Protecting Caches Against Multi-Bit Errors Using Embedded Erasure Coding

Abbas BanaiyanMofrad*, Mojtaba Ebrahimi†, Fabian Oboril‡, Mehdi B. Tahoori† and Nikil Dutt*
* University of California, Irvine, CA 92697, USA
Email: {abaniya, dutt}@uci.edu
† Karlsruhe Institute of Technology, Karlsruhe, Germany
Email: {mojtaba.ebrahimi, fabian.oboril, mehdi.tahoorig}@kit.edu

Abstract—Technology scaling advancement coupled with operational and environmental effects make embedded memories more vulnerable to both manufacturing and transient errors including multi-bit upsets. Conventional error correcting codes incur high latency, area, and power overheads to correct multi-bit errors. In this paper, we propose embedded erasure coding (EEC), a low-cost technique that can correct multi-bit upsets with low overheads. It employs interleaved parity bits to provide a fast and low-cost multi-bit error detection. Using the erasure coding concept, the error correction is done by reconstructing the contents of the erroneous cache blocks within each cache set. It trades the performance for higher reliability by reserving a part of the cache (e.g. one way) to store the erasure codes. Our simulation results show that EEC provides high reliability (100% error detection and correction) with a faster error recovery with lower area overhead and less than 3% performance loss as compared to other state-of-the-art techniques.

I. INTRODUCTION

Exponentially growing number of devices per chip along with a gradual reduction in operating voltage lead to an increase in the effective Soft Error Rate (SER) of circuits over the past few years [1, 2]. Additionally, it has been shown that the SRAM arrays are the major contributor to the overall SER (by more than 95%) [3], hence, applying an effective technique to mitigate the errors in these components is of decisive importance. In previous technology nodes, a soft error caused by a single particle strike in an SRAM array was very likely to only affect a single cell. This kind of errors are called Single Event Upsets (SEUs) and could be corrected by a low-cost single error correction technique [2, 4]. With smaller device geometries in nanoscale technology nodes, a single high energy particle strike might affect several adjacent cells resulting in Multiple Bit Upsets (MBUs). Recent experiments show that not only the ratio of MBUs to SEUs increases, but also the average number of affected cells by an MBU grows [1, 5]. Therefore, an effective MBU correction technique has to be used in memory arrays to ensure their reliability with minimum cost.

A variety of error-correction-codes (ECC) have been proposed to protect SRAM cache memories against soft errors [6, 7, 7–12]. However, the majority of previous MBU correction techniques are only able to correct a predefined number of errors per word. The area, latency, and power overheads of these techniques exponentially grow by increasing the MBU size. Moreover, due to high complexity and overhead of ECC encoding/decoding, it is hard to implement such techniques in current processors with affordable cost [7, 10]. Hence, to protect caches against MBUs, it is essential to reduce the complexity and overheads of the error protection techniques.

In this paper, we propose an error protection approach for cache memories using an erasure coding technique. Three main advantages of our approach over existing methods include:

- Simplicity for cost-efficient and easy implementation in current processors;
- Configurability to trade performance versus reliability by using more/less cache ways as error correction storage;
- Much lower error recovery time compared to similar techniques.

These features make it scalable to larger caches in multicore processors and also for protecting against permanent faults. For example, in adaptive low power systems where we aggressively over-scale the voltage to trade performance versus energy efficiency, it is extremely useful.

In this approach, considering the fact that error detection can be done at much lower cost than error correction [13], we employ interleaved parity codes per-word to detect errors in each cache word or block. The interleaving distance is set based on the distribution of actual MBU patterns and their respective probabilities obtained from a detailed technology dependent analysis. For error correction, we adapt the notion of traditional erasure coding to define an Embedded Erasure Coding (EEC) approach that performs coding across multiple cache blocks in the same row. Therefore, it reserves a cache way to store erasure check bits for the remaining ways. Moreover, it is implemented in a configurable way that depending on the desired performance/reliability, the last way in different parts (banks) of a cache can be used as erasure storage or normal data storage. We evaluate the proposed scheme for both L1 and L2 cache of a high-performance processor and also for L1 cache of an embedded processor. Our simulation results for L1 and L2 caches of the high-performance processor show that EEC detects and corrects 100% injected MBUs with less than 3% performance and 4% energy overheads. We also compare the reliability and area overhead of our scheme against common 1-dimensional and 2-dimensional error coding approaches. Our simulation results show that EEC provides higher reliability with a faster error recovery compared to the state-of-the-art techniques with less implementation cost including added circuitry and coding area overhead.
II. PRELIMINARIES AND RELATED WORK

This section provides a general background on MBUs, introduces erasure coding, and reviews related work on ECC schemes.

