Silent Data Corruption and Reliability Risks due to Faults Affecting High Performance Microprocessors' Caches

Silent Data Corruption and Reliability Risks due to Faults Affecting High Performance Microprocessors' Caches*

M. Omaña
DEI, U. of Bologna
Bologna, Italy
martin.omana@unibo.it

A. Manfredi**
DEI, U. of Bologna
Bologna, Italy

C. Metra
DEI, U. of Bologna
Bologna, Italy
cecilia.metra@unibo.it

R. Locatelli
Intel Corp.
Pisa, Italy
riccardo.locatelli@intel.com

M. Chiavacci**
Intel Corp.
Pisa, Italy

S. Petrucci
Intel Corp.
Pisa, Italy
stefano.petrucci@intel.com

***Abstract*— Error Correcting Codes (ECCs) are frequently adopted to guarantee the correct operation in the field of caches of high performance microprocessors. They require the addition of proper encoding/decoding blocks (referred to as *checkers*) to the cache array. The occurrence of faults affecting such *checkers* has been typically neglected so far, due to their limited area compared to the cache array. This may be no longer acceptable, due to the increasing likelihood of faults possibly affecting microprocessors implemented by deeply scaled technologies, and due to the increasing requirements in terms of reliability of several applications *(e.g.,* data centers, autonomous vehicles, unmanned robots, etc.). Based on these considerations, in this paper we analyze the effects of bridging faults possibly affecting the ECCs' *checkers*, for two frequently adopted kinds of ECCs. We will show that the 68% (or the 61%) of BFs possibly affecting the considered ECCs' *checkers* are critical, since they may either inhibit the ECC correction ability of incorrect words read from the cache, or introduce errors in otherwise correct words read from the cache, with consequent risks for silent data corruption and microprocessor reliability. The remaining 32% (or 39%) of BFs may remain latent and accumulate with following faults or aging conditions affecting the cache, with consequent future risks for silent data corruption and microprocessor reliability. We then introduce a possible scheme to detect on line the occurrence of critical BFs that, compared to an alternate solution presented in the literature, features significantly lower impact on the ECC *checker* delay and area.**

Keywords*— *Silent Data Corruption, Error Correcting Codes, Microprocessor, Reliability, Cache.

I. INTRODUCTION

Reliability is becoming a main constraint for an increasing variety of application fields, such as autonomous vehicles, unmanned robots, healthcare, finance, etc. [1, 2, 3, 4]. It results in strong reliability requirements for the high performance microprocessors adopted for such applications [5]. Software and/or hardware protection approaches [5, 6, 7, 8, 9] are consequently typically useed to protect high performance microprocessors with respect to faults and aging conditions possibly affecting their operation in the field. Despite their adoption, silent data corruption (*i.e.,* the occurrence of data corruption with no error indication provided by the adopted protection techniques) has been reported as likely [10], and constitutes nowadays one of the main reliability challenges, especially for microprocessors of data centers [11].

Of course, the microprocessor reliability strongly depends on its caches' reliable operation, which is usually guaranteed by using Error Correcting Codes (ECCs) [12, 13, 14].

As known, ECCs mandate the addition of proper encoding/decoding blocks (hereinafter referred to as *checkers*) to the cache array. The occurrence of faults affecting such *checkers* has been typically neglected so far, due to their limited area compared to the cache array. This may be no longer acceptable, due to the increasing likelihood of faults possibly affecting microprocessors implemented by deeply scaled technologies, and due to the increasing requirements in terms of reliability of several applications (*e.g.,* data centers, autonomous vehicles, unmanned robots, etc.) [6, 10, 11].

In fact, as an example, we can expect that faults affecting the component blocks of ECCs' *checkers* may result in an output incorrect word, even if the original word read from the cache was error free. In this case, the *checker* would give an incorrect word to the cache output, that would be propagated throughout the system, with consequent risks for silent data corruption and microprocessor reliability. Other faults may instead not affect the cache output word, but constitute a future risk for silent data corruption and microprocessor reliability, in case of accumulation with following faults and/or aging conditions, such as Bias Temperature Instability - BTI [15, 16].

Based on these considerations, in this paper we analyze the effects of resistive bridging faults (BFs) possibly affecting the ECCs' *checkers* during their operation in the field. Preliminary analyses of this kind were introduced in [17].

