ISSN: 2395-1680 (ONLINE)
DOI: 10.21917/ijme.2019.0131

VLSI ARCHITECTURE FOR ERROR DETECTION AND CORRECTION BASED ON XOR AGAINST MULTIPLE CELL UPSETS WITH REDUCED REDUNDANT BITS

V. Bhanumathi and M. Sunandini

Department of Electronics and Communication Engineering, Anna University Regional Campus, Coimbatore, India

Abstract

Memories are in general protected with error correction codes per word in order to improve its reliability. The errors introduced by the radiation particles on memories will affect more than one cell leading to what is called as Multiple Cell Upsets (MCUs). As technology is scaled down, MCUs become a more problematic one in SRAM memory, because MCUs flip the logical state in memory thereby affecting its reliability by introducing errors. The existing Error Correction Codes (ECC) such as Matrix Code (MC), Punctured Difference Set code (PDS) and Decimal Matrix Code (DMC) are lagging in the number of bits that it can correct and also it utilizes more redundant bits for detection. Hence for detecting and correcting the consecutive errors as well as for reducing the redundant bits, we propose here VLSI architecture based on a simple XOR operation over the least significant bits. It is understood from the simulation analysis that the proposed architecture achieves low area, power, and delay with an improved capability of error correction and detection. The proposed design results in twice the number of corrected errors as that of DMC.

Keywords:

Multiple Cell Upsets, Static Random Access Memory, Exclusive-OR, Error Detection and Correction

1. INTRODUCTION

Multiple Cell Upsets (MCUs) are like a single event that induces several bits in an integrated circuit to fail at the same time. It affects mostly Static Random Access Memory (SRAM). The MCUs occur due to radiation particle striking the memory and the neutrons penetrate into the SRAM memory. Due to this, electron hole pair generation will take place resulting in an accumulation of the charges in the memory. When the charges exceed the critical charge limit, then it can flip the logical state in the memory [1]. It is stated that neutron irradiation reduced the single event latch-up and the sensitivity of CMOS SRAM [2].

Some error corrections codes like hamming [3], Triple error correction [4], Bose Chaudhuri Hocquenghem [5], Reed Solomon [6] and other codes [7], [8] are proposed to deal with the problems in memories. But these codes had a correction capability of up to 2 bits only. These codes also consumed more area, power and delay overheads. Some interleaving techniques are also used to restrain the MCUs but it is not possible in the case of Content Addressable Memory (CAM). The technique in [9] rearranged the cells to separate the bits present in the logical word. The new mix codes [10] are presented for fault - secure memories to overcome multiple bit upsets mitigation. The tight coupling of hardware structures in interleaving technique was not possible for CAM [11]. Built in Current Sensor (BICS) [12], [13] improved reliability of SRAM memory by coupling BICS with H-Tree architecture, but it can correct only two errors in word. Column-Line-Code (CLC) presented in [14] was for the detection and correction of MCU in memory devices. It utilized Hamming and parity bits.

Matrix Code (MC) [15] combined hamming and parity code to protect SRAM memory. It performed better than Hamming. MC corrects MCUs per word with lower decoding delay. In MC two bit errors can be detected by hamming, but these errors can be corrected only when two vertical syndrome bits were activated.

Hamming code combined with decimal algorithm [16] to detect and correct soft errors provided low delay overhead by the introduction of integer values. Hamming code did error detection and correction by generating parity codes [17]. The error correction for TCAM and for 130nm-180nm RAMs are also presented in [18] [19]. An enhancement of memory reliability of Decimal Matrix Code (DMC) than hamming based single error correction with double error detection [3] and matrix code is presented in [20]. A survey on DMC is given in [21].

DMC uses three methods to improve reliability and performance. First method is divide symbol to improve the reliability. If the information bits are 32-bit, the divide symbol method divides it into two rows and four columns. The second is decimal algorithm to detect errors. In decimal algorithm, both the decimal integer addition and subtraction was done to maximize error detection capability thereby enhancing the reliability of memory. The third method was Encoder Reuse Technique (ERT) in which the encoder circuit is used in decoder for minimizing the area overhead without disturbing the entire encoding and decoding process. DMC provided the memory reliability by detecting and correcting up to 7 bits. It is better than hamming and matrix. These two codes can correct up to two errors only and also a reduction in the area overhead is achieved with the help of ERT.

The decoding delay complexity was also reduced than PDS codes but it utilized more redundant bits. For a 32-bit information, a total of 36 redundant bits are needed in which 20 bits for horizontal redundancy and 16 bits for vertical redundancy. Another drawback was that it cannot detect consecutive MCUs in Symbol 0 and Symbol 2 resulting in an erroneous data. The drawback of DMC is overcome by the proposed bit manipulation based architecture. The main aim is to reduce the redundant bits and to detect the consecutive MCUs in symbol 0 and 2. It proves without the usage of decimal algorithm concept, the detection of MCUs can be processed and also the number of redundant bits can be reduced.

