Recovering Multiple-Cell Upsets Using Low Power Error Correction Codes in Memory Applications

A.Saran, M.E(VLSI DESIGN), Theni Kammavar Sangam College Of Technology, Theni.

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 is 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 paper, 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 (ECCs), fault tolerance, multiple-cell upsets (MCUs), reliability.

I. INTRODUCTION

Recently, the continued physical feature size down-scaling 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]–[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],...
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]–[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, such as space applications, more complex and sophisticated codes are used [14]–[17], [19]–[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 bit scales with the memory capacity. For example, if an ECC with 100% of redundancy is employed in a 2-GB memory, only 1 GB is available to store the payload (the “clean” data); the remaining 1 GB 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 paper, 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 Saiz-Adalid et al. [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.
II. TO THE DESIGN OF ERROR CORRECTION CODES

A. Background on ECCs for Space Applications

Different ECCs have been traditionally applied to space missions [20]. For instance, Berger code [22] or the well-known parity code has 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], Bose–Chaudhuri–Hocquenghem (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, and 64), Hamming codes can detect some double-error patterns, in addition to the SEC. Exploiting this feature, it is possible to systematize the detection of 2-bit adjacent errors with the same redundancy, as presented in [27] and [28]. In these works, different ECCs based on Hamming codes are introduced. These ECCs allow 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 inconvenience 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] and [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 paper, 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 \(\mathbf{u} = (u_0, u_1, \ldots, u_k)\) is a \(k\)-bit vector which represents the original data. The code word \(\mathbf{b} = (b_0, b_1, \ldots, b_{n-1})\) 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 \(\mathbf{b} \oplus \mathbf{e}\). Fig. 1 synthesizes this encoding, channel crossing, and decoding process.

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

\[
s^\top = \mathbf{H} \cdot \mathbf{r}^\top = \mathbf{H} \cdot (\mathbf{b} \oplus \mathbf{e})^\top = \mathbf{H} \cdot \mathbf{b}^\top \mathbf{H} \cdot \mathbf{e}^\top = \mathbf{H} \cdot \mathbf{e}^\top,
\]

(1)

There must be a different \(s\) for each correctable \(\mathbf{e}\). If \(s = \mathbf{0}\), 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 \(s\) with the decoded error vector \(\hat{\mathbf{e}}\). The decoded code word \(\hat{\mathbf{b}}\) is calculated as just discarding \(\mathbf{b} = \hat{\mathbf{b}} \oplus \hat{\mathbf{e}}\). From \(\mathbf{h}\), it is easy to obtain \(\hat{\mathbf{b}}\), \(\hat{\mathbf{e}}\), 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 SEUs (in random access memories) or single-event transients (in combinational logic) [32].
TABLE I
ENCODING FORMULAS FOR THE HAMMING (7, 4) CODE

<table>
<thead>
<tr>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$u_1$</th>
<th>$u_2$</th>
<th>$u_3$</th>
<th>$u_4$</th>
<th>$u_5$</th>
<th>$u_6$</th>
<th>$u_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>$s_1$</th>
<th>$s_2$</th>
<th>$s_3$</th>
<th>$u_1$</th>
<th>$u_2$</th>
<th>$u_3$</th>
<th>$u_4$</th>
<th>$u_5$</th>
<th>$u_6$</th>
<th>$u_7$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As it was commented in Section I, with the continuous increasing of the integration scale, multiple errors are becoming more frequent [4–8]. 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 adjacent 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 the following equation:

$$
\begin{bmatrix}
1010101 \\
0110011
\end{bmatrix}
\begin{bmatrix}
H
\end{bmatrix} =
\begin{bmatrix}
001211
\end{bmatrix}.
$$

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

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.

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

E. Flexible Unequal Error Control Methodology

The methodology employed to design the error control codes introduced in this paper can be found in [31]. This methodology was developed to obtain 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 $E^+$ represents the set of error vectors to be corrected, and $E_b$ is the set of error vectors that can be detected.

Hamming codes can be extended to correct single errors and detect double random errors. These codes are known as SEC–DED 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:

$$
b_7 = b_0 \oplus b_1 \oplus b_2 \oplus b_3 \oplus b_5 \oplus b_6 \oplus b_8.
$$

The decoding process checks two conditions: 1) the parity of the whole received word and 2) 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 nonrecoverable error (NRE) detected but it cannot be corrected.

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.
vectors to be detected. That is, each correctable error must generate a different syndrome

$$\mathbf{H} \cdot \mathbf{e}^T = \mathbf{r}_i$$  \quad \forall \mathbf{e}_i \in \mathcal{E}_r, \mathbf{e}_j \in \mathcal{E}_s. \tag{4}$$

In addition, each detectable error must generate a syndrome which is different to all the syndromes generated by the correctable errors

$$\mathbf{H} \cdot \mathbf{e}^T = \mathbf{r}_i$$  \quad \forall \mathbf{e}_i \in \mathcal{E}_r, \mathbf{e}_j \in \mathcal{E}_s. \tag{5}$$

However, several detectable errors may have the same syndrome.

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 \(\mathbf{H}\) matrix. New columns, with \(n-k\) rows, are added, and the new partial matrices are checked recursively. The added columns must be nonzero, 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 \(\mathbf{H}\) 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 2-D 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).

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 paper, this division is in groups of 4 bits. Each group is codified by a \((7, 4)\) Hamming code \((C_i)\). Last, a set of vertical parity bits \((P_i)\) completes the matrix. The Matrix code implemented in this paper 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, 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 an SEC–DED \((8, 4)\) extended Hamming code \((C_i \text{ and } P_i)\). 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.

