Soft Error Tolerance in Memory Applications

Muhammad Sheikh Sadi, Md. Shamimur Rahman, Shaheena Sultana, Golam Mezbah Uddin, Kazi Md. Bodrul Kabir
Department of Computer Science and Engineering
Khulna University of Engineering & Technology
Khulna, Bangladesh

***Abstract*—This paper proposes a new method to detect and correct multi bit errors in memory applications using a combination of a clustering approach, Bit-Per-Byte error detection technique, and Majority Logic Decodable (MLD) codes. The likelihood of soft errors accelerates with system complexity, reduction in operational voltages, exponential growth in transistor per chip, increases in clock frequencies, breakdown of memory reliability and device shrinking. Memories are the sensitive part of a computer system. Soft errors in memories may cause an instruction to malfunction. Several techniques are already in practice to mitigate the soft errors. Majority logic decodable codes are proved as effective for memory applications because of their ability to correct a massive number of errors. Since memories are used to hold large number of bits that's the restraint of Majority logic decodable codes method, so we emphasize on the size of data word in this method. The proposed method aims to detect and correct up to seven bit errors with lesser computational time. It works in an efficient manner in case of adjacent errors which is not possible in Majority logic decodable codes (MLD). It is delineated by Experimental reviews that the proposed approach outperforms existing dominant approach with respect to number of erroneous bit detection and correction, and computational time overhead.**

Keywords*—Soft error tolerance; bit-per-byte; majority logic decodable codes; clustering; adjacent errors

I. Introduction

The unusual condition of multifaceted nature, and the way that the software and hardware are so unpredictably connected, denotes that the system might be extremely delicate to soft errors. In particular, soft errors are a matter of great concern when planning high accessibility systems or systems utilized as a part of electronic-antagonistic situations [1]-[4]. In memory applications, soft error can change an instruction or any data value [3]-[5]. Almost all system chips have embedded memories like ROM, DRAM, SRAM, flash memory etc. But soft errors in such memory applications are increasing alarmingly as technology these days is focusing on smaller dimension of devices which leads to the integration of circuits [6]. Integrated circuits are prone to particle strike or radiation which can cause the memory cell to change its state and obtain a different value than what was desired. Small size of transistors, capacitors and low operating voltages are also the reasons for soft error in memories. So, fault tolerant technique in memory architecture is fundamental issue to ensure its reliability to the users. A small flaw or glitch in a memory cell can change an instruction or can cause a whole program to work incorrectly leading to inappropriate information or loss of valuable data.

There are some existing dominant approaches to provide fault tolerance in memory applications. For example, for satellite applications, hamming code and parity codes are used to secure memory devices. There are some other methods for error detection and correction such as Error Correction Code (ECC) [7]-[9], Euclidean geometry low-density parity check (EG-LDPC) codes [10], [11], etc. However, almost all of these methods are facing area, and time overhead, and significant power consumption penalty. Also these methods have low error detection and correction rate and exhibits lower performance while working with large data word. To overcome these barriers, we came up with a fault tolerant technique which can work with larger data word and consume lesser processing time.

In this paper, an error detection and correction technique is proposed to protect the memory applications. This method combines the salient features of clustering approach [12], Bit-Per-Byte error detection technique, and Majority Logic Decodable (MLD) codes [13]-[16]. Majority Logic Decodable codes are used because of their ability to detect multiple bit upsets; Bit-per-byte technique minimizes the required time to detect the error; and the clustering approach works in a very efficient manner in case of adjacent errors. The proposed method provides high efficiency for error detection and correction and can correct up to 7-bit upsets in a 49-bits' data block.

The rest of this paper is presented as follows. Section 2 provides several related work in this area of research. The proposed methodology and associated examples are discussed in Section 3. Experimental analysis is shown in Section 4. Section 5 concludes the paper.

II. Related Work

First Several techniques are already in practice to provide error detection and correction. Some of them are discussed below.

Naeimi et al. [8] proposed a fault-tolerant memory architecture which can tolerate faults both in the storage unit and in the encoder or decoder. A fast and compact error correcting technique is proposed in that paper which is known as one step majority logic correction. One step majority logic correction works in a way that it corrects every erroneous bit at each step and will output the correct code word after full processing. This method requires the same number of cycles as the number of bits for both detection and correction which is a major degrade in performance in terms of access time in memory.

Shih-Fu et al. [7] presented an error detection method for different set cyclic codes using majority logic decoding scheme. Majority logic decodable codes are most appropriate for memory applications because they deal with large number of errors but it may lower the memory performance with excessive decoding time. MLD was first introduced for Reed-Muller codes. They described a plain majority logic decoder (MLD) whose circuit arrangement includes four components: i) a cyclic shift register; ii) an XOR matrix; iii) a majority gate; and iv) an XOR for correcting the code word bit under decoding. It can correct multiple bit-flips depending on the number of parity check equations [6]. They proposed a modified version of MLD which is known as Majority Logic Detector/Decoder. The MLDD technique needs 15 cycles to correct an error. However, it can detect and correct only two bit errors from a 15-bitdata word and the time requirement of this method is high enough to degrade its performance in terms of access time in memory.

