Error Detection and Correction in Content Addressable Memories by Using Bloom Filters

Salvatore Pontarelli and Marco Ottavi, *Senior Member*, *IEEE*

Abstract—A content addressable memory (CAM) is an SRAM-based memory that can be accessed in parallel to search for a given search word, providing as a result the address of the matching data. Like conventional memories, a CAM can be affected by the occurrence of single event upsets (SEUs) that can alter the content of one of more memory cells causing different effects such as pseudo-HIT or pseudo-MISS events. It is well known that, because of the parallel search performed by a CAM during the query of a word, a standard error correction code could not defend it against SEU events. In this paper, we propose a method that does not require any modification to a CAM's internal structure and, therefore, can be easily applied at system level. Error detection is performed by using a probabilistic structure called "Bloom filter," which can signal if a given data is present in the CAM. Bloom filters permit to efficiently store and query the presence of data in a set. But, while a CAM suffers from SEU induced errors, the probabilistic nature of Bloom filters has as a consequence the so called false-positive effect. This paper shows that, by combining the use of a Bloom filter with a CAM, the complementary limitations of these modules can be compensated. The combined use of a CAM and a Bloom filter is analyzed in different cases, showing that the proposed technique can be implemented with a low penalty in terms of area and power consumption.

Index Terms—Bloom filter, content addressable memories, error detection and correction

1 Introduction

A content addressable memory (CAM) is an SRAM-based memory capable of comparing the input data against the data stored in the memory, providing as result the address of the matching data [14], [5]. CAMs with small dimensions are commonly used in cache or translation lookaside buffers (TLB), [15] while large CAMs are used in systems that must perform rapid searches within a large amount of data. Nowadays, one of the most used applications where CAMs are used is packet forwarding and classification in high-speed network systems [16]. A particular kind of CAM-like structure is the array realizing the tag field of a cache memory. When a processor needs to read or write a location in the main memory, first it checks whether that memory location is in the cache. This is accomplished by comparing the address of the memory location to all tags in the cache that might contain that address. n-way associative caches are commonly used to simplify the cache architecture and to limit the power consumption of the tag search. With an n-way set associative cache a memory location can be stored only in a subset of n locations of the cache. When n corresponds to the total number of cache rows, the cache is called fully associative, while if each entry in main memory can go in just one place in the cache, the cache is directly mapped.

Currently, the use of nanometric scale technology, and the increase in the overall number of stored bits have caused a consequent increase in the error rate due to the occurrence of single event upsets (SEUs). SEUs occur because of particles striking a sensitive area of a circuit. The interaction between silicon and particles creates free charges that can be collected by the sensitive circuit nodes close to the location of the particle impact. The collected charge can change the state of a circuit, for example, by flipping the value of a bit from 0 to 1 or *vice versa*.

These effects are well known for SRAM and DRAM memories, and many strategies have already been proposed to mitigate the effects of SEUs in memories, based on information redundancy or on technology/circuit level solutions. Information redundancy usually has been exploited by using error-detection and correction codes, while technology and circuit solutions are aimed at increasing the critical charge value [1].

However, these techniques are not well suited to be directly applied to a CAM, and therefore new approaches to mitigate SEU effects in CAM should be developed to use large CAMs in complex systems while ensuring high levels of reliability. In the literature different techniques have been proposed to enhance robustness against SEUs in CAM. Almost all the proposed techniques require modifications to the CAM architecture, performed at circuit and/or at architectural level.

The effects of an SEU on a memory device closely relate to the technology node at which the device is realized. While until few years ago an SEU on a memory corresponded to a single-bit upset (SBU), in a memory realized with feature size less than 90 nm a single particle can change the value of multiple bits [17], [19]. This kind of effect is commonly known as multibit upset (MBU). In the remainder of this paper, we use SEU to refer to a generic radiation induced error, while we use the SBU and MBU terms to refer to errors affecting one or more than one bit,

• *The authors are with the University of Rome "Tor Vergata" Via del Politecnico 1, 00133 Rome, Italy.*
E-mail: {pontarelli, ottavi}@ing.uniroma2.it.

Manuscript received 20 Dec. 2011; revised 8 Feb. 2012; accepted 15 Feb. 2012; published online 21 Feb. 2012.
Recommended for acceptance by J. Xue.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2011-12-0963.
Digital Object Identifier no. 10.1109/TC.2012.56.

respectively. In particular, we refer to an l-bit MBU for an SEU affecting up to l bits in the same word. Note that an l-bit MBU can change up to l bits stored in consecutive locations, but can also change only a subset of the bits stored in these locations, depending on the values stored previous to the occurrence of the particle impact.

While error-correcting codes (ECCs) can detect and correct errors in SRAM and DRAM, and can be easily extended to MBU (see, e.g., [18], [2]) they cannot directly be used in associative memories such as CAMs or caches. For example, suppose that an error occurs in an entry of a CAM, if that entry is queried, the CAM will respond that the entry is not present, without giving any information about the original value of that entry. Therefore, a standard CAM search operation cannot detect and correct a corrupted codeword.

The use of ECC in a cache is usually applied to protect the cache data, while for the tag, the only protection is given using a parity bit, which is unable to perform error correction. An error affecting a tag of the cache usually causes a miss event when that tag is queried, causing the processor to fetch data from a higher level cache or from the external memory.

We propose a method that does not require modifications to the internal structure of the CAM, since this is often designed in a full custom design flow to minimize power and area consumption of the core cell, therefore it is preferable to add error-detection and correction features without compromising the internal structure and, therefore, the overall performance of the circuit.

Finally, while the problem of SEU in cache memories has been discussed previously, the aim of this paper is to focus on a generic CAM device that differently from cache memories is inherently fully associative. The full associativity has consequences both on power consumption of these devices and on the applicability of other techniques for error detection and correction.

The underlying idea of this paper is to add in parallel to the CAM a well-know data structure, called Bloom filter, to efficiently detect if the CAM has provided a correct result or if it is affected by an error. A Bloom filter is a structure that can be realized efficiently with limited hardware resources, or with efficient software algorithms.

In a Bloom filter when data has to be stored (or queried) it is hashed with multiple hash functions, and at the output of each hash a corresponding memory location is written (read). A Bloom filter performs a limited number of memory accesses, one for each hash output, and therefore its memory can be easily protected against SEU by using a standard ECC. This characteristic permits to design a system without considering the possible SEU occurrence in the Bloom filter memory, thus supposing that the filter memory can be efficiently protected by ECC. A Bloom filter performs two tasks: 1) stores a set of items in its memory, and 2) quickly responds to a query about the presence of an item. The drawback of using such a structure lies in its probabilistic nature that yields to the so-called false-positive effect. A false positive occurs because, due to the aliasing given by the hash functions, a query performed on the Bloom filter has a certain probability, (called false-positive rate), of producing an erroneous result, i.e., signaling a data as present. However, when a Bloom filter signals that a data is not present, this will always be true (i.e., a false negative never occurs in a Bloom filter).

Therefore, an architecture using both a CAM and a Bloom filter could potentially be affected by two very different (and opposed) undesirable effects are as follows:

1. The CAM could give a wrong answer due to the occurrence of an SEU.
2. The Bloom filter could give a wrong answer due to a hash collision.

It will be shown that these two effects are complementary and that can be used for mutual benefit, i.e., on one side the CAM can detect a false positive occurring in the Bloom filter, while on the other side the Bloom filter, (made resistant to SEU by using a standard ECC), can detect SEU induced errors occurring into the CAM.

This paper also proposes a suitable algorithm, similar to the one presented in [10], that allows correcting an error in the CAM after its detection by comparing data stored in the CAM with those stored in the Bloom filter. Finally, to manage the dynamic behavior of a CAM that usually deletes and update its content, a well-known extension of the Bloom filter, called counting Bloom filters [33] is applied. Experiments performed on the realized system by using a cacti-based model [34] show that the technique proposed in this paper introduces an area overhead ranging from 10 to 50 percent and a 20-30 percent additional power consumption. Moreover, the proposed solution is particularly suited for CAM with wide word sizes since the overhead is independent on the CAM word size. Simulation experiments have also been performed on the caches and TLBs of a microprocesss protected by our proposed method, to show its effectiveness.

The remainder of the paper is structured as follows: Section 2 presents a survey of previously proposed methods for protecting CAM against the occurrence of SEU. Section 3 discusses the basic properties of a CAM, and shows the effects of an SEU hitting a CAM. Section 4 gives a background on Bloom filters, while Section 5 presents the proposed architecture of a CAM with error-detection and error-correction capabilities. Section 6 discusses the sizing of the parameters of the block introduced in Section 5 (the counting Bloom filter and the auxiliary CAM) while Section 7 discusses the resource evaluation. Section 8 presents the experiments related to the application of our method for protecting caches and TLBs of a microprocessor. Finally, Section 9 draws the conclusions.

2 Related Work

A discussion on the effects of SEU in CAM devices has been presented in [27], [28], and [24]. The presence of an error in these devices can give different types of incorrect responses that have been classified like pseudo-HIT or pseudo-MISS events. When a word value becomes incorrect due to the occurrence of an SEU, if a query looks for the original value the response will be an incorrect miss, while if a query looks for the erroneous value the response will be an incorrect hit. To protect these memories against the SEU, different methods have been developed. Here, a literature survey is

proposed starting from the methods that modify the CAM cell at circuit level, to the methods that exploit ECC, up to system level methodologies.

An example of CAM reinforced at circuit level against SEU is given in [7], where a set of core cell of the CAM array based on feedback or cross-coupled inverters that strengthen the cell against SEU events are proposed. Instead, in [20], the use of DRAM instead of SRAM has been proposed, exploiting the assumption that DRAM are less susceptible to SEU than SRAM. Salice et al. [9] propose a methodology to produce a CAM structural architecture starting from a functional description of some high-level properties of the device. In [21], the content of the CAM is continuously refreshed by an associated DRAM with ECC features to scrub the memory recovering the CAM from errors due to SEU. In [22], the words stored in a CAM are protected against SEUs by utilizing one or more parity bits, and the SEU induced errors detection and correction is demanded to a modified encoder block that effectively works also as an embedded error-correction block based on Hamming codes. This encoder, therefore, requires a number of several cascaded XOR gates (see also [29]), which degrade area occupancy and most of all the timing performances with respect to a nonprotected CAM. Moreover, the solution proposed in [22] for match-line sense amplifiers although quite interesting, could be affected by issues both related to power consumption and noise immunity.

Krishnan et al. [8] substitute the sense amplifier at the end of a match line with a comparator to signal a match even if some ternary bits mismatch and then adds a suitable error-correction code for ternary CAM (called TECC). The techniques proposed in these papers to prevent SEU induced errors use a circuit level approach that requires changes in the internal structure of the CAM, and consequently a redesign of the entire chip.