A. Multi-Bit Upsets

A realistic quantification of the proposed MBU correction technique requires detailed information on possible MBU patterns and their occurrence rate over the SRAM array. In order to achieve this, a 3D-TCAD simulation is performed using a commercial soft error evaluation tool [14]. In this experiment, the neutron-induced MBU patterns and their occurrence probabilities are obtained for a 45 nm low-power SRAM memory working in a terrestrial environment. The distribution of MBU sizes is demonstrated in Figure 1. As shown in this figure, almost half of the soft errors in the employed 45 nm SRAM memory are MBUs. It also depicts particular MBU patterns with high occurrence probabilities. The experiment shows that the largest MBU pattern affects 24 bits. Moreover, the ratio of MBUs as well as the largest MBU size will further increase in smaller technology nodes [1]. Hence, depending on the employed technology in SRAM memories, an effective coding technique with appropriate correction capability is required to be able to detect and correct the largest possible MBU incidents in the target memory.

B. Erasure Coding

At system level, erasure coding [15] has been very successful in disk error protection [16] as well as providing reliable data storage for the storage/distributed systems domain [17, 18]. A variety of erasure coding techniques with different properties such as recovery coverage, encoding/decoding complexity, and area overhead are proposed in the literature [15]. In general, an optimal erasure code is a coding technique which transforms $m$ blocks into $m+n$ blocks by adding $n$ redundant blocks. This transformation or coding is done in a way that in case of erasure (loss of data), the original blocks can be recovered from each possible set of arbitrary $m$ blocks among the $m+n$ coded blocks (see Figure 2). The recovery coverage of an erasure code is specified by the number of erasures that could be tolerated. For an optimal erasure code, the recovery coverage is equal to the number of redundant blocks, called erasure blocks (i.e. $n$). In general, the area overhead of an erasure code is defined as the ratio of redundant blocks to original data blocks which equals to $n/m$ for the optimal erasure codes.

In general, erasure codes are designed to recover the original blocks when a subset of blocks is not available (i.e. erased). Although this coding approach is not designed to detect or correct errors, once an error is localized in particular blocks using a detection technique, the original blocks could be reconstructed using an erasure code in a fast way.

C. Related Work

A wide range of ECC techniques have been used to protect memories [13]. Single-error correction double-error detection (SECDED) is proven as an effective mechanism for handling soft errors [6]. However, in a high-failure rate situation, most simple coding schemes like SECDED are not practical because of the strict bound on the number of tolerable faults in each protected data chunk [7]. As technology advanced, a variety of more complex coding schemes with higher level of protection have been proposed to protect against multi-bit errors [8, 9, 19, 20]. However, using strong ECC incurs a high overhead in terms of storage for the correction code, large latency, slow and complex decoding [7, 10]. Also, some approaches use a bit-interleaved memory with SECDED ECC to correct small-scale physically-contiguous multi-bit errors [21, 22]. However, power, area, and delay overheads grow significantly as the interleaving factor increases beyond four, depending on the memory design [23].

Typically, ECC schemes used in caches and memory rely on codes that detect and correct errors since the positions of the errors are not known a priori. On the other hand, Erasure coding is a well-known technique for correcting errors when the position of an error is known [15]. By separating error detection from error correction in common error coding approaches used in memory arrays, erasure codes can be leveraged to recover erroneous parts as an alternative to common error correction technique. The main advantages of erasure coding, compared to common multi-bit ECC techniques, are its low complexity, simple implementation, and high coverage [15]. However, it requires more code storage than conventional ECCs. Some schemes inspired by erasure coding have been recently proposed to protect cache memories against soft errors. A two-dimensional coding approach in which, the horizontal parity detects errors and the vertical parity corrects them, is presented in [7]. In CPPC[11], two registers are added to the cache to store the XOR of all data written to the cache and all dirty data that evicted from it, respectively. In [12], the proposed scheme extends SECDED to correct one more faulty bit using erasure coding concept. Note that both 2D error coding [7] and CPPC [11] approaches, incur a lot of area and power overhead due to over-design for correcting up to 8x8 clustered MBUs. Also, the error recovery in both approaches is a time-consuming and complex process which needs to access multiple cache rows.

