

# Delay and Area Efficient First-level Cache Soft Error Detection and Correction

Karl C. Mohr and Lawrence T. Clark  
Arizona State University Dept. of Electrical Engineering  
Tempe, AZ USA  
[{karl.mohr,lawrence.clark}@asu.edu](mailto:{karl.mohr,lawrence.clark}@asu.edu)

**Abstract**—Soft error rates are an increasing problem in modern VLSI circuits. Commonly used error correcting codes reduce soft error rates in large memories and second level caches but are not suited to small fast memories such as first level caches, due to the area and speed penalties they entail. Here, an error detection and correction scheme that is appropriate for use in low latency first level caches and other small, fast memories such as register files is presented. The scheme allows fine, e.g., byte write granularity with acceptable storage overhead. Analysis demonstrates that the proposed method provides adequate soft error rate reduction with improved latency and area cost.

**Index Terms**—Error detection and correction, memory soft errors, error correcting codes.

## I. INTRODUCTION

Single event upset (SEU) due to charge injected at Si junctions by impinging ionizing radiation is an increasing problem in modern integrated circuits and particularly microprocessors, which employ large static random access memories (SRAM's) for cache memory. Process scaling reduces the critical charge needed to upset storage nodes and increasing SRAM bits per chip exacerbates the problem [1]. Reported values for SRAM per bit soft error rate (SER) vary substantially with technology details, ranging from 0.01 to 0.0001 FIT/bit. One failure in time (FIT) is commonly defined as 1 failure in  $10^9$  hours of device operation [1]-[4].

Soft errors have driven designers to add error correcting codes (ECC) or parity protection to caches in order to meet SER requirements [5]-[9]. Conventional ECC protection imposes significant area and cycle time penalties, making it practical only for large embedded memories and second-level (L2) caches where the increased latency has less impact on performance. To maintain low latency, first level (L1) caches tend to employ parity checking that allows single bit error detection, but no correction. This is adequate for instruction caches, as the SRAM is never modified and another copy exists elsewhere in the memory hierarchy. For data caches, parity protection alone mandates a write-through policy so two copies of the written data exist.

Granularity of the protection is also an issue. Typically, microprocessor instruction sets allow loads and stores with byte granularity. Where a parity bit is stored with each byte, a byte write updates just the corresponding parity bit, which is calculated as the data is delivered to the data array. With standard ECC the large number of check bits result in a cost that is impractically high for such small granularities. Arrays with ECC usually employ a write buffer, where data writes at less than a line width are combined with the old data. This is accomplished by reading the cached line, merging it with the new data and recalculating the ECC bits for storage in a multi-cycle operation [9]. L1 caches with write buffering can use similar techniques [5][6].

In this paper a novel “lightweight” error detection and correction (LEDAC) scheme with overhead and timing similar to standard parity protection is proposed. Based on two dimensional parity checking, it imposes a minimal increase in circuit area and latency and is thus appropriate for high-speed memory structures such as L1 caches. Two dimensional parity schemes date back decades [10], but have heretofore not been used for cache or SRAM EDAC. Here, the operation, necessary circuits, and correction operations are outlined. With a combined memory read-write cycle, the scheme can be supported with minimal timing penalty.

### A. Microprocessor Cache EDAC

Many modern microprocessors employ ECC to protect the L2 caches. L1 caches commonly employ parity protection without correction capabilities, although some employ ECC at the cost of additional cycles of latency. This is beneficial in the case that the L2 is non-inclusive, i.e., the L1 contents are not replicated in the L2. Although parity can be effective for instruction and write-through data caches, it is ineffective for write-back data caches where the only valid (dirty) data resides in the L1. Write-back caches reduce memory traffic to the L2 cache and so are less likely to stall the machine when a large number of back-to-back load/store operations are executed. In small, embedded processors, there is not always an L2 cache and hence the performance penalty for write-through operation is higher.

ECC's used in microprocessors have been based on two related schemes, the Hamming and the Hsiao or odd-weight-column code [11][12]. They work in a similar way in that they add  $r$  check bits for every bundle of  $k$  data bits, which when

This work was sponsored by AFRL/VSS at Kirtland AFB, in Albuquerque, NM under contract F29601-00-D-0244 0020.

