Received September 26, 2019, accepted October 9, 2019, date of publication October 14, 2019, date of current version October 29, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2947315

Ultrafast Codes for Multiple Adjacent Error Correction and Double Error Detection

LUIS-J. SAIZ-ADALID, JOAQUÍN GRACIA-MORÁN, DANIEL GIL-TOMÁS, J.-CARLOS BARAZA-CALVO, AND PEDRO-J. GIL-VICENTE, (Member, IEEE)
Institute of Information and Communication Technologies (ITACA), Universitat Politècnica de València, 46022 Valencia, Spain

Corresponding author: Luis-J. Saiz-Adalid (ljsaiz@itaca.upv.es)

This work was supported in part by the Spanish Government under Project TIN2016-81075-R, and in part by the Primeros Proyectos de Investigación, Vicerrectorado de Investigación, Innovación y Transferencia de la Universitat Politècnica de València (UPV), València, Spain, under Project PAID-06-18 20190032.

ABSTRACT Reliable computer systems employ error control codes (ECCs) to protect information from errors. For example, memories are frequently protected using single error correction-double error detection (SEC-DED) codes. ECCs are traditionally designed to minimize the number of redundant bits, as they are added to each word in the whole memory. Nevertheless, using an ECC introduces encoding and decoding latencies, silicon area usage and power consumption. In other computer units, these parameters should be optimized, and redundancy would be less important. For example, protecting registers against errors remains a major concern for deep sub-micron systems due to technology scaling. In this case, an important requirement for register protection is to keep encoding and decoding latencies as short as possible. Ultrafast error control codes achieve very low delays, independently of the word length, increasing the redundancy. This paper summarizes previous works on Ultrafast codes (SEC and SEC-DED), and proposes new codes combining double error detection and adjacent error correction. We have implemented, synthesized and compared different Ultrafast codes with other state-of-the-art fast codes. The results show the validity of the approach, achieving low latencies and a good balance with silicon area and power consumption.

INDEX TERMS Adjacent error correction, double error detection, error control codes, fast codes, reliability.

I. INTRODUCTION

As technology scaling increases, the information stored in key elements of a computer system, such as registers and memories, may be perturbed by different physical mechanisms [1]–[3]. Traditionally, error control codes (ECCs) have been extensively employed in computers as a very efficient method to protect information against errors [4]. The design of ECCs is in continuous evolution, adapting their coverage to new design needs and error conditions.

However, when using ECCs to increase computers reliability, the protected circuits also increment their delay (due to data encoding and decoding processes), silicon area occupied and power consumption (both caused by the additional interconnection lines and/or storage needed, as well as by the encoder and decoder circuits).

Hence, the challenge when designing ECCs is to reduce this overhead. According to the requirements, the design of new ECCs tries to provide a good balance between error coverage, redundancy, and efficiency of their encoding and decoding circuits in terms of area, delay and power consumption.

For example, the main objective of an ECC designed to protect the memory is to reduce, for a given coverage, the redundancy. That is, the number of (redundant) bits added by the ECC [5], [6]. As they are added to each word stored in memory, its minimization is a key criterion in the ECC design.

Nevertheless, redundancy is not so important for all computer components. For instance, the register file is an integral element of any microprocessor architecture. Although its overall area is small, this is one of the most frequently accessed components. Corrupted data in registers can quickly spread to other elements of the system, due to their high access rate. If protected by using an ECC, the temporal overhead of the register file should be reduced as much as possible, as this element is in the critical path of the processor pipeline. An excessive delay introduced by encoding and decoding operations might cause the register file to be a

The associate editor coordinating the review of this manuscript and approving it for publication was Luca Cassano.

bottleneck, requiring a longer clock cycle and resulting in a reduction on the working frequency. In the same way, the clock cycle may also be affected when adding ECCs to other registers, like inter-stage latches in pipelined processors.

Current technology scaling processes enable manufacturing high-speed (faster) and high-density (smaller) processors, but this makes registers to be much more sensitive to errors. The problem becomes more important as the voltage level supply used to operate the registers decreases [7]. In this case, the memory cell critical charge and the energy needed to provoke a single-event upset (SEU) in storage is reduced [8]. As shown by different experiments, in addition to traditional single-cell upsets (SCUs), this energy reduction can provoke multiple-cell upsets (MCUs), i.e. simultaneous errors in more than one cell induced by a single particle hit [2], [9], [10]. Usually, they occur in the same word and, very likely, in adjacent bits [11].

As deduced from previous paragraphs, protecting registers against multiple adjacent errors seems of upmost importance [6]. This challenge increases as soft error rate does, as predicted in [12]. In this way, the mechanisms for register protection should be as fast as possible [13]. Different approaches are summarized in [14] and [15]. Other remarkable alternatives are the use of interleaving of simple codes [4] (employed in this paper for comparison) and the use of hardened memory cells [16]. Although some authors conclude that ECCs are a less interesting option, modern processors for servers use ECCs in registers for fault tolerance (e.g. Intel® Xeon® Processor E7 Family [17]).

This paper focuses on low-delay, multiple adjacent error correction codes. Recently introduced, Ultrafast error control codes are a family of ECCs with very low encoding/decoding latencies. Moreover, the logic depth of the circuits does not depend on the data word length. Ultrafast codes with single error correction (SEC), single error correction-double error detection (SEC-DED) and single error correction-double adjacent error correction (SEC-DAEC) capabilities were introduced in [14] and [15]. In this work, the error coverage is still hardened, combining double error detection with multiple adjacent error correction (xAEC), obtaining new Ultrafast SEC-xAEC-DED codes. These codes have been obtained applying the Flexible Unequal Error Control (FUEC) methodology, developed and described in [18]. The delays of the new codes mainly depend on the error coverage, although a small dependency on the data word length may appear due to implementation details, as explained later.

Ultrafast codes have been implemented and synthesized in order to validate the error coverage and to measure the delay, area and power overheads. They have also been compared to other state-of-the-art fast codes, obtaining interesting results.

Therefore, the main novelty presented in this paper is the combination of double error detection with multiple adjacent error correction, achieving new Ultrafast SEC-xAEC-DED codes. Furthermore, we have complemented the comparisons performed in [14] and [15]. In this work, we have implemented different designs in VHDL hardware description language, and we have employed CMOS standard cell synthesis to evaluate and compare them.

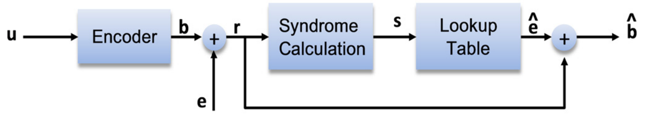

FIGURE 1. Encoding, channel crossing and decoding processes.

Although Ultrafast codes have been presented as a possible solution for register protection, it is remarkable that their usefulness is not restricted to this case. If the required redundancy is affordable, Ultrafast codes offer fast encoding and decoding processes with a moderate area and power overhead. For instance, they can also be useful to protect high-speed memories or caches.

This paper is organized as follows. Section II introduces previous works and basic concepts about error correction codes. The methodology used to build the proposed codes is described in Section III. Section IV includes an evaluation of the codes and a comparison with previous proposals. Finally, Section V presents conclusions and ideas for future work.

II. RELATED WORK

A. BASICS ON ERROR CONTROL CODING

An (n, k) binary ECC encodes a k-bit input word in an n-bit output word [19]. The input word $\mathbf{u} = (u_0, u_1, \ldots, u_{k-1})$ is a k-bit vector that represents the original data. The code word $\mathbf{b} = (b_0, b_1, \ldots, b_{n-1})$ is an n-bit vector, where the $(n-k)$ added bits are called parity, code or redundant bits. $\mathbf{b}$ is transmitted through an unreliable channel that 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, then $e_i = 0$; otherwise, $e_i = 1$. Therefore, $\mathbf{r}$ can be interpreted as $\mathbf{r} = \mathbf{b} \oplus \mathbf{e}$. Fig. 1 synthesizes this encoding, channel crossing and decoding processes.

The parity check matrix $\mathbf{H}_{(n-k)\times n}$ of a linear code defines the code [4]. For the encoding process, $\mathbf{b}$ must meet the requirement $\mathbf{H} \cdot \mathbf{b}^T = \mathbf{0}$. For syndrome decoding, 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 syndrome $\mathbf{s}$ for each correctable error vector $\mathbf{e}$. Syndrome decoding is done through a lookup table that relates each $\mathbf{s}$ to the decoded error vector $\hat{\mathbf{e}}$. If $\mathbf{s} = \mathbf{0}$, we can assume that $\hat{\mathbf{e}} = \mathbf{0}$ and $\mathbf{r}$ is correct. Otherwise, an error has occurred. The decoded code word $\hat{\mathbf{b}}$ is calculated as $\hat{\mathbf{b}} = \mathbf{r} \otimes \hat{\mathbf{e}}$. From $\hat{\mathbf{b}}$, it is easy to obtain $\hat{\mathbf{u}}$ by discarding the parity bits. If the fault hypothesis used to design the ECC is consistent with the behavior of the channel, $\hat{\mathbf{u}}$ and $\mathbf{u}$ must be equal with a very high probability.

Referred to a binary word, the term Hamming weight w denotes the number of ones in that word. As explained later, the Hamming weights of rows and columns of the parity check matrix determine the properties of a code.

Hamming Single Error Correction (SEC) codes [5] can correct an erroneous bit with simple and fast encoding and decoding operations, and the lowest redundancy. An example of implementation for these codes can be found in [14].

