Gracia-Morán, J.; Saiz-Adalid, L.; Gil Tomás, DA.; Gil, P. (2018). Improving Error Correction Codes for Multiple-Cell Upsets in Space Applications. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 26(10):2132-2142. https://doi.org/10.1109/TVLSI.2018.2837220

# Improving Error Correction Codes for Multiple Cell Upsets in Space Applications

J. Gracia-Morán, L.J. Saiz-Adalid, D. Gil-Tomás, P.J. Gil-Vicente, *Member, IEEE*

***Abstract*—Currently, faults suffered by SRAM memory systems have increased due to the aggressive CMOS integration density. Thus, the probability of occurrence of Single Cell Upsets (SCUs) or Multiple Cell Upsets (MCUs) augments. One of the main causes of MCUs in space applications are cosmic radiation. A common solution is the use of Error Correction Codes (ECCs). Nevertheless, when using ECCs in space applications, they must achieve a good balance between error coverage and redundancy, and their encoding/decoding circuits must be efficient in terms of area, power and delay.**

**Different codes have been proposed to tolerate MCUs. For instance, Matrix codes use Hamming codes and parity checks in a bi-dimensional layout to correct and detect some patterns of MCUs. Recently presented, Column-Line-Code (CLC) has been designed to tolerate MCUs in space applications. CLC is a modified Matrix code, based on extended Hamming codes and parity checks. Nevertheless, a common property of these codes is the high redundancy introduced.**

**In this work, we present a series of new low-redundant ECCs able to correct MCUs with reduced area, power and delay overheads. Also, these new codes maintain, or even improve, memory error coverage with respect to Matrix and CLC codes.**

***Index Terms*—Error Correction Codes, Multiple Cell Upsets, Fault Tolerance, Reliability**

## I. Introduction

PRESENTLY, the continued physical feature size downscaling of CMOS technology provides memory systems with a great storage capacity. Nevertheless, this size decreasing has also caused an augment in the memory fault rate [1][2]. With the present aggressive scaling, the memory cell critical charge and the energy needed to provoke a Single Event Upset (SEU) in storage have been reduced [3]. As shown by different experiments, in addition to traditional Single Cell Upsets (SCUs), this energy reduction can provoke Multiple Cell Upsets (MCUs), that is, simultaneous errors in more than one memory cell induced by a single particle hit [4][5][6][7][8].

In the case of space applications, the MCU problem must be taken into account for the design of the corresponding fault tolerance methods, as space is an aggressive environment subjected to the impact of high energy cosmic particles [4][7][9].

Traditionally, Error Correction Codes (ECCs) have been used to protect memory systems. Common ECCs employed to protect standard memories are Single Error Correction (SEC) or Single Error Correction-Double Error Detection (SEC-DED) codes [11][12][13]. SEC codes are able to correct an error in one single memory cell. SEC-DED codes can correct an error in one single memory cell, as well as they can detect two errors in two independent cells.

In critical applications, like space applications, more complex and sophisticated codes are used [14][15][16][17][19][20][21]. For instance, Matrix code [17] is a well known code that combines Hamming codes with parity check in a matrix, allowing the correction of two bits in error. Recently presented, Column-Line-Code (CLC) [19] follows a similar approach, that is, it uses extended Hamming codes and parity bits to correct up to two adjacent bits in error.

The main problem when memory systems employ an ECC is the redundancy required. The extra bits added are used to detect and/or correct the possible errors occurred. Also, redundant bits must be added for each data word stored in memory. In this way, the amount of storage occupied for redundant bits scales with the memory capacity. For example, if an ECC with 100% of redundancy is employed in a 2GB memory, only 1GB is available to store the payload (the "clean" data); the remaining 1GB is required for code bits.

In addition, the usage of an ECC implies overheads in the area, power and delay employed by the encoder and decoder circuits. These overheads must be maintained as low as possible, especially in space applications.

In this work, we present a series of ECCs that greatly reduces the redundancy introduced, while maintaining, or even improving, memory error coverage. In addition, area, power and delay overheads are also reduced. These new codes have been designed using the Flexible Unequal Error Control (FUEC) methodology, developed by the authors in [31], where an algorithm (and a tool) to design FUEC codes is introduced. FUEC codes are an improvement of the well known Unequal Error Control (UEC) codes [11]. Nevertheless, the FUEC methodology can also find other kinds of codes. In this paper, it is employed to find low redundancy codes. These novel codes are different than those presented in [31]. They only share the design methodology, but with different parameters.

Manuscript received December 26, 2017. This work was supported in part by the Spanish Government under the research project TIN2016-81075-R.

All authors are with Instituto ITACA, Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain.

E-mail: { jgracia, ljsaiz, dgil, pgil }@itaca.upv.es

Mailing address: Escuela Técnica Superior de Ingeniería Informática, Edificio 1G, Despacho 2S7, Universitat Politècnica de València, Camino de Vera s/n, 46022 Valencia, Spain.

In this way, by using the tool, we can generate the parity check matrix of an ECC in an automatic and efficient way, just defining its error detection and/or correction capabilities.

This work is organized as follows. Section II introduces the design of ECCs. Section III summarizes the codes used in this work. Section IV describes the different results obtained during the evaluation of the ECCs. Finally, Section V concludes this paper.

## II. Introduction to the Design of Error Correction Codes

### A. Background on Error Correction Codes for Space Applications

Different ECCs have been traditionally applied to space missions [20]. For instance, Berger code [22] or the well known Parity code have been used for detection purposes.

On the other hand, when error correction is needed, more complex codes can be used, such as Hamming [12][20], Hadamard [23], Repetition [24], Golay [25], BCH [24], Reed-Solomon [24], Reed-Muller [21], multidimensional [26] or Matrix [17] codes.

Hamming codes [12] can be easily built for any word length. Also, the encoding and decoding circuits are easy to implement. Their main drawback is that only one bit in error can be corrected. Nevertheless, for common data word lengths (8, 16, 32, 64), Hamming codes can detect some double error patterns, in addition to the single error correction. Exploiting this feature, it is possible to systematize the detection of 2-bit adjacent errors with the same redundancy, as presented in [27][28]. In these works, different ECCs based on Hamming codes are introduced. These ECCs allows the correction of single bit errors, or the detection of 2-bit adjacent errors with the same redundancy.

The main problem of Hadamard and Repetition codes [23][24] is that they introduce a great redundancy for common data word lengths [20]. This great redundancy provokes the necessity of a great memory storage capacity, which is an inconvenient for space applications.

Golay code [25] is able to correct up to 3-bit errors. Nevertheless, Golay code presents a redundancy of almost 100%. Also, this code presents a high time and power consuming ratio, as it has to execute sequentially two complementary sequences.

Although BCH and Reed-Solomon codes [24][38] can correct multiple errors, their main drawbacks are the great complexity and difficulty to implement them, as well as their great latency and speed. These weaknesses can be very problematic in space applications.

Concerning Reed-Muller codes [21], although vastly used in critical applications, they present a great complexity. In this way, the overheads introduced are higher than the overheads introduced by Matrix or CLC codes, as shown in [17][19].

Multidimensional codes [26] are a class of matrix codes that uses parity bits to detect and correct errors. With a low redundancy, these codes present several drawbacks. When more than two errors must be corrected, the code design is very complicated. Also, it is very difficult to adapt these codes to standard data word sizes (i.e. 16, 32 or 64 bits).

A better alternative are the Matrix codes based on Hamming codes [17][19]. These codes still present a great redundancy, but they are more cost effective than previous multiple-error correcting codes.

In this work, we have designed several new ECCs using the FUEC methodology [31]. The main characteristic of these new codes is their low redundancy. In order to check the behavior of our codes, we have compared them with two types of codes. On the one hand, with matrix codes based on Hamming codes, as these last codes present a good relationship between redundancy and area, power and delay overheads. On the other hand, with Hamming-based codes due to the very low redundancy of these codes.

