Enhancing Instruction TLB Resilience to Soft Errors

Alfonso Sánchez-Macián, Luis Alberto Aranda, Pedro Reviriego, *Senior Member, IEEE*, Vahdaneh Kiani, and Juan Antonio Maestro, *Senior Member, IEEE*

Abstract—A translation lookaside buffer (TLB) is a type of cache used to speed up the virtual to physical memory translation process. Instruction TLBs store virtual page numbers and their related physical page numbers for the last accessed pages of instruction memory. TLBs like other memories suffer soft errors that can corrupt their contents. A false positive due to an error produced in the virtual page number stored in the TLB may lead to a wrong translation and, consequently, the execution of a wrong instruction that can lead to a program hard fault or to data corruption. Parity or error correction codes have been proposed to provide protection for the TLB, but they require additional storage space. This paper presents some schemes to increase the instruction TLB resilience to this type of errors without requiring any extra storage space, by taking advantage of the spatial locality principle that takes place when executing a program.

Index Terms—Translation lookaside buffer, fault tolerance, error detection, reliability

1 Introduction

When using virtual memory, a translation process between virtual and physical addresses needs to be performed by the operating system. Checking the page table through page walks and computing the actual physical address may take a non-negligible amount of time. During the execution of a program, this process may affect system performance. Translation lookaside buffers are used to speed up the translation process by storing recently accessed virtual memory page numbers and their related physical page numbers. They also store control information related to these entries, such as a bit to indicate if an entry is valid or permission bits. A common way of implementing a TLB [1] consists of a content-addressable memory (CAM) that stores the virtual page numbers and an associated Random Access Memory (RAM) which keeps the physical page number for each CAM entry.

A bit flip in a valid entry of the CAM produces a change in the stored virtual page number which may cause false negatives or false positives. False negatives may impact the performance of the system as a new page walk is required. False positives may have a more harmful effect, producing silent data corruption, freezing the system or causing a hard fault as the program may try to execute a set of instructions different to the ones originally planned.

Bit flips may be caused by a number of reasons. Radiation effects over systems running on harsh environments may corrupt the data and produce single or even multiple errors. A single event upset (SEU) usually causes the modification of a single bit (Single Bit Upset –SBU), but multiple bit upsets (MBUs) can also occur [2], [3], especially when the linear energy transfer (LET) increases [4]. MBUs affect more than one bit, usually close to each other and, in many cases, they are adjacent [5]. The steady reduction in supply voltage to limit power consumption also decreases the charge needed by the different particles to cause this effect [6], [7]. In this type of harsh environments, programs running in embedded systems tend to be smaller and with a more deterministic behavior, and this typically favors the spatial locality, principle used in instruccion caches.

Different methods have been proposed to protect TLBs against errors. When using a parity bit [8], single errors can be detected in the TLB. A parity bit can be added to the CAM to detect errors in the virtual address page number and another one to the RAM for the physical page number. If an error is detected, the entry is invalidated. This produces a page walk to retrieve again the virtual to physical mapping. When the performance cost of this operation is high, Error Correction Codes (ECCs) have been proposed [9], [10]. To avoid the additional delay to check and correct the error, the ECC may be used in a backup copy [9] and correction and scrubbing refresh can be performed periodically. Alternatively, the ECC code can be adapted [10] to provide fast error detection and perform the correction operation only if the error was detected (or again, using a scrubbing mechanism).

This paper presents two schemes to enhance the instruction TLB resilience to errors. This is done by using an encoding mechanism (not requiring extra parity bits) that propagates errors in the virtual page number to the most significant bit (MSB) and takes advantage of the spatial (sequential) locality of the instructions being executed. The rest of the paper is organized as follows. Section 2 presents the different

• *The authors are with the Universidad Antonio de NebrijaC/Pirineos, 55, Madrid E-28040, Spain.*
E-mail: {asanchep, laranda, previrie, vkiani, jmaestro}@nebrija.es.

Manuscript received 6 Apr. 2018; revised 25 Sept. 2018; accepted 2 Oct. 2018. Date of publication 9 Oct. 2018; date of current version 22 Jan. 2019.
(Corresponding author: Alfonso Sánchez-Macián.)
Recommended for acceptance by Dr. P. Girard.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TC.2018.2874467

types of errors that can affect these memories and their consequences when focusing on instruction TLBs. It also provides information about existing protection techniques applied to TLBs and, in general, to CAM memories. In Section 3, the proposed techniques are described and the scenarios of applicability are stated. In Section 4, an evaluation of the techniques is performed by using traces from different CPU2006 SPEC benchmarck [11] programs and validating the delay and area overhead. Finally, Section 5 summarizes the conclusions.

2 Preliminaries

2.1 Errors in Instruction TLBs

As previously stated, a SEU may corrupt a single bit or may affect multiple bits from the same word. When the TLB is implemented as a CAM with an associated RAM, the error can disturb either of those structures.

Bit flips in the virtual page information stored in the CAM part of the TLB may produce a false positive when the queried tag matches the corrupted entry. The result of this false positive is that a program executes a wrong set of instructions as they are retrieved from a different (but valid) physical page, with several possible consequences (e.g., hard fault, silent data corruption, system freeze). Another effect derived from a bit flip may be a false negative, where the virtual page being checked no longer matches any entry of the table, and a miss is produced, which may affect the performance of the processor.

An error in the CAM values stored in the TLB not always produces an undesired behavior. If the program does not access the specific virtual page in error, or the entry is overwritten due to the TLB replacement algorithm, the error will be masked.

An error may also occur in the RAM. This would modify the physical address, which may now point to a valid (but different) physical page or to a page that is not allocated to the process.

The techniques presented in this paper provide protection to the virtual page entries stored in the CAM. RAM may be protected using a traditional scheme.

2.2 Fault Tolerance in TLBs

There are several studies that have proposed different approaches to protect TLBs from errors. Additionally, as pointed out in the previous section, the virtual page information of the TLBs can be implemented as a CAM. This section examines the solutions proposed to implement fault tolerance in TLBs and in CAM memories.

The simplest approach to provide fault tolerance in TLBs consists of including a parity bit [8] to detect single errors. In general, if a memory with integrated parity is used, the additional storage requirements are already satisfied and the delay produced by the parity calculation is acceptable. If the memory does not provide parity support, an overhead in terms of storage has to be considered. When an error is detected in the TLB, the entry is invalidated and the information is retrieved from main memory (or other level caches/TLBs if available).

In those cases where a miss has a significant impact in system performance, ECC codes are proposed. In [9], a copy of the TLB is kept in a backup storage. This copy is protected by an ECC. When a miss occurs, the TLB copy is decoded and the original TLB is rewritten with the correct values. A second check is performed to validate if there is a match or if a real miss was present. A scrubbing mechanism can also be used to periodically update the original copy with the decoded value of the TLB backup. The work in [10] also provides an ECC that protects both the CAM and RAM information with a single code. This code is adapted so that error detection can be performed in a faster way using a subset of the parity bits of the code, and then correction is only performed if an error was previously detected (by using now the total set of bits). Scrubbing can also be applied in this case. These methods require extra storage space (for the parity bits of each word) and may significatively increase the delay when an error is detected.

