Implementing Triple Adjacent Error Correction in Double Error Correction Orthogonal Latin Squares Codes

P. Reviriego, S. Liu, J.A. Maestro
Universidad Antonio de Nebrija
Madrid, Spain
{previrie,sliu,jmaestro}@nebrija.es

S. Lee
Seoul National University of Science and Technology
Seoul, Korea organization
seung.lee@seoultech.ac.kr

N.A Touba, R.Datta
University of Texas at Austin
Austin, USA
touba@ece.utexas.edu,
rudrajit@utexas.edu

Abstract— **Soft errors have been a concern in memories for many years. In older technologies, soft errors typically affected a single memory cell but as technology scaled, Multiple Cell Upsets (MCUs) that affect a group of nearby cells have become more common. This trend is expected to continue making MCUs more frequent and also increasing the number of cells affected. To avoid data corruption in memories, Error Correction Codes (ECCs) are used. Single Error Correction (SEC) codes that can correct one bit error per word are effective only against single errors. To protect against MCUs, one option is to use more sophisticated error correction codes like for example, Orthogonal Latin Squares Codes (OLSC). In this paper, a modification of the OLSC decoding algorithm is proposed for codes that can correct two random errors. This modification has little impact on circuit complexity and enables triple adjacent error correction which is interesting when MCUs are present.**

Keywords — Error correction codes, majority logic decoding, memory, Multiple Cell Upsets (MCUs).

I. INTRODUCTION

Multiple Cell Upsets (MCUs) that corrupt more than one memory cell have become common in memories in recent years and are expected to increase as technology scales [1]. MCUs limit the effectiveness of Single Error Correction (SEC) codes [2] unless interleaving is used. With interleaving, the bits that belong to the same logical word are placed in cells that are physically apart [3]. Since the errors caused by an MCU are originated from a single particle hit, the cells affected are physically close [4]. Therefore interleaving can ensure that an MCU will corrupt only a single bit in each memory word. However, interleaving makes the routing of the memory more complex and can increase area and power consumption [3],[5]. In addition, interleaving cannot be used in small register sets and its use in content addressable memories is not practical [6].

An alternative to interleaving is the use of more advanced ECCs that can correct multiple errors per word. These codes require more complex encoders and decoders and also more parity bits per word. This increases the circuit cost and reduces memory speed as encoding and decoding take more time. In the last years, a number of works have proposed different alternatives to optimize the implementation of advanced ECCs in memories. As an example, in [7] a parallel decoder for Double Error Correction codes was proposed to minimize the decoding time. The use of Bose–Chaudhuri–Hocquenghem (BCH) codes combined with a Hamming code has also been studied in [8] to reduce the average decoding time by using the Hamming code to detect errors. The use of codes that are One Step Majority Logic Decodable (OS-MLD) [9] has also been proposed. OS-MLD enables a simple implementation of the decoder that reduces area and delay and therefore is interesting for memory applications. In [10], the use of a class of Euclidean Geometry codes that are OS-MLD was proposed. Further work on the use of these codes for memory protection was presented in [11]. Other OS-MLD codes like Difference Set codes have also been considered for memory protection [12]. One issue with both Euclidean Geometry and Difference Set Codes is that they provide a limited range of choices for the block sizes and error correction capabilities.

Orthogonal Latin Squares codes (OLSC) are a type of OS-MLD codes that have also been used to protect memories and interconnections [13],[14]. In contrast with Euclidean Geometry and Difference Set codes, OLSC provide a wide choice of error correction capabilities that can be implemented for each word size. They also support data word sizes which are a power of two as commonly used in memories. Recently, the correction of bursts of adjacent errors using OLSC has been studied in [15] . This is useful when MCUs are present as the errors will tend to be adjacent. In the method proposed in [15], additional parity bits are added to the OLSC to enable the correction of bursts of adjacent errors. This implies an area overhead in the memory implementation. In this paper, a method to modify the decoding of Double Error Correction (DEC) Orthogonal Latin Square (OLS) codes to correct bursts of adjacent errors is proposed. The method enables the correction of bursts of three adjacent errors and does not

978-1-4799-1585-9/13/$31.00

require adding more parity bits to the original OLS code. This is important as there is no overhead in terms of memory size to implement the burst error correction capabilities.

The rest of the paper is organized as follows. Section II introduces Orthogonal Latin Square codes and illustrates the OS-MLD decoding process with an example. The results for burst error correction in [15] are also summarized. In Section III, the proposed scheme is described. Section IV validates the proposed technique through simulation and evaluates its cost in terms of circuit area and delay. Finally, the conclusions from this work are presented in Section V.