This paper is organized as follows. The overview of error detection and correction is discussed in section 2. The proposed system encoder and decoder are presented in section 3. The overhead analysis of the proposed code is analyzed in section 4. The result analysis for 32-bit and 64-bit data are described in section 5. Finally, conclusion of this paper is shared in section 6.

2. OVERVIEW OF ERROR DETECTION AND CORRECTION

The general idea of achieving error detection and correction is to add some redundancy to a message, by which receivers can use to check consistency of the delivered message, and to recover data determined to be corrupted. Error-detection and correction schemes can be either systematic or non-systematic. In a systematic scheme, the transmitter sends the original data, and attaches a fixed number of check bits (or parity data), which are derived from the data bits by some deterministic algorithm. If only error detection is required, a receiver can simply apply the same algorithm to the received data bits and compare its output with the received check bits; if the values do not match, then it is understood that there is an error in transmission. If the code is non-systematic one, then the original message is transformed into an encoded message that has at least as many bits as that of the original message.

Error detection is most commonly realized using a suitable hash function (or checksum algorithm). A hash function adds a fixed-length tag to a message, which enables receivers to verify the delivered message by re-computing the tag and comparing it with the one given. A code with minimum Hamming distance, 'd' can detect upto (d-1) errors in a code word. Codes with minimum Hamming distance of $d = 2$ can detect only single bit errors. The parity bit is an example of a single-error-detecting code. The next section describes the proposed VLSI architecture for consecutive error detection with minimum number of redundant bits.

3. PROPOSED ARCHITECTURAL DESIGN

The main goal of the proposed system is to reduce redundant bits in DMC and to assure reliability in the presence of consecutive MCUs with reduced performance overheads. The encoder and decoder for 32-bit and 64-bit word are designed as an example.

3.1 THE SCHEMATIC OF FAULT TOLERANT MEMORY

The proposed schematic of Fault tolerant memory is shown in Fig.1. The data bits of 32 and 64 are given as input to encoder part which does two main processes. One is the generation of parity bits based on bit manipulation method and another one is generation of vertical redundant bits by Exclusive-OR (XOR) operation. The outputs of the encoder are 12 parity bits, 16 vertical redundant bits and 32-bit data. The outputs of encoder are stored in two SRAM memories. One SRAM memory is for 32-bit information and another is for storing 28-bit redundant bits.

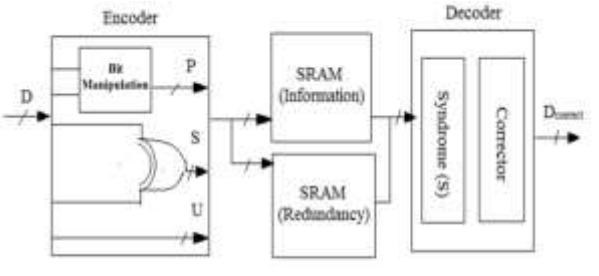

Fig.1. Proposed Schematic of Fault-Tolerant Memory

The stored information and redundant bits are given as input to the decoder and the parity bits and vertical redundant bits are calculated for MCU bits. Then the syndrome bits are calculated for vertical redundant bits. By comparing the parity and vertical redundant bits of the original and MCU data bits, the MCU bits are detected and corrected. The concept of parity bits, encoder and decoder of proposed system are detailed in the following sections.

3.2 ARCHITECTURAL DESIGN OF 64-BIT WORD ENCODER

In the proposed, first, the N-bit word is divided into k symbols of m information bits and the k symbols are divided into $k_1 \times k_2$ i.e., k_1 rows and k_2 columns. For a 32-bit word, it is divided into two 16 bits. So there are 2 rows and 4 columns. Two 16 bits are arranged in two rows i.e., four symbols in each row resulting in a total of 8 symbols of 4 bits; hence it is arranged as 2×4. When k = 4×4, m = 2 are chosen and only 3 bit errors can be corrected.

The logical organization for 32-bit word is shown in the above Fig.2. The structure of 64-bit word has two rows of each 32-bit and 8 columns of 8 symbols. The bit manipulation method is shown in Fig.3 and it is applied for each symbol containing four bits. For each four bits, three parity bits are generated. Here, the method is applied for 16 bit data, so 12 parity bits will be generated.

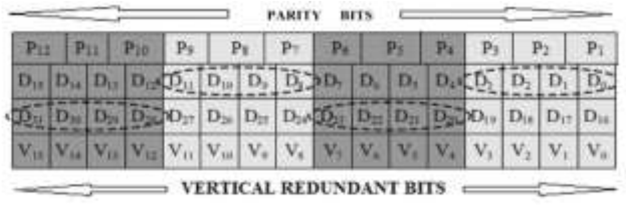
PARITY BITS
P12 P11 P10 P9 P8 P7 P6 P5 P4 P3 P2 P1
D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
D31 D30 D29 D28 D27 D26 D25 D24 D23 D22 D21 D20 D19 D18 D17 D16
V15 V14 V13 V12 V11 V10 V9 V8 V7 V6 V5 V4 V3 V2 V1 V0
VERTICAL REDUNDANT BITS

