Dynamic Cache Resizing Architecture for High Yield SOC

Baker Mohammad[1], Muhammad Tauseef Rab[1,3], Khadir Mohammad[2], and M. Aater Suleman[3]
[1]Qualcomm Incorporated, [2]University of Texas at San Antonio, [3]University of Texas at Austin

Abstract—. Dynamic cache resizing coupled with Built In Self Test (BIST) is proposed to enhance yield of SRAM-based cache memory. BIST is used as part of the power-up sequence to identify the faulty memory addresses. Logic is added to prevent access to the identified locations, effectively reducing the cache size. Cache resizing approach can solve for as many faulty locations as the end user would like, while trading off on performance. Reliability and long term effect on memory such as pMOS NBTI issue is also compensated for by running BIST and implementing cache resizing architecture, hence detecting faults introduced over time. Since memory soft failures are worst at lower voltage operation dynamic cache resizing can be used to tradeoff power for performance. This approach supplements existing design time optimizations and adaptive design techniques used to enhance memory yield. Performance loss incurred due to the cache reduction is determined to be within 1%.

***Index Terms*— sram memory, caches, high yield, memory architecture, SOC design, processors design**

I. INTRODUCTION

Increasing process variability for new process technologies [1] coupled with increased reliability effects like Negative Bias Temperature Instability (NBTI) [2] all contribute to increased yield loss in chips due to SRAM-based memory failures. Caches constitute in excess of 50% of modern SOC and processors area and have more than 80% of the transistor count [3]. In addition to the fact that the SRAM cell is the most frequently used cell it also uses the smallest geometry transistor to increase area utilization which make it more susceptible to both device electrical and geometrical variations [4]. Memory hard failures due to manufacturing defects and soft failures due to voltage, temperature variations have rarely been considered an architectural problem. Yield issues have been viewed as a circuit related problem and almost all the research for improving yield has been in the circuits' area. Our proposal addresses the yield loss due to SRAM-based memory failure at the architectural level. This approach is based on dynamic cache resizing upon detection of a failure in the cache. It uses existing cache logic and Built In Self Test (BIST) to implement the cache resizing.

Traditionally, design time optimizations along with some adaptive design techniques, and redundancy have been employed to reduce the yield loss due to manufacturing or parametric shift. These approaches have limitations on the number and type of failures (read disturb, write failures, access failures) it can repair [4]. It also has an area, timing and cost overhead.

Section II will describe the background and few of the recent attempts to address yield in SRAM-based memory. Section III will introduce our approach (dynamic resizable cache) and in section IV an analysis of impact on performance will be studied. We will conclude with a summary and future work in section V

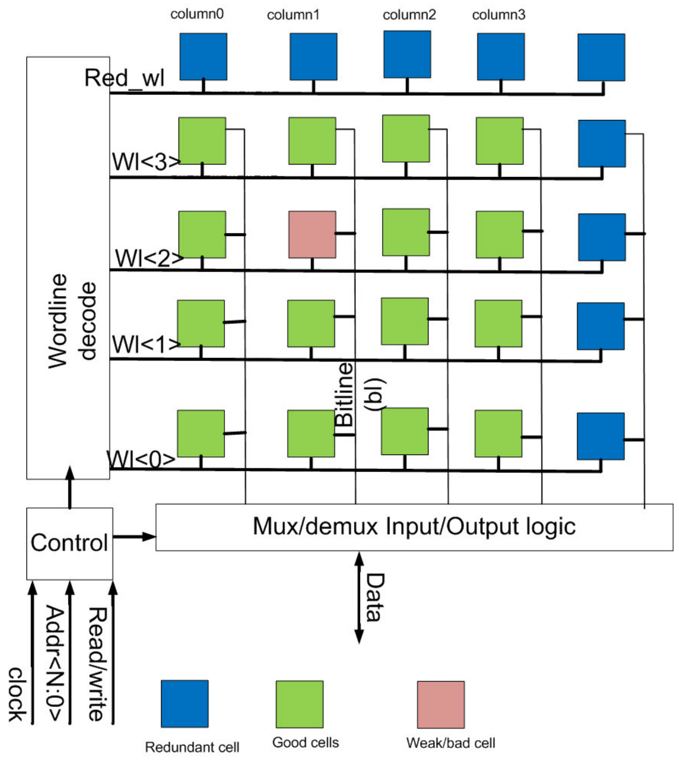

Figure 1 : SRAM-based memory blocks showing redundant cell

II. Background and Review of Current Approaches to Memory Yield Loss

Figure 1 shows the basic organization of SRAM-based Memory main block and interface. The number of columns and rows are determined based on banking options, performance, and power. The main purpose of the on chip memory is to keep the intermediate data and to act as a buffer between the main memory and the processor. The size of the on chip memory is growing due to the ever increasing gap between processor frequency and main memory (memory wall). The probability of the part to be non functional increases as the number of SRAM cell is increased. Table 1 illustrates the yield as a function of the memory size and the probability of the SRAM cell to fail

