Integration, the VLSI Journal 74 (2020) 71–80

Contents lists available at ScienceDirect

Integration, the VLSI Journal

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

PCoSA: A product error correction code for use in memory devices targeting space applications

David Freitas [a,*], David Mota [a], Roger Goerl [b], César Marcon [b], Fabian Vargas [b], Jarbas Silveira [a], João Mota [c]

[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] Wireless Telecommunications Research Group (GTEL) - DETI, Federal University of Ceará, Fortaleza, Brazil

ARTICLE INFO

Keywords:
Error correction code
Fault tolerance
Radiation effect
Space radiation
Computer simulation

ABSTRACT

The radiation sensitivity of integrated memory cells increases dramatically as the supply voltage decreases. Although there are some Error Correcting Code (ECC) studies to prevent faults on memories used in space applications, there is no consensus on choosing the best ECC product-type with two-dimensional Hamming to mitigate data faults in memory. This work introduces the Product Code for Space Applications (PCoSA), an ECC product based on Hamming and parity in both rows and columns for use in memory with space-application reliability requirements. The potentialities of PCoSA were evaluated by injecting (i) thirty-six error patterns already available in the literature and (ii) all possible combinations of up to seven bitflips. PCoSA has corrected all cases of the thirty-six error patterns, and it has a correction rate of 100% for any three bitflips, 82.67% for four bitflips, and 69.7% for five bitflips.

1. Introduction

The Integrated Circuit (IC) manufacture is in the nanoscale era, implying a very large-scale integration. However, due to the decreased supply voltage and node capacitance, the amount of charge stored in a circuit node decreases, making the circuit more susceptible to various types of particles, such as protons, neutrons, heavy ions, alpha particles and high energy electrons, that generate faults during space applications [1–4]. Faults in ICs have been studied for over 40 years [5–8]. The most common appearance of these particles in space is the Single Event Upset (SEU) that cause unpredictable effects on running processes and electronic system outputs, such as data loss or error that may damage components, reduce performance, disrupt the data processing and even cause accidents [9–11].

There are several techniques for mitigating space application faults such as shielding, Process Technology, Hardened Memory Cell, Triple Modular Redundancy (TMR), and Error Correction Code (ECC). To minimize Process Technology failures, Silicon on Insulator (SoI) technology uses a thin layer of silicon on top of the insulator during the chip manufacturing process. In Hardened Memory Cells, different parts of original circuits are replaced by their hardened versions, which are less susceptible to faults but consuming more area and can imply more latency. The TMR technique uses three identical implementations of the same logic function, and the outputs are connected to a voter that decides mostly the correct result [12]. Lastly, ECCs are used to protect digital circuit data against errors that may occur in memory cells or transmission channels. The basic concept is to have an encoding and decoding algorithm to restore the correct value of the information [13].

The works of [14–25] show that the ECCs most widely used in space applications are Low-Density Parity Code (LDPC), Hamming, Reed-Muller (RM), Bose-Chaudhuri-Hocquenghem (BCH), Product code, Reed-Solomon and its variations. Erosan and Tavli [14] exploit LDPC to detect adjacent errors in SRAMs. Cui and Zhang [16] use an ECC(22, 16) based on Hamming and interleaving for 32-bit memories. Varghese et al. [18] investigate the possibility of using RM code for multi-bit errors in high-speed aerospace applications. The work of Su et al. [19] combines the BCH(15,7) code, which can correct two errors, with the TMR technique, achieving higher detection and correction rates when compared to the standard Hamming, TMR, and BCH. Silva et al. [20] compare CLC (a product-code with extended Hamming and parity bits) to RM(2, 5) and another product-code called Matrix. Goerl et al. [25] propose an ECC for memories, called Parity per Byte and Duplication (PBD), which is based

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

https://doi.org/10.1016/j.vlsi.2020.04.006
Received 22 October 2019; Received in revised form 19 March 2020; Accepted 23 April 2020
Available online 10 May 2020

on a configuration with the addition of parity bits and data duplication.

Although there are several studies of ECCs targeting memories for space applications, there is no consensus on the choice of codes used to compose an ECC product-type and whether using two-way Hamming product code along with parity is the best approach to mitigate the upsets in the data of a memory; as well as, there is no consensus in choosing the error patterns used for verifying the performance of each ECC. The investigations consider various combinations, such as Hamming, Extended Hamming, Parity, and BCH. Some researches use extended Hamming and parity, such as [20,21], but fail to correct all the patterns presented in this paper.

Regarding error patterns, the literature presents several studies. Castro et al. [21] generate pseudorandom words for each scenario, and the cells are positioned adjacent to each other. Radiation tests are also used to characterize the shapes of these patterns. Radaelli et al. [26] obtain error patterns using tests with different energy levels, i.e., 22 MeV, 47 MeV, 95 MeV, and 144 MeV. There are also researches like the one presented by Rao et al. [27] that consider error patterns and the probability of error occurrence.

Product codes are combinations of codes for creating error correction and detection methods. The term product code is used when the codeword forms a rectangle composed of linear blocks; these blocks are arranged so that the rows are encoded by one code, and the columns are encoded by another code [28,29]. Liu et al. [30] propose a scheme that combines extended line Hamming and column parity to avoid Multiple Bit Upset (MBU), keeping the correction capability with small area consumption and power dissipation. Moran et al. [31] show two types of product code based on Matrix to correct errors in adjacent memory cells. Additionally, Yang, Emre and Chakrabarti [32] propose a product code that codifies rows and columns using Reed-Solomon and Hamming, respectively; the proposed code reaches a high correction rate with significant hardware reduction.

The main contribution of this paper is to propose, implement and discuss a new configuration of Hamming-based product code that includes parity on both rows and columns. This code called Product Code for Space Applications (PCoSA) reaches high correction and error detection rates enabling it to be used in applications where reliability is a critical requirement, such as space applications.

2. PCoSA background

R. Hamming [33] has developed the Ham(n, k) linear block code for correcting a single error, where n is the total bits of the codeword, and k is the number of information bits. Equation (1) shows that r is the number of check bits that are added to the data to compose the codeword.

$$r = n - k \tag{1}$$

Fig. 1 shows that Hamming is a binary linear block code with $r \geq 2$, which is based on Equations (2) and (3).

Parity is an error detection method that adds an extra bit to the codeword. The parity bit can be either 0 or 1, depending on the number of 1s are in the codeword. Moreover, Extended Hamming is a code having a parity bit for increasing the Hamming code capacity to detect double errors besides to correct a single error [20]. The code proposed here uses even parity so that the total of 1s in the word, including the added parity bit, is even [34].

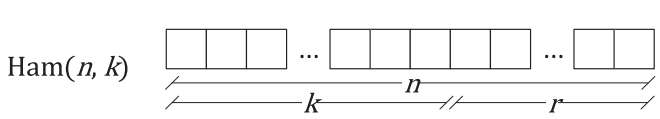

Fig. 1. Representation of a generic Hamming code Ham(n, k); n is the total number of bits, k is the amount of data bits, and r is the redundancy.

$$n = 2^r - 1 \tag{2}$$

$$k = 2^r - r - 1 \tag{3}$$

Fig. 2 shows that given two linear codes $C_1(n_1, k_1)$ and $C_2(n_2, k_2)$, then the product code is the combination of both codes (n_1n_2, k_1k_2), which is denoted by C_1C_2. The data is written in a matrix k_1k_2. Each one of the k_2 rows is coded using code C_1, forming n_1 columns. Each one of the n_1 columns is encoded using code C_2, forming a matrix n_1n_2. The linearity of the product code allows coding to begin with C_1 followed by C_2, or vice versa [28,29,35]. Also, if C_1 has minimum distance d_1 and C_2 has minimum distance d_2, then de product code C_1C_2 has minimum distance $d_1 \times d_2$. As the minimum distance of a code increases, the higher the code detection and correction capability [29].

