By Using Bloom Filters Detecting and Correcting Errors In Cam

M.Manoj [1], R. Manoj Srinivasan [2], P.Suresh [3], T.Shanthi [4]

[1]M.E VLSI DESIGN, Akshaya College Of Engineering And Technology
[2]M.E VLSI DESIGN, Akshaya College Of Engineering And Technology
[3]M.E VLSI DESIGN, Akshaya College Of Engineering And Technology
[4]M.E VLSI DESIGN, Akshaya College Of Engineering And Technology

Abstract

The 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. 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 using a probabilistic structure called "Bloom filter,"which can signal if given data is present in the CAM. These filters permit to efficiently store and query the presence of data in a set. When a CAM suffers from SEU induced errors, the probabilistic nature of Bloom filters has consequence the so called false-positive effect. This paper proves 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 based area and power consumption.

1.Introduction

The content addressable memory (CAM) is an SRAM based memory capable of comparing the input data against the data stored in memory, providing the address of the matching data. CAMs with small dimensions are commonly used in translation look aside buffers (TLB), while large CAMs are used in systems must perform rapid searches within a large amount of data. One of the most used applications where CAMs are used is packet forwarding and classification in high-speed network systems. When a processor needs to read or write a location in the main memory, it has to check whether that memory location is in the cache. This is done 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. 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.

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. The collected charge can change the state of a circuit. These effects are well known for SRAM and DRAM memories. Information redundancy has been exploited by using error-detection and correction codes, technology and circuit solutions are aimed to increase the critical charge value. 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. 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 or at architectural level.

The effects of 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. This effect is commonly known as multibit upset (MBU). In this paper, we use SEU to refer to a generic radiation induced error, while we use the SBU and MBU terms to refer errors affecting one or more than one bit, respectively. In particular, we refer to an l-bit MBU for an SEU affecting up to l bits in the same word. While error-correcting codes (ECCs) can detect and correct errors in SRAM and DRAM, and can be easily extended to MBU they cannot directly used in associative memories such as CAMs or caches. 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.We propose a method that does not require modifications to the internal structure of the CAM, therefore it is preferable to add error-detection and correction features without compromising the internal

structure and the overall performance of the circuit. The aim of this paper is to focus on a generic CAM device that is different from cache memories is inherently fully associative. 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 two tasks: 1) stores a set of items in its memory and 2) quickly responds to a query about the presence of an item. An architecture using both CAM and Bloom filter could potentially be affected by two very different 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, can detect SEU induced errors occurring into the CAM. This paper also proposes a suitable algorithm, for 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 behaviour of a CAM that usually deletes and update its content, a well-known extension called counting Bloom filters is applied. Experiments performed on the realized system by using a cacti-based model shows the technique proposed in this paper introduces an 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.

2. Bloom Filters

Bloom Filter consists of several hash functions and a bit vector. A given N-bit address is hashed into k hash values using k different random hash functions. The output of each hash function is an m-bit index that addresses the 2m entry bit vector, where m is much smaller than N. Initially, the Bloom filter bit vector is zero. Whenever an N bit address is observed, it is hashed to the b vector and the bit value hashed by each m-bit index is set to one. When a query is to be made, the given N-bit address is hashed using the same hash functions and the bit values are read from the locations indexed by the m-bit hash value. If at least one of the bits is 0, it indicates that this address was definitely not observed before. This is called a true miss. Whereas, if all of the bit values are 1, the address may have been observed but with no guarantee, which is called a false hit. As the number of hash functions increases, the Bloom filter bit vector is polluted faster. On the other hand, the probability of finding a zero during a query is increased if more hash functions are used. The major drawback of the original Bloom filter is the high false hit rate as it can be quickly filled up with all 1's. Also, once a bit is set, there is no way to reset it. Thus, as more bits are set, the number of false hits increases. To address this issue, the counting Bloom filter was proposed for web cache sharing to provide capability of resetting entries in the filter. First, an array of counters is added along with the bit vector of the original Bloom Filter. Each L-bit counter has a one-to-one association with each bit in the bit vector. Queries to a counting Bloom filter are similar with a slight modification: when an address is entered, each m-bit hash index will increment its corresponding counter of the counter array in addition to setting the bit vector. Similarly, when an address is removed from the Bloom filter, each m-bit hash index will decrement its corresponding counter.

