Microelectronics Reliability 106 (2020) 113582

Contents lists available at ScienceDirect

Microelectronics Reliability

journal homepage: www.elsevier.com/locate/microrel

# Extended Matrix Region Selection Code: An ECC for adjacent Multiple Cell Upset in memory arrays

Felipe Silva[a], Walter Freitas[a], Jarbas Silveira[a,*], César Marcon[b], Fabian Vargas[b]

[a] Departamento de Engenharia de Teleinformática, Campus do Pici, Universidade Federal do Ceará, Fortaleza, CE CEP 60455-970, Brazil
[b] Pontifical Catholic University of Rio Grande do Sul - PUCRS, Brazil

ARTICLE INFO

*Keywords:*
Error Correction Code (ECC)
Multiple Cell Upset (MCU)
Adjacent cell
Transient error
Reliable memory
Critical application

ABSTRACT

With the widespread of electronics nowadays, Single Event Effects (SEEs) have become a significant concern, not only for critical applications like aerospace and military but also for the automotive industry and medical instruments, where reliability is always at a premium. This concern is notable in environments containing ElectroMagnetic (EM) and ionizing radiations, whose interactions with the matter may change the state of memory elements and thus, degrading system reliability. Technology scaling down increases the probability that the strike of a charged particle or a power bus fluctuation due to conducted EM interference affects more than one cell; therefore, resulting in a Multiple Cell Upset (MCU). Single Error Correction–Double Error Detection (SEC-DED) codes are among the most applied techniques to provide reliability to memory systems. However, standard implementations of SEC-DED codes are not suitable anymore to provide information reliability because they cannot satisfactorily handle a considerable number of bit-flips per coded word, i.e., MCU occurrence. In this context, this paper proposes the extended Matrix Region Selection Code (eMRSC), an improved version of MRSC, which extends the original 16-bit code previously published to a new MRSC version of 32 data bits. Additionally, it is proposed a new scheme of data matrix regions for reducing the number of generated redundant bits. The proposed codes were compared to well-known codes, presenting outperforms in all experiments. The synthesis analysis showed that the proposed codes are not only reliable, but they also result in low implementation cost (i.e., low area, coding/decoding delay and power overheads).

## 1. Introduction

Single Event Effect (SEE) is a vital concern for the reliability of electronic systems devoted to critical applications due to the continued scaling down of CMOS technology. This concern is even more notable in environments containing electromagnetic radiation [1], which may change the state of memory elements, degrading the system reliability [2]. The technology scaling down also increases the probability that the strike of a charged particle affects more than one cell in the neighborhood; thus, resulting in Multiple Cell Upsets (MCU) [3]. Many sources, such as direct ionization or nuclear recoil after the passage of a high-energy ion, can induce MCUs [4,5]. Unfortunately, these effects cannot be avoided by the implementation of packaging and shielding solutions, as showed in [1], and their occurrences increase drastically as the technology shrinks [6].

Error Correcting Codes (ECCs) are among the most applied techniques to provide reliability to memory systems. Additionally, Single Error Correction-Double Error Detection (SEC-DED) are ECCs applied to avoid data corruption due mainly to their efficiency and simple implementation [7,8]. However, a standard implementation of an SEC-DED code is not suitable to provide high information reliability. Some researchers proposed the use of the interleaving technique (also named scrambling) combined with the SEC-DED codes to deal with MCUs [9]. This combined approach distributes an MCU occurrence into different logical words, making the ECC circuit perceive the MCU as single errors that can be corrected by SEC-DED codes. However, this combined method does not perform well for MCU occurrence in a Ternary Content Addressable Memory (TCAM) because of the tight physical coupling in the hardware structures of both the cell and comparing circuits [10], requiring more efficient and complex ECCs.

Satoh et al. [11] presented the geometric effect of MCUs induced by neutrons in Dynamic Random-Access Memories (DRAMs). The results showed that the Soft Error Rate of the DRAM decreases as the distance between storage nodes increases. More recently, Rao et al. [12]

* Corresponding author.
*E-mail addresses:* gaspar@lesc.ufc.br (F. Silva), walter@gtel.ufc.br (W. Freitas), jarbas@lesc.ufc.br (J. Silveira), cesar.marcon@pucrs.br (C. Marcon), vargas@computer.org (F. Vargas).

https://doi.org/10.1016/j.microrel.2020.113582
Received 8 April 2019; Received in revised form 20 August 2019; Accepted 7 January 2020
Available online 22 January 2020

analyzed the possible patterns for multiple errors in Static Random-Access Memories (SRAMs), concluding that the majority of the MCUs occurred in adjacent cells.

Matrix Region Selection Code (MRSC) is a Two-Dimensional (2-D) code that can correct multiple patterns of MCUs. This code, which was proposed in [13], presents lower area overhead and higher efficiency to correct MCUs than other ECCs commonly found in the literature. MRSC sorts the data bit matrix into regions, allowing correcting bit errors by analyzing the redundancy bit syndromes.

This paper presents the extended MRSC (eMRSC), an improved version of MRSC, which extends the original 16-bit code proposed in [13] into a new MRSC version of 32 data bits. We implemented eMRSC in two code formats: one with lower increase of redundancy and another one with higher correction capability. Also, we propose a new scheme of data matrix regions that reduces the number of generated redundant bits. This new code is an alternative solution that generates a tradeoff between area and power overheads and reliability degree.

This work is structured as follows. Section 2 describes the related work used as fundamentals of the proposed approach. Sections 3 and 4 describe the fundamentals of MRSC and details the encoder and decoder algorithms of the proposed eMRSC. Section 5 exemplifies some cases of using eMRSC to correct data and redundancy errors. Sections 6 to 8 describe the experiments used to evaluate the proposed eMRSC. These experiments deal with parameters such as error correction efficiency, reliability estimation, and synthesis analysis. Finally, Section 9 draws the main conclusion of this work.

## 2. Related work

Since the beginning of the outer space exploration, the ECC study has played an essential role in given reliability to computational systems of critical application. These codes were initially applied to the communications systems, providing reliable data transmission between the ground station and satellite. In the early 70s, the Mariner mission used Reed-Muller code to send images captured from Mars surface to Earth [14]. At that time, Reed-Muller code represented a tremendous improvement on the reliability of electronic systems, with the capacity to correct MCUs occurrence [15].

In 1994, Vargas and Nicolaidis [16] developed a 2-D memory structure that combines current checking with parity. After that, the study and development of 2-D ECCs became a trend due to its efficiency to deal with SEU and reduced cost compared to other codes. However, the correction procedure is more time consuming, requiring reading all memory content. More recently, in 2007, Argyrides et al. [17] proposed Matrix that combines Hamming code and parity to perform a 2-D scheme for correcting some patterns of MCUs. When compared to Reed-Muller, Matrix has limited capacity to detect and correct errors, but the Matrix encoding and decoding circuits consume much less energy and area, with much less delay and reduced reliability for larger block sizes.

In 2011, Reviriego et al. [18] proposed an ECC technique that uses SEC-DED code and parity. This ECC forms a 2-D structure, which is similar to the Matrix code with more redundant bits. Their code improves the Mean Time To Failure (MTTF) of a memory substantially, but with reduced reliability for large block sizes. Their work did not provide details on MCU detection and correction efficiency of the proposed method.

In 2014, Guo et al. [19] proposed the Decimal Matrix Code (DMC), which uses decimal sum with parity to form a 2-D ECC scheme. DMC is a low-cost code, making it a sound ECC for applications with limited energy resources. The DMC algorithm provides reduced reliability since it can detect and correct only specific types of MCUs.

Liu et al. [2] proposed in 2016 the Extended Orthogonal Latin Square (EOLS) codes, which are 32-bit adaptations of the original Orthogonal Latin Square (OLS) codes [20]. The original OLS codes can only be applied for a perfect square number of data bits (i.e., 16, 64 and 256). Liu et al. developed two versions of OLS codes: (i) one with a reduced number of redundancy bits and lower correction capability that consumes less area and dissipates less power, and (ii) another higher-cost version with a higher number of redundancy bits and more correction capability.

In 2017, Klockmann et al. [21] employed finite fields to develop a 3-bit burst error code, which corrects from one to three adjacent errors and two nearly adjacent errors. This code has low redundancy overhead and a decoding process with low complexity. In the same year, Silva et al. [13] described MRSC, which is a new 2-D scheme of ECC that provides high reliability with lower synthesis cost when compared to other 2-D ECCs. MRSC splits the data bits into regions (groups of data bits) and uses redundancy bits to find and correct the region with errors. This code was designed for adjacent errors, while the occurrence of non-adjacent errors reduces the code correction efficiency.

