This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2498547, IEEE Transactions on Computers

Unequal Error Protection Codes Derived from Double Error Correction Orthogonal Latin Square Codes

Mustafa Demirci, Pedro Reviriego and Juan Antonio Maestro

***Abstract*—In recent years, there has been a growing interest in multi-bit Error Correction Codes (ECCs) to protect SRAM memories. This has been caused by the increased number of multiple errors that memories suffer as technology scales. To be suitable to protect an SRAM memory, an ECC has to be decodable in parallel and with low latency. Among the codes proposed for memory protection are Orthogonal Latin Square (OLS) codes that provide low latency decoding and a modular construction. For some applications, like multimedia or signal processing, the effect of errors on the memory bits can be very different depending on their position on the word. Therefore, in these cases, it is more effective to provide different degrees of error correction for the different bits. This is done with Unequal Error Protection (UEP) codes. In this paper, UEP codes are derived from Double Error Correction (DEC) Orthogonal Latin Square (OLS) codes. The derived codes are implemented for an FPGA platform to evaluate the decoder complexity and latency. The results show that the new codes can be implemented with lower decoding delay than traditional SEC-DED codes and with a cost similar to that of both DEC OLS and SEC-DED codes.**

***Index Terms*—Unequal Error Protection, Orthogonal Latin Squares codes, Majority logic decoding.**

I. Introduction

There are several sources of errors that affect modern electronic circuits, such as manufacturing defects, circuit ageing, electromagnetic disturbances or radiation induced soft errors [1],[2]. Many different techniques can be used to either prevent failures from occurring or to detect and correct them [3]. Those include modifications to the manufacturing process, circuit level and logic level techniques. Techniques at different levels are commonly combined to achieve the desired reliability target [4].

SRAM memories are one of the most commonly used electronic circuits. They are present as standalone devices and also embedded in most Digital Signal Processors (DSPs), microcontrollers, System On Chip (SoCs) and FPGAs. Therefore their protection is critical to ensure system reliability [5]. To deal with defects, redundant rows and columns are commonly used [6]. For other types of failures, Error Correction Codes (ECCs) are widely used [7],[8].

Traditionally, the ECCs used to protect SRAM memories have focused on providing Single Error Correction and Double Error Detection (SEC-DED) [9]. However, as electronic technology scales, there is an increased number of multiple bit errors. For example, for radiation induced soft errors, the percentage of errors that affect more than one memory cell has increased with each technology node [10]. To correct multiple bit errors, more advanced ECCs are needed. Although there are many such codes [11], most of them do not fit the needs of an SRAM memory. To be used with an SRAM memory, encoding and decoding need to be done in parallel in less than one clock cycle. However, most multi-bit error correction codes are serially decoded or require complex decoding circuitry when the decoder is implemented in parallel. This is for example the case of Bose Chaudhuri Hocquenghem (BCH) codes, for which a parallel decoder requires complex circuitry [12]. However, there is a property called "one step majority logic decodable" (OS-MLD) that only a few ECCs have, that makes them suitable for fast parallel encoding.

For a code that is one step majority logic decodable, decoding can be done by computing some parity check equations for each bit and taking the majority vote among them to decide if the bit needs to be corrected [11]. A number of OS-MLD codes have been recently proposed to protect memories. The first works focused on the use of Euclidean Low Density Parity Check Codes (EG-LDPC) [13],[14] and showed the benefits of OS-MLD codes. Difference Set codes have also been studied [15],[16]. More recent works have analyzed the use of Doubly Transitive Invariant (DTI) codes in combination with Hamming codes [17]. All those OS-MLD codes have a major limitation: the number of available word sizes and error correction capabilities is limited. In addition, the available sizes have a number of data bits that are not a power of two and therefore do not fit the SRAM memory word sizes that are commonly a power of two, like 32 or 64 bits. This limitation has drawn attention to another family of OS-MLD codes, Orthogonal Latin Squares (OLS) codes [18].

Orthogonal Latin Squares codes are derived from the

Manuscript received 2 September 2015.

M. Demirci is with Aselsan, Mehmet Akif Ersoy Mahallesi 296, 16, 06370 Yenimahalle / Ankara, Turkey, (email: mdemirci@aselsan.com.tr).

P. Reviriego and J.A. Maestro are with Universidad Antonio de Nebrija, C/ Pirineos, 55 E-28040 Madrid, Spain (phone: +34-914521100; fax: +34-914521110; email: {previrie, jmaestro}@nebrija.es).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2498547, IEEE Transactions on Computers

concept of Latin Square [19] and have been recently proposed to protect interconnects [20], caches [21] and memories [22]. Their main advantage compared to other OS-MLD codes is that they provide a larger number of options in terms of word size and error correction capabilities. This however, comes at the cost of a larger number of parity check bits for the same word size. To address this issue, a method to extend OLS codes so that they can protect more data bits with the same number of parity check bits, has recently been proposed [23].

In some applications, the bits on a memory word have different importance. This occurs for example in signal processing and multimedia applications [24],[25].For example, in a signal processing application, error correction can focus on the most significant bits or in a video compression system like H.264 on the more relevant parameters [26]. In those cases, it may be more effective to provide different protection levels for the different bits in a word. This provides an opportunity to optimize the ECCs for these applications reducing the overhead required to implement protection. This idea has been recently explored in several works like for example [24],[25],[26],[27] that propose different Unequal Error Protection (UEP) codes.

In this paper, two techniques to derive UEP codes from DEC OLS codes based are presented. The proposed codes ensure that important bits are corrected when there is a double error and that all single errors are corrected. Additionally, some double errors that affect the least significant bits can also be corrected. The codes can be decoded in parallel with low complexity and latency and do not require any additional parity bit compared to standard DEC OLS codes. Therefore they can be useful to implement UEP when low latency decoding is required.