II. Orthogonal Latin Squares Codes

A Latin square of size m is an $m * m$ matrix that has permutations of the digits $0,1,..m-1$ in both its rows and columns [16]. An example of a Latin square for $m = 4$ is shown in the following equation:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \\ 3 & 4 & 1 & 2 \\ 4 & 3 & 2 & 1 \end{bmatrix} \quad (1)$$

Two Latin squares are orthogonal if when they are superimposed every ordered pair of elements appears only once. Orthogonal Latin Squares Codes (OLSC) are derived from Orthogonal Latin squares [13]. These codes have $k=m^2$ data bits and $2tm$ check bits where t is the number of errors that the code can correct. For a Double Error Correction (DEC) code $t=2$ and therefore $4m$ check bits are used. One advantage of OLS codes is that their construction is modular. This means that to obtain a code that can correct $t+1$ errors, simply $2m$ check bits are added to the code that can correct t errors. This can be useful to implement adaptive error correction schemes as discussed in [14]. The modular property also enables the selection of the error correction capability for a given word size.

OLS codes can be decoded using One Step Majority Logic Decoding (OS-MLD) as each data bit participates in exactly $2t$ check bits and each other bit participates in at most one of those check bits. This enables a simple correction when the number of bits in error is t or less. The $2t$ check bits are recomputed and a majority vote is taken, if a value of one is obtained, the bit is in error and must be corrected. Otherwise the bit is correct. As long as the number of errors is t or less this ensures the error correction as the remaining $t-1$ errors can, in the worst case affect $t-1$ check bits so that still a majority of $t+1$ triggers the correction of an erroneous bit. The process of OS-MLD is illustrated in Figure 1 for a given bit d_i.

The parity check matrix for OLSCs is constructed from the Orthogonal Latin squares. As an example, the matrix for a code with $k = 16$ and 8 check bits that can correct single errors is shown in Figure 2. As discussed before, due to the modular construction of OLSCs this matrix will form part of the H matrix for codes that can correct more errors. For example, to obtain a code that can correct two errors eight additional rows and columns are added to the H matrix.

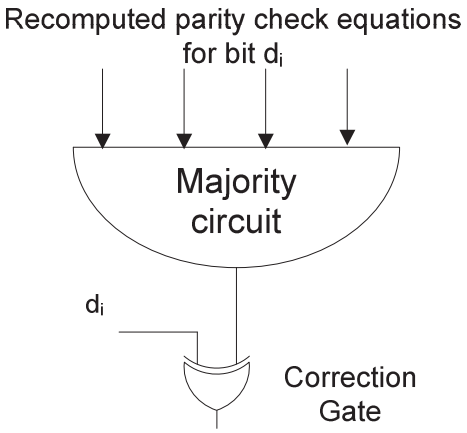

Fig. 1: OS-MLD of one bit in an OLS code.

For an arbitrary value of $k = m^2$, the H matrix for a SEC OLSC is constructed as follows:

$$H = \begin{bmatrix} M_1 \; I_{2m} \\ M_2 \end{bmatrix} \quad (2)$$

where I_{2m} is the identity matrix of size $2m$ and M_1, M_2 are matrixes of size $m * m^2$. The matrix M_1 has m ones in each row. For the r^{th} row the ones are at positions $(r-1) * m+1, (r-1) * m+2, \ldots (r-1) * m+m-1, (r-1) * m+m$. The matrix M_2 is constructed as follows:

$$M_2 = [I_m \quad I_m \quad \ldots \quad I_m] \quad (3)$$

For $m = 4$, the matrixes M_1 and M_2 can be clearly observed in Figure 2. As discussed in the next section, the structure of the parity checks in this matrix can be used to correct triple adjacent errors. Finally, to obtain a DEC code $2m$ additional check bits are added to the SEC code. Each data bit participates in exactly two of those new check bits.

The correction of bursts of adjacent errors using OLS codes has been recently studied in [15]. This is done by adding some additional check bits to the OLS codeword. The number of bits required for a double error correction code to implement also the correction of bursts of three adjacent errors is shown in Table I. It can be observed that the overhead is significant, especially for small block sizes. As discussed in the next section, the scheme proposed in this paper achieves the correction of three adjacent errors with no additional check bits thus lowering the implementation cost.

