HDL IMPLEMENTATION OF SRAM BASED ERROR CORRECTION AND DETECTION USING ORTHOGONAL LATIN SQUARE CODES

[(1)]**Nallaparaju Sneha, PG Scholar in VLSI Design,**
[(2)]**Dr. K. Babulu, Professor, ECE Department,**
[(1)(2)]UCEK, JNTUK, Kakinada

Abstract: Reliability is a major concern in advanced electronic circuits. To ensure that errors do not affect the circuit functionality a number of mitigation techniques can be used. Among them, Error Correction Codes (ECC) are commonly used to protect memories and registers in electronic circuits. When ECCs are used, it is of interest that in addition to correcting a given number of errors, the code can also detect errors exceeding that number. This ensures that uncorrectable errors are detected and therefore silent data corruption does not occur. Among the ECCs used to protect circuits, one option is Orthogonal Latin Squares (OLS) codes for which encoding and decoding can be efficiently implemented. In this paper, an HDL implementation of SRAM based error detection and correction using OLS CODES is designed. The proposed design is applied to SRAM and reduces the probability of silent data corruption by implementing mechanisms to detect errors that affect two bits.

***Keywords*—**Concurrent error detection, error correction codes (ECC), Latin squares, majority logic decoding (MLD), parity, memory.

1. INTRODUCTION

The general idea for achieving error detection and correction is to add some redundancy which means to add some extra data to a message, which receiver can use to check uniformity of the delivered message, and to pick up data determined to be corrupt. Error-detection and correction scheme may be systematic or it may be non-systematic. In the system of the module non-systematic code, an encoded is achieved by transformation of the message which has least possibility of number of bits present in the message which is being converted. Another classification is the type of systematic module unique data is sent by the transmitter which is attached by a fixed number of parity data like check bits that obtained from the data bits. The receiver applies the same algorithm when only detection of the error is required to the received data bits which is then compared with its output with the receive check bits if the values does not match, there we conclude that an error has crept at some point in the process of transmission. Error correcting codes are regularly used in lower-layer communication, as well as for reliable storage in media such as CDs, DVDs, hard disks and RAM.

Provision against soft errors that apparent they as the bit-flips in memory is the main motto of error detection and correction. Several techniques are used present to midi gate upsets in memories. For example, the Bose – Chaudhuri–Hocquenghem codes, Reed–Solomon codes, punctured difference set codes, and matrix codes has been used to contact with MCUs in memories. But the above codes mentioned requires more area, power, and delay overheads since the encoding and

decoding circuits are more complex in these complicated codes. Reed-Muller code is another protection code that is able to detect and correct additional error besides a Hamming code. But the major drawback of this protection code is the more area it requires and the power penalties.

Reliability is a major issue for advanced electronic circuits. As technology scales, circuits become more vulnerable to error sources such as noise and radiation and also to manufacturing defects and process variations. A number of error mitigation techniques can be used to ensure that errors do not compromise the circuit functionality. Among those, Error Correction Codes (ECCs) are commonly used to protect memories or registers. Traditionally, Single Error Correction (SEC) codes that can correct one bit error in a word are used as they are simple to implement and require few additional bits. A SEC code requires a minimum Hamming distance between code-words of three. This means that if a double error occurs, the erroneous word can be at distance of one from another valid word. In that case, the decoder will miss-correct the word creating an undetected error. To avoid this issue, Single Error Correction Double Error Detection (SEC-DED) codes can be used. Those codes have a minimum Hamming distance of four. Therefore, a double error can in the worst case cause the word to be at a distance of two of any other valid word so that miss-correction is not possible. More generally, for a code that can correct t errors, it is of interest to also detect t+1 error. This reduces the probability of undetected errors that can cause Silent Data Corruption (SDC). SDC is especially dangerous as the system continues its operation unaware of the error and this can lead to further data corruption or to an erroneous behavior long after the original error occurred.