In more detail, the main contributions of this paper are:

- To apply the method in [23] to derive UEP codes.
- To propose a new method to derive UEP codes that enables the protection of larger data blocks.
- To propose and evaluate efficient decoder implementations for the new UEP codes.

The rest of the paper is organized as follows. Section II introduces Orthogonal Latin Squares codes. In Section III, the proposed Unequal Error Protection (EUP) codes are presented. Then in Section IV, the error correction capabilities and the implementation of the proposed codes in an FPGA are evaluated. The results are also compared with those of standard Orthogonal Latin Squares codes. Finally, the paper ends with some conclusions and ideas for future work in Section V.

II. Orthogonal Latin Squares Codes

As mentioned in the introduction, Orthogonal Latin Square codes are derived from Latin Squares [19]. A Latin Square of size m is a square matrix of size m that has permutations of the numbers 0 to m-1 on both its rows and columns. Two Latin squares are orthogonal if when superimposed every ordered pair of elements appears only once. As their name indicates, Orthogonal Latin Square codes are derived from orthogonal Latin squares. The number of data bits is equal to the number of elements in the Latin square (mxm) and the number of parity bits is given by the number of Latin Squares used times m. Each Latin square is used to form m rows of the parity check matrix (H). This is done by assigning a data bit to each bit in the Latin Square and forming each of the rows having ones only in the positions that have the value that corresponds to the row. This ensures that the columns of the parity check matrix (H) share at most one position with a value of one for each group of rows that corresponds to a Latin square. Then, since the Latin Squares are orthogonal, the sharing applies to the entire column and not only to each of the groups.

The general form of the parity check matrix of an OLS code is:

$$H = \begin{bmatrix} M_1 \\ M_2 \\ \dots \\ M_{2t} \end{bmatrix} I_{2tm} \quad (1)$$

where I_{2tm} is the identity matrix of size $2tm$ and $M_1, M_2, \dots, M_{2t}$ are matrixes with size $m \times m^2$ derived from Orthogonal Latin Squares of size $m \times m$ as described before.

By design, the columns of the parity check matrix share at most a position with a one and have weight $2t$ on the data bits. Therefore, each data bit can be corrected by simply taking the majority vote of the $2t$ parity checks on which it participates.

In Figure 1, the parity check matrix of a (32,16) Double Error Correction (DEC) OLS code is shown. It can be observed that the columns that correspond to data bits have a weight of four and that they share at most a position with a one. The decoding circuitry for the first data bit is shown in Figure 2. The decoding is the same for the rest of the bits, but using the relevant parity checks.

The discussion so far shows 1) the flexibility in the construction of OLS codes, as the error correction capability can be adjusted using a different number of M_i matrixes and 2) the simple decoding due to the OS-MLD property. The parameters of the DEC OLS codes that will be used in the rest of the paper are summarized in Table I. It can be observed that words of 16 and 64 bits can be protected with existing DEC OLS codes while there is no code that fits a 32-bit data word.

TABLE I. PARAMETERS OF THE DEC OLS CODES STUDIED

k	n-k	m
16	16	4
25	20	5
64	32	8

In a recent paper [23], it was observed that the groups of parity check bits of each M_i matrix can be used to extend OLS codes. This is because any number of errors on one of those groups can affect at most one bit in the decoding of a data bit. Therefore those groups can be used to form additional codes. In [23], the extension of DEC and TEC codes was studied. The obtained extended codes for the DEC codes in Table I are summarized in Table II.

TABLE II. PARAMETERS OF THE EXTENDED DEC OLS CODES [23]

k	n-k	m
20	16	4
29	20	5
72	32	8

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2498547, IEEE Transactions on Computers

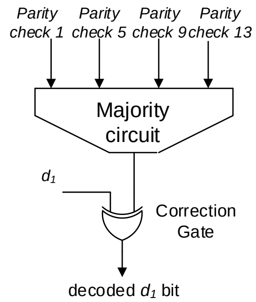

Fig. 2: Illustration of OS-MLD decoding for the first bit of the (32,16) DEC OLS code.

III. PROPOSED UNEQUAL ERROR PROTECTION CODES

In this section, two schemes to derive Unequal Error Protection (UEP) codes from DEC OLS codes are proposed. The first is based on the method presented in [23] but using odd weight SEC-DED codes for the extension. The second presents a new method to extend DEC OLS codes that allows the combination of bits from different M_i sub-blocks. This results in a more efficient code in terms of block size and parity check bits at the expense of a slightly more complex decoding.

A. Single M_i sub-block code extension

As discussed in the previous section, OLS codes can be extended by using the parity check bits on each of the M_i sub-blocks to form a new code. One option for DEC OLS codes is to use SEC-DED codes for the extended codes. For example, for the DEC OLS code with k_{OLS} = 16 and n = 32, the sub-blocks have four parity check bits and a SEC-DED code that protects four bits can be formed for each sub-block. Therefore, sixteen additional bits can be protected with SEC-DED. This means that a 32-bit data word can be protected with two protection levels, DEC and SEC-DED. Let us denote as $k_{SEC\text{-}DED}$ the additional data bits that are added. Then, we have a code with k_{OLS} = 16, $k_{SEC\text{-}DED}$ = 16 and n = 48.

For the SEC-DED codes, a Hsiao code with columns of weight three can be used [9] so that when a double error occurs there is no miscorrection in the data bits. The obtained parity check matrix is illustrated in Figure 3. It can be observed that the added columns for the SEC-DED codes have all the ones in a single M_i sub-block. Conversely, the columns for the DEC OLS protected data bits have a single one per sub-block as discussed before.