Column-Line-Code (CLC) [20] is a product code that uses Extended Hamming in format Ham(8, 4) (i.e., $n = 8$, $k = 4$, and $r = 4$), as shown in Fig. 3. CLC is a code (40, 16), wherein a 16-bit word (represented by bits D_0-D_{15}) is encoded in a 40-bit codeword with 16 data bits, 12 check bits, 8 column-parity bits, and 4 row-parity bits. This code format makes CLC have a minimum distance $d = 8$ since Extended Hamming has $d = 4$ and parity has $d = 2$.

The column-parity bits Pc_0 to Pc_3 are used to detect errors in data bit columns, while the remaining bits Pc_4 to Pc_7 are used to detect errors in check bits and row parity bits. The combination of extended Hamming on each line with the parity bits allows correcting MCUs in the data word as well as the check and parity bits.

Equations (4)–(6) describe how to calculate C_q, performed by XOR ($\oplus$) operations, where q is the check bit index.

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

$$C_{q+1} = D_{4q/3} \oplus D_{\frac{4q}{3}+2} \oplus D_{\frac{4q}{3}+3} \; \forall q \in \{0, 3, 6, 9\} \tag{5}$$

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

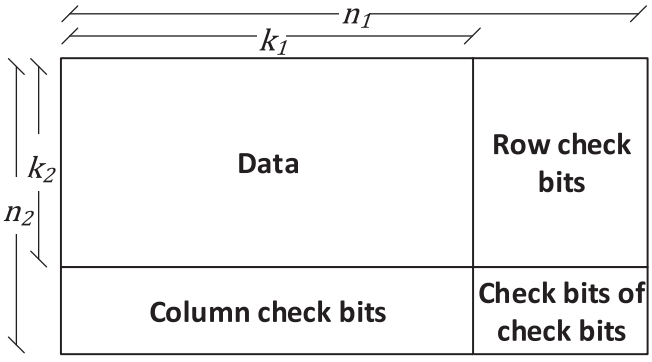

Fig. 2. Product code with k_1k_2 data bits and $n_1n_2 - k_1k_2$ check bits in rows and columns (adapted from [29]).

D_0	D_1	D_2	D_3	C_0	C_1	C_2	Pr_0
D_4	D_5	D_6	D_7	C_3	C_4	C_5	Pr_1
D_8	D_9	D_{10}	D_{11}	C_6	C_7	C_8	Pr_2
D_{12}	D_{13}	D_{14}	D_{15}	C_9	C_{10}	C_{11}	Pr_3
Pc_0	Pc_1	Pc_2	Pc_3	Pc_4	Pc_5	Pc_6	Pc_7

Fig. 3. CLC product code with data D, check bits C, parity of row Pr, and parity of column.Pc (adapted from [20]).

Equation (7) presents all *Pr* bits that are calculated to implement the extended Hamming code.

$$Pr_q = D_{4q} \oplus D_{4q+1} \oplus D_{4q+2} \oplus D_{4q+3} \oplus C_{3q} \oplus C_{3q+1} \oplus C_{3q+2}, \ \forall\ 0 \le q \le 3 \tag{7}$$

Equations (8)–(10) show how to calculate Pc_q, where q is the column index of *Pc*.

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

$$Pc_q = C_{q-4} \oplus C_{q-1} \oplus C_{q+2} \oplus C_{q+5} \forall\ \ 4 \le q \le 6 \tag{9}$$

$$Pc_7 = Pr_0 \oplus Pr_1 \oplus Pr_2 \oplus Pr_3 \tag{10}$$

CLC checks the integrity of the codeword by analyzing each row and column. The CLC algorithm starts generating the syndrome vectors of column-parity (*sPc*) and check bits (*sC*). These vectors require the Recalculated Check bit (*rC*) and the Recalculated Column-parity bits (*rPc*) that are computed using the same equations of *C* (Equations (4)–(6)) and *Pc* (Equations (8)–(10)), respectively. Equations (11) and (12) describe the computation of *sC* and *sPc*, respectively.

$$sC_q = C_q \oplus rC_q \ \ \forall\ \ 0 \le q \le 11 \tag{11}$$

$$sPc_q = Pc_q \oplus rPc_q \ \forall\ 0 \le q \le 7 \tag{12}$$

The CLC algorithm detects errors in the codeword when any of the *sC* and *sPc* bits are nonzero. Additionally, the *Pr* bits are also recalculated (*rPr*) to cover triple errors on the same line, allowing generating the line-parity syndrome (*sPr*), which is calculated according to Equation (13).

$$sPr_q = Pr_q \oplus rPr_q \ \forall\ \ 0 \le q \le 3 \tag{13}$$

Equation (14) calculates sCr_q, which is the reduction to one bit of all *sC* bits of each line, where + represents the logical OR operator.

$$sCr_q = sC_{3q} + sC_{3q+1} + sC_{3q+2} \ \forall\ 0 \le q \le 3 \tag{14}$$

Table 1 allows us to check whether an error has occurred and, if so, to indicate what type of error, as well as whether the number of errors is an even or odd, and thus apply the associated correction method; i.e., parity, Hamming or Hamming with parity.

According to the error pattern, CLC achieves higher correction and detection rates than Matrix and RM(2, 5); however, there are several error patterns produced by SEUs that are not covered by CLC.

3. PCoSA codeword definition

PCoSA employs the same extended Hamming-based structure of CLC but applying also Hamming to the columns. Therefore, PCoSA(64, 16), which is based on Extended Ham(8, 4), is the smallest possible product code format that implements PCoSA.

Fig. 4 illustrates PCoSA(64, 16) format, wherein a 16-bit word (represented by bits D_0-D_{15}) is encoded into 64 bits distributed as follows: (i) 16 data bits, (ii) 12 row-check bits *C*1, (ii) 7 row-parity bits *P*1, (iii) 21 column-check bits *C*2 and (iv) 8 column-parity bits *P*2. This code format

Table 1
Correction table based on the syndrome bits [20].

sCr	*sPr*	*sPc*	Status	Correction method
0	0	0	No error	–
0	0	1	Error detected	–
0	1	0	Error detected	–
0	1	1	Triple error corrected	Parity
1	0	0	Error detected	–
1	0	1	Even errors corrected	Parity
1	1	0	Odd errors corrected	Hamming
1	1	1	Odd errors corrected	Hamming and Parity

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. 4. PCoSA structure with 16 data bits. The code has five regions: data (*D*), check bits of the *D* rows (*C*1), check bits of the columns *D* and *C*1 (*C*2), parity of the rows *D* and *C*1, and *C*2 (*P*1), and parity of all columns (*P*2).

makes PCoSA have a minimum distance $d = 16$ since Extended Hamming has $d = 4$, increasing the detection and correction capability of PCoSA compared to CLC.