Silva et al. [22] presented in 2018 the CLC code, which uses extended Hamming and parity to compose a 2-D structure for providing high reliability to the information stored in memory arrays. The authors developed three CLC structures by changing the applied extended Hamming, which in turn affected the number of redundancy bits and error coverage. The obtained results indicate that each one of the proposed structures has its benefits, allowing more application flexibility. CLC has as main drawback the high redundancy increase for the proposed structures.

Li et al. [23] developed in 2018 two schemes to correct multiple bits, one based on the interleaving of Single Error Correction-Double Adjacent Error Correction Code (SEC-DAED), and another one is an optimized version of a 4-bit burst error. The synthesis results of the proposed codes showed that, despite the low increase of redundancy, a significant overhead was achieved in the decoder area. In the same year, Li et al. [24] proposed a scheme that interleaves a simple x-bit burst error code, increasing the error correction capability significantly, with lower or little increase of redundancy, when compared with the original code. The codes achieved a small increase in delay, although this is an acceptable increase for some applications.

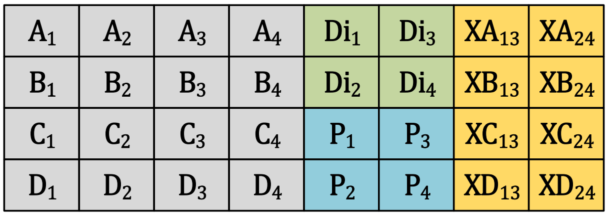
Related works show that, in the last 25 years, researchers have been looking for 2-D ECC formats aiming at efficiency and efficacy. eMRSC is an MRSC code extension that achieves this desired efficacy with low implementation cost.

## 3. Fundamentals of MRSC

The original Matrix Region Selection Code (MRSC) proposed in [13] uses parity and check bits to encode a 16-bit data in a 32-bit codeword. Fig. 1 shows the basic structure of the original MRSC code.

The 32-bit codeword of MRSC includes four sets of bits: (i) 16 data bits divided into A, B, C and D groups of four bits each; (ii) four diagonal bits ($Di_1$, $Di_2$, $Di_3$, $Di_4$); (iii) four parity bits ($P_1$, $P_2$, $P_3$, $P_4$); and (iv) eight check bits ($XA_{13}$, $XA_{24}$, $XB_{13}$, $XB_{24}$, $XC_{13}$, $XC_{24}$, $XD_{13}$, $XD_{24}$). Fig. 2 shows that MRSC splits the data bits matrix into three regions ($R_1$, $R_2$, and $R_3$).

The MRSC correction procedure consists in analyzing the syndromes of the redundancy bits and choosing one of the regions to be corrected. MRSC is suitable to correct adjacent error due to the following features: (i) the appliance of the *Di* bits allows detecting some particular error patterns such as two errors in the same column; and (ii) the code region



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $Di_1$ | $Di_3$ | $XA_{13}$ | $XA_{24}$ |
| $B_1$ | $B_2$ | $B_3$ | $B_4$ | $Di_2$ | $Di_4$ | $XB_{13}$ | $XB_{24}$ |
| $C_1$ | $C_2$ | $C_3$ | $C_4$ | $P_1$ | $P_3$ | $XC_{13}$ | $XC_{24}$ |
| $D_1$ | $D_2$ | $D_3$ | $D_4$ | $P_2$ | $P_4$ | $XD_{13}$ | $XD_{24}$ |

**Fig. 1.** MRSC structure - 16-bit data [13].

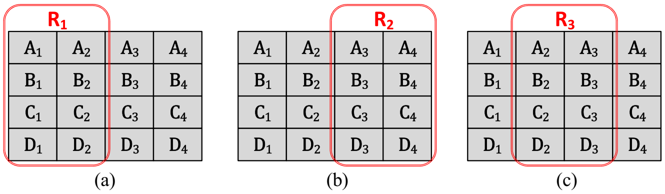


**Fig. 2.** MRSC regions of data bits [13].

strategy that allows the correction of aggressive MCUs occurring in the same region (e.g., five errors).

Moreover, the strategy of having regions on MRSC also optimizes the use of the check bits, as they are used to cover all data regions. Since adjacent errors are the most common error case in memory devices, it is expected that multiple errors occur in the same region. Thus, the check bits will correct only one region. The check bits matrix must have the same dimensions of the MRSC regions to perform the correction properly.

## 4. Extended Matrix Region Selection Code (eMRSC)

eMRSC($m$,$r$,$n$) extends the basic MRSC code from 16 to 32 data bits, including three fields: (i) the amount of data bits being encoded ($m$), (ii) the number of regions for this code ($r$), and (iii) the total number of bits generated ($n$). This paper describes the algorithm of eMRSC(32,3,64) and eMRSC(32,7,56), which are two versions of eMRSC with 32-bit data that deal with three and seven regions, respectively.

The eMRSC code is designed to deal only with error patterns that are encapsulated within the region format defined by the 3-tuple ($m$, $n$, $r$). Therefore, the designer has to consider in the eMRSC code selection, not only the number of errors, but also the error pattern to be covered.

It is important to emphasize that we planned to implement MRSC and eMRSC in specific memories, built to arrange the codeword in the spatial distributions exemplified in this work; i.e., 4 rows and 16/14 columns (considering parity bits) adjacent to each other. Using commercial memories require inserting a coding/decoding wrapping circuit in the input/output of the memory for remapping the memory addresses according to the code format, while performing more than one read/write access (typically four) to each read/write memory access.

### 4.1. eMRSC(32,3,64) encoding process

eMRSC(32,3,64) uses similar encoding logic as presented in [13], although this code codifies a 32-bit data word to a 64-bit codeword. Fig. 3 shows the structure of the eMRSC(32,3,64) codeword.

eMRSC(32,3,64) duplicates the data and redundancy bits of the basic MRSC. Eqs. (1) to (4) describe how the check bits ($XA$, $XB$, $XC$ and $XD$) are calculated using XOR (⊕) operations between data columns indexed by $v$ = {1, 2, 3, 4} and $w$ = {5, 6, 7, 8} (e.g., $XA_{15} = A_1 \oplus A_5$).

$$XA_{vw} = A_v \oplus A_w \tag{1}$$

$$XB_{vw} = B_v \oplus B_w \tag{2}$$

$$XC_{vw} = C_v \oplus C_w \tag{3}$$

$$XD_{vw} = D_v \oplus D_w \tag{4}$$

Eq. (5) describes the $Di$ bits calculation, which uses $x$ = {1, 2, 3, 4, 5, 6, 7, 8} and $y$ = {2, 1, 4, 3, 6, 5, 8, 7} to select the data columns (e.g., $Di_1 = A_1 \oplus B_2 \oplus C_1 \oplus D_2$, $Di_3 = A_3 \oplus B_4 \oplus C_3 \oplus D_4$).

$$Di_x = A_x \oplus B_y \oplus C_x \oplus D_y \tag{5}$$

Eq. (6) computes the parity bits $P$ according to the data columns indexed by $x$ = {1, 2, 3, 4, 5, 6, 7, 8} (e.g., $P_1 = A_1 \oplus B_1 \oplus C_1 \oplus D_1$).

$$P_x = A_x \oplus B_x \oplus C_x \oplus D_x \tag{6}$$

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ | $A_7$ | $A_8$ | $Di_1$ | $Di_3$ | $Di_5$ | $Di_7$ | $XA_{15}$ | $XA_{26}$ | $XA_{37}$ | $XA_{48}$ |
| $B_1$ | $B_2$ | $B_3$ | $B_4$ | $B_5$ | $B_6$ | $B_7$ | $B_8$ | $Di_2$ | $Di_4$ | $Di_6$ | $Di_8$ | $XB_{15}$ | $XB_{26}$ | $XB_{37}$ | $XB_{48}$ |
| $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ | $C_6$ | $C_7$ | $C_8$ | $P_1$ | $P_3$ | $P_5$ | $P_7$ | $XC_{15}$ | $XC_{26}$ | $XC_{37}$ | $XC_{48}$ |
| $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ | $D_8$ | $P_2$ | $P_4$ | $P_6$ | $P_8$ | $XD_{15}$ | $XD_{26}$ | $XD_{37}$ | $XD_{48}$ |

**Fig. 3.** Structure of the eMRSC(32,3,64) codeword.

### 4.2. eMRSC(32,3,64) decoding process

The eMRSC(32,3,64) decoding encompasses four steps: (I) Calculation of the syndromes of the redundancy bits; (II) Verification of the error decoding conditions; (III) Selection of the region; and (IV) Correction of errors.

