INTEGRATION, the VLSI journal 84 (2022) 131–141

Contents lists available at ScienceDirect

Integration

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

OPCoSA: an Optimized Product Code for space applications

David Freitas [a,*], Jarbas Silveira [a], César Marcon [b], Lirida Naviner [c], João Mota [d]

[a] Engineering and Computer Systems Laboratory (LESC) - DETI, Federal University of Ceará, Fortaleza, Brazil
[b] Pontifical Catholic University of Rio Grande do Sul (PUCRS), Porto Alegre, Brazil
[c] Laboratoire Traitement et Communication de l'Information, Télécom Paris, Palaiseau, France
[d] Wireless Telecommunications Research Group (GTEL) - DETI, Federal University of Ceará, Fortaleza, Brazil

ARTICLE INFO

Keywords:
Error correction code
Fault tolerance
Radiation effect
Computer simulation

ABSTRACT

The integrated circuit shrinkage increases the probability and the number of errors in memories due to the increase in the sensitivity to electromagnetic radiation. Critical application systems employ Error Correction Codes (ECC) to mitigate memory faults. This work introduces the Optimized Product Code for Space Applications (*OPCoSA*), an ECC that optimizes its original version called *PCoSA*, reducing 16-redundancy bits and keeping high error correction capacity. We evaluated the optimized ECC through tests with 36 specific error patterns, burst errors, and exhaustive analysis. Additionally, we compared the synthesis results in hardware, reliability, and redundancy to four other ECCs dedicated to the space application. Tests have shown that *OPCoSA* corrects all 36 error patterns and 100% of cases for up to four burst errors; besides, it has correction rates of 100%, 100%, 95.4%, and 78.9% for exhaustive errors of dimension one to four, respectively.

1. Introduction

The continuous decrease of electronic devices allows increasing the storage capacity of memory circuits significantly. On the other hand, this decrease implies an increase in the number of errors caused by electromagnetic radiation, mainly in space applications. These errors occur due to the ionizing radiation from particles, such as protons, neutrons, heavy ions, alpha particles, and high-energy electrons. This radiation can change the contents of memory cells, causing faults that can be spread throughout the computational system, producing severe damage [1–4].

The radiation effects and consequent permanent or transient change of cells have been studied for almost 60 years [5–7]. There are several approaches to mitigate this problem in space applications, such as using shielding, hardened cells or Triple Modular Redundancy (TMR), changing process technology, or applying Error Correcting Codes (ECCs) [3]; this paper focuses on this latter approach.

In recent years, ECCs have been widely used to correct errors in critical system memories. This technique considers a codeword composed of data and redundancy bits. The technique requires additional circuitry to the computational system to encode and decode the codewords allowing to detect and correct errors in memory cells [8–12].

The first ECCs were designed to correct or detect single or double errors, known as Single Error Correction - Double Error Detection (SECDED). The electronic device shrinking increased the number of multiple errors in digital circuits [8]. For this reason, ECCs have evolved, obtaining a higher correction and detection capacity; thus, two-dimensional or product ECCs have emerged [1,4].

Morán et al. [1] explain that these complex ECCs are built for use in critical applications, such as space systems. These ECCs have high redundancy, raising area consumption, power dissipation, and critical path delay.[1]

This work proposes the Optimized Product Code for Spatial Application (*OPCoSA*), a matrix format ECC that protects 16 data bits employing 32 redundancy bits, i.e., 16 fewer redundancy bits than its predecessor, PCoSA [3]. *OPCoSA* maintains part of the PCoSA encoding structure, eliminating one of the check bit areas and associated equations. The main originality of this work is the new decoding algorithm, which reduces the synthesis cost by up to 60% in some cases, keeps correction rates close to PCoSA (average reduction of 7.2% up to six bitflips), and improves code reliability.

* Corresponding author.
E-mail address: davidciarlinifreitas@gmail.com (D. Freitas).

[1] The term *Matrix* refers to the linear code block proposed by Argyrides, Zarandi and Pradhan [13]. To avoid confusion between the ECC proposed by [13] and the matrix structure, all ECCs in this work are in italics.

https://doi.org/10.1016/j.vlsi.2022.02.005
Received 18 November 2020; Received in revised form 21 September 2021; Accepted 12 February 2022
Available online 14 February 2022

2. State-of-the-Art

There are several studies to mitigate memory faults using two-dimensional ECCs. The main current works come from the *Matrix* [13–15]; in turn, *Matrix* is based on Elias [16], which introduced in 1954 the pioneering work containing a simple and effective way to build long codes based on structures of smaller codes.

Castro et al. [17] describe the Column-Line Code (*CLC*) implementation and evaluation to detect and correct multiple errors in memories using extended Hamming and parity bits. The ECC is described in a 5 × 8 matrix format, with 16-data bits and 24-redundancy bits, forming the *CLC* (40,16). The authors show that *CLC* has more advantages than Reed-Muller (*RM*) and *Matrix* for scenarios with more than three upsets. Silva et al. [18] propose other ECC formats based on *CLC*, concluding that the highest correction efficiency occurs with the CLC (16,54) format in the extended form; however, this format has the highest area consumption. In 2020, Silva et al. [18] propose *CLC-A*, an ECC that introduces a syndrome analysis circuit to the system to check whether a second check is needed to correct the data. The experimental results show that *CLC-A* achieves a higher correction rate than *CLC* and the correction values are close to the values of the extended *CLC*, having a significant cost reduction concerning its extended version.

In [19–22], the authors implement approaches with the Horizontal Vertical Diagonal (*HVD*) technique, also called the 3D technique for applying horizontal, vertical, and diagonal codes in a matrix format. Tambatkar et al. [19] use the *HVD* and Hamming techniques to increase error correction capacity. The experimental results show that the correction capacity is three bits of data plus three bits of redundancy. The proposal is tested for 32, 64, and 128 data bits, resulting in a reduction of the encoder and decoder delays and the power dissipated by the encoder concerning the Multidirectional Parity Check (*MPC*) code. Raha e Murty [20] propose the Horizontal-Vertical Parity and Diagonal Hamming (*HVPDH*) method to detect up to eight errors, correct 100% of cases up to two, and correct most combinations of three to five errors. *HVPDH* is tested for 32-bit data words with 28-bit redundancy, and the results are compared with other ECCs that are also based on the Matrix code. Sai et al. [21] propose a technique for detecting and correcting multiple errors using Hamming code in the diagonal direction. This approach detects up to eight errors and corrects up to five errors; some combinations of six to eight errors are also correctable. The proposal attains a high correction rate with less area consumption and delay than the 3D *Parity Check Code*. Neelima e Subhas [22] propose an ECC based on *HVD* codes with format (data, redundancy) of sizes (64, 39) and (64, 67). The authors verified the maximum number of bits that can be corrected in each proposal but do not carry out a study with error injection to find the error correction rate of the proposed codes.

In [2,23], the authors propose an ECC in two dimensions divided by regions. Silva et al. [23] developed and validated the Matrix Region Section Code (*MRSC*), an ECC with low implementation cost to detect and correct multiple errors in memories. The code is structured in a 4 × 8 matrix, with 16 data bits and 16 redundancy bits, achieving a correction capacity similar to *CLC* and better than *Matrix*, with lower implementation costs than both ECCs. The authors in [2] develop the Extended Matrix Region Selection Code (*eMRSC*), an improved version of *MRSC* that extends the original 16 data bits to 32 bits. The authors propose an error correction scheme by region to reduce the number of generated redundancy bits. *eMRSC* is compared with the Orthogonal Latin Square (*OLS*), Decimal Matrix Code (*DMC*), and Matrix codes, presenting several tradeoffs; e.g., up to three bitflips, *OLS* has the highest error correction rates; however, with more errors, *eMRSC* achieves better results, being a low cost of implementation ECC.

Afrin e Sadi [24] propose an ECC in a 4 × 16 matrix format, with 32 data bits and 32 redundancy bits, which corrects 100% of the scenarios evaluated up to 8 bitflips. The tests are performed exhaustively only for the data bits, and the results are compared with *Matrix*, *HVD*, and *DMC* codes. Erozan e Çavus [25] propose a method of fault correction using a two-dimensional structure that is based on Single Parity Check (*SPC*) for coding the columns and Low-Density Parity Check (*LDPC*) for coding the lines. The proposed code can provide up to 95% correction coverage for up to 4 upsets, and the approach improves MTTF by 63% compared to the *Matrix* code. Morán et al. [26] present two two-dimensional ECCs designed to correct adjacent error patterns; both codes have the same error coverage with different redundancy levels, i.e., 8 and 16 bits. The correction approach is based on the Flexible Unequal Error Control (*FUEC*) methodology, developed to satisfy a certain number of syndromes. *FUEC* was designed to correct errors of one bit, as well as 2 and 3 adjacent bits in the same row or column. It can also correct errors in a 2 × 2 two-dimensional format.