In [12], the CAM can be configured to use parity or a key duplication solution. For bit-oriented search, a duplicated tag memory (in complementary form) with its own matching logic is produced. A hit is recorded if any of the two elements provides a match, and a second register records a fault if the results of both copies do not agree. Results are compared to detect a possible faulty element. It does not only tries to avoid false positives, but also false negatives. The additional required logic and tag memory imply that storage overhead is duplicated for the CAM part of the memory.

The approach in [13] uses a shortened ECC code that increases the Hamming distance between words to 4, so single errors can be corrected and double errors produce a miss. Ad-hoc comparators are then used to detect the exact number of mismatches, instead of using the traditional matchlines. When there is a difference of one bit, a match is identified. For a higher number of differences, a miss is triggered. The shortened ECC code uses a higher number of parity bits that need to be stored in the memory. Additionally, matchlines need to be customized.

A similar approach that also uses ECCs and modifies the matchline to detect one-bit differences is used in [14]. In this case, the ECC produces a minimum Hamming distance of 3 so it assumes that a single bit difference is still a match. The design implements a modified NAND-type, dual matchline organization. Although it is useful in scenarios where false misses need also to be avoided, it has the same disadvantages as the previous study.

Keys can also be split into two parts and stored in two sub-CAM memories as in [15]. A concept of "close-hit" is used when a match is found in one of the CAMs, but a miss is generated in the other. The entry for the one that produced the miss is then checked (and corrected if needed) by using parity information stored in these CAMs. Again, extra memory requirements (for the parity ECC bits) and possibly delays are introduced, especially for nearby addresses where most of the information matches and the check needs to be performed even without error.

Finally, Bloom filters can also be used as a parallel structure to the CAM memory [16] and use them to detect if an error is present avoiding false positives. Additional storage is required for the Bloom filter in this solution.

3 Proposed Schemes

Both schemes presented in this paper make use of the spatial locality principle to increase the instruction TLB resilience to faults produced by false positives. This is

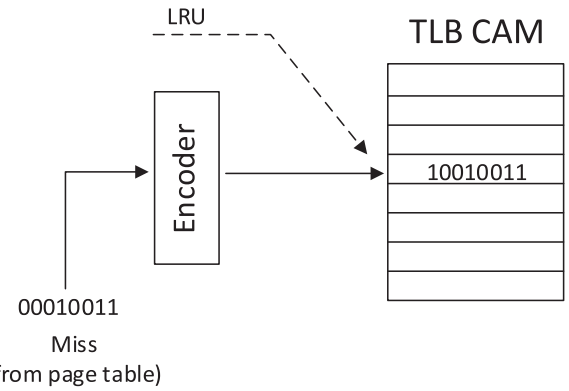

Fig. 1. Encoding process in the TLB.

achieved by protecting the virtual page information stored in the table. The schemes do not require extra storage as they encode the virtual page numbers in a way that errors are propagated to the most significant bit.

The first scheme is designed for architectures where no parity bit is available in the underlying memory and it is capable of reducing and, in most cases, eliminating false positives caused by single bit upsets.

The second approach is oriented to systems where the TLB virtual page information is stored in a memory that already includes parity bit support. In those cases, the parity bit provides single (and in general odd) error detection capability. The proposed scheme adds protection against false positives produced by double-adjacent errors.

3.1 TLB Without Parity Bit

As explained before, the first scheme is designed for architectures where no parity bit is available in the underlying memory.

When a TLB miss happens and the valid entry is fetched from the page table, it is not directly stored (as usually happens) but an encoding process is previously performed on the Virtual Page Number (VPN). In this way, what is stored in the TLB CAM is not the VPN, but an encoded version of it, VPN′. This encoding process is straightforward and quick and consists in calculating the XOR of all the bits that form the VPN. So, basically it is like calculating the parity bit, but instead of storing it in an additional extra bit, the result of that parity is overwritten in the MSB of the VPN. In this way, no extra bits are needed [17].

Using this scheme, VPN′ would be calculated as:

$$\begin{aligned} &\mathrm{VPN'[i]} = \mathrm{VPN[i]} \forall \neq \mathrm{MSB} \\ &\mathrm{VPN'[MSB]} = \mathrm{VPN[MSB]} \oplus \mathrm{VPN[MSB-1]} \oplus \ldots \oplus \mathrm{VPN[0]}. \end{aligned} \tag{1}$$

The whole process is depicted in Fig. 1. The interesting property of this encoding is that all "nearby" values generated through it, and therefore stored in the TLB CAM, would be separated by at least a Hamming distance of 2. Therefore, any single error affecting such codification would not create false positives. By "nearby" we mean all page numbers that differ in less than 2^{w-2} positions from each other, where w is the width of the VPN (see Section 3.3). Thus, for an 8-bit VPN, $\mathrm{VPN}\,0\,(00000000)$ and $\mathrm{VPN}\,63\,(00111111)$ are nearby values as $63 - 0 < 2^{8-2} = 64$. However, Hamming distance 2 cannot be guaranteed for a higher difference between VPNs such as $126(01111110)$ and $190(10111110)$ where 190-126 does not hold the condition "< 64". Hopefully, and thanks to the spatial locality principle, all VPNs in the same program should be close enough to meet the condition above. This will be assessed in Section 4 with some experiments on the CPU2006 SPEC benchmark.

a)

Lower VPNs	Encoded VPNs′
00000000	00000000
00000001	10000001
00000010	10000010
00000011	00000011

b)

Higher VPNs	Encoded VPNs′
11111100	01111100
11111101	11111101
11111110	11111101
11111111	01111111

c)

Intemediate VPNs	Encoded VPNs′
01111110	01111110
01111111	11111111
10000000	10000000
10000001	00000001

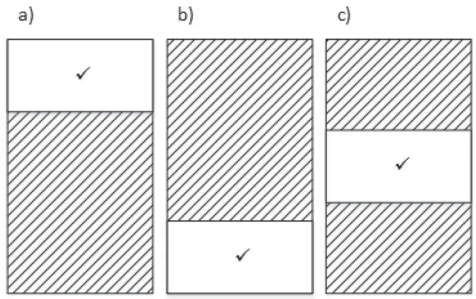

Fig. 2. VPN positions in the virtual address space.

Let us examine some examples to check how the encoding process works. 8-bit VPNs will be used for the sake of clarity, but the whole technique is applicable to any number of bits.

As can be seen, the encoding technique produces VPN′ words that are separated, at least by a Hamming 2 distance, if the original VPN words are close enough. However, if they are not close enough, then two different VPN words could produce the VPN′ that are just separated by Hamming 1 distance, and therefore a single error could create false positives in such scenario. For example, VPN words 0000000 and 10000001 would produce VPN′ words 0000000 and 00000001 respectively. A single error in the LSB of any of them would be undetectable, since it would produce another valid VPN′ in the representation. Therefore, for the technique to work, all VPN that are simultaneously stored in the TLB should be contained in a subset of the virtual address space such i) it is at most 25 percent of the total space and ii) it is consecutive. Fig. 2 shows the maximum valid subsets for cases a), b) and c) where the TLB instructions must be contained for the technique to work.