As for single event transients (SETs) possibly affecting *checkers*, due to the *checkers*' combinational structure, the likelihood that they propagate till the *checker* outputs, be sampled by *checker* outputs' sampling elements and give rise to Single Event Upsets (SEUs) may be considered limited [18]. Moreover, proper design strategies for *checkers'* outputs' sampling blocks have been proposed (*e.g.,* those in [19]), that can be used to avoid that unlikely *checkers*' outputs' SETs result in SEUs. Therefore, SETs possibly affecting ECCs' *checkers* have not been included in our performed analyses.

As representative case studies, we will consider *checkers* for two widely adopted ECCs for caches: the Single Error

* This work has been partially supported by Intel Corporation (Pisa, Italy) and by the Italian National Resilience and Recovery Plan (PNRR) through the National Center for HPC, Big Data and Quantum Computing.

** At the time of this research, A. Manfredi was with the Univ. of Bologna, and M. Chiavacci was with Intel Corp., Pisa.

Correcting – Double Error Detecting (SEC-DED) Hsiao ECC [12], and the Double Error Correcting (DEC) Orthogonal Latin Square (OLS) ECC [14].

We will show that the 68% (61%) of BFs possibly affecting the considered SEC-DED Hsiao (OLS) ECC *checker* are critical, since they may either inhibit the ECC correction ability of incorrect words read from the cache, or introduce errors in otherwise correct words read from the cache, with consequent risks for silent data corruption and microprocessor reliability. The remaining 32% (or 39%) of BFs possibly affecting the considered SEC-DED Hsiao (OLS) ECC *checker* may remain latent and accumulate with following faults or aging conditions, with consequent future risks for silent data corruption and microprocessor reliability.

Some solutions have been presented in [20-22] to prevent the catastrophic consequences of faults possibly affecting ECC *checkers*. In particular, in [20] the adoption of differential EXOR gates to implement ECCs' *checkers* has been proposed. However, such an approach does not guarantee the detection of all critical faults possibly affecting all blocks composing the cache *checkers*. Instead, in [21, 22], the Authors propose the adoption of normal *checkers* (*i.e., checkers* of the kind in [23]) to implement the *checkers* of Hamming, BCH and Cyclic ECCs. In particular, the information bits produced at the output of the ECC decoding block are first re-encoded (by using an additional encoder), in order to regenerate the word read from the cache. Then, the regenerated word is subtracted (modulo 2) from the word read from the cache, in order to produce an error message. In case of a fault-free *checker*, the error message resulting from such a subtraction should be equal to the word generated by the ECC syndrome decoder block inside the *checker*. Based on this property, the Authors propose to detect the presence of faults affecting ECCs' *checkers* by using a two-rail code checker to compare the generated error message with the word at the ECC syndrome decoder output [22]. These solutions are effective in detecting faults affecting the *checkers*, but they imply a significant impact on the ECC *checker* delay (which may exceed the 100%, depending on the considered ECC). They also require non-negligible costs in terms of area.

Based on these limitations of existing solutions, we here propose a novel lower-cost approach to detect, during the microprocessor in-field operation, the occurrence of critical BFs constituting a risk for silent data corruption and microprocessor reliability. Compared to the alternative solution in [22], our detector enables significantly lower impact on the ECC *checkers'* delay (up to the -60%), and area (up to the -27%).

It should be noted that our detection approach can be adopted also for *checkers* of ECCs different from those considered here as realistic examples.

The remainder of this paper is organized as follows. In Section II, we present the block structure and implementation of the *checkers* for the two considered case study ECCs. In Section III, we report the results of extensive electrical level simulations performed to analyze the effects of bridging faults possibly affecting such *checkers*. In Section IV, we introduce our detection approach and compare its costs to those of the alternate solution in [22]. Finally, in Section V, we draw some conclusive remarks.

II. Block Structure and Implementation of the Considered ECC Checkers

As representatives case studies, we consider *checkers* for two widely adopted ECCs for caches: 1) the Single Error Correcting – Double Error Detecting (SEC-DED) Hsiao ECC [12], whose *checker* conventional structure is represented in Fig. 1(a); 2) the Double Error Correcting (DEC) Orthogonal Latin Square (OLS) ECC [14], whose *checker* conventional structure is shown in Fig. 1(b).