Equations 15 to 17 and 18 to 20 compute the recalculated check bits $rC1_q$ and $rC2_q$, respectively, where q is the bit-index. Additionally, Equations (2), (21) and (22)3-25 compute the recalculated parity bits *rP*1 and *rP*2, respectively.

$$rC1_q = D_{\frac{4q}{3}} \oplus D_{\frac{4q}{3}+1} \oplus D_{\frac{4q}{3}+3} \forall q \ \in \{0, 3, 6, 9\} \tag{15}$$

$$rC1_{q+1} = D_{4q/3} \oplus D_{\frac{4q}{3}+2} \oplus D_{\frac{4q}{3}+3} \forall q \ \in \ \{0, 3, 6, 9\} \tag{16}$$

$$rC1_{q+2} = D_{\frac{4q}{3}+1} \oplus D_{\frac{4q}{3}+2} \oplus D_{\frac{4q}{3}+3} \forall q \in \{0, 3, 6, 9\} \tag{17}$$

$$rC2_q = \begin{cases} D_q \oplus D_{q+4} \oplus D_{q+12} \\ C1_{q-4} \oplus C1_{q-1} \oplus C1_{q+5} \end{cases} \forall\ 0 \le q \le 3 \ \forall\ 4 \le q \le 6 \tag{18}$$

$$rC2_{q+7} = \begin{cases} D_q \oplus D_{q+8} \oplus D_{q+12} \\ C1_{q-4} \oplus C1_{q+2} \oplus C1_{q+5} \end{cases} \forall\ 0 \le q \le 3 \ \forall\ 4 \le q \le 6 \tag{19}$$

$$rC2_{q+14} = \begin{cases} D_{q+4} \oplus D_{q+8} \oplus D_{q+12} \\ C1_{q-1} \oplus C1_{q+2} \oplus C1_{q+5} \end{cases} \forall\ 0 \le q \le 3 \ \forall\ 4 \le q \le 6 \tag{20}$$

$$rP1_q = D_{4q} \oplus D_{4q+1} \oplus D_{4q+2} \oplus D_{4q+3} \oplus C1_{3q} \oplus C1_{3q+1} \oplus C1_{3q+2} \ \forall\ 0 \le q \le 3 \tag{21}$$

$$rP1_q = C2_{7q-28} \oplus C2_{7q-27} \oplus C2_{7q-26} \oplus C2_{7q-25} \oplus C2_{7q-24} \oplus C2_{7q-23} \oplus C2_{7q-22} \ \forall\ 4 \le q \le 6 \tag{22}$$

$$rP2_q = D_q \oplus D_{q+4} \oplus D_{q+8} \oplus D_{q+12} \oplus C2_q \oplus C2_{q+7} \oplus C2_{q+14} \ \forall\ 0 \le q \le 3 \tag{23}$$

$$rP2_q = C1_{q-4} \oplus C1_{q-1} \oplus C1_{q+2} \oplus C1_{q+5} \oplus C2_q \oplus C2_{q+7} \oplus C2_{q+14} \ \forall\ 4 \le q \le 6 \tag{24}$$

$$Pr2_7 = P1_0 \oplus P1_1 \oplus \ P1_2 \oplus \ P1_3 \oplus \ P1_4 \oplus \ P1_5 \oplus P1_6 \tag{25}$$

Applying Equations (26)–(29), the decoding algorithm computes **sind** = [*sC*1, *sP*1, *sC*2, *sP*2] - a vector composed of four syndromes; i.e., *sC*1 and *sC*2, which are the check bit syndromes of *C*1 and *C*2, respectively, and *sP*1 and *sP*2, which are the row and column parity syndromes,

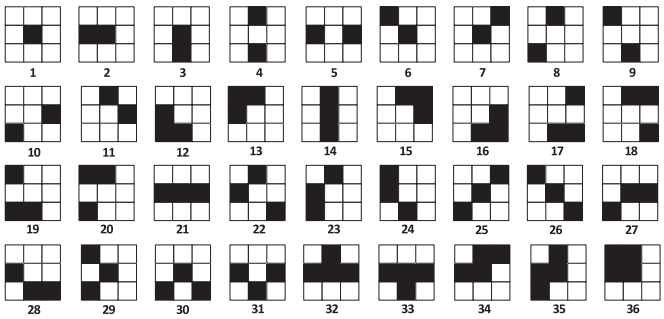

Fig. 5. Thirty-six error patterns used in the experiments, encompassing one simple error, ten double errors, twenty triple errors, and five quadruple errors (adapted from [27]).

respectively.

$$sC1 = \sum_{q=0}^{3} \left(C1_{3q} \oplus rC1_{3q}\right) + \left(C1_{3q+1} \oplus rC1_{3q+1}\right) + \left(C1_{3q+2} \oplus rC1_{3q+2}\right) \tag{26}$$

$$sC2 = \sum_{q=0}^{3} \left(C2_{q} \oplus rC2_{q}\right) + \left(C2_{q+7} \oplus rC2_{q+7}\right) + \left(C2_{q+14} \oplus rC2_{q+14}\right) \tag{27}$$

$$sP1 = \sum_{q=0}^{3} P1_{q} \oplus rP1_{q} \tag{28}$$

$$sP2 = \sum_{q=0}^{3} P2_{q} \oplus rP2_{q} \tag{29}$$

The PCoSA decoding algorithm explores the **sind** = [*sC*1, *sP*1, *sC*2, *sP*2] vector in the binary format $\mathbf{sind_b}$ = [*sc*1, *sp*1, *sc*2, *sp*2]. Equation (30) presents how to compute a binary element of $\mathbf{sind_b}$ from its counterpart in **sind**. For example, if **sind** = [0,2,2,3], then $\mathbf{sind_b}$ = [0,1,1,1].

$$sx = \begin{cases} 0, & if\ sX = 0 \\ 1, & else \end{cases} \tag{30}$$

4. PCoSA correction background

We designed PCoSA to reach high correction rates for error patterns with high incidence in spatial memories. This section presents these patterns, the format for injecting these patterns into a codeword for verification purposes and the PCoSA error correction method.

4.1. Basic error patterns

Rao et al. [27] proposed assessing ECCs using the 36 error patterns of most incidence in memories, attained with simulation results with a commercial tool for evaluating strikes of neutron particles. Fig. 5 shows these patterns that were employed to assess ECCs in other works like [20, 36].

We inserted these error patterns in the PCoSA codeword performing some adjustments. For example, the error pattern 2 has two adjacent bitflips on the same line; thus, it is inserted into the memory as follows: (i) the leftmost bit of this pattern is set as reference; (ii) the possible insertion positions are established (as shown in Fig. 6(a)); (iii) this pattern is placed in all valid positions of the matrix; (iv) each pattern is placed w times in the 8×8 matrix, with $w = 64$ only for the patterns with 1 bitflip, the other error patterns make $w < 64$.

Fig. 6 exemplifies areas where error patterns 2, 3, and 18 can be placed; these areas take into account the size and shape of the pattern. We must delimit the insertion region boundaries of each error pattern to ensure that the codeword obtains the exact pattern format. The region boundaries, which limits the number of patterns placed into the codeword, are highlighted by the red rectangles in Fig. 6; for instance, error patterns 2 and 3 have 56 insertion possibilities, while error pattern 18 has only 42 insertion possibilities.

The reference of all error patterns is the upper left bit of each pattern. Fig. 6(a) illustrates that the pattern 2 cannot be placed in the last column, as there are not two memory spaces available. Similarly, Fig. 6(b) displays that pattern 3 cannot be placed on the last line. Finally, Fig. 6(c) shows that the pattern 18 is limited to row six and column seven.