Fig.2. Logical organization of 32-bit word (k = 2×4, m = 4)

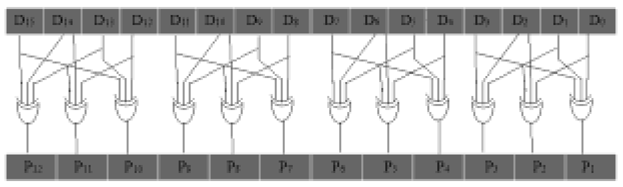
D15 D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1 D0
P12 P11 P10 P9 P8 P7 P6 P5 P4 P3 P2 P1

Fig.3. Bit manipulation based Parity bit generation

The parity are generated using XOR operation of data bits. For generating 3 parity bits of each symbol three XOR gates are needed. In Fig.2, P_1, P_2 and P_3 are the parity bits for the data bits D_0 to D_3. The Parity bit P_1 is generated by performing XOR operation for the data bits D_0 D_3 D_1. Likewise the parity bits P_2 and P_3 will be generated based on the Eq.(1) to Eq.(3).

$$P_1 = D_0 \oplus D_3 \oplus D_1 \tag{1}$$

$$P_2 = D_0 \oplus D_3 \oplus D_2 \tag{2}$$

$$P_3 = D_1 \oplus D_3 \oplus D_2 \tag{3}$$

It is same for other three symbols, so for each symbol three parity bits are generated. Totally 12 parity bits are generated. The concept of parity bit generation is same for 64-bit word but the total parity bit is varied from 12 to 24. The vertical redundant bits are calculated by using first and second row data bits. It is calculated by XOR function as in Eq.(4) and Eq.(5).

$$V_0 = D_0 \oplus D_{16}, \tag{4}$$

$$V_1 = D_1 \oplus D_{17} \quad (5)$$

The vertical redundant bits for 64-bit word are 32. The parity bits used to detect the consecutive errors in first row of data bits, but the vertical redundant bits are used to detect and correct consecutive errors in second row data bits.

The encoder and decoder are designed for both 32-bit and 64-bit. With the help of the input bit information, the encoder will calculate its parity bits and vertical redundant bits. The whole operation is controlled by an enable signal. When enable signal is 0, the encoder will not compute parity and redundant bits. When enable signal is 1, the encoder will compute parity and vertical redundant bits. The encoder operation for 64-bit word is shown in Fig.4.

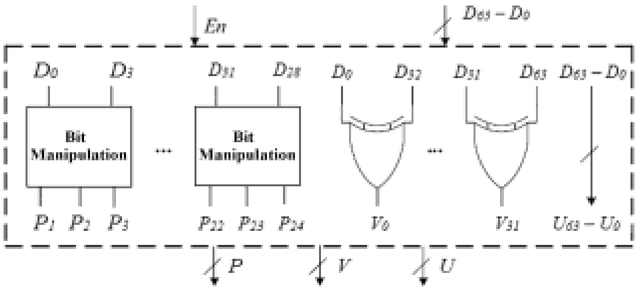

Fig.4. Proposed 64-bit Encoder Design

3.3 ARCHITECTURAL DESIGN OF 64-BIT WORD DECODER

The structure of 64-bit word decoder is shown in Fig.5.

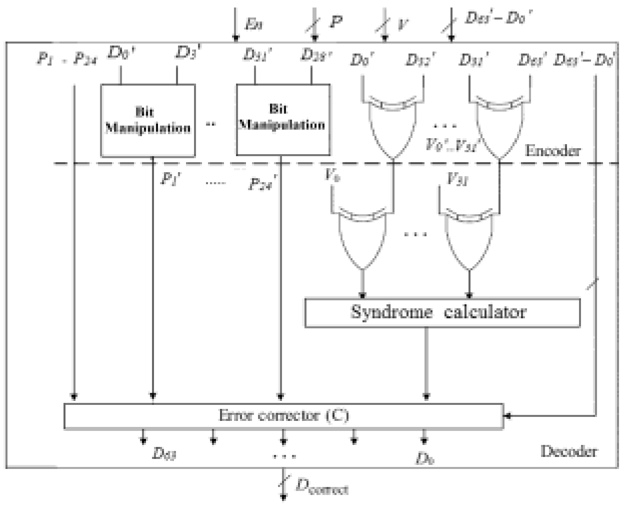

Fig.5. Proposed 64-bit Decoder Design

For example, first the MCU parity bits and vertical redundant bits are generated from the received MCU bits. The MCU bits are D_0' to D_{31}'. Second, vertical syndrome bits are calculated and parity bits for MCU bits and then the original bits are compared.

$$P_1\, P_2\, P_3 = P_1'\, P_2'\, P_3' \quad (6)$$

$$V_0 = V_0' \quad (7)$$

$$S_0 = V_0 \oplus V_0' \quad (8)$$

The decoder receives 32-bit or 64-bit information with parity bits and vertical redundant bits which is stored in SRAM memory. When the information bits stored in SRAM memory it may be hit by radiation particle, so it will be received as MCU bits in the decoder. The parity bits and vertical redundant bits for received MCU bits are calculated as given in the Eq.(1) to Eq.(5).