TABLE I
CHECK BITS REQUIRED FOR TRIPLE ADJACENT ERROR CORRECTION

k	OS-MLD	Modified	Overhead
16	16	20	25.00%
64	32	36	12.50%
256	64	67	4.69%

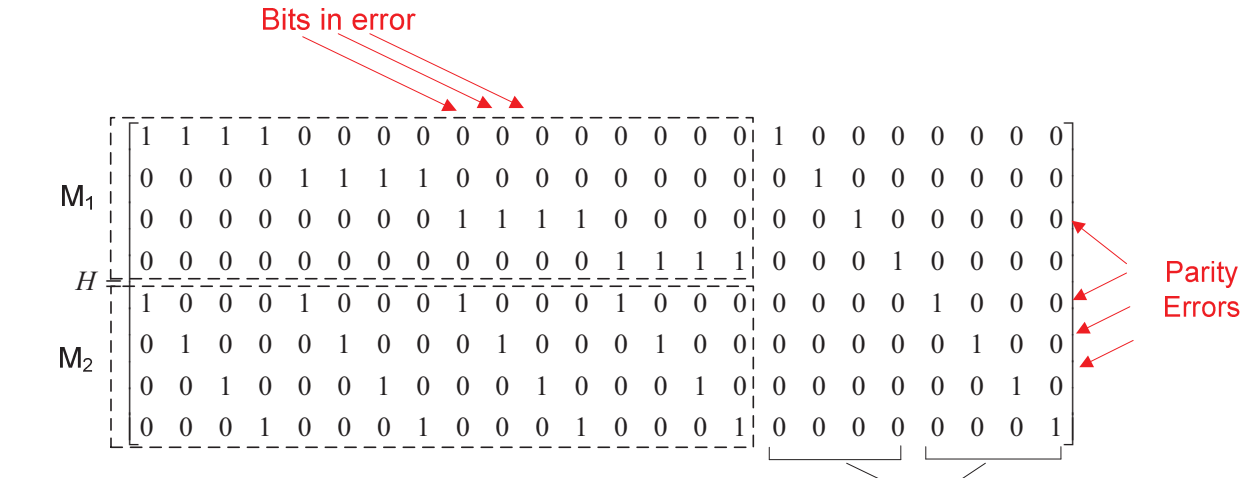

Fig. 2: Parity check matrix for the OLS code with $k = 16$ and $t=1$.

$$H_{proposed} = \begin{bmatrix}
1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0\\
0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&1\\
1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0\\
0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0&0\\
0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0&0\\
0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&1&0&0&0&0
\end{bmatrix}$$

Fig. 3: Modified parity check matrix for the OLS code with $k = 16$ and $t=1$.

III. Proposed Adjacent Error Correction Scheme

In this section, the modification to the decoding process to achieve triple adjacent error correction is described. Let us consider that a DEC OLS code with $k = 16$ is used and there are errors in data bits 9,10 and 11 as shown in Figure 2. When the check bits are recalculated, there will be errors in parity check bits 3,5,6 and 7. That is only one bit of the first four check bits (those in M_1) is in error and three of the second group (those in M_2) of four check bits are in error. This will be so for most triple adjacent errors given the structure of the first $2m$ rows in H matrix, as discussed in Section 2. Therefore, this pattern can be used to identify triple adjacent errors. The only problem is when the errors occur in the last data bits and the first check bits where the structure of the H matrix changes. For example, when errors occur in data bits 15 and 16 and the first check bit, there will be only two errors in the second group of four rows. To avoid this issue, the H matrix can be rearranged by interchanging the last columns as shown in Figure 3. In the memory implementation this would only mean that some bits are placed differently in the word. With this modification, the triple error previously considered will cause three errors in the second group of four rows. Therefore, three errors in the second group of four rows will be used to detect triple adjacent errors in the data bits. Error correction can be achieved as the patterns of those errors in the first $2m$ rows are different. For a general case of a code with $k = m^2$, a triple adjacent error affecting data bits i,i+1,i+2 will produce errors in the following rows:

a) if mod(i,m)= m-1 then the errors affect rows ceil(i/m)+1 and m+1,2m-1,2m.

b) if mod(i,m)= 0 then the errors affect rows i/m and m+1,m+2,2m.

c) in the rest of the cases the errors affect rows ceil(i/m) and m+mod(i,m), m+mod(i,m)+1, m+mod(i,m)+2.

The decoder is modified to detect those patterns in the recomputed check bits and correct the corresponding bits. In the previous analysis, errors in the check bits have not been considered. If we consider the H matrix in groups of m consecutive rows, a triple adjacent error in the check bits will affect at most two of those groups. This means that by the construction of the code, each data bit will be affected in the worst case in two of the parity checks that it participates. For a DEC code correction of a data bit will only take place when there are errors in a majority of the four checks in which the

bit participates. This ensures that a triple adjacent error in the check bits will not cause a miscorrection. Therefore, the proposed scheme can effectively correct triple adjacent errors.