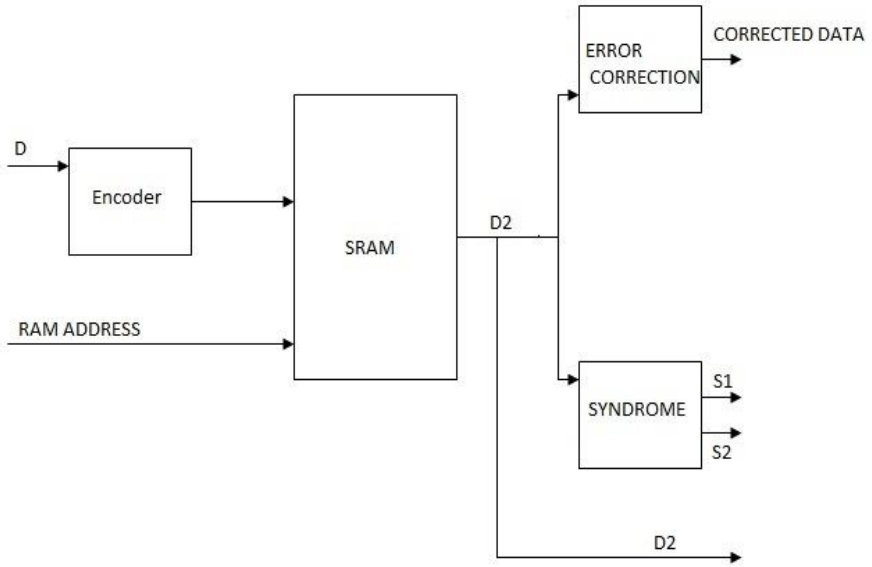

Fig 1: Design Block diagram

2. ORTHOGONAL LATIN SQUARE CODES

The concept of Latin squares and their applications are well known [12]. 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. For each value of m there can be more than one Latin square. When that is the case, two Latin squares are said to be orthogonal if when they are superimposed every ordered pair of elements appears only once. Orthogonal Latin Squares (OLS) codes are derived from Orthogonal Latin squares [9]. These codes have k=m2 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. The modular property enables the selection of the error correction capability for a given word size. As mentioned in the introduction, 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. For an OLS code that can correct t errors using OS-MLD, t+1 error can cause miss-corrections. This occurs for example if the errors affect t+1 parity bits in which bit di participates as this bit will be miss-corrected. The same occurs when the number of errors is larger than t+1. Each of the 2t check bits in which a data bit participates is taken from a group of m parity bits. Those groups are bits 1 to m, m+1 to 2m, 2m+1 to 3m and 3m+1 to 4m.

3. DESIGN ASPECTS

3.1 PARITY CHECK MATRIX

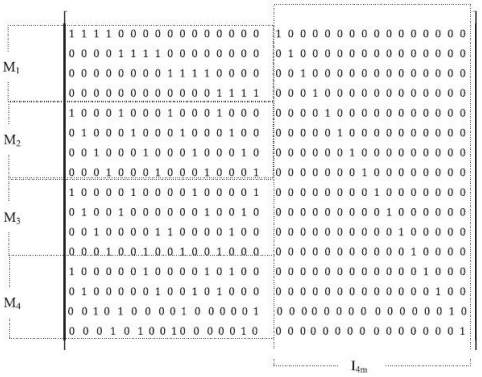

Fig 2: Parity check matrix for OLS code having k=16 and t=2

The H‟ matrix for OLS codes is build from their properties. The matrix is capable of correcting single type error. By the fact that in direction of the modular structure it might be able to correct many errors. They have check bits of number “2tm” in which, „t‟ stands for numeral of errors such that code corrects. If we wanted to correct a double bit then we have „2‟ as the value of t and thereby the check bits required are 4m.the H matrix, of Single Error Code „OLS‟ code is construct as :

$$H = \begin{bmatrix} M_1 & I_{2m} \\ M_2 & \end{bmatrix} \qquad (1)$$

a. In the above, I2m is the identity matrix of size 2 m.

b. M1, M2 is the matrices of given size m × m2.

„‟The matrix M1 have m ones in respective rows. For the rth row, the 1‟s are at the position (r − 1) × m + 1,(r − 1)× m +

2,…….(r − 1) × m+ m − 1, (r − 1) × m + m". The matrix M2 is structured as:

M2 = [Im Im . . . Im] **(2)**

For the given value 4 for m, the matrices M1 and M2 can be evidently experiential in Fig. H Matrix in the check bits we remove is evidently the G Matrix:

$$G = \begin{bmatrix} M_1 \\ M_2 \end{bmatrix} \qquad (3)$$

On concluding the above mentioned, it is evident that the encoder is intriguing m2 data bits and computing 2tm parity check bits by using G matrix . These resulted from the Latin Squares have the below properties:

a. Exactly in 2t parity checks each info bit is involved.

b. Utmost one in parity check bits info bits takes participation. We use the above properties in the later section to examine our proposed technique.