### B. Basics on Coding Theory

An ($n$, $k$) binary ECC encodes a $k$-bit input word in an $n$-bit output word [29]. The input word $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ is a $k$-bit vector which represents the original data. The codeword $\mathbf{b} = (b_0, b_1, \ldots, b_{n-1})$ is a vector of $n$ bits, where the $(n-k)$ redundant bits added are called parity or code bits. $\mathbf{b}$ is transmitted across an unreliable channel which delivers the received word $\mathbf{r} = (r_0, r_1, \ldots, r_{n-1})$. The error vector $\mathbf{e} = (e_0, e_1, \ldots, e_{n-1})$ models the error induced by the channel. If no error has occurred in the $i$th bit, $e_i$=0; otherwise, $e_i$=1. In this way, $\mathbf{r}$ can be interpreted as $\mathbf{r} = \mathbf{b} \oplus \mathbf{e}$. Fig. 1 synthesizes this encoding, channel crossing and decoding process.

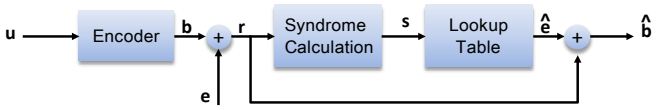


Fig. 1. Encoding, channel crossing and decoding process.

The parity check matrix $\mathbf{H}_{(n-k)\times n}$ of a linear block code defines the code [11]. For the encoding process, $\mathbf{b}$ must accomplish the requirement $\mathbf{H}\cdot\mathbf{b}^T=\mathbf{0}$. For syndrome decoding, the syndrome is defined as $\mathbf{s}^T=\mathbf{H}\cdot\mathbf{r}^T$, and it exclusively depends on $\mathbf{e}$:

$$\mathbf{s}^T = \mathbf{H}\cdot\mathbf{r}^T = \mathbf{H}\cdot(\mathbf{b}\oplus\mathbf{e})^T = \mathbf{H}\cdot\mathbf{b}^T \oplus \mathbf{H}\cdot\mathbf{e}^T = \mathbf{H}\cdot\mathbf{e}^T \quad (1)$$

There must be a different $\mathbf{s}$ for each correctable $\mathbf{e}$. If $\mathbf{s}=\mathbf{0}$, we can assume that $\mathbf{e}=\mathbf{0}$. Therefore, $\mathbf{r}$ is correct. Otherwise, an error has occurred. Syndrome decoding is performed by addressing a lookup table that relates each $\mathbf{s}$ with the decoded error vector $\hat{\mathbf{e}}$. The decoded codeword $\hat{\mathbf{b}}$ is calculated as $\hat{\mathbf{b}} = \mathbf{r} \oplus \hat{\mathbf{e}}$. From $\hat{\mathbf{b}}$, it is easy to obtain $\hat{\mathbf{u}}$ just discarding the parity bits. If the fault hypothesis employed to design the ECC is consistent with the channel behavior, $\hat{\mathbf{u}}$ and $\mathbf{u}$ must be equal with a very high probability.

### C. Error models

In coding theory [11], the term *random error* commonly refers to one or more bits in error, distributed randomly in the encoded word (data bits plus code bits generated by the ECC). *Random errors* can be *single* (only one bit affected) or *multiple*. *Single errors* are the simplest ones, as they only

affect a single memory cell. They are commonly produced by single event upsets (SEU, in random access memories) or single event transients (SET, in combinational logic) [32].

As it was commented in the Introduction section, with the continuous increasing of the integration scale, *multiple errors* are becoming more frequent [4][5][6][7][8]. *Multiple errors* mainly manifest as bursts [34]. We can define a *burst error* as a multiple error that spans *l* bits in a word [11], i.e. a group of contiguous bits where, at least, the first and the last bits are in error. The separation *l* is known as burst length. Notice that a*djacent errors* are a particular type of burst errors where all the erroneous bits are contiguous. The main physical causes of a burst error in the context of RAM memories are diverse: high energy cosmic particles that hit some neighbor cells, crosstalk between adjacent cells, etc. [1][33].

### D. Hamming codes

Hamming codes [12] are able to correct single bit errors with the lowest redundancy. For example, the parity check matrix for the Hamming (7, 4), i.e. $n = 7$ and $k = 4$, is shown in (2).

$$\mathbf{H} = \begin{bmatrix} 1010101 \\ 0110011 \\ 0001111 \end{bmatrix} \tag{2}$$

From (2), it is easy to deduce the encoding and decoding operations. The encoding formulas are shown in TABLE I.

TABLE I
ENCODING FORMULAS FOR THE HAMMING (7, 4) CODE

| $b_0$ | $b_1$ | $b_2$ | $b_3$ | $b_4$ | $b_5$ | $b_6$ | *Encoding formulas* |
|---|---|---|---|---|---|---|---|
| | | $u_0$ | | $u_1$ | $u_2$ | $u_3$ | |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | $b_0 = u_0 \oplus u_1 \oplus u_3$ |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | $b_1 = u_0 \oplus u_2 \oplus u_3$ |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | $b_3 = u_1 \oplus u_2 \oplus u_3$ |

In the same way, it is also possible to obtain the syndrome decoding formulas from the parity check matrix (2). TABLE II shows the expressions obtained to calculate the syndrome bits for the Hamming (7, 4) code.

TABLE II
SYNDROME BITS FOR THE HAMMING (7, 4) CODE

| $r_0$ | $r_1$ | $r_2$ | $r_3$ | $r_4$ | $r_5$ | $r_6$ | *Syndrome bits* |
|---|---|---|---|---|---|---|---|
| | | $u_0$ | | $u_1$ | $u_2$ | $u_3$ | |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | $s_0 = r_0 \oplus r_2 \oplus r_4 \oplus r_6$ |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | $s_1 = r_1 \oplus r_2 \oplus r_5 \oplus r_6$ |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | $s_2 = r_3 \oplus r_4 \oplus r_5 \oplus r_6$ |

If an error occurs, the syndrome bits will locate the erroneous bit. A look-up table (implemented, for example, using a binary decoder) selects the erroneous bit. Applying the "exclusive-or" operation, the output of the look-up table correct the erroneous bit.

For common word lengths, such as 8, 16, 32 and 64 bits, there exist (12, 8), (21, 16), (38, 32) and (71, 64) Hamming codes respectively. As it can be seen, redundancy decreases with longer data words. For instance, the (12, 8) Hamming code presents a 50% of redundancy, whereas the (71, 64) Hamming code introduces about 11% of redundancy.

Hamming codes can be extended to correct single errors and detect double random errors. These codes are known as SEC-DED (Single Error Correction-Double Error Detection) Extended Hamming codes [12]. These codes need an additional parity bit to achieve the double error detection. It is calculated as the even parity for the whole encoded word. In this way, and just adding an extra bit $b_7$, the Hamming SEC code (7, 4) shown previously converts into an extended Hamming SEC-DED code (8, 4). $b_7$ can be obtained as follows:

$$b_7 = b_0 \oplus b_1 \oplus b_2 \oplus b_3 \oplus b_4 \oplus b_5 \oplus b_6 \tag{3}$$

The decoding process checks two conditions: i) the parity of the whole received word; and ii) the syndrome bits, which are calculated as in the Hamming code. TABLE III shows the possible results, the corresponding meaning and the actions to be taken. As it can be seen, single-bit errors can be corrected as in the Hamming code. In the case of a double-bit error, a non-recoverable error (NRE) is detected but it cannot be corrected.

TABLE III
EXTENDED HAMMING ERROR DETECTION/CORRECTION

| SYNDROME BITS | PARITY (WHOLE WORD) | EVENT | ACTION |
|---|---|---|---|
| Zero | Even | NO ERROR | - |
| Zero | Odd | SINGLE ERROR (in the parity bit) | - |
| Nonzero | Even | TWO-BIT ERROR (non-recoverable error) | Disable correction logic<br>Signal the error |
| Nonzero | Odd | SINGLE ERROR (in bits 0 to 6) | Enable correction logic |

There exist also (13, 8), (22, 16), (39, 32) and (72, 64) SEC-DED extended Hamming codes. As in the SEC codes, redundancy decreases with higher data word lengths.

### E. Flexible Unequal Error Control methodology

The methodology employed to design the error control codes introduced in this work can be found in [31]. This methodology was developed to obtain Flexible Unequal Error Control (FUEC) codes. However, it can be generalized to find any kind of codes. Although a detailed explanation is out of the scope of this paper, it is briefly summarized in the following. This methodology is based on formulating the problem as a Boolean Satisfiability problem. An algorithm developed by the authors is employed to solve it and to obtain a parity check matrix, which defines the code to be designed.