After calculating parity bits and vertical redundant bits for MCU bits, the MCU parity bits are compared with original parity bits as in the Eq.(6). By comparing this, the MCU bits in the first row (D_0 to D_{15}) are detected. Then comparison of the vertical redundant bits of original with that of the MCU bits is done as shown in the Eq.(7). By comparing vertical redundant bits of both, the MCU bits in second row will be detected. Next, the syndrome is calculated by performing XOR function of original and MCU vertical redundant bits as in the Eq.(8).

$$D_{correct} = D_3'\, D_2'\, D_1'\, D_0' \oplus S_3\, S_2\, S_1\, S_0 \quad (9)$$

After the detection of MCU bits, it is corrected by syndrome bits. The correction of MCU bits is performed based on the Eq.(9). The procedure of correcting MCU bits are as follows.

- First the parity bits of original and MCU bits are compared if it is found to be equal, then it is understood that the received bits are not MCU bits. If it is not equal, then the corresponding MCU bits are XOR with syndrome bits and stored as corrected bits in SRAM memory.
- Second the vertical redundant bits of original and MCU bits are compared if it is equal then it is clear that the received bits are not MCUs. If it is not equal, then the corresponding MCU bits are XOR with syndrome bits and stored as corrected bits in SRAM memory.

Finally, the whole corrected bits are stored in memory. The decoder has an enable signal to control the operation. When enable signal is 0, the decoder will not detect and correct the MCU bits. When it goes 1, the decoder will detect the MCU bits and correct it.

3.4 LIMITATIONS OF DECIMAL MATRIX CODE

Decimal Matrix Code proposed in [20] requires more redundant bits for error detection and correction. The consecutive error information bits cannot be detected and corrected by DMC. The limitation of DMC is shown in Fig.6 with an example. In DMC, horizontal redundant bits are calculated for both original and MCU bits. Based on that decimal difference the MCU bits are detected. When decimal difference between horizontal redundant bits results in logic ‘1’, then the corresponding symbol are in error.

$$\Delta H_4H_3H_2H_1H_0 = H_4H_3H_2H_1H_0' - H_4H_3H_2H_1H_0$$
$$= 01111 - 01111 = 00000$$

The above result shows that there is no error in received bits. But actually there are four consecutive errors in two symbols. Based on this result, it can be concluded that the errors are not detected and corrected in DMC.

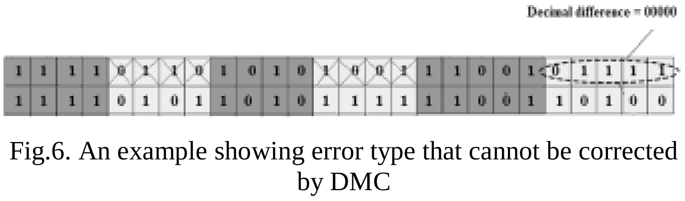

Fig.6. An example showing error type that cannot be corrected by DMC

The error bits will occur as an output. The redundant bits needed for 32-bit DMC is 36. If there is a reduction in the number of redundant bits, it is understood that more number of errors can be detected and corrected.

3.5 FEATURES OF VLSI ARCHITECTURE FOR CONSECUTIVE ERROR DETECTION AND CORRECTION

From the previous discussion, it is clear that the DMC [20] cannot detect and correct the consecutive errors. Also it requires more redundant bits to detect the MCU bits. Hence, it is decided to propose architecture based on bit manipulation method to overcome the drawback of DMC. The operation of consecutive error detection and correction is shown in Fig.7.

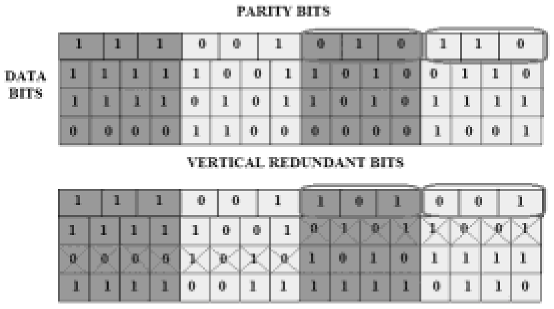

Fig.7. Consecutive error detection and correction

The proposed method concentrates on generating parity bits for both original and MCU bits. It is seen that the parity bits for symbol 0 of original and MCU bits are 110 and 001, and parity bits are not matched so it can be clearly identified that there are error bits in the symbol. At the same time the parity bits for symbol 2 of original and MCU bits are 001 and 110, and it shows that the parity bits are different so the bits are in error. Here the eight consecutive errors are detected using bit manipulation by generating parity bits. With DMC, it is impossible to detect four consecutive errors. When the parity bits of original and MCU bits are different then the corresponding symbols are in errors. Then the corresponding symbol performs XOR operation with syndrome bits. When vertical redundant bits of original and MCU bits are different then the corresponding symbols are in error. Here the correct bits will be obtained by performing XOR operation with error bits. The detection and correction of MCU bits are explained below.

