International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 3 Issue 9, September- 2014

Concurrent Error Detection and Correction for Orthogonal Latin Squares Encoders and Syndrome Computation

Sudhakar Kongala
Department of ECE
Vidya Jyothi Institute of Technology
Hyderabad, India

D. Kishore Kumar
Department of ECE
Vidya Jyothi Institute of Technology
Hyderabad, India

***Abstract* – In advanced electronic circuits the reliability has been a major concern. There are number of mitigation techniques proposed to make sure that the errors do not affect the circuit functionality. Among them, to protect the memories and registers in electronic circuits Error Correction Codes (ECC) are commonly used. Whenever any ECC technique is used, the encoder and decoder circuit may also suffer errors. In this brief, concurrent error detection and correction technique for OLS encoders and syndrome computation is proposed and evaluated. The proposed method efficiently implements a parity prediction scheme that detects all errors that affect a single circuit node using the properties of OLS codes.**

Key Words— Concurrent error detection and correction, error correction codes (ECC), Latin squares, One step majority logic decoding (OS-MLD).

I. INTRODUCTION

Since many years, for detecting and correcting errors ECCs were used [1], [3]. Researchers have proposed wide range of codes for memory applications. For correcting one bit per word, single error correction (SEC) codes are used in general. Advanced codes which can also correct double adjacent errors [2] are also been studied. But the problem with some complex codes that corrects more errors is generally limited by their impact on delay and power, which in turn will limit their applicability to memory designs [4]. To run-over those concerns, a technique is proposed by the use
of codes that are one step majority logic decodable (OS-MLD). OS-MLD codes are low-latency decodable codes. So, for protecting memories, they are used [7] [8]. In [10], use of different types of codes has also been discussed. The other type of code that is OS-MLD is orthogonal latin squares (OLS) code [9].

For interconnections [11], memories [13], and caches [12] use of OLS codes have gained a renewed interest, because of the modularity such that error correcting capabilities can be easily adapted to the error rate [11] or to the mode of operation [13]. Typically more parity bits are required for OLS codes than other codes for correcting the same number of errors. However, due to their modularity and the simple, low delay decoding implementations (as OLS codes are OS-MLD); neutralize this disadvantage in many applications.

A major issue is that the encoder and decoder circuits needed to use (ECCs) may also suffer from error sometimes. Whenever, an encoder is affected by an error, to the memory an incorrect word may be written. In decoder, a correct word may be interpreted as erroneous or an incorrect word may be interpreted as a correct word.

Protection of the encoders and decoders for Different ECCs has been studied in [14] and [17]. For example, EG codes were studied in [8]. The protection of Reed-Solomon and Hamming Codes were studied in [14] and [15]. Finally, the protection of encoders for SEC codes against soft errors was discussed in [18].

The ECC encoder first computes the parity bits, and in majority of the cases the decoder detects and corrects the errors by checking the parity bits. In general, this is known as syndrome computation. In some codes, based on the properties of the code, serial encoding and syndrome computation are performed. But, for low delay parallel implementations are preferred. It is the case for OLS codes which are commonly used in high-speed applications. After syndrome computation, the errors are detected and corrected. This means in encoder and decoder generating and checking the parity bits are the important parts. Therefore, its important issue is its protection.

In this brief, for SRAM memories and caches, the protection of the encoders and syndrome computation for OLS codes are considered. Depending on some definite properties, it is presented that parity prediction is an productive technique to detect and correct errors in the encoder and syndrome computation. For most block codes it is not the case for which parity prediction will not provide effective protection. So, it is an advantage of OLS codes in addition to its modularity and low-decoding capability.

The rest of this paper is organized as follows. OLS codes are introduced and the some of the properties are summarized in Section II. The proposed parity prediction scheme is discussed in Section III. Evaluation of its time delay is discussed in Section IV. Finally, a brief conclusion is presented in Section V.