	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
M_1	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	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	1	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	
M_2	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	
	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	
	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	I_{4m}
M_3	0	1	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	
	0	0	1	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	
	0	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	
	1	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
M_4	0	1	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	
	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	
	0	0	0	1	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	

Fig. 1: Parity check matrix H for the OLS code with k = 16 and t=2.

The encoder can be implemented as a combination of the DEC-OLS and SEC-DED encoders. This is illustrated in Figure 4. Basically, the encoders can operate in parallel and the final parity check bits are obtained doing the xor of the DEC-OLS results, (c_o bits in the Figure) and SEC-DED results, (c_e bits in the Figure). Similarly, the decoder can also be implemented as a combination of DEC-OLS and SEC-DED decoders. The decoder is illustrated in Figure 5. In this case, the syndrome computation is common to both and consists of re-computing the parity check bits as in the encoder and performing the xor with the stored parity check bits. The syndrome bits (s bits in the Figure) are then used as inputs to the OLS majority voters that determine if the correction is needed for the OLS data bits (d bits in the Figure). The syndrome bits that correspond to each of the sub-blocks are also compared with the SEC-DED syndrome patterns to determine if a correction is needed on the SEC-DED data bits (d_e bits in the Figure). From the description, both the encoder and decoder have a similar complexity per bit to that of the OLS code. The delay for the OLS decoding should also be similar as only one level of xor gates is added to the encoder and syndrome computation.

Let us now discuss the error patterns that the proposed code can correct. The first comment is that the goal is to correct errors on the data bits as those are the ones used by the application. This is a common assumption when OLS codes are used as the majority voting decoding can only be used for the data bits [18]. Therefore, errors on the parity bits are only a problem when they cause a miscorrection on the data bits. For example, in the proposed code with k_{OLS} = 16, $k_{SEC\text{-}DED}$ = 16 and n = 48 whose parity check matrix is shown in Figure 3, an error that affects the first three parity check bits would cause a miscorrection on bit d_{e1}. Since the weight of the columns is three for the SEC-DED data bits and four for the OLS data bits, only more than two errors on the parity check bits can cause a miscorrection.

The analysis of the error patterns considers single and double errors. For single errors, when the error occurs on a data bit, it is corrected by either the OLS code (d bits) or by the SEC-DED code (d_e bits) as both codes can correct single bit errors. As discussed before, a single error on a parity bit cannot cause a miscorrection. Therefore, the proposed code ensures that when a single bit error occurs, the correct data bits are recovered after decoding.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2498547, IEEE Transactions on Computers

For double errors, the analysis is more complex as the bits affected can be:

1. Two parity check bits.
2. Two OLS data bits.
3. Two SEC-DED data bits.
4. One parity bit and one OLS data bit.
5. One parity bit and one SEC-DED data bit.
6. One OLS data bit and one SEC-DED data bit.

In the first case, as discussed before, no miscorrection can occur and therefore the data will be correct.

In the second case, since the OLS code is DEC the two errors are corrected.

In the third case, not all the errors can be corrected. When the two data bits are on different SEC-DED words, both appear as single errors on each word and can be corrected. However, when the two errors affect bits of the same SEC-DED word (for example bits d_{e1} and d_{e2}) the errors cannot be corrected but they will not cause miscorrection on the SEC-DED data bits. This is a property of odd weight codes for which a double bit error produces an even weight syndrome [9] that cannot cause a correction. As there are four SEC-DED words the probability that the two errors fall on the same word would be approximately 25% and therefore the majority of the errors will be corrected.

In the fourth case, the affected data bit is an OLS bit and the error is corrected by the OLS code and the correct data is obtained.

In the fifth case, the data bit is a SEC-DED bit and the error will not always be corrected. This will only occur when the parity bit in error belongs to a different M_i sub-block. For example, for the matrix in Figure 3, let us suppose there is an error on bit d_{e1} and on the fifth parity check bit. Then the syndrome bits (s_1,s_4, s_3,s_4) used to correct d_{e1} will take the values 1,1,1,0 and the error will be corrected. However, if the error affects the fourth parity check bit, the values would be 1,1,1,1 and the error is not corrected. This means that, on average, 75% of these type of errors will be corrected as there are four sub-blocks.