4.2. Correction method

We validate PCoSA with MatLab scripts that insert all error patterns into all regions of the memory matrix illustrated in Fig. 4. For example, an error pattern containing a simple error is placed in the five regions (*D*, *C*1, *P*1, *C*2 and *P*2); an error pattern with an adjacent double error on the same line, is placed in 7 possibilities: *D*, $D \cup C1$, *C*1, $C1 \cup P1$, *C*2, $C2 \cup P1$ and *P*2. The operator $\cup$ represents an area composed of more than one region; for instance, $D \cup C1$ indicates that a double error has occurred, and one bit of this error is in region *D* and the other one is in region *C*1, as shown in Fig. 7(a).

Simulating all the possibilities of placing the 36 error patterns, considering the cases of miscorrection (which occur in patterns that have triple errors in the same row) allows us to create a table that associates the error pattern, its positioning and the values of the **sind** vector. Fig. 7(b) exemplifies the error pattern 32 mapped in $3D \cup C1$, indicating three errors in region *D* and one error in region *C*1.

The PCoSA algorithm starts checking and correcting the column check bits associated with the data (i.e., $C2_0$ to $C2_3$, $C2_7$ to $C2_{10}$, and $C2_{14}$ to $C2_{17}$) using the check bits $C2_4$ to $C2_6$, $C2_{11}$ to $C2_{13}$ and $C2_{18}$ to $C2_{20}$, as well as the parity bits $P1_4$ to $P1_6$. This correction aims to achieve higher reliability for the column check bits associated with the data. Also, fields $P2_4$ to $P2_7$ allow checking the consistency of the check bits and parity columns, and in case of inconsistent values, the number of errors detected is increased. Then, the algorithm calculates the syndromes, generating $\mathbf{sind_b}$.

Table 2 shows 16 possibilities of syndrome $\mathbf{sind_b}$; only bold patterns marked with '*' need to go through the correction algorithm as errors outside region *D* can be recalculated from *D* information.

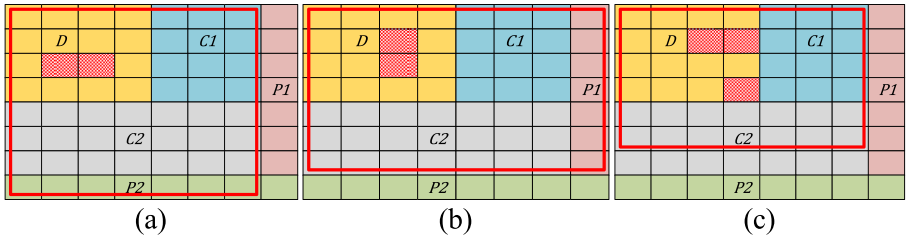

Fig. 6. (a), (b) and (c) show error patterns 2, 3 and 18, respectively. The red rectangles show regions where the pattern can be inserted into memory. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

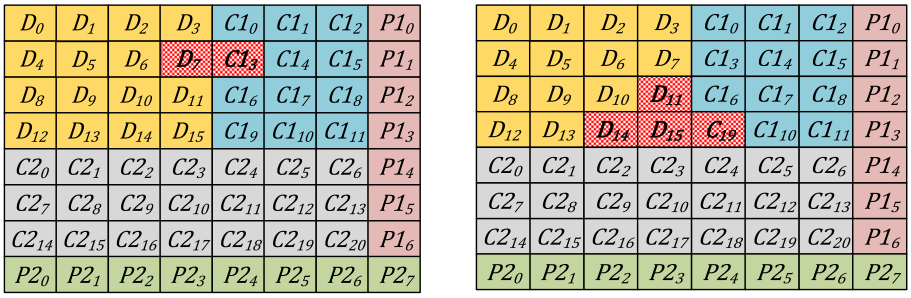

Fig. 7. Error patterns (a) 2 and (b) 32 of Fig. 5 placed in regions $D \cup C1$, and $3D \cup C1$, respectively. The bold bits with red-shaded background represent the error pattern. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

Table 2
Mapping of error patterns using $\mathbf{sind_b} = [sc1, sp1, sc2, sp2]$.

	$sind_b$	Error type	Number of error patterns
	0 0 0 0	No error	–
	0 0 0 1	Outside region *D*	4
	0 0 1 0		4
	0 0 1 1		20
	0 1 0 0		4
	0 1 0 1		8
	0 1 1 0		8
*	**0 1 1 1**	**Patterns 21, 33**	**Total 84 - only 4 in region *D***
	1 0 0 0	∅	–
	1 0 0 1		
*	**1 0 1 0**	**Pattern 36**	**Total 84 - only 4 in region *D***
*	**1 0 1 1**	**Patterns 2,5,34**	**Total 17 - only 9 in region *D***
	1 1 0 0	∅	–
*	**1 1 0 1**	**Pattern 14**	**Total 4 - only 2 in region *D***
*	**1 1 1 0**	**Patterns 3,4,35**	**Total 18 - only 9 in region *D***
*	**1 1 1 1**	**Several patterns**	**Total 213 - several in region *D***

∅ - means an unreachable syndrome.

The column **Number of error patterns** shows the number of patterns displayed in Fig. 5 for a given $\mathbf{sind_b}$. For example, $\mathbf{sind_b} = [0, 0, 0, 1]$ encompasses four error patterns occurred outside region D; these patterns are 1, 2, 5 and 21, and all patterns falling in the region P2. The following paragraphs detail the error patterns, the correction method applied and region that generates a given $\mathbf{sind_b}$ combinations, which are bolded and marked with '*' in Table 2.

$\mathbf{sind_b} = [0, 1, 1, 1]$.

Pattern	Region	sind	Correction method
21	*D* $2D \cup C1$	[0 1 3 3]	Triple error row is obtained with *SP*1, and bitflips are corrected using *SP*2
33	$3D \cup C2$ $2D \cup C1 \cup C2$	[0 2 3 2]	The bitflip out the triple error row is corrected by selecting the center column using *SC*2, and the second error row using the lower *SP*1. After recalculating the syndromes, the error row is known by *SP*1 and bitflips are changed using *SP*2

$\mathbf{sind_b} = [1, 0, 1, 0]$.

Pattern	Region	sind	Correction method
36	*D* $D \cup C1$ $D \cup C2$ $D \cup C1 \cup C2$	[2 0 2 0] [1 0 2 0]	The four bitflips are corrected by referencing the upper left bit of the pattern. This bit 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

$\mathbf{sind_b} = [1, 1, 0, 1]$.

Pattern	Region	sind	Correction method
14	*D* $2D \cup C2$	[3 3 0 1] [2 3 0 1]	It is a triple error in the column marked *SP*2; bitflips are corrected by rows using *SP*1

$\mathbf{sind_b} = [1, 0, 1, 1]$.

Pattern	Region	sind	Correction method
2	*D* $D \cup C1$	[1 0 2 2]	The row is obtained using *SC*1, and bitflips are corrected by knowing the column positions through *SP*2
5	*D* $D \cup C1$	[1 0 2 2]	
34	*D* $3D \cup C1$ $D \cup 3C1$ $2D \cup 2C2$ $D \cup C1 \cup 2C2$	[2 0 3 2] [1 0 3 2]	Correction algorithm applies Hamming to the leftmost and rightmost columns, causing a double error to remain in the column. Next, the syndromes are recalculated, and the two bitflips are corrected by obtaining the column by *SC*2 and the rows by *SP*1

$\mathbf{sind_b} = [1, 1, 1, 0]$.