Eqs. (7) to (9) show how Step I computes the syndrome bits applying XOR operations between the original values of the redundancy bits ($Di$, $P$ and $X$) and the recalculation of the same bits after the occurrence of errors ($RDi$, $RP$ and $RX$).

$$SDi = Di \oplus RDi \tag{7}$$

$$SP = P \oplus RP \tag{8}$$

$$SX = X \oplus RX \tag{9}$$

Fig. 4 illustrates the flows, blocks, and operations employed by the decoder to compute $SDi$, $SP$ and $SX$. Note that there are two types of $SX$ matrices applied to eMRSC(32,3,64) that depend on the selected region.

Step II consists of analyzing the two the following conditions: (i) Both $SDi$ and $SP$ syndrome vectors must have at least one value equal to one, or (ii) More than one value of the syndromes of the check bits $SX$ must be equal to one.

This analysis determines whether MRSC proceeds with the error correction process. If one of these two conditions is satisfied, the coding algorithm understands that errors were detected and proceeds to the region selection step. In the event of none of the conditions are met, the coding algorithm ignores the correction process and delivers the data bits.

Fig. 5 illustrates the three possible regions containing data errors that Step III can select. eMRSC(32,3,64) comprises the same number of regions of the basic MRSC [13], but with larger dimensions (each one is composed by a 4 × 4 matrix of bits). The size of the region has a significant impact on the correction capability of the eMRSC algorithm. A

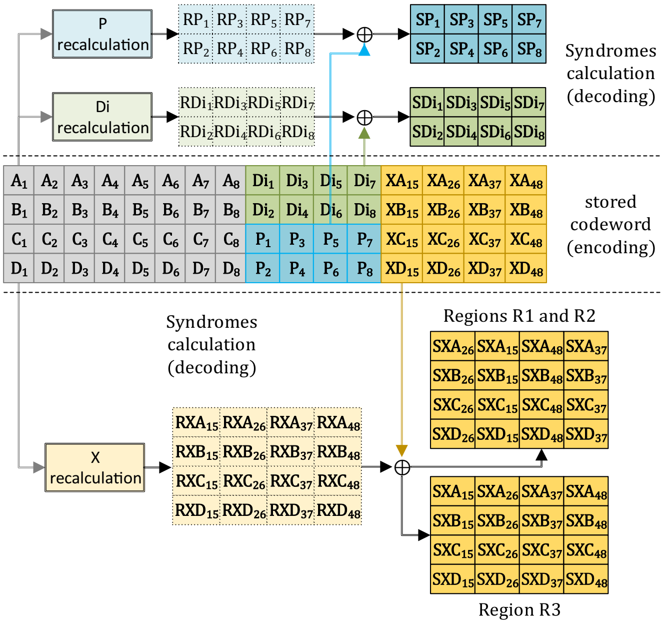


**Fig. 4.** $SDi$, $SP$ and $Sx$ calculations during the decoding process.

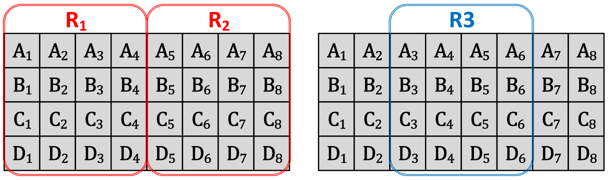


**Fig. 5.** Data regions of eMRSC(32,3,64).

**Table 1**
Criteria for region selection of the eMRSC(32,3,64) code.

| Region | Selection criterion |
|---|---|
| $R_1$ | $\sum_{i=1}^{4} SDi_i + SP_i \; > \; \sum_{i=5}^{8} SDi_i + SP_i$ |
| $R_2$ | $\sum_{i=1}^{4} SDi_i + SP_i \; < \; \sum_{i=5}^{8} SDi_i + SP_i$ |
| $R_3$ | $\sum_{i=1}^{4} SDi_i + SP_i \; = \; \sum_{i=5}^{8} SDi_i + SP_i$ |

larger region increases the chance of MCU occurrences being concentrated in a single region, facilitating the error detection. Only one region is selected to be corrected by the *X* field, and this is done by analyzing the results of *SP* and *SDi*, through comparison of which region has the higher number of errors detected.

We considered MCU patterns with only adjacent errors to build the region used in the eMRSC codes. However, when a more aggressive MCU pattern happens (i.e., an error pattern whose a single region cannot encompass all bit-flips), then, eMRSC is not able to correct the data in the codeword. Section 6 details the MCU patterns employed here.

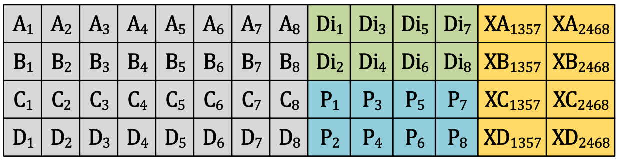
Table 1 contains the equations applied to select the region of the eMRSC(32,3,64) code containing errors. Note that the symbol + stands for the decimal sum of bits.

Once selected the region containing data errors, the correction algorithm performs XOR operation between the data region selected and the *SX* matrix of the region selected.

Fig. 6 presents the four steps of the eMRSC(32,3,64) decoding algorithm, emphasizing the information flow and the basic operations performed in each step, in order to facilitate the understanding of how regions are selected, as well as the error correction is performed.

### 4.3. eMRSC(32,7,56) encoding process

eMRSC(32,7,56) is a compact version of the MRSC code for a 32-bit data, which was developed aiming to reduce the redundancy impact; i.e., eMRSC(32,7,56) has 8 bits less than eMRSC(32,3,64), meaning a redundancy reduction by a factor of 25%. We attained this reduction dividing the data matrix into more regions; i.e., eMRSC(32,7,56) employs seven regions, resulting in four regions more than eMRSC(32,3,64). Fig. 7 shows the codeword for eMRSC(32,7,56).



| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ | $A_7$ | $A_8$ | $Di_1$ | $Di_3$ | $Di_5$ | $Di_7$ | $XA_{1357}$ | $XA_{2468}$ |
| $B_1$ | $B_2$ | $B_3$ | $B_4$ | $B_5$ | $B_6$ | $B_7$ | $B_8$ | $Di_2$ | $Di_4$ | $Di_6$ | $Di_8$ | $XB_{1357}$ | $XB_{2468}$ |
| $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ | $C_6$ | $C_7$ | $C_8$ | $P_1$ | $P_3$ | $P_5$ | $P_7$ | $XC_{1357}$ | $XC_{2468}$ |
| $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ | $D_8$ | $P_2$ | $P_4$ | $P_6$ | $P_8$ | $XD_{1357}$ | $XD_{2468}$ |

**Fig. 7.** Structure of the eMRSC(32,7,56) codeword.

Although eMRSC(32,7,56) provides a significant reduction of the redundancy bits, the eMRSC(32,7,56) encoder does not present substantial changes in the codeword encoding. For example, bits *Di* and *P* of are calculated using the same Eqs. (5) and (6) applied in the eMRSC(32,3,64) encoder. However, this new code requires some adjustment to compress the *X* bits from 16 to 8 bits. Eqs. (10) to (13) display how the *X* matrix of eMRSC(32,7,56) is determined using the indexes $v = \{1, 2\}$, $w = \{3, 4\}$, $x = \{5, 6\}$ and $y = \{7, 8\}$; e.g., $XA_{1357} = A_1 \oplus A_3 \oplus A_5 \oplus A_7$.

$$XA_{vwxy} = A_v \oplus A_w \oplus A_x \oplus A_y \tag{10}$$

$$XB_{vwxy} = B_v \oplus B_w \oplus B_x \oplus B_y \tag{11}$$

$$XC_{vwxy} = C_v \oplus C_w \oplus C_x \oplus C_y \tag{12}$$

$$XD_{vwxy} = D_v \oplus D_w \oplus D_x \oplus D_y \tag{13}$$

### 4.4. eMRSC(32,7,56) decoding process

The eMRSC(32,7,56) decoding is also performed in four steps, with Steps I and II being the same already discussed in Section 4.2. eMRSC(32,7,56) employs seven data regions, each region containing a 4 × 2 matrix of bits. Fig. 8(a) and (b) displays the seven regions for this code. Compared to MRSC [13], eMRSC(32,7,56) has more regions, although these regions have the same sizes. Also, eMRSC(32,7,56) has a codifying model that employs two overlapped sets of regions $Reg_1 = \{R_1, R_2, R_3, R_4\}$ and $Reg_2 = \{R_5, R_6, R_7\}$.

eMRSC(32,7,56) implements this overlapped model of regions because its algorithm requires selecting a unique region to be corrected. Considering the occurrence of only adjacent errors for MCU events, eMRSC(32,7,56) decoding algorithm requires that the errors have to be only in a single region. However, when there are errors in two regions of the set $Reg_1$, the algorithm tries to find a single region in the set $Reg_2$. For instance, supposing errors on data cells $B_4$ and $B_5$, belonging to regions $R_2$ and $R_3$, respectively; thus, the algorithm selects region $R_6$ that includes both cells.