Table 1: Chip yield as a function of memory size and cell failure rate

memory size in Kilo Byte	total number of cells	Yield when 1 out 100K cell fail	Yield when 1 SRAM cell out of 1 million cell fail
256	2097152	0%	12.3%
128	1048576	0%	35.0%
64	524288	0.5%	59.2%
32	262144	7.3%	76.9%
16	131072	27.0%	87.7%

Memory failures are traditionally compensated for by three main approaches. Firstly, a design time optimization through selecting the right SRAM cell which involves a complex tradeoff between area utilization, performance, power and yield [?] may be used. The bigger cell area usually means more stable SRAM and better yield but the cell density per area decreases. .Secondly, an adaptive and tunable design which changes the behavior of the memory cell electrical characteristics based on the process, voltage, and temperature status on the chip [5] is used. This approach improves yield but can not fix all the failures and there will still be some yield loss due to un-tunable cells. The third approach employed is to repair the failure through swapping the identified failing cells with working cells. This is done through adding extra memory elements referred to as redundant columns or rows. Figure 1 shows the basic SRAM-based memory block with main interface signals and main blocks showing potential redundant cells and fail cells. . If there is a failure in a certain memory bank (identified by BIST) then the entire column/row of the failing location is swapped out with the spare column/row. This technique works well if the failures are limited as each additional failure requires a spare column/row. Every bad cell in a unique memory column requires an additional spare column to rectify it. As we see in this approach there is a linear dependency of area with the number of fails that can be tolerated. The addition of these redundant SRAM cells and the associated logic (fuses, special BIST) add to the cost of the product [8]. The approach is also limited to repair only certain number of cells in a block resulting in yield loss.

III. Resizable Cache Architecture

The cache sub-system as shown in Figure 2 consists of the data array which has the main data storage and tag array which determine the way-hit in the cache based on comparing the physical page number (ppn) with the tag value determined by the index bits. Additionally, each tag has state bits which hold important information regarding the validity of the tag. Depending on the cache architecture, the state bits can range from 1 to 3 or 4 bits for each cache line [7]. Both the hit way and the state array along with the index bits determine the data array access. Caches are organized so that every memory location has a specific entry or entries, depending upon the associativity of the cache. Any one of these possible entries is called a 'way'. Therefore every way in the cache has its own state bits. Our approach is to add an extra state bit per way to indicate if the cache memory location corresponding to this way is bad or not. We will call this bit, a ***mask*** bit.