As an example, for both ECC *checkers* in Fig. 1 we considered the case of 16 information bits ($d_0 \ldots d_{15}$). For this case, the SEC-DED *checker* (Fig. 1(a)) requires the generation of 6 check bits [12], while the DEC *checker* (Fig. 1(b)) mandates the generation of 16 check bits [14]. We implemented both *checkers* considering a 16nm standard CMOS technology, with power supply voltage V_{DD}=0.8V.

As for the SEC-DED *checker* in Fig. 1(a), its Encoder (*Enc*) receives the 16 information bits ($d_0 \ldots d_{15}$) and generates 6 check bits ($c_0 \ldots c_5$), as described in [12]. The resulting word ($d_0 \ldots d_{15}$ $c_0 \ldots c_5$) is stored in the cache.

The SEC-DED *checker* includes also a Decoder, that receives the word read from the cache, and gives as output the corrected information bits ($dc_0 \ldots dc_{15}$), together with signals (E, DE) that indicate the presence of no error (E, DE) = (0, 0), single error (E, DE) = (1, 0) or double error (E, DE) = (1, 1) in the word read from the cache. In Fig. 1, the word read from the cache is denoted by ($d'_0 \ldots d'_{15}$ $c'_0 \ldots c'_5$) to indicate that it may differ from the correct word ($d_0 \ldots d_{15}$ $c_0 \ldots c_5$), due to faults (*e.g.,* soft errors) affecting the cache. In case of no error, it is ($d'_0 \ldots d'_{15}$ $c'_0 \ldots c'_5$) = ($d_0 \ldots d_{15}$ $c_0 \ldots c_5$).

As known, the Decoder consists of a Syndrome Generator (*SG*), which receives the word read from the cache ($d'_0 \ldots d'_{15}$

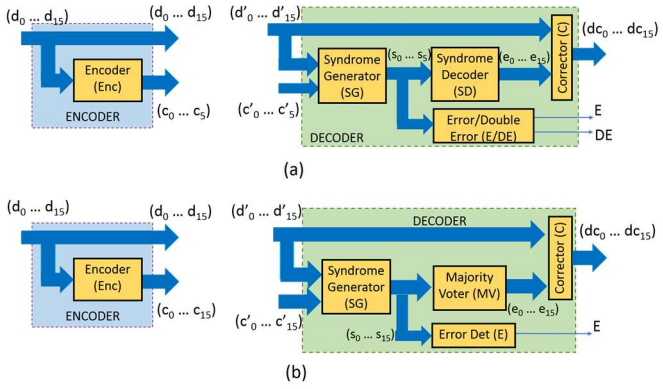

Fig. 1. Schematic representation of *checkers'* conventional structure for the considered: a) Single Error Correcting – Double Error Detecting (SEC-DED) Hsiao ECC [12]; b) Double Error Correcting (DEC) Orthogonal Latin Square (OLS) ECC [14].

$c'_0 \ldots c'_5$) and generates six syndrome bits ($s_0 \ldots s_5$). In case of no error in ($d'_0 \ldots d'_{15}$ $c'_0 \ldots c'_5$), all $s_i = 0$ (i=0 …5), otherwise (in case of single or double errors) at least one $s_i = 1$. The *SG* block (Fig. 1(a)) has been implemented as in [12].

The Decoder also presents a Syndrome Decoder (*SD*), that receives the syndrome bits ($s_0 \ldots s_5$) and generates 16 error bits ($e_0 \ldots e_{15}$) indicating the position of the erroneous bit in the information bits read from the cache ($d'_0 \ldots d'_{15}$). In case of no error, it is $e_k = 0$ (k=0…15), while if bit d'_k (k=0…15) is erroneous, then $e_k = 1$ (k=0…15). We considered the *SD* block implemented as in [12].

The Decoder presents a Corrector block (*C* in Fig. 1(a)) that receives the information bits read from the cache ($d'_0 \ldots d'_{15}$), as well as the error bits ($e_0 \ldots e_{15}$) from *SD*, and generates the corrected information bits ($dc_0 \ldots dc_{15}$). It has been implemented by EXOR gates.

Finally, the Decoder also presents an Error/Double Error block (*E/DE*) that receives the syndrome bits and generates signals (E, DE) described above.