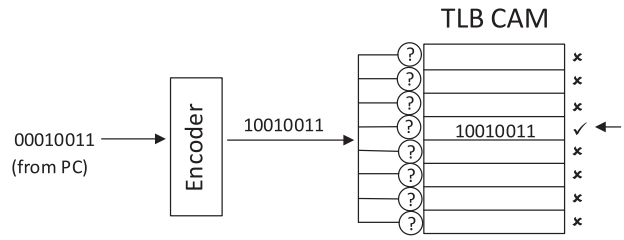

Fig. 3. Regular TLB hit mechanism.

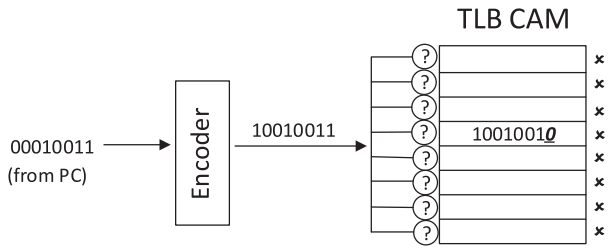

Fig. 4. Error producing a false negative.

This constraint does not mean that the whole program that is being executed has to be contained in at most 25 percent of the virtual address space. On the contrary, the whole program may contain an arbitrarily large number of instructions, as long as the subet of instructions that are stored in the TLB at any given moment meets the conditions above. This constraint, as explained before, is very reasonable, since due to the spatial locality principle, instructions recently executed (which are those in the TLB) should occupy close virtual addresses. To summarize, by applying the proposed coding method, all VPN′ words stored in the TLB would have a separation of at least Hamming 2 distance. Now, let us examine how the whole technique handles errors taking advantage of this property.

When a VPN arrives for virtual to physical translation, it is initially encoded with the presented method. After this, the standard comparison process in the TLB CAM is performed, in order to look for a match. Notice that this comparison is performed on the encoded space, and in that way VPN′ words stored in the TLB never need to be decoded back to their original value, thus avoiding a large delay overhead. The TLB search process can produce any of the following outcomes:

A) There was not any error in the TLB CAM values. In this case, a normal hit / miss situation would happen. In other words, the outcome of this process would be the same as in the traditional case where VPN words are stored without any kind of encoding. This situation is depicted in Fig. 3. In that particular example, $\mathrm{VPN}\,00010011$, coming from the Program Counter (PC), is first encoded into $\mathrm{VPN}'10010011$, and then it is compared with the stored encoded VPN′. In this case, a match is produced.

B) There was an error in a TLB CAM value. If this happens, the following scenarios may occur:

1. If the incoming VPN was already stored in the TLB (and therefore it was a match) and the error has affected it, then the match would not happen, producing a false negative. This false negative would imply a page walk with the corresponding performance penalty, but the execution of the program would be correct. This scenario is shown in Fig. 4. The encoded VPN′ 10010011 is compared with the content of the TLB entries. Initially, that address had been stored in the table, but an error on its LSB has changed it into 10010010. No match is produced due to this false negative, which has no consequence on the program execution integrity.
2. If the incoming VPN was not stored in the TLB (and therefore it was a miss), no error can transform any of the TLB values into that incoming VPN. In other words, no false positives may happen, which is the worst case scenario since it can produce a wrong control flow of the program. This is avoided thanks to the presented encoding technique. Since all the encoded VPN′ word stored in the TLB are separated at least by a Hamming 2 distance, any VPN′ affected by a SEU would be transformed into a Hamming 1 distance word that is not valid in the representation, and therefore it will not match with any incoming value. An example of this is shown in Fig. 5. $\mathrm{VPN}\,00010010$ was previously stored in the TLB as encoded $\mathrm{VPN}'\,00010010$ (in this particular case VPN and VPN′ coincide). Then, the incoming VPN 00010011 is encoded as 10010011 and compared with the TLB content. Original VPNs (00010010 and 00010011) have a Hamming distance 1, and therefore an error in the LSB would convert one into the other. But encoded VPNs′ (00010010 and 10010011) have a Hamming distance 2 thanks to that encoding process. So, an error in the LSB would not make them coincide, and therefore no false positive can happen.

Given that errors are unfrequent events, the overhead penalty due to scenario B.1 (false negatives) is negligible with respect to the standard number of TLB misses that happen in the program execution. And since all false positives are avoided (scenario B.2), the execution flow of the program is not compromised, saving a potentially large overhead needed to handle the error recovery.

3.2 TLB with Parity Bit

The technique presented in the previous section is a good approach to errors in systems where the TLB CAM does not have any kind of protection. However, it is quite common to have TLBs that do incorporate a parity bit as a basic protection mechanism against soft errors. In this case, the system can handle single errors by default. However, the technique presented in the previous section can still be used in this case, increasing the resilience of the system. In

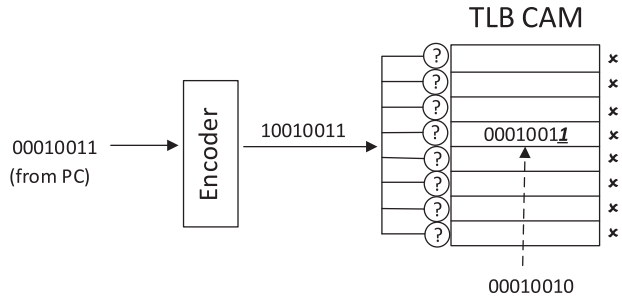

Fig. 5. No false positives can happen.

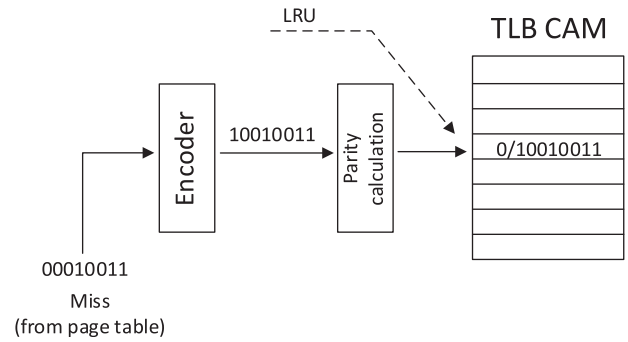

Fig. 6. Encoding process and parity calculation.

particular, combining a parity bit with this technique would avoid not only single errors, but also double adjacent and triple adjacent errors (this last one also due to the parity). Multiple adjacent errors are those produced by a single event, and have the property of being located in adjacent positions. They can be produced by different causes. For instance, in harsh environments, as Space, radiation effects can cause Multiple Cell Upsets [5], [18]. The cells affected by an MCU are physically close and in many cases adjacent [19]. This effect is becoming more and more common as technology scales.