Jayalakshmi et al. [5] came out with a modified representation of MLDD. It overcomes the existing techniques by detecting errors in lesser cycles. They used additional logic which results in an area overhead. Another limitation is that this method needs additional three cycles to correct any error.

III. Proposed Methodology to Detect and Correct Errors

In this chapter, the proposed method will be discussed and explained elaborately. The chapter will take you step by step through our method to have a better understanding about the method. Some examples along with pictorial representation will be provided with the method explanation.

A. Memory with MLDD

The existing MLDD [5] is modified to improve its performance. Euclidean Geometry Low Density Parity Check Codes (EG-LDPC) [6] works behind the existing MLDD. The following Fig. 1 shows how the MLDD modification proposed by us will be used in a memory system.

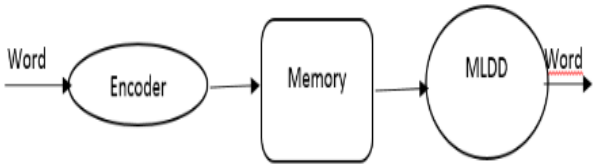

Fig. 1. Proposed Structure of a Memory System with MLDD.

B. Encoder Architecture

The design of encoder is generated from the EG-LDPC codes. The following parameter are in the function of EG-LDPC for any integer t >= 2, where t is the number of errors that the code can correct.

- Information bits, k = 22t – 3t
- Code word Length, n = 22t – 1
- Minimum distance, dmin = 2t + 1

Let's consider t=2 and if the other parameters are determined accordingly then we would have a (15, 7, 5) EG-LDPC code which will have a generator matrix like Fig. 2 and if Fig. 3 the architecture of an encoder circuit [7] for (15, 7, 5) EG-LDPC code is shown. The information bits are indicated from $i_0 \ldots i_6$. The check bits are calculated using linear sum (XOR) operation of the information bits. The information bits are copied to the encoded vector from $c_0 \ldots c_6$ and the check bits are copied from $c_7 \ldots c_{14}$. Thus the encoded matrix is generated.

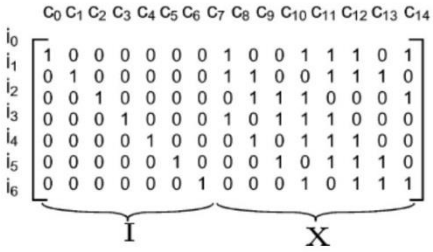

Fig. 2. Generator Matrix of (15, 7, 5) EG-LDPC code [8].

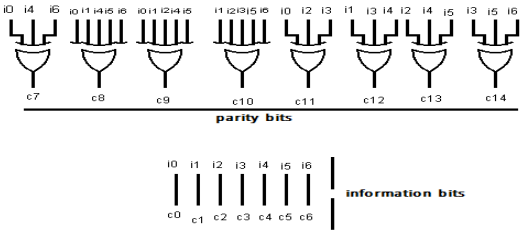

Fig. 3. Architecture of an Encoder Circuit for the (15, 7, 5) EG-LDPC code.

C. Design Structure of Corrector

One-step majority-logic is a fast and efficient error-handling technique [10]. There is a class of ECCs that are one-step-majority correctable. Type-I two-dimensional EG-LDPC is one of the example of one-step-majority correctable codes. In this section, the one-step majority-logic corrector for EG-LDPC codes is shown.

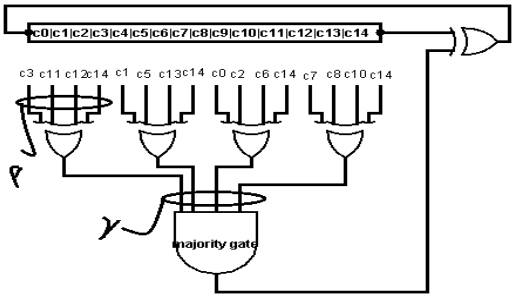

Fig. 4. Serial One-Step Majority Logic Structure to Correct Last Bit (Bit 14th) of 15-bit (15, 7, 5) EG-LDPC code [8].

A linear sum named Parity-Checksum can be formed by computing the internal product of the received vector and a row of a parity-check matrix. The principle of the one-step majority-logic corrector is generating parity-check sums from the defined rows of the parity-check matrix. These steps correct a potential error in one bit e.g., c_{n-1}.

1) Generate parity-check sums by calculating the inner product of the received vector and the defined rows of parity-check matrix.

2) The check sums are fed into a majority gate. If the output of majority gate is "1", then the bit cn-1 is corrected by inverting the value of cn-1.