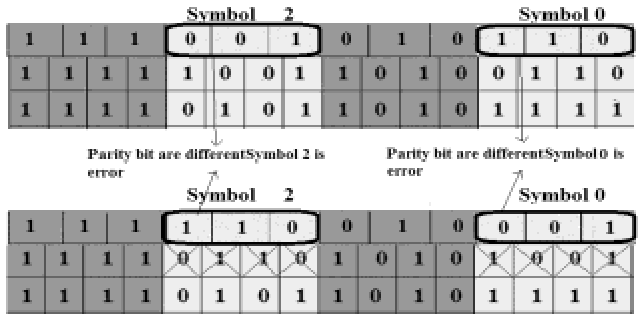
Symbol 2
Symbol 0
Parity bit are differentSymbol 2 is error
Parity bit are differentSymbol 0 is error
Symbol 2
Symbol 0

Fig.8. Detection of symbol 0 and 2 MCU bits

As seen from the Fig.7, if the actual parity bits of symbol 0 and MCU bits are different as in the Eq.(10); XOR operation is done for the corresponding MCU bit in symbol 0 and syndrome bits to correct the MCU bits.

$$P_1\,P_2\,P_3 \neq P_1'\,P_2'\,P_3' \tag{10}$$

$$S_3S_2S_1S_0 = 1111$$

$$D_{correct}(3\text{ to }0) = D_3'D_2'D_1'D_0' \oplus S_3S_2S_1S_0 = 1001 \oplus 1111 = 0110$$

Finally the original bits are obtained from the error corrector part. Then the corrected bits are stored in same bit location in the memory. Otherwise, when parity bits of original and MCU bits are same and vertical redundant bits are different as in the Eq.(11) and Eq.(12); the corresponding MCU bit in symbol 0 is XOR with syndrome bits to correct the MCU bits.

$$P_9\,P_8\,P_7 = P_9'\,P_8'\,P_7' \tag{11}$$

$$V_{11}\,V_{10}\,V_9\,V_8 \neq V_{11}'\,V_{10}'\,V_9'\,V_8' \tag{12}$$

$$S_3S_2S_1S_0 = 1111$$

$$D_{correct}(27\text{ to }24) = D_{27}'\,D_{26}'\,D_{25}'\,D_{24}' \oplus S_{27}\,S_{26}\,S_{25}\,S_{24}$$

$$= 1010 \oplus 1111 = 0101$$

Then the corrected bits are stored in same bit location in the memory. The same procedure is repeated for 64-bit word with eight symbols. The bit manipulation method overcomes the consecutive error bits in symbol 0 and 2 which is shown in Fig.8. If the radiation strikes symbol 0 and 2 bits in memory, DMC cannot detect the MCU bits which results in error data. It is due to calculation of horizontal redundant bits. But it is detected by using bit manipulation method. The parity bits of original and MCU bits of symbol 0 are different. Likewise the parity bits of original and MCU bits of symbol 0 are different. This show the bits in symbol 0 and 2 are in error. Finally the bit that cannot be detected by DMC is detected by bit manipulation method.

4. RESULTS AND DISCUSSION

The proposed design is tested for its functionality by varying the inputs. The area, power and delay of the new architecture are obtained and the results are compared and are shown in Table.1.

Table.1. Performance Analysis

Parameters	DMC		Bit Manipulation	
	32-bit	64-bit	32-bit	64-bit
Area (ALUTs)	245	489	160	228
Power (mW)	327.2	330.8	327.3	330.7
Time (ns)	7.595	8.492	6.065	5.699

It can be visualized from the Table.1 that the area overhead is much reduced in the proposed system. Power reduction cannot be taken in a positive sense but there is a significant reduction in time from 7.595ns of DMC to 6.065ns in bit manipulation method for a 32-bit word and 8.492ns to 5.699ns for a 64-bit word correspondingly.

The proposed design of encoder and decoder is implemented in VHDL and simulated with ModelSim. The Table.2 shows the reduction in the number of redundant bits for the proposed 32-bit and 64-bit word compared with 32 and 64-bit DMC.

Table.2. Comparison of Redundant bits

Technique	**DMC**		**Bit Manipulation**	
Redundant Bits	**Horizontal Bits**	**Vertical Bits**	**Parity Bits**	**Vertical Bits**
	20	16	12	16
Total no. of Redundant Bits	36		28	

4.1 ENCODER OUTPUT FOR 32-BIT WORD

The encoder has two inputs such as 32-bit input and enable signal as shown in Fig.9.