The region selection (Step III) of eMRSC(32,7,56) is more complex than eMRSC(32,3,64); it is performed in one or two sub-steps that require to analyze *SP* and *SDi*. The first sub-step employs the Inequations described in Table 2, which allows finding the region with more errors of the $Reg_1$.

Table 2 is not complete; there are some cases where all Inequations returns false, meaning that none of these regions has to be selected.

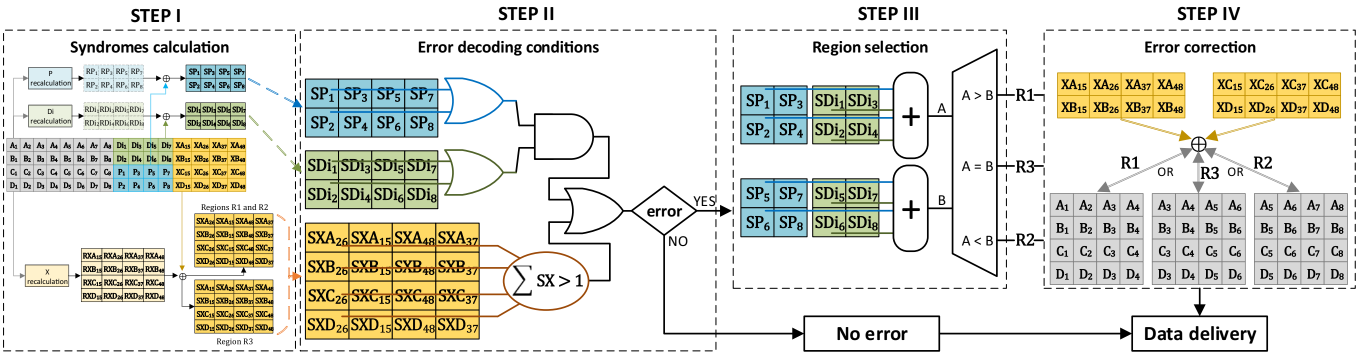
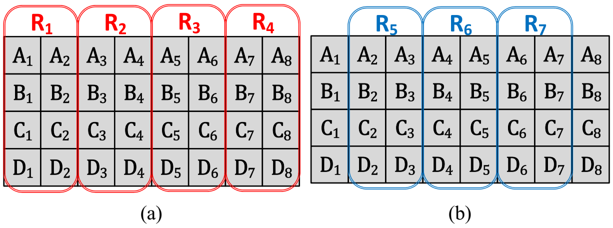


**Fig. 6.** Decoding steps for eMRSC(32,3,64).



| $R_1$ | | $R_2$ | | $R_3$ | | $R_4$ | |
|---|---|---|---|---|---|---|---|
| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ | $A_7$ | $A_8$ |
| $B_1$ | $B_2$ | $B_3$ | $B_4$ | $B_5$ | $B_6$ | $B_7$ | $B_8$ |
| $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ | $C_6$ | $C_7$ | $C_8$ |
| $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ | $D_8$ |

(a)

| | $R_5$ | | $R_6$ | | $R_7$ | | |
|---|---|---|---|---|---|---|---|
| $A_1$ | $A_2$ | $A_3$ | $A_4$ | $A_5$ | $A_6$ | $A_7$ | $A_8$ |
| $B_1$ | $B_2$ | $B_3$ | $B_4$ | $B_5$ | $B_6$ | $B_7$ | $B_8$ |
| $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ | $C_6$ | $C_7$ | $C_8$ |
| $D_1$ | $D_2$ | $D_3$ | $D_4$ | $D_5$ | $D_6$ | $D_7$ | $D_8$ |

(b)

**Fig. 8.** Data regions of eMRSC(32,7,56).

**Table 2**
Criteria employed in the eMRSC(32,7,56) decoding for selecting regions belonging to the set $Reg_1$.

| Region | Criterion to selection |
|---|---|
| $R_1$ | $\sum_{i=1}^{2} SDi + SP_i > \sum_{i=3}^{4} SDi + SP_i, \sum_{i=5}^{6} SDi + SP_i, \sum_{i=7}^{8} SDi + SP_i$ |
| $R_2$ | $\sum_{i=3}^{4} SDi + SP_i > \sum_{i=1}^{2} SDi + SP_i, \sum_{i=5}^{6} SDi + SP_i, \sum_{i=7}^{8} SDi + SP_i$ |
| $R_3$ | $\sum_{i=5}^{6} SDi + SP_i > \sum_{i=3}^{4} SDi + SP_i, \sum_{i=3}^{4} SDi + SP_i, \sum_{i=7}^{8} SDi + SP_i$ |
| $R_4$ | $\sum_{i=7}^{8} SDi + SP_i > \sum_{i=1}^{2} SDi + SP_i, \sum_{i=3}^{4} SDi + SP_i, \sum_{i=5}^{6} SDi + SP_i$ |

**Table 3**
Criteria employed in the eMRSC(32,7,56) decoding for selecting regions belonging to the set $Reg_2$.

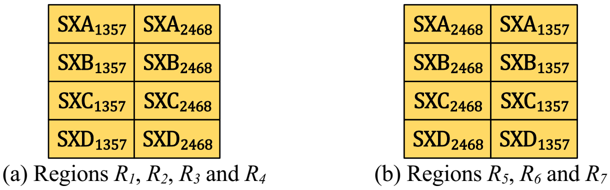
| Region | Criterion to selection |
|---|---|
| $R_5$ | $\sum_{i=1}^{2} SDi + SP_i = \sum_{i=3}^{4} SDi + SP_i$ |
| $R_6$ | $\sum_{i=1}^{2} SDi + SP_i \neq \sum_{i=3}^{4} SDi + SP_i$ AND<br>$\sum_{i=3}^{4} SDi + SP_i = \sum_{i=5}^{6} SDi + SP_i$ |
| $R_7$ | $\sum_{i=1}^{2} SDi + SP_i \neq \sum_{i=3}^{4} SDi + SP_i$ AND<br>$\sum_{i=3}^{4} SDi + SP_i \neq \sum_{i=5}^{6} SDi + SP_i$ AND<br>$\sum_{i=5}^{6} SDi + SP_i = \sum_{i=7}^{8} SDi + SP_i$ |

These are the cases where the eMRSC(32,7,56) decoding algorithm employs the Equations of Table 3 to find into the set of regions $Reg_2$ the one that encompasses all the data errors. It is essential to point out that the regions of set $Reg_2$ are only selected if errors were detected. eMRSC is an appropriate code only to correct SEUs or MCUs containing errors in adjacent cells; this premise does not allow the existence of errors in two or more non-adjacent regions. Also, when an error pattern is in two adjacent regions, our proposed code defines a new region (the overlapped region) that contains the adjacent cells of both adjacent regions, and choose the first accepted condition.

Fig. 9(a) shows the *SX* matrix used to correct regions $R_1$ to $R_4$ – the regions selected with Table 2, and Fig. 9(b) shows the *SX* matrix used to correct regions $R_5$ to $R_7$ – the regions selected with Table 3.

## 5. Examples of error correction using eMRSC

This section presents some examples of the MRSC operation employing 32-bit data containing the following sequence: "10001000 11111111 10101010 00000000". Fig. 10 presents the codewords for eMRSC(32,3,64) and eMRSC(32,7,56) when codifying this sequence.