As for the DEC *checker* in Fig. 1(b), its Encoder (*Enc*) receives the 16 information bits ($d_0 \ldots d_{15}$) and generates 16 check bits ($c_0 \ldots c_{15}$), according to [14]. The resulting word ($d_0 \ldots d_{15}$ $c_0 \ldots c_{15}$) is stored in the cache.

The Decoder of the DEC *checker* receives the words read from the cache ($d'_0 \ldots d'_{15}$ $c'_0 \ldots c'_{15}$), and gives to its output the corrected information bits ($dc_0 \ldots dc_{15}$) [14], together with an error signal E indicating the presence of no error (E = 0), or single/double errors (E = 1) in the word read from the cache.

The Decoder includes a Syndrome Generator (*SG*), which receives the word read from the cache and generates 16 syndrome bits ($s_0 \ldots s_{15}$). In case of no error in ($d'_0 \ldots d'_{15}$ $c'_0 \ldots c'_{15}$) all $s_i = 0$ (i=0…15), otherwise at least one $s_i = 1$. It is implemented using EXOR trees, that receive the information bits ($d'_0 \ldots d'_{15}$) and regenerate the check bits ($rc'_0 \ldots rc'_{15}$), as shown in [14]. The syndrome bits ($s_0 \ldots s_{15}$) are obtained by comparing (by 2-input EXORs) each rc'_i (i=0…15) bit with the respective check bit read from the cache c'_i (i=0…15).

The Decoder includes also a Majority Voter (*MV*) that receives the syndrome bits ($s_0 \ldots s_{15}$) and generates 16 error bits ($e_0 \ldots e_{15}$) [14], that indicate the position(s) of the erroneous bit(s) in the information bits ($d'_0 \ldots d'_{15}$). In case of no error, it is $e_i = 0$ (i=0…15), while in the presence of one (or two) error(s) on d_k (k=0…15), for instance on d_n (and d_j), it is $e_n = 1$ ($e_n = 1$ and $e_j = 1$). We implemented the *MV* block as in [14].

The *C* block receives the information bits ($d'_0 \ldots d'_{15}$) and the error bits ($e_0 \ldots e_{15}$), and generates the corrected information bits ($dc_0 \ldots dc_{15}$) = ($d_0 \ldots d_{15}$). We implemented the *C* block of the DEC OLS ECC *checker* (Fig. 1(b)) as the *C* block of the SEC-DED *checker* described before.

Finally, the *Error Det* block in Fig. 1(b) has been implemented as a simple 2-input OR tree receiving the syndrome bits ($s_0 \ldots s_{15}$) and generating the error signal E.

III. Checker Bridging Faults' Effects

In this Section, we report the results of the fault simulations that we have performed by means of HSPICE [24], in order to evaluate the effects of BFs possibly affecting the ECC *checkers* described in Section II.

In particular, for each *checker* block, we have considered all possible BFs, each with values of connecting resistance (RB) in the interval [0Ω,100kΩ]. We have also assumed faults occurring one at a time, as usual for on-line testing [25].

As for the considered SEC-DED ECC *checker,* the obtained results are summarized in Fig. 2.

We can see that, out of the considered 1190 *checker* BFs, the 68% (808 BFs) result in an incorrect word given to the cache output, thus they are critical faults that can cause silent data corruption and compromise the microprocessor reliability.

The remaining 32% of *checker* BFs (382 BFs) either result in an incorrect word written into the cache that will be corrected by the Decoder when read, or do not have any effect on the *checker* operation, thus they remain present in the *checker* as latent faults. However, latent faults may accumulate with following faults and/or aging conditions (such as BTI), thus constituting a future risk for silent data corruption and microprocessor reliability.

More specifically, from Fig. 2 we can see that, for the Encoder, all its possible 354 BFs may result in an incorrect word written into the cache, that will be corrected by the Decoder when read. Thus, all Encoder BFs are latent faults, that constitute a risk for future silent data corruption and microprocessor reliability.