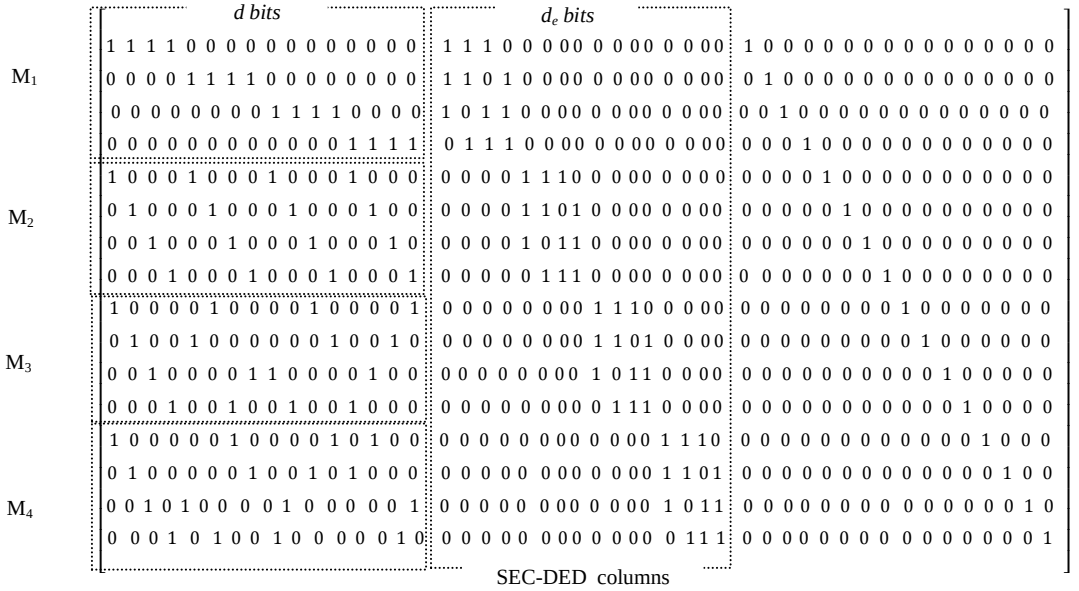

Fig. 3: Parity check matrix H for the single sub-block SEC-DED extended OLS code with k_{OLS} = 16 and $k_{SEC\text{-}DED}$ = 16.

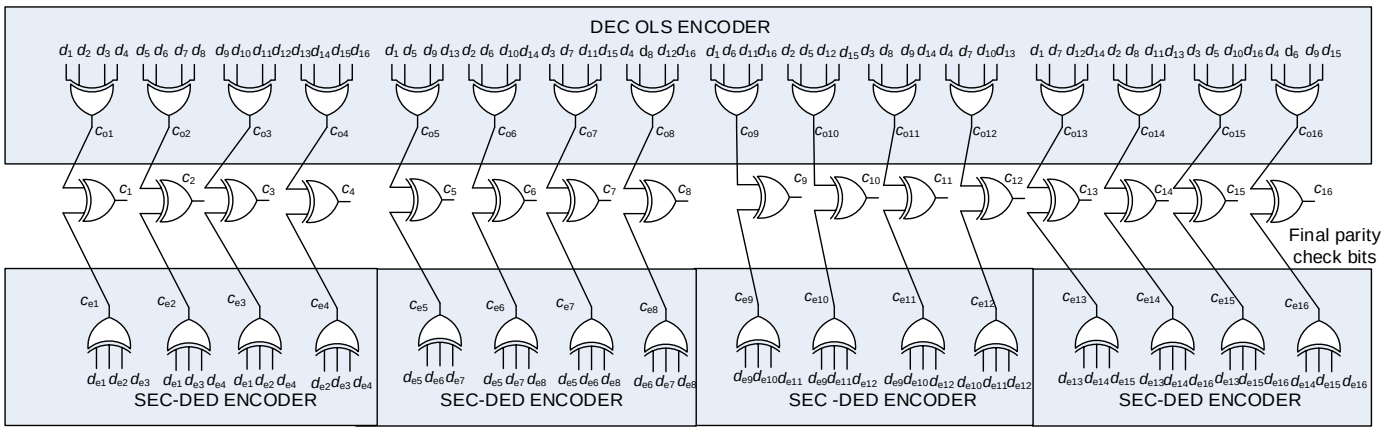

Fig. 4: Encoder for the proposed (48,16,16) DEC OLS code single sub-block extended with SEC-DED.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2498547, IEEE Transactions on Computers

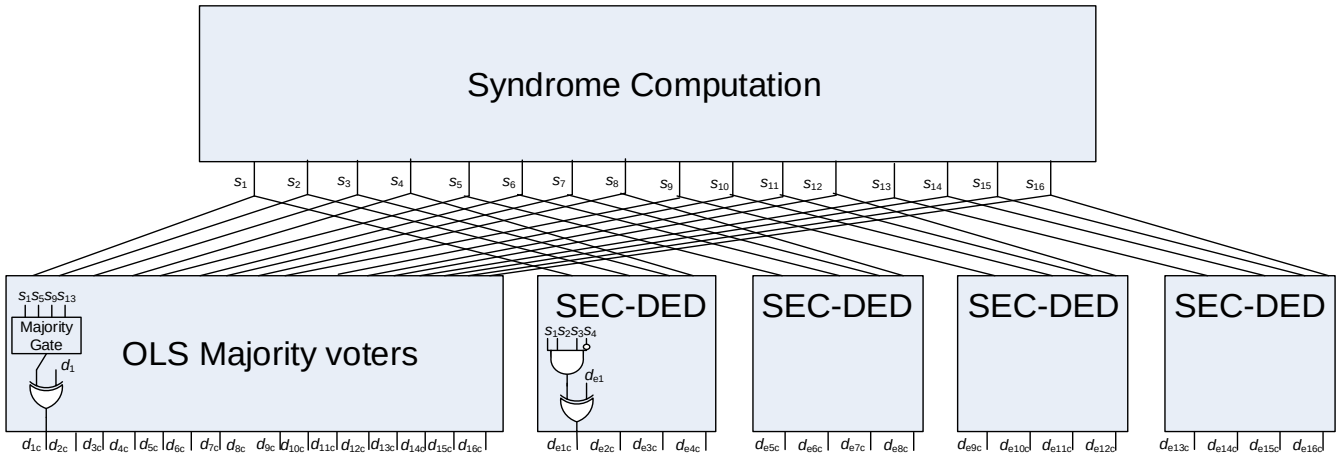

Fig. 5: Decoder for the proposed (48,16,16) DEC OLS code single sub-block extended with SEC-DED.