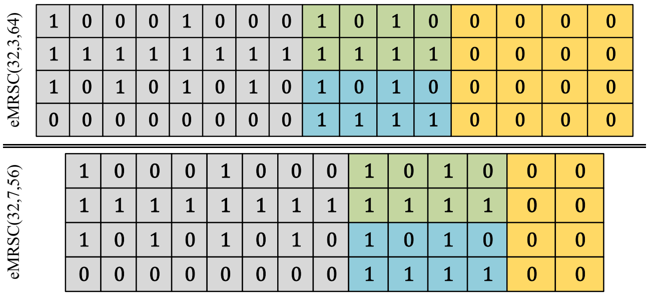
| | |
|---|---|
| $SXA_{1357}$ | $SXA_{2468}$ |
| $SXB_{1357}$ | $SXB_{2468}$ |
| $SXC_{1357}$ | $SXC_{2468}$ |
| $SXD_{1357}$ | $SXD_{2468}$ |

(a) Regions $R_1$, $R_2$, $R_3$ and $R_4$

| | |
|---|---|
| $SXA_{2468}$ | $SXA_{1357}$ |
| $SXB_{2468}$ | $SXB_{1357}$ |
| $SXC_{2468}$ | $SXC_{1357}$ |
| $SXD_{2468}$ | $SXD_{1357}$ |

(b) Regions $R_5$, $R_6$ and $R_7$

**Fig. 9.** *SX* matrices for correcting regions of eMRSC(32,7,56).



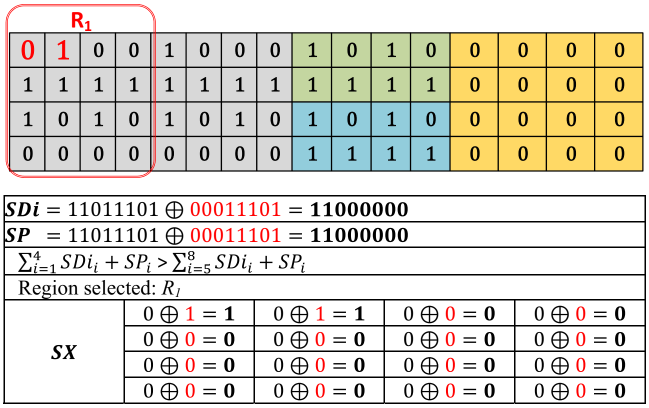
eMRSC(32,3,64)

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

eMRSC(32,7,56)

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

**Fig. 10.** eMRSC(32,3,64) and eMRSC(32,7,56) codewords.



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

$SDi$ = 11011101 ⊕ 00011101 = **11000000**
$SP$ = 11011101 ⊕ 00011101 = **11000000**
$\sum_{i=1}^{4} SDi_i + SP_i > \sum_{i=5}^{8} SDi_i + SP_i$
Region selected: $R_1$

| | | | | |
|---|---|---|---|---|
| ***SX*** | 0 ⊕ 1 = **1** | 0 ⊕ 1 = **1** | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** |
| | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** |
| | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** |
| | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** |

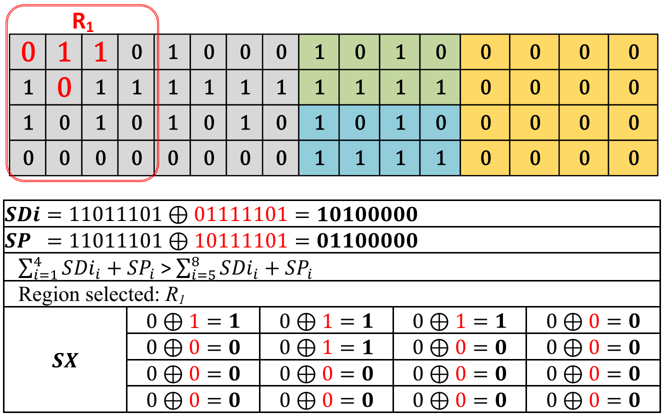
**Fig. 11.** eMRSC(32,3,64) correcting a two bit-flip error.

Fig. 11 shows the first error correction example containing two bit-flip error (two bits in red of the two leftmost columns of the first line of the codeword matrix) in the eMRSC(32,3,64) codeword. Fig. 11 also shows the decoding results together with the selected region.

The numbers in bold are the results of the syndromes; the red numbers associated with *SDi*, *SP* and *SX* syndromes are the recalculated bits considering the error bits. Note that the condition for region $R_1$ correction is satisfied; therefore, $R_1$ can be corrected by the *SX* matrix.

Fig. 12 displays the second error correction example that contains four bit-flips. eMRSC(32,3,64) is still able to correct more aggressive error patterns, like the one presented here, due to its large data region that allows the detection of MCU patterns that occur in the same region. Note that eMRSC(32,7,56) has lower correction capability precluding to correct this same error pattern.

Fig. 13 illustrates the third error correction example, which considers an error of two bit-flip occurring in an eMRSC(32,7,56) codeword.



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |

$SDi$ = 11011101 ⊕ 01111101 = **10100000**
$SP$ = 11011101 ⊕ 10111101 = **01100000**
$\sum_{i=1}^{4} SDi_i + SP_i > \sum_{i=5}^{8} SDi_i + SP_i$
Region selected: $R_1$

| | | | | |
|---|---|---|---|---|
| ***SX*** | 0 ⊕ 1 = **1** | 0 ⊕ 1 = **1** | 0 ⊕ 1 = **1** | 0 ⊕ 0 = **0** |
| | 0 ⊕ 0 = **0** | 0 ⊕ 1 = **1** | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** |
| | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** |
| | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** |

**Fig. 12.** eMRSC(32,3,64) correcting a four bit-flip error.

| | $R_5$ | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

| | | |
|---|---|---|
| $SDi$ = 11011101 ⊕ 10001101 = **01010000** | | |
| $SP$ = 11011101 ⊕ 10111101 = **01100000** | | |
| $\sum_{i=1}^{2} SDi_i + SP_i = \sum_{i=3}^{4} SDi_i + SP_i$ | | |
| Region selected: $R_5$ | | |
| ***SX*** | 0 ⊕ 0 = **0** | 0 ⊕ 1 = **1** |
| | 0 ⊕ 1 = **1** | 0 ⊕ 0 = **0** |
| | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** |
| | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** |

**Fig. 13.** eMRSC(32,7,56) correcting a two bit-flip error.

eMRSC(32,7,56) requires to verify if there is any draw situation between two adjacent regions of the set $Reg_1$ before identifying the region containing more detected errors. This verification procedure must occur because there are three draws solved by applying the Equations of Table 3. In this case, the decoding algorithm selects the region $R_5$. Subsequently, the data region is corrected with the $SX$ matrix.

Fig. 14 displays the last error correction example that uses eMRSC(32,7,56). If the errors happen in the redundancy bits, both eMRSC encodings can deliver the correct information in the presence of two bit-flips; i.e., the less complex MCU patterns. Errors in the redundancy bits break Step II of the decoding process.

## 6. Efficacy of the Error Correction Code

We performed an error correction experiment to evaluate the efficacy of eMRSC(32,3,64) and eMRSC(32,7,56) compared to four other well-known ECCs: Matrix(32,60), OLS(32,55), OLS(32,68) and DMC(32,68). All codes were developed for 32-bit data and implemented in MATLAB. The experiment encompasses eight scenarios inserting from one to eight errors into the codewords. The error patterns for scenarios with two or more errors employ only distance between bit-flips equal to one, which is the most probable distance to happen in memories operating in critical environments [11,25]. The term “distance 1” defines that the experiments do not accept patterns with two or more disjoint sets of errors; i.e., the test set only includes patterns containing a single grouping of errors, regardless of the number of errors in this grouping. Fig. 15(a) and (b) illustrates sets of acceptable and not acceptable error patterns, respectively.

The efficacy of the eMRSC codes is in the correction of errors that are grouped within the limited region defined for each eMRSC format; i.e., a 4 × 4 region for the eMRSC(32,3,64) code and a 2 × 8 region for the eMRSC(32,7,56) code. We choose only adjacent errors to address the patterns with a higher probability of occurrence. However, eMRSC is not able to correct all occurrences of non-adjacent errors, even if the occurrence is in the same region since the code is sensitive to the positioning and quantity of errors. The error correction experiment involves the following steps: (i) Generation of the 32-data bits (*D-32*); (ii) Encoding of *D-32* with the ECC selected to be analyzed; thus, creating a codeword (*C-32*); (iii) Selection of an error scenario and insertion of errors into *C-32*, producing codewords with errors on data or redundancy bits (*E-32*). The error pattern generation takes into account the works [11,25]; (iv) Decoding *E-32* to generate the decoded data (*DD-32*); (v) Register the corrected or uncorrected error by comparing the *DD-32* to *D-32*.