Pattern	Region	sind	Correction method
3	*D* $D \cup C2$	[2 2 1 0] [1 2 1 0]	These patterns contain double errors pointed by *SC*2; bitflips are corrected by rows using *SP*1
4	*D* $D \cup C2$	[2 2 1 0] [1 2 1 0]	
35	*D* $2D \cup 2C1$ $3D \cup C2$ $D \cup 2C1 \cup C2$ $D \cup 3C2$	[3 2 2 0] [2 2 2 0] [1 2 2 0]	*SCI* points to the first row of the pattern, which is corrected with Hamming. Next, the syndrome is recalculated, and Hamming is applied to the second row (pointed by *SCI*). Finally, the algorithm recalculates the syndrome and corrects the remaining two bitflips knowing; rows are pointed by *SPI*

$\mathbf{sind_b} = [1, 1, 1, 1]$.

This syndrome occurs with error patterns in the format $m \times c$, where m and c are the numbers of rows and columns with errors, respectively. For example, Fig. 7(a) and (b) display a 1×2 and 2×3 error format, respectively. In cases of miscorrection, or in cases where the error is in regions such as $D \cup C2$, the error patterns may have different dimensions from those calculated. The PCoSA algorithm uses Equations (31)–(33) to calculate the error size T.

$$T = m \times c \tag{31}$$

$$m = \max(sC1, sP1) \tag{32}$$

Table 3
Correction method applied to each format of $\boldsymbol{sind_b} = [1,1,1,1]$.

T	Pattern	Region	Correction method
1 × 1	1	–	*SC*1 and *SC*2 point to bitflip row and column that is corrected by Hamming
1 × 2	12, 13, 15-20	$D \cup C2$	All these patterns have a size of 2 × 2 but are defined as 1 × 2 because a bitflip occurs outside region *D*. The correction algorithm checks the first row of the bitflip through *SC*1 or *SP*1, corrects the bitflip that is inside *D* and then applies Hamming to the first seven columns
	13, 15, 18, 20	$D \cup C1 \cup C2$	
1 × 3	21	D	Only error pattern 21 has dimension 1 × 3; the other error patterns are set here because they have bitflips outside of the region *D*. The correction algorithm checks for columns that have errors using *SC*2 and applies Hamming.
		$D \cup 2C1$	
	27	$2D \cup C2$	
		$D \cup C1 \cup C2$	
	28	$D \cup 2C2$	
	30	$D \cup 2C2$	
	31	$2D \cup C2$	
		$D \cup C1 \cup C2$	
2 × 1	–	$2D$	The 2 × 1-dimension error only occurs in the second verification set (which considers all combinations of up to 7 bitflips). These cases use Hamming in the first four rows
2 × 2	6-13, 15-20	D	The error correction algorithm first checks if any rows have double errors. If so, it corrects the bitflip whose column is pointed by *SP*2 and fixes by Hamming each row. If there is no double-column error, Hamming fixes each of the first four rows
		$D \cup C1$	
	6–11	$D \cup C2$	
2 × 3	32	D	The correction algorithm checks if there are any double errors in the column. If so, it corrects the error using the top row pointed by *SP*1. Subsequently, the algorithm checks the columns pointed by *SP*2 and corrects the bitflips applying Hamming. Otherwise, it checks the columns pointed by *SP*2 and corrects bitflips using Hamming
		$3D \cup C1$	
		$D \cup 3C1$	
		$D \cup 3C2$	
	33	D	
		$3D \cup C1$	
		$D \cup 3C1$	
		$D \cup 2C1 \cup C2$	
	27, 28, 30, 31	$2D \cup C1$	
		$D \cup 2C1$	
3 × 1	14	D	This pattern refers to a triple error in the column; so, the error correction algorithm applies Hamming on the first four rows
		$D \cup 2C2$	
3 × 2	22–24	D	Since these are simple line error patterns, the error correction algorithm applies Hamming to the first four rows
		$D \cup C1$	
		$2D \cup C2$	
		$D \cup 2C2$	
	22, 23	$D \cup C1 \cup C2$	
	29	D	
		$2D \cup C1$	
		$2D \cup C2$	
		$D \cup C1 \cup C2$	
		$D \cup 2C2$	
3 × 3	25, 26	D	They are just simple error patterns in the rows; the error correction algorithm applies Hamming on rows one to four
		$2D \cup C1$	
		$D \cup 2C1$	
		$2D \cup C2$	
		$D \cup C1 \cup C2$	
		$D \cup 2C2$	
–	–	–	If none of the above cases, the error correction algorithm applies Hamming to the first four lines

$$c = \max(sC2, sP2) \tag{33}$$

Table 3 describes all possibilities of T, together with the corresponding pattern, region and correction method.

5. Exploring scalability and redundancy rate

The Redundancy Rate *rr*, computed by Equation (34), is a metric that indicates the cost in bits of an ECC, which is given by the ratio between the redundancy and codeword bits. The higher the *rr*, the greater the weight of the redundancy bits of the codeword. However, the lower the *rr*, the lower the redundancy impact; consequently, the lower the ECC cost.

$$rr = \frac{r}{n} \times 100\% \tag{34}$$

PCoSA 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 maintains the redundancy rate, while code scaling reduces *rr*.

The PCoSA scaling is achieved, using other Hamming formats that preserve the same minimum Hamming distance of four, making the product-code to have the same minimum $d = 16$; consequently, preserving the same the correction and detection rates for all scaled versions of PCoSA. 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. 8 and Fig. 9, respectively.

On the one hand, the purpose of the PCoSA configurations presented in Figs. 8 and 9 is to decrease *rr*, contributing to a code with less redundancy and, consequently, lower cost in bits. For instance, PCoSA(256, 121) and PCoSA(1024, 676) have *rr* equal to 52.7% and 33.98%, respectively, which is a significant reduction of the cost in bits when compared to 75% of the basic PCoSA(64, 16). On the other hand, since PCoSA 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

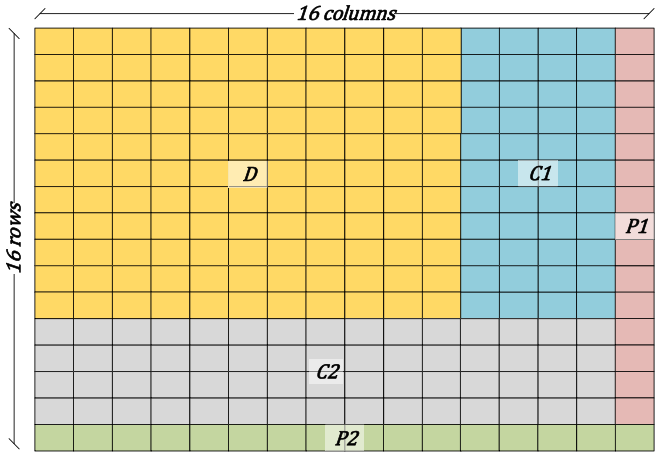

Fig. 8. Configuration of PCoSA(256, 121) using Ham(15, 11). The 121 data bits are encoded in 256 bits; *rr* is 52.7%, with 135 bits of redundancy.

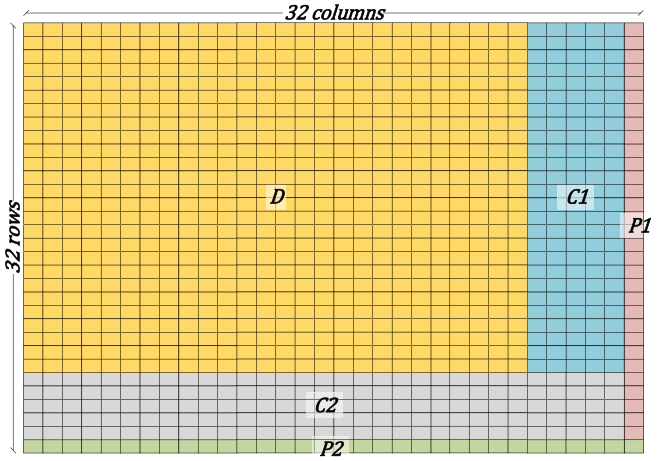

Fig. 9. PCoSA(1024, 676) employing Ham(31, 26) and encompassing 676 data bits encoded in 1024 bits; *rr* is 33.98%, with 348 bits of redundancy.

by PCoSA is reduced.