After defining the values of *n* and *k* for the code, the first step is the selection of error patterns to be corrected and detected. For instance, single errors are represented with error vectors (…1…), and error vectors for double random errors show the pattern (…1…1…), where 1's represent the bits in error, and the dots represent the correct bits.

The next step is to find the parity check matrix **H** that satisfies the conditions (4) and (5), where $\boldsymbol{E}_+$ represents the set of error vectors to be corrected, and $\boldsymbol{E}_\Delta$ is the set of error

vectors to be detected. That is, each correctable error must generate a different syndrome (4). In addition, each detectable error must generate a syndrome which is different to all the syndromes generated by the correctable errors (5). However, several detectable errors may have the same syndrome.

$$\mathbf{H}\cdot\mathbf{e}_i^T \neq \mathbf{H}\cdot\mathbf{e}_j^T; \forall \mathbf{e}_i, \mathbf{e}_j \in \boldsymbol{E}_+ \mid \mathbf{e}_i \neq \mathbf{e}_j \quad (4)$$

$$\mathbf{H}\cdot\mathbf{e}_i^T \neq \mathbf{H}\cdot\mathbf{e}_j^T; \forall \mathbf{e}_i \in \boldsymbol{E}_\Delta, \mathbf{e}_j \in \boldsymbol{E}_+ \quad (5)$$

To find the matrix, a recursive backtracking algorithm is used. It checks partial matrices and adds a new column only if the previous matrix satisfies the requirements. In this way, the algorithm starts with an empty **partial_H** matrix. New columns, with *n–k* rows, are added, and the new partial matrices are checked recursively. The added columns must be non-zero, so there are $2^{n-k} - 1$ combinations for each column.

The complete execution of the algorithm is commonly unfeasible. Nevertheless, the first solutions are usually found quickly, if the code exists. Once selected the **H** matrix, it is easy to determine the logic equations to calculate each parity and syndrome bit, as well as the syndrome lookup table. They are required for the encoder and decoder implementation.

In addition, we can apply two different optimization criteria. If we want to decrease the delay of the encoders and decoders, we have to reduce the number of 1's in those rows with the highest number of 1's of the parity check matrix. In the case of area reduction, the total number of 1's in the parity check matrix must be reduced.

A detailed explanation of this algorithm, as well as a code design example, can be found in [31].

## III. Error Correction Codes Description

### A. Previous proposals

As commented previously, Matrix and CLC codes have been designed to tolerate MCUs [17][19], a critical concern in space applications.

Combining Hamming codes and parity checks [17][18], Matrix codes form a two dimensional scheme for correcting and detecting some patterns of MCUs. For instance, in this paper, we have used the bit layout shown in Fig. 2 (extracted from [18]), where $X_i$ are the data bits, $C_i$ are the horizontal check bits (calculated as a Hamming code), and $P_i$ are the column parity bits (even parity).

| | | | | | | |
|---|---|---|---|---|---|---|
| $X_1$ | $X_2$ | $X_3$ | $X_4$ | $C_1$ | $C_2$ | $C_3$ |
| $X_5$ | $X_6$ | $X_7$ | $X_8$ | $C_4$ | $C_5$ | $C_6$ |
| $X_9$ | $X_{10}$ | $X_{11}$ | $X_{12}$ | $C_7$ | $C_8$ | $C_9$ |
| $X_{13}$ | $X_{14}$ | $X_{15}$ | $X_{16}$ | $C_{10}$ | $C_{11}$ | $C_{12}$ |
| $P_1$ | $P_2$ | $P_3$ | $P_4$ | | | |

Fig. 2. Layout of a 16 data-bit word for the (32, 16) Matrix code [18].

The basic behavior of this Matrix code is as follows. The primary data input ($X_i$) is divided into groups of several bits. In this work, this division is in groups of 4 bits. Each group is codified by a (7, 4) Hamming code ($C_i$). Lastly, a set of vertical parity bits ($P_i$) completes the matrix. The Matrix code implemented in this work presents better correction and detection performance than an extended Hamming code, as it is able to correct all single errors and to correct or to detect all 2-bit burst errors. Nevertheless, this Matrix code presents a higher redundancy than an extended Hamming code. In this way, memory required for code bits is increased, and also, area, energy and delay overheads.

Recently presented, CLC code [19][30] is another matrix code proposed to be used in space applications. The layout we have employed for the implementation of this code is shown in Fig. 3 (extracted from [19]), where $X_i$ is the primary data input, divided into groups of 4 bits. Each group is codified by a SEC-DED (8, 4) Extended Hamming code ($C_i$ and $Pa_i$). Finally, a set of vertical parity bits ($P_i$) form the matrix.

As just commented, and unlike the Matrix code, CLC uses an Extended Hamming code, allowing the correction of all single and 2-bit burst errors. Nevertheless, CLC introduces a higher number of redundant bits, provoking a greater area, power and delay overheads, and reducing the available memory for the payload.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $X_1$ | $X_2$ | $X_3$ | $X_4$ | $C_1$ | $C_2$ | $C_3$ | $Pa_1$ |
| $X_5$ | $X_6$ | $X_7$ | $X_8$ | $C_4$ | $C_5$ | $C_6$ | $Pa_2$ |
| $X_9$ | $X_{10}$ | $X_{11}$ | $X_{12}$ | $C_7$ | $C_8$ | $C_9$ | $Pa_3$ |
| $X_{13}$ | $X_{14}$ | $X_{15}$ | $X_{16}$ | $C_{10}$ | $C_{11}$ | $C_{12}$ | $Pa_4$ |
| $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_5$ | $P_6$ | $P_7$ | $P_8$ |

Fig. 3. Layout of a 16 data-bit word for the (40, 16) CLC [19].

On the other hand, [27] introduces a SEC-DAED (Single Error Correction-Double Adjacent Error Detection) code with a very low redundancy. This code is able to correct all single errors, or to detect all double adjacent errors in a 16-bit data word with only 5 redundant bits. Fig. 4 shows the data word layout of this code, where $C_i$ are the code bits, and $X_i$ are the data bits.

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ | $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ | $X_7$ | $X_8$ | $X_9$ | $X_{10}$ | $X_{11}$ | $X_{12}$ | $X_{13}$ | $X_{14}$ | $X_{15}$ | $X_{16}$ |

Fig. 4. Layout of a 16 data-bit word for the (21, 16) SEC-DAED [27].

On the contrary, [28] presents a different approach to generate Hamming based ECCs. In this case, we have selected a SEC-DAED code with the same coverage characteristics, as well as the same redundancy of the SEC-DAED code from [27]. The main difference is the code layout, shown in Fig. 5.

To obtain the value of the different $C_i$ bits, the parity check matrix of these two SEC-DAED codes can be seen in [27][28] respectively.

| | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $X_1$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ | $X_7$ | $X_8$ | $C_1$ | $X_9$ | $C_2$ | $C_3$ | $C_4$ | $X_{10}$ | $C_5$ | $X_{11}$ | $X_{12}$ | $X_{13}$ | $X_{14}$ | $X_{15}$ | $X_{16}$ |

Fig. 5. Layout of a 16 data-bit word for the (21, 16) SEC-DAED [28].

### B. Our approach

By using the FUEC methodology [31], we have been able to design several codes that improve the coverage and/or the redundancy of the different codes presented previously (Matrix, CLC and both SEC-DAED codes). The layout of our codes is presented in Fig. 6, where $C_i$ are the code bits, and $X_i$ are the data bits.

| $C_0$ | .. | .. | $C_{n-1}$ | $X_0$ | $X_2$ | $X_3$ | $X_4$ | $X_5$ | $X_6$ | $X_7$ | $X_8$ | $X_9$ | $X_{10}$ | $X_{11}$ | $X_{12}$ | $X_{13}$ | $X_{14}$ | $X_{15}$ | $X_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

Fig. 6. Layout of a 16 data-bit word for the FUEC codes.

Using our algorithm, we have found a code (we will call it FUEC-DAEC) that can correct an error in a single bit, or an error in 2 adjacent bits, or it can detect one 3-bit burst error or one 4-bit burst error.