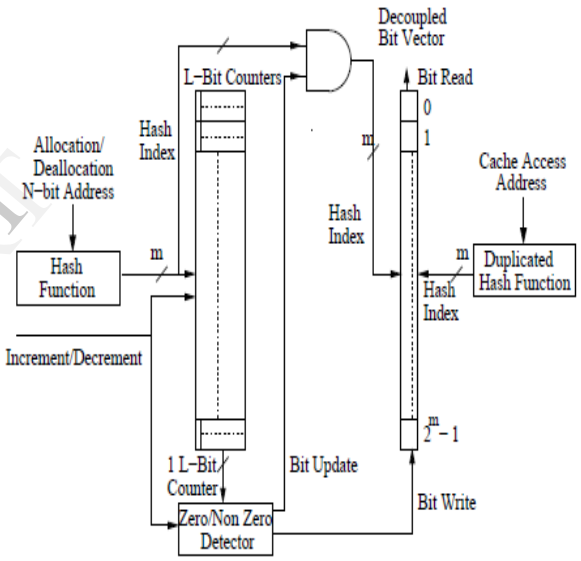

Fig.1. Segmented Bloom filter

If more than one hash indexes to the same location for a given address, the counter is incremented or decremented only once. Finally, when a counter is reduced to zero, its associated bit in the bit vector will be cleared.

3. Segmented Counting Bloom Filter

One application of the counting Bloom filter is to keep track of the line-fills and replacements of a cache and indicate whether an address is present in the cache. Query to a counting Bloom filter consumes less energy and quicker than accessing the entire cache. Ghosh et al. has shown a cache miss detection technique using a segmented counting Bloom filters. Their design redrawn contains the counter array (L bits per counter) decoupled from the bit vector with a duplicated hash function on the bit vector side. The cache line

fill/eviction addresses are sent to the counter array using one hash function while the cache request address from the processor is sent to the bit vector using a copy \ of the same hash function. The segmented Bloom filter design allows the counter array and bit vector to operate in separate physical locations.

There are several reasons for a segmented Bloom filter:

1) We only need the bit vector, which is smaller than the counter, to obtain the outcome of a query. Decoupling the bit vector enables faster and lower energy accesses to the Bloom Filter. Hence the result of a query issued from the core can be obtained by just looking up the bit vector.

2) The update to the counters is not time-critical with respect to the core. So, the segmented design allows the counter array to run at lower frequency than the bit vector. The vector part being smaller provides fast access time, whereas the larger counterpart runs at a lower frequency to save energy. The only additional overhead of the segmented design is the duplication of the hash function hardware. We now describe an innovative application of the segmented counting Bloom filter to avoid unnecessary cache way lookups.

4. Mechanism of Bloom Filter

Cache hierarchy has become a main consumer of both static and dynamic energy in processors. Even so, the trend in modern processor designs continues to increase both capacity and associativity to accommodate the ever-growing workloads and alleviate conflict misses. For processors employing highly associative caches, the energy consumption gets even worse as N-tag comparisons are needed for each parallel lookup of an N-way cache. In fact, most of the energy consumed for such lookups is redundant as the requested data can only be present in one particular way. This redundancy provides a good opportunity for saving dynamic energy.

We propose a technique called based on segmented counting Bloom filters to exploit these energy saving opportunities. Our scheme uses counting Bloom filters to efficiently skip the lookup of cache lines that do not contain the requested data to save significant energy in cache accesses. Bloom filters are simple, fast structures that can eliminate the need of performing associative lookup especially when the lookup address space is huge. They can replace the expensive set-associative tag matching with a simple bit vector that precisely identifies addresses that have not been observed before. This mechanism provides early detection of events to avoid an associative buffer lookup. This improves energy consumption significantly without adversely affecting performance given the efficient hardware structures.

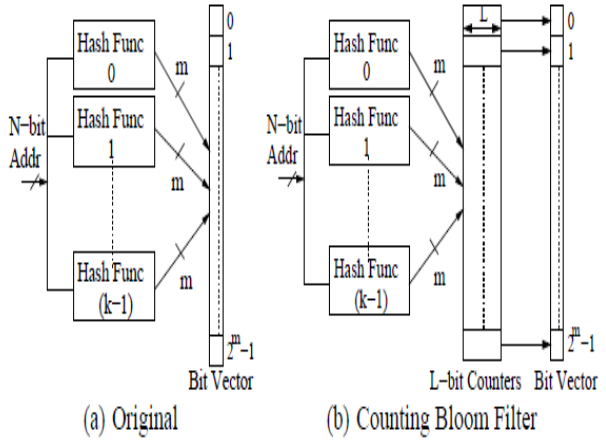

Fig.2. Bloom Filters

5. Related Work

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.