Extended Hamming codes are able to correct single errors and detect double errors. They need an additional parity bit (calculated as the even parity for the whole encoded word) to achieve the double error detection. Additional explanation, with an example of implementation, can be found in [15].

Hsiao SEC-DED codes [20] are an optimized version of Extended Hamming codes. They are optimal minimum odd-weight-column codes, i.e. all columns of the parity check matrix have an odd number of ones, allowing the DED coverage. The detection logic is simplified, achieving lower delay, silicon area and power consumption than conventional Hamming SEC-DED codes. We will compare them to Ultrafast codes in Section IV.

Increasing the error coverage frequently makes ECCs more complex and slower. For example, correction of multiple errors in adjacent bits is achieved by the codes presented in [6] (up to three) and in [21] (up to four). Multiple random error correction is achieved by well-known BCH codes [4]. All these codes are designed to reduce the redundancy, but at the cost of more complex and slower decoders.

B. FASTER ERROR CONTROL CODES

Hamming or Hsiao codes have been employed to protect registers and memories against single and double errors. However, as working frequency of VLSI systems increases, reducing the delays introduced by the encoding and decoding circuits becomes of paramount importance.

Different approaches can be found in the literature. The terms "fast", "high speed", "low delay", etc. are frequently associated to different ECCs. Anyway, what is considered "fast" depends on each application.

For instance, a method to reduce the timing impact of an ECC for multilevel flash memories is described in [22]. In multilevel flash memories, each memory cell stores more than one bit. Thus, from a digital viewpoint, each cell stores a multibit symbol, and a faulty cell may result in several erroneous bits (belonging to the same symbol). The proposal is compared to other classic approaches, and the results show a reduction in the temporal overhead. The strategy is to increase the redundancy to reduce both the Hamming weight of the parity check matrix and the weight of the heaviest row. Nevertheless, non-binary symbol codes are commonly more complex than binary codes.

Fast decoding using binary ECCs is proposed in [23], but only for a subset of critical bits. That is, the original data includes "standard" bits and a small number of "important" bits. The proposed ECCs correct single errors and they can decode faster the "important" bits. The strategy, in this case, is based on correcting the "important" bits using only a subset of the parity bits. Recently, these codes have been improved to correct adjacent errors [24].

Fast decoding for the whole word is the objective of Orthogonal Latin Square (OLS) codes [25]. They are one-step majority logic decodable codes. "One-step" means that the decoding is performed in a combinational circuit (without iterative steps). This circuit implements a voter of several check bits to correct, if necessary, the received bits. The majority voter constitutes a simple and fast corrector. These codes correct random errors, at the cost of a high redundancy and additional overhead. In addition, these codes only exist for a few word lengths.

Correction of adjacent errors in a simple and efficient way is achieved by the codes presented in [26]. They combine multiple error detection with vertical parity in a two-dimensional layout. Due to their fast decoding, these codes will be compared to our proposal in Section IV.

Finally, Low Delay (LD) codes proposed in [27] and [28] can be applied to CPU registers protection. They share objectives with our proposal. Due to the special interest for our work, they are explained in detail in Section II.C.

A problem of all the aforementioned codes is that the logic depth of the encoder and decoder circuits scales with the word length. Hence, their latency will grow with longer words. As explained later, one of the main advantages of our proposal is that the logic depth introduced by the encoder and decoder circuits do not depend on the word length.

C. LOW DELAY CODES

Low Delay (LD) SEC and SEC-DED codes [27] reduce the time required to correct 1-bit errors when only the correction of data bits is needed. These codes take advantage of the minor interest of correcting parity bits in registers, as the information stored is not rewritten in the same register once it has been read, and the input data come from other processor elements.

The idea behind LD codes is minimizing the number of 1s in each column and in each row. By reducing the number of 1s in the columns, the decoding logic (i.e. the implementation of the lookup table) can be simplified if all the columns for data bits have the same Hamming weight w (more details can be found in [27]). As the columns for parity bits have $w = 1$, the columns for data bits will have $w = 2$ if only single error correction is required. If additional double error detection is desired, the columns for data bits will have $w = 3$.

On the other hand, the delay when computing the parity bits in the encoder and the syndrome bits in the decoder can be decreased by reducing the number of 1s in the rows.

Summarizing, the main characteristics of LD codes are:

- Encoder and decoder circuits are simpler than equivalent Hamming codes, presenting equal or lower logic depths. This reduces the delay.
- LD codes slightly increase the redundancy over traditional SEC and SEC-DED codes.
- LD codes only correct errors in data bits.

- The logic depth of the encoder and decoder circuits scales with the word length, as stated above.

Other Low Delay codes, with single and double adjacent error correction properties, were presented in [28]. These codes take advantage of the simplification of logic functions when implementing the lookup table, among other techniques.

D. PREVIOUS WORKS ON ULTRAFAST CODES

Ultrafast SEC codes were firstly introduced in [14]. We designed them assuming that redundancy is not the main concern, and focusing on reducing the temporal overhead. Although these codes present a higher redundancy than equivalent Hamming codes, the delay introduced is very short. The requirements to build their parity check matrices are:

1) *Each column must be different and nonzero*. It allows the correction of single errors.
2) *Each column assigned to code bits must have* $w = 1$. It allows easy encoding operations.
3) *Each column assigned to data bits must have* $w = 2$. If the correction of parity bits is not required, it allows simpler error location in the decoder circuit [27].
4) *Each row must have* $w = 3$. It allows decoder circuits whose logic depth does not scale with the word length.

Matrices, circuits and more information can be found in [14]. A similar approach was independently presented in [29]. However, these approaches do not allow the double error detection. In order to ease this coverage, Ultrafast codes were reformulated in [15].

The new formulation allowed the design of Ultrafast SEC-DED codes [15]. Their parity check matrices are generated using these requirements:

1) *Each column must be different and nonzero.*
2) *Each column assigned to code bits must have* $w = 1$.
3) *Each column assigned to data bits must have* $w = 3$.
4) *Each row must have* $w = 4$.

Conditions 2 and 3 determine the new coverage: all the columns of the parity check matrix have odd weight. Therefore, all syndromes for single errors have odd weight, whereas all syndromes for double errors have even weight. This allows the detection of 2-bit errors. Condition 4, as stated above, allows decoder circuits whose logic depth does not scale with the word length.

Examples, with their parity check matrices, circuits and detailed explanation and information can be found in [15].

As adjacent errors are becoming more and more frequent, we also presented Ultrafast SEC-DAEC codes in [15]. They can correct single and double adjacent errors. To add this coverage, a new requirement must be considered:

- All possible double adjacent errors must produce a different syndrome.

As all columns have odd Hamming weight, single errors produce an odd weight syndrome and double adjacent errors generate an even weight syndrome. If all these syndromes are different, double adjacent errors can be corrected.

An example, with its parity check matrix, is detailed in [15]. The complete process to design the decoder circuit, where the simplification of logic functions is essential to achieve the 4-gate delay, is also described. As explained in Section III, this methodology is employed again to get low decoding latencies.

III. ULTRAFAST ERROR CONTROL CODES

A. DELAY INDEPENDENT OF THE WORD LENGTH

When redundancy is not the main concern, and reducing the temporal overhead is the main objective in ECCs construction (e.g. register protection), designers can employ different techniques, mainly based on minimizing the number of ones in rows and columns of the parity check matrix. With these premises in mind, Ultrafast codes presented in Section II.D were designed for fast encoding and decoding operations. Although these codes present a higher redundancy than equivalent codes, the delay introduced is very short.

Let us consider Ultrafast codes as formulated in [15]. These codes are well suited for high-speed operation because: i) encoders are 2-gate-delay circuits (assuming 2-input XOR gates); ii) decoders are 4-gate-delay circuits; and iii) these delays do not depend on the word length. As stated in Section II.A, the decoding process has two steps: syndrome computation, and error(s) location and correction. The delay introduced mainly depends on:

- The Hamming weight w of the heaviest row of the parity check matrix $\mathbf{H}$, for the first part.
- The complexity of the code, for the second part. It is related to the weights of the columns of $\mathbf{H}$, mainly a consequence of its error coverage.

The weights of the rows in $\mathbf{H}$ scale with the word length k for most of the codes. Hence, they have length-dependent delays. Conversely, Ultrafast codes keep constant the weights of the rows, independently of k. This can be achieved by adding parity bits, which is their main concern. In fact, for these codes, $n - k = k$, that is, the number of parity bits is the same as the number of data bits (100% redundancy).

As Ultrafast codes have $n - k = k$, $n = 2k$. Therefore, they have a $k \times 2k$ parity check matrix. By definition, the k columns for data bits must have $w = 3$, and the k columns for parity bits must have $w = 1$. For a given value of k, a parity check matrix will have $w = 3k + k = 4k$ (data plus parity columns). Thus, a balanced matrix will have $4k / k = 4$ ones on each row, independently of the value of k. Therefore, the encoding, as well as the first part of the decoding logic, has always the same logic depth, independently of the word length. In a hardware implementation, minimum variations may appear on their delays due to wiring, parasitic capacitances and inductances, and other implementation details. It applies to the previous Ultrafast codes published in [15] and to the new Ultrafast SEC-xAEC-DED codes presented in this work.

In addition, when only single error correction is required, we can get the lowest decoding latency if the decoder only corrects errors in data bits [27]. In this case, the second part of the decoding logic only depends on the weights of the columns for data bits of **H**. As $w = 3$ by definition, this part of the decoding process is also independent of the word length.