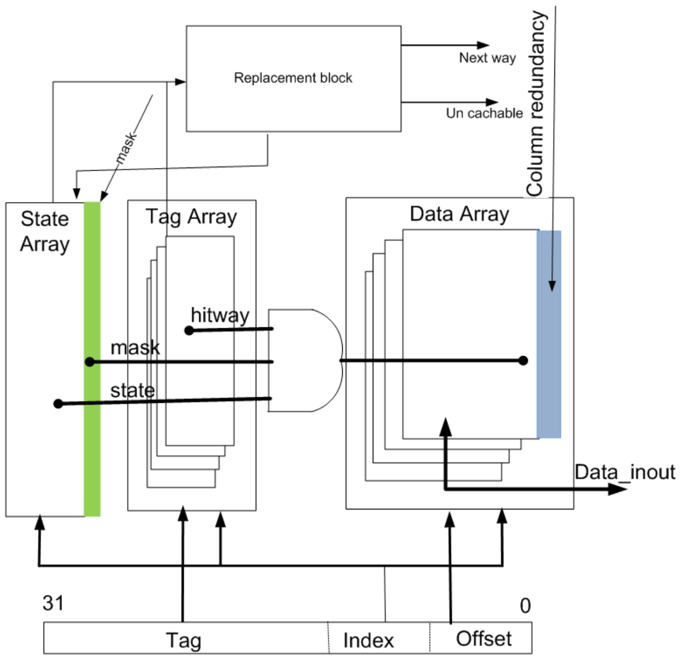

Figure 2 : Cache subsystem main blocks

Our approach utilizes Built In Self Test (BIST) which is used traditionally to screen memory for failures and identify functional versus non functional parts. We modified BIST logic to identify the failing memory addresses and then retain that information by setting an appropriate mask bit corresponding to the failing location in the memory. This requires adding an additional bit for each way in the index; we will refer to it as the mask bit. The mask bit will be like any other state bit present (valid and dirty) in the traditional caches used today. The mask bits will be reset to a '0' and if in BIST a particular memory location fails, it will set its corresponding mask bit to a '1'. The idea is to repeat this cycle every time we power up so that we can keep updating the mask bits over time, taking in to account all the recent failures incurred due to long term effects. The architecture can decide on the total number of failures the design can handle ahead of time and store this information in an architecture register which can be updated through software. A counter will be used to calculate the total number of failing locations, which can be done by counting the number of mask bits set. If and when the maximum number of tolerated failures exceeds, the chip can then be marked as 'bad' accordingly.

Additionally, we have to take into account the loss of performance by effectively reducing the cache size, the details of which can be found in the performance analysis section (IV)

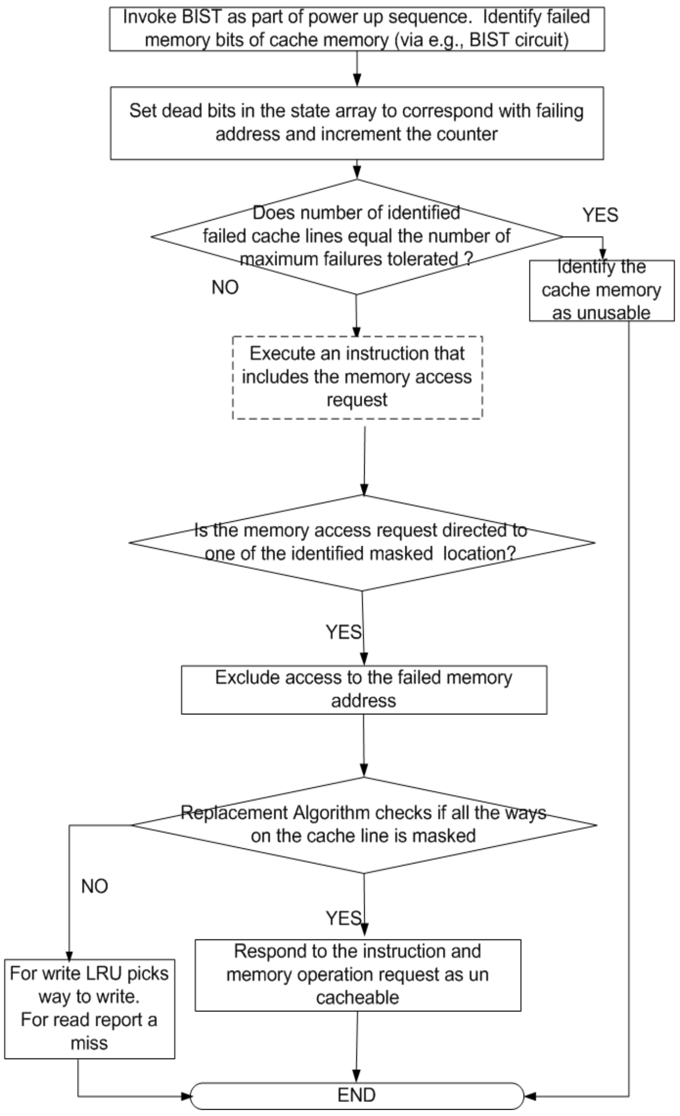
Invoke BIST as part of power up sequence. Identify failed memory bits of cache memory (via e.g., BIST circuit)
Set dead bits in the state array to correspond with failing address and increment the counter
Does number of identified failed cache lines equal the number of maximum failures tolerated ?
YES
NO
Identify the cache memory as unusable
Execute an instruction that includes the memory access request
Is the memory access request directed to one of the identified masked location?
YES
Exclude access to the failed memory address
Replacement Algorithm checks if all the ways on the cache line is masked
NO
YES
For write LRU picks way to write. For read report a miss
Respond to the instruction and memory operation request as un cacheable
END

Figure 3: Flow chart of the proposed dynamic cache architecture

The trade-off here is between the loss in performance and the gain in yield.
Figure 3 shows a detail flow chart of how the cache access will look like when the dynamic cache resize is used to improve yield.