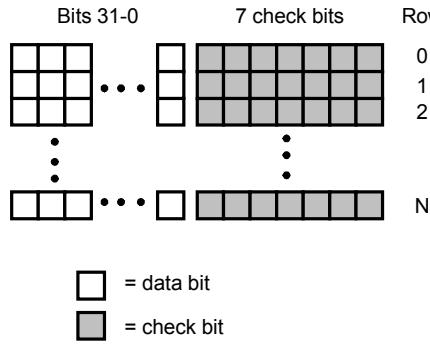


Fig. 1.  $N \times 32$  memory with conventional EDAC protection ( $k = 32, r = 7$ ).

decoded allow a single bit error to be corrected and any two bit error to be detected within the bundle. When data is written into the memory the check bits are calculated and stored along with the data bits (although as a practical matter they are sometimes stored in a separate array). After data is read the check bits are recalculated and exclusive-ORed with the stored check bits to generate a “syndrome” that is also  $r$  bits in length. If the syndrome contains all zeros then no single or double bit error has occurred and the data is assumed to be correct. If any bits in the syndrome are ones at least one data bit has been corrupted.

Double bit errors produce a quasi-unique syndrome that allows for the detection but not correction of the error. Thus these codes can allow double error detection (DED) but not correction. An example of the data and check bit storage for an  $N$  word  $\times$  32-bit memory is shown in Fig. 1. Note that the bundle size for this  $N \times 32$  array is 32 and thus it contains  $N$  32-bit bundles, one on each row, and each bundle is stored with seven check bits. For the Hamming/Hsiao codes each bundle can have one error and all the errors are correctable, i.e., they are independent. For a write width less than the bundle size, e.g., to write a byte, the entire bundle must be read, the data to be written inserted into the bundle, and the ECC check bits recalculated. Due to the width and depth of the parity trees required, this implies a multi-cycle operation. Where supported, this is handled by a fully-associative write buffer. Alternatively an 8-bit bundle could be used (not shown) with

five check bits. While this would remove the need for a write buffer it requires 62.5% area overhead for check bit storage.

## II. LIGHTWEIGHT EDAC

### A. Checking Scheme

The LEDAC scheme provides speed and area benefits for small fast memories, such as L1 caches, compared to standard ECC’s. As mentioned, the method is based on two-dimensional parity checking [10]. As illustrated in Fig. 2, one row check bit is added to every  $k$  bit bundle. The row check bit is a simple parity bit and thus the check bit calculation incurs the same latency as parity only protection. Row check bits are calculated as

$$\begin{aligned} \text{For } sb &= 0 \dots B-1 \\ \text{For } row &= 0 \dots N-1 \\ CR_{row,sb} &= b_{row,sb*k+0} \oplus b_{row,sb*k+1} \oplus \\ &b_{row,sb*k+2} \oplus \dots \oplus b_{row,sb*k+k-1} \end{aligned}$$

Here,  $N$  is the number of rows in the super-bundle,  $b_{row,col}$  indicates the bit in row ( $row$ ) and column ( $col$ ) of the array,  $CR_{row,sb}$  is the row check bit for row ( $row$ ) and super-bundle ( $sb$ ), and  $B$  is the number of super-bundles in the array. Here we defined the term super-bundle to be the set of data bits that have an associated set of  $k$  column check bits and  $N$  row check bits. The  $CR_{row,sb}$  bit indicates which  $k$  bit bundle has an error. To determine which of the  $k$  bits in the bundle has been upset  $c$  column check bits  $CC$  are added, at least one for each column in the array. The  $CC$  is simply a parity bit calculated column-wise and is calculated by XORing of all bits in the column. The  $CC$  bit storage and generation logic incurs a small overhead as will be shown. Unlike conventional ECC, repair is through a multi-cycle error correction routine, which can be implemented in software or hardware. The memory must also be initialized to a known state at reset.

Column check bits  $CC$  are calculated as

$$\begin{aligned} \text{For } sb &= 0 \dots B-1 \\ \text{For } col &= 0 \dots k-1 \\ CC_{sb*k+col} &= b_{sb*k+col,0} \oplus b_{sb*k+col,1} \oplus \\ &b_{sb*k+col,2} \oplus \dots \oplus b_{sb*k+col,N-1} \end{aligned}$$

For the  $N \times 32$  array of Fig. 2,  $k = 8$  and there are a total of  $4N$  bundles, and 4 super-bundles. Each super-bundle has  $N$