But when multiple error correction is required, different syndromes may indicate an error in the same bit. Even more, these syndromes may have different Hamming weights and, therefore, the technique proposed in [27] cannot be employed. In these cases, the simplification of logic equations helps to reduce the complexity of the decoder. In fact, the second part of the decoding logic, i.e. the implementation of the lookup table, is a truth table of a group of logic functions where the inputs are the syndrome bits, and the outputs are the bits of the estimated error vector. This technique was employed in [15] to obtain the Ultrafast SEC-DAEC decoder with 4-gate delay, and it is employed in this work as well.

For simplification, it is important to get as many "don't care" terms as possible. Due to their high redundancy, Ultrafast codes commonly have a high quantity of free syndromes that can be treated as "don't care" terms. In addition, Ultrafast codes have a low Hamming weight of their parity check matrices ($2n$ for all Ultrafast codes) and a low average weight for their columns (2 ones per column). All these characteristics allow better simplifications, compared to other ECCs, where this ratio is frequently larger and dependent of k. Later, we deepen into this methodology, but the results show that the decoding delay mainly depends on the error coverage.

New Ultrafast SEC-xAEC-DED codes are presented next. As an example, codes for 8-bit data words have been designed, but longer word lengths can be easily achieved, as discussed in Section III.C. Some comparisons are shown in Section IV.

B. ULTRAFAST SEC-xAEC-DED CODES

The main novelty presented in this work is the enhancement of Ultrafast codes to correct multiple adjacent errors, and to detect double (non-adjacent) errors. It is done maintaining the redundancy and low delays (proportional to the error coverage), with reasonable increments in area and power overheads.

The requirements for these Ultrafast codes are:

1) *Each column must be different and nonzero.*
2) *Each column assigned to code bits must have $w = 1$.*
3) *Each column assigned to data bits must have $w = 3$.*
4) *Each row must have $w = 4$.*
5) *All correctable errors must have different syndromes.*
6) *All detectable errors must have a syndrome that is different from all syndromes reserved for correction.*

The explanation for requirements 1 to 4 is the same given in Section II.D. Searching a matrix that achieves requirements 5 and III-C may result very complex. We have used the Flexible Unequal Error Control (FUEC) methodology, presented in [18]. Although a detailed explanation of the methodology is out of the scope of this paper, it is briefly summarized in the following.

After determining the values of n and k for the code to be designed, error patterns to be corrected and detected must be selected. Then, the parity check matrix **H** that satisfies (2) and (3) is searched. $\boldsymbol{E}_+$ represents the set of error vectors to be corrected, and $\boldsymbol{E}_\Delta$ is the set of error vectors to be detected.

$$\mathbf{H} \cdot \mathbf{e}_i^T \neq \mathbf{H} \cdot \mathbf{e}_j^T; \quad \forall \mathbf{e}_i,\ \mathbf{e}_j \in \boldsymbol{E}_+ | \mathbf{e}_i \neq \mathbf{e}_j \tag{2}$$

$$\mathbf{H} \cdot \mathbf{e}_i^T \neq \mathbf{H} \cdot \mathbf{e}_j^T; \quad \forall \mathbf{e}_i \in E_\Delta,\ \mathbf{e}_j \in \boldsymbol{E}_+ \tag{3}$$

That is, each correctable error must have a different syndrome (2), and each detectable error must have a syndrome that is different from all the syndromes generated by correctable errors (3).

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. There are $2^{n-k}-1$ combinations for each column. A big amount of these combinations is discarded, as the algorithm is configured to employ columns with Hamming weight 1 or 3 only, to meet requirements 2 and 3 for Ultrafast codes. In addition, we have included requirement 4 to discard partial matrices that have rows with Hamming weight $w > 4$.

As an example, using the FUEC methodology we have found a (16, 8) Ultrafast code which is SEC-5AEC-DED. That is, it can correct single errors and up to five adjacent errors, and it can detect double non-adjacent errors. Let **H8** be 8×16 parity check matrix for this code. It can be represented as:

$$\mathbf{H8} = \begin{bmatrix} \mathbf{I}_{8\times8} & \mathbf{A}_{8\times8} \end{bmatrix} = \begin{bmatrix} 10000000 & 10100010 \\ 01000000 & 01000101 \\ 00100000 & 10101000 \\ 00010000 & 01010100 \\ 00001000 & 10001010 \\ 00000100 & 01010001 \\ 00000010 & 00101010 \\ 00000001 & 00010101 \end{bmatrix} \tag{4}$$

where **I** is the identity matrix (columns for parity bits), and **A** is the second half of the matrix, with the columns for data bits.

It is noticeable that the same parity check matrix can be used for lower error coverage. How to design a SEC-DAEC-DED decoder (or 3AEC/4AEC versions) using the same matrix is explained in Section III.D. Results are also evaluated in Section IV. As stated later, lower error coverage results in higher simplifications and, hence, in faster decoders.

The encoding and syndrome equations can be obtained from the parity check matrix. The encoding equations for this code are:

As it can be observed, the parity check matrix relates each parity bit ($b_0 \ldots b_7$) with the data bits ($u_0 \ldots u_7$) required for its calculation. Each parity bit is calculated XORing three

b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}	***Encoding equations***
								u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	
1	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	$b_0 = u_0 \oplus u_2 \oplus u_6$
0	**1**	0	0	0	0	0	0	0	1	0	0	0	1	0	1	$b_1 = u_1 \oplus u_5 \oplus u_7$
0	0	**1**	0	0	0	0	0	1	0	1	0	1	0	0	0	$b_2 = u_0 \oplus u_2 \oplus u_4$
0	0	0	**1**	0	0	0	0	0	1	0	1	0	1	0	0	$b_3 = u_1 \oplus u_3 \oplus u_5$
0	0	0	0	**1**	0	0	0	1	0	0	0	1	0	1	0	$b_4 = u_0 \oplus u_4 \oplus u_6$
0	0	0	0	0	**1**	0	0	0	1	0	1	0	0	0	1	$b_5 = u_1 \oplus u_3 \oplus u_7$
0	0	0	0	0	0	**1**	0	0	0	1	0	1	0	1	0	$b_6 = u_2 \oplus u_4 \oplus u_6$
0	0	0	0	0	0	0	**1**	0	0	0	1	0	1	0	1	$b_7 = u_3 \oplus u_5 \oplus u_7$

data bits. Hence, the encoding implementation can be a circuit whose logic depth is two (assuming the use of 2-input XOR gates). As stated above, it does not depend on the word length.

The expressions for syndrome calculation are:

r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}	r_{12}	r_{13}	r_{14}	r_{15}	***Syndrome bits***
								u_0	u_1	u_2	u_3	u_4	u_5	u_6	u_7	
1	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	$s_0=r_0\oplus r_8\oplus r_{10}\oplus r_{14}$
0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	1	$s_1=r_1\oplus r_9\oplus r_{13}\oplus r_{15}$
0	0	1	0	0	0	0	0	1	0	1	0	1	0	0	0	$s_2=r_2\oplus r_8\oplus r_{10}\oplus r_{12}$
0	0	0	1	0	0	0	0	0	1	0	1	0	1	0	0	$s_3=r_3\oplus r_9\oplus r_{11}\oplus r_{13}$
0	0	0	0	1	0	0	0	1	0	0	0	1	0	1	0	$s_4=r_4\oplus r_8\oplus r_{12}\oplus r_{14}$
0	0	0	0	0	1	0	0	0	1	0	1	0	0	0	1	$s_5=r_5\oplus r_9\oplus r_{11}\oplus r_{15}$
0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	0	$s_6=r_6\oplus r_{10}\oplus r_{12}\oplus r_1$
0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	$s_7=r_7\oplus r_{11}\oplus r_{13}\oplus r_1$

Again, the syndrome calculation can be implemented in a circuit whose logic depth is two, independently of the word length. Following the process shown in Fig. 1, next step is the implementation of the lookup table, i.e. the computation of the $\hat{\mathbf{e}}$ vector as a function of the syndrome. If the design of the parity check matrix is correct, each correctable error will have a different syndrome. This step is described, for example, in [15]. The non-recoverable error (NRE) signal indicates that an error is detected but cannot be corrected. It is calculated in the same way. As the full lookup table is too long, some sample lines are included here.

$s_7 s_6 s_5 s_4 s_3 s_2 s_1 s_0$	$\hat{e}_{15}\hat{e}_{14}\hat{e}_{13}\hat{e}_{12}\hat{e}_{11}\hat{e}_{10}\hat{e}_9\hat{e}_8\hat{e}_7\hat{e}_6\hat{e}_5\hat{e}_4\hat{e}_3\hat{e}_2\hat{e}_1\hat{e}_0$	***NRE***	**Error in …**
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	No error
0 0 0 0 0 0 0 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1	0	r_0
0 0 0 0 0 0 1 0	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0	0	r_1
0 0 0 0 0 0 1 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1	0	r_0, r_1
0 0 0 0 0 1 0 0	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0	0	r_2
0 0 0 0 0 1 0 1	? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?	1	Double
0 0 0 0 0 1 1 0	0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0	0	r_1, r_2
0 0 0 0 0 1 1 1	0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1	0	r_0,r_1,r_2
. . .	. . .		. . .
0 1 0 0 0 1 0 1	0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0	0	r_{10}
0 1 0 0 0 1 1 0	? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?	?	Undefined
. . .	. . .		. . .
1 1 1 1 0 0 1 1	1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0	r_{14}, r_{15}
. . .	. . .		. . .
1 1 1 1 0 1 1 0	? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?	1	Double
1 1 1 1 0 1 1 1	? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ?	?	Undefined
. . .	. . .		. . .
1 1 1 1 1 1 1 1	0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0	0	r_6,r_7,r_8,r_9

In this truth table, we can observe 17 logic functions (16 bits from the estimated error vector, and the NRE signal). The input of these functions are the eight syndrome bits. Different cases can be described from the table:

- The no error situation generates the zero syndrome.
- Single errors produce a syndrome that must coincide with the assigned column in the parity check matrix (e.g. 00000010 for r_1 or 01000101 for r_{10}).
- Multiple errors included in the correction coverage generate a syndrome that is the bitwise XOR of the columns assigned to the affected bits (e.g. 00000011 for r_0 and r_1 or 11111111 for r_6, r_7, r_8 and r_9).