FUEC-DAEC needs only seven code bits for a 16-bit data word. Fig. 7 shows the parity check matrix **H** for this code, where $C_i$ are the code bits and $X_i$ are the primary data bits.

$$\mathbf{H} = \begin{array}{c} C_0 \cdots\cdots C_6\ X_0\ X_1 \cdots\cdots X_{15} \\ \left|\begin{array}{cccccccccccccccccccc} 1&0&0&0&0&0&0&1&0&0&0&1&0&0&1&1&0&0&1&1&1&0&0 \\ 0&1&0&0&0&0&0&0&1&0&1&0&1&0&1&0&1&1&1&0&0&1&0 \\ 0&0&1&0&0&0&0&1&0&1&0&0&0&1&1&0&1&0&0&1&0&1&1 \\ 0&0&0&1&0&0&0&0&1&0&0&1&0&0&0&1&1&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0&1&0&0&1&1&0&0&1&0&0&0&0&1&1&0&1 \\ 0&0&0&0&0&1&0&0&1&1&0&0&1&0&0&0&1&1&0&1&0&1&0 \\ 0&0&0&0&0&0&1&0&0&1&1&0&0&1&0&1&0&1&1&0&1&0&1 \end{array}\right| \end{array}$$

Fig. 7. Parity check matrix **H** for the (23, 16) FUEC-DAEC code.

Once **H** has been obtained, it is very easy to design the encoder/decoder circuitry. For example, the formulas to calculate the code bits for the FUEC-DAEC code are:

$$\begin{aligned} C_0 &= X_0 \oplus X_4 \oplus X_7 \oplus X_8 \oplus X_{11} \oplus X_{12} \oplus X_{13} \\ C_1 &= X_1 \oplus X_3 \oplus X_5 \oplus X_7 \oplus X_9 \oplus X_{10} \oplus X_{11} \oplus X_{14} \\ C_2 &= X_0 \oplus X_2 \oplus X_6 \oplus X_7 \oplus X_9 \oplus X_{12} \oplus X_{14} \oplus X_{15} \\ C_3 &= X_1 \oplus X_4 \oplus X_8 \oplus X_9 \oplus X_{12} \\ C_4 &= X_0 \oplus X_3 \oplus X_4 \oplus X_7 \oplus X_{12} \oplus X_{13} \oplus X_{15} \\ C_5 &= X_1 \oplus X_2 \oplus X_5 \oplus X_9 \oplus X_{10} \oplus X_{12} \oplus X_{14} \\ C_6 &= X_2 \oplus X_3 \oplus X_6 \oplus X_8 \oplus X_{10} \oplus X_{11} \oplus X_{13} \oplus X_{15} \end{aligned} \quad (6)$$

On the other hand, the syndrome formulas that can be obtained from **H** to check if an error has occurred are:

$$\begin{aligned} S_0 &= C_0 \oplus X_0 \oplus X_4 \oplus X_7 \oplus X_8 \oplus X_{11} \oplus X_{12} \oplus X_{13} \\ S_1 &= C_1 \oplus X_1 \oplus X_3 \oplus X_5 \oplus X_7 \oplus X_9 \oplus X_{10} \oplus X_{11} \oplus X_{14} \\ S_2 &= C_2 \oplus X_0 \oplus X_2 \oplus X_6 \oplus X_7 \oplus X_9 \oplus X_{12} \oplus X_{14} \oplus X_{15} \\ S_3 &= C_3 \oplus X_1 \oplus X_4 \oplus X_8 \oplus X_9 \oplus X_{12} \\ S_4 &= C_4 \oplus X_0 \oplus X_3 \oplus X_4 \oplus X_7 \oplus X_{12} \oplus X_{13} \oplus X_{15} \\ S_5 &= C_5 \oplus X_1 \oplus X_2 \oplus X_5 \oplus X_9 \oplus X_{10} \oplus X_{12} \oplus X_{14} \\ S_6 &= C_6 \oplus X_2 \oplus X_3 \oplus X_6 \oplus X_8 \oplus X_{10} \oplus X_{11} \oplus X_{13} \oplus X_{15} \end{aligned} \quad (7)$$

Our second proposal, called FUEC-TAEC, is able to correct an error in a single bit, or an error in 2 adjacent bits (2-bit burst errors) or a 3-bit burst error, or it can detect a 4-bit burst error. This is possible by adding one more code bit. In this case, for a 16-bit data word, the FUEC-TAEC code needs eight code bits. The parity check matrix **H** for this code is presented in Fig. 8. As in the case of the FUEC-DAEC, $C_i$ are the code bits and $X_i$ are the primary data bits. Similarly, from **H** it is very easy to design the encoder/decoder circuitry.

Finally, FUEC-QUAEC is the last code we have designed. This code is able to correct an error in a single bit, or an error in 2 adjacent bits (2-bit burst errors) or a 3-bit burst error or a 4-bit burst error. This can be done by using only nine code bits. The parity check matrix **H** for this code is shown in Fig. 9. As in the previous codes, $C_i$ are the code bits and $X_i$ are the primary data bits.

$$\mathbf{H} = \begin{array}{c} C_0 \cdots\cdots C_7\ X_0\ X_1 \cdots\cdots X_{15} \\ \left|\begin{array}{cccccccccccccccccccc} 1&0&0&0&0&0&0&0&1&0&0&1&1&0&0&1&1&0&0&1&0&0&0&1 \\ 0&1&0&0&0&0&0&0&0&1&0&0&1&1&0&0&1&1&0&0&1&1&0&0 \\ 0&0&1&0&0&0&0&0&0&0&1&0&0&1&0&0&0&1&1&0&1&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&1 \\ 0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&1&0&0&0&1&1&1 \\ 0&0&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&1&1&1&1&0 \\ 0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&1&0&1&1&1&0&0&1&0 \\ 0&0&0&0&0&0&0&1&0&0&0&1&0&0&1&0&1&0&0&0&1&0&1&0 \end{array}\right| \end{array}$$

Fig. 8. Parity check matrix **H** for the (24, 16) FUEC-TAEC code.

$$\mathbf{H} = \begin{array}{c} C_0 \cdots\cdots C_8\ X_0\ X_1 \cdots\cdots X_{15} \\ \left|\begin{array}{cccccccccccccccccccccc} 1&0&0&0&0&0&0&0&0&1&0&0&0&1&1&1&1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&1&0&0&0&1&0 \\ 0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&1&0&1&0&0&1&0&0&0&1 \\ 0&0&0&1&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&1&1&0&0&0 \\ 0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&1&0&1 \\ 0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&0&1&0&0&1&0 \\ 0&0&0&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&1&1&0&0&0 \end{array}\right| \end{array}$$

Fig. 9. Parity check matrix **H** for the (25, 16) FUEC-QUAEC code.

We have to remark that we have generated a parity check matrix optimized to achieve the lowest delay for our three codes, that is, with a reduced number of 1's in the rows with the highest number of 1's.

It is also remarkable that the names FUEC-*AEC only indicate that they have been designed using the FUEC methodology. The codes presented here must not been confused with the FUEC codes from [31], an improvement of UEC codes [11].

With respect to the code bits needed, our three codes present a very low redundancy with respect to the Matrix and CLC codes. Nevertheless, the lowest redundancy corresponds to both SEC-DAED codes, but this low redundancy only permits the correction of single errors, as it can be seen in TABLE IV. This table shows the number of code bits introduced by each ECC, as well as the redundancy introduced with respect to a 16-bit data word. Notice that although in this paper we have worked with 16-bit data word, FUEC methodology and algorithm can be applied to longer codeword sizes. Calculus of the redundancy has been done with:

$$Redundancy = \frac{No.\ code\ bits}{No.\ data\ bits} \times 100 \quad (8)$$

The importance of a low redundancy comes from the fact that these extra bits must be also stored in memory in order to check if an error has occurred. In this way, a higher redundancy means a lower storage available for data bits. As an example, if we use a 1GB memory chip, only 512MB are available to store data bits in the case of the Matrix code, requiring the remaining 512MB to store the code bits. In the case of the CLC code, only about 410MB are available to store data bits. On the other hand, in the case of the SEC-DAED codes, the available memory for data is about 780MB.