II. ORTHOGONAL LATIN SQUARES CODES

The concept behind OLS codes is the Latin Squares. A Latin square of size m is an m X m matrix that has permutations of 0, 1… m-1 digits in both its rows and columns [20]. Two Latin squares are orthogonal if when they are overlap every ordered pair of elements appears only once. OLS codes are derived from OLS [9]. These codes have m^2 data bits (k), where "m" is an integer. The 2tm check bits are required for correcting t-error correction, which means the H-matrix contains 2tm rows. The "t" and "m" are related as in (1). The modular property guarantees the error correction capability selection for a given word size.

As mentioned before, OLS codes can be decoded using OS-MLD as each bit participates in exactly 2t check bits and each other bit participates in at most one of those check bits participates in exactly 2t check bits. This enables a simple correction when the number of bits in error is t or less.

$$t(max) \leq [(m+1)/2] \qquad (1)$$

The majority vote is taken by recomputing the 2t check bits. When a value of "1" is obtained, the bit is in error and the check bits of duplicated circuit are taken. Otherwise, it is error free. In any case, the decoding process is started by recomputing the parity check bits and checking against the stored parity check bits.

OLS are used for the construction of the parity check matrix H for OLS codes. As a case, a matrix for a code with k=16 and 8 check bits that can correct single errors is shown in Fig. 1.

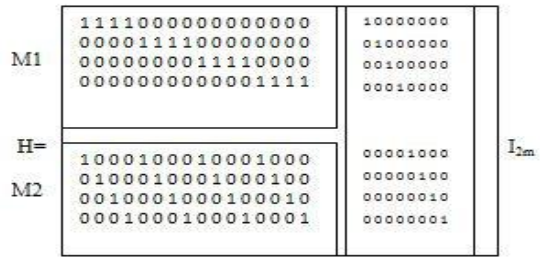

Fig 1: OLS code Parity check matrix with k=16 & t=1

As discussed earlier, due to the modular construction of OLS codes this matrix forms 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 are added to the H matrix.

The H-matrix for an arbitrary value of $k=m^2$ for a SEC-OLS code is constructed as follows:

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

Where I_{2m} is the identity matrix of size 2m and M1, M2 are all sub matrices of size m × m2. The matrix M1 has m ones in each row. The sub matrices in general have the following forms

$$M_1 = \begin{bmatrix} 11\ldots1 & & \\ & 11\ldots1 & \\ & & 11\ldots1 \end{bmatrix} \qquad (3)$$

The construction of matrix M2 is as follows:

$$M_2 = [I_m I_m \ldots I_m]. \qquad (4)$$

i.e., all are identity matrices of order m.

For m=4, the matrices M1 and M2 can be clearly observed in Fig. 1. The encoding matrix G is similar to the H matrix on which the check bits are removed.

$$G = \begin{bmatrix} M_1 \\ M_2 \end{bmatrix}. \qquad (5)$$

In summary, using the matrix G, for computing the 2tm check bits (ci), $k=m^2$ data bits are required for the encoder, which are adopted from the Latin squares and has the following properties.

1) One data bit participates exactly in 2t parity checks.

2) A pair of data bits participates (both bits) in at most one of the parity checks.

These properties are used in the following section to review the proposed technique.

III. PROPOSED CONCURRENT ERROR DETECTION AND CORRECTION TECHNIQUE

Before discussing the proposed error detection and correction techniques, the standard definition of self-checking circuits that are used in this section is presented. During fault-free or normal operation, a circuit receives only a subset of the input space, called the input code space, and produces a subset of the output space, called the output code space. The outputs that are not members of the output code space form the output error space. In general, a circuit may be designed to be self-checking only for an assumed fault set. In this brief, we consider the fault set F corresponding to the single stuck-at fault model [19].

Self-checking property is verified for the circuit if and only if it satisfies the following properties: 1) self-testing 2) fault secure.

A circuit is said to be running under self-testing if, in the fault set **F** for each fault f, there is at least one input which belongs to the input code space, for which circuit gives an output that belongs to the output error space.