All the above conditions maintain the NRE signal inactive, as all of them represent correctable situations. Two additional situations can be found in the truth table for this code:

- Double non-adjacent errors must activate the NRE signal. The syndrome is also the bitwise XOR of the affected columns, but different errors may generate the same syndrome (e.g., 00000101 syndrome is produced by an error in r_0 and r_2, but also by an error in r_{12} and r_{14}). The $\hat{\mathbf{e}}$ vector cannot be determined, as it is not possible to know the positions of the erroneous bits.
- Once assigned all the above situations to the corresponding syndromes, several syndromes may remain unassigned (e.g. 11110111). They represent multiple errors not considered in the code coverage. Nevertheless, as they are very uncommon situations (according to the fault assumptions for which the code has been designed), they can be not considered for NRE activation. Instead, they can be employed for the logic simplification of $\hat{e}_i$ and NRE signals, as detailed in Section III.D.

Finally, the output of the decoder is the result of XORing the received bits with the corresponding estimated error vector: $\hat{\mathbf{b}}_i = r_i \otimes \hat{\mathbf{e}}_i$. $\hat{\mathbf{u}}$ is easily obtained just extracting the corresponding bits from $\hat{\mathbf{b}}$.

C. CODES FOR LONGER DATA WORDS

Applying the FUEC methodology, matrices for codes with different word lengths and error coverages can be found. Nevertheless, as the number of parity bits grows, the process becomes heavier and it requires a lot of computation time. However, Ultrafast codes for long data words can be easily obtained combining matrices for shorter data words. As the common word lengths in computers are powers of two (8, 16, 32, 64…), we can use Ultrafast (16, 8) matrices to generate (32, 16), (64, 32), etc. codes. For example, considering **H8** matrix shown in (4), the 16 × 32 parity check matrix for an Ultrafast (32, 16) code, with the same redundancy and error coverage, can be generated as:

$$\mathbf{H16} = \left[\mathbf{I}_{16\times16} \quad \begin{matrix} \mathbf{A}_{8\times8} & \mathbf{0}_{8\times8} \\ \mathbf{0}_{8\times8} & \mathbf{A}_{8\times8} \end{matrix} \right] \qquad (5)$$

This new code has the same complexity, as the construction is equivalent to having two 16-bit sub-words —bits (0..7, 16..23) and bits (8..15, 24..31)—, each one covered by independent (16, 8) codes. It also maintains the error coverage:

- Single and multiple adjacent errors on each half word are corrected as in the (16, 8) code.
- Adjacent errors affecting both half words become two shorter adjacent errors for the (16, 8) codes.
- Double non-adjacent errors, where each bit belongs to a different half, become single errors for each code.
- Double non-adjacent errors inside a half are detected by the corresponding (16, 8) code.

Even more, we can modify **H16** by means of column permutations to achieve better coverage. Alternating columns of both **H8** matrices, the new **H16'** matrix represents an Ultrafast (32, 16) SEC-10AEC-DED, where c_i **H8** represents the *i*-th column of the **H8** matrix. As it can be observed, a 10-bit adjacent error is treated as two 5-bit adjacent errors, which can be corrected by each original code.

$$\mathbf{H16}' = \left[\begin{array}{cccccc} c_0\,\mathbf{H8} & \mathbf{0}_{8\times1} & c_1\,\mathbf{H8} & \mathbf{0}_{8\times1} & \cdots \\ \mathbf{0}_{8\times1} & c_0\,\mathbf{H8} & \mathbf{0}_{8\times1} & c_1\,\mathbf{H8} & \end{array}\right] \tag{6}$$

Expressions (5) and (6) can be generalized for other word lengths (32, 64, ... bits). For example:

$$\mathbf{H32} = \left[\begin{array}{c|cccc} & \mathbf{A}_{8\times8} & \mathbf{0}_{8\times8} & \mathbf{0}_{8\times8} & \mathbf{0}_{8\times8} \\ \mathbf{I}_{32\times32} & \mathbf{0}_{8\times8} & \mathbf{A}_{8\times8} & \mathbf{0}_{8\times8} & \mathbf{0}_{8\times8} \\ & \mathbf{0}_{8\times8} & \mathbf{0}_{8\times8} & \mathbf{A}_{8\times8} & \mathbf{0}_{8\times8} \\ & \mathbf{0}_{8\times8} & \mathbf{0}_{8\times8} & \mathbf{0}_{8\times8} & \mathbf{A}_{8\times8} \end{array}\right] \tag{7}$$

$$\mathbf{H32}' = \left[\begin{array}{cccccc} c_0\,\mathbf{H8} & \mathbf{0}_{8\times1} & \mathbf{0}_{8\times1} & \mathbf{0}_{8\times1} & c_1\,\mathbf{H8} & \\ \mathbf{0}_{8\times1} & c_0\,\mathbf{H8} & \mathbf{0}_{8\times1} & \mathbf{0}_{8\times1} & \mathbf{0}_{8\times1} & \\ \mathbf{0}_{8\times1} & \mathbf{0}_{8\times1} & c_0\,\mathbf{H8} & \mathbf{0}_{8\times1} & \mathbf{0}_{8\times1} & \cdots \\ \mathbf{0}_{8\times1} & \mathbf{0}_{8\times1} & \mathbf{0}_{8\times1} & c_0\,\mathbf{H8} & \mathbf{0}_{8\times1} & \end{array}\right] \tag{8}$$

D. DECODER IMPLEMENTATION

As stated above, the objective of Ultrafast codes is to achieve encoding and decoding circuits as fast as possible. The "real" performance of a circuit will depend on several factors, such as the implementation technology, the logic depth of the signals or the complexity of the logic equations. Discussion about this matter can be found in [14] and [15].

As described in Section II.A, the encoding operations and the syndrome calculation, the first part of the decoding, can be easily obtained from the parity check matrix. In most ECCs, the complexity of these operations depends on the word length. Conversely, the expressions for Ultrafast codes have low and constant complexity, independently of the word length, as stated in Section III.A.

Anyway, the most complex part of the decoder is the implementation of the lookup table, especially when several syndromes may indicate an error in the same bit (i.e. when the decoder allows multiple error correction). It is necessary to simplify the logic equations required to obtain $\hat{e}_i$ and NRE signals. Simplifying the expressions can reduce delay, area and power consumption of the corrector circuit.

Non-simplified equations can be obtained from the lookup table as sum of minterms or product of maxterms. Ultrafast codes take great advantage of their redundancy to get a high number of free syndromes, which become "don't care" terms for simplification. They can be obtained when the value of the error vector, or the NRE signal, is not important:

- Different errors can cause the syndromes assigned to double errors. As this situation is indicated by the NRE signal, the erroneous bits cannot be determined and these syndromes can be used to simplify the $\hat{e}_i$ signals.
- If the fault hypothesis considered for the design of the ECC is correct, the "undefined" syndromes only appear under very uncommon situations. Hence, the probability of a wrong detection/correction due to a bad estimation of the error vector is negligible. Therefore, the value of the error vector or the NRE signal is not significant, and these syndromes can be employed to simplify them.

Additional simplification, if required, can be obtained by reducing the error coverage of a code. For example, the SEC-5AEC-DED code presented in (4) allows a great error coverage, maybe excessive. If, according to a given fault hypothesis, adjacent errors may affect two adjacent bits at most, we can design a SEC-DAEC-DED decoder using the same parity check matrix. In this case, the lookup table will consider as "undefined" the syndromes assigned to 3- to 5-bit adjacent errors, improving the simplification. In the same way, 3AEC/4AEC decoders can be designed.