The strategy of the technique is quite similar to the previous case with no parity bit, but adapted to handle double adjacent errors. In the same way, incoming VPN are encoded into modified VPN′ words before they are stored in the TLB CAM. The procedure is as follows (see Fig. 6).

First the VPN′ is calculated by overwriting the MSB with the accumulated XOR, but this time instead of using all the bits for the calculation, only half of them will be utilized, in this way.

If the number of bits in the VPN is odd, then:

$$\begin{aligned} &\mathrm{VPN}'[\mathrm{i}] = \mathrm{VPN}[\mathrm{i}] \forall \neq \mathrm{MSB} \\ &\mathrm{VPN}'[\mathrm{MSB}] = \mathrm{VPN}[\mathrm{MSB}] \oplus \mathrm{VPN}[\mathrm{MSB}-2] \oplus \ldots \oplus \mathrm{VPN}[0]. \end{aligned} \tag{2}$$

If the number of bits in the VPN is even, then:

$$\begin{aligned} &\mathrm{VPN}'[\mathrm{i}] = \mathrm{VPN}[\mathrm{i}] \forall \neq \mathrm{MSB} \\ &\mathrm{VPN}'[\mathrm{MSB}] = \mathrm{VPN}[\mathrm{MSB}] \oplus \mathrm{VPN}[\mathrm{MSB}-2] \oplus \ldots \oplus \mathrm{VPN}[1]. \end{aligned} \tag{3}$$

So by doing this, any bit participating in the VPN′ XOR calculation would have two neighbours that do not participate, and vice versa (except of course the MSB and LSB that have only one neighbor). By applying this technique, any double adjacent error would always affect a bit in the VPN′ XOR operation and a bit outsite the VPN′ XOR operation. This means that any adjacent double error in the encoded word will produce an invalid word in this representation. In other words, no adjacent double error will be able to transform a valid TLB VPN′ into another valid VPN′, thus avoiding any kind of false positive. Again, as happened in the previous case, all the existing VPNs in the TLB must be close enough, but that should be the normal situation.

After this, the standard parity bit would then be calculated on the encoded VPN′ word and also stored in the TLB. With this parity bit, all single and triple errors (and in general any odd number of errors) would be avoided. As a final remark, the place where the parity bit is stored is also important. If the parity bit is stored in any structure ouside the TLB CAM, then there will not be any problem in this respect. In the other way, if the parity bit is going to be stored together with the VPN′ bits, in the TLB CAM array, then it is important to store it in an adjacent position to the MSB. This is relevant in order to handle double adjacent errors in the parity bit and in any other bit. As stated before, in order for the technique to work, bits in the VPN′ XOR operation should always be placed adjacent to bits outside the VPN′ XOR operation. Since the parity bit is not taking part in the VPN′ XOR operation (in fact it is calculated later), then it needs to be placed adjacent to a bit that is always part of that VPN′ XOR operations, as the MSB. Notice that the LSB not always takes part in the XOR operation. According to expression (2), the LSB (i.e., $\mathrm{VPN}[\,0\,]$) would be considered if there is an odd number of bits in the VPN, but not in the case of an even number.

This is an example of the two stages to calculate the final encoding (VPN $\rightarrow$ VPN′ $\rightarrow$ Parity bit), where the bits in the VPN that take part in the VPN′ XOR calculation has been marked in bold.

Lower VPNs	Encoded VPNs′	With Parity Bit
00000**1**0	10000110	1/10000110
00000**1**1	10000111	0/10000111

As can be seen, no adjacent double error can convert the first VPN′ into the second one and vice versa.

The technique works in a similar way to the case without parity bit. When a VPN arrives at the TLB, the VPN′ and parity bit are calculated. Then a standard compare process happens in the TLB CAM in order to identify a potential match. Again, if no errors are present in the TLB CAM, a hit or miss happens just in the same way as if no encoding is used. If a number of odd errors or a double adjacent error happen, then the affected VPN′ will never be transformed into any other valid VPN′, and therefore no false positives may happen. It may still occur that a TLB VPN′ word that matches an incoming VPN is affected by an error, leading to a false negative. But as explained before, the performance penalty of this situation is negligible. Notice that the "nearby" concept is different in this case, where by this attribute we mean all page numbers that differ in less than 2^{w-3} units from each other, where w is the width of the VPN. Thus, for an 8-bit VPN, $\mathrm{VPN}\,0\,(00000000)$ and $\mathrm{VPN}\,31\,(00111111)$ are nearby values as $31 - 0 < 2^{8-3} = 32$. However, avoiding double adjacent errors cannot be guaranteed for a higher difference between VPNs such as $126\,(01111110)$ and $158\,(10011110)$ where 158-126 does not hold the condition "< 32".

Notice that we have supposed, for the sake of simplicity, a distribution where physically adjacent bits are adjacent in the binary pattern too. However, the technique is also applicable to other configurations using different bits to calculate the parity, as long as the restrictions presented in this paper are met.

3.3 Explanation of "Nearby" Restrictions

In previous sections, a "nearby" concept was introduced to identify the maximum address distance that any two VPN

in the TLB can be from each other for the schemes to work properly. This section tries to clarify how these values are calculated.

In Section 3.1, a scheme was defined so neighbor VPNs are encoded to be separated by a Hamming distance of 2. The way this is done is by storing the parity of all bits in the most significant bit. Any two VPNs that share the MSB value (e.g., 00000000 and 01000000) have, originally, at least a Hamming distance of 1. Two different cases appear in this situation:

- The original Hamming distance is odd. If this is the case, one VPN must have an odd number of 1s (e.g., 01000000) and the other one an even number of 1s (e.g., 00000000). The most sensitive scenario in this situation is when the Hamming distance is 1, because just flipping one bit will turn one VPN into the other. Anyway, when encoding is performed in this case, the MSB will change in one of the cases (e.g., 11000000) and will remain the same in the other (e.g., 00000000), increasing the Hamming distance by one. Therefore, if the initial Hamming distance was 1, it would be 2 after encoding, or $+1$ in any other case of odd Hamming distance.
- The original Hamming distance is even. In this case, both VPN will have an odd number of 1s (e.g., 00000111 and 00000001) or both will have an even number of 1s (e.g., 01100000 and 01101100). In the first scenario, both MSB will turn to 1 when the VPN are encoded, and in the second scenario both MSB will remain as 0 when encoded. This means that the Hamming distance will remain the same as the original after the encoding process, being Hamming distance 2 the minimal in this case.