Instead, also from Fig. 2, we can notice that, as for the Decoder, out of its possible 836 BFs, the 97% (808 BFs) result in an incorrect word given to the cache output. Therefore, they are critical faults that can cause silent data corruption and compromise the microprocessor reliability. The remaining 3% (28 BFs) do not have any effect on the *checker* operation, thus they remain present in the *checker* as latent faults, that may

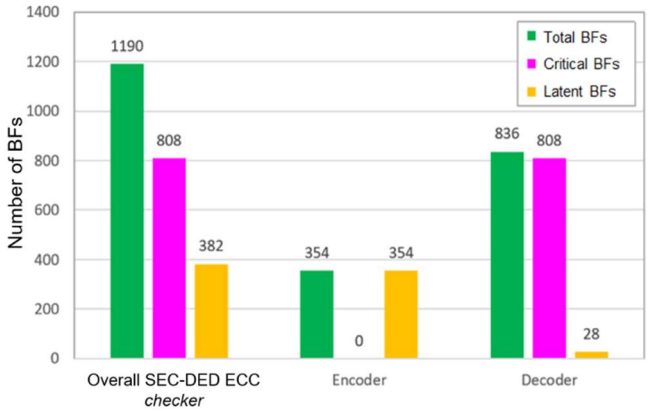

Fig. 2. Number of critical and latent BFs affecting the blocks of the considered SEC-DED ECC *checker*.

accumulate with following faults and/or aging conditions, thus constituting a future risk for silent data corruption and microprocessor reliability.

As for the considered DEC ECC *checker*, the obtained results are summarized in Fig. 3.

We can see that, out of the considered 2784 *checker* BFs, the 61% (1690 BFs) result in an incorrect word given to the cache output, thus they are critical faults that can cause silent data corruption and compromise the microprocessor reliability.

The remaining 39% of *checker* BFs (1094 BFs) either result in an incorrect word written into the cache that will be corrected by the Decoder when read, or do not have any effect on the *checker* operation, thus these faults remain present in the *checker* as latent faults. Such latent BFs may accumulate with following faults and/or aging conditions, thus constituting a future risk for silent data corruption and microprocessor reliability.

More specifically, from Fig. 3 we can see that, for the Encoder, similarly to the SEC-DED ECC *checker*, all its possible 736 BFs may result in an incorrect word written into the cache, that will be corrected by the Decoder when read. Thus, all the Encoder faults are latent faults, that constitute a risk for future silent data corruption and microprocessor reliability.

Moreover, from Fig. 3 we can also notice that, for the Decoder, out of its possible 2048 BFs, the 83% (1690 BFs) result in an incorrect word given to the cache output. Therefore, they are critical faults that can cause silent data corruption and compromise the microprocessor reliability. The remaining 17% of Decoder BFs (358 BFs) do not have any effect on the *checker* operation, thus they remain present in the *checker* as latent faults, that may accumulate with following faults and/or aging conditions, thus constituting a future risk for silent data corruption and microprocessor reliability.

Therefore, to summarize, our analyses highlight that a large percentage of possible BFs affecting ECC *checkers* are critical faults, that can cause silent data corruption and compromise the microprocessor reliability. They may: 1) inhibit the ECC's correction ability, so that the word given to the cache output is the same as the (erroneous) word read from the cache, despite the presence of signal E = 1 (indicating that the word read from the cache is erroneous), or 2) introduce errors in otherwise correct words read from the cache, so that the word given to the cache output is different from the (correct) word read from the cache, depsite the presence of signal E = 0 (indicating that the word read from the cache is correct).

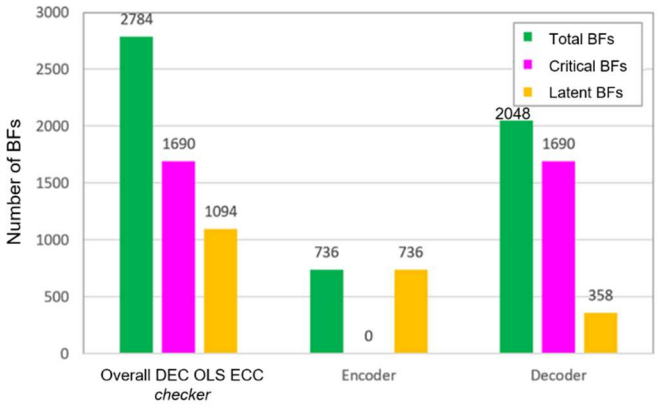

Fig. 3. Number of critical and latent BFs affecting the blocks of the considered DEC ECC *checker*.

IV. Strategy for Critical Fault Detection

In order to address the problems highlighted by the analyses described in Section III, we here propose an approach to detect, during the microprocessor in field operation, critical faults possibly affecting ECC *checkers* for caches.