The following are the modules included in the design:

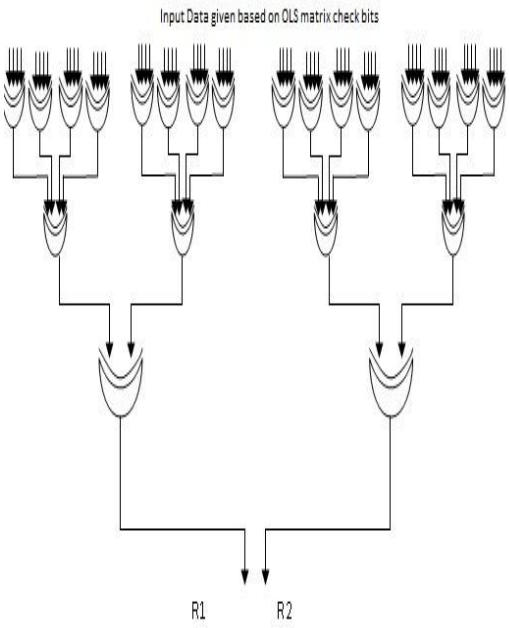

Fig 3: Logic diagram for Encoding

3.2 ENCODER:

For every check bit, the corresponding data bits in the matrix shown in fig 2 are compared and check bit is generated.

Suppose if odd number of ones are present check bit will be 1, if even number of ones are present check bit will be 0. In encoder we are generating parity bits for the given data.

3.3 DECODER:

Decoding consists of syndrome computation and error-correcting block. In decoder the encoded data from SRAM is given to syndrome computation where error is detected and then it is given to error correction block where error is corrected if error had occurred in syndrome computation.

3.4 SYNDROME COMPUTATION:

Data collected from the encoder contains information bits and the check bits. For easy computation check bits are divided into four groups as C0-C3, C4-C7, C8-C11, and C12-C15. The output obtained from the

encoder is given as an input to the syndrome computation which is compared with the check bits. If both are equal, no error has been detected. Syndrome computation is shown in Fig 6.

3.5 ERROR CORRECTION:

OS-MLD is a simple procedure in which each bit is decoded by simply taking the majority value of the set of the recomputed parity check equations, in which it participates [6]. This is shown in fig 4 for a given data bit *di*. The reasoning behind OS-MLD is that when an error occurs in bit *di*, the recomputed parity checks in which it participates will take a value of one. Therefore, a majority of ones in those recomputed checks is an indication that the bit is in error and therefore needs to be corrected. However, it may also occur that errors in other bits different from *di* provoke a majority of ones that would cause mis-correction. For a few codes, their properties ensure that this mis-correction cannot occur, and therefore OS-MLD can be used. For example, the first column that corresponds to the first data bit has ones in positions 0, 4, 8, and 12. For OLS codes, as described before, the decoding is done by taking a majority vote of the syndrome bits in which the bit participates (0, 4, 8, and 12 in our example).

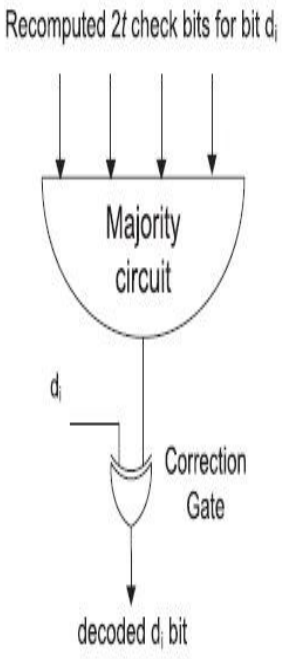

Fig 4: Error correction block

If the majority is one, then the data bit is in error and is corrected by inverting the bit.
In the example of Fig.2, all the data bits (first 16 columns) participate in exactly four parity bits *(2t)* and each pair of columns share at most one position with a value of one.

3.6 SRAM:

SRAM is defined as static random access memory. It is type of semi conductor memory that uses bi-stable latching circuitry to store each bit. It is faster and more expensive than DRAM. It requires less power. In SRAM all the operations are controlled in control logic. When reset is equal to zero, whatever data from encoder is written to SRAM. In SRAM whenever read enable is one the encoded in data SRAM is given to decoder.