For those VPNs where the MSB differs, the following possibilities may occur:

- No other bit differs between the VPNs (e.g., 00000000 and 10000000). In this case, the Hamming distance is 1. When encoding, the Hamming distance will remain the same. These values are separated between them by 2^{w-1} (2^7 in the example) where w is the VPN width.
- A second bit is different between the two original VPNs so Hamming distance is 2 (e.g., 00000000 and 10000001). In this case, both VPNs have an even number of 1s or both have an odd number of ones. Thus, MSB will match after encoding and distance will be reduced to 1. Considering that the MSB is different, the closest values with an additional bit mismatch are those where the second most significant bit has the opposite value of the MSB (e.g., 10000000 and 01000000 that are encoded into 10000000 and 11000000). In those cases, the original values of the VPNs are separated by $2^{w-1} - 2^{w-2} = 2^{w-2}$ (2^6 in the example) where w is the VPN width.
- In the rest of cases, the original Hamming distance is 3 or higher so, in the best case, it is reduced to 2 when encoding the MSB.

Consequently, the maximum separation between VPNs that guarantees the requirements for Section 3.1 is 2^{w-2}.

For the scheme in Section 3.2, the MSB is encoded by storing the parity of the bits of the same type of position (even or odd) as the MSB in the original VPN (including the MSB itself). Then, a parity bit of the encoded word is added. Single and triple error will be detected by the parity bit. False positives due to double adjacent errors must be avoided by the encoding process. Thus, a distance higher that 2 must be guaranteed for adjacent positions. With this in mind, the maximum separation between VPNs that are stored in the TLB at a specific moment can be explained as follows:

- Any two VPNs with an original Hamming distance of 1 may, after encoding, keep the same distance (e.g., $01000000, 00000000 \rightarrow 01000000,\ 00000000$) or increase their distance to 2 ($00100000, 00000000 \rightarrow 10100000,\ 00000000$). This second case implies that the two bits that differ are in non-adjacent positions (MSB and another odd position in the example) and can be ignored. In the first case where Hamming distance is kept, parity bit will increase this distance to 2. As the parity bit is placed next to the MSB, the only option that the other difference is in an adjacent bit, is that both VPNs have a different MSB (e.g., $10000000, 00000000 \rightarrow \mathbf{1}/\mathbf{1}0000000, \mathbf{0}/\mathbf{0}0000000$). The original VPNs are, consequently, separated by 2^{w-1} where w is the VPN width.
- For an original Hamming distance of 2, if the difference has to be in adjacent bits they will affect an even and an odd bit (e.g., 00000000, 01100000). Thus, distance may increase after encoding (e.g., $00000000, 01100000 \rightarrow\ 00000000,\ 11100000$) or, if one of the bits is the MSB, distance will remain as 2 (e.g., $00000000, 11000000\ \ \rightarrow 00000000,\ 11000000$). In the latter case, when adding the parity, distance is still 2. The two possible patterns affecting MSB and next bit are 00 versus 11 and 01 versus 10. The smaller separation between these VPNs is $2^{w-1} - 2^{w-2} = 2^{w-2}$ (2^6 in the example) where w is the VPN width.
- For a Hamming distance of 3, it is possible that the difference is reduced to 2 and that it applies to adjacent bits. For this to happen, the MSB must be different in the original VPNs and it must be flipped in one of them when encoding. As the remaining differences must affect adjacent positions, they are placed in an even bit and an odd bit. An example of this would be VPNs 00000011 and 10000000 which are encoded (with parity) into $1/\ 10000011$ and $1/\ 10000000$. The smallest separation between original VPNs fulfilling these requirements corresponds to those where the two other different bits are contiguous to the MSB and with an opposite value to that MSB, e.g., 10000000 and 01100000. Separation between these VPNs is $2^{w-1} - 2^{w-2} - 2^{w-3} = 2^{w-3}$ (2^5 in the example) where w is the VPN width.

Consequently, the maximum separation between VPNs that guarantees the requirements for Section 3.2 is 2^{w-3}.

4 Evaluation

To compare the original and proposed TLB designs, a fault injection campaign was performed over a subset of programs from the CPU2006 SPEC benchmarks.

Additionally, the overhead in terms of area and delay was calculated for the extra circuitry of the proposed designs when implemented into a Field Programmable Gate Array (FPGA).

4.1 Fault Injection Campaign

A dynamic instrumentation tool, Pin [20], was used to extract the virtual memory address request information of a subset of programs from the CPU2006 SPEC benchmark, when executing them in an Intel x64 architecture running Windows 7. This information was then used to simulate the behavior of the original non-protected and parity-protected TLB vs. the proposed schemes.

The virtual memory addresses obtained from Pin were truncated to obtain the VPN (removing the 12 bits corresponding to the offset for a page size of 4 KB). This VPN was used to feed the simulated TLBs in a step by step process. Error injection was performed by flipping specific bits of a single entry of the TLB in a particular moment. The possible number of injections is extremely high as it depends not only on the size of the TLB or the VPN bits stored in each entry, but also on the number of instructions run during the program execution, as the error may be produced at any point of the program. To obtain a valid percentage of injections that produce false positives, statistical fault injection is needed.

The scenario was designed as follows:

- The TLB is configured with 8 entries (like the Level 0 TLB in the AMD Zen microarchitecture [21] or the default lowRISC TLB implementation [22]).
- An LRU (Least Recently Used) algorithm is used to replace the entries when the TLB is full.
- Each entry holds a VPN of 32 bits. Notice that, for this particular version of the operating system, only 44 bits are assigned to the program virtual memory address.
- A set of 27,000 injection experiments is performed. A randomly generated triplet with the "entry number", "bit position(s)" and "trace line" identifies every injection experiment, defining when and where the error injection is applied for a specific execution of a benchmark.
- Traces are processed step by step and, when the "trace line" defined in the injection triplet is reached, a bit flip is produced in the "bit position(s)" of the value stored in the "entry number".
- Execution is continued until a false positive is produced or the entry is overwritten.

The fault injection campaign was performed for two TLB scenarios:

- Scenario A: TLB without parity bit (no initial protection). Single errors are injected. Results are compared when protecting the TLB with the first technique, explained in Section 3.1.
- Scenario B: TLB initially protected with parity bit. Single errors, double-adjacent and triple-adjacent errors are injected. Results are compared when protecting the TLB with the second technique, explained in Section 3.1.

Ten benchmark programs were selected, five integer benchmarks (429.mcf, 445.gobmk, 456.hmmer, 464.h264ref and 471.omnetpp) and five floating point benchmarks (433.milc, 444.namd, 453.povray, 470.lbm and 483.xalancbmk).

TABLE 1
Benchmark Results for Single Error Injection in Scenario A (Non-Protected and MSB Encoding Schemes), with 8 Entries

Benchmark	Scheme	Percentage of False Positives
429.mcf	Non-protected	0.66%
	MSB encoding	0%
455.gobmk	Non-protected	0.30%
	MSB encoding	0%
456.hmmer	Non-protected	0.98%
	MSB encoding	0%
464.h264ref	Non-protected	0.96%
	MSB encoding	0%
471.omnetpp	Non-protected	0.88%
	MSB encoding	0%
433.milc	Non-protected	1.49%
	MSB encoding	0%
444.namd	Non-protected	2.2%
	MSB encoding	0%
453.povray	Non-protected	0.45%
	MSB encoding	0%
470.lbm	Non-protected	0.27%
	MSB encoding	0%
483.xalancbmk	Non-protected	0.37%
	MSB encoding	0%

To handle double matches due to errors we have considered a priority encoder where the first match in the table is selected. Notice that this is an acceptable solution as the injections are produced randomly and can affect any of the entries in the table. This scheme has been applied in the same way to the original TLBs and the protected ones so the effect is the same for both.