In the sixth case, one error affects the OLS data bits and the other the SEC-DED data bits. The error on the OLS data bits will always be corrected, but the error in the SEC-DED bits will not be corrected. This is because an error on an OLS data bit sets to one a bit in each of the M_i groups and therefore will change the syndrome bits needed to correct the SEC-DED data bit. It would be possible to modify the decoder shown in Figure 5 to correct this type of double errors. This can be done by first decoding the OLS bits and after re-computing the syndrome and decoding the SEC-DED bits. With the first step the OLS bit error is corrected in such a way that the second step only has to deal with a single error on the SEC-DED bits. This modification would however significantly increase the decoding delay compared to a standard OLS decoder. For this reason this alternative decoding scheme has not been implemented.

Throughout this subsection, the presentation of the proposed codes has focused on one particular DEC OLS code example. The same approach can be used to extend other DEC OLS codes. The extended codes for some OLS codes are summarized in Table III. It can be observed that as the size of the original DEC OLS code increases, the number of additional SEC-DED bits also increases. It is also worth mentioning that the first code can protect a 32-bit data word and the second one a 64-bit data word (by shortening one bit). Therefore both codes fit word sizes of SRAM memories that are commonly used in signal processing applications. The third code can protect larger memory words and could be useful for caches.

TABLE III. PARAMETERS OF THE SINGLE SUB-BLOCK SEC-DED EXTENDED DEC OLS CODES

k_{OLS}	$k_{SEC-DED}$	n-k	m
16	16	16	4
25	40	20	5
64	224	32	8

B. *Double M_i sub-block code extension*

From the discussion of the single sub-block code extension, it becomes clear that it would be beneficial to be able to combine different sub-blocks to form larger codes. For example, combining two four-bit sub-blocks would enable the use of a SEC-DED code with eight parity bits that can protect more than 64 bits. An example of a code extended using a SEC-DED on two sub-blocks is shown in Figure 6. In this case, the (32,16) DEC OLS code is extended using DEC codes on M_1 and M_2 and a single odd weight code that covers both M_3 and M_4. More bits could be protected with the SEC-DED code, but the code is designed to protect a 32-data bit word.

The main problem when a code spans several sub-blocks is that a single bit error can affect more than one of the OS-MLD equations used to decode the DEC OLS protected bits. This means that the OLS DEC code will not work properly. For example, considering the code in Figure 6, let us assume that there are errors on the first data bit that is DEC OLS protected and on data bit 19 that is the first protected by the SEC-DED code. In this case, the first error will affect parity checks 1, 5, 9 and 13 and the second parity checks 9, 13, 14, 15 and 16. Therefore, only parity checks 1, 5, 14, 15 and 16 will take a value of one and the first data bit will not be corrected. This shows how using more than one sub-block for the extended codes can cause problems.

To overcome this limitation, a key observation is that when there is a single error on the SEC-DED protected bits the weight of the M_3 and M_4 parity check bits will be odd. If there is a double error affecting a SEC-DED protected bit and an OLS protected bit, the weight will also be odd. This is because the OLS code will have a weight two on the M_3 and M_4 parity check bits that when combined with the odd weight of the SEC-DED code results in an odd weight.

Now let us suppose that the decoding is modified so that when there is an odd weight on the M_3 and M_4 parity check bits, the DEC OLS data bits are decoded using only M_1 and M_2 (as in a SEC OLS code). This is illustrated in Figure 7 for the first data bit. Now, if the same example of an error on bits 1 and 19 is considered, it can be seen that the new decoder is able to correct the error on the first data bit.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2498547, IEEE Transactions on Computers

$$
\begin{bmatrix}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
1&0&0&0&0&1&0&0&0&0&1&0&0&0&0&1&0&0&1&0&0&1&0&1&1&1&1&0&1&1&1&1&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&1&0&0&0&0&0&0&1&0&0&1&0&0&0&0&1&1&1&1&0&1&1&0&1&1&1&1&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&1&0&0&0&0&1&1&0&0&0&0&1&0&0&0&0&0&0&1&0&1&1&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0\\
0&0&0&1&0&0&1&0&0&1&0&0&1&0&0&0&0&0&0&0&1&1&1&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\
1&0&0&0&0&0&1&0&0&0&0&1&0&1&0&0&0&0&1&1&1&0&1&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&1&0&0&0&0&0&1&0&0&1&0&1&0&0&0&0&0&1&1&0&1&1&1&0&0&0&0&1&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&1&0&1&0&0&0&0&1&0&0&0&0&0&1&0&0&1&1&1&0&0&1&1&1&1&1&1&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&1&0&1&0&0&1&0&0&0&0&0&1&0&0&0&1&1&0&1&0&1&1&1&1&1&0&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1
\end{bmatrix}
$$

Fig. 6: Parity check matrix H for the SEC-DED double sub-block extended OLS code with k_{OLS} = 18 and $k_{SEC\text{-}DED}$ = 14.

The proposed modification clearly corrects all single errors affecting a data bit. Let us consider the different double error scenarios that are the same considered in the previous subsection:

1. Two parity check bits.
2. Two OLS data bits.
3. Two SEC-DED data bits.
4. One parity bit and one OLS data bit.
5. One parity bit and one SEC-DED data bit.
6. One OLS data bit and one SEC-DED data bit.

In the first case, the M_3 and M_4 parity check bits can have even or odd weight depending on the affected parity bits. If the weight is even, the decoding is the same as in the single block extended codes and no data bit is miscorrected. If the weight is odd, then the other error will be on the M_1 and M_2 parity check bits that are used to decode the OLS bits with the modified procedure. However, since to perform a correction both the corresponding bits on M_1 and M_2 have to be one, no miscorrection can occur.

In the second case, the decoding is the same as in the single block extended codes and errors on the OLS protected bits are corrected.

In the third case, errors can affect only parity check bits on M_3 and M_4 and will have an even weight. Therefore no miscorrection can occur.