The architecture of a serial one-step majority logic corrector for (15, 7, 5) EG-LDPC code is shown in Fig. 4.

D. Fundamental Concepts of Proposed Methodology

The proposed methodology uses the MLDD [5] technique described above as a part of correction method. Our proposed method is tested for a 49-bit data block and it can correct up to 7 bit errors. We proposed a clustering idea to divide consecutive seven bit placed in different cluster. That's why this proposed method can be applied where there is need to detect and correct adjacent multiple cell upset (MCU). Because adjacent bits are in different cluster and change in adjacent bits can detect easily and correct. The method is discussed below:

1) At first the data word which has the size of 49 bit, is clustered into 7 clusters keeping distance 7 between the data bits or information bits. We will keep 7 bits in each cluster. So this will result in 49/7=7 clusters. Now each cluster will have the information as shown in Fig. 5. The 49 data bits are represented as a_1, a_2, a_3..., a_{49}. Then form 7 different clusters such as a_1, a_8, a_{15}, a_{22}, a_{29}, a_{36}, a_{43} and adjacent bits like a_1, a_2, a_3 are placed in different clusters.

2) Each cluster has 7 information bits. Now we calculate even parity for each cluster. It is quite similar to the idea of bit-per-byte technique. If we consider each cluster as a byte (although each cluster here has 7 bits and a byte is formed of 8 bits) then we can apply the bit per byte technique on the clusters like a bit-per-cluster. We have used even parity technique here to assign parity to the clusters. Even parity means the number of 1's must be even. If number of 1's is even then parity is 0, otherwise parity is 1to make number of 1's is even. So after this step, each cluster has it corresponding parity which will be sent with the information bits. We can visualize it as shown in Fig. 6.

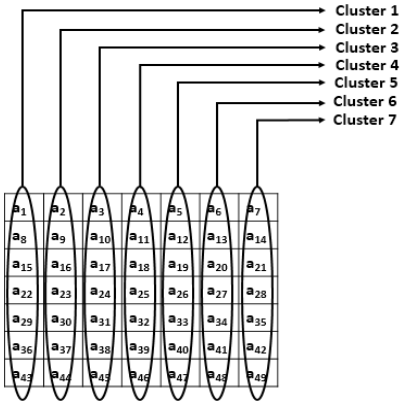

Fig. 5. Architecture of Seven Clusters with 49 Information Bits.

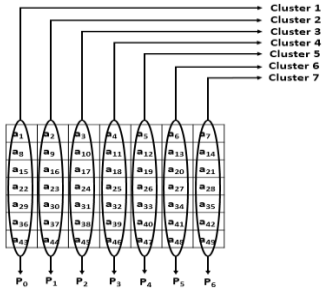

Fig. 6. Calculated Parity Bits for Each Cluster.

3) Now we are going to apply Majority Logic Detector Decoder (MLDD) scheme for each cluster. Let's consider each cluster has information bits denoted as i_0…. i_6. Then according to the MLD [7] we have generated the check bits from the information bits which are the checksums (XOR) of information bits. The check bits are generated as shown in Fig. 7.

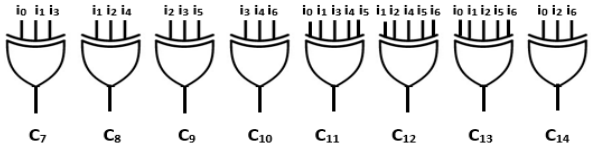

Fig. 7. The Architecture to Generate Cheek Bits.

Now the clusters have 7 information bits and 8 check bits which is 15 bits.

4) In this step, the information bits will be sent to the receiving side in the form which was seen at the first step like a_1, a_2, a_3…., a_{49}. With the information bits, parity bits of each cluster will also be sent which was calculated using odd parity. Along with these, the check bits for each cluster are also sent to the receiving end. So, the following information are sent from the sending end.

- Information bits (a_1, a_2, a_3, …., a_{49})
- Parity bits for each cluster (p_0, p_1, p_3, …, p_6)
- Check bits generated for each cluster (C_7, C_8, C_9, …, C_{14})

5) This information is sent to the receiving side. While transmitting the above information, any bit may flip and change the state from 0 to1 or 1 to 0 resulting in misleading information. At the receiving end the information bits will be received but they may not be error free. Let the received information bits are a_1, a_2, a_3, …, a_{49})

6) At the receiving end, we will form clusters like we did in step 1. So we will have 7 clusters keeping distance as 7 among the information bits of each cluster. Finally, the generated clusters are- Cluster1, Cluster2, Cluster3, …, Cluster 7.

7) After forming the clusters, we will calculate the parity bits for each cluster using odd parity. So the parity of each cluster at the receiving end may look like- parity (Cluster1), parity (Cluster2), parity (Cluster3) … parity (Cluster7).