A. Read operation

The tag array is compared with the in coming tag to figure out if it is a hit or a miss. There are state bits associated with the tag entries also which keep track of its validity and in the case of a write back cache, if the content of the memory has changed in the cache since it was brought inside the cache. We added another bit called the mask bit which will now be used as an additional gating to figure out if it is a hit or not. The mask bit is set for each of the location which is determined to be faulty during the BIST. If there is an access to one of the locations whose mask bit was set by the BIST, it will now be reported as a miss in the case of a read.

B. Write Operation

A multiple-way associative cache requires an algorithm to determine which way to write for a given cache line. This is done using a replacement algorithm like Least Recently Used (LRU) block. Traditionally the LRU block uses the index bits to access the state rams and uses its output and the state bits to determine the next way. In our approach we added 'n' mask bits (n = number of ways) and the LRU reads them in addition to the original state bits to determine which way to write next. By doing this the LRU masks out all the ways which have faults. This dynamically reduces the over-all cache size but increases yield.

IV. PERFORMANCE ANALYSIS

Our approach to improve yield is based on cache resizing. This has impact on the performance due to the potential decrease in the available entries in the cache. This is because we take a performance hit when ever we find a bad cell and internally treat it as an un cacheable address. This effectively reduces the memory size but improves yield. To quantify the impact of cache resizing on processor performance we simulated variant cache sizes with different fault numbers.

A. Methodology

We used a cycle accurate x86 simulator [9] to analyze all 28 SPEC CPU2006 [10] benchmarks using the reference input set. Each program was run for 200 million instructions and the representative program slices were chosen using the Simpoint methodology [11]. The cache we simulated is 1 Meg, 8-way set associative and employs an LRU replacement policy. We randomly inject faults in the cache. For our experiments, we vary the cache sizes and fault ratios across simulations.

B. Results

Figure 4 shows how the hit ratio of each benchmark changes as the percentage faults increases in presence of our scheme. Intuitively, as the percentage of faults increase, the hit ratio decreases. When the fault ratio is 1%, the hit ratio reduces by less than 7% across all benchmarks. The mean reduces by only 1%. When fault is 5%, the reduction hit ratio is less than 10% across all benchmarks except sjeng and gamess. The data set of these benchmarks is approximately 1MB. Since the faults reduce the effective cache size, the data set no longer fits in the cache. Thus, the hit ratio reduces rapidly in presence of faults. Similar behavior is seen for several benchmarks as the fault ratio increases to 10% and 15%. The average reduction in hit ratio is 3%, 12%, and 23% for 5%, 10%, and 15% faults respectively. This reduction in hit ratio is tolerable since the chip yield increases to a 100% using our proposed scheme**.**