Replication is performed using any PCoSA code, such as the minimum PCoSA(64, 16), or other scaled versions as shown above: PCoSA(256, 121) and PCoSA(1024, 676). Fig. 10 illustrates PCoSA(256, 64) for protecting a memory with 64-bit words; this code format was achieved by replicating four PCoSA(64, 16).

On the one hand, this replicating process makes both PCoSA(64, 16) and PCoSA(256, 64) have *rr* equal to 75%, which is a high cost in bits. On the other hand, the replicating process scaling keeps the number of bits detected and corrected, and also the detection and correction rate of the memory protected by PCoSA.

6. Results and discussions

Fig. 11(a) and (b) shows the detection and correction rates, respectively, for the experimental result that compares PCoSA to Matrix, CLC, RM(2, 5) and PBD. Although PCoSA was designed to correct and detect the 36 error patterns illustrated in Fig. 5, we performed an exhaustive set of simulations containing all combinations from one to seven bitflips in an 8 × 8 memory to explore PCoSA's ability to correct more errors. Note that using all error patterns up to 7 bits reduces the error correction capacity of PCoSA but increases the fairness of the results.

Fig. 11(a) shows that all error cases using PCoSA were detected. This high detection rate happens because PCoSA has a matrix structure with two syndromes (the Hamming-check and parity bits) for each row and column. The PCoSA and PBD codes reach the highest detection rates up to 7 bitflips. PBD does not achieve 100% detection for only in the 4 bitflip cases (in this case, the code reaches 99.9%). This analysis is done in the work of [37], which explains that some cases of 4 and 8 bitflips are not detected. Matrix has the worst performance, detecting only 16% of cases with 7 bitflips.

Fig. 11(b) demonstrates that the PCoSA and RM codes have 100% correction rates for up to three bitflips. For cases with more bitflips, PCoSA reaches higher correction rates. For example, our ECC reaches 82.7% of correction rate in the experiments with four bitflips; whereas, the PBD code has the lowest correction rates up to three bitflips. Additionally, from four to seven bitflips, PBD, Matrix, and RM have the lowest correction rates, reaching 0.76%, 0.34%, and 0.16% for 7 bitflips, respectively.

7. Reliability and synthesis cost analysis

We performed a reliability analysis based on the works of Silva et al. [20] and Argyrides et al. [38], taking into account the same statements assumed by Ref. [38]: i) transient faults occur according to the Poisson distribution and ii) bit faults are statistically independent.

Le *Ne* be the maximum number of errors, and *FC* be the errors corrected, both occurring during time *t*, then, Equation (35) illustrates the fault correction computation in a codeword $F_c(t)$. Additionally, *MF* indicates if memory fails and *iF* indicates that there are *i* upsets in memory.

$$F_c(t) = \sum_{i=1}^{Ne} \left(P\{FC|iF\} \times P\{iF|MF\}\right) \tag{35}$$

Let *n* be the number of bits in the codeword and λ be the one-bit fault rate, then, Equations (36) and (37) compute the probability of having *i* upsets in memory $P\{iF\}$ and the probability of memory fail $P\{MF\}$, respectively. Finally, Equation (38) calculates the probability of having exact *i* upsets in a faulty memory $P\{iF|MF\}$.

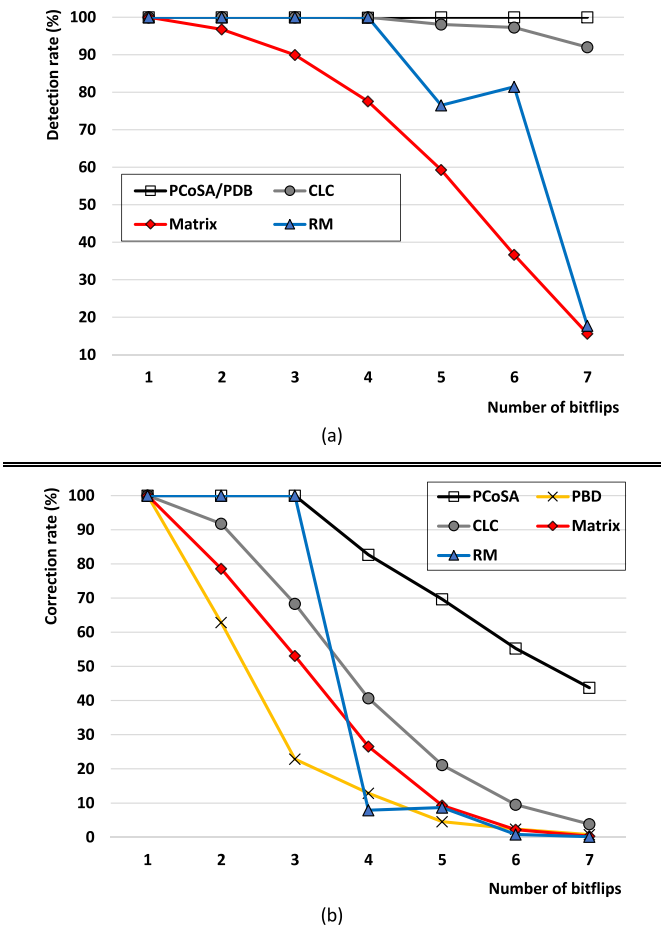

Fig. 11. (a) Detection and (b) correction rates of PCoSA, PBD, CLC, Matrix and RM codes. The simulation is done using all combinations from 1 to 7 bitflips.

$$P\{iF\} = \binom{n}{i}(1 - e^{-\lambda t})^i e^{-\lambda(n-i)t} \tag{36}$$

$$P\{MF\} = 1 - e^{-\lambda nt} \tag{37}$$

$$P\{iF|MF\} = \frac{P\{iF\}}{P\{MF\}} \tag{38}$$

The $P\{FC|iF\}$ values are obtained in the previous section through the simulation results presented in Fig. 11(b). Besides, let *M* be the number of codewords in memory, then, Equation (39) shows that the reliability of a memory $R(t)$ is the product of the reliability of all words in a given time *t*. Additional information on the equations can be found in Ref. [38].

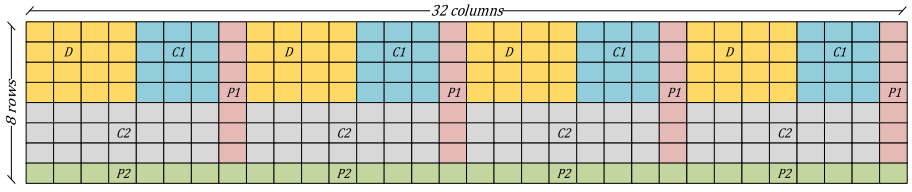

Fig. 10. PCoSA configuration for use in 64-bit memories. PCoSA(256, 64) has 64 data bits and 192 redundancy bits.

$$R(t) = \left(1 - P\{MF\} + \sum_{i=1}^{Ne} P\{iF\} \times P\{FC|iF\}\right)^{M} \quad (39)$$

Fig. 12 shows the reliability $R(t)$ of the five ECCs regarding correction capacity and a memory with $M = 1000$. The results demonstrate that PCoSA is the most reliable ECC throughout the period, having a much smoother reliability drop curve over time. The reliability of PCoSA is 99.99%, 94.63%, and 48.41% at times 100, 500, and 1000, respectively. On the opposite side is PBD, having an abrupt reliability drop, with $R(t)$ =48.13% at time 100 and tending to zero from time 300.