TABLE IV
NUMBER OF CODE BITS FOR A 16-BIT DATA WORD.

| CODE | No. Code Bits | % Redundancy | Burst error detection & correction capabilities |
|---|---|---|---|
| Matrix | 16 | 100 % | Correction: 100% of single bit errors<br>Detection: 100% of 2-bit burst errors |
| CLC | 24 | 150 % | Correction: 100% of single bit errors & 2-bit burst errors<br>Detection: 100% of 2-bit burst errors |
| SEC-DAED [27] | 5 | 31.25 % | Correction: 100% of single bit errors<br>Detection: 100% of 2-bit burst errors |
| SEC-DAED [28] | 5 | 31.25 % | Correction: 100% of single bit errors<br>Detection: 100% of 2-bit burst errors |
| FUEC-DAEC | 7 | 43.75 % | Correction: 100% of single bit errors & 2-bit burst errors<br>Detection: 100% of 3- & 4-bit burst errors |
| FUEC-TAEC | 8 | 50 % | Correction: 100% of single bit errors & 2- and 3-bit burst errors<br>Detection: 100% of 4-bit burst errors |
| FUEC-QUAEC | 9 | 56.25 % | Correction: 100% of single bit errors & 2-, 3- and 4-bit burst errors<br>Detection: 100% of 4-bit burst errors |

Following with this example, our FUEC-DAEC code allows storing about 712MB of primary data. In contrast, our FUEC-TAEC and FUEC-QUAEC codes allow storing 682MB and 655MB respectively. As it can be seen, the increment in the storage available is very significant, taking into account the improvement in the coverage properties of our three codes.

Last column of TABLE IV shows the burst error coverage of the different ECCs, a concern to have into account in space applications [9][10]. As shown in [9], MCUs mainly provoke 2-bit adjacent errors in earth observation satellites; although a non negligible percentage of longer burst errors are also presented. In deep space exploration, a higher impact of longer MCUs is expected.

## IV. ERROR CORRECTION CODES EVALUATION

In this section, we present the different results obtained during the evaluation of the ECCs presented in Section III. We have carried out two different processes. During the first one, we have injected faults in C models of the ECCs for error coverage evaluation. In a second step, we have implemented the different ECCs in VHDL and we have synthesized them, in order to estimate area, power and delay overheads. This section finishes with the analysis of the results obtained.

### *A. Error coverage evaluation*

In order to study the error coverage of the different ECCs, we have developed a simulator that allows injecting different types of error. The basic scheme is shown in Fig. 10.

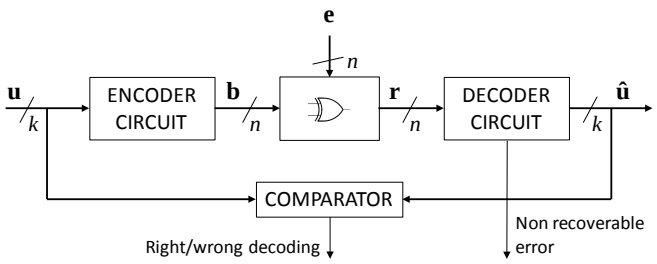


Fig. 10. Block diagram of the fault injector simulator.

This tool allows the injection of different types or error. By comparing the input and output words, the simulator can check if the error injected leads to a right or wrong decoding. Also, the decoder circuit can activate the NRE (*Non Recoverable Error*) signal when an error is detected but it cannot be corrected. Repeating the process for all errors of a given size and model (i.e. *random* or *burst*), it is possible to count the number of corrected and/or detected errors with respect to the total number of possible errors, that is, it is possible to calculate the coverage of each ECC. In this paper, we have injected *single errors*, as well as *burst errors* with a burst length varying from 2 to 8. This is a representative range in space applications [6][7]. Notice that burst errors can be adjacent or not, and affect to the layout of the rows. In the case of the Matrix code, we have considered that $C_3$ is adjacent to $X_5$, $C_6$ to $X_9$, etc. (see Fig. 2). In the same way, for the CLC code, $P_{a1}$ is adjacent to $X_5$, $P_{a2}$ to $X_9$, etc. (see Fig. 3).

We have to remark that we have not injected errors according to their probability of occurrence, as our goal is to measure the correction and detection coverages, that represent percentages. Specifically, we have injected each type of error (single errors or burst errors of different lengths) in all bits of the codeword to verify the error correction/detection capabilities of the different ECCs. Obviously, burst errors of length 8 will be much less frequent than burst errors of length 2, as bibliography shows [6][7].

All blocks of the fault injection tool have been developed in C, using the bitwise logic operators for an accurate simulation of the hardware behavior. Encoder and decoder circuits can be easily obtained from the parity check matrix **H**, as stated in Section III. These circuits are implemented in C as encoding and decoding functions. Changing the simulator for a different ECC is as simple as adjusting the word lengths and replacing the encoding and decoding functions for the new ECC.

Fig. 11 shows the results obtained for the correction coverage of each code. This coverage has been calculated as:

$$C_{correc} = \frac{Errors\_Corrected}{Errors\_Injected} \times 100 \qquad (9)$$

where $Errors\_Corrected$ are the number of errors corrected by the ECC, and $Errors\_Injected$ are the total amount of errors injected for a given burst length.

As expected, FUEC-TAEC and FUEC-QUAEC codes

present better correction coverage than Matrix, CLC and both SEC-DAED codes, as they are able to correct up to 3- and 4-bit burst errors respectively. On the other hand, our FUEC-DAEC code can correct up to 2-bit burst errors, like CLC.

Nevertheless, for longer burst errors (5 bits or more), the correction capabilities of our codes degrade more deeply than Matrix and CLC ones. This is provoked by the lower redundancy of our codes, which causes a lower number of available syndromes to be used to correct longer burst errors. In other words, our codes employ the available syndromes in a more efficient way inside the expected error rank, but degrading quickly outside this rank. This effect can be seen also in both SEC-DAED codes. In fact, these codes present the greatest degradation from multiple bit errors (2-bit burst errors or longer). As it happens with our codes, the very low redundancy of both SEC-DAED codes provokes this result.

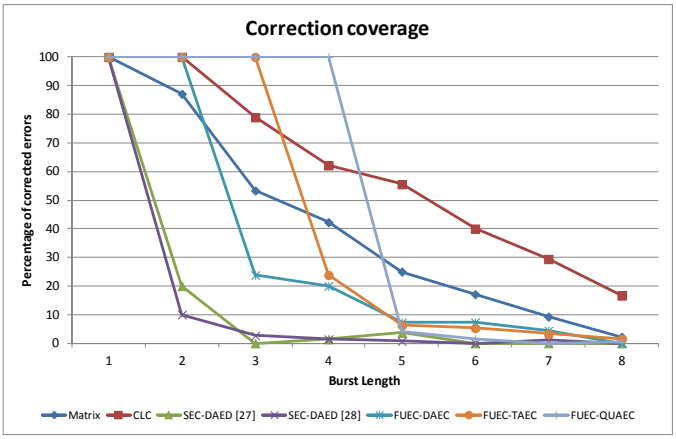


Fig. 11. Correction coverage for burst errors.

Finally, Fig. 12 shows the detection coverage for all codes. This detection coverage is calculated as:

$$C_{detec} = \frac{Errors\_Corrected + Errors\_Detected}{Errors\_Injected} \times 100 \qquad (10)$$

where $Errors\_Detected$ corresponds to the number of errors not corrected but detected by the ECCs.

As it can be seen in Fig. 12, all our codes present a 100% detection of up to 4-bit burst errors, improving the behavior of Matrix, CLC and both SEC-DAED codes.

As in the correction coverage, the percentage of detected errors in our FUEC-TAEC and FUEC-QUAEC codes degrades sharply for longer bursts. By contrast, FUEC-DAEC maintains a coverage over 60%, near the Matrix detection capability.

In conclusion, our codes are very efficient to tolerate burst errors of from 2 to 4 bit length. Beyond 4-bit burst errors, the performance of our codes decreases notably due to their low redundancy. If these errors are expected to occur, more powerful codes (with a higher redundancy) must be employed. Nevertheless, the probability of occurrence of burst errors decreases significantly when increasing its length [6][7].