8) In this step parity of each cluster of sending end will be compared with the parity of receiving end's cluster. If a mismatch is found at any cluster, then that cluster will be taken under consideration and that cluster is assumed to have error in its bits. Now let's assume Cluster (i) have a mismatch and it has errors. Now check bits will be generated for that cluster using the technique as described in step 3. So after generating the check bits (C_7, C_8, C_9, …, C_{14}) we will have total 15 bits to apply the majority logic decoding. The information bits are copied to C_0, C_1, …, C_6. So the code word will be like: C_0, C_1, C_3…, C_{14}.

9) The process of majority logic decoding is outlined shortly as follows:

Step 1: Initialize counter variable to 0.

Step 2: Calculate majority values B_j as follows:

$$B_1 = C_3 \oplus C_{11} \oplus C_{12} \oplus C_{14} \quad \text{Eq.} \quad (1)$$

$$B_2 = C_1 \oplus C_5 \oplus C_{13} \oplus C_{14} \quad \text{Eq.} \quad (2)$$

$$B_3 = C_0 \oplus C_2 \oplus C_6 \oplus C_{14} \quad \text{Eq.} \quad (3)$$

$$B_4 = C_7 \oplus C_8 \oplus C_{10} \oplus C_{14} \quad \text{Eq} \quad . (4)$$

Step 3: If majority value is greater than 2 then go to step 4, else go to step 5.

Step 4: Inverse the 14th bit. Store the counter which is the erroneous bit position. Go to step 5

Step 5: Perform one-bit cyclic left shift.

Step 6: Increment the counter

Step 7: If counter variable equals to 8 then go to step 8 else go to step 2

Step 8: End

10) Now we have the positions where bit flip in a cluster has occurred during transmission and those erroneous bits are corrected. We store those positions in a cluster to determine the actual positions in the data word. Next we examine other clusters to fine errors (if any) and find their positions in the corresponding cluster and thereby correct them. If we follow this method, then we would be able to detect and correct adjacent bit upsets which is a common issue in memory applications. Let's walk through an example to describe our method with sending end code word of Fig. 8 and receiving end code word of Fig. 9. Sending code word is the original data with parity bits and receiving code word is the erroneous collection of original code word.

For the above example, total seven clusters can be formed with the above forty-nine data bits. Now, the parity bits of receiving clusters are compared with those of the sending clusters.

0	1	1	1	0	0	0
1	1	0	0	1	0	0
1	1	1	0	1	1	1
0	0	0	0	1	0	0
0	1	0	0	1	0	0
0	1	1	0	1	0	1
0	1	1	1	0	0	0

Fig. 8. Sending Code Word.

0	0	1	1	0	0	0
1	1	0	0	1	0	0
1	1	1	0	1	1	1
0	0	0	0	1	0	0
0	1	1	0	1	0	1
0	1	1	0	1	0	1
0	1	1	1	0	0	0

Fig. 9. Receiving Code Word.

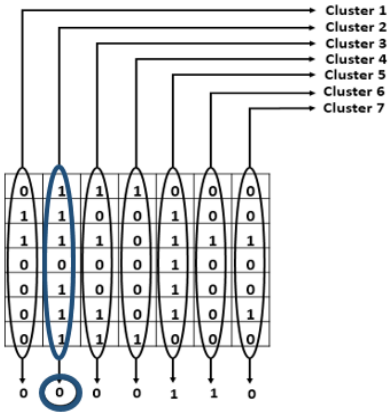

Fig. 10. Parity Bits of Sending Part.

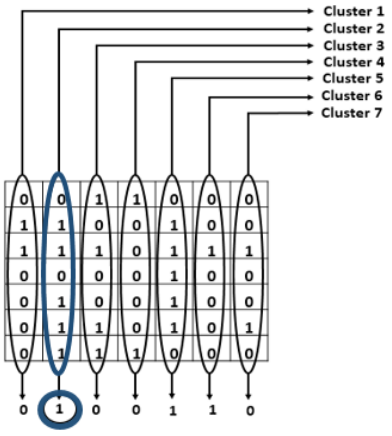

Fig. 11. Parity Bits of Receiving Par.

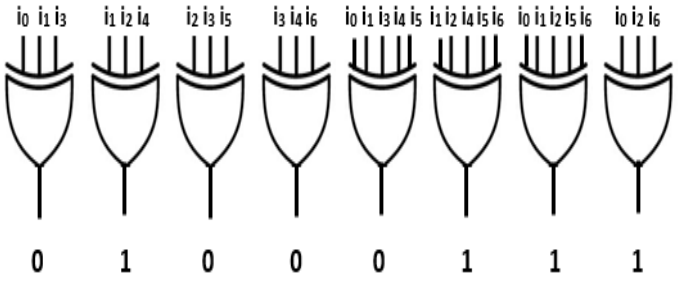

Fig. 12. Calculate Cheek Bits when Mismatching in Sending and Receiving Parity Bits.