Because of the widespread usage of cache memories, many works focus on cache memories as for example [23], [25], [26]. The methods presented in [25] and [26] are developed for a specific cache level , ([25, L1], [26, L2]), and suppose that the used caches have a limited associativity. In particular, in [25], a four-way associative cache is used, while in [26] data are provided for an eight-way set associative cache. The methods are developed to work well under the assumption of data locality, as it is commonly the case for microprocessor caches. This assumption cannot hold when the CAM is used for high speed look-up or for networking systems. Moreover, all the methods described in [23], [25], and [26] are based on the use of an additional (small) fully associative cache in parallel to the main n-way associative cache. The fully associative cache can be maintained small exactly because the main cache has a limited associativity, but an extension of these methods to protect a generic full associative CAM is not trivial and the achievable results cannot be foreseen.

Another solution using a redundant CAM for error detection and correction has been proposed in [30]. However, the use of a duplicated CAM in [30] has a serious drawback in power consumption, due to the power hungry nature on these devices. Both the methods proposed in [25] and [26] and the one described in [30] do not require any modification to the CAM internal structure, and, therefore, can be applied at a system level, using suitable redundant components to obtain error-detection and correction capabilities. In this paper, we present a solution that modifies the architecture proposed in [30] to minimize the resource occupation and the power consumption of the architecture described in [30]. Our solution is based on the use of a probabilistic structure called "Bloom filter." The duplicated CAM is substituted by the Bloom filter and a store/query operation to the CAM is given in parallel also to the Bloom filter. The combined use of CAM and Bloom filters has been proposed in [11], where the performances of content addressable memory aided hash table are evaluated. However, Wan et al. [11] propose to combine CAM and Bloom filters only for performance enhancement, and not against the occurrence of Soft errors. Finally, Bloom filters have been proposed to reduce latency [12] and power consumption [13] in cache memories.

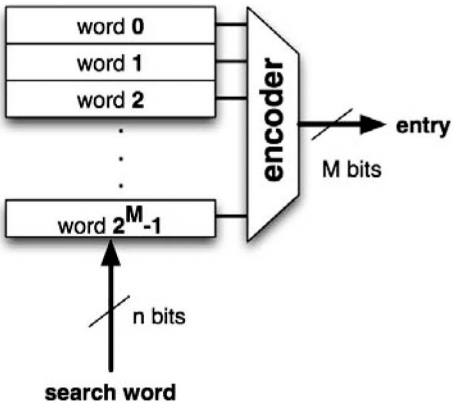


Fig. 1. Scheme of a CAM.

3 Architecture of Content Addressable Memories and Consequences of a Soft Error in CAM

3.1 Architecture of Content Addressable Memories

Fig. 1 shows the schematic of a CAM. The input search word is an n-bit string which is concurrently compared to all the $J = 2^M$ words stored in the CAM. The number of bits of the search word (n) ranging from 36 to 144 bits is usually much larger than M that usually ranges from 7 to 15 bits [14]. The memory array of a CAM has a structure that is similar to a conventional SRAM, with an arrangement in rows and columns. From the operational point of view, the write operation inside the CAM is very similar to the write operation of a RAM. The data is written through the CAM bit-lines, while the word-line identifies which row of the array must be written by the data driving the bit-lines. Instead, the specific CAM functionality of searching a data inside the memory is carried out in parallel by exploiting the suitable additional circuitry that is not present in the SRAM array. From the SEU susceptibility point of view, the core cell of a CAM is quite similar to the conventional SRAM cell, and a particle hitting the CAM produces similar consequence on the bits stored in the array.

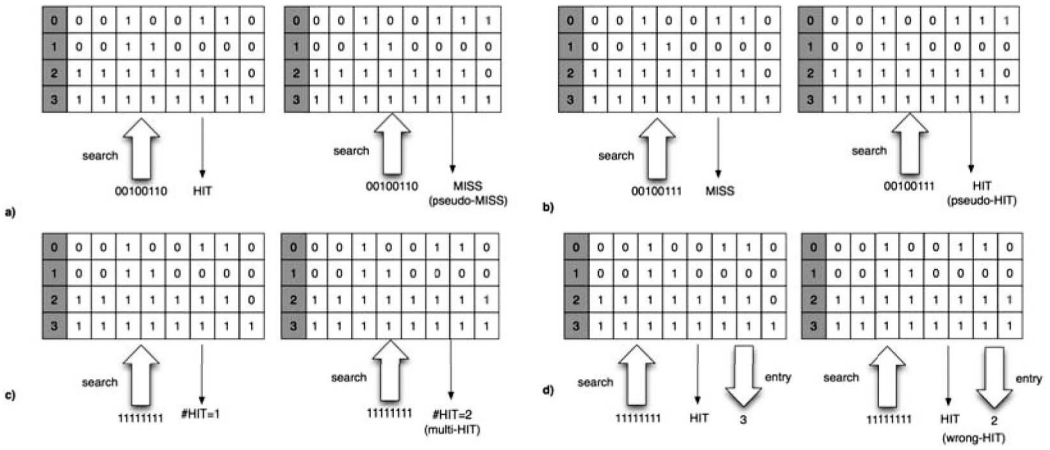


Fig. 2. Possible errors occurring in a CAM. Original and corrupted CAMs are presented side by side. In (a), the SEU hits a bits of entry 0 of the CAM. When the word is requested the CAM responds with a pseudo-MISS signal. In (b), the same SEU also produces a pseudo-HIT. In (c), a multi-HIT error is caused by the SEU affecting a bit of entry 2. In (d), the same error produces a wrong-HIT.

When the same data is stored in different location of a CAM, and this data is queried, then two different policies can be applied. The most used policy uses a priority encoder designed to establish a priority in the encoded output thus providing only one output [4]. For instance, the priority encoder provides as an output the matching word with the lowest (or highest) value: We call this $resolved$ multimatch approach. A second policy requires the introduction of a signal called N_{match} that provides the number of matched lines [6]; on the encoder output the matched lines will be provided as an output one per clock cycle: We call this $unresolved$ multimatch approach.

3.2 Consequences of a Soft Error in a CAM

A bit flip occurring in the memory composing a CAM can have different effects depending both on the location of the SEU and on the functional usage of the CAM. In this section, we discuss the effects of soft errors, by isolating four possible cases:

1. *pseudo-MISS.* An SEU changes the content of the memory in a certain location, therefore when that content is searched, the CAM does not provide a match. An example is provided in Fig. 2a. An SEU hits the entry 0 of the CAM changing its content from 00100110 to 00100111. When the word 00100110 if requested, the CAM will respond with a miss signal.
2. *pseudo-HIT.* A corrupted memory content corresponds to another content. If this word is searched, the CAM gives as response the location in which the error has occurred. With the same error condition of the previous case, if the word 00100111 is requested, the CAM will respond giving the entry 0 as an output. This situation is shown in Fig. 2b. It should be noted that the same SEU can, therefore, produce both a pseudo-MISS and a pseudo-HIT effect.
3. *multi-HIT.* If the word changed by a bit flip assumes the same value stored in another entry of the CAM, a multi-HIT error occurs. The effects of a multi-HIT error also depend on the kind of policy applied in case of a multiple match. If a priority encoding ($resolved$ multiple matching) is used, the outcome of a multi-HIT error could be masked if the priority of the correct match is higher than the priority of the wrong one. Instead, if the CAM uses the $unresolved$ multiple matching policy by providing all of the matched records in subsequent clock cycles, then in case of a multi-HIT error, the number N_{match} of matched output will be increased of 1 with respect to the correct value. This case is presented in Fig. 2c.
4. *wrong-HIT.* This error occurs only in case of multiple matching. Suppose to have words stored in k different entries and that one of these words is affected by an SEU. If the CAM uses the $unresolved$ multiple matching policy, in case of a wrong-miss error the number N_{match} of matched output lines will be $k-1$. Instead, if a priority encoding ($resolved$ multimatch) is used, the SEU produces an error if the location affected is the one with the highest priority (with the lowest value in our assumption), and the CAM responds with an erroneous entry that is an entry with priority lower than the correct one. An example for this error is presented in Fig. 2d. Note that when the SEU hits an entry that does not have the highest priority the error is inherently masked by the other entries with higher priority.

From the above description, it can be seen that different kinds of multiple matching policies, provide different behavior when an SEU occurs in the CAM memory.

4 Overview on Bloom Filters

In this section, we present an overview of the well-know Bloom filter [31] and counting Bloom filter [32], [33]. In particular, we investigate the effects related to the saturation of a counter in the counting Bloom filters and its effects on the overall false-positive probability of the filter.

A Bloom filter [31] is a probabilistic data structure used to check the membership of an element in a set. The structure allows the occurrence of false positives (i.e., the filter signals an element as present even if it is not true), but false negatives are not possible (i.e., if an element is present the filter will never signal the opposite). Elements can be

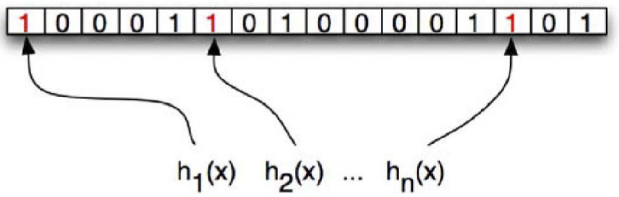


Fig. 3. Scheme of a Bloom filter. The data x to be stored is hashed n times and the bits addressed by the hashes are set.

added to the set, but not removed and the more elements are added to the set, the larger the probability of false positives. In this paragraph, we report the equations needed to correctly dimensioning the filter with respect to the required false-positive probability and the expected number of element to be stored. A Bloom filter is implemented as a bit array of m bits accessed via k hash functions $H_1(x) \dots H_k(x)$, each of which maps a set member x to one of the m bits within the bit array. We denote as $v(i)$ the value of bit i within the bit array.

Two operations are possible with a Bloom filter are as follows:

1. *Insertion.* An element x is inserted into the filter by setting to one all the indexes of the bit array addressed by the k hash functions. In a mathematical notation, this corresponds to
$$\forall i \in \{1..k\}, v(H_i(x)) \leftarrow 1.$$
2. *Querying.* An element is present in the filter if all the values of the bit array addressed by the k hash functions are equal to 1
$$result \leftarrow \min\{v(H_i(x))\},\ i \in \{1..k\}.$$

For a Bloom filter in which n elements are stored, the probability $\rho(n)$ that a given bit in the filter is zero is given by

$$\rho(n) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-n\frac{k}{m}}. \quad (1)$$

If we test membership of an element that is not in the set, each of the k bit array values indexed by the hash is 1 with probability $1 - \rho(n)$. The probability of all of them being 1, which would cause the false positive, is then

$$P_{fp}(n) = (1 - \rho(n))^k \approx (1 - e^{-\frac{kn}{m}})^k. \quad (2)$$

The probability of false positives decreases as m increases, and increases as n increases. For a given m and n, the value of k (the number of hash functions) that minimizes the probability is $k = m/n \ln 2 \approx 0.7 \cdot m/n$.