Li et al. [27] tested two ECC proposals in a matrix format for 16-, 32- and 64-bit words. For 32-bit words, the proposals add 14 and 24 redundancy bits for data formats of 2 × 16 and 4 × 8, respectively. The code used on each line allows for correcting up to three burst errors. The authors use the interleaving technique in the proposed schemes. The 24-bit redundancy approach increases the correction capacity by up to 300% compared to the two 32-bit ECCs with 20 and 24 redundancy bits. Priya e Vijay [28] detail and analyze the Improved Redundant Matrix Code (*IRMC*), which adds 32 redundancy bits to protect 32 data bits. The authors do not carry out correction tests with any fault injection method and focus on implementing an FGPA.

The authors in [29,30] suggest improving the reliability of the system by adding an extra circuit to the circuits that deal with the ECC. Liu et al. [29] propose implementing a system to improve the reliability of encoders and decoders of *Matrix* codes. The scheme can detect all errors derived from a single node in the encoder and decoder circuits. The results show that the proposal has lower area and energy overloads, using a simple technique. Athira e Yamuna [30] test the *Matrix* code with an extra reliability system that reuses the encoder inside the decoder. The experimental results show that the proposed system has a higher error correction rate in relation to *RM* and *Hamming* ECCs and lower than *DMC*, with less delay and power dissipation among all ECCs.

The papers presented tend to use two-dimensional codes to mitigate multiple errors in memories used in critical applications. In 2020, the *PCoSA* product code for space applications [3] was proposed, composed of 16 data bits and 48 redundancy bits *PCoSA* (64,16), providing a high capacity to correct multiple errors. Exhaustive tests show that *PCoSA* corrects 100% of cases for up to three bitflips and has an 87% correction rate for four errors. *PCoSA* was compared with other codes used in space applications of one and two dimensions, such as *PBD* [37], *CLC* [17,18], *RM* [31], and *Matrix* [13], obtaining the highest efficiency in error correction. In [37], the authors present an error detection and correction approach called Parity per Byte and Duplication to protect data stored in memory. The technique uses parity at each byte and duplicates all content to perform the correction. Thus, two bytes need 20 bits of redundancy, forming the *PBD* (36,16). The traditional Reed-Muller code [31] used in this work has the parameters $r = 2$ and $m = 5$; $n = 2^m$ and $k = \sum_{i=0}^{r} \binom{m}{i}$ give the codeword and message sizes to be encoded, respectively, forming *RM*(32,16). Finally, *Matrix* [13] is a product code with 16 bits of data in a 4 × 4 matrix format that uses only *Ham*(7,4) in rows and parity in columns. Therefore, it has 16 bits of redundancy and a total of 32 bits, forming the *Matrix* (32,16).

3. *PCoSA* structure

PCoSA is a product code that combines two linear codes $C_1(n_1, k_1)$ and $C_2(n_2, k_2)$, denoted by $C_1 \times C_2$, as shown in Fig. 1. This code applies two sets of check bits in a two-dimensional format. The data is written in a $k_1 \times k_2$ matrix. Each line k_2 is coded using C_1, forming n_1 columns. Each column n_1 is coded using C_2, forming the $n_1 \times n_2$ matrix. The product code linearity allows us to start coding C_1 and then C_2 and vice versa [31–33]. As C_1 has a minimum distance d_1 and C_2 has a minimum

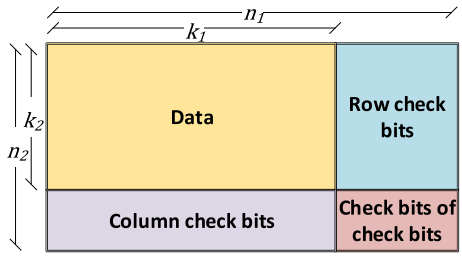

Fig. 1. Product code with $k_1 \times k_2$ data bits and $n_1 \times n_2$ total bits after encoding a word with codes C_1 and C_2 (based on [16]).

distance d_2, so the product code has a minimum distance $d_1 \times d_2$. As the code distance increases, the greater the detection and correction capacity about the one-dimensional ECCs [32].

The Minimum Distance (d) of a code is the smallest number of bit changes required to change from any codeword to any other codeword; it is a metric used to measure the code capacity in correcting and detecting errors. Equations (1) and (2) calculate the maximum number of errors *in any codeword position* that a Hamming-based code can correct *ec* or detect *ed* [31], respectively.

$$\mathrm{ec} = (d-1)/2 \tag{1}$$

$$\mathrm{ed} = d - 1 \tag{2}$$

Equations (1) and (2) are exclusives, i.e., *ec* or *ed*, but not *ec* and *ed* simultaneously. The simultaneity relationship among *ec*, *ed*, and d is given by Equation (3) (further details in [36]). It is important to point out that the range of the value given by Equations (1) and (2) depends on the decoding method, increasing or even decreasing this value.

$$\mathrm{ed} = \mathrm{d} - \mathrm{ec} - 1 \tag{3}$$

PCoSA code is similar to *CLC* [17], but it applies extended *Hamming* to both rows and columns. Fig. 2 shows *PCoSA* with 16 data bits (D_0 to D_{15}), 12-row check bits ($C1_0$ to $C1_{11}$), 7-row parity bits ($P1_0$ to $P1_6$), 21-column check bits ($C2_0$ to $C2_{20}$), and 8-column parity bits ($P2_0$ to $P2_7$). This format has 48 redundancy bits and a minimum distance of 16; since C_1 and C_2 are extended *Hamming* codes with $d_1 = d_2 = 4$. *PCoSA* encoding/decoding equations are available in [3].

4. *OPCoSA* definition

Fig. 3 shows the *OPCoSA* organization consisting of 16 data bits (D_0 to D_{15}), 12-row check bits ($C1_0$ to $C1_{11}$), 4-row parity bits ($P1_0$ to $P1_3$), 12-column check bits ($C2_0$ to $C2_{11}$), and 4-column parity bits ($P2_0$ to $P2_3$); it is the same organization as *PCoSA* [3], but without the check bits of check bits region, reducing 16-redundancy bits (33% of reduction). This modification removes *OPCoSA* from the product code class, being considered a modified product code.

This reduction of a PCoSA check-bit region to form the *OPCoSA* and the consequent alteration of the decoding algorithm is the focus of this work. This change in format makes the minimum distance of a modified product code, obtained by Equation (4), less than a conventional product code [31,33].

D_0	D_1	D_2	D_3	$C1_0$	$C1_1$	$C1_2$	$P1_0$
D_4	D_5	D_6	D_7	$C1_3$	$C1_4$	$C1_5$	$P1_1$
D_8	D_9	D_{10}	D_{11}	$C1_6$	$C1_7$	$C1_8$	$P1_2$
D_{12}	D_{13}	D_{14}	D_{15}	$C1_9$	$C1_{10}$	$C1_{11}$	$P1_3$
$C2_0$	$C2_1$	$C2_2$	$C2_3$	$C2_4$	$C2_5$	$C2_6$	$P1_4$
$C2_7$	$C2_8$	$C2_9$	$C2_{10}$	$C2_{11}$	$C2_{12}$	$C2_{13}$	$P1_5$
$C2_{14}$	$C2_{15}$	$C2_{16}$	$C2_{17}$	$C2_{18}$	$C2_{19}$	$C2_{20}$	$P1_6$
$P2_0$	$P2_1$	$P2_2$	$P2_3$	$P2_4$	$P2_5$	$P2_6$	$P2_7$

Fig. 2. *PCoSA* structure with 16-data bits.

D_0	D_1	D_2	D_3	$C1_0$	$C1_1$	$C1_2$	$P1_0$
D_4	D_5	D_6	D_7	$C1_3$	$C1_4$	$C1_5$	$P1_1$
D_8	D_9	D_{10}	D_{11}	$C1_6$	$C1_7$	$C1_8$	$P1_2$
D_{12}	D_{13}	D_{14}	D_{15}	$C1_9$	$C1_{10}$	$C1_{11}$	$P1_3$
$C2_0$	$C2_1$	$C2_2$	$C2_3$				
$C2_4$	$C2_5$	$C2_6$	$C2_7$				
$C2_8$	$C2_9$	$C2_{10}$	$C2_{11}$				
$P2_0$	$P2_1$	$P2_2$	$P2_3$				

Fig. 3. *OPCoSA* structure with 16 data bits.

$$d_{C_1C_2} = d_{C_1} + d_{C_2} - 1 \tag{4}$$

Equation (4) shows that *OPCoSA* has a minimum distance of 7 because it uses a minimum distance in both codes (rows and columns) equal to 4. The organization of the *OPCoSA* matrix causes rows and columns to cross in just a single bit, and changing this bit implies the variation of three other bits in the line and three other bits in the column, thus modifying 7 bits; i.e., the minimum distance of 7.

Fig. 4 describes the *OPCoSA* encoding and decoding processes, which are detailed by applying Equations (5)–(25).