Following with the above example, $\hat{e}_{10}$ will be the sum of 15 minterms (of 8 variables) if no simplification and maximum error coverage (SEC-5AEC-DED) is considered:

$$\begin{aligned}\hat{e}_{10} = {} & \bar{s}_7\bar{s}_6\bar{s}_5 s_4 s_3\bar{s}_2\bar{s}_1 s_0 + \bar{s}_7\bar{s}_6 s_5 s_4\bar{s}_3\bar{s}_2 s_1 s_0 \\ & + \bar{s}_7 s_6\bar{s}_5\bar{s}_4\bar{s}_3 s_2\bar{s}_1 s_0 + \bar{s}_7 s_6\bar{s}_5 s_4\bar{s}_3\bar{s}_2 s_1\bar{s}_0 \\ & + \bar{s}_7 s_6 s_5\bar{s}_4\bar{s}_3\bar{s}_2 s_1\bar{s}_0 + \bar{s}_7 s_6 s_5\bar{s}_4 s_3 s_2 s_1 s_0 \\ & + \bar{s}_7 s_6 s_5 s_4 s_3\bar{s}_2 s_1\bar{s}_0 + s_7\bar{s}_6\bar{s}_5\bar{s}_4\bar{s}_3 s_2 s_1\bar{s}_0 \\ & + s_7\bar{s}_6\bar{s}_5 s_4\bar{s}_3\bar{s}_2 s_1 s_0 + s_7\bar{s}_6 s_5 s_4 s_3\bar{s}_2\bar{s}_1 s_0 \\ & + s_7\bar{s}_6 s_5 s_4 s_3\bar{s}_2 s_1\bar{s}_0 + s_7 s_6\bar{s}_5\bar{s}_4\bar{s}_3 s_2 s_1 s_0 \\ & + s_7 s_6\bar{s}_5 s_4\bar{s}_3\bar{s}_2 s_1\bar{s}_0 + s_7 s_6 s_5\bar{s}_4 s_3 s_2\bar{s}_1 s_0 \\ & + s_7 s_6 s_5 s_4 s_3\bar{s}_2 s_1\bar{s}_0\end{aligned}$$

Simplifying for maximum error coverage, we obtain:

$$\begin{aligned}\hat{e}_{10} = {} & s_6\bar{s}_4 s_3 + s_7\bar{s}_5\bar{s}_3 s_1 + \bar{s}_7 s_6 s_2 s_0 + \bar{s}_6 s_4\bar{s}_2 s_0 \\ & + s_6\bar{s}_2 s_1\bar{s}_0 + s_4\bar{s}_2 s_1\bar{s}_0\end{aligned}$$

And simplifying for SEC-DAEC-DED error coverage:

$$\hat{\mathbf{e}}_{10} = \bar{s}_4 s_2 s_0$$

These are all the simplifications for the SEC-DAEC-DED decoder. They implement the lookup table for this code:

$$\begin{array}{llll} \hat{e}_0 = \bar{s}_4\bar{s}_2 s_0 & \hat{e}_1 = \bar{s}_5\bar{s}_3 s_1 & \hat{e}_2 = \bar{s}_4 s_2\bar{s}_0 & \hat{e}_3 = \bar{s}_5 s_3\bar{s}_1 \\ \hat{e}_4 = s_4\bar{s}_2\bar{s}_0 & \hat{e}_5 = s_5\bar{s}_3\bar{s}_1 & \hat{e}_6 = s_6\bar{s}_2\bar{s}_0 & \hat{e}_7 = s_7\bar{s}_3\bar{s}_1 \\ \hat{e}_8 = s_4 s_2 s_0 & \hat{e}_9 = s_5 s_3 s_1 & \hat{e}_{10} = \bar{s}_4 s_2 s_0 & \hat{e}_{11} = s_5 s_3\bar{s}_1 \\ \hat{e}_{12} = s_4 \quad s_2\bar{s}_0 & \hat{e}_{13} = \bar{s}_5 s_3 s_1 & \hat{e}_{14} = s_4\bar{s}_2 s_0 & \hat{e}_{15} = s_5\bar{s}_3 s_1 \end{array}$$

Different matrices may meet the Ultrafast requirements enumerated in Section III.B. They may result in distinct minimizations and decoding delays. Further research is required to determine if it is possible to find matrices with better minimizations than matrix (4).

An additional question that may help to design simpler decoders, is the necessity (or not) of correcting the parity bits. Of course, errors covered by a code must lead to a right decoding, independently of the kind (data or parity) of the bits affected. However, what is the result of a right decoding? Sometimes, only $\hat{\mathbf{u}}$ is required (that is, only data bits; for example, when protecting processor registers, as commented before). Other times it is interesting to obtain $\hat{\mathbf{b}}$ (i.e. data and parity bits; for example, when a scrubbing mechanism is employed in DRAMs). Obviously, correcting the parity bits will require additional silicon area and power consumption, and it may influence the delay of the decoder.

Ultrafast codes could correct parity bits, as all $\hat{e}_i$ can be calculated from the lookup table. Nevertheless, as our objective is to maximize the simplification, only data bits have been corrected in our designs.

All the techniques presented in this section have been applied in the design of Ultrafast circuits. In next section, they will be evaluated and compared to other state-of-the-art codes. For a fair comparison, these techniques have been applied to all codes, when appropriate. For example, all decoders have been implemented for correcting data bits only.

IV. EVALUATION AND COMPARISON

A. PREVIOUS CONSIDERATIONS

In previous works, Ultrafast SEC [14] and Ultrafast SEC-DED [15] codes have been compared to other codes, in terms of number of parity bits, logic depth and number of logic gates. The number of parity bits measures the redundancy. The logic depth of the encoder and decoder circuits is an estimator of the propagation delay introduced by those circuits. The number of logic gates influences the silicon area occupied and the power consumption of the circuits. These measurements allow an approximated estimation of the efficiency of the codes, but it is less accurate than the real synthesis for a given technology.

So, in this paper, the encoder and decoder circuits for all ECCs have been implemented in VHDL hardware description language. Then, using CADENCE software [30], we have carried out a logic synthesis for 45-nm technology by using the NanGate FreePDK45 Open Cell Library [31], [32]. Standard cells are based on SCMOS design rules. Power voltage and temperature conditions are 1.1V and 25 °C, respectively. Logic synthesis allows obtaining better estimation of the overhead induced by different ECCs. Although the main objective is diminishing the propagation delay of the circuits, information about the silicon area and power consumption is also compared.

As stated above, the simplification techniques employed to improve the performance of the circuits have been applied equally to all codes, when appropriate, for a fair comparison. Synthesis data consider all encoding and decoding steps, including the lookup table implementation.

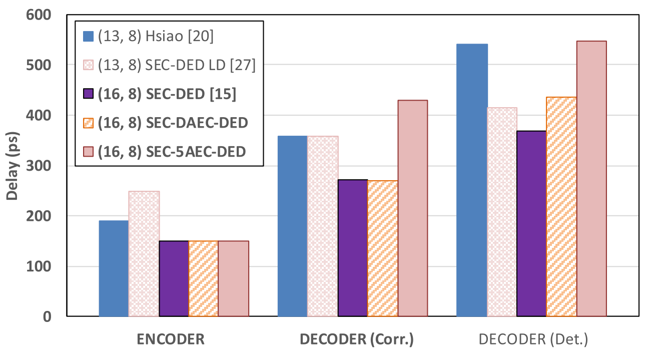

FIGURE 2. Propagation delay for SEC-DED codes.

Prior to the synthesis, all compared codes have been simulated, injecting all possible correctable and detectable errors, and we have verified that the planned coverages have been achieved.

B. COMPARISON OF CODES FOR 8-BIT DATA WORDS

In this section, several codes for 8-bit data words are compared: a Hsiao code [20], a SEC-DED Low Delay code [27] and a SEC-DED Ultrafast code [15]. In addition, the new SEC-xAEC-DED Ultrafast code described in Section III.B is included in two different versions: one with maximum coverage (i.e. SEC-5AEC-DED), and a simplified version with a SEC-DAEC-DED decoder (as described in Section III.D, this technique allows flexibility to adjust the adjacent error coverage to the design requirements).

All the compared codes share the ability of detecting double errors. Although their correction coverage is different, our aim is to compare our proposal with simple and fast codes.

Fig. 2 compares the propagation delays of their circuits. Three different groups of measurements have been obtained: encoder delays, decoder delays (considering only the correction logic), and decoder delays for the detection logic. The correction logic is followed by the actual circuit logic. For example, when protecting processor registers, the correction logic is in the datapath. Hence, it increases directly the circuit delay [27]. Thus, the delay of the correction logic is an important parameter. Error detection is larger in most cases, and it will only be used to signal an unrecoverable error. Therefore, it only must be smaller than the clock cycle.

Regarding the encoder delays, Ultrafast codes achieve the best scores, whereas the Low Delay code obtains the worst result. Hsiao code outperforms Low Delay code due to a better balance in the Hamming weights of the rows in the parity check matrix. Anyway, the delays of both codes scale with the word length, whereas Ultrafast codes maintain their delay almost unchanged for longer data words.

Considering the propagation delay for the correction logic of the decoder circuit, Ultrafast SEC-DED and SEC-DAEC-DED codes obtain the best values. The Ultrafast SEC-DAEC-DED decoder is considerably (over 30%) faster than Hsiao and Low Delay codes, whereas increasing the error correction

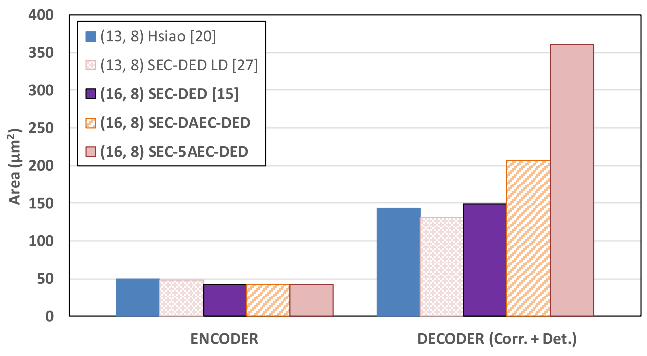

FIGURE 3. Silicon area for SEC-DED codes.

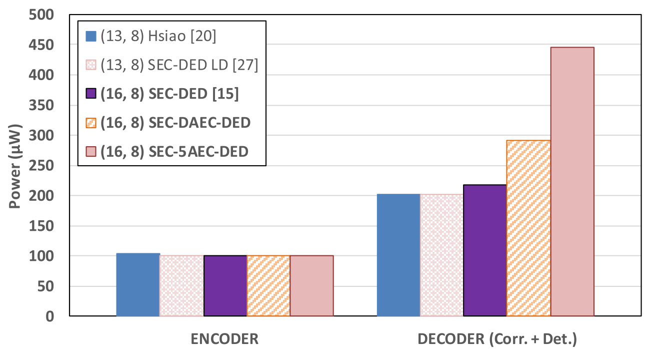

FIGURE 4. Power consumption for SEC-DED codes.

capabilities. The greatest delay is found on the Ultrafast SEC-5AEC-DED. This is an expected result, as these codes are designed for 8-bit data words, and 5 adjacent error correction is a very high coverage. Nevertheless, the delay is in the same order of magnitude, so it could be probably affordable if such error coverage would be required.