If there is any mismatch, then only for this cluster we will generate 8-bit parity using Majority Logic Detector Decoder (MLDD) scheme.

As shown in Fig. 10 and 11, we can observe that in second cluster there is a mismatch and for this cluster we will generate 8-bit parity using the following architectures shown in Fig. 12.

Then for the erroneous cluster, the size of the code word will be 15-bit. i.e. C_0, C_1, C_2, C_3, C_4, C_5, C_6, C_7, C_8, C_9, C_{10}, C_{11}, C_{12}, C_{13} and C_{14}.In this case, it will be 011011101000111.

Using majority decoding circuit, we will perform eight left cyclic shift. At each step of shift operation, the majority values B_1, B_2, B_3, and B4 will be calculated. If the majority values are 1 then it is confirmed that the current bit under decoding is erroneous. Then an inverter is added to the 14th bit position in the register. The whole procedure of eight cycles is shown in Fig. 13.

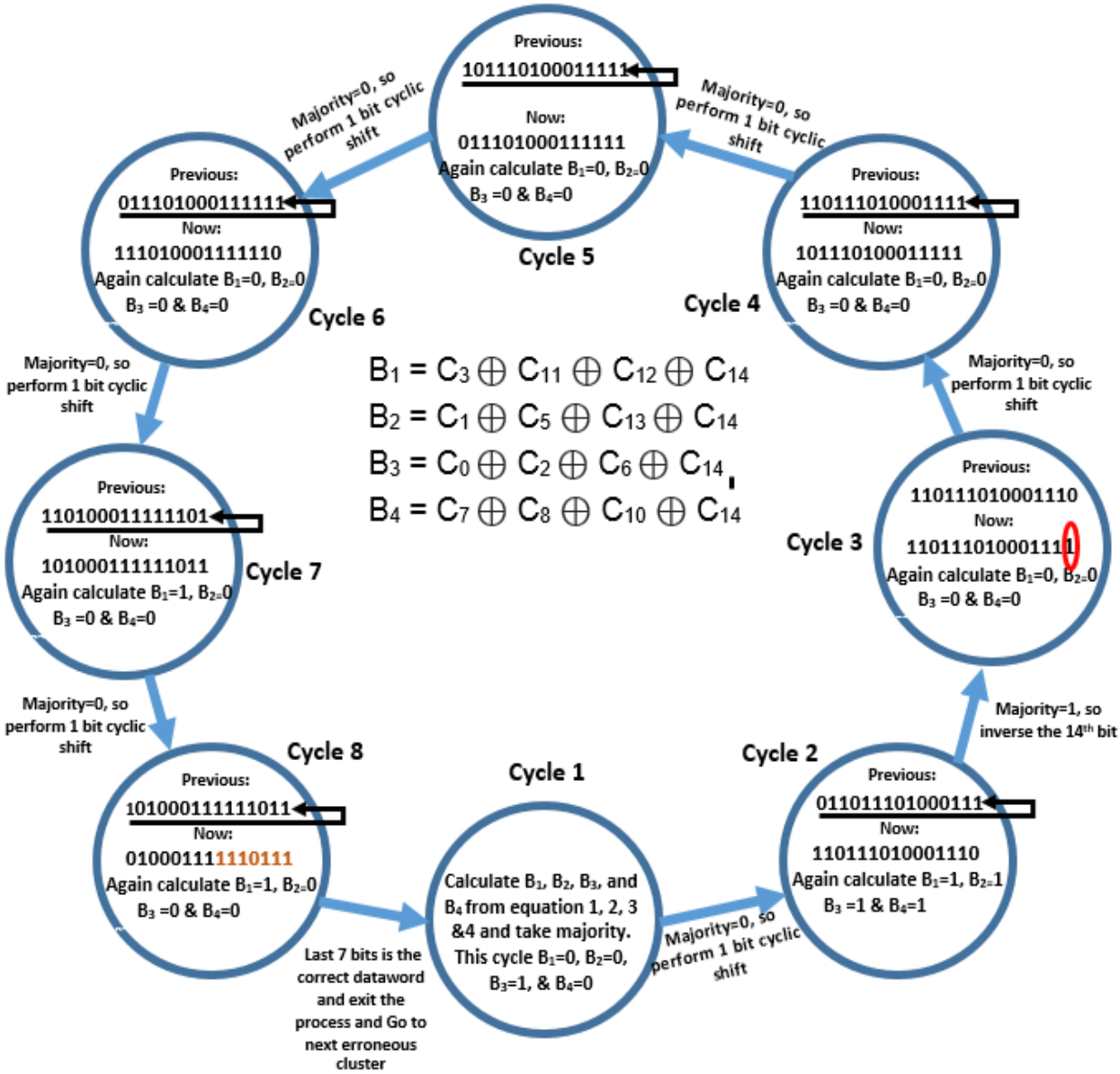

Fig. 13. Performing Eight Left Cyclic Shift for Acquiring the Error Free Code Word.