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

| | | |
|---|---|---|
| $SDi$ = 11011101 ⊕ 11011101 = **00000000** | | |
| $SP$ = 11011101 ⊕ 01111101 = **10100000** | | |
| Step II is violated since $SDi$ and $SP$ must be ≠ 0 to proceed the correction | | |
| Data bits will remain the same | | |
| ***SX*** | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** |
| | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** |
| | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** |
| | 0 ⊕ 0 = **0** | 0 ⊕ 0 = **0** |

**Fig. 14.** eMRSC(32,7,56) with two bit-flip error in the redundancy bits.

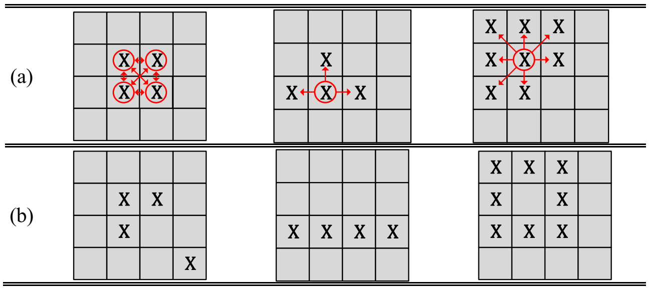


**Fig. 15.** (a) Acceptable error patterns as there is at least one error that is adjacent to all others and (b) Not-acceptable error patterns.

Fig. 16 presents the experimental results of the correction capability of all ECCs for all error scenarios.

For each error scenario, we produced all possible combinations of adjacent errors. All ECCs except DMC(32,68) and Matrix(32,60) reached 100% of correction rate for all error patterns with two errors; however, only OLS(32,68) maintained this rate with 3-error scenarios. The experimental results exhibit that eMRSC(32,3,64) presents the highest correction rates from four to eight errors. The reason for these high rates is the large size of the regions containing errors. The correction rates of eMRSC(32,3,64) are above 65% in all error scenarios; consequently, this code provides high reliability for memory banks. Besides, only in the 3-error scenario eMRSC(32,3,64) was surpassed.

From three to five errors, OLS(32,68) achieved better correction results than eMRSC(32,7,56). However, even though OLS(32,68) generates more redundancy bits, this code loses efficiency faster than eMRSC(32,7,56) - note that for six or more errors, eMRSC(32,7,56) presents better performance. The slopes of the curves representing the correction rates of eMRSC(32,3,64) and eMRSC(32,7,56) show that these ECCs are the ones less affected by the increase of errors.

Matrix(32,60), DMC(32,68) and OLS(32,55) are the codes with the lower capability of error correction. The last one presents a shorter version of OLS(32,68), losing efficacy abruptly for three or more bit-flips. The second one corrects multiple errors only for a specific set of error pattern. Finally, besides losing efficacy abruptly, Matrix(32,60) cannot correct more than three errors.

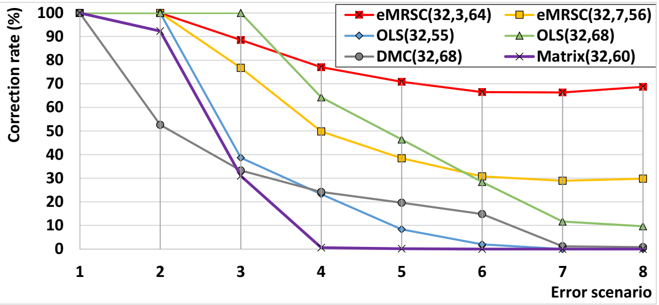


**Fig. 16.** Correction rates for each error scenario regarding five ECCs.

## 7. Reliability and MTTF estimations

This section evaluates the reliability and MTTF of a memory using eMRSC(32,3,64), eMRSC(32,7,56), Matrix(32,60), OLS(32,68), OLS (32,55) and DMC(32,68). These experiments were inspired by the MTTF analysis presented in [17], although the experiments consider that each MCU is the result of a single event that affected adjacent cells.

Let $w$ and $c$ be the data and redundancy bits of each codeword, $\lambda$ the error rate of a single bit, and $t$ the time parameter (in days) associated to $\lambda$; then, Eq. (15) computes $pEM(t)$, which is the probability of having codeword errors at time $t$. Note that $\lambda$ is a rate that depends on several aspects as the memory technology and the environment in which the memory is inserted (typical value of $\lambda$ is $10^{-5}$ upsets/bit/day) [17].

$$pEM(t) = 1 - e^{-(w+c)\lambda t} \tag{15}$$

Let $i$ be the number of errors; then, Eq. (16) estimates $pE_i(t)$, which is the probability of occurring $i$ errors in a given codeword of $(w + c)$ bits at time $t$.

$$pE_i(t) = \binom{w+c}{i} \times (1 - e^{-\lambda t}) \times e^{-\lambda(w+c-i)t} \tag{16}$$

Let $pCM_i$ be the probability of $i$ errors be correct by the evaluated ECC; then, $(pE_i(t) \times pCM_i)$ is the probability of correcting codewords with $i$ errors. Let $Me$ be the maximum number of errors that can arise (our experiments considers $Me = 8$); then, the expression $\sum_{i=1}^{Me} pE_i(t) \times pCM_i$ defines the probability of error occurrence on a codeword that can be corrected by the evaluated ECC at time $t$. Besides, $(1 - pEM(t))$ is the probability of not having errors in a memory over time $t$. Subsequently, Eq. (17) estimates $r(t)$, which is the reliability of a codeword at time $t$. Note that $pCM_i$ is extracted from the Efficacy of Error Correction experiment (Section 6); i.e., the values shown in Fig. 16.

$$r(t) = 1 - pEM(t) + \sum_{i=1}^{Me} pE_i(t) \times pCM_i \tag{17}$$

Let $M$ be the number of codewords containing a memory; then, Eq. (18) calculates $R(t)$, which is the memory device reliability at time $t$, through the product of the reliabilities of all codewords.

$$R(t) = r(t)^M \tag{18}$$

Eq. (19) describes the $MTTF$ of a memory device protected by an ECC is estimated by the integration of the reliability function over time $t$.

$$MTTF = \int_0^{\infty} R(t)\, dt \tag{19}$$

We developed this experiment in MATLAB applying for memories with $M \times (w + c)$ format, where $(w + c)$ is the codeword size of each ECC and $M$ is the number of codewords the memory encompasses. Figs. 17 to 19 present $R(t)$ considering $M = 1$, 8 and 16 over 8000 days (nearly 22 years).

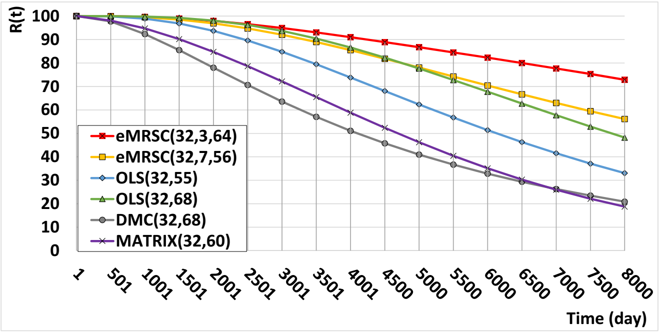


**Fig. 17.** Reliability estimated for five ECCs, considering several codeword sizes, $M = 1$, and $\lambda = 10^{-5}$ upsets/bit/day.

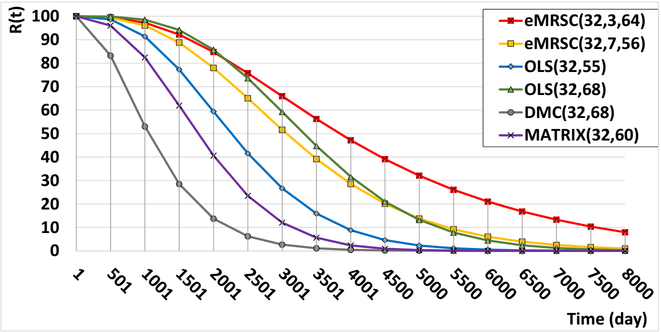


**Fig. 18.** Reliability estimated for five ECCs, considering several codeword sizes, $M = 8$, and $\lambda = 10^{-5}$ upsets/bit/day.

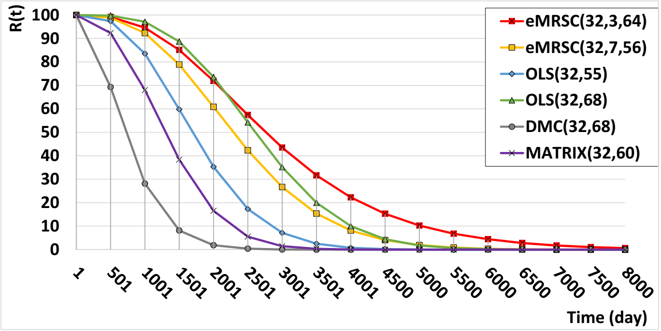


**Fig. 19.** Reliability estimated for five ECCs, considering several codeword sizes, $M = 16$, and $\lambda = 10^{-5}$ upsets/bit/day.