In the fourth case, if the parity bit is on M_3 or M_4, the weight will be odd number and the OLS bit is corrected using M_1 and M_2. When the error is on M_1 or M_2, the OLS bit is decoded and corrected using the standard OS-MLD decoding.

In the fifth case, the parity check bit in error can be in M_3 and M_4 and then the weight will be even and decoding is done normally with no miscorrection. When the parity check bit in error is in M_1 and M_2, the weight on M_3 and M_4 will be odd and the modified correction is applied. Again, as there will be a single one in M_1 and M_2, no miscorrection will occur.

In the sixth case, weight will be odd on M_3 and M_4 and the affected OLS data bit will be corrected with the modified procedure.

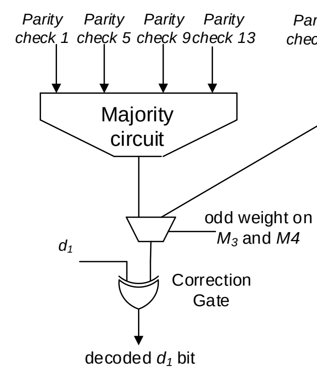

Fig. 7: Illustration of the proposed modified decoding for the first bit of the (48,18,14) double sub-block extended DEC OLS code.

The use of two sub-blocks can have another effect when there are errors on a DEC OLS protected bit and a parity bit: there can be three errors on M_3 and M_4. For example, an error on the first data bit and the last parity bit will cause parity checks 9, 13 and 16 to take a value of one. This can cause a miscorrection of a SEC-DED bit if weight three is used for the odd-weight SEC-DED columns that have ones on both M_3 and M_4. To avoid this issue, weight five or larger should be used on those columns (as shown in Figure 6). This reduces the total number of bits that can be protected for the case of m = 4 to 72. That is the number of combinations of weight five and seven plus the weight three combinations that have ones only on M_3 or on M_4 but not in both.

This discussion shows how the double sub-block extended code will be able to correct the DEC OLS protected data bits when there is a double error affecting any bits on the block. As in the single sub-block extended codes, some errors on the SEC-DED data bits cannot be corrected for some double error patterns. For example, when there is a double error on the SEC-DED bits. The analysis for each case is similar to the one presented for the single sub-block extended codes and is omitted for brevity. In any case, the experimental results

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2498547, IEEE Transactions on Computers

obtained by analyzing all the possible cases are presented in the next section.

The parameters for some double sub-block SEC-DED extended OLS codes that protect 32 and 64-bit data words are summarized in Table IV. In this case, only OLS codes with $m = 4$ are considered as those are enough to protect both 32 and 64-bit data words. Using codes with larger m to protect these word sizes has little interest as the single sub-block extended codes with the same m that can also protect 32 and 64-bit data words. When compared with the codes on Table III, it can be seen that the proposed codes increase the number of DEC bits that are protected for a 32-bit data word and reduce the number of the parity check bits needed to protect a 64-bit data word. This, as mentioned before, is done at the expense of increasing the decoding complexity.

TABLE IV. PARAMETERS OF THE DOUBLE SUB-BLOCK SEC-DED EXTENDED DEC OLS CODES

k_{OLS}	$k_{SEC\text{-}DED}$	$n\text{-}k$	m
18	14	16	4
18	46	16	4

IV. EVALUATION

The proposed codes have been evaluated both in terms of their error correction capabilities and the encoder and decoder complexity. Each of those aspects is covered in the following subsections. The evaluation has focused on the first two single sub-block extended codes in Table III and the double sub-block extended codes in Table IV. The reason is that these codes can protect words of 32 and 64-data bits that are used in multimedia and signal processing applications.

A. Error correction

The single and double bit error patterns (all combinations of single and double bit errors) have been exhaustively tested for the first two codes in Table III and the codes in Table IV. The results are summarized in Table V. It can be observed that all single errors are corrected as expected. For double errors, the results are grouped using the types described in the previous section. It can be observed that all double errors of types 1,2,4 are corrected. Those correspond to errors on the parity bits or on the OLS bits. For the remaining types, (3,5,6) the percentage of errors that can be corrected matches the theoretical values estimated in the previous section for the single sub-block extended codes. For the double sub-block extended codes, the results are also consistent with a theoretical analysis that, as discussed before, is not presented for brevity. This confirms that the OLS protected data bits are always corrected in the presence of two bit errors and that all single errors are also corrected. Most double errors that affect two SEC-DED data bits or a SEC-DED data bit and a parity bit (types 3 and 5) are also corrected in the single sub-block extended codes. The percentage of corrected errors is reduced significantly for double sub-block extended codes. This is expected as in this case, an error on a SEC-DED bit affects more bits and it is more likely to interfere with a second error. On the other hand, for errors that affect one OLS and one SEC-DED bit, single sub-block extended codes cannot correct the error on the SEC-DED bit. In this case, double sub-block extended codes can correct the errors that affect the two extended OLS bits (17 and 18) as their parity bits do not overlap with the parity bits of SEC-DED bits. This gives a 2/18 percentage of error correction for the SEC-DED bits. Finally, the table also shows the overall percentage of double errors that can be corrected. In this case single sub-block extended codes have a larger percentage. However it should be noted that the goal of the codes is to correct 100% of the errors on the OLS data bits, thus providing the desired UEP capability. This is achieved by both single and double sub-block extended codes.