A circuit is to have fault-secure if, in the fault set **F** for each fault f and for each input belonging to the input code space, the circuit gives the correct output, or an output that belongs to the output error space.

The fault-secure property assures that the circuit gives the correct response, or alerts the presence of a fault that provides an output in the error space. Faults are always detected, since an output is provided for the input which can identify the presence of the fault.

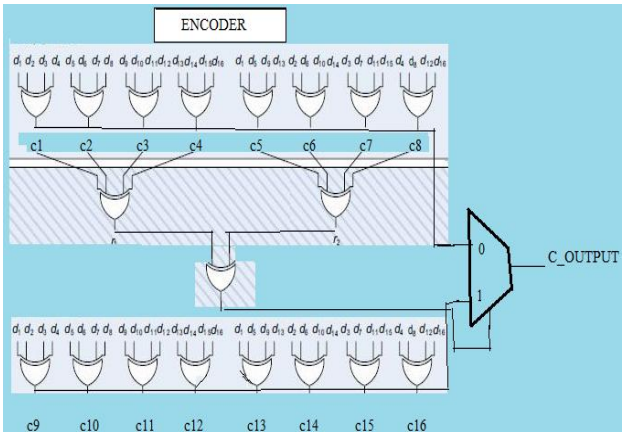

Fig 2: Proposed self-correcting encoder for OLS code with k=16 & t=1

The proposed technique is based on the use of parity prediction, which is one of the techniques that are used to detect and correct error in typical logic circuits. In our case, the problem is much simpler, given the structure of the OLS codes. For an encoder, the proposed is that the parity of the calculated check bits (ci) is set against the parity of all the check equations. The equation obtained by calculating the parity of the columns in G is simply the parity of all check equations. Since for OLS codes, each column in G will have exactly 2t ones, so null equation is obtained (see, for example, Fig. 1). Therefore, the concurrent error detection (CED) is normally to check

$$c1 \oplus c2 \oplus c3 \oplus \cdot \quad \cdot \quad \cdot \oplus c2tm = 0 \quad (6)$$

$$r1 \oplus r2 = 0 \quad (7)$$

and for Concurrent error detection and correction (CEDC) it is

$$c\_output = e\tilde{}(ci) + e(cj) \quad (8)$$

where ci are the check bits of the original circuit and cj represents the check bits of the duplicated circuit.

This gives an efficient implementation which is not possible in other ECC codes. For example, in the Hamming code a major part of the columns in G have an odd weight and for some different codes the number is even larger as they are designed to have odd weights and also in that codes we need to correct the codes manually, but in this case the codes are automatically corrected.

The input code space of the OLS encoder corresponds to the input space, since the encoder can receive all the possible 2k input configurations. The output code space of the OLS encoder is composed by the outputs satisfying (6), (7) and (8), while the output error space is the complement of the output code space.

In order to check whether the output of the OLS encoder belongs to the output code space or the output error space, a self-checking implementation of a parity checker is used [19]. The checker and correction unit controls the parity of its inputs and is realized with a repetition code. The two outputs (r1, r2) are each equal to the parity of one of two disjoint subsets of the checker inputs (ci), as proposed in [21]. When a set of inputs with the correct parity is provided, the output code {r1, r2} takes the values 00 or 11 and the two bits are compared against each other using a checker to get the output 0. When the checker receives an erroneous set of inputs, the checker provides the output codes 01 or 10 and so the output checker e will be equal to 1. Also, if a fault occurs in the checker, the outputs are 01 or 10 and the final checker output will be 1. This guarantees the self-checking property of the parity checker [21]. The proposed encoder is illustrated in Fig. 2 for the code with k = 16 and t = 1.

For correction of the errors in the encoder, first the circuit is partially duplicated till the generation of check bits by giving the same input combinations. Then the check bits(cj) of the duplicated circuit are compared against the check bits(ci) of the original (CED) circuit by using the 2X1 multiplexer, where the original circuit check bits (ci) are connected to the first input while the check bits (cj) obtained from duplicated circuit are connected to the second input of multiplexer and the selection bit for that is given from the output(e) of the original circuit, whenever the selection bit is 0, then the original circuit is selected and if it is not equal to 0 then the duplicated circuit gets selected.