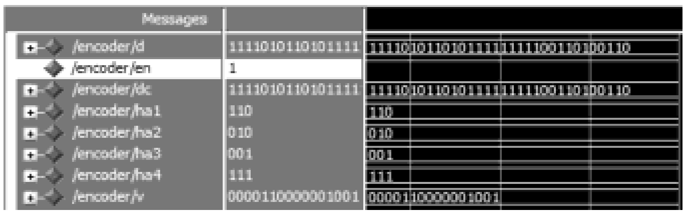

Fig.9. 32-bit Encoder Output

- Input Data: 11110101101011111111100110100110

When enable signal is high, the parity bits and vertical redundant bits are calculated.

- $P_1P_2P_3$ = 110, $P_4P_5P_6$ = 010, $P_7P_8P_9$ = 001, $P_{10}P_{11}P_{12}$ = 111
- V_{15} to V_0 = 0000110000001001

4.2 DECODER OUTPUT FOR 32-BIT WORD

The decoder inputs are MCU bits, vertical redundant bit and parity bits and it is shown in Fig.10. The decoder enable signal is kept high to detect and correct the MCU bits.

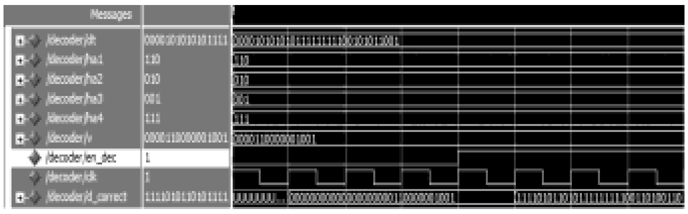

Fig.10. 32-bit Decoder Output

- MCU input 00001010101011111111100101011001
- $P_1P_2P_3$ = 110, $P_4P_5P_6$ = 010, $P_7P_8P_9$ = 001, $P_{10}P_{11}P_{12}$ = 111
- V_{15} to V_0 = 0000110000001001
- The output of decoder is the corrected one and it is as follows, 11110101101011111111100110100110

4.3 COMBINED OUTPUT OF 32-BIT WORD

If the RAM enable signal is '0' and decoder enable is '1', only then, the decoder will detect the MCU bits and correct it and it is shown in Fig.11.

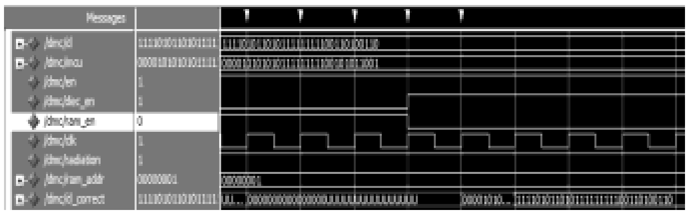

Fig.11. 32-bit Combined Output

- Input Data: 11110101101011111111100110100110.
- Fault injected MCU data: 00001010101011111111001010 1100.

The resultant output of decoder is, 1111010110101111111110 0110100110

4.4 ENCODER OUTPUT FOR 64-BIT

The reliability for 64-bit data is analyzed with ModelSim. The encoder has two inputs 64-bit input and enable signal. It is shown in Fig.12.

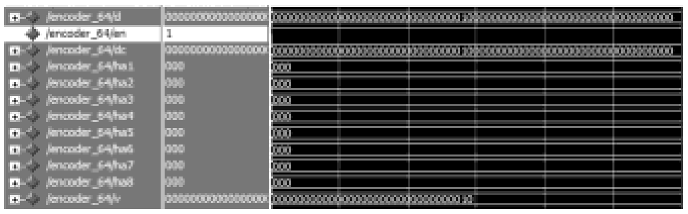

Fig.12. 64-bit Encoder Output

- Data input given: 00000000000000000000000000000100 00000000000000000000000000000000

When enable signal is high, the parity bits and vertical redundant bits are calculated.

- $P_1P_2P_3$ = 000, $P_4P_5P_6$ = 000, $P_7P_8P_9$ = 000, $P_{10}P_{11}P_{12}$ = 000, $P_{13}P_{14}P_{15}$ = 000, $P_{16}P_{17}P_{18}$ = 000, $P_{19}P_{20}P_{21}$ = 000, $P_{22}P_{23}P_{24}$ = 000
- V_{31} to V_0: 00000000000000000000000000000010

4.5 DECODER OUTPUT FOR 64-BIT

The decoder output is shown in Fig.13.

- MCU bit: 11111111111111110000000000000000100000000 000000000111111111111111
- $P_1P_2P_3$ = 000, $P_4P_5P_6$ = 000, $P_7P_8P_9$ = 000, $P_{10}P_{11}P_{12}$ = 000, $P_{13}P_{14}P_{15}$ = 000, $P_{16}P_{17}P_{18}$ = 000, $P_{19}P_{20}P_{21}$ = 000, $P_{22}P_{23}P_{24}$ = 000
- V_{31} to V_0: 00000000000000000000000000000010

Here the decoder enable signal will enable the MCU bits to detect and correct.