We synthesize the encoding and decoding modules of the five ECCs evaluated in this work to analyze their implementation costs. Fig. 13 illustrates the encoding and decoding schemes considering various types of memories (i.e., manufacturing technologies, sizes, formats, and protocols) with specific reading and writing drivers to clarify the synthesized modules. It is important to note that while the ECC encoder and decoder modules are only dependent on the ECC algorithms, the driver modules are memory configuration dependent.

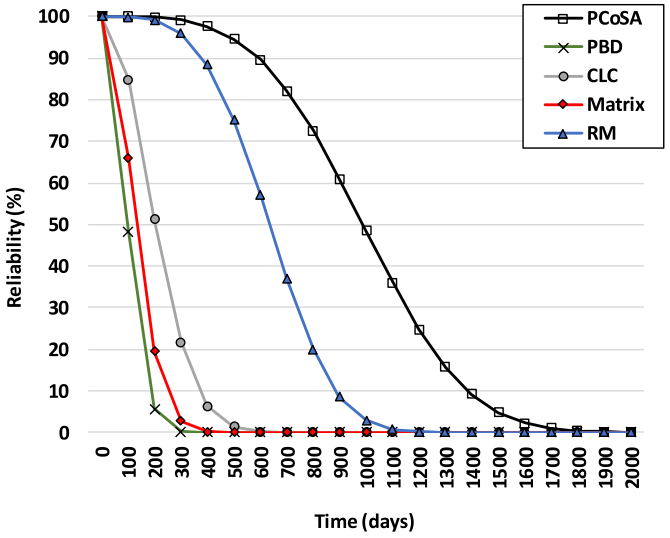

Fig. 12. Reliability of PCoSA, PBD, CLC, Matrix and RM.

Table 4
Area consumption, power dissipation, and delay for the encoder and decoder synthesis analysis of all evaluated ECCs.

	Area (μm^2)		Power (mW)		Delay (ns)	
	Encoder	Decoder	Encoder	Decoder	Encoder	Decoder
PCoSA	528	10051	0.043	0.848	0.43	1.91
Matrix	298	1090	0.010	0.050	0.15	1.01
CLC	435	1351	0.024	0.076	0.35	1.33
PBD	154	415	0.010	0.031	0.13	0.60
RM	504	4312	0.037	0.737	0.74	2.12

Table 4 displays the synthesis results for the encoder and decoder of PCoSA(64, 16) and all other evaluated ECCs, considering the encoding and decoding of a 16-bit data; these results encompass area consumption, power dissipation, and delay, which were achieved with the Cadence RTL Compiler software synthesis for 65 nm CMOS technology under normal operating conditions.

Independent of the evaluated ECC since most calculations are performed on the decoder side, it has higher values of area consumption, power dissipation, and delay when compared to the encoder. For example, the area consumption and power dissipation of the PCoSA decoder are about twenty times greater than the encoder, while the encoder delay is four and a half times less than the decoder.

Also, comparing Fig. 12 to Table 4 enables us to observe a tradeoff between reliability and synthesis costs. On the one hand, Fig. 12 clearly demonstrates that PCoSA has a great advantage in terms of reliability over other ECCs; on the other hand, Table 4 shows that this reliability implies high costs in area consumption and power dissipation, especially when compared to PBD that it is a very low-cost ECC. Finally, the comparison of PCoSA with RM (which is the second most reliable code) shows that PCoSA consumes slightly more than twice the area and dissipates only 15% more power.

8. Conclusions

This work proposes the Product Code for Space Applications (PCoSA)

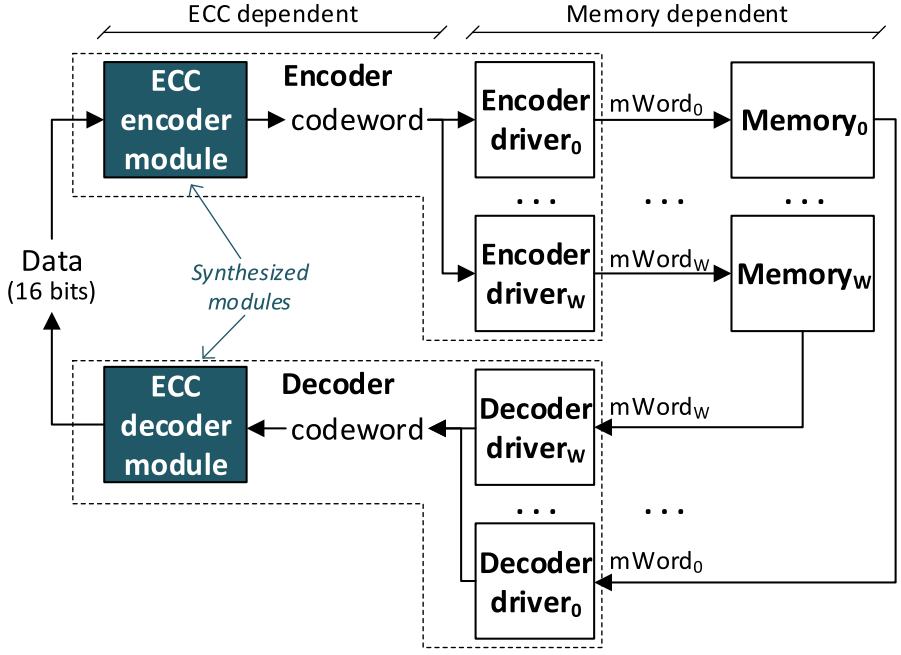

Fig. 13. Flow of encoding and decoding of the five ECCs evaluated, highlighting the modules (in green) that were synthesized. (For interpretation of the references to colour in this figure legend, the reader is referred to the Web version of this article.)

- a product-type ECC that uses Hamming and parity on both rows and columns. The error detection and correction capabilities enable using PCoSA in space application memories.

The validation of the proposed technique was performed using two sets of simulations. The first set considers 36 patterns, containing double, triple and quadruple errors, which were captured in memory simulation focused on spatial applications. The second set of simulations was exhaustively performed using all possible combinations of one to seven bitflips within an 8 × 8-bit memory.

The results were analyzed and discussed comparing with four other codes (Matrix, CLC, RM and PBD), equally designed for use in space application memories.

In all error cases, adjacent or not, PCoSA presented 100% of detection rate. This capability is due to (i) its matrix format and (ii) the existence of two syndromes for each row and column (Hamming check and parity bit). The other codes presented lower detection rates; the Matrix code achieves the worst performance, detecting only 16% of seven bitflips.

PCoSA and RM showed 100% of correction rate up to 3 bitflips. From 4 to 7 bitflips, PCoSA has 82.7%, 69.7%, 55.3% and 43.7% of correction rate, respectively. PBD has the worst correction rates up to 3 bitflips, and from 4 bitflips the lowest rates are for PBD, Matrix and RM; For 7 bitflips these codes have 0.76%, 0.34% and 0.16% of correction rate, respectively.

Our work shows that PCoSA can scale to more memory configurations, considering, for example, 16, 32 and up to 64 bits. When scaling the code, the redundancy rate remains at 75%. However, the number of redundancy bits can be reduced by using different Hamming configurations. For instance, with Ham(15, 11) and Ham(31, 26), the redundancy rates reduce to 52.5% and 33.98%, respectively.