Although less critical, as stated above, the propagation delay for the detection signal is also included in the figure. Again, the best score is achieved by the Ultrafast SEC-DED code, and all other values are affordable, as explained before.

Fig. 3 shows the silicon area occupied by the encoder and decoder circuits. Although Ultrafast codes present higher redundancy, the area employed by their encoder circuits is slightly smaller than Hsiao or Low Delay codes. Regarding the decoder circuits, the codes with the same error coverage (Hsiao, Low Delay and Ultrafast SEC-DED) occupy a similar area, with small variances. As the error coverage is improved, the decoder size increases, as expected.

Fig. 4 presents the power consumption of the circuits. The trends are the same as observed for the silicon area: all the encoder circuits have very similar values; there are small differences in the decoders with the same error coverage; and the power consumption grows as the error coverage augments.

To conclude this comparison, it is noticeable that Ultrafast SEC-DED code gets the best scores in almost all ranks. When it does not have the best value, the difference with it is small. Ultrafast SEC-DAEC-DED code achieves very good results. If the redundancy is affordable and the error coverage meets the fault hypothesis, these codes can be a good choice. If additional error coverage is required, Ultrafast codes can correct longer adjacent errors. As an example, the SEC-5AEC-DED code compared here shows the upper bounds for the different values. Depending on the application and design requirements (error coverage, speed, etc.) this code (or 3AEC/ 4AEC versions) may result interesting.

C. CODES FOR LONGER DATA WORDS

The previous results show the good performance of Ultrafast codes when $k = 8$. Nevertheless, from the delay viewpoint, their main advantage becomes more evident as k raises. In this section, we study the evolution of propagation delay, silicon area and power consumption, for different codes and typical values of k in registers (8, 16, 32, and 64 bits). In these comparisons, we have analyzed different high-speed codes:

- SEC-DAEC-DED and SEC-5AEC-DED Ultrafast: the (16, 8) compared previously, and (32, 16), (64, 32) and (128, 64) codes obtained as explained in Section III.C.
- SEC-DED Low Delay [27]: in addition to the (13, 8) used in previous comparisons, we have also included here the (22, 16), (39, 32) and (73, 64) codes.
- SEC-DAEC Low Delay: the codes published in [28]. It is remarkable that these codes do not detect double errors. Anyway, their low delay decoders make them an interesting option for comparison.
- DEC Orthogonal Latin Square [25]: the (20, 8) code described in [33], the (32, 16) code shown in [34], the (55, 32) code presented in [35], and the (96, 64) code defined in [36].
- SEC-4AEC Two-Dimensional [26]: (20, 8), (32, 16), (56, 32) and (104, 64) codes. They achieve very fast decoding due to a simple but efficient layout.

In addition, we have also obtained synthesis data for (13, 8), (22, 16), (39, 32) and (72, 64) Hsiao codes. They are not included in the figures, as their results are very similar to the SEC-DED Low Delay codes.

Fig. 5 plots the delay introduced by the correction logic of the decoder circuits. As expected, the decoding delay for all non-Ultrafast codes raises as k does. This is mainly due to the increasing number of bits to be considered during syndrome computation, and the difficulty for obtaining good simplifications. The worst results for higher values of k can be observed for the SEC-DAEC Low Delay codes.

Regarding Ultrafast codes, the SEC-DAEC-DED version offers the best results when $k = 64$, which is 160% faster than the worst code. In addition, it has a constant decoding delay. This is an expected result, although small differences may appear depending on the implementation, due to logical synthesis details, such as the types of logic gates employed or their number of inputs. For example, although the logic expressions have the same complexity for the different SEC-5AEC-DED Ultrafast decoders, small variations arise, as it can be observed in Fig. 5. This slight dependency on the word length is due to logical implementation details, not to a higher

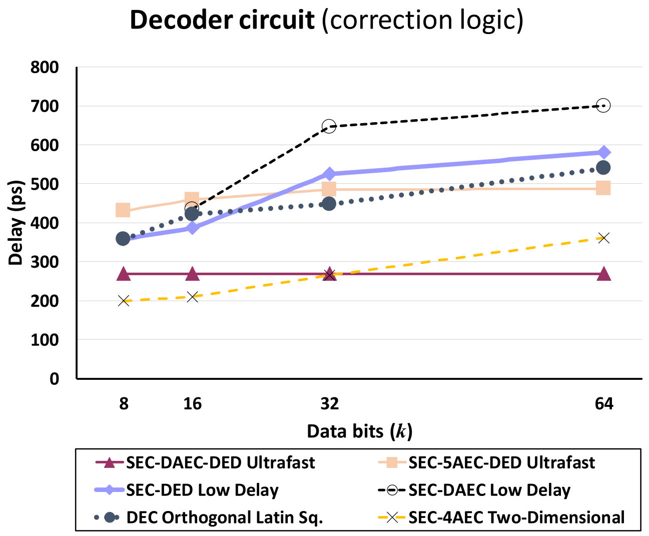

FIGURE 5. Propagation delay evolution for different values of k.

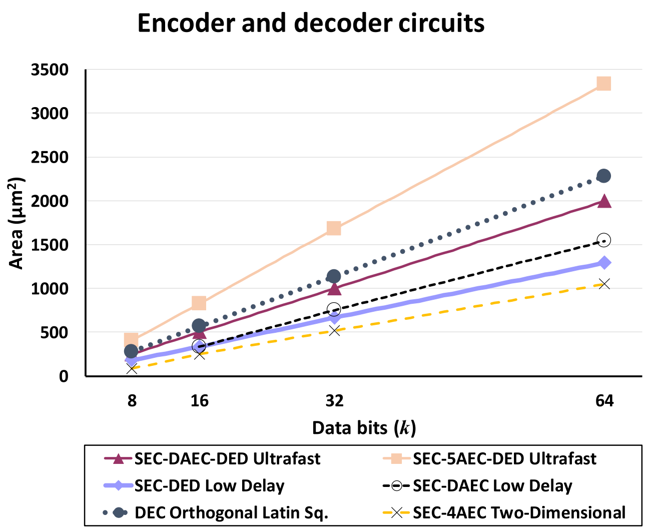

FIGURE 6. Silicon area evolution for different values of k.

logic complexity. Anyway, the increment has a significantly smoother slope, with a negligible difference between the 32- and 64-bit data word versions. Although this code has the worst delay when $k = 8$, the $k = 64$ version is only slower than the Two-Dimensional and the Ultrafast SEC-DAEC-DED codes, while having higher error coverage than both of them.

Notice the low latencies achieved by the Two-Dimensional codes. The 8-bit version is the fastest code, at the cost of a higher redundancy than Ultrafast codes. Faster results are also obtained by the 16-bit code. Anyway, these codes do not detect double errors, and the latency raises as k does, whereas it does not in Ultrafast codes.

Fig. 6 shows the silicon area employed by the encoder and decoder circuits for the different codes. In this case, the value compared is the sum of the area of encoders and decoders. In general, codes with more error coverage get

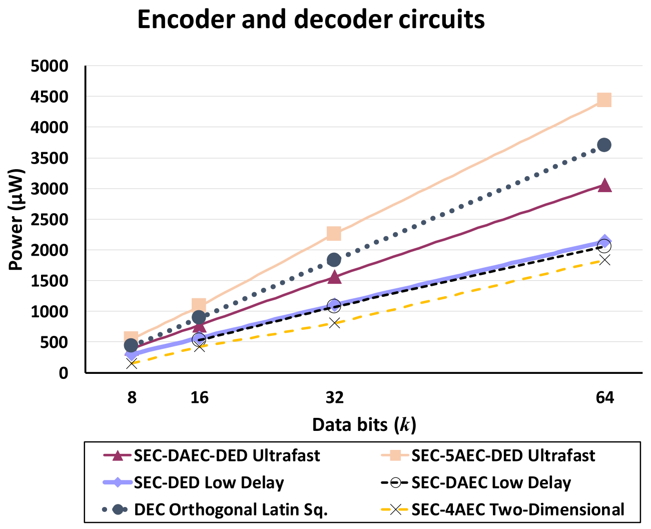

FIGURE 7. Power consumption evolution for different values of k.

higher values. When k raises the area occupied augments proportionally. The lowest values correspond to the SEC-4AEC Two-Dimensional codes. Obviously, as they do not implement double error detection, their decoders are very simple. On the other side, the highest values are obtained for the SEC-5AEC-DED Ultrafast codes. The SEC-DAEC-DED Ultrafast codes reach affordable values, lower than the DEC OLS codes, eventually employed for the register file protection [37].

Fig. 7 presents the evolution of the power consumption for the different data word lengths. Again, the sum of the power employed for both encoder and decoder circuits is considered. The trends are similar to those observed for the silicon area in Fig. 6. Again, the lowest consumption is achieved by the SEC-4AEC Two-Dimensional codes (remember that they lack of the detection circuitry), and the highest values belong to Ultrafast SEC-5AEC-DED codes. The Ultrafast SEC-DAEC-DED codes are less power consumers than the OLS codes.