It consists of adding to ECC *checkers* a *Detector*, as schematically represented in Fig. 4 for the case of the SEC-DED ECC *checker* (Fig. 1(a)). However our *Detector* can be as well adopted also for the DEC ECC *checker,* as well as for *checkers* for other ECCs.

Our proposed *Detector* compares the word read from the cache (($d'_0 \ldots d'_{15}$ $c'_0 \ldots c'_5$) in Fig. 4) with the word given to the cache output (($dc_0 \ldots dc_{15}$) in Fig. 4) and check bits generated from the information bits present in the word read from the cache (($c_{0R} \ldots c_{5R}$) in Fig. 4), and gives (EC_1, EC_2) = (0, 1) or (1, 0) if they are the same, and (EC_1, EC_2) = (0, 0) or (1, 1) otherwise.

Then, signals (EC_1, EC_2) can be analyzed together with the signal E generated by the *checker* Decoder to detect the possible presence of *checker* critical BFs. In particular, in case of fault-free ECC *checkers*, it will be:

(EC_1, EC_2, E) = (0, 0, 1) or (1, 1, 1), or

(EC_1, EC_2, E) = (0, 1, 0) or (1, 0, 0).

In fact, in the absence of *checker* critical BFs and no error in the word read from the cache, it is E = 0, ($d'_0 \ldots d'_{15}$) = ($dc_0 \ldots dc_{15}$) and ($c'_0 \ldots c'_5$) = ($c_0 \ldots c_5$), so that the *Detector*

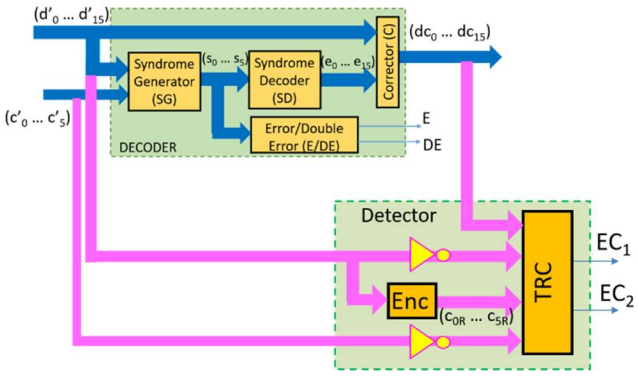

Fig. 4 Schematic representation of the proposed detection approach, for the case of the SEC-DEC *checker*.

produces to its output (EC_1, EC_2) = (0, 1) or (1, 0), depending on the particular word read from the cache. The condition (EC_1, EC_2) = (0, 1) or (1, 0) is consistent with the indication E=0, so this combination is recognized as an indication of absence of critical BFs in the *checker*. Instead, in case of an incorrect word read from the cache, it is E = 1, and ($d'_0 \ldots d'_{15}$) ≠ ($dc_0 \ldots dc_{15}$) or ($c'_0 \ldots c'_5$) ≠ ($c_0 \ldots c_5$), so that the Detector produces to its output (EC_1, EC_2) = (0, 0) or (1, 1).

Instead, in case of critical BFs affecting ECC *checkers*, it results:

(EC_1, EC_2, E) = (0, 1, 1) or (1, 0, 1), or

(EC_1, EC_2, E) = (0, 0, 0) or (1, 1, 0).

In fact, in case of critical BFs affecting ECC *checkers* that inhibit the ECC's correction ability it is E = 1 (indicating that the word read from the cache is erroneous), but the word given to the cache output is the same as the (erroneous) word read from the cache, so that the *Detector* produces to its output (EC_1, EC_2) = (0, 1) or (1, 0). Insetad, in case of critical faults affecting ECC *checkers* that introduce errors in otherwise correct words read from the cache, it is E = 0 (indicating that the word read from the cache is correct), but the word given to the cache output is different from the (correct) word read from the cache, so that the *Detector* gives to its output (EC_1, EC_2) = (0, 0) or (1, 1).