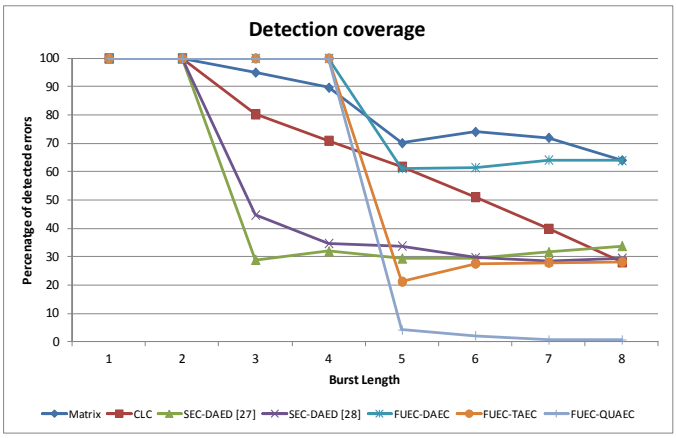


Fig. 12. Detection coverage for burst errors.

### *B. Synthesis results*

We have shown in Section III that our three codes, FUEC-DAEC, FUEC-TAEC and FUEC-QUAEC, present a lower redundancy with respect to Matrix and CLC codes. This lower redundancy provokes a lower storage requirement for code bits.

Nevertheless, the question that arises now is that this lower redundancy would translate into an improvement on the area, power and delay overheads in the encoder and decoder circuits with respect to Matrix, CLC and both SEC-DAED codes. Although the area overhead of the encoders and decoders may be negligible in comparison with memory overhead, power and delay overheads can be important, especially in deep space systems.

To solve this question, we have synthesized the encoder and decoder circuits for all ECCs. To do this, we have implemented them in VHDL, and using CADENCE software [35], we have carried out a logic synthesis for 45 nm technology by using the NanGate FreePDK45 Open Cell Library [36][37].

TABLE V (and Fig. 13) shows the area occupied by the different circuits in $nm^2$ ($1nm = 10^{-9}m$). As it can be seen, our encoders present a slightly greater area than the encoders of both SEC-DAED codes. The worst area numbers for the encoders correspond to the CLC and Matrix codes. These results are provoked by their higher redundancy, which provokes complex encoders' circuitry.

TABLE V
AREA OCCUPIED BY THE DIFFERENT ECCS (IN $nm^2$)

| CODE | ENCODER | DECODER | TOTAL |
|---|---|---|---|
| Matrix | 142 | 451 | 593 |
| CLC | 214 | 482 | 696 |
| SEC-DAED [27] | 87 | 242 | 329 |
| SEC-DAED [28] | 93 | 246 | 339 |
| FUEC-DAEC | 91 | 348 | 439 |
| FUEC-TAEC | 92 | 563 | 655 |
| FUEC-QUAEC | 95 | 790 | 885 |

Regarding decoders' area, the SEC-DAED ones present the lowest numbers. On the other hand, decoders for FUEC-TAEC and FUEC-QUAEC codes present the biggest area overhead, as they are able to correct and/or detect more errors.

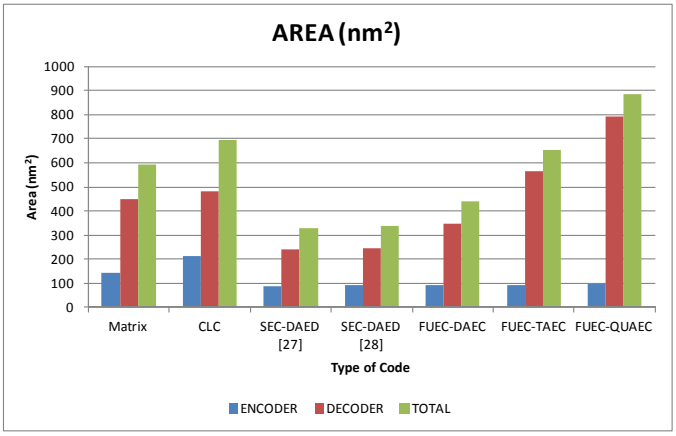


Fig. 13. Area occupied by the different ECCs (in $nm^2$).

In general, both SEC-DAED codes present the lowest area overhead. Nevertheless, their coverage capabilities are very simple with respect to the other codes. Comparing codes with similar coverage capabilities (i.e. correction of 2-bit burst errors done by Matrix, CLC and FUEC-DAEC codes), the FUEC-DAEC circuitry occupies the smallest area. In addition, its redundancy is much smaller (less than a half). This low redundancy reduces also the silicon area needed by the memory to store the code bits.

With respect to the FUEC-QUAEC code, it presents higher error coverage, and therefore, it introduces a higher hardware cost and a larger area, mainly in the decoders. Nevertheless, as stated above, its low redundancy reduces the area needed by the memory. On the other hand, the area occupied by the FUEC-TAEC code is lower than the area overhead of CLC, with better error coverage.

TABLE VI (and Fig. 14) shows the power (static and dynamic) consumption overhead of the different ECCs (in μW, $1\mu W = 10^{-6}W$). In global terms, both SEC-DAED codes present the lowest power consumption (encoder plus decoder). With respect to our three codes, our FUEC-DAEC code presents a slightly higher power consumption than both SEC-DAED codes, and much better numbers than Matrix and CLC codes. The worst score corresponds to the FUEC-QUAEC code. It is noticeable the low power consumption of the FUEC-TAEC code with regard to its error coverage.

Particularly, we can observe that power consumption of the encoders presents a similar trend with respect to area overhead, that is, CLC codes present the higher numbers, while the rest of codes show similar numbers. On the other hand, as our three codes present a better error coverage, the decoders power overhead also increase.

It is also interesting to remember that power consumption of memory has not been taken into account in these results. As our codes present a very low redundancy, they need a lower storage capacity. In this way, our codes would imply an additional power reduction of the redundant memory.

TABLE VI
POWER CONSUMPTION (IN μW)

| CODE | ENCODER | DECODER | TOTAL |
|---|---|---|---|
| Matrix | 191,31 | 492,76 | 684,07 |
| CLC | 331,79 | 457,62 | 789,41 |
| SEC-DAED [27] | 141,80 | 321,77 | 463,57 |
| SEC-DAED [28] | 181,09 | 320,90 | 501,99 |
| FUEC-DAEC | 156,91 | 396,19 | 553,10 |
| FUEC-TAEC | 194,11 | 616,15 | 810,26 |
| FUEC-QUAEC | 191,00 | 836,47 | 1027,47 |

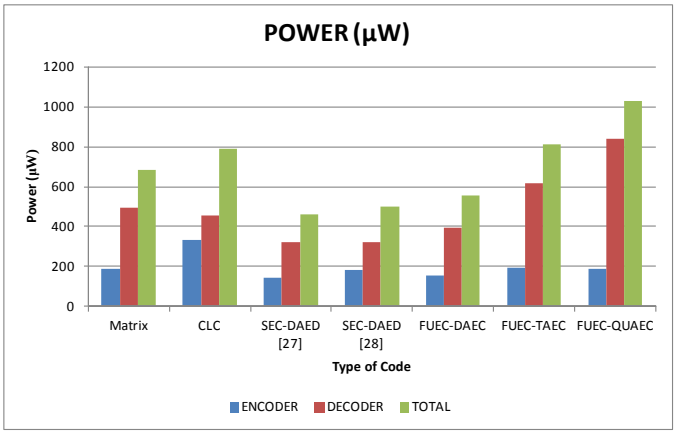


Fig. 14. Power consumption (in μW).

Finally, TABLE VII (and Fig. 15) shows the delay introduced by the different codes (in ps, $1ps = 10^{-12}s$). We do not represent the total delay because encoders and decoders work in independent operations (writing and reading).

With respect to the encoders delay, CLC presents the highest delay, while Matrix shows the lowest delay. With regard to our encoders, they present an intermediate delay, except the FUEC-DAEC encoder, that introduces a delay slightly lower than the CLC one. It is noticeable the low delay introduced by the encoder of the FUEC-QUAEC code. Although this code presents the highest redundancy of our three codes, its parity check matrix presents a more balanced number of 1's in each row, reducing in this way the delay needed.

In the case of the decoders, our FUEC-DAEC code presents the fastest correction, while FUEC-TAEC and FUEC-QUAEC codes present the highest delay, but also the highest error coverage. The rest of the codes show similar results.