Using q as the bit position index and $\oplus$ an XOR operation, *OPCoSA* coding employs Equations (5)–(7) to compute the row parity check bits, Equation (8) to calculate the row parity bits, Equations (9)–(11) to calculate the column parity check bits, and Equation (12) to compute the column parity.

$$C1_q = D_{\frac{4q}{3}} \oplus D_{\frac{4q}{3}+1} \oplus D_{\frac{4q}{3}+3},\ \forall q \in \{0,3,6,9\} \tag{5}$$

$$C1_{q+1} = D_{\frac{4q}{3}} \oplus D_{\frac{4q}{3}+2} \oplus D_{\frac{4q}{3}+3},\ \forall q \in \{0,3,6,9\} \tag{6}$$

$$C1_{q+2} = D_{\frac{4q}{3}+1} \oplus D_{\frac{4q}{3}+2} \oplus D_{\frac{4q}{3}+3},\quad \forall q \in \{0,3,6,9\} \tag{7}$$

$$P1_q = D_{4q} \oplus D_{4q+1} \oplus D_{4q+2} \oplus D_{4q+3} \oplus C1_{3q} \oplus C1_{3q+1} \oplus C1_{3q+2},\ \forall 0 \le q \le 3 \tag{8}$$

$$C2_q = D_q \oplus D_{q+4} \oplus D_{q+12},\ \forall 0 \le q \le 3 \tag{9}$$

$$C2_{q+4} = D_q \oplus D_{q+8} \oplus D_{q+12},\ \forall 0 \le q \le 3 \tag{10}$$

$$C2_{q+8} = D_{q+4} \oplus D_{q+8} \oplus D_{q+12},\ \forall 0 \le q \le 3 \tag{11}$$

$$P2_q = D_q \oplus D_{q+4} \oplus D_{q+8} \oplus D_{q+12} \oplus C2_q \oplus C2_{q+4} \oplus C2_{q+8},\ \forall 0 \le q \le 3 \tag{12}$$

In *OPCoSA* decoding, Equations (13)–(15) and (17)–(19) calculate the recalculated check bits *rC*1 (rows) and *rC*2 (columns), respectively; also, Equations (16) and (20) recalculate the parity bits of the rows and columns, respectively.

$$rC1_q = D_{\frac{4q}{3}} \oplus D_{\frac{4q}{3}+1} \oplus D_{\frac{4q}{3}+3},\ \forall q \in \{0,3,6,9\} \tag{13}$$

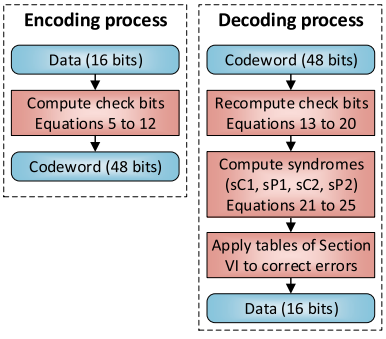

Fig. 4. *OPCoSA* encoding and decoding process, relating the equations that are used in each of the steps.

$$rC1_{q+1} = D_{\frac{4q}{3}} \oplus D_{\frac{4q}{3}+2} \oplus D_{\frac{4q}{3}+3},\ \forall q \in \{0,3,6,9\} \quad (14)$$

$$rC1_{q+2} = D_{\frac{4q}{3}+1} \oplus D_{\frac{4q}{3}+2} \oplus D_{\frac{4q}{3}+3},\quad \forall q \in \{0,3,6,9\} \quad (15)$$

$$rP1_q = D_{4q} \oplus D_{4q+1} \oplus D_{4q+2} \oplus D_{4q+3} \oplus C1_{3q} \oplus C1_{3q+1} \oplus C1_{3q+2},\quad \forall 0 \le q \le 3 \quad (16)$$

$$rC2_q = D_q \oplus D_{q+4} \oplus D_{q+12},\quad \forall 0 \le q \le 3 \quad (17)$$

$$rC2_{q+4} = D_q \oplus D_{q+8} \oplus D_{q+12},\ \forall 0 \le q \le 3 \quad (18)$$

$$rC2_{q+8} = D_{q+4} \oplus D_{q+8} \oplus D_{q+12},\quad \forall 0 \le q \le 3 \quad (19)$$

$$rP2_q = D_q \oplus D_{q+4} \oplus D_{q+8} \oplus D_{q+12} \oplus C2_q \oplus C2_{q+4} \oplus C2_{q+8},\ \forall 0 \le q \le 3 \quad (20)$$

Equations (21)–(24) perform the next decoding step by calculating $sC1$ and $sC2$, which are the check bits syndromes $C1$ and $C2$, respectively, and by calculating $sP1$ and $sP2$, which are the syndromes of the parity bits of the rows and columns, respectively. These four syndromes constitute the vector $S = [sC1\ sP1\ sC2\ sP2]$.

$$sC1 = \sum_{q=0}^{3} \left(C1_{3q} \oplus rC1_{3q}\right) + \left(C1_{3q+1} \oplus rC1_{3q+1}\right) + \left(C1_{3q+2} \oplus rC1_{3q+2}\right) \quad (21)$$

$$sC2 = \sum_{q=0}^{3} \left(C2_q \oplus rC2_q\right) + \left(C2_{q+4} \oplus rC2_{q+4}\right) + \left(C2_{q+8} \oplus rC2_{q+8}\right) \quad (22)$$

$$sP1 = \sum_{q=0}^{3} P1_q \oplus rP1_q \quad (23)$$

$$sP2 = \sum_{q=0}^{3} P2_q \oplus rP2_q \quad (24)$$

The decoding algorithm uses $S = [sC1\ sP1\ sC2\ sP2]$ in binary format $S_b = [sc1\ sp1\ sc2\ sp2]$. The way to perform this calculation is given by Equation (25). For example, if $S = [2\ 1\ 2\ 0]$, then $S_b = [1\ 1\ 1\ 0]$.

$$sx = \begin{cases} 0, & if\ sX = 0 \\ 1, & else \end{cases} \quad (25)$$

The sequence of the decoding algorithm is based on the *OPCoSA* correction method, presented in Section VI.

5. Experimental setup and methodology

In the experimental setup, the potential of *OPCoSA* was assessed through three test cases and compared with *PCoSA* and other four ECCs used in space applications. Fig. 5 describes the methodology used to obtain and evaluate *OPCoSA*. Note that activities 1 and 2 comprise the steps to design the *OPCoSA* decoding algorithm, while the other activities contain the steps to collect the experimental results.

Activity 1 presents the mapping of 36 error patterns shown in Fig. 6; these standards were proposed in [34] and used to design the *OPCoSA* algorithm. These error patterns were obtained with a commercial tool using strike simulation of neutron particles. The tool has as input the radiation environment and memory and as output the MBUs patterns and their respective probabilities [34]. As these error patterns have a 3 × 3 matrix format, their analysis was performed in two parts with the mapping on the *OPCoSA* codeword: (i) first 4 rows to the 8th column and (ii) last 4 rows from columns 1 to 4. Thus, each error pattern was placed

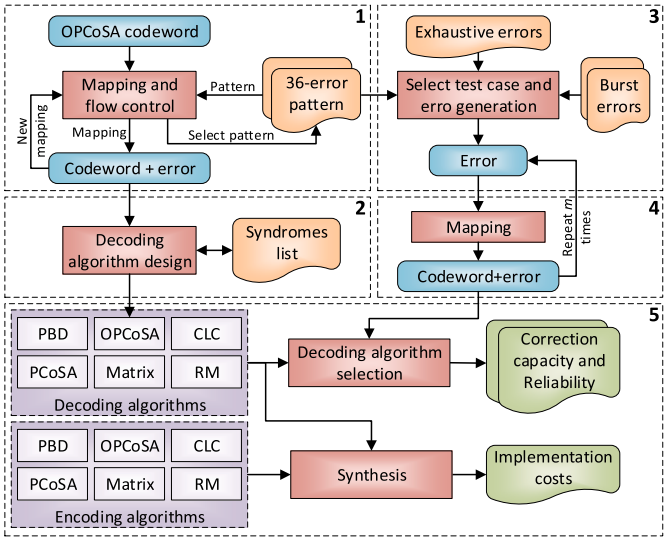

Fig. 5. Applied methodology containing the five main activities.

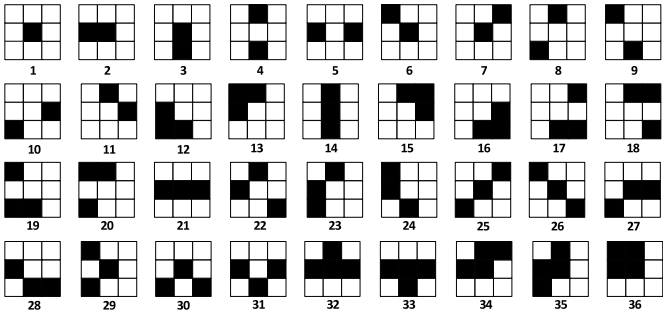

Fig. 6. Thirty-six error patterns used in the experiments, encompassing one simple error, ten double errors, twenty triple errors, and five quadruple errors (based on [34]).