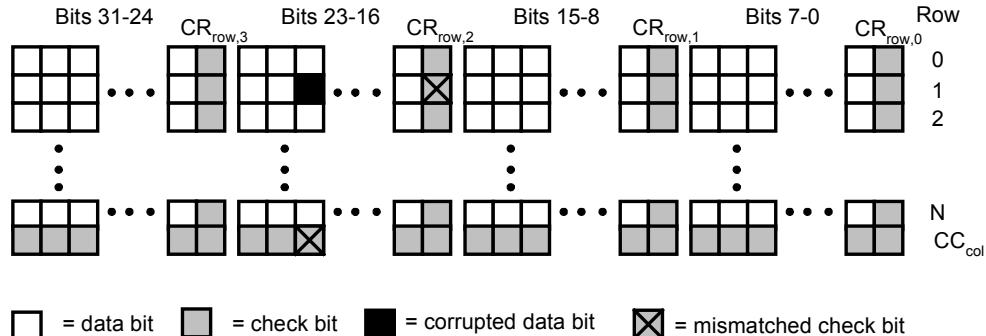


Fig. 2  $N \times 32$  bit memory array with LEDAC ( $k=8, r=1, c=32, d=32$ ). A SEU is shown in bit 20 of row 2.

rows of 8 bits. Every super-bundle is independent, i.e., any single error in each can be corrected.

### B. Number of Check Bits

As few as  $k$  column check-bits can be used, but  $k$  should not be less than  $d$ , the maximum data width that can be read or written in a single cycle. For example, if  $k = 8$ , and  $d$  can be 8, 16, or 32 bits, then  $c$  should not be less than  $\max(k, d) = 32$ . A smaller value increases the depth of the column check generation logic, which is generally in a timing critical path. On every read the row check-bits are generated from the read data and compared to the  $CR$  stored in the array. If the calculated and stored  $CR$  match, then no single bit error has occurred and the data is assumed to be correct. A discrepancy between the calculated and stored row  $CR$  indicates an error whereupon an error correction routine is invoked.

### C. Circuits and Operation

On every write  $CR$  is calculated and written into the array for each  $k$  bit bundle. For example, when a 32-bit word is written with one row check bit for each byte ( $k = 8$ ) then four row  $CR$  are calculated and written into the array along with the 32 data bits. The  $CC$  are also updated to reflect the new data being written into the array. This requires that the data bits to be written be exclusive-ORed with the old data bits to be overwritten and the result exclusive-ORed with the presently stored  $CC$  bit. The  $CC$  update circuit is shown in Fig. 3. The need to use the data being replaced to update the state of the column check bit requires a read of the old data before the write can occur. An appropriate single-cycle read before write scheme identical to that presently used in virtually tagged microprocessors [13] can be used. The XOR path from the sense amplifier to the  $CC$  storage does not comprise a timing critical path since the new  $CC$  will not be used for at least another clock phase, i.e., at the next write. While the  $CC$  storage is shown as a flip-flop in Fig. 3, SRAM storage can also be used.

### D. Correction Algorithm

An error correction routine must be run to correct an error in the SRAM once it is found. Fig. 4 illustrates one erroneous bit to show the correction procedure. For simplicity, and with no loss of generality, the array contains all 0's except for the error bit. The bit in row 1 and column 1 (which shows a logic 1 state) is the bit in error since the corresponding row and column parity are incorrect. During a read of row 1, the calculated and stored  $CR$  bits mis-match, invoking the correction routine. The first step is to store  $CC$  to a temporary register. In subsequent steps, the bits in adjacent rows are exclusive-ORed and stored in the  $CC$  bits, which are used as a local temporary register giving (by step)

- Step 1: Store  $CC$  to a temporary register.
  - Step 2:  $CC = 0\ 1\ 0\ 0$ , XOR row 1 and row 2.
  - Step 3:  $CC = 0\ 1\ 0\ 0$ , XOR result and row 3.
  - Step 4:  $CC = 0\ 1\ 0\ 0$ , XOR result and row 0.
  - Step 5: Syndrome =  $CC = 0\ 1\ 0\ 0$ , XOR result and temporary register.
- For the final step, the resulting  $CC$  bits are exclusive-ORed

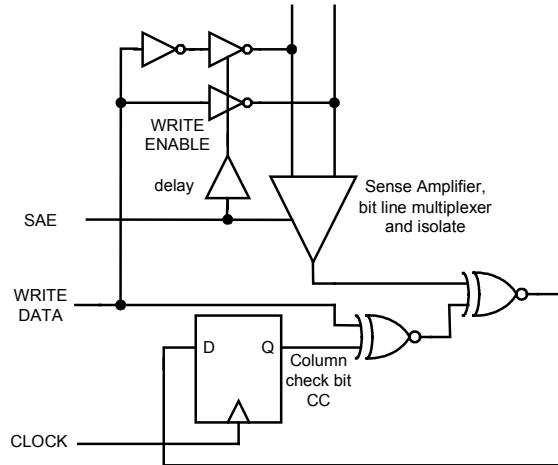


Fig. 3. CC bit update circuitry. The XNOR gates are not in the critical timing path.

with the temporary register containing the original column parity bits. The result is the syndrome where a 1 points to the incorrect bit in column 1.