TABLE VII
DELAY OVERHEAD FOR THE DIFFERENT ECCS (IN PS)

| CODE | ENCODER | DECODER |
|---|---|---|
| Matrix | 197 | 552 |
| CLC | 290 | 543 |
| SEC-DAED [27] | 274 | 501 |
| SEC-DAED [28] | 274 | 576 |
| FUEC-DAEC | 286 | 383 |
| FUEC-TAEC | 245 | 780 |
| FUEC-QUAEC | 219 | 772 |

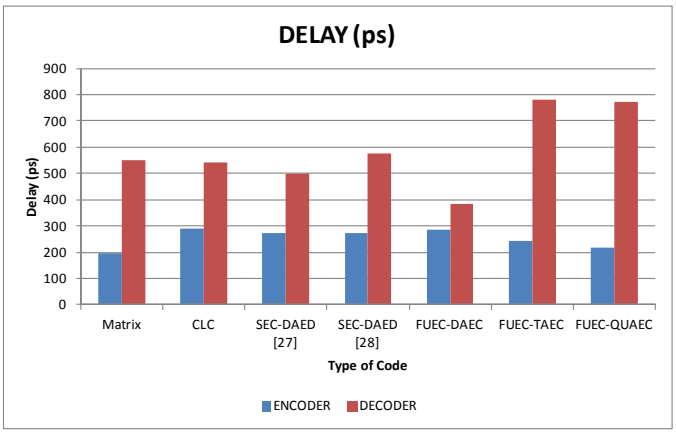


Fig. 15. Delay overhead (in ps).

### C. *Global evaluation of ECCs. M metric*

With the results obtained in previous sections, it is not easy to provide a global evaluation of the ECCs. To solve this question, several metrics has been proposed [17][18][19]. For instance, the TCC metric [19] can be used to tradeoff area, power, delay and error coverage. We have modified this metric to add the contribution of an important factor: the redundancy. We have included this parameter because the storage size for the different ECCs is reduced with a lower redundancy. Indirectly, area and power overheads can be affected. In this way, we have presented a generic metric with a very simple expression. Thus, if we want to enhance a parameter in a specific application, it is possible to weigh any of them with different weights; or we can include some limitations, such as reaching a certain correction or detection level. In this way, by including the redundancy, we can expect a metric more complete and accurate. The new metric, that we have called **M**, is presented in (11):

$$M = \frac{C_{correc} \times C_{detec}}{Area \times Power \times Delay \times Redundancy} \quad (11)$$

With respect to the *delay* factor, we have analyzed the longest delay (the worst case), that corresponds to the decoder part. In the case of area and power, we have used the total values (encoder plus decoder).

As it can be seen in Fig. 16, both SEC-DAED codes present the best **M** score for single errors. This is an expected result due to the simplicity of their encoders and decoders and their low redundancy. When multiple errors are present, the FUEC-DAEC code presents the best **M** score for 2-, 3- and 4-bit burst error length. In this way, FUEC-DAEC code seems the most adequate for short burst errors, as it can correct all single and 2-bit burst errors, as well as to detect all 3- and 4-bit burst errors.

In any case, when multiple errors are present, our three codes present a best **M** score than Matrix and CLC codes. We can observe also that CLC performance is the worst from single errors up to 4-bit burst errors length. Due to its great redundancy, **M** metric for CLC is greatly penalized.

To sum up, our three codes (FUEC-DAEC, FUEC-TAEC and FUEC-QUAEC) present the best **M** scores for 2-, 3- and 4-bit burst errors. On the other hand, CLC code presents the worst **M** numbers for these burst lengths. In this way, our three codes seems a good choice when multiple errors are present, as it is expected in spatial applications.

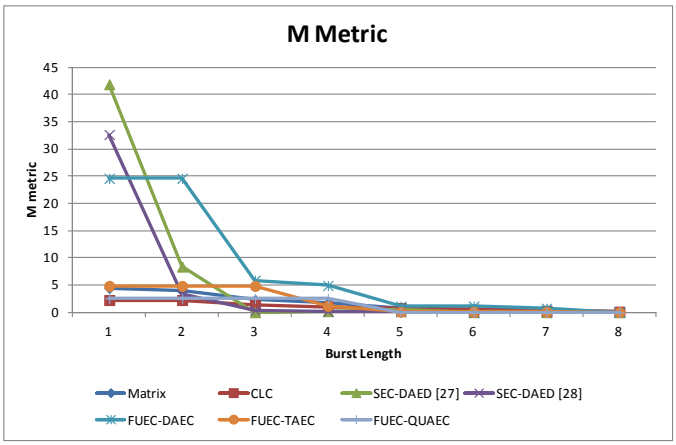


Fig. 16. Global evaluation applying **M** metric.

## V. CONCLUSION

In this work, a series of new ECCs have been presented. These new ECCs improve the behavior of the well-known Matrix code, the recently introduced CLC code and different SEC-DAED codes.

A characteristic of our codes is the low redundancy they introduce. This fact provokes a reduction in the storage needed for the code bits. In addition, the detection and correction capabilities are maintained, or even increased.

The insertion of an ECC in memory also provokes the introduction of area, power and delay overheads. Relating to the area overhead, we have seen that FUEC-DAEC code introduces a much lower area overhead than the CLC and Matrix codes, while FUEC-TAEC code presents a similar area overhead than Matrix and CLC codes. As expected, FUEC-QUAEC code exhibits the highest overhead, related to its high correction and detection capabilities. Also, we have to take into account the reduction of memory area overhead due to the low redundancy of our codes.

Concerning the power overhead, the trend is similar to the area overhead. FUEC-DAEC code introduces a much lower power overhead than the CLC and Matrix codes. Even, the power overhead of the FUEC-DAEC code is similar to the SEC-DAED codes ones. FUEC-TAEC and FUEC-QUAEC codes present a power consumption similar or a little bigger than CLC code, but with much better error correction and detection capabilities. And also, we have to take into account the reduction of memory power consumption due to the low redundancy of our codes.

With respect to the delay, FUEC-DAEC presents the fastest correction, even better than both SEC-DAED codes. On the contrary, FUEC-TAEC and FUEC-QUAEC codes introduce the highest correction delay, because they are designed to correct longer burst errors. In this way, decoder circuits are more complex. Nevertheless, encoder circuits are quite quick.

In general, and according to the **M** metric, introduced to evaluate the overall features of the analyzed codes, our FUEC-

DAEC code presents the best performance to correct single or 2-bit burst errors, or to detect 3- or 4-bit burst errors. Additionally, FUEC-TAEC and FUEC-QUAEC have demonstrated good features to correct 3-bit and 4-bit burst errors respectively. Thus, our codes become an appropriate option for critical applications in embedded systems. Beyond 4-bit burst errors, the performance of our codes decreases notably due to their low redundancy. If these errors are expected to occur, more powerful ECCs must be employed.

In a future work, we want to continue developing ECCs, decreasing area, power and delay overhead while maintaining, or even increasing, the code coverage. We want focus on long burst errors, which are expected to have more and more impact in space systems.

## REFERENCES


[1] The International Technology Roadmap for Semiconductors 2013. [Online]. Available at: http://www.itrs2.net/2013-itrs.html

[2] S.K. Kurinec and K. Iniewsky. *Nanoscale Semiconductor Memories: Technology and Application*, CRC Press, Taylor & Francis Group, 2014.

[3] J. Barak, M. Murat, and A. Akkerman, "SEU due to electrons in silicon devices with nanometric sensitive volumes and small critical charge", Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms, vol. 287, pp. 113–119, September 2012.

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

[5] G. Tsiligiannis et. al., "Multiple Cell Upset Classification in Commercial SRAMs", IEEE Transactions on Nuclear Science, vol. 61, no. 4, August 2014.

[6] G.I. Zebrev, "Multiple Cell Upset Cross-Section Uncertainty in Nanoscale Memories: Microdosimetric Approach", 15th European Conference on Radiation and its Effects on Components and Systems (RADECS 2015), September 2015.

[7] N.G. Chechenin and M. Sajid, "Multiple cell upsets rate estimation for 65 nm SRAM bit-cell in space radiation environment", 3rd International Conference and Exhibition on Satellite & Space Missions, May 2017.

[8] N.N. Mahatme, B.L. Bhuva, Y.P. Fang, and A.S. Oates, "Impact of strained-Si PMOS transistors on SRAM soft error rates", IEEE Trans. on Nuclear Science, vol. 59, no. 4, pp. 845–850, August 2012.