Using the optimal value of k we obtain

$$P_{fp}(n) = 2^{-k} \approx 0.61^{m/n}. \quad (3)$$

Using (3), we can size the Bloom filter according to a required number of elements to be stored and to a required minimum false-positive rate. In Fig. 3, the scheme of a Bloom filter is presented.

A *counting Bloom filter* [32] is a generalization of a Bloom filter that is implemented as an array of bins of m cells each containing b bits.[1] CBF can be used to implement multisets, which maintain a cardinality for each element within the set; in the case of CBF, the cardinality can be between 0 and $2^b - 1$. Differently from Bloom filters, CBF are able also to delete without incurring in false negative. The three operations of the CBF are as follows:

Increment (or insertion) of a bin for a set member x in a CBF consists of setting

$$\forall i \in \{1..k\}, v(H_i(x)) \leftarrow \min\{v(H_i(x)) + 1, 2^b - 1\}.$$

Decrement (or deletion) of a bin for a set member x in a CBF consists of setting

$$\forall i \in \{1..k\}, v(H_i(x)) \leftarrow \max\{v(H_i(x)) - 1, 0\}.$$

Querying of a set member x within a CBF is the same as in a Bloom filter.

Through the deletion operation, the counting Bloom filters preserve the characteristic absence of false negatives typical of Bloom filters until no overflow occurs on the counters. To explain this concept, consider the case in which a number of items $n > 2^b - 1$ saturates the ith counter. After 2^{b-1} deletions, the counter is set to zero, even if not all the n items corresponding to the ith counter have been evicted from the filter. A modified version of the decrement operation can be used to limit this behavior. The operation is modified as follows (see [32]):

$$\forall i \in \{1..k\}, v(H_i(x)) \leftarrow \begin{cases} \max\{v(H_i(x)) - 1, 0\}, & \text{if } v(H_i(x)) < 2^b - 1, \\ 2^b - 1, & \text{otherwise.} \end{cases}$$

With this modification if a saturated counter will not be decremented anymore. The CBF, therefore, keeps track of the items that have been stored but it will not decrement the counters that have been saturated. The effect of the presence of this "dirty" counters is the increase of the false-positive rate of the filter. In fact, if a number s of counters is saturated, it is like $n_s = \lceil s/k \rceil$ additional items have been stored in the filter. The false-positive probability of the counter can be, therefore, evaluated as

$$P_{fp}(n) \approx \left(1 - e^{-\frac{k(n+n_s)}{m}}\right)^k. \quad (4)$$

The increment in the false-positive probability given by n_s is negligible since usually n has a magnitude of 1,000 or more, while the magnitude of n_s is very small. In fact, the probability that a specific i counter in the array of counters is saturated, can be computed as [32]:

$$P_{sat}(i) = P(v(i) > 2^b) < (e \cdot ln(2)/2^b)^{2^b}. \quad (5)$$

And if $P_{sat}(i) \ll n$ the probability of having n_s saturated counters can be computed by using the binomial distribution

$$P(n_s) \approx \binom{n}{n_s} \cdot P_{sat}(i)^{n_s} \cdot (1 - P_{sat}(i))^{n-n_s}. \quad (6)$$

From (6), it can be seen that using 4 bits, the probability of having a saturated counter is less than 1.3E-9 for a CBF with n=1E6. Using 2 bits counters the probability of having $n_s > 10$ saturated counters is 2E-11. Therefore, in the

1. Note that a Bloom filter is simply a counting Bloom filter where $b = 1$.

remainder of this paper, we will assume $b = 2$ the number of bits of the CBF, and we will use (2) to compute the false-positive probability neglecting the inappreciable contribution given by the saturated counters.

A counting Bloom filter can, therefore, be used to approximately answer to the questions: Is a member of a set present in the filter *now*? How many occurrences of this member are present? In the first question, the word *now* indicates that the member has been added, but not deleted. The second question is similar to the $unresolved$ multiple matching CAM behavior, providing the number of the instances of an element present in the filter.

5 Error Detection and Correction in a CAM

In this section, we discuss how to detect and correct SEU induced errors in a CAM. As stated in Section 1, we will focus on a solution that will not require substantial modifications to existing CAM circuits. As in [22], we make use of parity check bits, and, by introducing a Bloom filter we correct SEU induced errors at a higher system level. Therefore, while we assume that the CAM output could be affected by an error, we monitor the inputs and outputs of the CAM and, by leveraging the characteristics of the fault model described above, we show that we can correct the occurrence of errors. Differently from [22] the address that is provided to the CAM already includes the parity bits, this encoding can be performed in a block that is externally instantiated with respect to the CAM itself, therefore, not requiring any structural modification to existing commercial CAMs. Another main difference between our method and the method proposed in [22] or [10] is that the redundant parity bits are used to form an error-detection code, leaving the error-correction phase to the combined use of the information provided by the Bloom filter and the error-correction algorithm. The separation between the detection and the correction phases allows the designer to avoid the use of decoders for ECC that can be very costly, especially for multiple-bit error correction [3]. The parity check bits protection scheme is based on memory interleaving.

Memory interleaving is one of the most common approaches to deal with multiple errors in memories [18]. This approach exploits the topological contiguity of the MBUs, as discussed in [17] arranging the codeword so that neighboring cells belong to different codewords. Therefore, errors in an MBU will cause single errors in different words that can be detected by the use of single-error-detection codes. The scheme of a memory protected by an interleaved parity-based error-detection scheme is shown in Fig. 4a.

A memory of data width W is divided in $\lceil W/ID \rceil$ groups of ID bits, where ID is the interleaving distance. The check bits $P(0), \ldots, P(\lceil W/ID \rceil - 1)$ are computed staring from the data bits $D(0), \ldots D(W-1)$ in the following way:

$$P(i) = \bigoplus_{k=0}^{\lceil W/ID \rceil - 1} D(i + k \cdot ID). \quad (7)$$

In Fig. 4b it can be seen an MBU affecting three consecutive bits. The length l of the MBU is three and all the bits are affected by the error. The parities of the groups 0, 2, 3 are changed, while the parity of group 1 in unchanged, since no errors occur in this group. Instead, the picture in Fig. 4c shows an MBU with length $l = 5$, but the errors actually affect 3 bits, corresponding to an undetectable error. In fact, the errors affect two bits of the group 2, leaving its parity unchanged. To avoid these undetectable errors the interleaving distance ID should be chosen as $ID \geq l$, where l is the maximum expected MBU size. The redundancy needed to protect the CAM against l-MBU is, therefore, linear with respect to the maximum expected length of the MBU. Under this assumption, all the errors caused by an SEU change al least one group parity, transforming the codeword in a noncodeword.

With the proposed parity scheme, we can always avoid the pseudo-HIT error in a CAM. In fact, with an interleaved parity encoded CAM if an SEU hits a codeword it will turn it into a noncodeword (rather than a wrong codeword) and, thus, since the CAM search words are always codewords pseudo-HIT or multi-HIT errors will never occur. When an error hits an entry of the CAM, changing it into a noncodeword, the CAM produces a MISS signal if this entry is queried. Note that this error induced false-MISS cannot be distinguished by the MISS signal produced when querying an item not stored in the CAM.

Therefore, a CAM affected by an SEU can be seen as a structure that, when affected by an error, gives a false-negative response to the query of an element. This is exactly the opposite behavior of a counting Bloom filter. The CAM is susceptible to the occurrence of false negatives (due to SEU), the CBF is susceptible to the occurrence of false positives (due to hash collision). Based on this assumption, we can describe the proposed solution as shown in Fig. 5

The address is fed to a counting Bloom filter and in parallel is given to a CAM by passing through the "GROUP PARITY ENCODER" module. The CBF is configured to provide as output a MISS response if all the counters corresponding to a query are set to 0, a HIT otherwise. The deletion operation was presented in Section 4 and avoids the presence of false negatives at the cost of a higher false-positive probability. With this approach the CBF acts as a classic BF with respect to the query operation, but can also perform the delete operations. When an entry in the CAM is substituted by another one, the old entry is also deleted from the CBF, while the new one is inserted. As discussed in Section 4. The number b of bits of the CBF is set to 2, since the number of saturated counters is always negligible. The outputs of the CAM and of the CBF are input to the "CHECKER" that detects whether there has been an error.

The possible cases are summarized in Table 1.

When the CAM responds with a HIT, its output can be considered correct (with parity encoding there are no false HITs) this will be different when we will make also the assumption of multiple HITs as we will discuss below. Instead, when the CBF provides a MISS signal, because there are no false negatives in a CBF, the "CHECKER" module can output a MISS signal. A problem arises when the CBF provides a HIT signal and the CAM a MISS signal. This discordance can be due either to the presence of a false positive in the CBF, or to the presence of an SEU in the