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. propose a methodology to produce a CAM structural architecture starting from a functional description of some high-level properties of the device. 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.

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 which degrade area occupancy and most of all the timing performances with respect to a non protected CAM. Moreover, the solution proposed is for match-line sense amplifiers although quite interesting, could be affected by issues both related to power consumption and noise immunity.

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.

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.

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 , where the performances of content addressable memory aided hash table are evaluated. However, Wan et al. 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 and power consumption in cache memories.

6. Architecture of Content Addressable Memories

Fig. 3 shows the schematic representation of a CAM. The input search word is an n-bit string which is 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 . The memory array of a CAM has a structure similar to a conventional SRAM, has an arrangement in rows and columns. In operational point of view, the write operation inside the CAM is 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 similar to the conventional SRAM cell, and a particle hitting the CAM produces similar consequence on the bits stored in the array

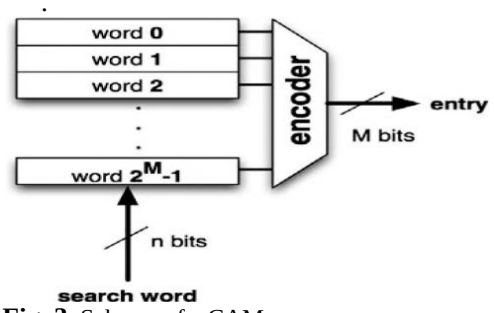

Fig. 3. Scheme of a CAM.

6.1 Consequences of a Soft Error in a CAM

In this section, we discuss the effects of soft errors, by isolating four possible cases:

(i). **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. 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.

(ii). Pseudo-HIT. Corrupted memory content corresponds to content. If this word is searched, the CAM gives response as the location in which the error has occurred. Fig. 2b. It should be noted that the same SEU can, therefore, produce both a pseudo-MISS and a pseudo-HIT effect.

(iii). 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. This case is re presented in Fig. 2c.

(iv). Wrong-HIT. This 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. 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 behaviour when an SEU occurs in the CAM memory.

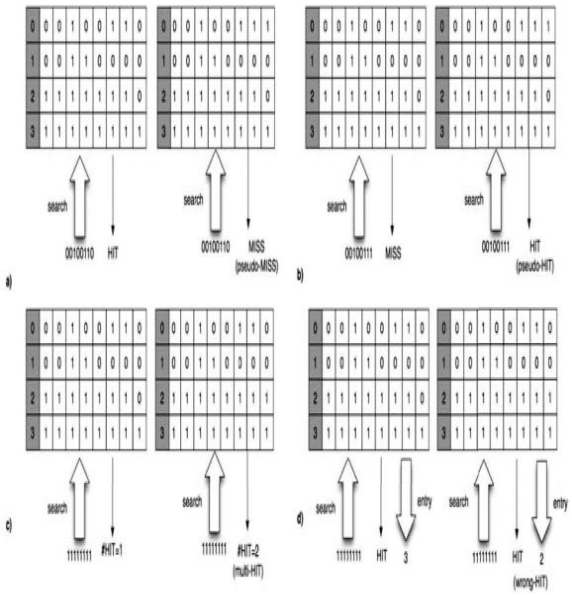

Fig. 4. Possible errors occurring in a CAM. Original and corrupted CAMs are presented side by side. In (a), the SEU hits a bits of entry 0 of the CAM. When the word is requested the CAM responds with a pseudo-MISS signal. In (b), the same SEU also produces a pseudo-HIT. In (c), a multi-HIT error is caused by the SEU affecting a bit of entry 2. In (d), the same error produces a wrong-HIT.

7. Overview on Bloom Filters

A Bloom filter 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 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 the equations needed to be 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 H1(x) . . .Hk(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,

$$result \leftarrow \min\{v(H_i(x))\},\ i \in \{1..k\}.$$

For a Bloom filter in which n elements are stored, the probability that a given bit in the filter is zero is given by

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

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 The probability of all of them being 1, which would cause the false positive, is then

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

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 > 2b _ 1 saturates the ith counter. After 2b_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 behaviour. The operation is modified as follows

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

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 ns ¼ ds=ke additional items have been stored in the filter. The false-positive probability of the counter can be, therefore, evaluated as

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

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

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

The probability of having ns saturated counters can be computed by using the binomial distribution

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

8. Error Detection and Correction in a Cam

This part, deals about how to detect and correct SEU induced errors in a CAM. Here we will focus on a solution that will not require substantial modifications

to existing CAM circuits. 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. Different from 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 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 .The parity check bits protection scheme is based on memory interleaving. 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 non codeword (rather than a wrong codeword) and since the CAM search words are always code words pseudo-HIT or multi-HIT errors will never occur. When an error hits an entry of the CAM, changing it into a non codeword, 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). The address is given 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 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. 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. 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.