The circuit that is proposed can detect and correct any error that affects an odd number of ci bits. For a general code, in most cases there is logic sharing among the calculations of the ci bits [18]. This means that an error may increase to more than one ci bit, and if the number of bits affected is even, then the error is also detected by the proposed scheme.

This means that an error may increase to more than one ci bit, and if the number of bits affected is even, then the error is also detected by the proposed scheme. But this would increase the area of the circuit and also increase the cost compared to an unrestricted implementation. Additionally, even if the error propagates to an odd number of outputs, the delay of each path can be different. Which may cause detecting of only some of the output errors at the clock edge. For OLS codes, as discussed in the previous section a pair of data bits shares at most one parity check. This assures that there isn't any logic sharing among the computation of the ci bits. Therefore, the proposed technique works well to detect and correct all errors that affect a single circuit node.

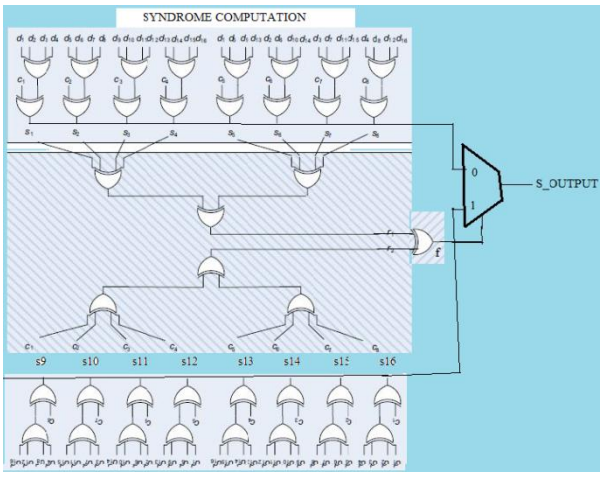

Fig 3: Proposed self-checking and correcting syndrome computation for OLS code with

k = 16 and t = 1.

The parity prediction for the syndrome computation can be implemented by checking that the following two equations take the same value.

$$r1 = s1 \oplus s2 \oplus s3 \oplus \cdot \cdot \cdot \oplus s2tm \quad (9)$$

$$r2 = c1 \oplus c2 \oplus c3 \oplus \cdot \cdot \cdot \oplus c2tm \quad (10)$$

Or else by simply checking the following equation

$$f = r1 \oplus r2 \quad (11)$$

where si bits are the calculated syndrome bits. The proposed circuit is shown in Fig. 3 for the code with k = 16 and t = 1.

If there are any errors in the computed syndrome bits, then for correcting them, the following equation is used:

$$S\_output = f\tilde{}(si) + f(sj) \quad (12)$$

Where si re the syndrome bits of the original circuit while sj are the duplicated circuit syndrome bits.

The output code space of the OLS syndrome computation is composed by the outputs given by (9), (10), (11) and (12), while the output error space is the complement of the output code space.

The fault-secure property for the syndrome computation is easily demonstrated for the faults in **F** by observing that the circuits that compute e do not share any gate and both circuits are only composed of XOR gates. Therefore, a single fault could propagate to only one of the outputs, producing an output on the output error space.

Though the circuits are different for encoder and syndrome computation, the syndrome computation circuit operation is similar to that of the operation of encoder, except that in encoder we calculate check bits, here in syndrome computation, syndrome bits are generated.

The duplicated circuit of syndrome computation circuit is that partial circuit of the original syndrome circuit i.e., upto the generation of the syndrome bits. The inputs of the duplicated circuit are taken from the original inputs bits.