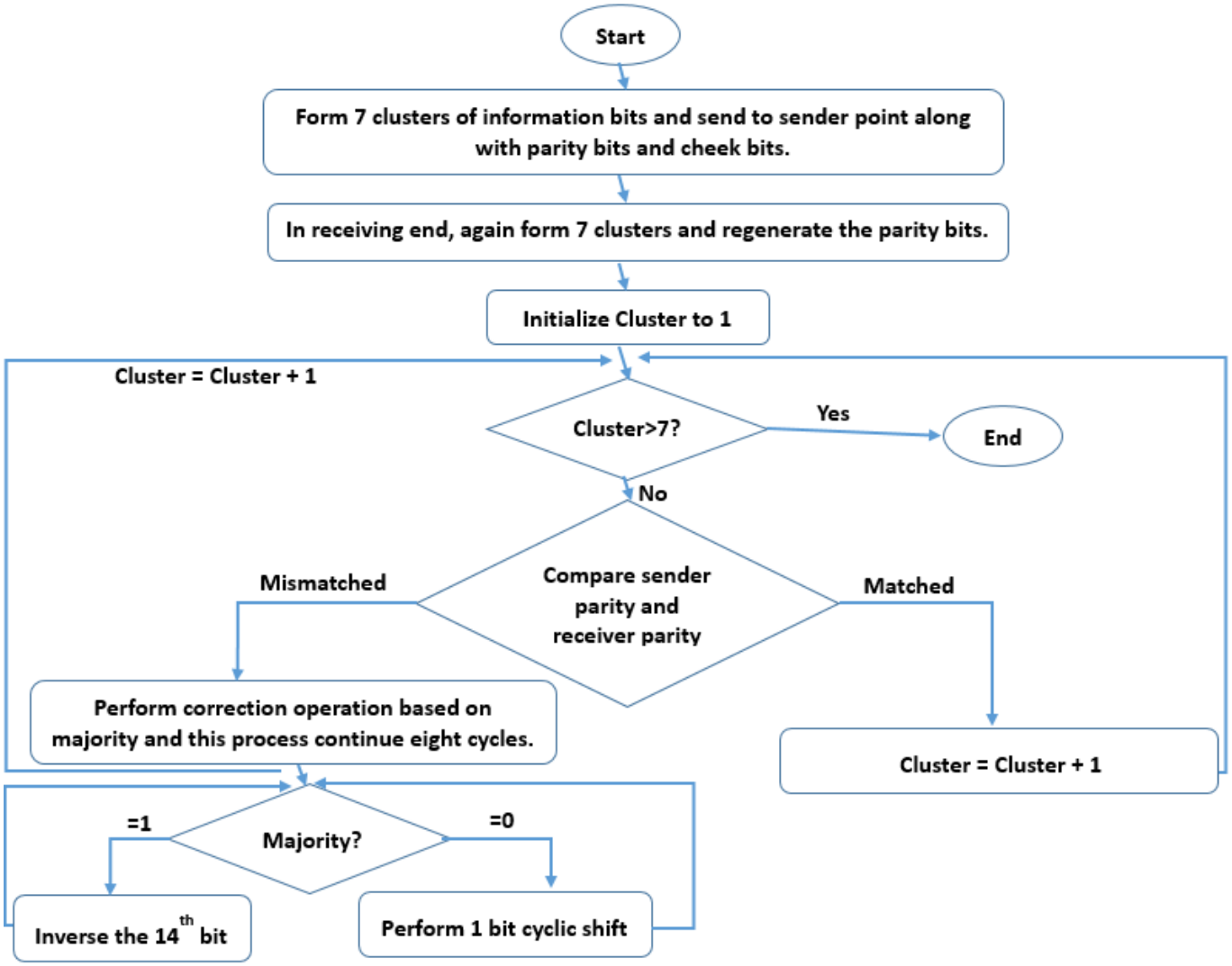

Fig. 14. Flow Chart of the Proposed Method.

In cycle 1, calculate B_1, B_2, B_3 and B_4 using the above (1), (2), (3) and (4). Then check the majority and this cycle we get B_1=0, B_2=0, B_3=1 and B_4 = 0. So majority is 0 and performs one bit cyclic shift and goes to cycle 2. The values of B_1, B_2, B_3 and B_4 are again calculated and this time majority is 1. So according to the proposed algorithm, the 14^{th} bit is inversed and goes to cycle 3. This procedure is repeated till cycle 8 with the two possibilities, one is majority 0 then perform one bit cyclic shift and another is majority 1 then inverse the 14^{th} bit.

After the 8^{th} cycle we can see the original 7 information bits are in last 7 position. Hence, if we do seven right shift then we will get the corrected code word

The corrected code word is: 1 1 1 0 1 1 1 0 1 0 0 0 1 1 1.