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.

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.

Acknowledgement

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, and the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), both Brazilian research support agencies.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.vlsi.2020.04.006.

References

[1] A. Yan, Z. Wu, J. Guo, J. Song, X. Wen, Novel double-node-upset-tolerant memory cell designs through radiation-hardening-by-design, layout, IEEE Trans. Reliab. 68 (1) (Mar. 2019) 354–363.

[2] C. Qi, L. Xiao, T. Wang, J. Li, L. Li, A high reliable memory cell design combined with layout-level approach to tolerant single-event upsets, IEEE Trans. Device Mater. Reliab. 16 (3) (Sep. 2016) 388–395.

[3] E. Keren, S. Greenberg, N. Yitzhak, D. David, N. Refaeli, A. Haran, Characterization, mitigation of single-event transients in xilinx 45-nm SRAM-based FPGA, IEEE Trans. Nucl. Sci. 66 (6) (Jun. 2019) 946–954.

[4] T. Li, H. Liu, H. Yang, Design, characterization of SEU hardened circuits for SRAM-based FPGA, IEEE Trans. Very Large Scale Integr. Syst. 27 (6) (Jun. 2019) 1276–1283.

[5] G. Brucker, B. Parson, Radiation test, simulation of CMOS/SOS/Si-gate ALU, ROM Devices, IEEE Trans. Nucl. Sci. 23 (6) (Dec. 1976) 1720–1727.

[6] R. Kjar, B. Peterson, J. Blandford, "radiation hardened 64-bit CMOS/SOS RAM", IEEE Trans. Nucl. Sci. 23 (6) (Dec. 1976) 1728–1731.

[7] G. Brucker, Characteristics of CMOS/bulk, SOS memories in a transient environment, IEEE Trans. Nucl. Sci. 24 (6) (Dec. 1977) 2209–2212.

[8] T. Ellis, Radiation effects characterization of the SBP9900A 16-bit microprocessor, IEEE Trans. Nucl. Sci. 26 (6) (Dec. 1979) 4735–4739.

[9] Y. Yan, C. Hu, L. Zhang, W. Ning, B. Guo, Multilevel redundancy allocation for SEU mitigation of DSP codes based on OMA, IEEE Trans. Device Mater. Reliab. 19 (1) (Mar. 2019) 201–210.

[10] T. Kuwahara, Y. Tomioka, K. Fukuda, N. Sugimura, Y. Sakamoto, Radiation effect mitigation methods for electronic systems, in: Proceedings of the IEEE/SICE International Symposium on System Integration (SII), Dec. 2012.

[11] A. Macian, P. Reviriego, J. Maestro, Combined SEU, SEFI protection for memories using orthogonal Latin square codes, IEEE Transactions on Circuits, Systems I: Regular Papers 63 (11) (Nov. 2016) 1933–1943.

[12] M. Cannon, A. Keller, H. Rowberry, C. Thurlow, A. Celis, Strategies for removing common mode failures from TMR designs deployed on SRAM FPGAs, IEEE Trans. Nucl. Sci. 66 (1) (Jan. 2019) 207–215.

[13] F. Kastensmidt, in: Designing Single Event Upset Mitigation Techniques for Large SRAM-Based FPGA Components, Ph.D. dissertation, Dept. Comp., Federal University of Rio Grande do Sul, Brazil, 2003.

[14] A. Erosan, B. Tavli, High performance adjacent error detection for nanometer Devices, Electron. Lett. 52 (21) (Oct. 2016) 1788–1789.

[15] B. Li, Y. Pei, N. Ge, Area-Efficient fault-tolerant design for low-density parity-check decoders, Proceedings of the IEEE Vehicular Technology Conference (VTC-Fall) (2016) 1–5.

[16] Y. Cui, X. Zhang, Research and implementation of interleaving grouping hamming code algorithm, Proceedings of the IEEE International Conference on Signal Processing, Communication, Computing (ICSPCC) (2013) 1–4.

[17] Y. Cui, M. Lou, J. Xiao, X. Zhang, S. Shi, P. Lu, Research and implementation of SEC-DED hamming code algorithm, Proceedings of the IEEE International Conference of IEEE Region 10 (TENCON) (2013) 1–5.

[18] B. Varghese, S. Sreelal, P. Vinod, A.R. Krishnan, Multiple bit error correction for high data rate aerospace applications, Proceedings of the IEEE Conference on Information & Communication Technologies (CICT) (2013) 1–5.

[19] X. Su, J. Bao, B. Zhao, J. Su, Protecting router forwarding table in space, in: Proceedings of the International Conference on Mobile Ad-Hoc, Sensor Networks (MSN), 2011, pp. 446–450.

[20] 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) (Jun. 2018) 417–433.

[21] H. Castro, J. da Silveira, A. Coelho, F. Silva, P. Magalhães, O. Lima, A correction code of multiple cells upsets in memory Devices for space applications, Proceedings of the IEEE International New Circuit, Systems Conference (NEWCAS) (2016) 1–4.

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

[23] A. Appathurai, P. Deepa, Design for reliability: a novel counter code for fpga based quality applications, Proceedings of the Asia Symposium on Quality Electronic Design (ASQED) (2015) 56–61.

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

[25] R. Goerl, P. Villa, L. Poehls, E. Bezerra, F. Vargas, An efficient EDAC approach for handling multiple bit upsets in memory array", Microelectron. Reliab. 88–90 (Sep. 2018) 214–218.

[26] D. Radaelli, H. Puchner, S. Wong, S. Daniel, Investigation of multi-bit upsets in a 150nm technology SRAM device, IEEE Trans. Nucl. Sci. 52 (6) (Dec. 2005) 2433–2437.

[27] P. Rao, M. Ebrahimi, R. Seyyedi, M. Tahoori, Protecting SRAM-based FPGAs against multiple bit upset using erasure codes, Proceedings of the ACM/EDAC/IEEE Design Automation Conference (DAC) (2014) 1–6.

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

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

[30] S. Liu, L. Xiao, J. Li, Y. Zhou, Z. Mao, Low redundancy matrix-based codes for adjacent error correction with parity sharing, Proceedings of the International Symposium on Quality Electronic Design (ISQED) (2017) 76–80.

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

[32] C. Yang, Y. Emre, C. Chakrabarti, Product code schemes for error correction in MLC NAND flash memories, IEEE Trans. Very Large Scale Integr. Syst. 20 (12) (Dec. 2011) 2302–2314.
[33] R. Hamming, Error detecting, error correction codes, The Bell System Technical Journal 29 (2) (Apr. 1950) 147–160.
[34] R. Tocci, N. Widmer, G. Moss, Digital Systems – Principles, Applications, tenth ed., Pearson, 2007, pp. 41–46.
[35] R. Zaragoza, The Art of Error Correcting Coding, " second ed., Wiley, West Sussex, England, 2006, pp. 170–201. Wiley.
[36] A. Ahilan, P. Deepa, Radiation induced multiple bit upset prediction and correction in memories using cost efficient CMC 46, Dez, 2016, pp. 257–266. Journal of Microelectronics, Electronic Components and Materials 4.
[37] R. Goerl, An Efficient EDAC Approach for Handling Multiple Bit Upsets in Memory Arrays, Polytechnic School - Graduate Program in Electrical Engineering, PUCRS, Brazil, 2017. M.S. thesis.
[38] 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 Systems (DFT), 2007, pp. 340–348.