SRAM contains three states:

1. Write state
2. Read state and

3. Stable state

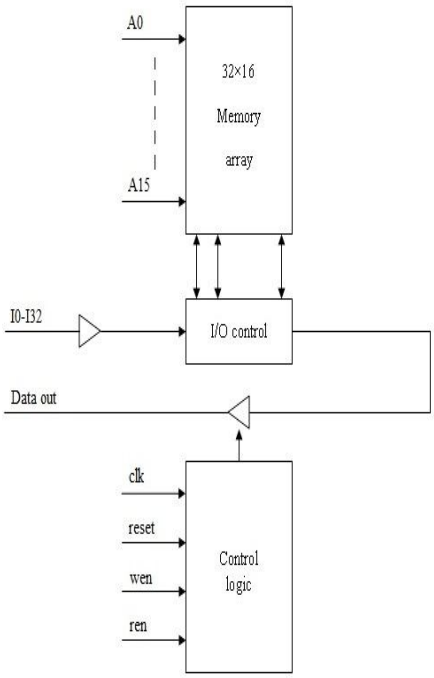

Fig 5: SRAM Block diagram

SRAM is random access memory that retains data bits in its memory as long as power is supplied unlike DRAM. SRAM does not have to be periodically refreshing circuit. Block diagram of SRAM is shown in Fig 5. In this project we have corrected and detected the errors occurred in data of SRAM.