After going through the whole process, we will get original information bits as expected to be received. Then from the clusters we obtain the information bits of the form a_1, a_2, a_3, …, a_{49}. Now the overall workflow of the proposed method is shown in Fig. 14 as a flow chart which provides a better overview of the method.

IV. Experimental Analysis

This proposed methodology is experimented through a simulation procedure. The simulation process includes 'error-detection' phase and 'error-correction' phase. It identifies the soft error through the detection phase and appropriately recovers it so that the original stored data is retrieved. In this section, the experimental results of proposed method and other existing methods are represented and discussed. The effectiveness of the proposed method is evaluated in this section.

A. Experimental Tools

The following tools are used for the evaluation process of the proposed method.

- Intel(R) Core i5-2430M CPU @ 2.40 GHz
- CPU RAM 6GB

- Language: Python 3.4
- IDE PyCharm 5.0.1

B. Experimental Result

The outcomes of the experiments are shown in this section along with some comparisons with the existing methods. The results ultimately indicate how the proposed method performs better in terms of the amount of cyclic shift needed. Also it shows that the proposed method performs better to deal with common mode errors or adjacent bit errors while the existing methods are not suitable for this purpose. Fig. 15 shows the comparison of cycle needed for error detection by the plain MLD [8] and existing MLDD [5], and the proposed method.

In all cases MLD [8] occupy 15 cycles to detect errors. In case of MLDD [5], if there is no error then it takes only three cycles to confirm that one. But if there is error, then it takes larger cycles. However, the proposed method requires fewer cycles than MLD [8] and MLDD [5] to detect any error for 14-bitcode word using bit per byte and clustering approach.

Fig. 16 shows the comparison of cycle needed for error correction by the plain MLD [8], existing MLDD [5] and the proposed method.

If an error is detected, MLD takes 15 cycles need to run the entire decoding process. The existing MLDD needs 18 cycles. The existing MLDD has same procedure. However, rather than 15 cycles, three additional cycles are required. The proposed method needs (15+3)/2 cycles that means 9 cycles.

If two-bits error are detected, MLD [8] needs 15 cycle for correction. MLDD [5] needs (15+3) cycles that means 18 cycles but the proposed clustering method it needs 16 cycles.

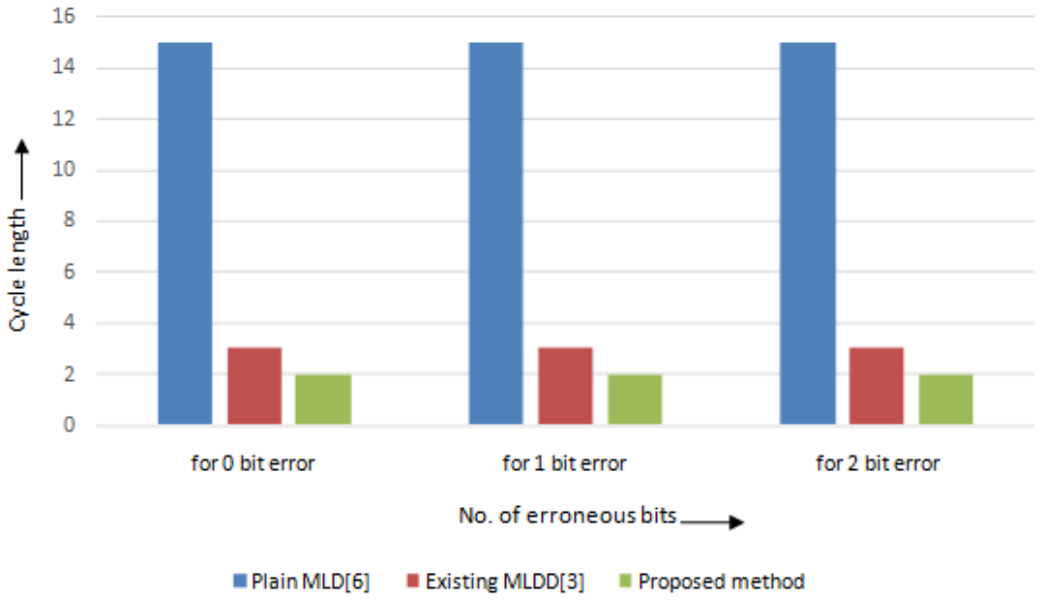

Fig. 15. The Comparison among Plain MLD [6], the Method Proposed by Jayarani et al. [3], and the Proposed Method for Error Detection.