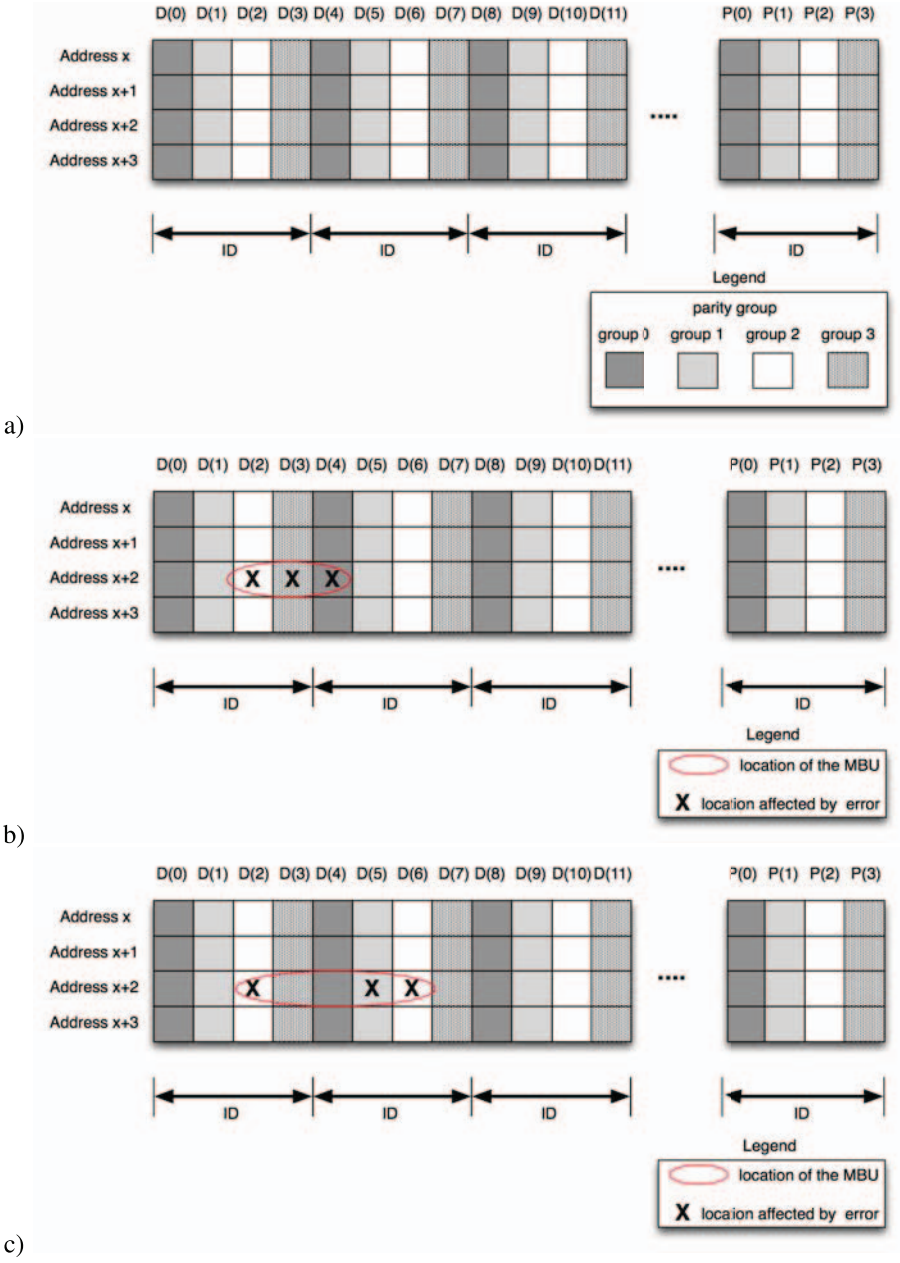


Fig. 4. (a) Scheme of a memory protected by interleaved parity-based error-detection code. (b) Detectable 3-bits MBU error. (c) Undetectable 5-bits MBU error.

CAM, or both (although this is a very unlikely case). When this situation occurs the checker raises an error signal and a correction algorithm must be applied to detect whether the CAM is affected by an SEU or the CBF has given a false positive. The algorithm will first check if the CAM has been affected by a false MISS due to the corruption of a bit of the searched word. To check if the CAM has been affected by an SEU it is possible to query the CAM against all the possible noncodewords that have a certain Hamming distance from to the word that has produced the error signal. Supposing that the maximum expected length of an MBU is l, not only the maximum Hamming distance between the modified word and the original word is l, but the modified bits are confined in l consecutive bits. Fig. 4b presents an MBU with $l = 3$ in which all the bits have been changed, while in Fig. 4c only a subset of the $l = 5$ consecutive bits have been changed by the SEU. Therefore, the possible number of noncodeword to check is $2^l \cdot W$, where W is the data width of the CAM and 2^l is the number of possible errors that can be caused by an l-bits MBU. After $2^l \cdot W$ queries, the algorithm is able to discover if the discordance has been caused by an SEU. If the noncodeword has been found, it can be corrected, resolving the discordance; this approach is similar to the method described in [10]. Otherwise, we know that no SEU has occurred and the CBF has been affected by a false positive. Because of the nature of CBFs, a false positive will be given each time the word causing the false positive is queried, we need a method to mark this entry in such a way that the algorithm for resolving discordances is executed only the first time, and not each time this word is queried. A possible solution is to store the words producing false positives in a

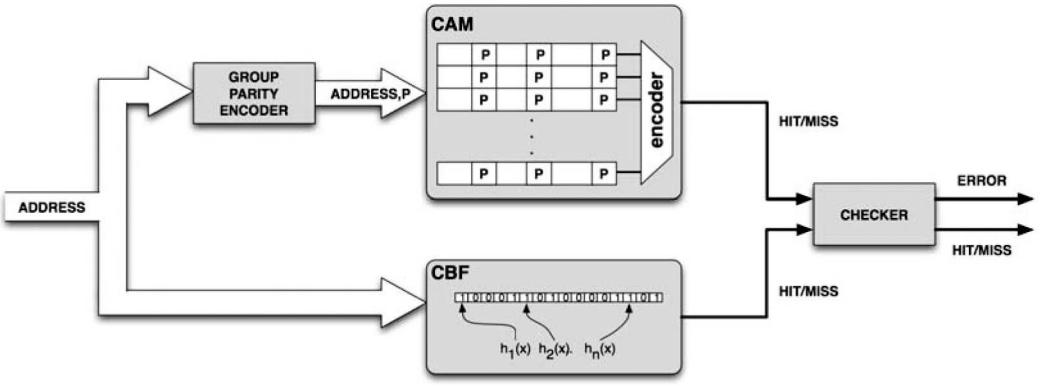


Fig. 5. Scheme of the proposed error-correction scheme for a CAM/CBF pair.

little auxiliary CAM, which signals to the checker that the HIT given by the CBF is a false positive. This auxiliary CAM can be very little compared to the main one, and can be easily integrated in the principal CAM either when the CAM is used in conjunction with a RAM, or when one of the multiple match policies is applied. The scheme of the CAM with CBF and auxiliary CAM is presented in Fig. 6.

The corresponding possible cases are now summarized in Table 2.

For a given query, the event that two CAMs give a MISS signal and the CBF gives a HIT can occur only once. In fact, if this event is due to an SEU, this will be detected and corrected by the correction algorithm, instead, if the event is due to a false positive generated by the CBF, the auxiliary CAM will flag the data causing a false positive, and each subsequent query will induce also a HIT in the auxiliary CAM. Therefore, the event of MISS in the two CAMs and HIT in the filter occurs only once, because the correction algorithm modifies this situation by either correcting an SEU in the CAM, or by flagging the offending data in the auxiliary CAM. A quantitative analysis of the size of the auxiliary CAM will be given in Section 6. With a correct sizing of the auxiliary CAM, the probability of an overflow in the auxiliary CAM can be made very low.

However, even if this event occurs, the system continues to work correctly. The drawback of having the auxiliary CAM full is that some false positive responses of the CBF cannot be inserted in the auxiliary CAM, therefore, each time one of these entries is requested to the system time will be wasted while performing the above described correction algorithm.

A different version of the above described approach can be introduced when the CAM is used in conjunction with a RAM to implement for example a Look-up address table. The Look-up address table is composed of a CAM and a RAM. The CAM outputs the index associated to the entry matching the search word at its input. This index is the address provided as an input to the RAM associated to the CAM which, therefore, always outputs a value. The value provided by the RAM is the final result of the look-up operation. This combination has a wide usage in the implementation of fast lookup tables for network routers. Usually, with the lookup operation, the provided input corresponds to an output of the same size.

Fig. 7 shows an example where a solution similar to what described in Fig. 5 is adopted to a case when the CAM is used in conjunction with a RAM to provide a Look-up address table. The integrity of the data stored in the RAM itself is ensured by the use of ECC (such as Hamming), moreover in each RAM entry a further bit is added. This additional bit is used to substitute the auxiliary CAM. The CAM system is composed of a CAM/RAM block, a parity encoder that provides the parity of the input address (search word), a counting Bloom filter and a checker that chooses the correct output based on the policy described in Table 2. The difference with respect to the previous case is that when the CAM produces a MISS signal and the BF provides a HIT signal due to a false positive, the correction algorithm detects the false positive, inserts the data producing the false positive directly into the CAM, and sets to 1 the auxiliary bit in the associated row of the RAM instead of using an auxiliary CAM. Therefore, a further query of the same data will produce a HIT response on both the CBF and the CAM, but the checker will detect that this corresponds to a false positive on the CBF by checking the additional bit A stored in the RAM. It should be noted that, while in [30] the parity bits were shifted from the CAM to the RAM, in this case they have been left into the CAM. In [30], the two CAMs used to detect errors were also used to manage the false HIT occurrence. Instead, with the proposed approach only one CAM is used, and the CBF replaces the duplicated CAM proposed in [30].

6 Sizing of Bloom Filter and Auxiliary Cache

In this section, we discuss how to define the size of the counting Bloom filters and of the auxiliary cache. First of all, we report the probability of false positive for a Bloom filter in Fig. 8.

TABLE 1
Possible Error Cases and Correct Output Selection

CAM	BF	Correct Output
HIT	HIT	HIT
HIT	MISS	impossible
MISS	HIT	false positive or SEU or both
MISS	MISS	MISS

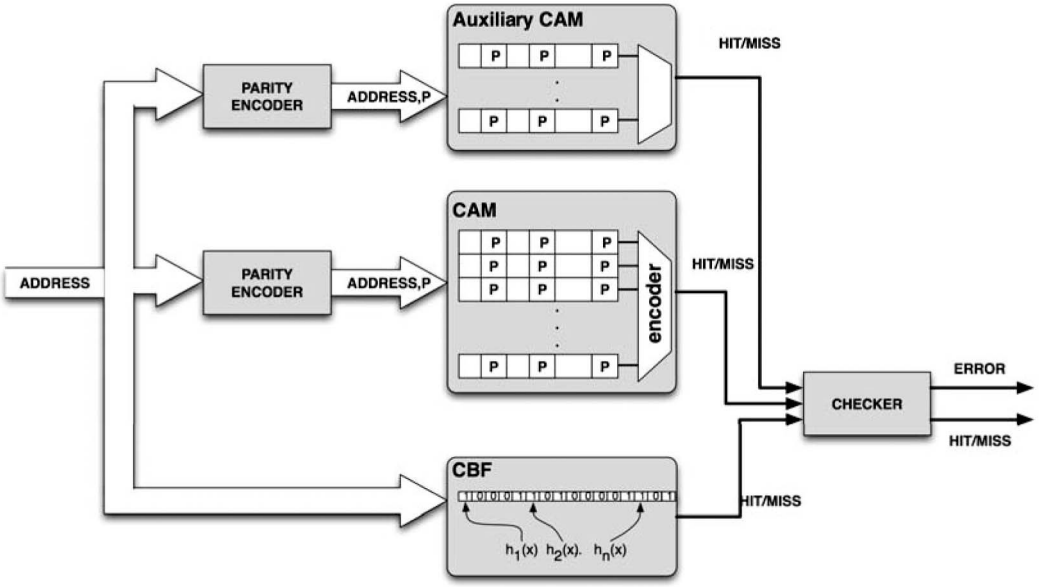


Fig. 6. Scheme of the proposed error-correction scheme for a CAM/CBF pair with auxiliary CAM.

In can be seen that the probability of false positive grows with the number of stored items and directly depends on the ratio between the number of stored items n and the filter size m. In particular, a filter occupation corresponding to a n/m ratio of $1/8$ has a false-positive probability of about 1E-3 (0.1 percent), while a value of $1/16$ provides a false-positive rate of about 1E-6. It should be noted that the size of the Bloom filters is independent on the width of the searched data, therefore, this method obtains better results for CAMs with wider word width.