Figs. 17 to 19 display that eMRSC(32,3,64), eMRSC(32,7,56) and OLS(32,68) are the most reliable codes, mainly because they present higher correction rates. The numbers also show that by the day 2001, OLS (32,68) shows slightly more reliability than the two eMRSC codes. However, eMRSC(32,3,64) has fewer redundancy bits and displays greater error correction capability for more aggressive MCUs, making eMRSC(32,3,64) the most reliable code after the day 2501 of the experiment.

Table 4 presents the MTTF results attained with the integration of the curves of Figs. 17 to 19. These results indicate that eMRSC(32,3,64) is the most reliable code to be applied in all memory configurations evaluated. Note that OLS(32,68) surpassed eMRSC(32,7,56) by 5.2% and 11.6% for the two greatest memory configurations. DMC(32,68) presented the lowest MTTF results for all scenarios, with the difference from the eMRSC(32,3,64) code being 60.7%,108.5% and 278,4%.

## 8. Synthesis analysis

This section presents and discusses the synthesis results of MRSC(32,3,64), MRSC(32,7,56), OLS(32,55) and OLS(32,68). Besides, we defined and shown the *Correction per Total Cost* (CTC) evaluation parameter of these codes, which allows getting a relative evaluation of

**Table 4**
MTTF of memory with $M = 1$, 8 and 16, and $\lambda = 10^{-5}$ upsets/bit/day.

| ECC | M = 1 | M = 8 | M = 16 |
|---|---|---|---|
| eMRSC(32,3,64) | 7171 | 4159 | 2992 |
| eMRSC(32,7,56) | 6640 | 3272 | 2404 |
| OLS(32,68) | 6562 | 3442 | 2678 |
| OLS(32,55) | 5736 | 2393 | 1770 |
| Matrix(32,60) | 4990 | 1880 | 1360 |
| DMC(32,68) | 4459 | 1190 | 789 |

**Table 5**
Encoder synthesis results.

| ECC | Area ($\mu m^2$) | Power (mW) | Delay (ns) |
|---|---|---|---|
| eMRSC(32,3,64) | 947 | 0.311 | 0.20 |
| eMRSC(32,7,56) | 972 | 0.283 | 0.21 |
| OLS(32,68) | 1536 | 0.388 | 0.33 |
| OLS(32,55) | 1131 | 0.301 | 0.33 |

**Table 6**
Decoder synthesis results.

| ECC | Area ($\mu m^2$) | Power (mW) | Delay (ns) |
|---|---|---|---|
| eMRSC(32,3,64) | 1709 | 0.374 | 1.54 |
| eMRSC(32,7,56) | 1623 | 0.304 | 1.49 |
| OLS(32,68) | 3944 | 0.708 | 0.96 |
| OLS(32,55) | 2541 | 0.486 | 0.91 |

how much cost the code to get such reliability. We did not include in this section Matrix(32,60) and DMC(32,68) since the reliability results have shown that these codes are not suitable for dealing with MCUs, which is a primary focus of this work.

### 8.1. Synthesis of the encoders and decoders

The ECCs selected for this experiment were implemented in Verilog and synthesized with Cadence's *G.E.N.U.S* for 65 nm CMOS technology. Tables 5 and 6 present the synthesis results for the encoder and decoder designs.

Table 5 shows that the encoder synthesis costs of the OLS model are higher than the equivalent costs of the eMRSC model for all analyzed formats; except, only, in the comparison between power dissipation of OLS(32,55) and eMRSC(32,3,64).

Considering each ECC model separately, it is clear that the extra redundancy bits rise the area and power costs of the OLS model. On the other hand, the same effect is not shown in the eMRSC model, where the encoding circuit of eMRSC(32,7,56) consumes a little more area than the equivalent circuit of eMRSC(32,3,64); this additional area consumption is due to the complexity of eMRSC(32,7,56) in working with more error assessment regions. Finally, it is possible to note that the effect of redundancy on the encoder delay is negligible for both the OLS and the eMRSC models.

Table 6 illustrates that the decoder synthesis of eMRSC(32,3,64) achieved higher values than eMRSC(32,7,56) but without a meaningful difference. On the one hand, eMRSC(32,7,56) has fewer redundancy bits, reducing the total cost; on the other hand, the logic to correct errors is more complex than the first eMRSC, which increases the cost. When compared to the OLS model, the eMRSC model has fewer costs of area consumption and power dissipation on the decoder synthesis. Additionally, the decoder synthesis depicts the delays of both OLS codes are smaller than the delays of the eMRSC codes, which is explained by the fact that the detection and correction process of OLS is more straightforward than eMRSC. OLS uses majority logic to evaluate each bit and determine the correct value. Although this approach can be slightly faster, this method has a high implementation cost.

### 8.2. Correction per Total Cost (CTC) analysis

We applied a metric proposed in [17] to evaluate the tradeoff between reliability and synthesis cost of eMRSC(32,3,64), eMRSC(32,7,56), OLS(32,68) and OLS(32,55), considering the results acquired in the Error Correction Experiment (Section 6) and Synthesis Analysis (Section 8.1).

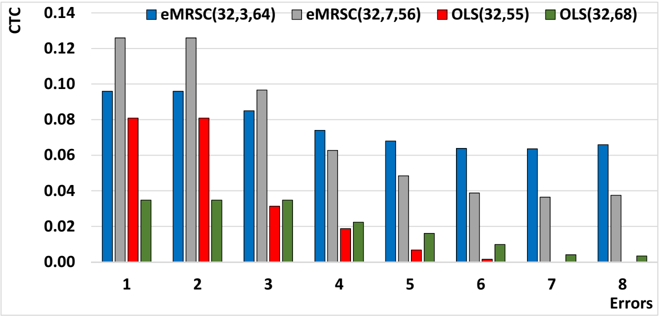


**Fig. 20.** CTC results.

Eq. (20) describes that the *Circuit Cost* of a particular circuit implementation is calculated by multiplying all the synthesis parameters of this circuit. Note that we use the same weight for all parameters since we are not considering specific design requirements here.

$$\textit{Circuit Cost}\,(\textit{circuit}) = \textit{Area} \times \textit{Power} \times \textit{Delay} \tag{20}$$

Eq. (21) presents the definition of the *ECC Cost* that takes into account the Encoder and Decoder circuits.

$$\textit{ECC Cost} = \textit{Circuit Cost}\,(\textit{Encoder}) + \textit{Circuit Cost}\,(\textit{Decoder}) \tag{21}$$

The metric Correction per Total Cost (*CTC*) divides the correction rate achieved in each error scenario (Fig. 16) by the *ECC Cost*. Eq. (22) describes the computation of the *CTC* metric, and Fig. 20 illustrates the application of *CTC* for each ECC and number of errors.

$$CTC = \frac{\textit{Correction Rate}}{\textit{ECC Cost}} \tag{22}$$

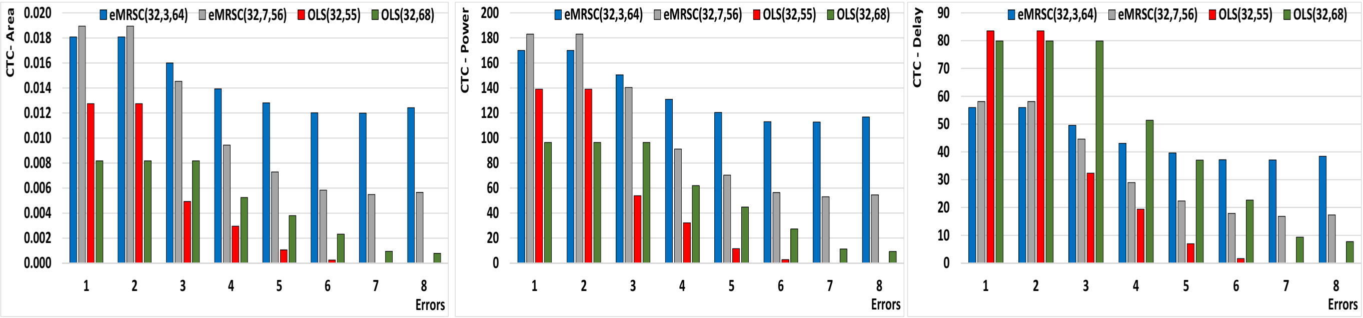
Fig. 20 displays that the eMRSC codes have the best tradeoff between correction rate and synthesis cost. Note also that the eMRSC(32,7,56) and eMRSC(32,7,56) trade positions during the analysis: from 1 to 3 errors, eMRSC(32,7,56) presents better *CTC* results, whereas from 4 to 8 errors, eMRSC(32,3,64) yields better ones. Note also that the eMRSC(32,7,56) and eMRSC(32,7,56) codes change position according to the number of errors analyzed: from 1 to 3 errors, eMRSC(32,7,56) reaches higher values of *CTC*, and from 4 errors, eMRSC(32,7,56) is surpassed by eMRSC(32,3,64). In more detail, the eMRSC(32,3,64) correction rates surpass eMRSC(32,7,56) by a considerable margin from 4 to 8 errors (more than 32%), which impacted the results in the *CTC* analysis. Although OLS(32,68) has correction rates higher than eMRSC(32,7,56), the synthesis cost of the former is much higher than the second one; consequently, the *CTC* results of both OLS codes never reach the *CTC* results of the eMRSC codes.