The *Detector* can be implemented as schematically represented in Fig. 4. In particular, it consists of: i) an Encoder (*Enc*); ii) a *two-rail* code checker *TRC* (*e.g.,* of the kind in [25]). *Enc* regenerates check bits ($c_{0R} \ldots c_{15R}$) starting from the information bits ($d'_0 \ldots d'_{15}$) present in the word read from the cache. *TRC* compares the complement of the information bits ($d'_0 \ldots d'_{15}$) present in the word read from the cache to the information bits ($dc_0 \ldots dc_{15}$) present in the word given to the cache output, and the complement of the check bits ($c'_0 \ldots c'_5$) present in the word read from the cache to the check bits ($c_{0R} \ldots c_{5R}$) generated by *Enc*. The NOTs inside the *Detector* are employed to obtain the complemented versions of the information bits and check bits present in the word read from the cache. From Fig. 4, we can observe that *Enc* and the NOTs of the *Detector* elaborate the information in parallel with the ECC decoder block, so that the impact on the ECC *checker* delay due to our *Detector* is given by the delay of the *TRC* only.

We have compared the impact on the ECC *checker* delay and area of our *Detector* to those of the alternate detector in [22], for the considered SEC-DED ECC and DEC ECC *checkers.*

The achieve results are reported in Table 1 where, for each considered *checker*, we report the additional input-output delay (*AD*) and the additional area (*AA*), expressed in equivalent gates (EqG), required by the compared solutions. Moreover, Table 1 also reports the percentage relative reductions in terms of impact on *checker'*s delay (*ΔAD%*) and area overhead (*ΔAA%*) enabled by our proposed detector. They are calculated as follows:

$$\Delta AD(\%) = 100\frac{(AD_{our} - AD_{[22]})}{AD_{[22]}}$$

$$\Delta AA(\%) = 100\frac{(AA_{our} - AA_{[22]})}{AA_{[22]}}$$

TABLE 1. CHECKER ADDITIONAL DELAY AND AREA DUE TO THE COMPARED DETECTORS, AND RELATIVE REDUCTIONS ENABLED BY OUR SOLUTION.

	SEC-DED Hsiao checker				DEC OLS checker			
	AD (ps)	ΔAD (%)	AA (EqG)	ΔAA (%)	AD (ps)	ΔAD (%)	AA (EqG)	ΔAA (%)
Solution in [22]	191ps	-	273 EqG	-	165.9ps	-	314EqG	-
Our scheme	76.6ps	-60%	204EqG	-25%	76.6%	-54%	228EqG	-27%

As can be seen from Table 1, compared to the solution in [22], our detector features a significantly lower impact on the ECC *checker* delay (-60% for the considered SEC-DED *checker*, and -54% for the considered DEC *checker*), and area (-25% for the considered SEC-DED ECC *checker*, and -27% for the considered DEC ECC *checker*).

V. CONCLUSIONS

We have analyzed the effects of bridging faults (BFs) possibly affecting *checkers* of two widely adopted ECCs for caches: the Single Error Correcting – Double Error Detecting (SEC-DED) Hsiao ECC, and the Double Error Correcting (DEC) Orthogonal Latin Square (OLS) ECC.

We have found that the 68% (61%) of BFs possibly affecting the considered SEC-DED (DEC) ECC *checkers* are critical, since they can either inhibit the ECC correction ability of incorrect words read from the cache and given to the cache output, or introduce errors in otherwise correct words read from the cache and given to the cache output, with consequent risks for silent data corruption and microprocessor reliability.

The remaining 32% (39%) of BFs possibly affecting the considered SEC-DED (DEC) ECC *checkers* either do not produce any effect, or introduce errors in the word written into the cache that will be corrected once such a word will be read. Therefore, they remain latent and may accumulate with following faults or aging conditions, with consequent future risks for silent data corruption and microprocessor reliability.

We have then introduced a possible approach to detect on line the occurrence of critical BFs. Compared to an alternate solution presented in the literature, our proposed scheme features significantly lower impact on ECC *checkers'* delay and area.

REFERENCES

[1] G. Li, S. K. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, S. Keckler, "Understanding Error Propagation in Deep Learning Network (DNN) Accelerators and Applications", in Proc. of the Int. Conf. for High Perf. Computing, Networking, Storage and Analysis, Nov. 2017, Article No. 8, pp. 1-12.

[2] A. Bosio, P. Bernardi, A. Ruospo, E. Sanchez, “A Reliability Analysis of a Deep Neural Network”, in Proc. of IEEE Latin American Test Symp. (LATS), 2019.

[3] V. Sze, T.-J. Yang, Y.-H. Chen, J. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," Proceedings of the IEEE, vol. 105, no. 12, December 2017, pp. 2295-2329.