Moreover, it should be noted that the theoretical results presented here correspond to a worst case scenario in which no address locality has been taken into account, in fact it can be shown that a sequence of consecutive addresses (typical of locality in caches) does not induce hash collisions. The interested reader can refer to the data presented in [12] and [13] that also provide simulation results from real-case scenarios.

From what discussed above, we recall that a higher false-positive rate in a Bloom filter corresponds to a temporal overhead due to the execution of the error-detection procedure. The Error-detection procedure searches in the CAM against $2^l \cdot W$ noncodewords, where W is the width of the searched word and l is the maximum expected length of the MBU. The estimated temporal overhead can, therefore, be computed as proportional to the product between the false-positive rate and the number of queries, as reported in the following equation:

$$T_{overhead} \approx P_{fp} \cdot 2^l \cdot W. \tag{8}$$

To limit the $T_{overhead}$, we propose to leverage the auxiliary CAM introduced above (Fig. 6). This approach gives a substantial advantage under the hypothesis that the inputs of the CAM are almost limited to a certain fixed subset of items. This is a plausible assumption because we can usually divide the set of data input to a CAM for a query into two subsets: The subset of entries that are most likely to the queried, corresponding to data stored in the CAM, for which we want to know the corresponding entry, and the subset of entries that are less likely to be queried which are those not present in the CAM. With this assumption also the words causing false positives in the BF belong to the same subset. Once these entries have been stored in the auxiliary CAM the false-positive rate of the filter is drastically reduced until an entry not belonging to the frequently used set causes an unexpected false positive.

Therefore, suppose that we have a set of frequently queried words composed of K items, corresponding to the number of entries of the CAM and P_{fp}, the probability that there are J different false positive in the set is given by the following equation:

$$P_N = J \cdot P_{fp}/K = J/K \cdot P_{fp}. \tag{9}$$

The ratio J/K can be seen as the relative dimension of the auxiliary CAM with respect to the CAM to be protected. The tradeoff for minimization of the $T_{overhead}$ can be, therefore, performed by replacing in (8) P_{fp} with P_N obtained from (9). For example, using a n/m ratio of $1/8$ and a J/K ratio of $1/128$ the value of P_N is around 1E-5.

7 Results

To evaluate the effectiveness of the proposed solution, we carried out several experiments with different values of CAM widths and sizes to assess the corresponding area and power overhead . Width values have been set to 32, 64, and 128 bits, while the number K of rows of the CAM varies from 32 to 32K entries. For the comparison between the CAM without error-correction capabilities and the one with

TABLE 2
Possible Error Cases and Correct Output Selection

CAM	CBF	Auxiliary CAM	Correct Output	Remark
HIT	HIT	-	HIT	
HIT	MISS	-	N/A	impossible
MISS	HIT	MISS	false positive and/or SEU	can occur only once
MISS	HIT	HIT	false positive	
MISS	MISS	-	MISS	

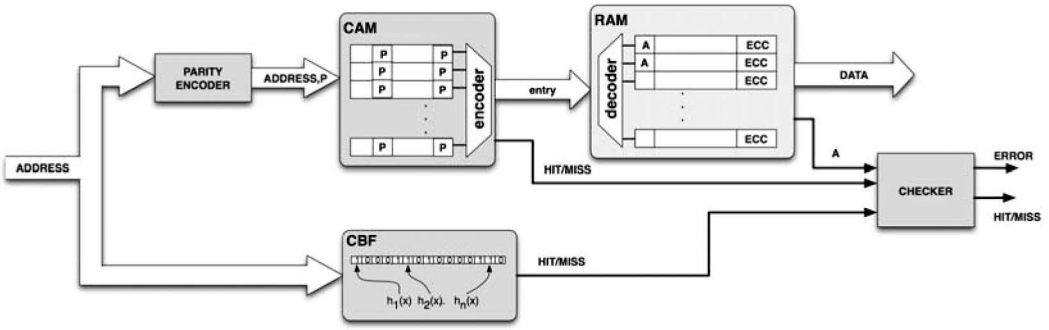


Fig. 7. Error detection and correction for a CAM used in conjunction with a RAM.

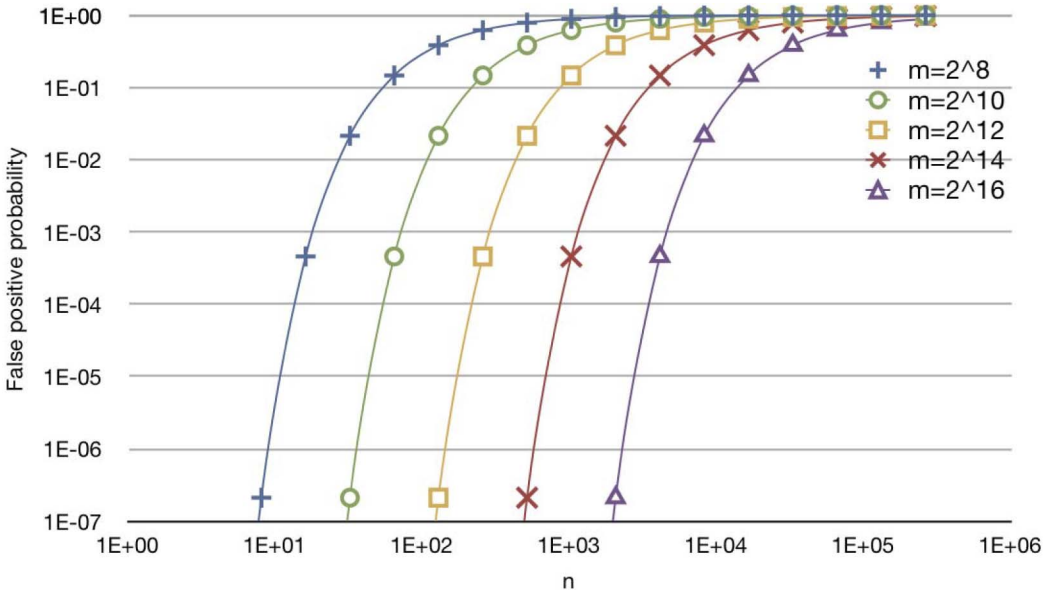


Fig. 8. The false-positive probability P_{fp} as a function of n and m. An optimal number of hash functions k has been assumed.

error-correction capabilities, we suppose to use an interleaved factor of $ID = 4$, therefore, the CAMs used for comparison have widths of 36, 68, and 132 bits. The auxiliary CAM, can be assumed to be 128 times smaller than the principal CAM, thus providing a negligible contribution to the overhead in terms of energy and area of the overall system. To compute the contributions of the CBF overhead in terms of area and power consumption, we fix for all the experiments the ratio $n/m = 1/8$. The CBF area evaluation has been done ignoring the contributions of the functions performing the hash of the incoming data, thus, assuming that the CBF is realized using an SRAM with a size within the n/m occupation ratio. For a given size number of row K, the corresponding size S of the CBF can be calculated imposing that the number of items stored in the CBF is equal to the number of rows of the CAM to be protected. With the assumption that $n = K$, and with 2 bits per counter we obtain

$$S = 2 \cdot m = 2 \cdot \frac{K}{n/m}.$$

With the ratio $n/m = 1/8$ the size of the SRAM is $16 \cdot K$. To collect the data for area evaluation, we used cacti 6.5 [34], while to estimate the CAM's power consumption we used a version that has been developed for fully associative CAMs [35]. All data refer to the 32 nm technology node. In Table 3, the parameters of the unprotected and protected CAMs are reported, in conjunction with the size of the SRAM used for the CBF.

Fig. 9 presents the area occupation of the different CAMs taken into account. In particular, Fig. 9a is the area of an unprotected CAM, Fig. 9b is the area of the corresponding SRAM, Fig. 9c presents the total area occupation, and Fig. 9d the overhead of our solution as the ratio between the unprotected CAM and the sum of the SRAM and the CAM with group parity protection.

The results of our evaluation in terms of power consumption are reported in Fig. 10. The graphic in Fig. 10a shows the power consumption of the original unprotected CAM. Fig. 10b presents the data of the parity group protected CAM and Fig. 10c the power consumption of the SRAM used for the CBF. Finally, Fig. 10d presents the power overhead of our solution.

The data presented in Fig. 9 shows that the overhead of our solution is broadly independent of the CAM width and, therefore, the percentage area overhead is reduced when wide CAMs are used. In particular, for 64-bits CAM the area overhead is less than 30 percent, while for 128-bits CAM this value decreases to 15 percent.

TABLE 3
Parameters of the CAM and of the Corresponding SRAM Used for the CBF

# of rows of the CAM	width of the unprotected CAM	width of the protected CAM	size of the SRAM (bits)
32	32	36	512
	64	68	
	128	132	
128	32	36	2K
	64	68	
	128	132	
512	32	36	8K
	64	68	
	128	132	
2048	32	36	32K
	64	68	
	128	132	
8192	32	36	128K
	64	68	
	128	132	
32768	32	36	512K
	64	68	
	128	132	

A similar discussion can be done for power consumption. According to [36], the energy associated to a query in a CAM is eight times higher than an SRAM access, hence we can give an approximation of the CBF power consumption. The energy overhead with respect to the unprotected CAM has two main contributions: the use of a wider CAM to accommodate the additional group parity bits and the energy consumption of the SRAM. The former grows with the width of the CAM, therefore corresponding to a constant overhead, the latter is constant and, therefore, its contribution is less important for wider CAMs. Also in this case the data obtained using [34] for the estimation of the SRAM power consumption and [35] are in accordance with these considerations. In fact, it can be seen that the power overhead starts from around the 25-30 percent for narrow CAMs and decreases for wider CAMs.

The estimation of the overhead introduced by the parity encoders was made according to [22]. The total number of XOR gates for computing the ID check bits is $W - ID$. Instead, the number of XOR levels for each check bit is $\lceil W/ID \rceil$. These gates can be realized with less than 1,000 transistors, therefore the encoder introduces a negligible contribution in terms of both area and power overhead. Moreover, according to [22], the encoder delay can be estimated to be approximately 1.0 ns, and the encoding can be performed in a pipeline stage before the search.

Also the hash function for the CBF can be realized with a network of XOR gates [37], and, therefore, both area and energy associated to this function can be considered negligible. To reduce the delay, the encoding and the hashing functions can be computed in parallel, achieving an overall latency penalty of one clock cycle.