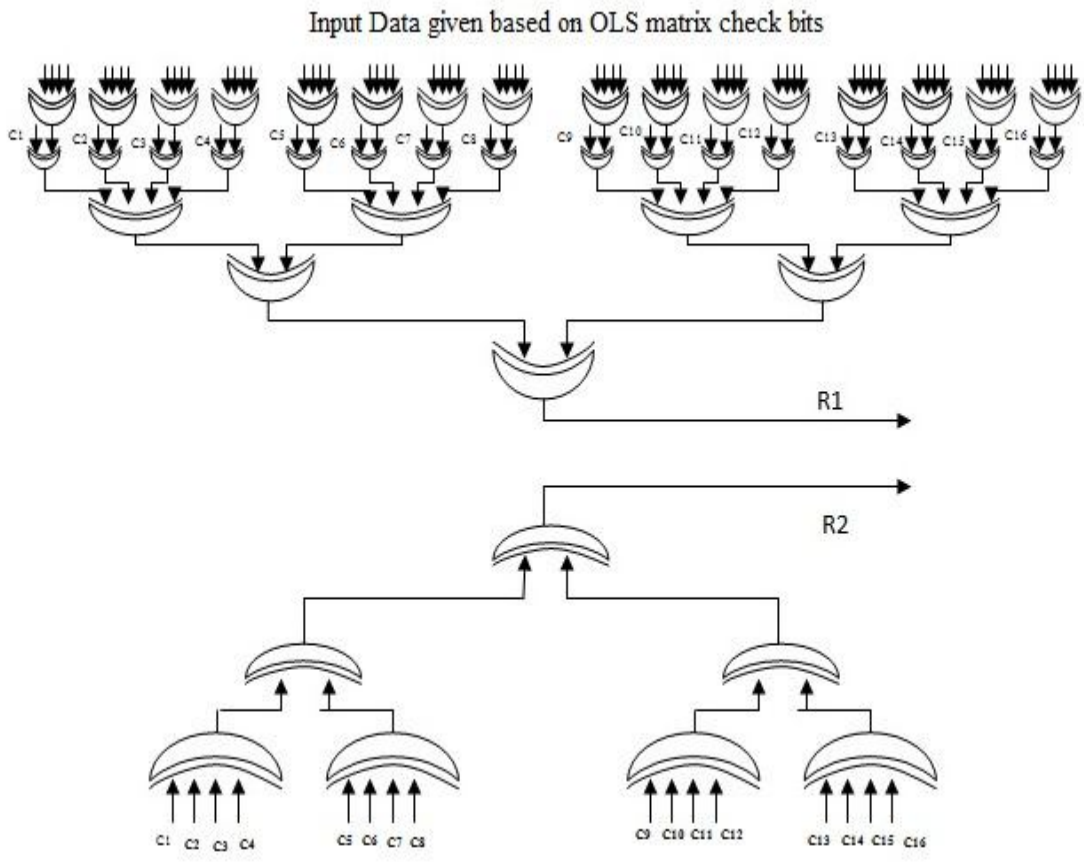

Fig 6: Syndrome Computation

4. SIMULATION RESULTS

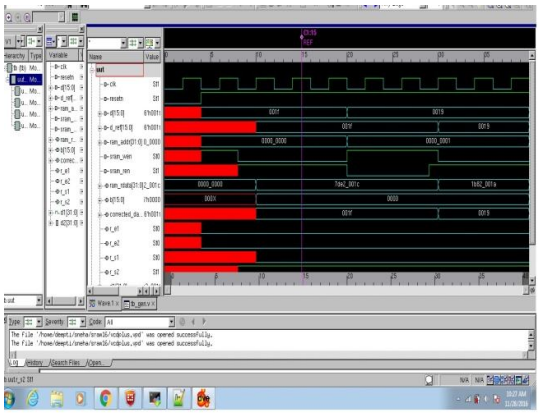

Fig 7: Waveform for data with error

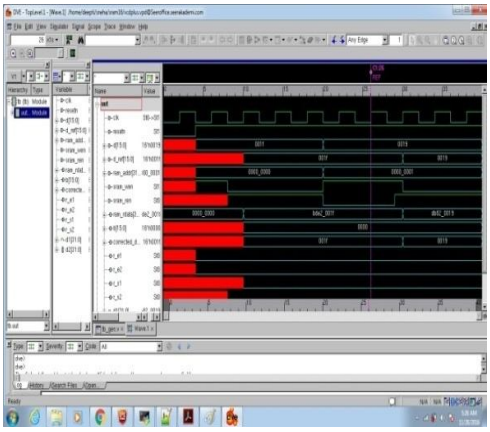

Fig 8: Waveform for data without any error

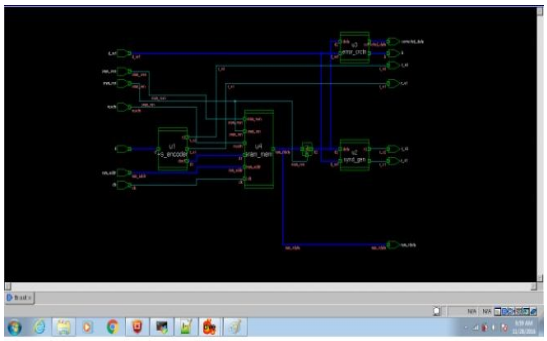

Fig 9: Top schematic

The efficiency of the work is analyzed using synopsis tools for both simulation and synthesis report. In Fig.7 and 8 it shows that error in data and without error in data.
Fig. 9 is schematic of the top design.

IMPLEMENTATION

REPORTS	k=16
AREA	12868.71nms
POWER	63µw
TIMING	19.65ns

Table 1: Synthesis Report

The above tabular form represents Synthesis Report about the Area, Delay and Power Consumption for top design by Synopsis tool.

5. CONCLUSION

In this brief, a method to extend OLS codes has been proposed. The method has been used to derive extended double ECCs for different block sizes. The extended codes have the same number of parity bits as the original OLS codes but a larger number of data bits. Therefore, the relative overhead is smaller. In encoder we generate parity bits, which are written into SRAM and then the output is given as input to the decoder. There error is detected in the syndrome computation and it is corrected in the error-correction block. In any case, as discussed in this brief, the proposed method is expected to provide better benefits for double ECCs.

6. REFERENCES

[1] 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.
[2] E. Fujiwara, Code Design for Dependable Systems: Theory and Practical Application. New York: Wiley, 2006.
[3] A. Dutta and N. A. Touba, "Multiple bit upset tolerant memory using a selective cycle avoidance based SEC-DED-DAEC code," in Proc. IEEE VLSI Test Symp., May 2007, pp. 349–354.
[4] R. Naseer and J. Draper, "DEC ECC design to improve memory reliability in sub -100nm technologies," in Proc. IEEE Int. Conf. Electron., Circuits, Syst., Sep. 2008, pp. 586–589.
[5] G. C. Cardarilli, M. Ottavi, S. Pontarelli, M. Re, and A. Salsano, "Fault tolerant solid state mass memory for space appl ications," IEEE Trans. Aerosp. Electron. Syst., vol. 41, no. 4, pp. 1353–1372, Oct. 2005.
[6] S. Lin and D. 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, "Fault secure encoder and decoder for nanoMemory applications," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 17, no. 4, pp. 473–486, Apr. 2009.