To ensure that multi-bit errors are corrected, a second pass is made through the array exclusive-ORing the syndrome bits with the data for each  $k$  bit bundle where the row check bit (parity) indicates an error has occurred, i.e., a scrub of the entire array is performed. At this time the  $CR$  are recalculated and written back into the array. The  $CR$  must be recalculated in case the bit that is in error is in the row-check bits rather than the data. While the checking and updating is much faster than with conventional ECC approaches, the correction, when required, requires  $N+I$  cycles, depending on whether or not the block (super-bundle) contains multiple errors. However, LEDAC requires little additional hardware. The LEDAC error correction routine is more complex than for either the Hamming or Hsiao codes but can rely on the microprocessor CPU. The error correction procedure will run infrequently as SEU are rare events and so will not affect throughput.

The example demonstrates that the proposed LEDAC scheme can handle SEC cases. For DED the correction routine must be modified to detect multiple row and/or column bit upsets. If more than one row or column check-bit error is detected by the error correction routine in any one super-bundle then more than one error has occurred and the errors are not correctable. A difference between the DED for the LEDAC scheme compared to the standard Hsiao/Hamming approach is where there are two errors in the same row of a super-bundle. In this case a double error will not be detected when the data is read, but will be discovered when scrubbing, since two CC mis-matches will be found. All schemes require proper physical interleaving of the bits to avoid a single strike causing multiple upsets as shown below.

## III. AREA AND SPEED

EDAC requires extra area for check bit storage and parity trees, and the time needed to calculate check bits can cause an

Row	
0	0 0 0 0 0
1	0 1 0 0 0
2	0 0 0 0 0
3	0 0 0 0 0
$CC_{col}$	0 0 0 0 0

Fig. 4. Example error in a four by four bit array.

increase in latency, as mentioned. Hsiao codes are slightly faster than Hamming codes and require the same number of check bits [14] so here only the Hsiao, and LEDAC schemes are compared.

#### A. Area Overhead

For a Hsiao code the relation between the number of check-bits  $r$  and number bits in a bundle  $k$  is

$$\min \left[ \sum_{\substack{i \leq i_{\min} \leq r \\ i=1 \\ i \text{ odd}}}^r \binom{r}{i} \right] \geq r + k \quad (1)$$

For the LEDAC approach the relationship between the number of column-check bits  $c$ , bits per bundle  $k$ , and the number of row check-bits  $r$  for any value of  $k$  is

$$c \geq k, r = 1 \quad (2)$$

The number of column bits should not be set to less than  $16 * \max(k, d)$  to avoid problems with particle strikes that cause multi-bit errors giving

$$c = 16 \cdot \max(k, d), r = 1 \quad (3)$$

This relationship can be used to calculate the minimum area overhead, i.e., check bits vs.  $k$ , with the results shown in Table I for a 32kB memory. The parity tree depth is calculated assuming the use of 2 input XOR gates. If XOR gates with a greater number of inputs are used then this number is reduced but it is reduced for all methods equally.

#### B. Latency Overhead

To calculate the worst-case path through the check-bit generation logic for Hsiao code we find the smallest value of  $\min(i)$  such that (1) is satisfied and

$$HD = \left\lceil \log_2 \left( \sum_{\substack{i \leq i_{\min} - 1 \\ i=1 \\ i \text{ odd}}}^r \frac{i}{r} \binom{r}{i} + \frac{i_{\min}}{r} \left\{ r + k - \sum_{\substack{i \leq i_{\min} - 1 \\ i=odd}}^r \binom{r}{i} \right\} \right) \right\rceil \quad (4)$$

Table I. LEDAC ( $k = 8$ ) vs. Hsiao code for  $k = 8, 32, 64, 128$  supporting a byte write.