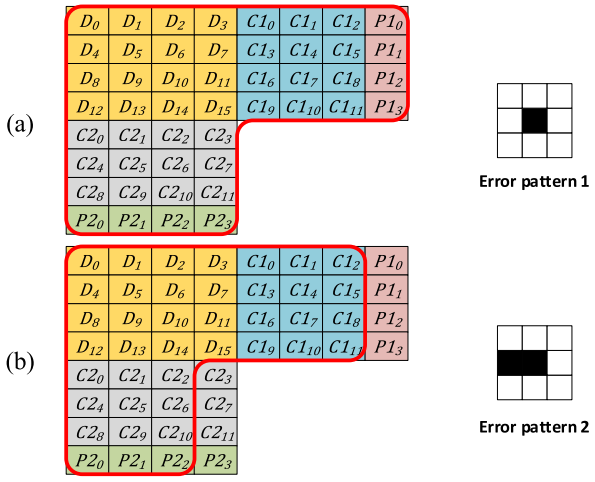

Fig. 7. Surrounded by the red line is the region that error pattern can be placed: (a) error pattern 1 can be placed in all positions, and (b) error pattern 2 can be placed in all positions except the rightest column.

in all possibilities σ_i, with i being the index to represent each of the 36 error patterns.

Fig. 7 exemplifies the positioning of the error patterns 1 and 2 in the areas described by (i) and (ii), respectively. Thus, the error pattern 1 can be placed in the 48 positions, as shown in Fig. 7(a), obtaining $\sigma_1 = 48$. In turn, error pattern 2 can be positioned only in the 40 positions indicated by the two regions of Fig. 7(b), obtaining $\sigma_2 = 40$. This decrease in the number of positions occurs because, as error patter 2 in Fig. 6 has two horizontally adjacent bitflips, and as we make the left bit as the reference for the test, it cannot be positioned in the last column because its adjacent bits would be outside the *OPCoSA* region. This mapping serves to find the syndromes of Section VI and to design the *OPCoSA* decoding algorithm.

Activity 2 describes the decoding algorithm design, which encompasses the evaluation of all mappings of the 36 error patterns to capture all syndromes in decimal and binary formats. In this activity, a table is created with the list of all syndromes found with the respective error type. Only binary syndromes that have errors in the data region are analyzed; this analysis indicates the correction method used for each syndrome found.

The three test cases are analyzed in Activity 3. While *PCoSA* analyzes only the 36 error patterns and exhaustive tests, this work adds a third test case that contemplates burst errors. A burst error is a multiple error that covers l contiguous bits in a word; where at least the first and last bits are wrong. The value l is known as the burst length. Note that adjacent errors are a specific type of burst error in which all the wrong bits are contiguous [1,26].

Activity 4 contemplates two ways of mapping errors in memory: (i) the mappings of the 36 error patterns illustrated in Fig. 6 and (ii) the mappings used for both burst and exhaustive errors; as Fig. 8 illustrates, form (ii) speeds up testing by treating *OPCoSA* as a contiguous 48-bit vector. The first eight bits of Fig. 8 are equivalent to the first line of Fig. 3, the bit intervals 9 to 16, 17 to 24, and 25 to 32 of Fig. 8 are equivalent, respectively, to lines 2, 3, and 4 Fig. 3; finally, the range from bit 33 to bit 48 in Fig. 8 is equivalent to the four-bit lines 5 to 8 in Fig. 3.

Activity 4 shows that for each test case, the error generation is repeated m times. In the case of 36 error patterns, $m = \sum_{i=1}^{36} \sigma_i$. The simulation is repeated $m_x = \binom{n}{x}$ times for each x bitflips in the exhaustive test. In the case of b *burst errors*, $m = (n - b + 1)2^{b-2}$, $m = n - 1$ and $m = n$, for $b > 2$, $b = 2$ and $b = 1$, respectively. For these cases, $n = 48$, which is the *OPCoSA* codeword size.

Activity 5 presents the correction capacity, reliability, and implementation costs of *OPCoSA* compared to *PCoSA*, *PBD, CLC*, *RM*, and *Matrix* codes. The correction data obtained in this activity were extracted from the works of [3,37].

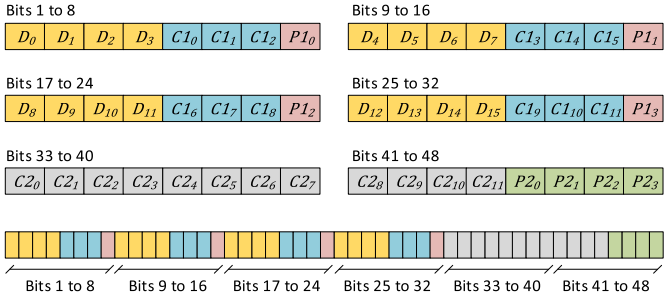

Fig. 8. *OPCoSA* representation in a 48-position vector format.

Table 1
Mapping of error patterns using S_b = [*sc*1*sp*1 *sc*2*sp*2].

	S_b	Error pattern	
		Type	Number and placement
	0 0 0 0	No error	–
	0 0 0 1		4 outside region *D*
	0 0 1 0		8 outside region *D*
	0 0 1 1		60 outside region *D*
	0 1 0 0		4 outside region *D*
	0 1 0 1	Unreachable syndrome	–
	0 1 1 0		
*	0 1 1 1	Patterns 21, 33	2 inside region *D*
	1 0 0 0		8 outside region *D*
	1 0 0 1	Unreachable syndrome	–
*	1 0 1 0	Patterns 13, 20, 36	5 inside region *D*
*	1 0 1 1	Patterns 2, 5, 13, 15, 18, 20, 27, 31, 34	17 inside region *D*
			1 outside region *D*
	1 1 0 0		60 outside region *D*
*	1 1 0 1	Pattern 14	1 inside region *D*
*	1 1 1 0	Patterns 3, 4, 12, 13, 19, 20, 23, 24, 29, 35	17 inside region *D*
			1 outside region *D*
*	1 1 1 1	Several patterns	94 in several regions

Lines marked with * are detailed in the next tables.

6. *OPCoSA* correction method

The *OPCoSA* correction method employs the same idea implemented in *PCoSA*. However, the algorithm has been completely changed with the reduction from 64 to 48 bits.

The *OPCoSA* algorithm was designed to correct the 36-error pattern illustrated in Fig. 6. Each error pattern was placed in all five regions of Fig. 3 (data D, row check bits C_1, row parity P_1, column check bits C_2, and column parity P_2); consequently, all cases were considered. For example, error pattern 2 was placed in regions D, $D \cup C_1$, C_1, $C_1 \cup P_1$, C_2 and P_2.

Table 1 describes the positioning of all 36 error patterns in all regions, including all sixteen combinations of S_b. The correction algorithm works only if at least one error occurs in region D; i.e., only if S_b = [0111], [1010], [1011], [1101], [1110] or [1111]. Besides, Table 1 shows the type, number, and placement of the error patterns.

After mapping each S_b, the decoding algorithm searches for the syndrome-based error pattern, as shown in the following tables. This procedure is done for the first five S_b patterns; for S_b = [1111], there is one more step.

S_b = [0 1 1 1]

Pattern	S	Correction method
21	[0 1 3 3]	Apply *Hamming* to all columns and then *Hamming* to all rows
33	[0 1 3 2]	
Default: Apply *Hamming* to all columns and then *Hamming* to all rows		

S_b = [1 0 1 0]

Pattern	S	Correction method
13, 20	[1 0 1 0]	Invert the intersecting bit between *sC*1 and *sC*2
36	[2 0 2 0] [2 0 1 0] [1 0 2 0]	The four bitflips are corrected by referencing the upper left bit of the pattern, which is found using the upper *sC*1 and the leftmost *sC*2. The position of this reference bit allows us to change the other three bitflips
Default: Apply *Hamming* to all rows		

S_b = [1 0 1 1]

(*continued on next page*)

(*continued*)

Pattern	S	Correction method
2, 5, 27, 31	[1 0 2 2]	Apply *Hamming* to all columns
2, 5	[1 0 1 1]	
13, 15, 18, 20, 34	[1 0 2 1]	Invert the bit indicated by the double error above and by the double error indicated by the column. After that, apply *Hamming* to all columns
27, 31	[1 0 3 3]	Apply *Hamming* to all columns
34	[2 0 1 1]	
34	[2 0 3 2]	Invert the bit indicated by the double error above and by the double error indicated by the column. After that, apply *Hamming* to all columns
34	[2 0 2 1]	
34	[1 0 3 2]	
Default: Apply *Hamming* to all columns		

S_b = [1 1 0 1]

Pattern	S	Correction method
14	[3 3 0 1]	Apply *Hamming* to all rows
Default: Apply *Hamming* to all rows		

S_b = [1 1 1 0]

Pattern	S	Correction method
3, 4, 29	[2 2 1 0]	Apply *Hamming* to all rows
3, 4	[1 1 1 0]	
12, 13, 19, 20, 35	[2 1 1 0]	Invert the bit indicated by the double line error and the leftmost one by the column and then applies *Hamming* to all lines
23, 24, 29	[3 3 1 0]	Apply *Hamming* to all rows
35	[3 2 2 0]	Invert the bit indicated by the double line error and the leftmost one by the column and then applies *Hamming* to all lines
35	[3 2 1 0]	
35	[2 1 2 0]	
35	[1 1 2 0]	Apply *Hamming* to all rows
Default: Apply *Hamming* to all rows		

Let r and c be the numbers of errors in the rows and columns, respectively, S_b = [1111] occurs with error patterns in the format $r \times c$. In cases of miscorrection or in cases where the error is in regions such as $D \cup C_2$, the error pattern can have diverse dimensions. The *OPCoSA* decoding algorithm uses Equations (26)–(28) to compute the Error Size (*ES*).

$$ES = r \times c \tag{26}$$

$$r = \max(sC1, sP1) \tag{27}$$

$$c = \max(sC2, sP2) \tag{28}$$

The table below presents all the *ES* possibilities, the corresponding error pattern S, and the correction method.