In contrast with all previous approaches, we first perform detailed technology dependent analysis to extract MBU patterns and their respective probabilities. Then, based on our
observation we try to minimize the overheads of MBU protection by optimizing the interleaving distance of parity checking and making the error recovery process less complicated and faster compared to similar approaches to reduce system-level costs in case of high error rates.

III. EMBEDDED ERASURE CODING (EEC)

To enable correcting a larger ratio of MBUs with affordable complexity and latency overhead, we propose the EEC approach. The main idea behind this work is to use the concept of erasure coding for error correction by reconstructing the contents of the erroneous cache blocks within each cache set. The light-weight interleaved per-word parity codes are applied for error detection. For error correction, we adapt the notion of traditional erasure coding to define a technique that performs coding across multiple cache ways in a set-associative cache. It reserves a cache way to store erasure check bits for the remaining ways. In this technique, we trade off cache capacity for a low-cost multi-bit error protection with fast error recovery without any significant hardware modification and overhead. Moreover, it is implemented in a configurable way that depending on the desired level of performance/reliability, the reserved way in different parts (banks) of the cache can be considered as either erasure storage or normal data storage. In contrast to similar approaches [7, 11, 12], our proposed scheme does not require any significant change to the existing cache architecture of processors to add redundant rows.

A. Parity-based Erasure Coding

There a variety of erasure codes with different attributes in the literature. In the context of on-chip memories, an appropriate erasure code should be selected that meets the memory constraints. Since the soft error occurrence rate is relatively small and read/write operations in cache arrays are continuously performed during the runtime, it is less likely to have more than one erroneous cache block per cache row at the same time [24]. On the other hand, the encoding time of the erasure code is critical in our design as it affects the write delay. Also a high decoding latency delays the error recovery process. Thus, depending on the granularity of error coding and the number of erasure blocks, we have multiple trade-offs between area, reliability, and performance.

In this work, we consider a parity-based erasure code [15] with only one redundant (parity) block that comes with short decoding and encoding latencies which satisfy our requirements. The parity based erasure code is an optimal erasure code which has one redundant block \( n = 1; m + 1 \) erasure code). In this case, it can recover the erased (erroneous) block from all cases with only one erasure. Figure 3 represents a sample erasure code with \( m = 4 \). As depicted in this figure, the redundant block contains the parity of data stored in all other blocks. In case of an erasure, the redundant block can be used as the parity of other \( m \) blocks to recover the contents of the erased block. Note that the factor \( m \) determines the trade-off between the encoding/decoding time and the area overhead. In this work, for a \( k \)-way set-associative cache, \( m \) is equal to \( k - 1 \).

B. Parity-based MBU detection

We employ a light-weight bit-interleaved parity code for the error detection. We consider \( n \)-way interleaved parity bits per word which is computed by XORing the bits whose distance is \( n \) (e.g., for \( n=8 \) Parity_bit[i] = data_bit[i] \( \oplus \) data_bit[i+8] \( \oplus \) data_bit[i+16] \( \oplus \) ...). In this way, we can detect up to contiguous \( n \)-bit MBUs in each cache word. The optimal value of \( n \) will be extracted based on some technology dependent MBU pattern and reliability analysis (See Section IV.C).

Figure 4 depicts the architecture of a 4-way set-associative L1 cache equipped with EEC and the process for updating the erasure codes on writes. It keeps an 8-way interleaved parity per block using 8 parity bits per word. It also reserves the last way of the data array for storage of the erasure codes. Note that the latency and area overhead of this parity coding is similar to byte-parity coding (common in timing-critical L1 caches) as well as SECDED coding [7].

Next, we first explain the operation of EEC scheme in both an error-free case and in the presence of errors. Then, we discuss its implementation issues in different architectures.

C. EEC Operation

During normal operation (no error is detected), every cache read checks the parity bits associated with the accessed word. On every write, parity and erasure codes are updated. As depicted in Figure 4, the update of the erasure codes is a three-step process that requires first reading the old data and erasure code, XORing them with the new data, and then writing both the new data and updated erasure code back to the cache. Thus, every write operation is converted to a "read-before-write" operation to read the old data before writing the new data. Note that these three steps are pipelined and the update of the erasure code is done either along with cache data write or after that depending on the cache architecture. Note that depending on the architecture of the cache (i.e. sequential or parallel) and the number of cache ports, this operation can be done as fast as a single cache access and as slow as four cache accesses.