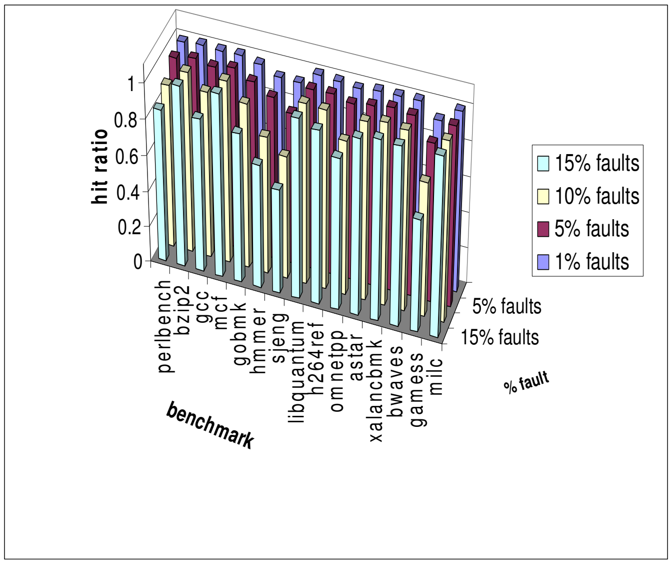

Figure 4: Performance impact due to reduce cache size

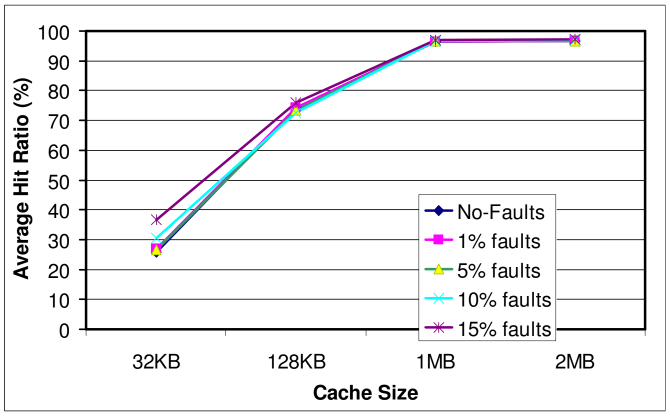

Figure 5: Hit rate with different cache size and varying failure rate

Figure 5 shows how the average hit ratio across benchmarks changes as the cache size increases. The Y-axis is harmonic mean of hit ratios across all SPEC2006 benchmarks. The five lines show the hit ratio at each cache size for caches with different percentage of faults. As expected, the performance of our scheme is best when the cache size is large and fault ratio is small. On the other hand, the scheme performs worst compared to the baseline at the small cache size and high fault ratio. This is because at a smaller cache size, even a small reduction in cache size can significantly impact cache performance. Since our scheme reduces effective cache size to increase yield, we see this reduction in performance. However, as the cache size increases, the difference between our scheme and the idealistic baseline, with no faults, closes. As technology enables more transistors on the chip, both cache sizes and fault ratios are expected to increase. Consequently, our scheme becomes feasible in the future since the performance loss will be marginalized while the improvement in yield will further increase.

V. Summary and Future Work

We see that our proposal solves for yield, which is one of the most critical design parameter, with minimum additional circuitry. Although there is a loss in performance, we have determined that this loss in performance is low. Furthermore, our approach can work in addition to the traditional approaches of solving for yield by using spare columns. In addition to yield the system power can also be optimized and trade off for performance as the memory failures increases at lower voltage. As part of the future study, we plan to look at hybrid solutions of spare columns and the currently proposed dynamic shrinkage of memory. The study of trading off power (by reducing the voltage) for performance by dynamic cache resizing method will be analyzed. In conclusion we will also do a study on area savings by eliminating the need of spare columns, which is how the traditional approach can be extrapolated to solve for multiple bad memory locations.

References

[1] Kuhn, K. Reducing variation in advanced logic technologies: Approaches to process and design for manufacturability of nano scale CMOS, Proc. IEDM, December 2007, pp. 471-474

[2] J. G. Massey. NBTI: what we know and what we need to know - a tutorial addressing the current understanding and challenges for the future, In IEEE International Integrated Reliability Workshop Final Report, 2004; pp. 199–211

[3] George R. Roelke, Rusty O. Baldwin, A cache Architecture for the Extremely unreliable nanotechnologies, IEEE Transactions on reliability, Vol. 56, June 2007

[4] Mukhopadhyay, S.; Mahmoodi, H, and Roy, K. Modeling of failure probability and statistical design of SRAM array for yield enhancement in nanoscaled CMOS, Proc. of TCAD , December 2005, pp. 1859-1880

[5] Baker Mohammad; Martin Saint-Laurent, Paul Bassett; and Jacob Abraham; Cache Design for Low Power and High Yield; IEEE ISQED Conference; March 2008

[6] Jitendra B. Khare, Memory yield improvement-SOC Design Prospective, ITC International test conference, 2004

[7] Hennessy, J. and Patterson, D. Computer Organization & Design, 3rd ed., Morgan Kaufmann 2005

[8] Said Hamdioui, Georgi Gaydadjiev, Ad J. van de Goor, The state-of-art and future trends in testing Embedded Memories, Memory technology, design and testing, 2004

[9] M. Aater Suleman, Moinuddin Qureshi., and Yale Patt. Feedback-driven threading: power-efficient and high-performance execution of multi-threaded workloads on CMPs. In ASPOS XIII, 2008.

[10] SPEC CPU Benchmark suite. http://www.spec.org

[11] Erez Perelman, Greg Hamerly and Brad Calder. Picking Statistically Valid and Early Simulation Points , In the International Conference on Parallel Architectures and Compilation Techniques, September 2003.