S_b = [1 1 1 1]

ES	Pattern	S	Correction method
1 × 1	1 14 21	1111	If the position of the row error is 1 and Column 7, apply *Hamming* to all columns. Otherwise, apply *Hamming* to all rows
1 × 2	35	1120 1121	Apply *Hamming* to all rows
	12, 16, 17, 19, 22, 23, 24, 29, 33		
	6, 7, 8, 9, 10, 11, 21	1122	
1 × 3	32	1132	Apply *Hamming* to all rows
	21, 25, 26, 28, 30	1133	Apply *Hamming* to all columns
2 × 1	15, 16, 17, 18, 27, 28, 30, 31	2111	
	6, 7, 8, 9, 10, 11, 14, 32, 33	2211	
2 × 2	12, 13, 15, 16, 17, 18, 19, 20	2121	Invert the bit indicated by row double errors and then apply *Hamming* to all columns
	27, 28, 30, 31	2122	Apply *Hamming* to all columns
	22, 23, 24, 29, 32, 33	2221	Apply *Hamming* to all rows
	6, 7, 8, 9, 10, 11, 25	2222	Apply *Hamming* to all columns
	32	1221	Apply *Hamming* to all rows and then to all columns
2 × 3	32, 33	1232	Invert the two bits indicated by the two rows and the double error column and then apply *Hamming* to all columns
	32, 33	2232	
	27, 28, 30, 31	2133	Apply *Hamming* to all columns
	25, 26	2233	
3 × 1	14, 22, 25, 26	3311	Apply *Hamming* to all rows
3 × 2	22, 23, 24, 29	3321	
	25, 26	3322	
3 × 3	25, 26	3333	
Default: Check r and c (Equations (27) and (28)). If $r \geq c$, apply *Hamming* to all rows. Otherwise, apply *Hamming* to all columns.			

The default conditions enable to correct several patterns in addition to those presented by the set of 36 error patterns. For example, the default condition for error patterns with S_b = [1111] is "Check r and c (Equations (27) and (28)). If $r \geq c$, apply Hamming to all rows. Otherwise, apply Hamming to all columns". The pattern that has four diagonal errors (bits D0, D5, D10, and D15), for instance, is corrected because S_b = [1111], S = [4444], and the default condition would do the correction of all bits. Fig. 9 exemplifies another 4-bit error pattern that *OPCoSA* can correct, which is not included in the 36 error patterns. The syndromes of this error pattern are S_b = [1111] and S_b = [2232] (*ES* = 2 × 3). Thus, the decoding algorithm corrects "Inverting the two bits indicated by the two rows and the double error column and then apply Hamming to all columns". The correction process is done in two parts: (i) first, D5 and D9 are corrected; then, (ii) the complete correction is performed applying Hamming to all columns to correct D6 and D7 bits.

7. Exploring scalability and redundancy rate

Equation (29) computes the Redundancy Rate (*rr*) metric that indicates the ratio between the redundancy (r) and codeword (r) bits. The higher the *rr*, the greater the weight of the redundancy bits. However, the lower the *rr*, the lower the redundancy impact; consequently, the lower the ECC cost.

$$rr = \frac{r}{n} \times 100\% \tag{29}$$

OPCoSA was designed to protect memories with words longer than 8 bits through code replication or code scaling. Replicating a smaller code is a technique that keeps *rr*, while code scaling reduces *rr*. Fig. 10 shows the decrease of *rr* as a function of the *Hamming* configuration. For instance, using *Ham*(8,4), the *rr* in the product code is 75% (case of *PCoSA* [3]), while a modified product code has *rr* = 66.6% (case of

OPCoSA). For all *Hamming* code configurations analyzed, *OPCoSA* has lower *rr* values than *PCoSA*, with the highest difference being 8.3% for the first case.

The *OPCoSA* scaling is achieved using other *Hamming* formats that preserve the same minimum *Hamming* distance (d) of four, making the modified product code to have the same $d = 7$; consequently, preserving the same correction and detection rates for all *OPCoSA* scaled versions. For instance, when using *Ham*(16, 11) and *Ham*(32, 26) in place of *Ham* (8, 4), increases the codeword length to the ones presented in Fig. 11 and Fig. 12, respectively.

On the one hand, the purpose of the *OPCoSA* configurations presented in Figs. 11 and 12 is to decrease *rr*, contributing to a code with less redundancy and, consequently, lower costs. For instance, *OPCoSA* (231, 121) and *OPCoSA* (988, 676) have *rr* equal to 47.6% and 31.5%, respectively, which is a significant reduction of the cost in bits compared to 66.7% of the basic *OPCoSA* (48, 16). On the other hand, since *OPCoSA* scaling keeps the number of bits detected and corrected, because it keeps the Hamming distance but increases the number of codeword bits, the detection and correction rate of the memory protected by *OPCoSA* is reduced.

Replication is performed using any *OPCoSA* code, such as the basic *OPCoSA* (48, 16) or other scaled versions as shown above - *OPCoSA* (231, 121) and *OPCoSA* (988, 676). Fig. 13 illustrates *OPCoSA* (192, 64) for protecting a memory with 64-bit words; this code format was achieved by replicating four *OPCoSA* (48, 16). The absence of one check-bit area implies that *OPCoSA* is replicated by rotating two *OPCoSA* codewords. Thus, *OPCoSA* can be written in an 8 × 24 region instead of the 8 × 32 region proposed in *PCoSA* [3], saving memory area.

This replicating process makes *OPCoSA* (48, 16) and *OPCoSA* (192, 64) have *rr* equal to 66.7%, which is a high cost in bits. However, the replication scaling keeps the number of bits detected and corrected, and also the detection and correction rate of the memory protected by *OPCoSA*.

8. Results and discussions

This section presents experimental results and discussions considering code correction capacity, reliability, and cost of implementation.

A. Error Correction Capability

To assess the correction capability of the 36 error patterns [34], *OPCoSA* was placed in a matrix format (Fig. 3), facilitating the mapping of each MBU in a 3 × 3 matrix format. The results display that, like *PCoSA*, *OPCoSA* obtained 100% of error correction for all 36 error patterns.

The error correction capacity evaluation, considering burst and exhaustive tests, is done with the vector-type structure presented in Fig. 8. *OPCoSA* can correct 100% of cases until burst error with $l = 4$; however, the ECC has 0% correction for $l > 4$.

The exhaustive tests were performed with sets of one to six bitflips. Fig. 14 shows that only *PCoSA* and *RM* have 100% correction up to three bitflips. *OPCoSA* achieves 100% error correction with two bitflips but reduces it to 95.4% with 3 bitflips. *CLC* performs better than *Matrix* and

D_0	D_1	D_2	D_3	$C1_0$	$C1_1$	$C1_2$	$P1_0$
D_4	~~D_5~~	~~D_6~~	~~D_7~~	$C1_3$	$C1_4$	$C1_5$	$P1_1$
D_8	~~D_9~~	D_{10}	D_{11}	$C1_6$	$C1_7$	$C1_8$	$P1_2$
D_{12}	D_{13}	D_{14}	D_{15}	$C1_9$	$C1_{10}$	$C1_{11}$	$P1_3$
$C2_0$	$C2_1$	$C2_2$	$C2_3$				
$C2_4$	$C2_5$	$C2_6$	$C2_7$				
$C2_8$	$C2_9$	$C2_{10}$	$C2_{11}$				
$P2_0$	$P2_1$	$P2_2$	$P2_3$				

Fig. 9. Error pattern example that is not part of the 36 patterns analyzed in Fig. 5 but is fixed by the *OPCoSA* decoding algorithm.

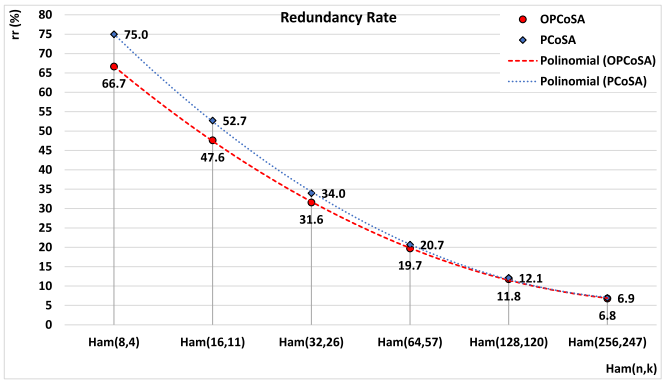

Fig. 10. *OPCoSA* and *PCoSA* redundancy rates for seven *Hamming* configurations with a second-degree polynomial approximation.