The diagram of the modified decoding algorithm is shown in Figure 4. To minimize the decoding delay, the triple adjacent error detection (check for three ones) can be performed in parallel with the error location for both the standard OS-MLD and the Triple adjacent error correction blocks. Then the output of the triple adjacent error detection is used to select which error correction of the two is applied. This reduces the delay at the cost of increasing the power consumption.

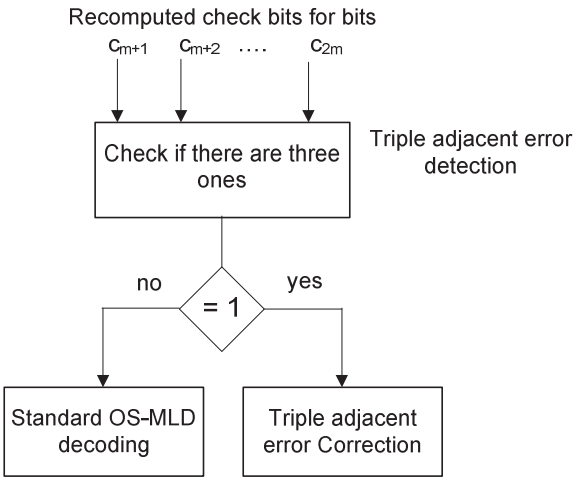

Fig. 4: Block diagram of the modified decoding algorithm.

The circuitry to correct each of the triple adjacent errors combinations is simple, as an example, the case of errors in the first three bits of the code with $k = 16$ is shown in Figure 5. This simplicity enables an implementation that requires a limited overhead when compared with a traditional OS-MLD decoder. In the next section, the implementation of the modified decoder is discussed in more detail.

The same scheme can be used for OLSCs that can correct three or more random errors, but in that case the correction of triple adjacent errors is of little use as those errors are already corrected by the traditional OS-MLD decoding. A more interesting observation is that the technique presented in this paper could be extended to correct larger bursts of errors for OLS codes that use larger word sizes and correct more than two random errors. For example for a code with *k=64* and *t=4*, bursts of five adjacent errors can be corrected using a modified decoder. The correction algorithm would be similar to the one discussed. In this case, there will be five errors in the second group of *m* rows of the H matrix and one in the first group. The error patterns are again unique for each combination and therefore, error correction can be easily implemented. Special care should be taken when the error occurs at the boundary between data and check bits. As with the scheme discussed, placing the check bits appropriately can ensure that those errors can also be corrected. This extension of the approach may be interesting for applications in which larger error correction capabilities are needed. The study of this extension of the proposed technique is left for future work.

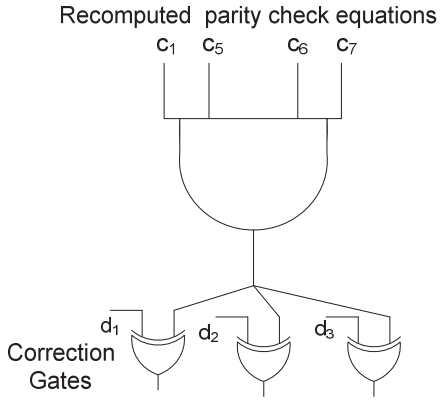

Fig. 5: Circuitry to correct a triple adjacent error on bits d_1,d_2,d_3.

IV. Evaluation

The proposed modification to the decoder has been evaluated both in terms of protection effectiveness and circuit complexity for the DEC OLS codes with $k = 16$, 64 and 256. The results are presented in the following subsections.

A. *Protection effectiveness*

The proposed scheme as described in Section III has been implemented and errors have been inserted to ensure that double random errors and triple adjacent errors are corrected. The data bits after the decoding are compared with the original data bits to detect errors that are not corrected. All possible combinations of double errors and triple adjacent errors have been exhaustively generated and tested. In all cases, the data bits after decoding are identical to the original data bits. Parity bits are not considered as they are not protected by the OLS code.