It should be noticed that, even if the data presented in [25] cannot be directly applied to a fully associative CAM, its result for 16 KB four-way associative data cache in terms of area penalty (10 percent) and energy penalty (20 percent) are comparable with those presented here. In particular, the use of CBF allows us to use an SRAM that requires less power than a CAM, while for the area occupation the best results are obtained for CAM with larger word width. With respect to the data shown in [22], area and energy overhead

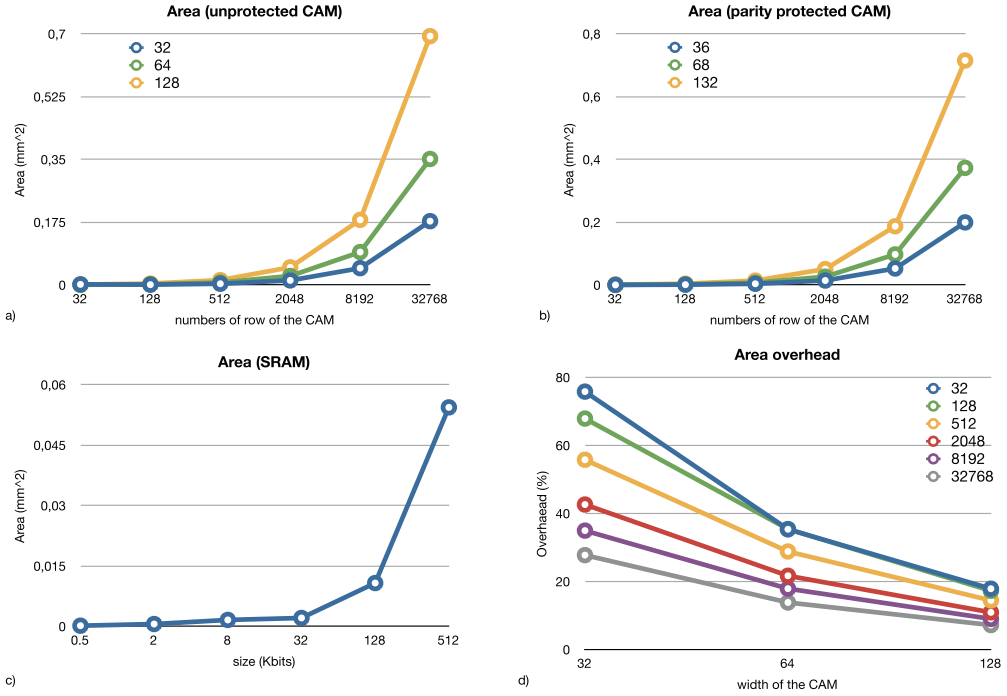


Fig. 9. Area occupation: (a) area of the unprotected CAM, (b) area of the SRAM used for the counting Bloom filter, (c) total area occupation, and (d) overhead in percentage of the proposed solution.

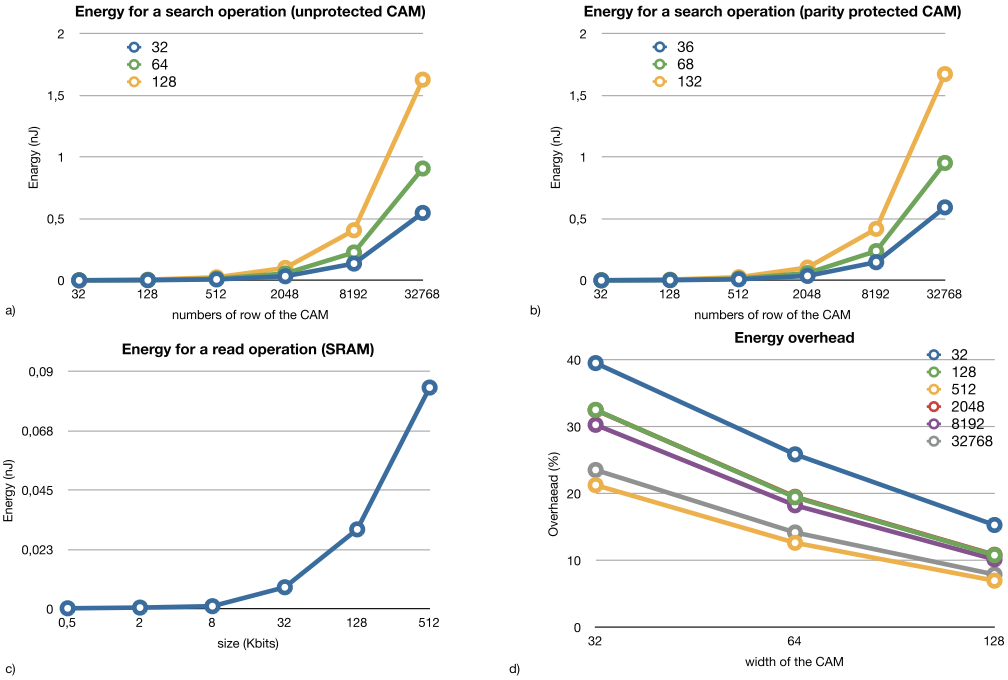


Fig. 10. Power consumption: (a) energy for a search in the unprotected CAM, (b) energy for parity group protected CAM, (c) energy of the SRAM used in the CBF, and (d) overhead in percentage of the proposed solution.

are higher, since Pagiamtzis et al. [22] report an area penalty of 12 percent and a near zero energy overhead. However, the solution presented in [22] is designed to face only single-bit errors, while our solution has been designed to tolerate up to 4-bits MBU. Moreover, while our solution can be easily configured to tolerate MBUs of different length, the scheme of [22] would require the use of complex ECCs and its applicability does not appear to be straightforward. From the results presented above, we can evaluate the overhead of our solution in two different real scenarios in which large CAMs are used. The first scenario is a Layer 2 Ethernet switch, which reads the 6 bytes ethernet destination address of each incoming Ethernet packet and selects the forward output port depending on the value of a forward table. Due to the dimension of the table (up to 128K entries) these switches use an external CAM to store the forward table. In this case, the width of the CAM is 48 bits, while the number of entries varies from 32K to 128K (see for example the CISCO Catalyst Family [41]). The use of these external CAMs limits their maximum operating frequency to 300-400 MHz. The dimension of the corresponding SRAM used for the CBF varies from 512 Kbits (64 Kbytes) to 2 Mbits (256 Kbytes). Since Layer 2 Ethernet switches are usually equipped also with high-speed SRAM memories, that provide several megabits of memories, the CBF can be stored in these memories that are already available. Alternatively, a commercial 18-Mbits SRAM would provide enough space to store the CBF. It can be noted that these SRAMs compared to the CAM have similar operating frequencies, consume only a fraction of their energy and have a much lower retail cost.

The second example is the tag array of caches and TLB for a server oriented microprocessor. The presented method is used to protect all these tags, and the effectiveness of our method is tested simulating the behavior of a microprocessor with respect to the insertion, query, and eviction operation in the tag arrays, and the corresponding behavior of the CBF used to protect the tags against the occurrence of a temporary fault. The results of these experiments are reported in the next section.

8 Simulation of the Microprocessor Tag Arrays Protection

For these experiments, we used the sim-outorder processor simulator provided in SimpleScalar 3.0 [39] and the SPEC INT2000 Benchmark suites [38]. The benchmarks have been applied to a modified SimpleScalar simulator configured with the same architecture parameters of [24] and reported in Table 4.

The modification of SimpleScalar allowed us extracting the addresses corresponding to the insertions, queries, and evictions from the CAM associated to the different caches and TLBs, i.e., from the level one instruction and data caches, from the level 2 unified cache and from the data and address TLBs. The results of the execution of these programs have been reported in Fig. 11. The leftmost bar of each group represents the number of executed instructions, while the other bars in each group represent the number of accesses to the IL1, DL1, UL2, ITLB, and DTLB, respectively (the y-axis is in logarithmic scale with unit of 1 million of instructions/accesses). Moreover, we reported in Fig. 12 the miss rate (in percentage) for each kind of cache/TLB (the y-axis in Fig. 12 is in logarithmic scale). We remark that the results presented in Fig. 12 are very dependent on the type of program that is being executed. For this reason, the cache miss rate is highly variable depending on the total number of accesses which is related to the overall instruction and data footprint.

TABLE 4
SimpleScalar Configuration Parameters

Parameter	Value
Processor	
Functional Units:	4 integer ALUs, 4 FP ALUs 1 integer multiplier/divider 1 FP multiplier/divider
LSQ Size / RUU Size	8 Instructions / 16 Instructions
Fetch / Decode	4 / 4 instructions/cycle
Issue / Commit Width	4 / 4 instructions/cycle
Fetch Queue Size	4 instructions
Cycle Time	1 ns
Cache and Memory Hierarchy	
L1 Instruction Cache (IL1)	64KB, 2-way, 64 byte lines, 1 cycle latency
L1 Data Cache (DL1)	64KB, 2-way, 64 byte lines Write-thru, no allocate on write miss 1 cycle latency
Unified L2 (UL2)	1MB, 4-way, 128 byte lines 6 cycle latency, write-back
Data TLB (DTLB)	32 entries, fully associative, 30 cycle latency
Instruction TLB (ITLB)	32 entries, fully associative, 30 cycle latency
Memory	100 cycle latency

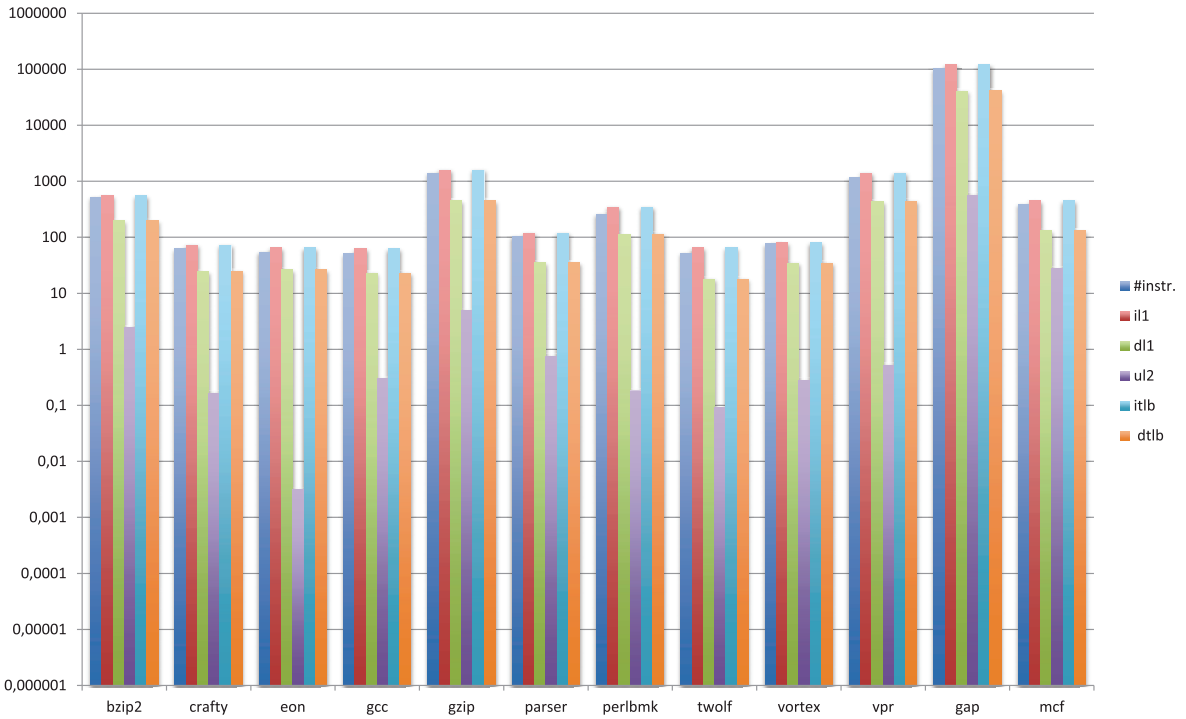


Fig. 11. Number of instructions and number of accesses (in millions) to the different caches/TLBs.

