier, H. Nagle, and C. Hawkins. Fundamentals of testability - a tutorial. *IEEE Trans. on Industrial Electronics*, 36(2):117–128, 1989.

[10] K. Ghose and M. Kamble. Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation. In *ISLPED '99: Procs. of the 1999 Int'l Symp. on Low Power Electronics and Design*, pages 70–75, 1999.

[11] T. Heijmen, F. Ruckerbauer, N. Seifert, M. Derbey, Y. Zorian, and P. Sanda. Panel: SER trends in 45nm and beyond. In *IOLTS '07: 13th Int'l On-Line Testing Symp.*, 2007.

[12] Y.-T. Hsing et al. Failure factor based yield enhancement for SRAM designs. In *DFT '04: Procs. of the 19th IEEE Int'l Symp. on Defect and Fault Tolerance in VLSI Systems*, pages 20–28, 2004.

[13] M. Ishida et al. A novel 6T-SRAM cell technology designed with rectangular patterns scalable beyond 0.18 μm generation and desirable for ultra high speed operation. In *IEDM '98: Int'l Electron Devices Meeting. Technical Digest*, pages 201–204, 1998.

[14] S. Kundu, T. Mak, and R. Galivanche. Trends in manufacturing test methods and their implications. In *ITC '04: Procs. of the Int'l Test Conf.*, pages 679–687, 2004.

[15] S. Mukherjee, M. Kontz, and S. Reinhardt. Detailed design and evaluation of redundant multithreading alternatives. In *ISCA '02: Procs. of the 29th ACM/IEEE Int'l Symp. on Computer Architecture*, pages 99–110, 2002.

[16] D. Nikolos and H. Vergos. On the yield of VLSI processors with on-chip CPU cache. *IEEE Trans. Computers*, 48(10):1138–1144, 1999.

[17] P. Parvathala, K. Maneparambil, and W. Lindsay. FRITS: A microprocessor functional BIST method. In *ITC '02: Procs. of the 2002 IEEE Int'l Test Conf.*, pages 590–598, 2002.

[18] L. Pileggi et al. Exploring regular fabrics to optimize the performance-cost trade-off. In *DAC '03: Procs. of the 40th Conf. on Design automation*, pages 782–787, 2003.

[19] R. Rajsuman. Design and test of large embedded memories: An overview. *IEEE Design and Test of Computers*, 18(3):16–27, 2001.

[20] E. Rotenberg. AR-SMT: A microarchitectural approach to fault tolerance in microprocessors. In *FTCS '99: Procs. of the 29th Int'l Symp. on Fault-Tolerant Computing*, page 84, 1999.

[21] S. Shyam et al. Ultra low-cost defect protection for microprocessor pipelines. In *ASPLOS-XII: Procs. of the 12th ACM Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 73–82, 2006.

[22] J. Smolens et al. Detecting emerging wearout faults. In *SELSE '07: Procs. of the 3rd Workshop on Silicon Errors in Logic - System Effects*, 2007.

[23] C.-L. Su and A. Despain. Cache design trade-offs for power and performance optimization: a case study. In *ISLPED '95: Procs. of the 1995 Int'l Symp. on Low Power Electronics and Design*, pages 63–68, 1995.

[24] D. Tarjan, S. Thoziyoor, and N. Jouppi. CACTI 4.0. *HP Technical Report HPL-2006-86*, 2006.

[25] H. T. Vergos, D. Nikolos, P. Mitsiadis, and C. Kavousianos. Reconfigurable CPU cache memory design: Fault tolerance and performance evaluation. In *VLSI '97: Procs. of the 9th IFIP Int'l Conf. on VLSI*, pages 103–114, 1997.

[26] D. Wu et al. An optimized DFT and test pattern generation strategy for an Intel high performance microprocessor. In *ITC '04: Procs. of the Int'l Test Conf.*, pages 38–47, 2004.