B. *Circuit complexity*

To evaluate the complexity of the modified decoder, it has been implemented in HDL and synthesized for a commercial 90nm library. The estimates of circuit area in μm^2 are shown in Table II. The decoder delay in nanoseconds is illustrated in Table III. In both cases, the results for a standard OS-MLD decoding are also included so that the overhead of the proposed modification is clear. The results show that the modified decoder has an impact in both area and delay. To minimize the impact on delay, error detection can be performed by checking if any of the parity check equations is in error. Then error correction is only done when an error is detected such that the average delay will be close to that of error detection [8], [12]. This delay was evaluated and was similar to that of the OS-MLD decoder. Therefore, the speed penalty can be effectively mitigated with this approach.

TABLE II
AREA OF THE DECODER IMPLEMENTATION

k	OS-MLD	Modified	Overhead
16	2852.04	3122.99	9.50%
64	8743.09	10665.14	21.98%
256	34166.56	41335.46	20.98%

TABLE III
DECODER SPEED

k	OS-MLD	Modified	Overhead
16	0.36	0.51	41.67%
64	0.41	0.57	39.02%
256	0.47	0.89	89.36%

V. CONCLUSIONS

In this paper, a modified decoder for Double Error Correction (DEC) Orthogonal Latin Squares (OLS) codes has been proposed. The proposed decoder does not require additional check bits and can correct triple adjacent errors in addition to double random errors. This can be useful when protecting memories that suffer Multiple Cell Upsets (MCUs). The proposed decoder has been evaluated both in terms of its error correction capabilities and its implementation cost. The results validate the theoretical analysis on error correction. The impact of the modification on circuit area although not negligible can be acceptable in many designs. The impact on decoding delay is larger but can be mitigated using a two step procedure in which errors are detected and only when there are errors the rest of the decoding is performed.

REFERENCES

[1] 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 a 22 nm Design Rule", *IEEE Trans. on Electron Devices*, vol. 57 , no. 7, pp. 1527-1538, July 2010.

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

[3] S. Baeg, S. Wen and R. Wong, "SRAM interleaving distance selection with a soft error failure model", *IEEE Transactions on Nuclear Science,* vol. 56, no. 4 (part 2), pp. 2111-2118, Aug. 2009.

[4] S. Satoh, Y. Tosaka, and S. A. Wender, "Geometric effect of multiple-bit soft errors induced by cosmic ray neutrons on DRAMs," *IEEE Electron Device Lett.*, vol. 21, no. 6, pp. 310–312, Jun. 2000.

[5] A. Dutta and N.A. Touba, "Multiple bit upset tolerant memory using a selective cycle avoidance based SEC-DED-DAEC code", *25th IEEE VLSI Test Symposium*, pp. 349–354, 2007.

[6] S. Baeg, S. Wen and R. Wong, "Minimizing soft errors in TCAM devices: a probabilistic approach to determining scrubbing intervals," *IEEE Trans on Circuits and Systems I*, vol. 57, no. 4, pp. 814–822, April 2010.

[7] R. Naseer and J. Draper, "DEC ECC design to improve memory reliability in Sub-100nm technologies", *Proc. of IEEE ICECS*, pp. 586-589, 2008.

[8] P. Ankolekar, S. Rosner, R. Isaac and J. Bredow, "Multi-bit error correction methods for latency-constrained flash memory systems," *IEEE Trans. on Device and Materials Reliability*, vol. 10, no. 1, pp. 33-39, 2010.

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

[10] 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), Snowbird, Utah, 2007.

[11] H. Naeimi and A. DeHon, "Fault secure encoder and decoder for nanoMemory applications", *IEEE Trans. on Very Large Scale Integration Systems*, vol. 17, no. 4, pp. 473-486, 2009.

[12] S. Liu, P. Reviriego and J. A. Maestro, "Efficient majority logic fault detection with difference-set codes for memory applications," *IEEE Trans. on Very Large Scale Integration Systems*, vol. 20, no. 1, pp 148-156, 2012.

[13] M.Y. Hsiao, D.C. Bossen, and R.T. Chien, ''Orthogonal Latin Square Codes,'' *IBM J. Research and Development*, vol. 14, no. 4, 1970, pp. 390-394.

[14] S. E. Lee, Y. S. Yang, G.S Choi, W. Wu and R. Iyer, "Low-Power, Resilient Interconnection with Orthogonal Latin Squares" *IEEE Design & Test of Computers*, vol.: 28, no. 2, pp. 30 – 39, 2011.

[15] R. Datta and N.A. Touba, "Generating Burst-Error Correcting Codes from Orthogonal Latin Square Codes -- A Graph Theoretic Approach" *IEEE International Symposium Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pp. 367 – 373, 2011.

[16] J. Dénes and A. D. Keedwell, "Latin squares and their applications" Academic Press, 1974.