Further analysis of the CTC can be applied to observe the tradeoff of the ECCs considering a unique design requirement; i.e., area consumption, power dissipation and delay. Fig. 21(a) to (c) presents the results of CTC regarding Eq. (20) composed by only area, power or delay, respectively. Fig. 21(a) and (b) showed that both eMRSC presented the best tradeoff considering the total cost of area and power for all the error scenarios. However, Fig. 21(c) depicts that due to the delay of the eMRSC codes being higher than the OLS codes, only from 5 to 8 errors the eMRSC(32,3,64) surpass the OLS code.

## 9. Conclusions

This paper presented the expansion of Matrix Region Selection Code (MRSC) [13] from 16 to 32-bit data, which allows the development of two versions of MRSC: eMRSC(32,3,64) and eMRSC(32,7,56). These codes present data bit regions with different sizes, which result in alternative solutions to balance reliability versus area, power and delay overheads. Consequently, eMRSC becomes a more flexible solution to respond to the needs of different application requirements.

The proposed ECCs were compared with well-known codes found in the literature, presenting outstanding results in all experiments. In terms of error correction and reliability analysis, eMRSC(32,3,64)



(a) (b) (c)

**Fig. 21.** Error coverage per total (a) Area consumption, (b) Power dissipation and (c) Delay.

presented the best results, achieving more than 60% of correction errors in all scenarios and MTTF results nearly 10% better than the most robust version of OLS – OLS(32,68). Moreover, the synthesis analysis showed that the proposed codes are not only reliable but also of low implementation cost (i.e., low area, coding/decoding delay and power overheads).

The CTC analysis describes that the codes employing the eMRSC model present the best tradeoff between error correction and synthesis cost. Additionally, the CTC analysis considering area, power and delay separately shows that the eMRSC codes present the better tradeoff in area consumption and power dissipation. Therefore, both eMRSC codes proved to be ECC suitable options to be applied in critical applications that suffer from MCU occurrence.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001.

## References

[1] P. Hazucha, C. Svensson, Impact of CMOS technology scaling on the atmospheric neutron soft error rate, IEEE Trans. Nucl. Sci. 47 (6) (2000) 2586–2594 Dec.
[2] S. Liu, Y. Xiao, G. Mao, Extend orthogonal Latin square codes for 32-bit data protection in memory applications, Microelectron. Reliab. 63 (2016) 278–283. Aug.
[3] D. Radaelli, H. Puchner, S. Wong, S. Daniel, Investigation of multibit upsets in a 150 nm technology SRAM device, IEEE Trans. Nucl. Sci. 52 (6) (Dec. 2005) 2433–2437.
[4] R. Hentschke, F. Marques, F. Lima, L. Carro, A. Susin, R. Reis, Analyzing area and performance penalty of protecting different digital modules with hamming code and triple modular redundancy, Proceedings of Symposium on Integrated Circuits and Systems Design (SBCCI), 2002, pp. 95–100.
[5] M. Nicolaidis, Soft Errors in Modern Electronic Systems, Springer Science & Business Media, Nov, 2012 (318 pp.).
[6] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, 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 57 (7) (2010) 1527–1538 Jul.
[7] P. Ferreyra, C. Marques, R. Ferreyra, J. Gaspar, Failure map functions and accelerated mean time to failure tests: new approaches for improving the reliability estimation in systems exposed to single event upsets, IEEE Trans. Nucl. Sci. 52 (1) (2005) 494–500 Feb.
[8] V. Gherman, S. Evain, F. Auzanneau, Y. Bonhomme, Programmable extended SEC-DED codes for memory errors, Proceedings of IEEE VLSI Test Symposium (VTS), 2011, pp. 140–145.
[9] S. Baeg, S. Wen, R. Wong, SRAM interleaving distance selection with a soft error failure model, IEEE Trans. Nucl. Sci. 56 (4) (Aug. 2009) 2111–2118.
[10] S. Baeg, S. Wen, R. Wong, Minimizing soft errors in TCAM devices: a probabilistic approach to determining scrubbing intervals, IEEE Trans. Circuits Syst. Regul. Pap. 57 (4) (2010) 814–822. Apr.
[11] S. Satoh, Y. Tosaka, A. Wender, Geometric effect of multiple-bit soft errors induced by cosmic ray neutrons on DRAM's, IEEE Electron Device Lett. 21 (6) (Jun. 2000) 310–312.
[12] P. Rao, M. Ebrahimi, R. Seyyedi, M. Tahoori, Protecting SRAM-based FPGAs against multiple bit upsets using erasure codes, Proceedings of ACM/EDAC/IEEE Design Automation Conference (DAC), 2014, pp. 1–6.
[13] F. Silva, W. Freitas, J. Silveira, O. Lima, F. Vargas, C. Marcon, An efficient, low-cost ECC approach for critical-application memories, Proceedings of Symposium on Integrated Circuits and Systems Design (SBCCI), 2017, pp. 198–203.
[14] B. Varghese, S. Sreelal, P. Vinod, A. Krishnan, Multiple bit error correction for high data rate aerospace applications, Proceedings of IEEE Conference on Information Communication Technologies (ICT), 2013, pp. 1086–1090.
[15] E. Weiss, Generalized Reed-Muller codes, Inf. Control. 5 (3) (1962) 213–222 Sep.
[16] F. Vargas, M. Nicolaidis, SEU-tolerant SRAM design based on current monitoring, Proceedings of IEEE International Symposium on Fault Tolerant Computing (FTCS), 1994, pp. 106–115.
[17] C. Argyrides, H. Zarandi, D. Pradhan, Matrix codes: multiple bit upsets tolerant method for SRAM memories, Proceedings of IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT), 2007, pp. 340–348.
[18] P. Reviriego, C. Argyrides, J. Maestro, D. Pradhan, Improving memory reliability against soft errors using block parity, IEEE Trans. Nucl. Sci. 58 (3) (2011) 981–986 Jun.
[19] J. Guo, L. Xiao, Z. Mao, Q. Zhao, Enhanced memory reliability against multiple cells upsets using decimal matrix code, IEEE Trans. Very Large Scale Integr. VLSI Syst. 22 (1) (2014) 127–135 Jan.
[20] M. Hsiao, D. Bossen, R. Chien, Orthogonal Latin square codes, IBM J. Res. Dev. 14 (4) (1970) 390–394 Jul.
[21] A. Klockmann, G. Georgakos, M. Goessel, A new 3-bit burst-error correcting code, Proceedings of IEEE International Symposium on On-Line Testing and Robust System Design (IOLTS), 2017, pp. 3–4.
[22] F. Silva, J. Silveira, J. Silveira, C. Marcon, F. Vargas, O. Lima, An extensible code for correcting multiple cell upset in memory arrays, J. Electron. Test. 34 (4) (2018) 417–433. Aug.
[23] J. Li, L. Xiao, J. Guo, X. Cao, Efficient implementations of multiple bit burst error correction for memories, Proceedings of IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), 2018, pp. 1–3.
[24] J. Li, L. Xiao, P. Reviriego, R. Zhang, Efficient implementations of 4-bit burst error correction for memories, IEEE Trans. Circuits Syst. Express Briefs 65 (2) (Dec. 2018) 2037–2041.
[25] C. Ogden, M. Mascagni, The impact of soft error event topography on the reliability of computer memories, IEEE Trans. Reliab. 66 (4) (2017) 966–979 Dec.