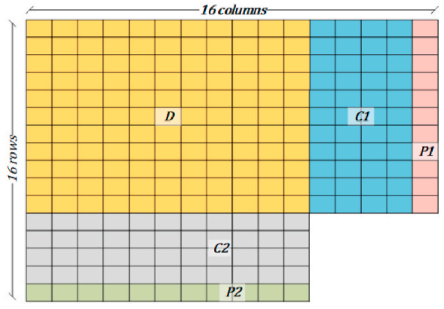

Fig. 11. Configuration of *OPCoSA* (231, 121) using *Ham*(16, 11). The 121 data bits are encoded in 231 bits; *rr* is 47.6%, with 110 bits of redundancy.

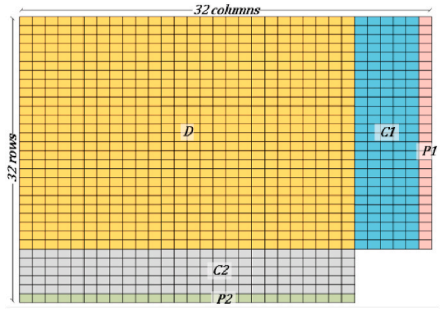

Fig. 12. *OPCoSA* (988, 676) employing *Ham*(32, 26) and encompassing 676 data bits encoded in 988 bits; *rr* is 33.5%, with 312 bits of redundancy.

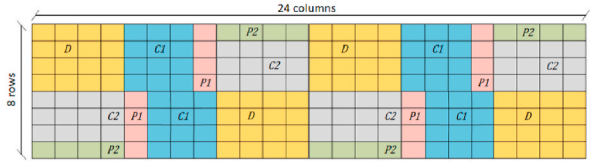

Fig. 13. *OPCoSA* configuration for use in 64-bit memories. OPCoSA (192, 64) has 64 data bits and 128 redundancy bits.

PDB in all range of errors, but these three ECCs show low correction capacity compared to the others from 1 to 3 errors. From four bitflips to six, *RM* presents a high error correction reduction, which is near to *PBD* that reaches only 13%, 5%, and 2%. In this same last error range, *PCoSA* and *OPCoSA* have rates much higher error correction capacity than the other ECCs; *OPCoSA* reaches 79%, 53%, and 35% of error correction for 4, 5, and 6 bitflips, respectively.

It is worth noting that the *OPCoSA* decoding algorithm was designed to correct 100% of the 36 error patterns. We define the δ metric to understand the percentage of these patterns within the set of all possible error scenarios. Additionally, δ_f displays the effectiveness of the decoding algorithm to correct other scenarios with the same f errors.

Let $p_f = (pS_f, pE_f)$ be a tuple containing the start and end numbers of the f error patterns exposed in Fig. 6, then $p_1 = (1,1), p_2 = (2,11), p_3 = (12,31)$ and $p_4 = (32,36)$. Let σ_p the be the number of all positions that pattern p can assume within the *OPCoSA* codeword, q_f be the sum of all σ_p with the same number of f errors, such that $q_f = \sum_{p=pS_f}^{pE_f} \sigma_p$. Let m_f be the total number of combinations with e errors within the *OPCoSA* codeword; i.e., an exhaustive analysis, such that $m_f = \binom{48}{f}$ (note that 48 is the number of bits of the *OPCoSA* codeword); thus, $\delta_f = \frac{q_f}{m_f} \times 100\%$ is the error pattern representativeness within all possibilities with e errors. For instance, $q_2 = \sum_{p=2}^{11} \sigma_p = 280$, and $m_2 = \binom{48}{2} = 1128$, thus, $\delta_2 = \frac{280}{1128} \times 100\% = 24.82\%$. Table 2 describes δ_f, q_f and, m_f for the range of 1–4 errors in Fig. 6.

On the one hand, the representativeness decrease with the increase in the number of errors is evident and understandable since exhaustive verification consider combinations of errors that leave the 9-cell envelope shown in Fig. 6; besides, this growth is proportional to the factorial of the number of errors f. On the other hand, even with low representativeness, the *OPCoSA* decoding algorithm achieves a high correction rate, reaching 79% of error correction for representativeness of only 0.06% in the case of $f = 4$. The explanation for this high correction efficacy comes from the *OPCoSA* matrix format, where every row and column has associated a Hamming code that can correct one error and detect two errors. This organization favors error patterns spaced in rows and columns, achieved in the exhaustive exploration, making these errors attended by different Hamming codes. For example, a 4-bit error pattern arranged on the same data row is perceived as four single errors in four data columns. The most complex error patterns that *OPCoSA* handles are the concentrated ones; therefore, an error pattern evaluation on a 3 × 3 matrix achieves such a high correction rate.

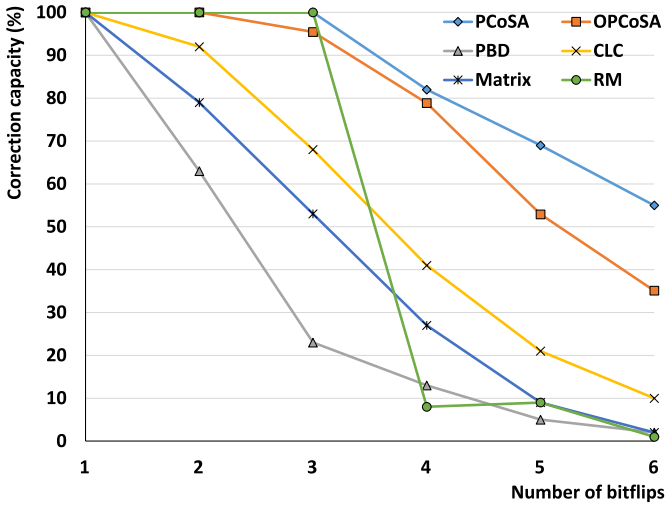

Fig. 14. Correction capacity of *PCoSA*, *OPCoSA*, *PBD*, *CLC*, *Matrix*, and *RM*. The simulation is done using all combinations from 1 to 6 bitflips.

B. Reliability

The reliability analysis is based on the work of [4,13]. Moreover, we assume the following assumptions: (i) the Poisson distribution allows representing the dispersion of transient errors over the memory lifetime [35]; (ii) errors occur statistically independently; (iii) the employment of the same range of errors for any code to evaluate the same time interval provides sound values for reliability comparison. While assumption (i) allows defining the equation that best represents the fault reliability behavior, assumptions (ii) and (iii) allow using the values presented in Fig. 14 as the error correction capacity of each code in the reliability computation.

Let $\alpha_{ECC,f}$ be the capacity of an ECC to correct f random errors, and $\beta_{ECC} = [\alpha_{ECC,1}, \alpha_{ECC,2}, \ldots, \alpha_{ECC,\Gamma}]$ be the vector that comprises the ECC correction rates from 1 to Γ errors, then $\beta_{OPCoSA} = [100\%, 100\%, 95\%, 79\%, 53\%, 35\%]$ and $\beta_{PCoSA} = [100\%, 100\%, 100\%, 82\%, 69\%, 55\%]$, with $\Gamma = 6$; the vectors β_{OPCoSA} and β_{PCoSA} were extracted from the same calculations that produced Fig. 14.

Let n be the number of codeword bits of a given ECC, f be and upset event in the code, t be a step time of a day, and λ probability of a single bit per t; then, the probability of having f errors in n bits during t days in a Poisson distribution $\Psi_f(t)$ is given by Equation (30).

$$\Psi_f(t) = \binom{n}{f} \left(1 - e^{-\lambda t}\right)^f e^{-\lambda(n-f)t} \tag{30}$$

Equation (31) computes $\Phi_{ECC}(t)$ - the ECC reliability parcel concerning its capacity to correct errors distributed according to Poisson distribution over t days.

$$\Phi_{ECC}(t) = \sum_{f=1}^{\Gamma} \alpha_{ECC,f} \times \Psi_f(t) \tag{31}$$

The probability of a n-bit codeword failure over t days is computed by Equation (32).

$$P(t) = 1 - e^{-\lambda n t} \tag{32}$$

Let M be the number of codewords in memory, such that events of error in codewords are independent, then Equation (33) computes the reliability of a memory protected by an ECC over t days $R_{ECC}(t)$ as the product of the reliability of all codewords.

$$R_{ECC}(t) = \left(1 - P(t) + \Phi_{ECC}(t)\right)^M \tag{33}$$

This work uses $M = 1$ (i.e., $R_{ECC}(t)$ is computed regarding a single codeword) to simplify the exhibition of the results. Extra information on the equations can be found in [13]. Fig. 15 shows $R_{PCoSA}(t)$ and $R_{OPCoSA}(t)$ encompassing three values of λ (1×10^{-4}, 5×10^{-5}, and 1×10^{-5}) and a range of 14,000 days. The horizontal axis is time expressed in days, while the vertical axis is the reliability of *OPCoSA* and *PCoSA* expressed in %. The reliabilities considering the other ECCs were not included in Fig. 15 because [3] already shown that *PCoSA* has a higher reliability than *PBD*, *CLC*, *Matrix*, and *RM* throughout the same range of days and considering the same values of λ.