On the other hand, Saiz-Adalid et al. [27] introduce an SEC–double-adjacent error detection (SEC–DAED) 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 five 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.

On the contrary, Sanchez-Macian et al. [28] present a different approach to generate Hamming-based ECCs. In this case, we have selected an 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, as 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] and [28], respectively.

#### 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 second proposal, called FUEC–triple adjacent error correction (TAEC), is able to correct an error in a single bit, or an error in two 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 \( \mathbf{H} \) 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 \( \mathbf{H} \) it is very easy to design the encoder/decoder circuitry.

Finally, FUEC–quadruple adjacent error correction (QUAEC) is the last code we have designed. This code is able to correct an error in a single bit, or an error in two 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 \( \mathbf{H} \) 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.

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 be 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 can be seen from Table IV. Table IV 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

\[
\text{Redundancy} = \frac{\text{No. code bits}}{\text{No. data bits}} \times 100.
\]

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 1-GB memory chip, only 512 MB are available to store data bits in the case of the Matrix code.
TABLE IV
NUMBER OF CODE BITS FOR A 16-BIT DATA WORD

<table>
<thead>
<tr>
<th>CODE</th>
<th>No. Code Bits</th>
<th>% Redundancy</th>
<th>Burst Error Detection &amp; Correction Capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix</td>
<td>16</td>
<td>100%</td>
<td>Correction: 100% of single bit errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Detection: 100% of 2-bit burst errors</td>
</tr>
<tr>
<td>CLC</td>
<td>24</td>
<td>150%</td>
<td>Correction: 100% of single bit errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Detection: 100% of 2-bit burst errors</td>
</tr>
<tr>
<td>SEC-DAED [27]</td>
<td>5</td>
<td>31.25%</td>
<td>Correction: 100% of single bit errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Detection: 100% of 2-bit burst errors</td>
</tr>
<tr>
<td>SEC-DAED [28]</td>
<td>5</td>
<td>31.25%</td>
<td>Correction: 100% of single bit errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Detection: 100% of 2-bit burst errors</td>
</tr>
<tr>
<td>FUEC-DAEC</td>
<td>7</td>
<td>43.75%</td>
<td>Correction: 100% of single bit errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Detection: 100% of 2-bit burst errors</td>
</tr>
<tr>
<td>FUEC-TAEC</td>
<td>8</td>
<td>59%</td>
<td>Correction: 100% of single bit errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Detection: 100% of 2-bit burst errors</td>
</tr>
<tr>
<td>FUEC-QUAEC</td>
<td>9</td>
<td>56.25%</td>
<td>Correction: 100% of single bit errors</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Detection: 100% of 2-bit burst errors</td>
</tr>
</tbody>
</table>

requiring the remaining 512 MB to store the code bits. In the case of the CLC code, only about 410 MB 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 780 MB. Following this example, our FUEC–DAEC code allows storing about 712 MB of primary data. In contrast, our FUEC–TAEC and FUEC–QUAEC codes allow storing 682 and 655 MB, 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 nonnegligible 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 very high speed integrated circuit hardware description language (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.

This tool allows the injection of different error types. 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 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 C3 is adjacent to X5, C6 to X6, etc. (see Fig. 2). In the same way, for the CLC code, Pst is adjacent to X5, Pst to Xn, 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 bibliographic 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

$$C = \frac{\text{Errors\_Corrected}}{\text{Errors\_Injected}} \times 100 \quad (9)$$

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, such as 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 range, but degrading quickly outside this range. 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.

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

$$C_{\text{detect}} = \frac{\text{Errors\_Corrected} + \text{Errors\_Detected}}{\text{Errors\_Injected}} \times 100 \quad (10)$$

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

As can be seen from 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].

### 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 capability.
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 \( \text{nm}^2 \) (1 \( \text{nm} = 10^{-9} \text{m} \)). 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.

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.

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 μW =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.

Finally, Table VII (and Fig. 15) shows the delay introduced by the different codes (in ps, 1 ps =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 FUOE–DAEC code presents the fastest correction, while FUEC–TAEC and FUEC–QUAE code present the highest delay, but also the highest error coverage. The rest of the codes show similar results.

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 have been proposed [17]–[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 the following equation:

$$ M = \frac{C_{cor} \times C_{dec}}{\text{Area} \times \text{Power} \times \text{Delay} \times \text{Redundancy}}. $$

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 can be seen from Fig. 16, both SEC–DAEC 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 FUOE–DAEC code presents the best $M$ score for 2-, 3-, and 4-bit burst error length. In this way, FUOE–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 (FUOE–DAEC, FUOE–TAEC, and FUOE–QUAE) 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.

V. CONCLUSION

In this paper, a series of new ECCs has been presented. These new ECCs improve the behavior of the well-known Matrix code, the recently introduced CLC code and different SEC–DAEC 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 FUOE–DAEC code introduces a much lower area overhead than the CLC and Matrix codes, while FUOE–TAEC code presents a similar area overhead than Matrix and CLC codes. As expected, FUOE–QUAE 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. FUOE–DAEC code introduces a much lower power overhead than the CLC and Matrix codes. Even, the power overhead of the FUOE–DAEC code is similar to the SEC–DAEC codes ones. FUOE–TAEC and FUOE–QUAE 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–TAE 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. In addition, FUEC–TAE 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 overheads while maintaining, or even increasing, the code coverage. We want focus on long burst errors, which are expected to have more and more impact on space systems.

REFERENCES