Results for Scenario A are presented in Table 1. For the non-protected TLB, the injection experiments (for single errors) produced false positives during the simulation in all the benchmarks, ranging from 73 in 470.lbm to 596 in 444.namd (0.27 to 2.2 percent of the total number of 27,000 injections). The number of false positives is reduced to 0 when the first scheme proposed in Section 3.1 is applied.

Results for Scenario B are presented in Table 2 for double-adjacent error injection experiments in the same ten benchmarks. Single and triple error injections were also performed and, as expected, no false positives were produced in either scheme due to the parity bit used. Other MBU patterns have not been tested, as the proposed technique does not provide additional, in general, benefits for those patterns when compared to the base parity-protected TLB. The double-adjacent errors produce false positives in the parity-protected scenario in all the benchmarks, ranging from 43 in 470.lbm to 423 in 444.namd (0.16 to 1.57 percent of the total number of 27,000 injections). For the sake of comparison with the previous results, the experiments were executed on fully associative CAMs. However, the technique is applicable to other ways of organizing the structure. Again, the proposed scheme shows that it is able to avoid this kind of false positives, due to double-adjacent errors, for these benchmarks.

TABLE 2
Benchmark Results for Double-Adjacent Error Injection in Scenario B (Parity-Protected and MSB Encoding with Parity Schemes), with 8 Entries

Benchmark	Scheme	Percentage of False Positives
429.mcf	Parity-protected MSB encoding with parity	0.45% 0
455.gobmk	Parity-protected MSB encoding with parity	0.23% 0
456.hmmer	Parity-protected MSB encoding with parity	0.41% 0
464.h264ref	Parity-protected MSB encoding with parity	0.87% 0
471.omnetpp	Parity-protected MSB encoding with parity	0.55% 0
433.milc	Parity-protected MSB encoding with parity	1.04% 0
444.namd	Parity-protected MSB encoding with parity	1.57% 0
453.povray	Parity-protected MSB encoding with parity	0.4% 0
470.lbm	Parity-protected MSB encoding with parity	0.16% 0
483.xalancbmk	Parity-protected MSB encoding with parity	0.51% 0

As explained in Section 3.3, this technique will work as long as the "nearby" restriction is fulfilled. Once the technique was checked with a TLB size of 8, by performing error injection, a TLB with 128 entries has also been analyzed to verify how the scheme scales to larger TLBs. Again, the execution of each of the benchmark programs has been simulated to check which bit flips in the entries of the TLB would produce a false positive. Single bit flips are considered in the TLB without parity and double-adjacent bit flips in the parity-protected TLB.

Tables 3 and 4 show the results of this analysis. The percentage of false positives in the unprotected scenarios varies from 0.47 to 2.25 percent for single errors in the TLB without parity and between 0.44 and 1.54 percent for double-adjacent errors. For the protected scenarios, no false positives are produced. This means that the proposed protection techniques are as effective in the 128-entry case as in the previous 8-entry scenerario. Therefore, the effectiveness of the solution seems to be independent on the actual TLB size. Notice that, even if the "nearby" restriction is not met, the probability of having two encoded remote addresses within the restricted Hamming distance existing simultaneously in the TLB is still very low. For those applications with low spatial locality, some false positives may occur, but it should still behave better than the original TLB scheme.

4.2 Area and Delay Overhead

The additional logic required by the two techniques presented in this paper has been implemented on a Xilinx FPGA (integrated in a Nexys4 DDR board). To calculate the parity, 32 VPN bits were considered (for a total of 44 virtual address bits). Basically, the implemented circuit corresponds to a XOR parity tree of 32 bits in the first case (Fig. 7, all the bits) and 16 bits in the second scheme (Fig. 8, odd bits). The results have been compared to different

TABLE 3
Benchmark Results in Percentage for False Positives in Scenario A (Non-Protected and MSB Encoding Schemes), with 128 Entries

Benchmark	Scheme	Percentage of False Positives
429.mcf	Non-protected MSB encoding	1.16% 0%
455.gobmk	Non-protected MSB encoding	0.47% 0%
456.hmmer	Non-protected MSB encoding	1.12% 0%
464.h264ref	Non-protected MSB encoding	1.51% 0%
471.omnetpp	Non-protected MSB encoding	1.27% 0%
433.milc	Non-protected MSB encoding	2.25% 0%
444.namd	Non-protected MSB encoding	1.78% 0%
453.povray	Non-protected MSB encoding	0.89% 0%
470.lbm	Non-protected MSB encoding	1.17% 0%
483.xalancbmk	Non-protected MSB encoding	1.12% 0%

TABLE 4
Benchmark Results in Percentage for False Positives in Scenario B (Parity-Protected and MSB Encoding with Parity Schemes), with 128 Entries

Benchmark	Scheme	Percentage of False Positives
429.mcf	Parity-protected MSB encoding with parity	1.54% 0%
455.gobmk	Parity-protected MSB encoding with parity	0.44% 0%
456.hmmer	Parity-protected MSB encoding with parity	0.81% 0%
464.h264ref	Parity-protected MSB encoding with parity	1.31% 0%
471.omnetpp	Parity-protected MSB encoding with parity	0.96% 0%
433.milc	Parity-protected MSB encoding with parity	1.50% 0%
444.namd	Parity-protected MSB encoding with parity	1.24% 0%
453.povray	Parity-protected MSB encoding with parity	0.84% 0%
470.lbm	Parity-protected MSB encoding with parity	1.15% 0%
483.xalancbmk	Parity-protected MSB encoding with parity	1.11% 0%

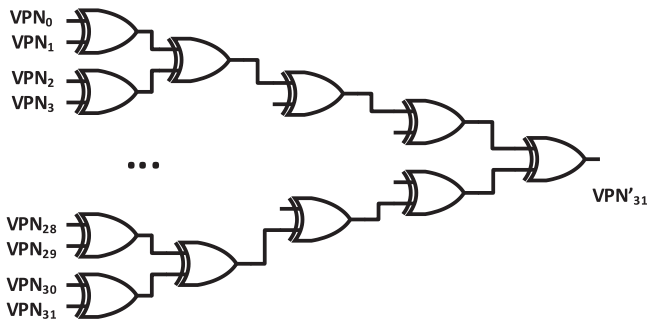

Fig. 7. XOR Parity tree of 32 bits for the first scheme.