For OLS codes, the cost of the encoder and syndrome computation in terms of the number of two-input XOR gates can be easily calculated (as each l−input XOR gate is assumed to be equivalent to l − 1 two-input XOR gates). There are 2tm check bits for a code with $k = m^2$ and that can correct t errors, and the computation of each of them requires m−1 two input XOR gates. Therefore, the encoder requires 2tm (m − 1) two-input XOR gates. For syndrome computation, an additional XOR gate is to be needed for each parity check bit, giving a total number to 2tm two-input XOR gates.

The proposed method requires 4tm−2 two-input XOR gates for the encoder and 8tm-4 two-input XOR gates for the syndrome computation. This means that the overhead required to implement

Our method for the encoder is

$$O_{encoder} = \frac{(4tm-2)}{(2tm\,(m-1))} \quad (13)$$

and for the syndrome computation is

$$O_{syndrome} = \frac{(8tm-4)}{(2tm^2)} \quad (14)$$

For larger values of m, (13) can be approximated by 2/m and (14) can be approximated by 4/m. This shows that whenever the block size of a code $k=m^2$ grows; the overhead will become smaller and independent of t.

TABLE I
OVERHEAD OF PROPOSED CEDC FOR SEC-OLS CODES

k	m	Encoder	Syndrome
16	4	58.33%	87.50%
64	8	26.78%	46.87%
256	16	12.91%	24.21%

The overhead in terms of delay is important as the proposed parity computation is done over the results of the encoder or syndrome computation. That means the delay of the new logic adds to the encoding or syndrome computation delay directly. However, the effect on memory access time is reduced by noting that for the encoder the error checking and correction can be done in parallel along with the writing of the data into the memory. For the syndrome computation, the error detection and correction can be performed in parallel with the majority logic voting and so the impact on access time is minimized.

TABLE II
DELAY ESTIMATES FOR ENCODER (in ns)

k	m	unprotected	With CEDC	overhead
16	4	6.236	9.028	44.7
64	8	6.579	9.401	42.89
256	16	6.771	9.783	44.48

TABLE III
DELAY ESTIMATES FOR SYNDROME COMPUTATION (in ns)

k	m	unprotected	With CEDC	overhead
16	4	6.446	9.346	44.98
64	8	6.574	9.544	45.17
256	16	6.785	9.801	44.45

Finally, it is worth noting that the overhead of implementing the proposed technique for other codes is much larger. This is because the CEDC check becomes more complex and also because logic sharing must also be avoided in the parity check computation.

IV. EVALUATION

The proposed CEDC mechanisms have been implemented in Verilog for codes with t = 1 and the values of k used in Table I. The designs have been implemented in Xilinx design compiler for synthesis and simulation.

From the Table I, it can be observed that the as the number of bits are increased and so the overhead has also gradually reduced, and for calculating the impact on delay, synthesis is run on delay optimization. The results are shown in Tables II and III. From that it is observed that the proposed technique introduces a significant delay. This is expected from the discussion in the previous section. However, the impact on memory access time can be largely

reduced by performing error checking and correction in parallel with the writing of the data for the encoder. For the syndrome computation, the checking and correcting can be done in parallel with the majority voting and error correction.

It is mentioned in the previous sections that the circuits of encoder and decoder are duplicated for correction of the check bits or syndrome bits. This results in a significant area increase. As an example, from the previous work, an encoder for a parity extended Hamming code with k = 32 implementation with and without logic sharing is considered.

In that the area overhead of avoiding logic sharing was 35%. The cost of the checker is also larger for Hamming. In this particular case, the total overhead for the proposed scheme would be over 80%. This confirms that the proposed technique is not effective in a general case and relies on the properties of OLS codes to achieve an efficient implementation.

V. CONCLUSION

In this brief, a CEDC technique for OLS codes encoders and syndrome computation is proposed. The proposed technique used the properties of OLS codes to design a parity prediction scheme that could be efficiently implemented and detect and correct all errors that affect a single circuit node.