To conclude this comparison, we must consider the different error coverages that each code can offer. OLS codes are the only compared codes that can correct all double errors. LD SEC-DED codes correct single errors and detect double errors. LD SEC-DAEC codes correct single and double adjacent errors. Finally, the SEC-4AEC Two-Dimensional codes correct single and up to 4-bit burst errors.

In the case of Ultrafast codes, if we consider the code construction proposed in (6) and (8), the delays will be the same (or almost the same) whereas the error coverage augments: due to the alternated columns in the parity check matrices, the (16, 8) SEC-DAEC-DED decoder becomes a (32, 16) SEC-4AEC-DED, a (64, 32) SEC-8AEC-DED, and a (128, 64) SEC-16AEC-DED. Likewise, the (16, 8) SEC-5AEC-DED can correct up to 40 adjacent errors when $k = 64$. This causes the high area required and power consumption.

As stated above, depending on the error coverage requirements and the constraints about delay, area or power consumption, designers may use different Ultrafast codes. 3AEC/4AEC versions can also be employed to flexibly adjust the desired error coverage.

D. CASE EXAMPLE: A REGISTER FILE

The previous comparisons show the validity of the codes proposed in this paper: Ultrafast codes achieve the lowest delays with moderate area usage and power consumption, probably affordable by several applications. Nevertheless, data shown only consider the overhead generated by the encoder and decoder circuits. Is the additional memory required to store the redundant bits a problem? As commented previously, it could be a problem for high storage memories, but it is less important for small memory structures.

To validate this sentence, we have implemented and synthesized a MIPS-like register file. It has 32 registers to store 32-bit data words. It has two read ports and one write port, allowing two reads and one store in the same clock cycle. We have compared a Triple Modular Redundancy (TMR) version and four ECC-based schemes. The first scheme employs the (64, 32) Ultrafast SEC-8AEC-DED code obtained using matrix (4), simplifying the decoder to obtain the SEC-DAEC-DED version, and combined with the construction proposed in (8). The second scheme interleaves four (13, 8) Hsiao SEC-DED codes, allowing the correction of four adjacent errors. The third code employed is the (40, 32) SEC-4AEC code proposed in [21]; and the last scheme is the (56, 32) SEC-4AEC Two-Dimensional code [26]. These last codes correct up to four adjacent errors, but they do not detect all double errors.

The TMR design allows fast write operations, as it has to store the same information three times in parallel. It also has fast read operations, as it only requires 3-bit voters working in parallel. The error coverage is high, as it corrects all possible errors if they occur only in one of the copies of a register. The main drawback of this design is the high redundancy required (200%), as each data bit needs two additional copies.

The Ultrafast-protected scheme has 100% redundancy. Ultrafast SEC-DAEC-DED code is the fastest ECC with double error detection, in our best knowledge (at least, it is the fastest ECC from those compared in Sections IV.B and IV.C). Using matrix (8), this code becomes SEC-8AEC-DED with the same delays, and a good error coverage.

The Hsiao-interleaved design has 62.5% redundancy. Interleaving simple codes is frequently employed for multiple adjacent error correction. Interleaving a SEC-DED code using distance four, the result is a SEC-4AEC-DED coverage.

The (40, 32) SEC-4AEC code proposed in [21] has 25% redundancy, the lowest value of all proposals. The (56, 32) Two-Dimensional code [26] has 75% redundancy.

Table 1 shows the results obtained. The fastest solution is TMR (30% faster than Ultrafast design), but at the cost of a high power consumption (39% higher) and silicon area usage (44% higher). All other proposals are slower.

The best area and power records belong to the (40, 32) SEC-4AEC scheme, mainly due to its low redundancy.

TABLE 1. **Synthesis results for different register file implementations.**

Scheme	Delay (ps)	Area (μm^2)	Power (mW)
TMR	525	40711	2413
Ultrafast	683	28276	1742
(56,32) Two-Dimensional	704	24067	1595
Hsiao-interleaved	829	22811	1643
(40,32) SEC-4AEC	1385	19169	1472

TABLE 2. **Different techniques applied by ECCs to reduce the delay.**

ECC	Methods to reduce the delay	Improved component
Hsiao SEC-DED	Reduce the number of 1s in **H** heaviest row	Encoder, Syndrome calculation
Low Delay SEC-DED	Reduce and fix the number of 1s in **H** columns	Corrector (final phase of decoding)
Low Delay SEC-DAEC	Simplification of error correction functions	Corrector (final phase of decoding)
Orthogonal Latin Square DEC	Majority logic decoding	Corrector (final phase of decoding)
Two-Dimensional xAEC	Horizontal-vertical detection	Simplified decoder
Ultrafast SEC-xAEC-DED	Reduce and fix the number of 1s in **H** rows and columns	Encoder, Syndrome calculation
	Simplification of error correction functions	Corrector (final phase of decoding)

Ultrafast solution employs 47% more area, but only 18% more power consumption. In return, Ultrafast solution is 103% faster and has higher error coverage.

Designers may decide between different solutions, which one fits better to the design constrains and requirements. Our proposal offers fast decoding and flexible error coverage with moderate area and power overheads.

E. SUMMARY OF METHODS EMPLOYED TO REDUCE THE DELAY

The analysis performed in this section allows concluding that Ultrafast codes achieve the lowest delay with a high error coverage. Table 2 summarizes the main methods employed by the different codes to reduce the delay and the improved component of the ECC.

V. CONCLUSION

This paper presents Ultrafast codes, a family of error control codes especially designed for very fast encoding and decoding operations. These codes are particularly well suited for applications where the key requirement is a very low latency in the encoder and decoder circuits. This is the case of the protection of the information stored in registers in a microprocessor, for example. As the number of registers is small, the use of ECCs with high redundancy, like Ultrafast codes, may be affordable. In return, Ultrafast codes offer high-speed encoder and decoder circuits, and interesting error coverages. These codes can also be useful to protect high-speed memories or caches.

Firstly, we have summarized Ultrafast SEC, SEC-DED and SEC-DAEC codes, presented in previous works. Then, new Ultrafast SEC-xAEC-DED codes have been introduced, describing the design methodology and the implementation details. Several examples of Ultrafast codes have been implemented and simulated in order to validate the error coverage, as well as they have been synthesized by using a standard cell library to evaluate the overhead induced by the encoder and decoder circuits. Ultrafast codes have been compared to several fast existing alternatives. The results confirm that Ultrafast codes achieve very low propagation delays, whereas adding very reasonable increments in the silicon area and the power consumption. Using the method proposed in Section III.C to construct codes for longer data words, delays do not depend on the word length from a logic viewpoint, but it may appear a small dependency due to implementation details. This is a distinguishing characteristic of Ultrafast codes versus other error control codes, as they take great advantage of the simplification of logic functions and the low Hamming weights of rows and columns of their parity check matrices. The speed-ups obtained for longer data words are very interesting (up to 160% for 64-bit data words).

There is still a lot of ongoing work. The simplifications depend on the parity check matrix employed. A deeper study is required to determine how to find the matrix with the best simplifications. Additional research on Ultrafast codes for long data words, and applications to different memory structures, are part of the ongoing and future work.

REFERENCES

[1] *International Technology Roadmap for Semiconductors*. Accessed: Sep. 25, 2019. [Online]. Available: http://www.itrs2.net

[2] 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, Jul. 2010.

[3] D. Gil-Tomás, J. Gracia-Morán, J. C. Baraza-Calvo, L. J. Saiz-Adalid, and P. J. Gil-Vicente, "Studying the effects of intermittent faults on a microcontroller," *Microelectron. Rel.*, vol. 52, no. 11, pp. 2837–2846, Nov. 2012.

[4] E. Fujiwara, *Code Design for Dependable Systems*. Hoboken, NJ, USA: Wiley, 2006.

[5] R. W. Hamming, "Error detecting and error correcting codes," *Bell Syst. Tech. J.*, vol. 29, no. 2, pp. 147–160, Apr. 1950.

[6] L.-J. Saiz-Adalid, P. Reviriego, P. Gil, S. Pontarelli, and J. A. Maestro, "MCU tolerance in SRAMs through low-redundancy triple adjacent error correction," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 10, pp. 2332–2336, Oct. 2015.

[7] R. C. Baumann, "Soft errors in advanced computer systems," *IEEE Design Test. Comput.*, vol. 22, no. 3, pp. 258–266, May/Jun. 2005.

[8] J. Barak, M. Murat, and A. Akkerman, "SEU due to electrons in silicon devices with nanometric sensitive volumes and small critical charge," *Nucl. Instrum. Methods Phys. Res. B, Beam Interact. Mater. At.*, vol. 287, pp. 113–119, Sep. 2012.

[9] G. Tsiligiannis, L. Dilillo, A. Bosio, P. Girard, S. Pravossoudovitch, A. Todri, A. Virazel, H. Puchner, C. Frost, F. Wrobel, and F. Saigné, "Multiple cell upset classification in commercial SRAMs," *IEEE Trans. Nucl. Sci.*, vol. 61, no. 4, pp. 1747–1754, Aug. 2014.

[10] G. I. Zebrev, K. S. Zemtsov, R. G. Useinov, M. S. Gorbunov, V. V. Emeliyanov, and A. I. Ozerov, "Multiple cell upset cross-section uncertainty in nanoscale memories: Microdosimetric approach," in *Proc. RADECS*, Moscow, Russia, Sep. 2015, pp. 1–5.