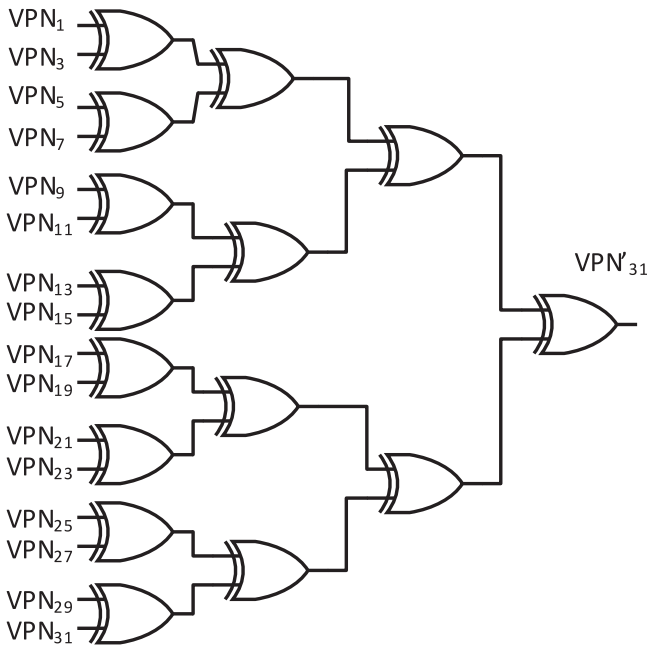

Fig. 8. XOR Parity tree of 16 bits for the second scheme.

approaches. They have been checked against traditional schemes that deliver similar protection: the circuits of a traditional parity that provides single error detection (SED), and a scheme with with two parity bits (one for odd data bits and the other one for even data bits), providing Single error and double-adjacent error detection (SED-DAED). Additionally, two other approaches used in previous studies were also included: a Hamming Single Error Correction (SEC) ECC, and a Single Error Correction, Double Error Detection (SEC-DED) ECC.

As expected, the overhead of the schemes is low as shown in Table 5. Only 7 LUTs are required for the first scheme, and 3 for the second scheme. In terms of delay, the system requires only to traverse 2 LUT levels and it is similar to that of a traditional parity calculation. Notice that these schemes do not require decoding (only encoding results are presented) and, consequently, they save resources and avoid delay penalties (when compared to the related traditional parity approaches). The SEC and SEC-DED encoders require at least 19 and 21 LUTs respectively and have to traverse 2 LUT levels as well. Moreover, additional resources are needed for decoding (65 LUTs for SEC and 84 bits for SEC-DED) and extra delay is introduced (4 LUT levels). Decoders for these two reference schemes can be replaced by special memory matchlines as pointed out in previous works, but this may imply additional costs and it is not feasible when using FPGAs.

TABLE 5
Implementation of the Additional Logic of the Proposed and Reference Schemes in a Xilinx FPGA

Scheme	Utilization (LUTS) Enc/Dec	Lut Levels Enc/Dec	Protection
Parity	7 / 7	2 / 2	SED
SEC ECC	19 / 65	2 / 4	SEC
MSB encoding	*7 / 0*	*2 / 0*	*SE avoidance*
Parities for odd / even bits	6/2	7/3	SED-DAED
SEC-DED ECC	21 / 84	2 / 4	SEC-DED
MSB encoding with parity	*3 / 0*	*2 / 0*	*SE, DAE AND TE avoidance*

TABLE 6
Storage Requirements of the Additional Logic of the Proposed and Reference Schemes in a Xilinx FPGA

Scheme	Storage overhead	Protection
Parity	3.13%	SED
SEC ECC	18.75%	SEC
MSB encoding	0%	SE avoidance
Parities for odd / even bits	6.25%	SED-DAED
SEC-DED ECC	21.88%	SEC-DED
MSB encoding with parity	0%	*SE, DAE AND TE avoidance*

In terms of storage overhead (Table 6), the proposed schemes do not require extra bits. The reference solutions have an overhead of 3.13 percent for traditional parity, 6.25 percent for the parities of odd/even bits, 18.75 percent (6 parity bits) for SEC or 21.88 percent (7 parity bits) for SEC-DED.

As a reference, 238 LUTs are used by the 8-entry iTLB default implementation of a lowRISC [22] microprocessor in the same Nexys4 DDR board. Thus, there is an increase of 1-3 percent in LUT utilization for the proposed schemes.

5 Conclusions

In this paper, two techniques to deal with soft errors in TLBs have been presented. The first technique considers the case where TLB does not have any kind of initial protection. Thanks to this technique, no single errors can affect the program execution in the benchmarks tested, avoiding all kind of false positives. Besides, the measured overhead is very limited, with a small increase in area utilization and delay. Being single upsets the majority of errors produced in an architecture, the presented technique is an effective way to handle them.

The second technique has been designed to work with TLBs protected with a parity bit mechanism, something usual in modern architectures. With this parity bit, all single errors could be detected. However, by using the proposed technique, double adjacent errors can also be avoided (and triple adjacent errors thanks to the parity). This provides an extra level of protection, again with a very limited overhead.

It is important to point out that the presented techniques make use of the spatial locality principle of the instruction

TLBs. For those applications with low spatial locality, the advantages may not be as good (not detecting all false positives), but it should still behave better than the original TLB scheme.

Notice that our initial target is oriented to provide fault tolerance to soft microprocessors embedded in systems used in harsh environments, but the principles behind both techniques (storing the parity over the data bits) can also be used on other applications, structures (such as CAMs in routers) or scenarios (e.g., [17]), avoiding or reducing undesired effects due to bit errors. Future work will focus on this.

References

[1] M. Fertig, U. Gaertner, N. Hagspiel, and E. Pfeffer, "Translation lookaside buffer and related method and program product utilized for virtual addresses," U.S. Patent No. 8,166,239, Apr. 24, 2012.

[2] N. Seifert, B. Gill, K. Foley, and P. Relangi, "Multi-cell upset probabilities of 45 nm high-k + metal gate SRAM devices in terrestrial and space environments," in *Proc. IEEE Int. Rel. Phys. Symp.*, 2008, pp. 181–186.

[3] D. Radaelli, H. Puchner, Skip Wong, and S. Daniel, "Investigation of multi-bit upsets in a 150 nm technology SRAM device," *IEEE Trans. Nuclear Sci.*, vol. 52, no. 6, pp. 2433–2437, Dec. 2005, doi: 10.1109/TNS.2005.860675.

[4] R. Naseer and J. Draper, "Parallel double error correcting code design to mitigate multi-bit upsets in SRAMs," in *Proc. 34th Eur. Solid-State Circuits Conf.*, 2008, pp. 222–225, doi: 10.1109/ESSCIRC.2008.4681832.

[5] R. K. Lawrence and A. T. Kelly, "Single event effect induced multiple-cell upsets in a commercial 90 nm CMOS digital technology," *IEEE Trans. Nuclear Sci.*, vol. 55, no. 6, pp. 3367–3374, Dec. 2008, doi: 10.1109/TNS.2008.2005981.

[6] A. R. Alameldeen, Z. Chishti, C. Wilkerson, W. Wu, and S. L. Lu, "Adaptive cache design to enable reliable low-voltage operation," *IEEE Trans. Comput.*, vol. 60, no. 1, pp. 50–63, Jan. 2011, doi: 10.1109/TC.2010.207.