Table 2
Representativeness of the 36 error patterns concerning the total error combinations.

f	q_f	m_f	δ_f	Correction
1	48	48	100.00%	100.0%
2	280	1128	24.82%	100.0%
3	464	17,296	2.68%	95.4%
4	122	194,580	0.06%	79.0%

The λ parameter indicates the error incidence rate in memory. For example, $\lambda = 10^{-4}$ indicates the probability of one error in a single bit every 10,000 days; since *OPCoSA* codeword has 48 bits, *OPCoSA* has the probability of one bitflip every 208 days. As errors occurrence in $R_{ECC}(t)$ are computed cumulatively, Fig. 15 illustrates that in 3000 days, the memory would have 14 bitflips, leading to reliability close to zero for both ECCs.

Fig. 15 displays that *OPCoSA* is more reliable than *PCoSA* for all periods and values of λ. For instance, with $\lambda = 10^{-5}$, *OPCoSA* reaches a rate of 100%, 96%, and 74% for days 1, 4000, and 8000, respectively, while for the same days, *PCoSA* reaches rates 100%, 92%, and 61%.

C. Redundancy and Synthesis Cost

ECCs were evaluated in terms of redundancy costs using two criteria: (i) code redundancy rate in, computed by Equation (34), and (ii) redundancy rate added in relation to the number of data bits, computed by Equation (35).

$$r1 = \frac{r}{n} \tag{34}$$

$$r2 = \frac{r}{k} \tag{35}$$

Table 3 contains the results of $r1$ and $r2$, which shows that the lowest redundancy rates are for the *Matrix* and *RM* codes; the highest rate is for *PCoSA*, while *OPCoSA* is 11.9% above the average for $r1$ and 23% above the average for $r2$.

Fig. 16 displays the sequence for obtaining the synthesis results. Initially, we described the encoder (encoder.v) and decoder (decoder.v) using Verilog in Register Transfer Level (RTL). To verify the encoder and decoder behavior, we implemented a TesbBench that includes a test file (test.v) and an error file (error.v). next, the waveforms of the circuits were validated using Xilinx's Integrated Development Environment (IDE) software known as Vivado Design Suite. Finally, we synthesized the Verilog codes to obtain the values of delay, area consumption, the power dissipation for encoder and decoder. The syntheses were performed using the RTL Compiler software with the 65 nm CORE65GPSVT standard cell library.

Fig. 17 displays the costs of the hardware synthesis of the evaluated ECCs, considering area consumption, power dissipation, and delay of

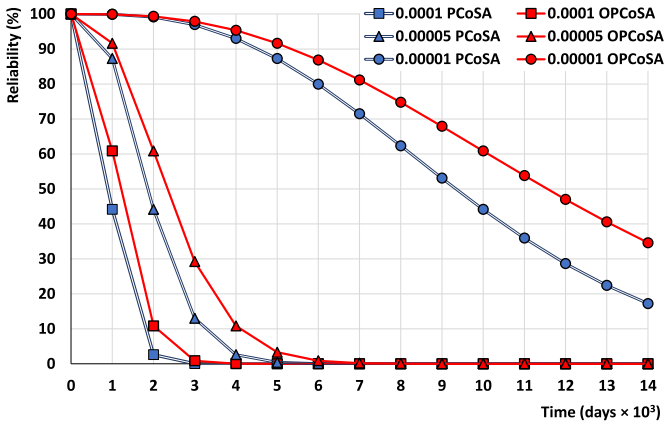

Fig. 15. Reliabilities provided by *PCoSA* and *OPCoSA*. The reliability regards three values of λ (probability of bit faults per day). The horizontal axis is the time in days, and the vertical axis is the reliability in %.

Table 3
Redundancy rate results.

ECC	$r1$(%)	$r2$(%)
PCoSA(64,16)	75.0	300
OPCoSA(48,16)	66.6	200
PBD(36,16)	55.5	125
CLC(40,16)	60.0	150
Matrix(32,16)	50.0	100
RM(32,16)	50.0	100

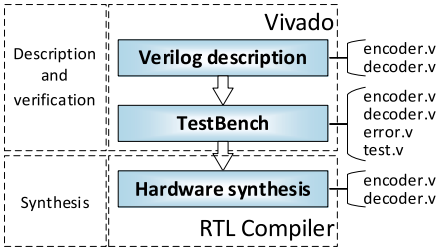

Fig. 16. Encoder and decoder description, verification and synthesis flow.

encoders and decoders.

The ECC decoder costs are much higher than the encoder ones since most calculations occur in the decoding process. The synthesis results for both encoder and decoder show that PDB, followed by Matrix, is the lowest cost ECC. On the one hand, considering only the encoder synthesis, OPCoSA appears in third place. On the other hand, considering only the decoder, CLC is the third most efficient ECC. Finally, except for the decoding delay, OPCoSA has lower synthesis costs than PCoSA, showing the efficiency of the proposed approach.

9. Conclusions

This paper presents *OPCoSA*, a product ECC requiring 32-redundancy bits to protect 16-data, which is based on *PCoSA* that requires more 16-redundancy bits. *OPCoSA* offers high correction capacity and a consequent decrease in hardware costs in relation to *OPCoSA*. The experimental results demonstrate that the correction rate up to four bitflips remains like *PCoSA* and above the other four ECCs (*CLC*, *PBD*, *Matrix*, and *RM*).

OPCoSA was evaluated through correction ability, reliability, redundancy, and hardware synthesis costs. *OPCoSA* reaches 100% of error correction for 36 specific error patterns and obtained 100% correction for burst errors of sizes one to four. The correction capacity difference between *OPCoSA* and *PCoSA* is a maximum of 4.5% for exhaustive error scenarios of up to four bitflips.

The great advantage of *OPCoSA* is that it offers the same functionality as *PCoSA*, but with 16 bits less redundancy, this directly contributes to the decreased area, power, and delay costs. As for reliability, three tests were performed varying the number of bit faults per day; in all cases, and for the entire period, *OPCoSA* has the highest reliability rates.

Authorship statement

All persons who meet authorship criteria are listed as authors, and all authors certify that they have participated sufficiently in the work to take public responsibility for the content, including participation in the concept, design, analysis, writing, or revision of the manuscript. Furthermore, each author certifies that this material or similar material has not been and will not be submitted to or published in any other publication.

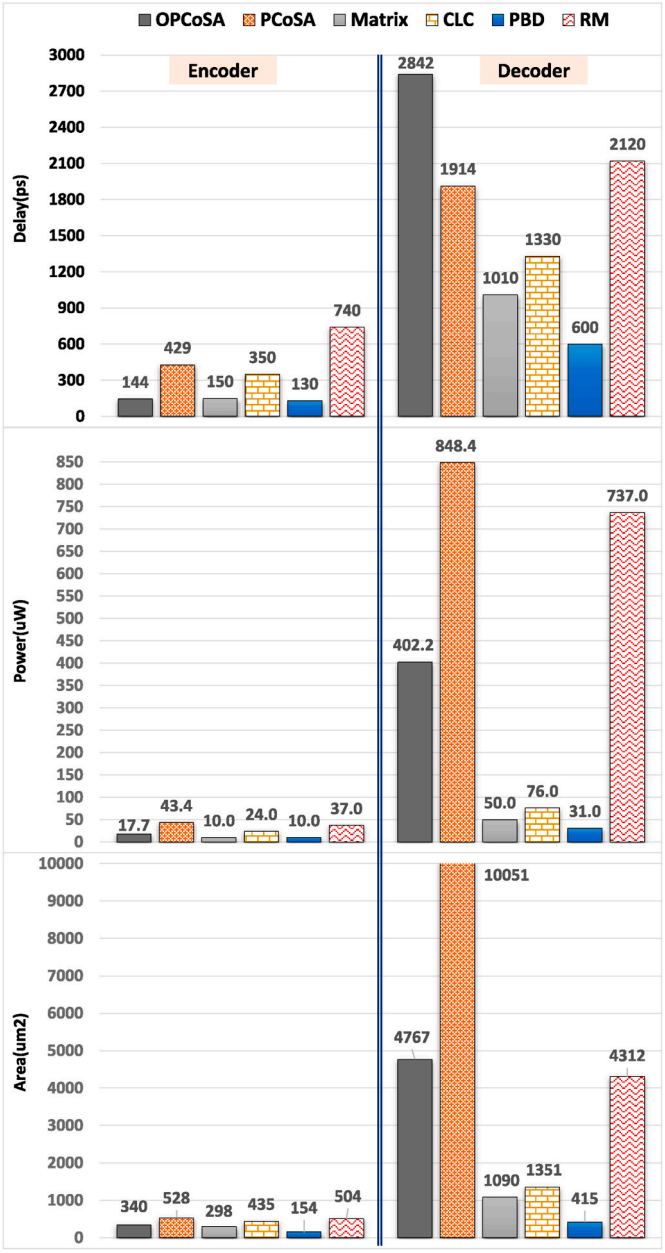

Fig. 17. Hardware cost of the encoder and decoder of the six ECCs, using Cadence's RTL Compiler synthesis tool for 65 nm CMOS technology.