[9] Y. Bentoutou, "Program memories error detection and correction on-board earth observation satellites", International Journal of electrical and Computer Engineering, vol. 4, nº 6, pp. 933-936, 2010.

[10] M.J. Gadlage et al., "Multiple-Cell Upsets Induced by Single High-Energy Electrons", IEEE Transactions on Nuclear Science, DOI: 10.1109/TNS.2017.2756441, September 2017.

[11] E. Fujiwara, Code Design for Dependable Systems: Theory and Practical Application, Ed. Wiley-Interscience, 2006.

[12] R. W. Hamming, "Error detecting and error correcting codes," Bell System Technical Journal, vol. 29, pp. 147–160, 1950.

[13] C.L. Chen and M.Y. Hsiao, "Error-correcting codes for semiconductor memory applications: a state-of-the-art review", IBM Journal of Research and Development, vol. 58, no. 2, pp. 124–134, March 1984.

[14] G.C. Cardarilli, M. Ottavi, S. Pontarelli, M. Re, and A. Salsano, "Fault Tolerant Solid State Mass Memory for Space Applications", IEEE Trans. on Aerospace and Electronic Systems, vol. 41, no. 4, pp. 1353–1372, October 2005.

[15] S. Pontarelli, G.C. Cardarilli, M. Re and A. Salsano, "Error correction codes for SEU and SEFI tolerant memory systems", 24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2009), pp. 425-430, 2009.

[16] A. Sánchez-Macián, P. Reviriego, J. Tabero, A. Regadío, and J.A. Maestro, "SEFI protection for Nanosat 16-bit Chip On-Board Computer Memories", IEEE Transactions on Device and Materials Reliability, DOI 10.1109/TDMR.2017.2750718, 2017.

[17] C. Argyrides, D.K. Pradhan, and T. Kocak, "Matrix codes for reliable and cost efficient memory chips", IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 19, nº 3, pp.420–428, March 2011.

[18] C. Argyrides, H.R. Zarandi and D.K. Pradhan, "Matrix Codes: Multiple Bit Upsets Tolerant Method for SRAM Memories", 22nd IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, 2007.

[19] H.S. de Castro, et al. "A correction code for multiple cells upsets in memory devices for space applications", 2016 14th IEEE International New Circuits and Systems Conference (NEWCAS 2016), pp.1–4, June 2016.

[20] S. Ahmad, M. Zahra. S.Z. Farooq, and A. Zafar, "Comparison of EDAC schemes for DDR memory in space applications", 2013 International Conference on Aerospace Science & Engineering (ICASE 2013), August 2013.

[21] D.E. Muller, "Application of boolean algebra to switching circuit design and to error detection", IRE Transactions on Electronic Computers, vol. 3, pp. 6–12, 1954.

[22] J. M. Berger, "A note on an error detection code for asymmetric channels", Information and Control, vol. 4, pp. 68–73, March 1961.

[23] O. Goldreich and L. Levin, "A hard-core predicate for all one-way functions", 21st ACM Symposium on Theory of Computing, pp. 25–32, 1989.

[24] M. Bossert, *Channel Coding for Telecommunications*. Wiley. 1999.

[25] M.J.E. Golay, "Notes on Digital Coding", Proc. IRE, vol. 57, p. 657, 1949.

[26] N.B. Anne, U. Thirunavukkarasu, S. Latifi, "Three and Four-dimensional Parity-check Codes for Correction and Detection of Multiple Errors", International Conference on Information Technology: Coding and Computing (ITCC 2004), April 2004.

[27] L. J. Saiz-Adalid et al., "Modified Hamming codes to enhance short burst error detection in semiconductor memories", IEEE European Dependable Computing Conference (EDCC), pp. 62-65, Newcastle, UK, May 13-16, 2014.

[28] A. Sanchez-Macian, P. Reviriego, and J. A. Maestro, "Hamming SEC-DAED and extended hamming SEC-DED-TAED codes through selective shortening and bit placement," IEEE Transactions on Device and Materials Reliability (TDMR), vol. 14, no. 1, pp. 574-576, 2014.

[29] A. Neubauer, J. Freudenberger, and V. Kühn, *Coding Theory: Algorithms, Architectures and Applications*. John Wiley & Sons, 2007.

[30] H.S. de Castro, et al. "Evaluation of multiple bit upset tolerant codes for NoCs buffering", 2017 IEEE 8th Latin American Symposium on Circuits & Systems (LASCAS), February 2017.

[31] L.J. Saiz-Adalid et al., "Flexible Unequal Error Control Codes with Selectable Error Detection and Correction Levels", 32th International Conference on Computer Safety, Reliability and Security (SAFECOMP 2013), pp. 178-189, September 2013.

[32] K.A. LaBel, "Proton single event effects (SEE) guideline" submitted for publication on the NASA Electronic Parts and Packaging (NEPP) Program web site, August 2009. Available online at https://nepp.nasa.gov/files/18365/Proton_RHAGuide_NASAAug09.pdf

[33] M. Greenberg, "Reliability, availability, and serviceability (ras) for ddr dram interfaces," in memcon.com. Available at: http://www.memcon.com/pdfs/proceedings2014/NET105.pdf, 2014.

[34] S. Shamshiri and K.T. Cheng, "Error-Locality-Aware Linear Coding to Correct Multi-bit Upsets in SRAMs," IEEE International Test Conference (ITC 2010), pp. 1-10, November 2010

[35] https://www.cadence.com/

[36] J.E Stine et al., "FreePDK: An Open-Source Variation-Aware Design Kit", IEEE International Conference on Microelectronic Systems Education (MSE'07), June 2007.

[37] http://www.nangate.com/?page_id=2325

[38] A. Mukati, "Review: A survey of memory error correcting techniques for improved reliability", Journal of Network and Computer Applications, vol. 34, no. 2, pp. 517-522, March 2011.

**Joaquín Gracia-Morán** is B.Sc. (1995), M.Sc. (1997) and Ph.D. (2004) in Computer Engineering from the Universitat Politècnica de València (UPV). He is currently an associate professor at the UPV, Spain, where he teaches in the Department of Computer Engineering (DISCA). He is member with the Fault-Tolerant Systems (STF) research line within the Instituto ITACA. His research interests include design and implementation of digital systems, design and validation of Fault-Tolerant Systems and VHDL-based Fault Injection

**Luis J. Saiz-Adalid** is M.Sc. (1995) and Ph.D (2015) in Computer Engineering from the Universitat Politècnica de València (UPV). After 15 years in the industry (IBM, 1995-Celestica, 1998), he is currently an associate professor at the UPV, Spain, where he teaches in the Department of Computer Engineering (DISCA). He is also a member with the STF-ITACA. His research interests include design and implementation of digital systems, design and validation of Fault-Tolerant Systems and design of Error Correction Codes.

**Daniel Gil-Tomás** is an associate professor at the Universitat Politècnica de València, Spain, in the DISCA. He received his B.Sc. degree in Electrical and Electronic Physics from the Universitat de València in 1985. He obtained his Ph.D. degree on Computer Engineering from the UPV in 1999. He is currently an associate professor at the UPV, Spain, where he teaches in the Department of Computer Engineering (DISCA). He is a member with the STF-ITACA. His research interests include design and validation of Fault-Tolerant Systems, Reliability Physics and Reliability of Emerging Nanotechnologies.

**Pedro J. Gil-Vicente** is professor at the Universitat Politècnica de València (UPV), where he has been head of the Department of Computer Engineering (DISCA). Professor Gil has taught courses on Computer Technology, Digital Design, Computer Networks and Fault Tolerant Systems. He is the head of the Fault-Tolerant Systems (STF) research line, within ITACA. His research focuses on the design and validation of real-time fault-tolerant distributed systems, the dependability validation using fault injection, the design and verification of embedded systems, and the dependability and security benchmarking. He has authored more than 100 research papers on these subjects. He has also served as Program Committee member in the IEEE International Conference on Dependable Systems and Networks (DSN), the European Dependable Computing Conference (EDCC) and the Latin American Symposium on Dependable Computing (LADC), and as reviewer in international magazines and congresses related to dependability and security. He was general chair of the EDCC-8 conference, held in Valencia on April 2010.