TABLE V. ERROR CORRECTION FOR THE PROPOSED CODES

$n,k_{OLS},$ $k_{SEC\text{-}DED}$	*Single Error*	*Double Error*				
		types 1,2,4	*type 3*	*type 5*	*type 6*	*total*
(48,16,16)	100%	100%	80%	75%	0%	70%
(48,18,14)	100%	100%	0%	50%	11%	62%
(85,40,20)	100%	100%	77%	75%	0%	61%
(80,46,16)	100%	100%	0%	50%	11%	32%

B. Encoder and decoder complexity

The encoders and decoders for the proposed codes have been implemented on a Xilinx Virtex-6 series XC6VLX75T-2FF484 FPGA using XILINX ISE Design Suite 14.7 [28]. The results are presented in Tables VI and VII that also include the results for standard DEC OLS encoder/decoders and for odd weight SEC-DED codes. To estimate the usage of FPGA resources and delays, physical synthesis is carried out. During synthesis, for all designs, high level of effort is put to minimize resource usage while maintaining timing performance. As a measure of complexity, namely FPGA resource usage, the number of logic Look-up Tables (LUTs) is reported. The delay (maximum combinational logic delay path) estimated by the XILINX ISE Design Suite as a result of synthesis is shown as an estimate of the latency required to implement the codes. It can be observed that the proposed codes require more resources than the DEC OLS codes from which they are derived. This is expected as they also protect more data bits. On the other hand, the proposed codes have similar resource usage to that of the DEC OLS or SEC-DED codes that protect data words of the same size. To make a fair comparison, the number of LUTs per data bit should be used. This is shown in Table VIII. It can be observed that the cost per data bit is in most cases lower for the proposed codes than for the standard DEC OLS codes. When comparing to SEC-DED codes, the cost is sometimes slightly higher and sometimes slightly lower but comparable in all cases.

In terms of delay, both DEC OLS codes and the proposed UEP codes achieve lower decoding delay than SEC-DED codes. However, the proposed codes incur in an additional delay overhead compared to DEC OLS codes. Again, part of the delay increase can be attributed to the larger number of data bits, but another part is not as seen in the case of the 64 bit DEC OLS code. The delay is also larger for the double sub-block extended codes than for the single sub-block extended codes. This is expected as the decoding is more complex as explained in the previous section. In any case, it should be borne in mind that all the proposed UEP codes are faster than SEC-DED codes, which are the ECCs commonly used to protect memories. Therefore, the proposed UEP code would meet the speed requirements of many existing applications.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TC.2015.2498547, IEEE Transactions on Computers

TABLE VI. ENCODER LUTS AND DELAY (in ns)

ECC type	*Data size*	$n,k_{OLS},k_{SEC\text{-}DED}$	LUTs	Delay
DEC OLS	16	(32,16,0)	16	0.99
Single sub-block	32	(48,16,16)	30	1.27
Double sub-block	32	(48,18,14)	25	1.61
SEC-DED	32	(39,0,32)	29	1.88
DEC OLS	25	(45,25,0)	20	1.12
Single sub-block	65	(85,25,40)	44	1.54
Double sub-block	64	(80,18,46)	51	2.31
SEC-DED	64	(72,0,64)	64	2.12
DEC OLS	64	(96,64,0)	64	1.36

TABLE VII. DECODER LUTS AND DELAY (in ns)

ECC type	*Data size*	$n,k_{OLS},k_{SEC\text{-}DED}$	LUTs	Delay
DEC OLS	16	(32,16,0)	32	1.77
Single sub-block	32	(48,16,16)	64	2.03
Double sub-block	32	(48,18,14)	73	2.80
SEC-DED	32	(39,0,32)	62	3.52
DEC OLS	25	(45,25,0)	65	1.92
Single sub-block	65	(85,25,40)	121	2.53
Double sub-block	64	(80,18,46)	140	3.43
SEC-DED	64	(72,0,64)	120	4.05
DEC OLS	64	(96,64,0)	128	2.08

TABLE VIII. LUTS PER DATA BIT

ECC type	*Data size*	$n,k_{OLS},k_{SEC\text{-}DED}$	Encoder	Decoder
DEC OLS	16	(32,16,0)	1.00	2.00
Single sub-block	32	(48,16,16)	0.94	2.00
Double sub-block	32	(48,18,14)	0.78	2.28
SEC-DED	32	(39,0,32)	0.91	1.94
DEC OLS	25	(45,25,0)	0.80	2.60
Single sub-block	65	(85,25,40)	0.68	1.86
Double sub-block	64	(80,18,46)	0.79	2.19
SEC-DED	64	(72,0,64)	1.00	1.88
DEC OLS	64	(96,64,0)	1.00	2.00

V. CONCLUSIONS

In this paper, two techniques to derive Unequal Error Protection (UEP) codes from Double Error Correction (DEC) Orthogonal Latin Squares (OLS) codes have been presented. The derived UEP codes can protect part of the word with DEC and the other part with SEC-DED. The codes can be decoded in parallel with low latency. Finally they do not require any additional parity bits compared to a standard OLS code.

The proposed techniques have been used to obtain codes that can protect 32 and 64-data bit words that are commonly found in embedded systems. The implementation results for an FPGA platform have confirmed the low complexity and latency of both the encoder and the decoder.

Future work will consider the derivation of UEP codes from OLS codes that can correct more than two errors. For example a TEC OLS code can be extended to also provide DEC for additional bits. It would also be interesting to derive a formal and general proof of the UEP capabilities of the proposed codes that does not rely on a case by case analysis and exhaustive error pattern testing. Finally, a natural extension of this work will be to apply the proposed codes, and more broadly OLS codes, to protect Block RAMs inside FPGAs.

REFERENCES

[1] N. Kanekawa, E. H. Ibe, T. Suga and Y. Uematsu, "Dependability in Electronic Systems: Mitigation of Hardware Failures, Soft Errors, and Electro-Magnetic Disturbances", Springer Verlag, 2010.