The addresses extracted during the execution of the benchmarks have been used to feed a counting Bloom filter for each kind of CAM used in caches and TLBs. We selected the CBF for each cache/TLB using the entries of Table 3, that should provide a theoretical false-positive rate of 0.1 percent. More specifically, for the IL1 and DL1 caches that have 1,024 entries, we used a 16-Kbits CBF, for the UL2 cache that has 8,192 entries we used the 128-Kbits CBF and finally, for the ITLB and DTLB we used the 512-bits CBF.

To each CBF, we associated a false-positive counter that operates as follows: When an address not present in a cache/TLB is signaled as present by the CBF, the corresponding false-positive counter is incremented. At the end of the program execution, the false-positive counter contains the total number of false positives actually signaled by each CBF. For all the caches and TLBs except UL2, the CBFs have been filled as expected (i.e., they reached the occupation ratio $n/m = 1/8$) and received more than one million of accesses, therefore the obtained results seems statistically

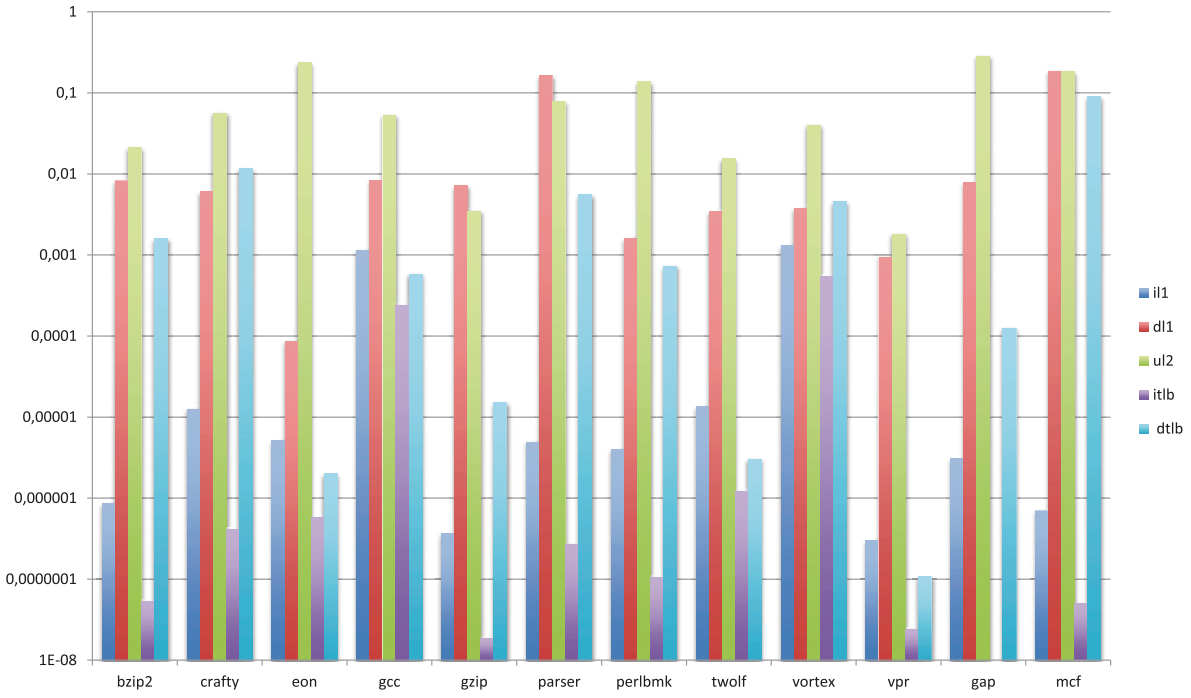


Fig. 12. Miss rate (in percentage) for IL1, DL1, UL2, ITLB, and DTLB.

valid. Instead, for UL2 the number of accesses was statistically relevant only for the *gzip*, *gap*, and *mcf* benchmark programs which have more than one million of accesses to the UL2 cache. The results of these simulations are shown in Table 5. For each type of CAM taken into account the worst case (in terms of false-positive ratio) among the results obtained from the benchmark has been reported. In particular, the second column reports the benchmark name, the third one the number of detected false positives, the fourth column presents the computed false-positive ratio. The fifth column presents the theoretical value obtained by using (2).

As already outlined in the previous section, the sequence of consecutive addresses of instruction cache and TLB, should greatly reduce the number of collisions in the corresponding CBF, achieving results in terms of false-positive ratio better than the theoretical ones. However, from our simulation, we found a false-positive rate higher than expected. We further investigated this behavior that occurs when the Bloom filter size is too small. This behavior is strictly dependent on the goodness of the hash functions used. For our experiments, we used the method presented in [40], which allows us to use hash functions that are not completely independent, but that are more efficient with a filter size m in the order of magnitude of several thousands. Therefore, by using these hash functions that are suboptimal in the case of small filters, the actual false-positive rate can be worse than the theoretical ones as indeed was in the outcome of our simulations. The last column of Table 5 presents the time overhead. As discussed in Section 6, (see (8)) the time overhead depends on the number of false positives and on the width of the tag. When a false positive occurs, the inspection of all the noncodeword near to the searched codeword has to be done in the Cache. In our simulation, we compute for each false positive an overhead of W clock cycles, where $W = 32$ is the tag width. We remark that this is a very pessimistic assumption, since the false positive corresponds to a MISS in the cache, therefore, when this condition occurs, the processor should wait until the data is fetched from the next level cache/memory. During this wait time, the scrubbing procedure used to check the presence of a fault is partially overlapped with the wait for data fetching.

TABLE 5
False-Positive Rate and Time Overhead for the CBF

CAM type	worst case Benchmark	total number of FP	computed FP ratio [%]	theoretical FP ratio [%]	maximum execution time overhead [%]
IL1	gcc	5755	9,1E-5	3,8E-5	0,003
DL1	perlbmk	18174	1,6E-4	5,7E-5	0,005
UL2	mcf	3502505	0,12	0,10	3,84
ITLB	gcc	1695	2,7 E-5	6,8 E-6	8,64E-4
DTLB	mcf	4755055	0,03	0,01	0,9

TABLE 6
Energy Overhead

CAM type	Energy overhead [%]
IL1	25
DL1	24
UL2	31
ITLB	38
DTLB	37

Finally, Table 6 presents the energy overhead of the proposed method, with respect to the energy consumption of the cache without any protection. The dissipated energy in case of the unprotected cache is given by the sum of the static and dynamic dissipation of the tag array, while the energy in the protected cache is dissipated also by the circuitry storing the additional parity bits in the tag array and by the SRAM memory storing the content of the Bloom filter. The static power dissipation contribution is application independent, and depends only on the memory array configurations. The dynamic contribution is affected by the program execution, depending on the rate of accesses to a certain cache. We computed this activity rate as the ratio between the number of accesses and the number of instructions. As expected, (see [34]) for UL2 cache the activity rate is less than 1 percent for all the benchmarks, consequently the power dissipation is dominated by the static power consumption and is the same in all the simulation performed. Instead, for DL1 and DTLB the activity rate varies from 32 percent ($gzip$) to 48 percent (eon), while for IL1 and ITLB it varies from 103 percent ($vortex$) to 133 percent ($perlbmk$). The activity rate affects both the cache and the SRAM, since any access to the cache corresponds to an access to the Bloom filter, therefore the sensitivity of the energy overhead to the application is always very small. The energy consumption overhead has a value between the overhead computed using only static power consumption estimate (applicable to caches with low activity rate), and the overhead computed using the dynamic power consumption estimate. In fact, when the activity rate increases. this dynamic part becomes dominant. The data reported in Table 6 are referred to the worst cases benchmark, therefore their value are representative of all the benchmarks.

9 Conclusion

Content addressable memories like other memories can be affected by the occurrence of SEU which can alter their operation causing different effects such as pseudo-HIT or pseudo-MISS events. To avoid the effects of SEUs the available technical literature proposes several approaches that require modifications to the internal architecture of the CAM. This paper proposed a method to detect and correct errors occurring on a CAM using interleaved parity bit encoding to avoid pseudo-HIT and comparing the output of the CAM with the response of a Bloom filter to detect the other types of errors that can occur in the CAM. This approach does not require any modification to the internal structure of existing CAMs. The interleaved parity bit encoding protects the CAM against MBU, while the combined use of a Bloom filter with a suitable error-correction algorithm allows to correct errors occurring in the CAM. Moreover, the use of a counting Bloom filter permits to consider the dynamic behavior of the CAM by keeping track of the previous insertions and deletions. Moreover, a discussion on the sizing of the Bloom filter and of the auxiliary CAM has been presented and finally, some simulation experiments showing the effectiveness of this techniques for the protection of caches and TLB of a microprocessor have been reported.

Acknowledgments

Salvatore Pontarelli would like to thank Dott. Simone Teofili and Professor Giuseppe Bianchi for introducing him to the use of Bloom filters and for their encouragement to search for new applications for these structures. This research was partially funded by the Italian Ministry for University and Research; Program "Incentivazione alla mobilità di studiosi stranieri e italiani residenti all'estero", D.M. n.96, 23.04.2001 and by DEMONS, a research project supported by the European Commission under its seventh Framework Program (contract no. 257315).

References

[1] N. Kanekawa, E.H. Ibe, T. Suga, and Y. Uematsu, *Dependability in Electronic Systems: Mitigation of Hardware Failures, Soft Errors, and Electro-Magnetic Disturbances.* Springer Verlag, 2010.

[2] G.C. Cardarilli, M. Ottavi, S. Pontarelli, M. Re, and A. Salsano, "A Fault-Tolerant Solid State Mass Memory for Space Applications," *IEEE Trans. Aerospace and Electronic Systems,* vol. 41, no. 4, pp. 1353-1372, Oct. 2005.