		LEDAC	Hsiao	Difference
Hsiao $k=8$	Area cost (%)	12.7	62.5	-49.8
	Logic delay	$t_{Read} + t_{Write} + t_{XOR}$	$t_{Write} + 3 t_{XOR}$	$t_{Read} - 2 t_{XOR}$
Hsiao $k=32$	Area cost (%)	12.7	21.9	-9.2
	Logic delay	$t_{Read} + t_{Write} + t_{XOR}$	$t_{Read} + t_{Write} + 4 t_{XOR}$	$-3 t_{XOR} - \text{clock cycle}$
Hsiao $k=64$	Area cost (%)	12.7	12.5	0.2
	Logic delay	$t_{Read} + t_{Write} + t_{XOR}$	$t_{Read} + t_{Write} + 5 t_{XOR}$	$-4 t_{XOR} - \text{clock cycle}$
Hsiao $k=128$	Area cost (%)	12.7	7	5.5
	Logic delay	$t_{Read} + t_{Write} + t_{XOR}$	$t_{Read} + t_{Write} + 6 t_{XOR}$	$-5 t_{XOR} - \text{clock cycle}$

is used to calculate the check-bit generation logic depth. The ceiling function rounds the result up to the nearest integer. For the case where  $c = 16 \max(k, d)$  the depth of the column and row parity tree depth for LEDAC is one XOR gate and  $\lceil \log_2(k) \rceil$ , respectively. The latter is not in the critical timing path. The extra time to read before writing is however, and increases the timing critical path length by 2 gate delays.

#### C. Latency Overhead

All EDAC schemes present an area and latency trade-off. As  $k$  is increased the area overhead decreases but the check-bit generation logic width and depth increases. Longer L2 cache pipelines provide time for deep check bit generation logic supporting large bundle sizes with low area overhead, e.g., with  $k = 256$  and  $r = 10$  using a Hsiao code, the bit area overhead is 3.9% and check-bit logic depth is 7 XOR gates (14 inversions). A 32-Kbyte cache using a LEDAC code with  $k = 8$ , and a 32 bit column has an area overhead of 12.7%, which is similar to the value required for parity based single-bit error detection. The row and column check bit generation logic depth is 3 and 1 respectively for a LEDAC bundle size of 8 (only the time after sensing is counted). The former is the same as for parity protection and the latter is timing non-critical.

Thus, LEDAC provides a significant latency advantage over the conventional ECC approaches. This is significant since the total number of gate delays (inversions) in one clock cycle for high performance microprocessors ranges from 10-20. Table I compares area overhead and check-bit generation logic depth for a LEDAC code with  $k = 8$  compared to Hsiao codes for various values of  $k$ . The last column shows the difference in area and gate delays. A negative value favors LEDAC.

A Hsiao code with  $k = 8$  is likely faster than the LEDAC approach (depending on the speed of the read vs. XOR gate delay) but has a much larger area overhead due to the need to

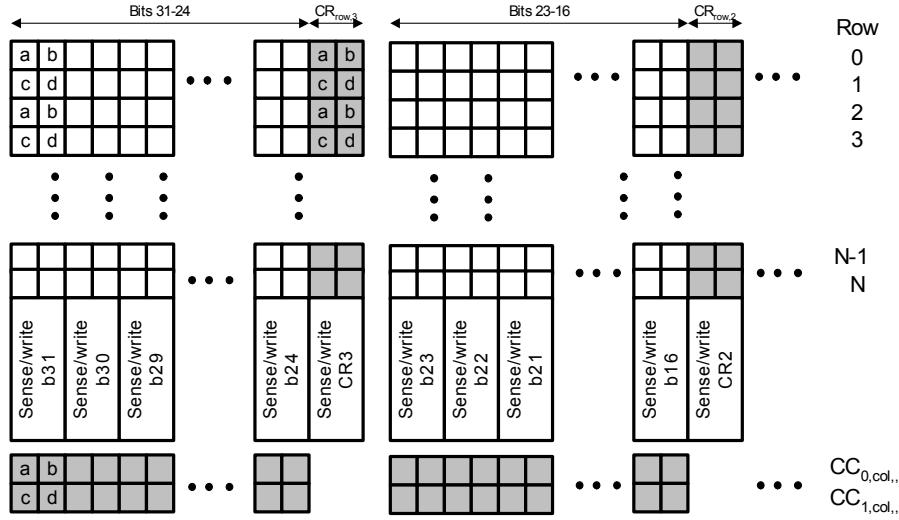


Fig. 5. Layout with row and column interleaving. Bits a and b are in different words, as are bits c and d.

add 5 check bits for every 8 data bits. Once  $k > 8$  the need to support byte writes forces a multi-cycle read-modify-write operation using the Hsiao approach and its speed drops significantly below that of LEDAC. Increasing  $k$  improves area efficiency at the cost of speed. Thus LEDAC with  $k = 8$  represents a good compromise for L1 caches.