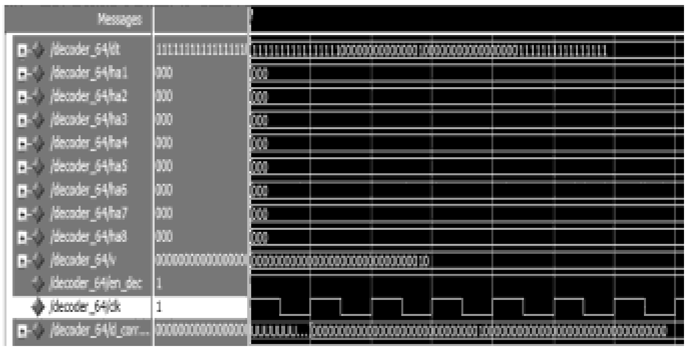

Fig.13. 64-bit Decoder Output

- The output of decoder is the corrected bit: 00000000000000 0000000000000000100000000000000000000000000000 000

4.6 COMBINED OUTPUT OF 64-BIT WORD

The whole process of proposed system is shown in Fig.14.

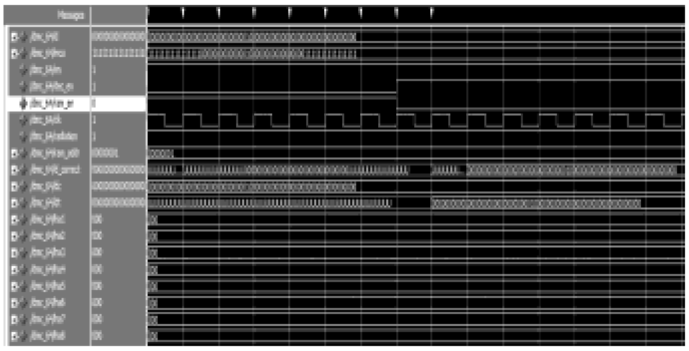

Fig.14. 64-bit Combined Output

- Data input: 0000000000000000000000000000000010000000 000000000000000000000000.
- Fault injected MCU data: 11111111111111111000000000000 0001000000000000000001111111111111111.

If the RAM enable signal is ‘0’and decoder enable is ‘1’, then only the decoder will detect the MCU bits and correct it, otherwise it will not detect and correct the MCU bits.

- The output of decoder is the corrected bit: 00000000000000 0000000000000000100000000000000000000000000000 000.

4.7 DETECTION OF SYMBOL 0 AND 2 MCU IN DMC

The detection of MCU bits in symbol 0 and 2 in DMC method is shown in Fig.15. It is done by bit manipulation method based on XOR operation.

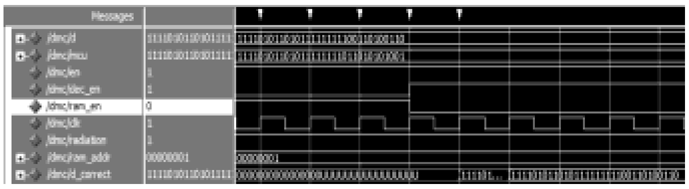

Fig.15. Detection of symbol 0 and 2 MCU bits

- Data input:
  11110101101011111111100110100110
- Fault injected MCU data:
  11110101101011111111011010101001

If the RAM enable signal is ‘0’and decoder enable is ‘1’, then only the decoder will detect the MCU bits and correct it, otherwise it will not detect and correct the MCU bits.

- The output of decoder is the corrected bit:
  11110101101011111111100110100110

5. CONCLUSIONS

It is very clear from the analysis that the reliability of memory is improved very much compared with the existing Decimal Matrix Code. Here, the design utilizes bit manipulation method to detect and correct 16 bit errors, hence it is possible to correct and detect a maximum of 16 bit errors. It is also seen that for a 32-bit word, the proposed system can correct and detect up to eight consecutive errors in each row. And for a 64-bit word, it can correct and detect 16 consecutive errors in each row. And also, it is noted that the proposed architecture achieves a reduction in the number of redundant bits i.e., 28 bits compared to DMC’s 36 bits. The power of proposed system is same as that of the DMC, but the delay overhead is reduced to a significant level compared to DMC. Hence, it can be concluded that the proposed VLSI architecture outperforms DMC in memory reliability. The limitation of the proposed is that it cannot detect and correct two parallel MCU symbols in logical organization. Therefore, it is decided to recover the parallel MCU symbols in near future.

REFERENCES

[1] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo and T. Toba, “Impact of Scaling on Neutron Induced Soft Error in SRAMs from an 250nm to a 22nm Design Rule”, *IEEE Transactions on Electron Devices*, Vol. 57, No. 7, pp. 1527-1538, 2010.

[2] X. Pan, H. Guo, Y. Luo, F. Zhang and L. Ding, “Analysis of Multiple Cell Upset Sensitivity in Bulk CMOS SRAM after Neutron Irradiation”, *Chinese Physics B*, Vol. 27, No. 3, pp. 1-7, 2018.