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.

References

[1] J. Morán, L. Adalid, D. Tomás, P. Vicente, Improving error correction codes for multiple-cell upsets in space applications, IEEE Trans. Very Large Scale Integr. Syst. 26 (10) (Oct. 2018) 2132–2142.

[2] F. Silva, W. Freitas, J. Silveira, C. Marcon, F. Vargas, Extended matrix region selection code: an ECC for adjacent multiple cell upset in memory arrays, Microelectron. Reliab. 106 (1) (Mar. 2020) 1–9.

[3] D. Freitas, D. Mota, R. Goerl, C. Marcon, F. Vargas, J. Silveira, J. Mota, PCoSA: a product error correction code for use in memory devices targeting space applications, Integrat. VLSI J. 74 (1) (Sep. 2020) 71–80.

[4] F. Silva, J. Silveira, J. Silveira, C. Marcon, F. Vargas, O. Lima Jr., An extensible code for correcting multiple cell upset in memory arrays, J. Electron. Test. 34 (1) (Jul. 2018) 417–433.

[5] G. Kinoshita, C. Kleiner, E. Johnson, Radiation induced regeneration through the P-N junction isolation in monolithic I/C's, IEEE Trans. Nucl. Sci. 12 (5) (Oct. 1965) 83–90.

[6] C. Kleiner, G. Kinoshita, E. Johnson, Simulation and verification of transient nuclear radiation effects on semiconductor electronics, IEEE Trans. Nucl. Sci. 11 (5) (Nov. 1964) 82–104.

[7] C. Rosenberg, D. Gage, R. Caldwell, G. Hanson, Charge-control equivalent circuit for predicting transient radiation effects in transistors, IEEE Trans. Nucl. Sci. 10 (5) (Nov. 1963) 149–158.

[8] S. Liu, P. Reviriego, F. Lombardi, Codes for limited magnitude error correction in multilevel cell memories, IEEE Trans. Circ. Syst. I 67 (5) (May. 2020) 1615–1626.

[9] H. Farbeh, F. Mozafari, M. Zabihi, S.G. Miremadi, RAW-tag: replicating in altered cache ways for correcting multiple-bit errors in tag array, IEEE Trans. Dependable Secure Comput. 16 (4) (Jul. 2019) 651–664.

[10] P. Reviriego, S. Liu, O. Rottenstreich, F. Lombardi, Two bit overlap: a class of double error correction one step majority logic decodable codes, IEEE Trans. Comput. 68 (5) (May. 2019) 798–803.

[11] L. Adalid, J. Morán, D. Tomás, J. Calvo, P. Vicente, Ultrafast codes for multiple adajacent error correction and double error detection, IEEE Access 7 (1) (Oct. 2019) 151131–151143.

[12] J. Samanta, J. Bhaumik, S. Barman, Compact and power efficient SEC-DED coded for computer memory, Microsyst. Technol. 1 (1) (Feb. 2019) 1–10.

[13] C. Argyrides, H. Zarandi, D. Pradhan, Matrix codes: multiple bit upsets tolerant method for SRAM memories, in: Proceedings of the IEEE International Symposium on Defect and Fault-Tolerance in VLSI System (DFT), 2007, pp. 340–348.

[14] C. Argyrides, P. Reviriego, D. Pradhan, J. Maestro, Matrix-based codes for adjacent error correction, IEEE Trans. Nucl. Sci. 57 (4) (Aug. 2010) 2106–2111.

[15] C. Argyrides, D. Pradhan, T. Kocak, Matrix codes for reliable and cost efficient memory chips, IEEE Trans. Very Large Scale Integr. Syst. 19 (3) (Mar. 2011) 420–428.

[16] P. Elias, Error-free coding, Trans. IRE Prof. Group Inf. Theory 4 (4) (Sep. 1954) 29–37.

[17] H. Castro, J. Silveira, A. Coelho, F. Silva, P. Magalhães, O. Lima, A correction code for multiple cells upset in memory devices for space applications, in: Proceedings of the IEEE International New Circuits and Systems Conference (NEWCAS), 2016, pp. 1–4.

[18] F. Silva, A. Muniz, J. Silveira, C. Marcon, CLC-A: an adaptative implementation of the column line code (CLC) ECC, in: Proceedings of the Symposium on Integrated Circuits and Systems Design (SBCCI), Aug. 2020, pp. 1–6.

[19] S. Tambatkar, S. Menon, V. Sudarshan, M. Vinodhini, N. Murty, Error detection and correction in semiconductor memories using 3D parity check code with hamming code, in: Proceedings of the International Conference on Communication and Signal Processing (ICCSP), 2017, pp. 974–978.

[20] P. Raha, N. Murty, Horizontal-vertical parity and diagonal hamming based soft error detection and correction for memories, in: Proceedings of the International Conference on Computer, Communication and Informatics (ICCCI), 2017, pp. 1–5.

[21] G. Sai, K. Avinash, L. Naidu, M. Rohith, M. Vinodhini, Diagonal hamming based multi-bit error detection and correction technique for memories, in: Proceedings of the International Conference on Communication and Signal Processing (ICCSP), 2020, pp. 746–750.

[22] K. Neelima, C. Subhas, Efficient adjacent 3D parity error detection and correction codes for embedded memories, in: Proceedings of the International Conference on Electronics, Computing and Communication Technologies (CONECCT), 2020, pp. 1–5.

[23] F. Silva, W. Freitas, J. Silveira, O. Lima, F. Vargas, C. Marcon, An Efficient, low-cost ECC approach for critical-application memories, in: Proceedings of the Symposium on Integrated Circuits and Systems Design (SBCCI), 2017, pp. 198–203.

[24] R. Afrin, M. Sadi, An efficient approach to enhance memory reliability, in: Proceedings of the International Conference on Advances in Electrical Engineering (ICAEE), 2017, pp. 170–175.

[25] A. Erozan, E. Çavus, An EG-LDPC based 2-dimensional error correcting code for mitigating MBUs of SRAM memories, in: Proceedings of the FPGA World Conference, 2015, pp. 21–26.

[26] J. Morán, L. Adalid, J. Calvo, P. Gil, Correction of adjacent errors with low redundant matrix error correction codes, in: Proceedings of the Latin-American Symposium on Dependable Computing (LADC), 2018, pp. 107–114.

[27] J. Li, L. Xiao, J. Guo, X. Cao, Efficient implementation of multiple bit burst error correction for memories, in: Proceedings of the International Conference on Solid-State and Integrated Circuit Technology (ICSICT), 2018, pp. 1–3.

[28] M. Priya, M. Vijay, Error detection and correction for SRAM systems using improved redundant matrix code, in: Proceedings of the International Conference on Recent Advances in Energy-efficient Computing and Communication (ICRAECC), 2019, pp. 1–8.

[29] S. Liu, L. Xiao, J. Guo, Z. Mao, Fault secure encoder and decoder design for matrix codes, in: Proceedings of the International Conference on Computer-Aided Design and Computer Graphics (CAD/Graphics), 2015, pp. 181–185.

[30] J. Athira, B. Yamuna, FPGA implementation of an area efficient matrix code with encoder reuse method, in: Proceedings of the International Conference on Communication and Signal Processing (ICCSP), 2018, pp. 254–257.

[31] F. Macwilliams, N. Sloane, The Theory of Error-Correcting Codes, third ed., vol. 16, North-Holland, 1977, pp. 568–570.

[32] T. Moon, Error Correcting Code – Mathematical Methods, Algorithms, first ed., vol. 1, Wiley, 2005, pp. 430–432.

[33] R. Zaragoza, The Art of Error Correcting Coding, second ed., Wiley, West Sussex, England, 2006, pp. 170–201.

[34] P. Rao, M. Ebrahimi, R. Seyyedi, M. Tahoori, Protecting SRAM-based FPGAs against multiple bit upsets using erasure codes, in: Proceedings of the ACM/EDAC/IEEE Design Automation Conference (DAC), 2014, pp. 1–6.
[35] G. Zebrev, M. Gorbunov, R. Useinov, V. Emeliyanov, A. Ozerov, V. Anashin, A. Kozyukov, K. Zemtsov, Statistics and methodology of multiple cell upset characterization under heavy ion irradiation, Nucl. Instrum. Methods Phys. Res. 775 (Mar. 2015) 41–45.
[36] S. Lin, D.J. Costello, Error Control Coding: Fundamentals and Applications, first ed., vol. 1, Prentice-Hall, 1983, pp. 67–68.
[37] R. Goerl, P. Villa, L. Poehls, E. Bezerra, F. Vargas, An efficient EDAC approach for handling multiple bit upsets in memory arrays, Microelectron. Reliab. 88–90 (Sep. 2018) 214–218.