As shown in Figure 4, the L1 cache has a fast and parallel access, so both data and erasure code can be accessed in parallel. But, in case of caches with a sequential architecture, we need to first read the old data, then read the erasure data during the next access. In this case, the XOR operation (step 2 in the figure), is done in two steps and an additional register is required to keep the intermediate value. Finally, it writes back the new data and updated erasure code consequently. So to have the minimum impact on performance, we apply this approach to fast L1 caches with parallel access and/or lower
Fig. 4. A 4-way L1 cache architecture protected with EEC that shows the updated write operation in three steps.

level caches with sequential access.

In case the processor has two separate read and write ports, those read and write operations can be done in parallel. However, this may cause some read port contention in out-of-order processors that leads to an increased load latency. This problem can be addressed to improve the performance penalty by using different techniques such as cache port stealing [25] or cycle-stealing [11]. For in-order processors with a single port, the extra read increases the cache write latency. However, there exist some approaches such as port stealing [25] or thread scheduling [26] to hide the L1 cache extra access latency.

D. Error Recovery

When the parity bits detect an error, the cache controller initiates the recovery process. The controller reads all data blocks in the set of accessed block including the erasure block. Then, it XORs the value of all blocks except the block that contains the erroneous word. The result of the XOR operation is the correct value of the faulty block, which is then written back to the accessed block. Note that the recovery operation requires accessing every cache block in the set. In fast L1 caches, since all blocks of the set are accessed in parallel, the recovery is very fast. But L2 caches usually have a sequential architecture in which tag checking is done first and then only one block is accessed. In this case, the cache controller needs to access the blocks of each set one by one that makes the latency of error recovery up to the latency of accessing all the blocks in the set. During each cache write-back, the controller also checks the dirty bit of the block. If the evicted block is clean, it does not call the recovery process.

E. Reconfigurability

In contrast to conventional erasure coding approaches, EEC does not need to add extra resources as redundant (erasure) blocks (mentioned in Section II.B). Instead, by changing the configuration of the cache, EEC can reserve a part of the cache (i.e., one or multiple ways) as redundancy. The primary benefit of EEC reconfigurability is that cache capacity could easily be reduced to suit the required erasure blocks, while only minor modifications to the cache controller and/or software layers are needed. So reliability can be improved with small performance impact. Hence, we add an extra bit to the tag bits of each row which indicates whether that row reserved a block for erasure codes.

IV. EVALUATION

In this section we present different sets of results to demonstrate the evaluation of three major contributions of our approach — simplicity, configurability, and lower error recovery time — mentioned in Section I, compared to similar approaches.

A. RTL Implementation

We implemented the EEC approach by modifying the HDL source code of Leon3 processor [27]. Leon3 is a simple embedded in-order processor that has only L1 data and instruction caches with the same read and write latency of 1 cycle. We modified its data cache controller and cache array modules to configure the last way for erasure code storage. We also modified the cache controller to implement “read-before-write” operation. Since Leon3 L1 cache has only one data port, we needed to modify the cache controller to perform extra read before each write operation. Also, since Leon3 has a single cycle Load/store operation, we considered one extra cycle to update the erasure code and write it back. For error recovery, since Leon3 has a fast cache, all the blocks are accessed in parallel. Hence, it takes only two cycles to recover the erroneous block. Table I shows the details of the simulated processor. We simulated the modified Leon3 processor by running multiple benchmarks including stringsearch, basicmath, bitcount, and qsort. The execution time results showed no more than 1% runtime overhead. We also synthesized it using Nandgate 45 ns technology. The synthesis results showed less than 5.8 % logic area overhead and increased latency of 0.1 ns.