[3] W.W. Peterson and E.J. Weldon, *Error-Correcting Codes.* The MIT Press, 1972.

[4] T. Yamagata, M. Mihara, T. Hamamoto, Y. Murai, T. Kobayashi, M. Yamada, and H. Ozaki, "A 288-kb Fully Parallel Content Addressable Memory Using a Stacked-Capacitor Cell Structure," *IEEE J. Solid-State Circuits,* vol. 27, no. 12, pp. 1927-1933, Dec. 1992.

[5] L. Chisvin and R.J. Duckworth, "Content-Addressable and Associative Memory: Alternatives to the Ubiquitous RAM," *IEEE Computer,* vol. 22, no. 7, pp. 51-64, July 1989.

[6] V. Lines, A. Ahmed, P. Ma, S. Ma, R. McKenzie, H-S. Kim, and C. Mar, "66 MHz 2.3 M Ternary Dynamic Content Addressable Memory," *Proc. IEEE Int'l Workshop Memory Technology, Design Testing,* pp. 101-105, 2000.

[7] N. Azizi and F.N. Najm, "A Family of Cells to Reduce the Soft-Error-Rate in Ternary-CAM," *Proc. 43rd Ann. Design Automation Conf.,* 2006.

[8] S.C. Krishnan, R. Panigrahy, and S. Parthasarathy, "Error-Correcting Codes for Ternary Content Addressable Memories," *IEEE Trans. Computers,* vol. 58, no. 2, pp. 275-279, Feb. 2009.

[9] F. Salice, M.G. Sami, and R. Stefanelli, "Fault-Tolerant CAM Architectures: A Design Framework," *Proc. IEEE 17th Int'l Symp. Defect and Fault Tolerance in VLSI Systems (DFT '02),* pp. 233-241, Oct. 2002.

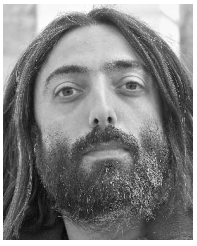
[10] A. Bremler-Barr, D. Hay, D. Hendler, and R.M. Roth, "PEDS: A Parallel Error Detection Scheme for TCAM Devices," *IEEE/ACM Trans. Networking,* vol. 18, no. 5, pp. 1665-1675, Oct. 2010.

[11] C. Wan, J. Lan, and Y. Hu, "Lookup with CAM Aided Hash Table," *Proc. IEEE Int'l Conf. Frontier of Computer Science and Technology,* 2009.

[12] J. Peir, S. Lai, S. Lu, J. Stark, and K. Lai, "Bloom Filtering Cache Misses for Accurate Data Speculation and Prefetching," *Proc. 16th Int'l Conf. Supercomputing (ICS '02).*

[13] M. Ghosh, E. Ozer, S. Ford, S. Biles, and H.S. Lee, "Way Guard: A Segmented Counting Bloom Filter Approach to Reducing Energy for Set-Associative Caches," *Proc. ACM/IEEE 14th Int'l Symp. Low Power Electronics and Design (ISLPED '09),* 2009.

[14] K. Pagiamtzis and A. Sheikholeslami, "Content Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey," *IEEE J. Solid-State Circuits,* vol. 41, no. 3, pp. 712-727, Mar. 2006.
[15] D.A. Patterson and J.L. Hennessy, *Computer Architecture: A Quantitative Approach,* third ed. Morgan Kaufmann, 2003.
[16] H. Chao, "Next Generation Routers," *Proc. IEEE,* vol. 90, no. 9, pp. 1518-1558, Sept. 2002.
[17] S. Satoh, Y. Tosaka, and S.A. Wender, "Geometric Effect of Multiple-Bit Soft Errors Induced by Cosmic Ray Neutrons On DRAMs," *IEEE Electron Device Letter,* vol. 21, no. 6, pp. 310-312, June 2000.
[18] P. Reviriego, J.A. Maestro, S. Baeg, S.J. Wen, and R. Wong, "Protection of Memories Suffering MCUs Through the Selection of the Optimal Interleaving Distance," *IEEE Trans. Nuclear Science,* vol. 57, no. 4, pp. 2124-2128, Aug. 2010.
[19] N. Seifert, B. Gill, K. Foley, and P. Relangi, "Multi-Cell Upset Probabilities of 45nm High-K + Metal Gate SRAM Devices in Terrestrial and Space Environments," *Proc. IEEE Int'l Reliability Physics Symp. (IRPS '08),* pp. 181-186, 2008.
[20] H. Noda et at., "A Cost-Efficient High-Performance Dynamic TCAM with Pipelined Hierarchical Search and Shift Redundancy Architecture," *IEEE J. Solid-State Circuits,* vol. 40, no. 1, pp. 245-253, Jan. 2005.
[21] H. Noda, K. Dosaka, F. Morishita, and K. Arimoto, "A Soft-Error Immune Maintenance-Free TCAM Architecture with Associated Embedded DRAM," *Proc. IEEE Custom Integrated Circuits Conf.,* pp. 451-454, Sept. 2005.
[22] K. Pagiamtzis, N. Azizi, and F.N. Najm, "A Soft-Error Tolerant Content-Addressable Memory (CAM) Using An Error-Correcting-Match Scheme," *Proc. IEEE Custom Integrated Circuits Conf.,* 2006.
[23] A.K. Somani and S. Kim, "Transient Fault Detection in Cache Memories by Employing a Small Shadow Cache," *Proc. Dependable Computing for Critical Applications Conf.,* pp. 19-39, 1998.
[24] A. Hossein, S. Vilas, M.B. Tahoori, and D. Kaeli, "Vulnerability Analysis of L2 Cache Elements to Single Event Upsets," *Proc. IEEE Design, Automation and Test in Europe (DATE '06),* pp. 1-6, 2006.
[25] W. Zhang, "Replication Cache: A Small Fully Associative Cache to Improve Data Cache Reliability" *IEEE Trans. Computers,* vol. 54, no. 12, pp. 1547-1555, Dec. 2005.
[26] H. Sun, N. Zheng, and T. Zhang, "Realization of L2 Cache Defect Tolerance Using Multi-Bit ECC," *Proc. IEEE 23th Int'l Symp. Defect and Fault Tolerance in VLSI Systems (DFT '08),* pp. 254-262, Oct. 2008.
[27] S. Kim and A.K. Somani, "Area Efficient Architectures for Information Integrity in Cache Memories," *Proc. Int'l Symp. Computer Architecture (ISCA '99),* pp. 246-255, May 1999.
[28] H. Asadi, V. Sridharan, M. Tahoori, and D. Kaeli, "Reliability Tradeoffs in Design of Cache Memories," *Proc. First Workshop Architectural Reliability,* 2005.
[29] H.J. Lee, "Immediate Soft Error Detection Using Pass Gate Logic for Content Addressable Memory," *Electronics Letters,* vol. 44, no. 4, pp. 269-270, 2008.
[30] S. Pontarelli, M. Ottavi, and A. Salsano, "Error Detection and Correction in Content Addressable Memories," *Proc. IEEE 25th Int'l Symp. Defect and Fault Tolerance in VLSI Systems (DFT '10),* Oct. 2010.
[31] B. Bloom, "Space/Time Tradeoffs in Hash Coding with Allowable Errors," *Comm. ACM,* vol. 13, no. 7, pp. 422-426, 1970.
[32] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol," *IEEE/ACM Trans. Networking,* vol. 8, no. 3, pp. 281-293, 1998.
[33] C. Estan and G. Varghese, "New Directions in Traffic Measurement and Accounting," *ACM SIGCOMM,* vol. 32, no. 4, pp. 323-336, Oct. 2002.
[34] S. Wilton and N. Jouppi, "An Enhanced Access and Cycle Time Model for On-Chip Caches," Technical Report 93/5, DEC Western Research Laboratory, 1994.
[35] B. Agrawal and T. Sherwood, "Ternary CAM Power and Delay Model: Extensions and Uses," *IEEE Trans. Very Large Scale Integration Systems,* vol. 16, no. 5, pp. 554-564, May 2008.
[36] B. Agrawal and T. Sherwood, "Modeling TCAM Power for Next Generation Network Devices," *Proc. Int'l Symp. Performance Analysis of Systems and Software (ISPASS '06),* pp. 120-129, 2006.
[37] H. Vandierendonck and K. De Bosschere, "XOR-Based Hash Functions," *IEEE Trans. Computers,* vol. 54, no. 7, pp. 800-812, July 2005.
[38] J.L. Henning, "SPEC CPU2000: Measuring CPU Performance in the New Millennium," *IEEE Computer,* vol. 33, no. 7, pp. 28-35, July 2000.
[39] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," *IEEE Computer,* vol. 35, no. 2, pp. 59-67, Feb. 2002.
[40] A. Kirsch and M. Mitzenmacher, "Less Hashing, Same Performance: Building a Better Bloom Filter," *Proc. 14th Ann. European Symp. Algorithms (ESA '06),* pp. 456-467, 2006.
[41] Cisco Catalyst 6500 Series Switches Data Sheet, http://www.cisco.com/en/US/products/hw/switches/ps708/products_data_sheets_list.html, 2012.



Salvatore Pontarelli received the master's degree in electronic engineering at the University of Bologna in 2000, and the PhD degree in microelectronics and telecommunications from the University of Rome Tor Vergata, in 2003. Currently, he is with the Department of Electronic Engineering at the same university. He has worked with the National Research Council, the Italian Space Agency, and the National Inter-University Consortium for Telecommunications and has been consultant for various Italian and European companies for projects related to digital design and to fault tolerance in digital systems. Since 2009, he has been working on the development and design of FPGA-based methods for high-speed network intrusion detection systems. His research interests include the development of highly reliable systems for space applications, error-detection and correction codes, fault detection and recovery for arithmetic circuits, use of post-CMOS technologies (in particular quantum-dot cellular automata) for the implementation of digital circuits at subnanometric integration scale.

Marco Ottavi (M'03-SM'10) received the PhD degree in microelectronics and telecommunications engineering from the University of Rome, "Tor Vergata," Italy, in 2004. He is currently a professor with the University of Rome "Tor Vergata," Italy, as the recipient of a "rientro dei cervelli" Fellowship awarded by the Italian Ministry of University and Research. Previously, he was a senior design engineer with AMD in Boxborough, Massachusetts, and held postdoctoral positions with Sandia National Labs in Albuquerque, New Mexico, and with the Electrical Communication Engineering Department of Northeastern University in Boston, Massachusetts. His research interests include yield and reliability modeling, test, design for tesatability, fault-tolerant architectures, and online testing and design of nanoscale circuits and systems. In these fields, he published five book chapters and more than 50 articles on archival journals and peer-reviewed conferences. Since December 2011, he is the chair of COST Action IC1103 "Manufacturable and Dependable Multicore Architectures at Nanoscale." He is a senior member of the IEEE.