[7] C. W. Slayman, "Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations," *IEEE Trans. Device Materials Rel.*, vol. 5, no. 3, pp. 397–404, Sep. 2005, doi: 10.1109/TDMR.2005.856487.

[8] T. W. Griffith Jr and L. E. Thatcher, "TLB parity error recovery," U.S. Patent No. 6,901,540, May 31, 2005.

[9] S. M. Lang, "Processor fault tolerance through translation lookaside buffer refresh," US 8429135 B1, Apr. 23, 2013.

[10] A. Sánchez-Macián, P. Reviriego, and J. A. Maestro, "Combined modular key and data error protection for content-addressable memories," *IEEE Trans. Comput.*, vol. 66, no. 6, pp. 1085–1090, Jun. 1, 2017, doi: 10.1109/TC.2016.2633998.

[11] J. L. Henning. "SPEC CPU2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006, doi: 10.1145/1186736.1186737.

[12] J. C. Lo, "Fault-tolerant content addressable memory," in *Proc. IEEE Int. Conf. Comput. Des.: VLSI Comput. Processors*, 1993, pp. 193–196.

[13] K. Pagiamtzis, N. Azizi, and F. N. Najm, "A soft-error tolerant content-addressable memory (CAM) using an error-correcting-match scheme," in *Proc. IEEE Custom Integrated Circuits Conf.*, 2006, pp. 301–304.

[14] A. Efthymiou, "An error tolerant CAM with nand match-line organization," in *Proc. 23rd ACM Int. Conf. Great Lakes Symp. VLSI*, pp. 257–262, 2013

[15] L. D. Hung, M. Goshima, and S. Sakai, "Mitigating soft errors in highly associative cache with CAM-based tag," in *Proc. Int. Conf. Comput. Des.*, 2005, pp. 342–347.

[16] S. Pontarelli and M. Ottavi, "Error detection and correction in content addressable memories by using bloom filters," *IEEE Trans. Comput.*, vol. 62, no. 6, pp. 1111–1126, Jun. 2013.

[17] J. A. Martínez, J. A. Maestro, and P. Reviriego, "A scheme to improve the intrinsic error detection of the instruction set architecture," *IEEE Comput. Archit. Letters*, vol. 16, no. 2, pp. 103–106, Jul.-Dec. 1 2017. doi: 10.1109/LCA.2016.2623628

[18] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba, "Impact of scaling on neutron-induced soft error in SRAMs from a 250 nm to a 22 nm design rule," *IEEE Trans. Electron Devices*, vol. 57, no. 7, pp. 1527–1538, Jul. 2010.

[19] S. Satoh, Y. Tosaka, and S. A. Wender, "Geometric effect of multiple-bit soft errors induced by cosmic ray neutrons on DRAM's," *IEEE Electron Device Lett.*, vol. 21, no. 6, pp. 310–312, Jun. 2000.

[20] C. Luk, et al., "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proc. ACM SIGPLAN Conf. Program. Language Des. Implementation*, Jun. 2005, pp. 190–200.

[21] M. Clark, "A new ×86 core architecture for the next generation of computing," in *Proc. IEEE Hot Chips 28th Symp.*, 2016, pp. 1–19, doi: 10.1109/HOTCHIPS.2016.7936224

[22] A. Bradbury, G. Ferris, and R. Mullins, "Tagged memory and minion cores in the lowRISC SoC," Computer Laboratory, University of Cambridge, Cambridge, UK, lowRISC-MEMO 2014-001, Dec. 2014. [Online]. Available: http://www.lowrisc.org/downloads/lowRISC-memo-2014-001.pdf.

Alfonso Sánchez-Macián received the MSc and PhD degrees in telecommunications engineering from the Universidad Politécnica de Madrid, Madrid, Spain, in 2000 and 2007, respectively. He has worked as a lecturer and a researcher with several universities, such as the Universidad Politécnica de Madrid; the IT Innovation Centre, the University of Southampton, Southampton, United Kingdom; and the Universidad Antonio de Nebrija, Madrid, where he is currently part of the ARIES Research Center. He previously worked in numerous national and multinational companies as project manager and senior consultant for IT projects. His current research interests include fault-tolerance and reliability, performance evaluation of communication networks and knowledge representation and reasoning in distributed systems.

Luis Alberto Aranda received the BSc degree in industrial engineering and the MSc in robotics from the Universidad Carlos III de Madrid, in 2012, 2015, respectvely, and the PhD degree (Honors) in industrial engineering from the Universidad Antonio de Nebrija, in 2018. He worked as a project engineer with Zeus Creative Technologies S.L. developing various computer vision projects from 2013 to 2014. He was responsible for both hardware and software design and implementation. He is currently with the ARIES Research Center, Universidad Antonio de Nebrija, Madrid. He is the author of several technical publications, both in journals and international conferences. His research interests include reconfigurable computing for space applications, computer vision and robotics.

Pedro Reviriego (A'03–M'04-SM'15) received the MSc and PhD Hons. degrees in telecommunications engineering from the Technical University of Madrid, Madrid, Spain, in 1994 and 1997, respectively. From 1997 to 2000, he was an R&D engineer with Teldat, Madrid, working on router im plementation. In 2000, he joined Massana to work on the development of Ethernet transceivers. During 2003, he was a visiting professor with the Universidad Carlos III de Madrid, Leganés, Spain. From 2004 to 2007, he was a distinguished member of the technical staff with LSI Corporation, working on the development of Ethernet transceivers. He is currently with the ARIES Research Center, Universidad Antonio de Nebrija, Madrid. He is the author of numerous papers in international conference proceedings and journals. He has also participated in IEEE 802.3 standardization activities. His research interests include fault-tolerant systems, communication networks, and the design of physical-layer communication devices. He is a senior member of the IEEE.

Vahdaneh Kiani received the BSc degree in computer engineering from South Tehran Branch, Islamic Azad University, Tehran, Iran, in 2011, and the MSc degree in computer engineering form the Department of Computer Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran, in 2015. She is currently with the ARIES Research Center, Universidad Antonio de Nebrija, Madrid. Her research interests include the areas of digital circuit design, memory systems and fault tolerance in microprocessors.

Juan Antonio Maestro (M'07-SM'15) received the MSc degree in physics and the PhD degree in computer science from the Universidad Complutense de Madrid, Madrid, Spain, in 1994 and 1999, respectively. He has served both as a lecturer and a researcher with several universities, such as the Universidad Complutense de Madrid; the Universidad Nacional de Educación a Distancia (Open University), Madrid; Saint Louis University, Madrid; and the Universidad Antonio de Nebrija, Madrid, where he currently directs the ARIES Research Center. His current activities are oriented to the space field, with several projects on reliability and radiation protection, as well as collaborations with the European Space Agency. Aside from this, he has worked for several multinational companies, managing projects as a Project Management Professional and organizing support departments. He is the author of numerous technical publications, both in journals and international conferences. His research interest include high-level synthesis and cosynthesis, signal processing, real-time systems, fault tolerance, and reliability.