B. Cycle-accurate Simulation

We used the gem5 cycle-accurate simulator [28] to model EEC for a single-core high-performance processor. We added EEC to both the L1 data and L2 cache of Alpha processor which is an out-of-order processor running at 3 GHz. In this processor, for both cache levels, read and write latency are independent and can use different values. Thus, if our proposed EEC technique is employed, the read access latency remains the same as normal value, while the write latency is doubled (due to the additional read operation for every write access).
In case of L1, we considered a fast cache in which both tag check and data access are done in parallel. So all data blocks including erasure check bits are read in parallel and the error recovery latency is equal to two cache accesses. But L2 cache has a sequential architecture, so the tag checking and data access are done separately. In this case, the cache blocks inside a set can be accessed one by one. So the write latency is equal to the latency of two writes and two reads. However, the rate of write to L2 is much lower than L1 that has a little impact on performance and dynamic energy overheads. Also, the error recovery is more longer and is equal to latency of one L2 cache access times degree of associativity. However, it is still much faster than similar approaches that access multiple cache rows. We added a block-size register to L2 to keep the intermediate data due to XOR of different cache blocks. We selected six benchmarks from SPEC2000 suite to evaluate the proposed EEC approach on Alpha processor. The benchmarks are fast-forwarded for one billion instructions as warm-up, then simulated for five billion instructions. Table I shows the details of the simulation systems.

C. Performance and Energy Analysis

Since our proposed technique not only reduces the number of available cache ways by one, but also increases the cache write latency, it affects the processor performance, as shown in Figure 5. However, as only the latencies for the write accesses are affected and not those for read operations, the overall performance impact of our proposed technique is very small. This is due to the fact that the number of writes are much lower than the reads and not in the critical path of the application. Also, EEC trades cache capacity and associativity which mainly affect the miss rate in highly memory-intensive applications. On average, over all benchmarks the performance is only reduced by less than 2% on the L1-Cache and less than 3% on L2-Cache of the Alpha microprocessor simulated by gem5.

![Fig. 5. Performance overhead of EEC for L1 and L2 caches on Alpha processor](image-url)

Based on these results, we also evaluated the energy impact of the EEC technique for the Alpha processor. Therefore, we employed CACTI [29] to extract the per-access energy for read and write operations as well as leakage power. Together with the gem5 statistics about the number of cache accesses and runtime we obtained the overall energy consumption of the microprocessor with and without EEC. Here, similar to other approaches, we consider a cache equipped with SECDED as the baseline. Since all the overheads are normalized to the baseline, the major contributors to the power and performance overheads are due to the extra read access for L1 and read/write for L2. According to this evaluation, the dynamic energy consumption increases by 3.9% if EEC is employed for the L1-Cache. 1.7% more energy is consumed if the L2-Cache uses EEC. In both cases, the reason for the increase in energy consumption is the fact that EEC adds a read operation for each write access in L1 and two reads and one more write for L2. However, as leakage is dominant in nowadays technology nodes and the ratio of write operations compared to the total number of cache accesses is just around 20%, the increase in write energy has just a small impact on the overall energy consumption. Here, we do not consider the leakage power of erasure data storage as explicit overhead, because our approach does not impose extra resources to the system and the cache ways that used for erasure data directly affect the performance of the system. However, if we consider the reduced capacity cache with reserved ways as the baseline, in that case we may have up to 25% leakage power overhead for L1 and 12.5% for L2.

D. Comparison to other approaches

Here we compare different approaches including proposed EEC, 2D parity cache [7], and SECDED against MBUs considering the actual MBU patterns of the employed 45 nm SRAM technology. In this regard, we injected one million MBUs in a SRAM array equipped with these coding techniques and compute the error detection and correction probabilities. When an MBU spans across multiple words, it is considered as detected/corrected if all affected words are detected/corrected by the employed coding technique. Table II shows the results of this experiment. 2D parity cache (i) means that this technique is equipped with i horizontal and i vertical interleaved parity bits. Also, EEC(j) indicates that j horizontal interleaved parity bits are used for error detection. As it can be seen in this table, all errors detected by EEC are successfully corrected by this technique, while some of the detected errors by 2D parity cache cannot be corrected as they affect more than one word affiliated with one redundant vertical parity line. On the other hand, the amount of Detected Unrecoverable Errors (DUE) in EEC is independent of the number of parity bits and is always zero. However, in case of 2D parity cache, it depends on the number of parity bits used and can be up to 17%.

In the last column, we compare the coding area which is defined as the amount of extra code bits that is used in each scheme for the corresponding level of protection. As can be observed from the table, the coding area overhead of EEC is less than 2D coding in all cases and less than SECDED in first four cases. Note that the amount of code storage in EEC, depending on the cache associativity, EEC configuration, and level of protection varies. For example, in a 4-way cache with EEC setting of fully block protection, can be as high as 25%. Since we trade the cache capacity instead of adding any extra
elements (spare cache/rows) to prepare the code storage, this is an implicit area overhead with much less cost compared to similar approaches that impose explicit costly overhead to the system.