#### IV. CONCLUSIONS

The point of adding EDAC is to improve the SER so that it becomes unimportant to the overall device MTTF. Assuming that SEU cause only single bit errors, which is ensured by proper layout then

$$MTTF = \frac{t}{1 - \left[ e^{-\lambda \frac{M}{B} t} + \lambda \cdot \frac{M}{B} \cdot e^{-\lambda \frac{M}{B} t} \right]^{B \cdot C}} \quad (5)$$

where  $t$  is the interval between scrubbing,  $C$  number of memory arrays per die,  $\lambda$  is bit failure rate,  $M$  is the array size in bits, and  $B$  is number of super-bundles in each memory. An example MTTF for a bit failure rate of 0.01 FIT/bit, a die with 32 cores, where each core contains one 64kB SRAM data array can be calculated. It is assumed that no code is being executed between scrub cycles. In reality, normal code execution would result in greatly reducing the effective scrub time and consequently the SER. Even for a large scrubbing interval of 1 day the MTTF is increased to well over 100,000 years assuming terrestrial neutrons cause all upsets.

The Hamming and Hsiao codes become slower and more space efficient with increased bundle size. They fit well with the relatively long latency requirements of L2 caches and are thus commonly used. For L1 caches, which are usually byte writable, traditional EDAC with byte-sized bundles is sufficiently fast, but requires 62.5% area overhead. Consequently, it is only used in concert with write buffering, where larger bundles can be accommodated. The LEDAC scheme proposed here has been shown to support the

necessary byte write granularity with a low area overhead of approximately 12%. A multi-cycle read-modify-write is not required. A single-cycle read-write is required, but has been used in microprocessor L1 caches and is readily adaptable to this purpose.

#### REFERENCES

- [1] R. Baumann, "The Impact of Technology Scaling on soft Error Rate Performance and Limits to the Efficacy of Error Correction", *IEDM Tech. Dig.*, pp. 329-332, 2002.
- [2] H. Kobayashi, et al., "Comparison Between Neutron-Induced System-SER and Accelerated-SER in SRAMs", *Proc. IRPS*, pp. 288-293, 2004.
- [3] D. Lambert, et al., "Neutron-Induced SEU in Bulk SRAMs in Terrestrial Environment: Simulations and Experiments", *IEEE Trans. Nuc. Sci.*, 51, 6, pp. 3435-3441, Dec. 2004.
- [4] T. Granlund, B. Granbom, N. Olsson, "Soft Error Rate Increase for New Generations of SRAMs", *IEEE Trans. Nuc. Sci.*, 50, 6, pp. 2065-2068, Dec. 2003.
- [5] Online: AMD Opteron Product Data Sheet, <http://www.amd.com>, June 2004.
- [6] Online: Intel Pentium Processor Extreme Edition 840 Datasheet, <http://intel.com>, April 2005.
- [7] J. Tendler, et al., "POWER4 System Microarchitecture", *IBM J. Res. & Dev.*, 46, 1, pp. 5-25, Jan 2002.
- [8] N. Seifert, D. Moyer, N. Leland, R. Hokinson, "Historical Trend in Alpha-particle induced soft Error Rates of the Alpha Microprocessor", *Proc. IRPS*, pp. 259-265, 2001.
- [9] F. Ricci, et al., "A 1.5 GHz 90nm Embedded Microprocessor Core", *Symp. VLSI Circ. Tech. Dig.*, pp. 12-15, 2005.
- [10] P. Calingaert, "Two-Dimensional Parity Checking", *J. ACM*, 8, 2, pp. 186-200, 1961.
- [11] M. Hsiao, "A Class of Optimal Minimum Odd-weight-column SEC-DEC Codes", *IBM J. Res. Dev.*, 14, pp. 395-401, July 1970.
- [12] C. Chen, M. Hsiao, "Error-Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review", *IBM J. Res. & Dev.*, 28, 2, pp. 124-134, Mar. 1984.
- [13] L. Clark, et al., "An Embedded Microprocessor Core for High Performance and Low Power Applications", *IEEE J. Solid-state Circuits*, 36, pp. 1599-1608, Nov. 2001.
- [14] J. Maiz, S. Hareland, K. Zhang, P. Armstrong, "Characterization of Multi-bit soft Error Events in Advanced SRAMs", *IEDM Tech. Dig.*, pp. 945-948, Dec., 2004.