[11] R. K. Lawrence and A. T. Kelly, "Single event effect induced multiple-cell upsets in a commercial 90 nm CMOS digital technology," *IEEE Trans. Nucl. Sci.*, vol. 55, no. 6, pp. 3367–3374, Dec. 2008.

[12] D. Teixeira-Franco, J.-F. Naviner, and L. Naviner, "Yield and reliability issues in nanoelectronic technologies," *Ann. Telecommun.*, vol. 61, nos. 11–12, pp. 1422–1457, Dec. 2006.

[13] J. A. Blome, S. Gupta, S. Feng, and S. Mahlke, "Cost-efficient soft error protection for embedded microprocessors," in *Proc. CASES*, Seoul, Republic Korea, 2006, pp. 421–431.

[14] L. J. Saiz-Adalid, P. Gil, J. Gracia-Morán, D. Gil-Tomás, and J. C. Baraza-Calvo, "Ultrafast single error correction codes for protecting processor registers," in *Proc. EDCC*, Paris, France, Sep. 2015, pp. 144–154.

[15] L. J. Saiz-Adalid, P. Gil, J. C. Ruiz, J. Gracia-Morán, D. Gil-Tomás, and J. C. Baraza-Calvo, "Ultrafast error correction codes for double error detection/correction," in *Procs. EDCC*, Gothenburg, Sweden, Sep. 2016, pp. 108–119.

[16] T. Calin, M. Nicolaidis, and R. Velazco, "Upset hardened memory design for submicron CMOS technology," *IEEE Trans. Nucl. Sci.*, vol. 43, no. 6, pp. 2874–2878, Dec. 1996.

[17] *Intel Xeon Processor E7 Family: Reliability, Availability, and Serviceability*, Intel, Santa Clara, CA, USA, 2012.

[18] L. J. Saiz-Adalid, P. J. Gil-Vicente, J. C. Ruiz, D. Gil-Tomás, J.-C. Baraza, and J. Gracia-Morán, "Flexible unequal error control codes with selectable error detection and correction levels," in *Proc. SAFECOMP*, Toulouse, France, 2013, pp. 178–189.

[19] A. Neubauer, J. Freudenberger, and V. Kühn, *Coding Theory: Algorithms, Architectures and Applications*. Hoboken, NJ, USA: Wiley, 2007.

[20] M.-Y. Hsiao, "A class of optimal minimum odd-weight-column SEC-DED codes," *IBM J. Res. Develop.*, vol. 14, no. 4, pp. 395–401, Jul. 1970.

[21] J. Li, P. Reviriego, L. Xiao, C. Argyrides, and J. Li, "Extending 3-bit burst error-correction codes with quadruple adjacent error correction," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 2, pp. 221–229, Feb. 2018.

[22] D. Rossi and C. Metra, "Error correcting strategy for high speed and high density reliable flash memories," *J. Electron. Test.*, vol. 19, no. 5, pp. 511–521, Oct. 2003.

[23] P. Reviriego, M. Demirci, A. Evans, and J. A. Maestro, "A method to design single error correction codes with fast decoding for a subset of critical bits," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 63, no. 2, pp. 171–175, Feb. 2016.

[24] K. Namba and F. Lombardi, "A single and adjacent error correction code for fast decoding of critical bits," *IEEE Trans. Comput.*, vol. 67, no. 10, pp. 1525–1531, Oct. 2018.

[25] M. Y. Hsiao, D. C. Bossen, and R. T. Chien, "Orthogonal Latin square codes," *IBM J. Res. Develop.*, vol. 14, no. 4, pp. 390–394, Jul. 1970.

[26] M. Zhu, L. Xiao, S. Li, and Y. Zhang, "Efficient two-dimensional error codes for multiple bit upsets mitigation in memory," in *Proc. 25th IEEE Int. Symp. Defect Fault Tolerance VLSI Syst.*, Kyoto, Japan, Oct. 2010, pp. 129–135.

[27] P. Reviriego, S. Pontarelli, J. A. Maestro, and M. Ottavi, "A method to construct low delay single error correction codes for protecting data bits only," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 32, no. 3, pp. 479–483, Mar. 2013.

[28] J. Li, P. Reviriego, L. Xiao, Z. Liu, L. Li, and A. Ullah, "Low delay single error correction and double adjacent error correction (SEC-DAEC) codes," *Microelectron. Rel.*, vol. 97, pp. 31–37, Jun. 2019.

[29] P. Reviriego, S. Pontarelli, J. A. Maestro, and M. Ottavi, "Low-cost single error correction multiple adjacent error correction codes," *Electron. Lett.*, vol. 48, no. 23, pp. 1470–1472, Nov. 2012.

[30] *Cadence: EDA Tools and IP for System Design Enablement*. Accessed: Dec. 23, 2018. [Online]. Available: https://www.cadence.com

[31] J. E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W. R. Davis, P. D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, and R. Jenkal, "FreePDK: An open-source variation-aware design kit," in *Procs. MSE*, San Diego, CA, USA, 2007, pp. 173–174.

[32] *NanGate FreePDK45 Open Cell Library*. Accessed: May 28, 2018. [Online]. Available: http://www.nangate.com/

[33] D. Siewiorek and R. S. Swarz, *Reliable Computer Systems: Design and Evaluation*. Natick, MA, USA: AK Peters, 1998.

[34] A. Sarkar, J. Samanta, A. Barman, and J. Bhaumik, "FPGA implementation of OLS (32, 16) code and OLS (36, 20) code," in *Communication, Devices, and Computing* (Lecture Notes in Electrical Engineering), vol. 470, J. Bhaumik, I. Chakrabarti, B. De, B. Bag, and S. Mukherjee, Eds. Singapore: Springer, 2017, pp. 151–161.
[35] S. Liu, L. Xiao, and Z. Mao, "Extend orthogonal Latin square codes for 32-bit data protection in memory applications," *Microelectron. Rel.*, vol. 63, pp. 278–283, Aug. 2016.
[36] A. Sánchez-Macián, P. Reviriego, and J. A. Maestro, "Combined SEU and SEFI protection for memories using orthogonal latin square codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 63, no. 11, pp. 1933–1943, Nov. 2016.
[37] J. A. Maestro, P. Reviriego, and M. F. Flanagan, "Error correction coding for electronic circuits," in *Energy-Efficient Fault-Tolerant Systems*, J. Mathew, R. Shafik, and D. Pradhan, Eds. New York, NY, USA: Springer, 2014, pp. 137–168.

LUIS-J. SAIZ-ADALID received the M.Sc. and Ph.D. degrees in computer engineering from the Universitat Politècnica de València (UPV), Spain, in 1995 and 2015, respectively.

After 15 years in the industry (IBM, 1995-Celestica, 1998). He is currently an Associate Professor with the Department of Computer Engineering (DISCA), UPV. His research interests include design and implementation of digital systems, design and validation of fault-tolerant systems and design of error control codes. He is a member with the Fault-Tolerant Systems (STF) research line within the Institute of Information and Communication Technologies (ITACA) from the UPV.

JOAQUÍN GRACIA-MORÁN received the B.Sc., M.Sc., and Ph.D. degrees in computer engineering from the Universitat Politècnica de València (UPV), Spain, in 1995, 1997, and 2004, respectively, where he is currently an Associate Professor with the Department of Computer Engineering (DISCA). His research interests include design and implementation of digital systems, design and validation of fault-tolerant systems and VHDL-based fault injection. He is a member with the Fault-Tolerant Systems (STF) research line within the Institute of Information and Communication Technologies (ITACA) from the UPV.

DANIEL GIL-TOMÁS received the B.Sc. degree in electrical and electronic physics from the Universitat de València, Spain, in 1985, and the Ph.D. degree in computer engineering from the Universitat Politècnica de València (UPV), Spain, in 1999, where he is currently an Associate Professor with the Department of Computer Engineering (DISCA). His research interests include design and validation of fault-tolerant systems, reliability physics and reliability of emerging nanotechnologies. He is a member with the Fault-Tolerant Systems (STF) research line within the Institute of Information and Communication Technologies (ITACA) from the UPV.

J.-CARLOS BARAZA-CALVO received the B.Sc. and Ph.D. degrees in computer engineering from the Universitat Politècnica de València, (UPV), Spain, in 1993 and 2003, respectively, where he is currently an Associate Professor with the Department of Computer Engineering (DISCA). His research interests include design and implementation of digital systems, design and validation of fault-tolerant systems and VHDL-based fault injection. He is a member with the Fault-Tolerant Systems (STF) research line within the Institute of Information and Communication Technologies (ITACA).

PEDRO-J. GIL-VICENTE (M'93) received the B.Sc. degree in electronic engineering from the Universitat Politècnica de Catalunya (UPC), Tarragona, Spain, in 1979, and the M.Sc. degree in industrial engineering and the Ph.D. degree in computer engineering from the Universitat Politècnica de València (UPV), Spain, in 1985 and 1992, respectively, where he is currently a Professor with the Department of Computer Engineering (DISCA). He has been the Head of the DISCA, and he is also the Head of the Fault-Tolerant Systems (STF) research line, within the Institute of Information and Communication Technologies (ITACA). His research interests include 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 taught courses on Computer Technology, Digital Design, Computer Networks, and Fault Tolerant Systems. He has authored more than 150 research articles on these subjects. Prof. Gil-Vicente has also served as a Program Committee member in the IEEE/IFIP 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 a Reviewer in international journals and congresses related to dependability and security. He was a General Chair of the EDCC-8 conference, held in Valencia on April 2010. He will be the General Co-Chair of the 50th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2020), to be held in Valencia.