Different word sizes are being evaluated using this technique, which showed that for large words the overhead is small. This is interesting as large word sizes are used, for example, in caches for which OLS codes have been recently proposed [13]. The proposed error checking and correcting scheme required a significant delay; however, its impact on access memory time could be minimized. This was achieved by performing the checking and correcting in parallel with the writing of the data in the case of the encoder and in parallel with the majority voting and error correction in the case of the decoder. In a general case, as the proposed technique requires a larger overhead as majority of ECCs do not have the properties of OLS codes. This limited the applicability of the proposed CED scheme to OLS codes. The availability of low overhead error detection and correction techniques for the encoder and syndrome computation is an additional reason to consider the use of OLS codes in high-speed memories and caches.

REFERENCES

[1] C. L. Chen, "Error-correcting codes for semiconductor memory," IBM in IEEE. pp. 245–247, Mar. 1984.
[2] Jushwanth Xavier. X, Benujah. B.R, "Multiple bit upset deduction/correction for memory applications," in IJAIS., February 2013, pp. 15-18.
[3] Eiji Fujiwara, Code Design for Dependable Systems: Theory and Practical Application. New York: Wiley, 2006.
[4] Salvatore Pontarelli, G. C. Cardarilli, M. Ottavi, 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.
[5] E. J. McCluskey, "Design techniques for testable embedded error checkers," IEEE Computer, vol. 23, no. 7, pp. 84–88, Jul. 1990.
[6] Lin Shu, Shu Lin and Daniel. J. Costello, Error Control Coding, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2004.
[7] S. Ghosh and P. D. Lincoln, "Dynamic low-density parity check codes for fault-tolerant nano-scale memory," in Proc. Found. Nanosci., 2007, pp. 1–5.
[8] H Naeimi and A. DeHon, "Corrections to Fault Secure Encoder and Decoder for NanoMemory Applications," .
[9] 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.
[10] Shu Liu, Pedro 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.
[11] 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 Comput., vol. 28, no. 2, pp. 30–39, Mar.–Apr. 2011.
[12] A.R.Alameldeen, Z. Chishti, W. Wu, C. Wilkerson, and S.-L. Lu, "Adaptive cache design to enable reliable low-voltage operation," IEEE Trans. Comput., vol. 60, no. 1, pp. 50–63, Jan. 2011.
[13] R. Datta and N. A. Touba, "Generating burst-error correcting codes from orthogonal latin square codes–a graph theoretic approach," in Proc. IEEE Int. Symp. Defect Fault Tolerance VLSI Nanotechnol. Syst., Oct. 2011, pp. 367–373.
[14] H. Jaber, F. Monteiro, S. J. Piestrak, and A. Dandache, "Design of parallel fault-secure encoders for systematic cyclic block transmission codes," Microelectron. J., vol. 40, no. 12, pp. 1686–1697, Dec. 2009.
[15] Andre Neubauer, Volker Kuhn, Jurgen Freudenberger,"Coding Theory: Algorithms, Architectures and Applications"
[16] G. C. Cardarilli, S. Pontarelli, M. Re, and A. Salsano, "Concurrent error detection in Reed-Solomon encoders and decoders," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 15, no. 7, pp. 842–846, Jul. 2007.
[17] Houssein Jaber, J. Piestrak, Stanislaw, Fabrice Monteiro and Abbas Dandache, A. Dandache, and F. Monteiro, "Design of parallel fault-secure encoders for systematic cyclic block transmission code," Microelectronics Journal, Vol 40, Issue 12, Dec, 2009.
[18] Antonio Maestro, C. Argyrides, Pedro Reviriego and D. K. Pradhan, "Fault tolerant single error correction encoders," J. Electron. Test., Theory Appl., vol. 27, no. 2, pp. 215–219, Apr. 2011.
[19] Parag. K. Lala, Self Checking and Fault Tolerant Digital Design: Morgan Kaufmann, 2001.
[20] J. Dénes and A. D. Keedwell, Latin Squares and Their Applications. San Francisco, CA: Academic, 1974.