[3] A. Sanchez-Macian, P. Reviriego and J.A. Maestro, “Hamming SEC-DAED and Extended Hamming SEC-DED-TAED codes through Selective Shortening and Bit Placement”, *IEEE Transactions on Devices and Materials Reliability*, Vol. 14, No. 1, pp. 574-576, 2014.

[4] P. Reviriego, M. Flanagan and J.A. Maestro, “A (64, 45) Triple Error Correction Code for Memory Applications”, *IEEE Transactions on Devices and Materials Reliability*, Vol. 12, No. 1, pp. 101-106, 2012.

[5] R. Naseer and J. Draper, “Parallel Double Error Correcting Code Design to Mitigate Multi-bit Upsets in SRAMs”, *Proceedings of 34th European Conference on Solid State Circuits*, pp. 222-225, 2008.

[6] G. Neuberger, D.L. Kastensmidt and R. Reis, “An Automatic Technique for Optimizing Reed-Solomon Codes to improve Fault Tolerance in Memories”, *IEEE Design and Test of Computers*, Vol. 22, No. 1, pp. 50-58, 2005.

[7] C. Argyrides and D.K. Pradhan, “Improved Decoding Algorithm for High Reliable Reed Muller Coding”, *Proceedings of IEEE International Conference on System On Chip*, pp. 95-98, 2007.

[8] S. Liu, P. Reviriego and J.A. Maestro, “Efficient Majority Logic Fault Detection with Difference-Set Codes for Memory Applications”, *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 20, No. 1, pp. 148-156, 2012.

[9] S. Baeg, S. Wen and R. Wong. “Interleaving Distance Selection with a Soft Error Failure Model”, *IEEE Transactions on Nuclear Science,* Vol. 56, No. 4, pp. 2111-2118, 2009.

[10] M. Zhu, L.Y. Xiao, L.L. Song, Y.J. Zhang and H.W. Luo, “New Mix Codes for Multiple Bit Upsets Mitigation in Fault-Secure Memories”, *Micro Electronics Journal*, Vol. 42, No. 3, pp. 553-561, 2011.

[11] K. Pagiamtzis and A. Sheikholeslami, “Content Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey”, *IEEE Journal on Solid State Circuits*, Vol. 41, No. 3, pp. 712-727, 2003.

[12] P. Reviriego and J.A. Maestro, “Efficient Error Detection Codes for Multiple-bit Upset Correction in SRAMs with BICS”, *ACM Transactions on Design Automation of Electronic Systems*, Vol. 14, No. 1, pp. 1-18, 2009.

[13] C. Argyrides, R. Chipana, F. Vargas and D.K. Pradhan, “Reliability Analysis of H-tree Random Access Memories implemented with Built in Current Sensors and Parity Codes for Multiple Bit Upset Correction”, *IEEE Transactions on Reliability*, Vol. 60, No. 3, pp. 528-537, 2011.

[14] F. Silva and J. Silveira, “An Extensible Code for Correcting Multiple Cell Upset in Memory Arrays”, *Journal of Electronic Testing*, Vol. 34, No. 4, pp. 417-433, 2018.

[15] C. Argyrides, D.K. Pradhan and T. Kocak. “Matrix Codes for Reliable and Cost Efficient Memory Chips”, *IEEE Transactions on Very Large Scale Integration* Systems, Vol. 19, No. 3, pp. 420-428, 2011.

[16] C.A. Argyrides, C.A. Lisboa, D.K. Pradhan and L. Carro, “Single Element Correction in Sorting Algorithms with Minimum Delay Overhead”, *Proceedings of IEEE Latin-American Test Workshop*, pp. 652-657, 2009.

[17] U.K. Kumar and B.S. Umashankar, “Improved Hamming Code for Error Detection and Correction”, *Proceedings of IEEE International Symposium on Wireless Pervasive Computing*, pp. 1-5, 2007.

[18] S. Baeg, S. Wen and R. Wong, “Minimizing Soft Errors in TCAM Devices: A Probabilistic Approach to Determining Scrubbing Intervals”, *IEEE Transactions on Circuits and Systems*, Vol. 57, No. 4, pp. 814-822, 2010.

[19] Y. Yahagi, H. Yamaguchi, E. Ibe, H. Kameyama, M. Sato, T. Akioka and S. Yamamoto, “A Novel Feature of Neutron-induced Multi-cell Upsets in 130 and 180nm SRAMs”, *IEEE Transactions on Nuclear Science*, Vol. 54, No. 4, pp. 1030-1036, 2007.

[20] J. Guo, L. Xiao, Z. Mao and Q. Zhao, “Enhanced Memory Reliability Against Multiple Cell Upsets Using Decimal Matrix Code”, *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 22, No. 1, pp. 127-135, 2014.

[21] T. Evangeline Santhia, R. Helen Ramya Bharathi and M. Revathy, “Error Detection and Correction using Decimal Matrix Code: Survey”, *Proceedings of IEEE International Conference on Electrical, Instrumentation and Communication Engineering*, pp. 12-17, 2017.