### E. Applicability to Permanent Faults

EEC’s features such as reconfigurability, short recovery time, and ability to correct sparse errors, give the ability to EEC that can also be easily applied for protecting cache memories against permanent faults. Here, we consider the high bit rate faults due to process variation in caches working at near-threshold operation voltages. Using our analytical models and the derived fault model and bit error rates similar to [30], we were able to compare our approach with similar approaches such as 2D error coding. Our analytical results show that while 2D error coding can correct bit error rate of $10^{-5}$, ECC corrects up to $10^{-3}$ without degrading the yield. Moreover, the ECC error correction is much faster than 2D coding (i.e., about ten times) which minimize the impact on performance and dynamic energy. Due to the lack of space, we consider any further discussion and analysis in this regard with more details as future work.

### TABLE II. RELIABILITY AND CODING AREA COMPARISON OF DIFFERENT TECHNIQUES

<table>
<thead>
<tr>
<th>Technique</th>
<th>Detection</th>
<th>Correction</th>
<th>DUE</th>
<th>Coding Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEED/ED</td>
<td>97.34%</td>
<td>65.95%</td>
<td>17.10%</td>
<td>12.5%</td>
</tr>
<tr>
<td>2D Parity Cache (1)</td>
<td>91.43%</td>
<td>52.33%</td>
<td>17.10%</td>
<td>3.22%</td>
</tr>
<tr>
<td>2D Parity Cache (2)</td>
<td>91.43%</td>
<td>78.27%</td>
<td>13.16%</td>
<td>6.44%</td>
</tr>
<tr>
<td>2D Parity Cache (3)</td>
<td>99.68%</td>
<td>97.37%</td>
<td>2.31%</td>
<td>9.67%</td>
</tr>
<tr>
<td>2D Parity Cache (4)</td>
<td>99.98%</td>
<td>99.37%</td>
<td>0.61%</td>
<td>12.89%</td>
</tr>
<tr>
<td>2D Parity Cache (5)</td>
<td>99.99%</td>
<td>99.99%</td>
<td>0.3%</td>
<td>16.11%</td>
</tr>
<tr>
<td>2D Parity Cache (6)</td>
<td>100.00%</td>
<td>99.98%</td>
<td>0.02%</td>
<td>19.34%</td>
</tr>
<tr>
<td>2D Parity Cache (7)</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0</td>
<td>22.56%</td>
</tr>
<tr>
<td>ECC (1)</td>
<td>69.23%</td>
<td>69.23%</td>
<td>0</td>
<td>3.12%</td>
</tr>
<tr>
<td>ECC (2)</td>
<td>91.43%</td>
<td>91.43%</td>
<td>0</td>
<td>6.25%</td>
</tr>
<tr>
<td>ECC (3)</td>
<td>99.68%</td>
<td>99.68%</td>
<td>0</td>
<td>9.37%</td>
</tr>
<tr>
<td>ECC (4)</td>
<td>99.98%</td>
<td>99.98%</td>
<td>0</td>
<td>12.5%</td>
</tr>
<tr>
<td>ECC (5)</td>
<td>99.99%</td>
<td>99.99%</td>
<td>0</td>
<td>15.62%</td>
</tr>
<tr>
<td>ECC (6)</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0</td>
<td>18.73%</td>
</tr>
</tbody>
</table>

### V. Conclusion

In this work, we presented embedded erasure coding to protect cache memories against multi-bit upsets at low latency and power overheads. It employs fast bit-interleaved parity coding as error detection in combination with optimal parity-based erasure coding as error correction over set-associative cache memories. This technique reserve a part or the whole of a cache way to store the redundant check bits. Its specific features including simple architecture, reconfigurability, and fast error recovery make it also an appropriate technique for protecting against permanent faults in voltage-scalable caches. We evaluated the proposed approach via RTL implementation as well as architecture-level simulation. Our experimental results on both L1 and L2 of a high-performance out-of-order processor showed that EEC provides same reliability with faster error recovery compared to similar approaches while incurs less than 3% performance and 4% dynamic energy overheads.

### REFERENCES