[2] T.Heijmen, "Soft Errors from Space to Ground Historical Overview," Soft Errors in Modern Electronic Systems, Springer, 2011.

[3] M. Nicolaidis, "Design for soft error mitigation", *IEEE Trans. on Device and Materials Reliability*, vol. 5, no. 3, pp. 405–418, Sept. 2005.

[4] A. DeHon, N.Carter, H.Quinn, "Final Report for CCC Cross-Layer Reliability Visioning Study", Computing Community Consortium, http://www.cra.org/ccc/xlayer.php.

[5] G.C. Cardarilli, M. Ottavi, S. Pontarelli, M. Re and A. Salsano, "Fault tolerant solid state mass memory for space applications", *IEEE Trans. Aerosp. Electron. Syst.*, vol. 41, no. 4, pp. 1353-1372, Oct. 2005.

[6] W. K. Huang, Y.-N. Shen, and F. Lombardi, "New approaches for the repairs of memories with redundancy by row/column deletion for yield enhancement," *IEEE Trans. Computer-Aided Des. Integr. Circuits Syst.*, vol. 9, no. 3, pp. 323–328, Mar. 1990.

[7] C. L. Chen and M. Y. Hsiao, "Error-correcting codes for semiconductor memory applications: a state-of-the-art review", *IBM J. Res. Develop.*, vol. 28, no. 2, pp. 124-134, Mar. 1984.

[8] E. Fujiwara, "Code Design for Dependable Systems: Theory and Practical Application", John Wiley & Sons, Inc., 2006.

[9] M.Y. Hsiao "A class of optimal minimum odd-weight column SEC-DED codes", *IBM J. Res. Develop.*,vol. 14, no. 4, pp. 395–301, Jul. 1970.

[10] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo and T. Toba, "Impact of scaling on neutron-induced soft error rate in SRAMs From a 250 nm to a22 nm Design Rule", *IEEE Trans. on Electron Devices*, vol. 57 , no. 7, pp. 1527-1538,July2010.

[11] S. Lin and D. J. Costello, "Error Control Coding" (2nd Ed.), Englewood Cliffs, NJ: Prentice-Hall. 2004.

[12] R. Naseer and J. Draper, "DEC ECC design to improve memory reliability in sub-100nm technologies", in *Proc. of IEEE Int. Conf. on Electronics, Circuits, and Systems (ICECS)*, pp. 586-589, 2008.

[13] S. Ghosh and P. D. Lincoln, "Dynamic low-density parity check codes for fault-tolerant nano-scale memory", in *Proc. of Foundations of Nanoscience* (FNANO07), 2007.

[14] H. Naeimi and A. DeHon, "Fault secure encoder and decoder for nano Memory applications", *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 4, pp. 473-486, Apr.2009.

[15] S. Liu, P. Reviriego and J. A. Maestro, "Efficient majority logic fault detection with difference-set codes for memory applications", *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 1, pp. 148-156, Jan. 2012.

[16] P. Reviriego, M. Flanagan, S. Liu and J.A. Maestro, "Multiple Cell Upset Correction in Memories Using Difference Set Codes", *IEEE Trans. on Circuits and Systems I*, vol. 59, no 11, pp. 2592-2599, Nov. 2012.

[17] J. Guo, L. Xiao, Z. Mao and Q. Zhao, "Novel Mixed Codes for Multiple-Cell Upsets Mitigation in Static RAMs", *IEEE Micro*, vol.33, no.6, pp.66,74, Nov.-Dec. 2013.

[18] 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.

[19] J. Dénes and A. D. Keedwell, "Latin Squares and their Applications", Academic Press, 1974.

[20] S. E. Lee, Y. S. Yang, G.S Choi,W. Wu and R.Iyer, "Low-power, resilient interconnection with Orthogonal Latin Squares", *IEEE Des.& Test of Comput.*,vol.: 28, no. 2, pp. 30 – 39, Mar.2011.

[21] A. R. Alameldeen, Z. Chishti, C. Wilkerson, W. Wu, and S.-L. Lu "Adaptive cache design to enable reliable low-voltage operation", *IEEE Trans. on Comput.*, vol. 60, no. 1, pp. 50-63, Jan. 2011.

[22] R. Datta and N.A.Touba, "Generating burst-error correcting codes from Orthogonal Latin Square codes -- A graph theoretic approach",in *Proc.of IEEE Int.Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp. 367 – 373, 2011.

[23] P. Reviriego, S. Pontarelli, A. Sánchez-Macián, J.A. Maestro, "A Method to Extend Orthogonal Latin Square Codes", *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 7, July 2014, pp. 1635-1639.

[24] X. Yang and K.Mohanram, "Unequal-error-protection codes in SRAMs for mobile multimedia applications", *IEEE/ACM International Conference Computer-Aided Design*, pp.21-27, 7-10 Nov. 2011.

[25] J. Park, J. Park and S. Bhunia, "VL-ECC: Variable Data-Length Error Correction Code for Embedded Memory in DSP Applications", *IEEE Trans. on Circuits and Systems II*, vol.61, no.2, pp.120-124, Feb. 2014.

[26] I. Lee, J. Kwon, J. Park, and J. Park, "Priority based error correction code (ECC) for the embedded SRAM memories in H.264 system", *J. Signal Process. Syst.*, vol. 73, no. 2, pp. 123–136, Nov. 2013.

[27] K. Namba and Lombardi, F., "Parallel decodable two-level unequal burst error correcting codes", *IEEE Trans. Computers* (in press).

[28] XST User Guide for Virtex-6, Spartan-6 and 7 FPGAs UG687 (v14.5).