[4]F. Finelli, M. Omaña and C. Metra, "Impact of Soft Errors on High Performance Autoencoders for Cyberattack Detection," *2022 IEEE 23rd Latin American Test Symp. (LATS),* 2022, pp. 1-6.

[5] C. Metra, D. Rossi, M. Omaña, A. Jas, R. Galivanche, "Function-Inherent Code Checking: A New Low Cost On-Line Testing Approach For High Performance Microprocessor Control Logic", in IEEE Proc. of the European Test Symposium, 2008, pp. 171—176.

[6] ISO 26262-1:2011(en) Road vehicles — Functional safety — Part 10: Guideline on ISO 26262. Int. Standardization Organization. pp. 5–6.

[7] https://it.wikipedia.org/wiki/IEC_61508

[8] https://www.iso.org/standard/70939.html

[9] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, F. Cappello, "Toward General Software Level Silent Data Corruption Detection for Parallel Applications," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 12, Dec. 2017, pp. 3642-3655.

[10] G. Papadimitriou, D. Gizopoulos, "Silent Data Corruptions: Microarchitectural Perspectives," in *IEEE Transactions on Computers*, vol. 72, no. 11, Nov. 2023, pp. 3072-3085.

[11] P. H. Hochschild et al., “Cores that don’t count,” in *Proc. Workshop Hot Topics Operating Syst.*, 2021, pp. 9–16.

[12] M.H. Hsiao, “A Class of Optimal Minimum odd-weight-column SEC-DED Codes”, IBM J. of Research and Development, Vol. 14, No. 4, July 1970, pp. 395-401.

[13] D. C. Bossen, “b-Adjacent Error Correction”, IBM Journal of Research and Development, Vol. 14, No. 4, July 1970, pp. 402-408.

[14] M. Hsiao, D. C. Bossen, R. T. Chien, “Orthogonal Latin Square Codes”, IBM Journal of Research and Development, Volume: 14, Issue: 4, 1970.

[15] M. Omaña, D. Rossi, T. Edara, C. Metra, “Impact of Aging Phenomena on Latches’ Robustness”, IEEE Trans. on Nanotechnology, Issue 2, March 2016, pp. 129-136.

[16] M. Omaña, T. Edara, C. Metra, “Low-Cost Strategy to Mitigate the Impact of Aging on Latches’ Robustness”, IEEE Trans. on Emerging Topics in Computing, Vol. 6, Issue 4, Dec. 2018, pp. 488-497.

[17] A. Manfredi, “Rischi per la Safety delle Cache di Microprocessori per Sistemi Intelligenti ad Elevato Livello di Autonomia”, Univ. Bologna Master Thesis, *AMS Laurea Institutional Theses Repository.*

[18] M. Omaña, D. Rossi, C. Metra, "Latch Susceptibility to Transient Faults and New Hardening Approach", in IEEE Transactions on Computers, vol. 56, no. 9, Sept. 2007, pp. 1255-1268.

[19] D. Ernst et al., "Razor: a low-power pipeline based on circuit-level timing speculation", in Proc. of IEEE/ACM Int. Symposium on Microarchitecture, MICRO-36, 2003, pp. 7-18.

[20] F. Aymen, H. Belgacem and K. Chiraz, "A new efficient self-checking Hsiao SEC-DED memory error correcting code", in Proc. of ICM, 2011, pp. 1-5.

[21] I. Boyarinov, “Self-Checking Circuits and Decoding Algorithms for Binary Hamming and BCH Codes and Reed-Solomon Codes over GF(2^m)”, Problems of Information Transmission, Springer, Vol. 44, No. 2, 2008, pp. 99-111.

[22] R. Redinbo, “Fault-Tolerant Decoders for Cyclic Error-Correcting Codes”, IEEE Transactions on Computers, Vol. C-36, No. 1, January 1987, pp. 47-63.

[23] M.J. Ashjaee and S.M. Reddy, “On Totally Self-CheckingCheckers for Separable Codes,”IEEE Transactions on Computers, Vol. C-26, pp. 737–744, Aug. 1977.

[24] https://www.synopsys.com/verification/ams-verification/hspice.html

[25] D.A. Anderson, G. Metze, “Design of Totally Self-Checking Circuits for m-Out-of-n Codes”, IEEE Transactions on Computers, vol. 22,no. 3, pp. 263-269, Mar. 1973.