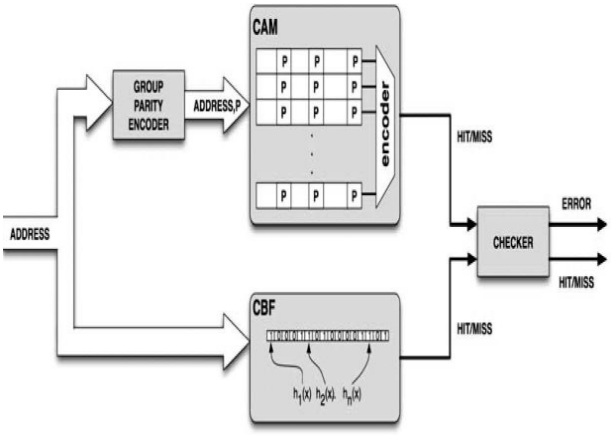

Fig. 5. Scheme of the proposed error-correction scheme for a CAM/CBF pair.

9. Simulation of Microprocessor Tag Array Protection

In this project sim-outorder processor simulator provided in Simple Scalar 3.0 and the SPEC INT2000 Benchmark suites .The modification of Simple Scalar 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 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 the miss rate (in percentage) for each kind of cache/TLB 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.

9.1. Sizing of Bloom Filter Auxiliary Cache

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.

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 2l _W noncodewords, whereW 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:

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

To limit the 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 Pfp, the probability that there are J different false positive in the set is given by the following equation:

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

The ratio J=K can be seen as the relative dimension of the auxiliary CAM with respect to the CAM to be protected. The trade off for minimization of the Overhead can be, therefore, performed by replacing in (8) Pfp with PN obtained from (9). For example, using a n=m ratio of 1=8 and a J=K ratio of 1=128 the value of PN is around 1E-5.

10. 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 error-correction capabilities, we suppose to use an interleaved factor of ID ¼ 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 ¼ 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.

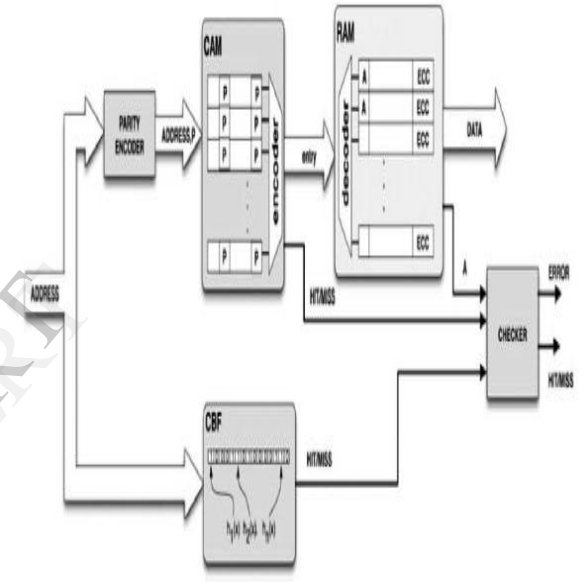

Fig.6.Error detection and correction for a CAM used in conjunction with a RAM.

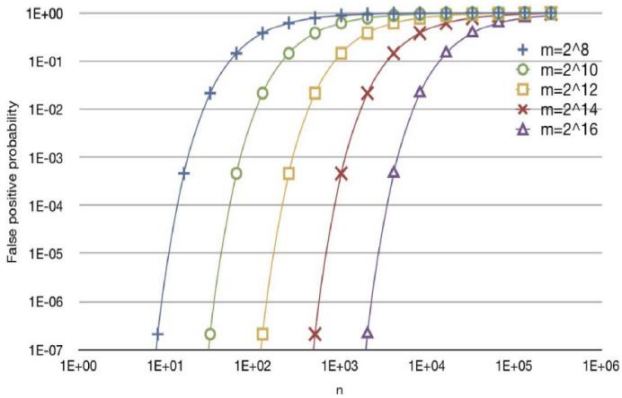

Fig.7. The false-positive probability Pfp as a function of n and m. An optimal number of hash functions k has been assumed.

It should be noticed that, even if the data presented 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.

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 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 M bits (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.

11. 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 these techniques for the protection of caches and TLB of a microprocessor have been reported.

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.