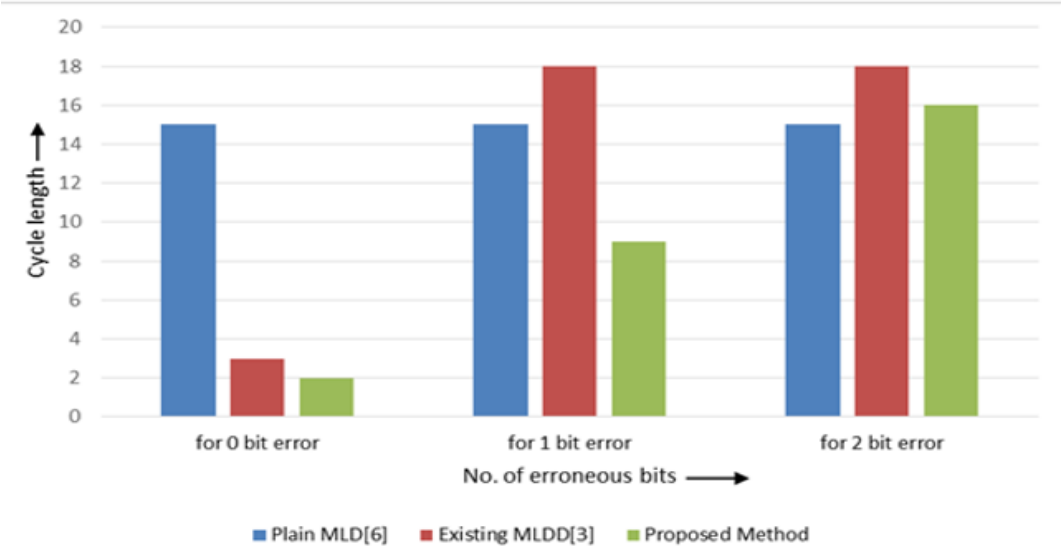

Fig. 16. The Comparison among Plain MLD [8], the Method Proposed by Jayarani et al. [5], and the Proposed7Method for Error Correction.

V. CONCLUSIONS

The proposed methodology focuses on the architecture of a Majority Logic Decoder/Detector (MLDD) with the utilization of bit-per-byte and clustering approaches for fault detection and correction, with decreased cycles. Along with this, the proposed method is very much useful when there are errors in adjacent bits because each adjacent bit is formed in different cluster. So that errors can be easily detected. So, those systems where much possibility to occur adjacent bit error then this proposed method perform better than any other MLDD system with minimum cycle. The proposed method is designed in a way so that it could deal with larger data block. Experiments are performed for large data word to prove its efficiency. To show better performance with larger data block our clustering based approach may consume more time than other methods which are good for smaller data word. The proposed method can detect and correct multiple adjacent cell upsets whereas, the existing cannot perform that. The main limitation is that when multiple errors occur in same cluster then the proposed method can't detect these faulty bits. This proposed method is only focused to detect adjacent error and minimum cycle than the exiting. In the later work, we try to detect and correct errors in same cluster and work with large data block quite faster that this proposed method.

REFERENCES

[1] Shanshan Liu , Jiaqiang Li, Pedro Reviriego , Marco Ottavi, and Liyi Xiao "A Double Error Correction Code for 32-Bit Data Words With Efficent Decoding" in IEEE TRANSACTIONS ON DEVICE AND MATERIALS RELIABILITY, VOL. 18, NO. 1, MARCH 2018.

[2] Jiaqiang Li, et al, "Efficient Implementations of 4-Bit Burst Error Correction for Memories" IEEE Transactions on Circuits and Systems II: Express Briefs.

[3] J. Yang et al., "Radiation-Induced Soft Error Analysis of STT-MRAM: A Device to Circuit Approach," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, no. 3, pp. 380-393, March 2016.

[4] Jing Guo; Liyi Xiao; Tianqi Wang; Shanshan Liu; Xu Wang; Zhigang Mao, "Soft Error Hardened Memory Design for Nanoscale Complementary Metal Oxide Semiconductor Technology,", IEEE Transactions on Reliability, vol.64, no.2, pp.596,602, June 2015.

[5] K.Jayalakshmi, B.Sivasankari, "Reduction of Decoding Time in Majority Logic Decoder for Memory Applications", "International Journal of Innovative Research in Computer and Communication Engineering", Vol.2, Special Issue 1, March 2014.

[6] R.Meenaakshi Sundhari, C.Sundarrasu, M.Karthikkumar, "An Efficient Majority Logic Fault Detection to reduce the Accessing time for Memory Applications","International Journal of Scientific and Research Publications", Volume 3, Issue 3, March 2013.

[7] Shih-Fu Liu, Pedro Revingo, and Juan Antonio Meastro "Efficient majority fault detection with difference set codes for memory applications", IEEE Trans. Very Large Scale Integration. (VLSI) Syst., vol. 20, no. 1, pp. 148–156, Jan. 2012.

[8] H. Naeimi and A. DeHon, "Fault secure encoder and decoder for Nano Memory applications," IEEE Trans.Very Large Scale Integration. (VLSI) Syst., vol. 17, no. 4, pp.473–486, Apr. 2009.

[9] R.C.Baumann,"Radiation-induced soft errors in advanced semiconductor technologies, " IEEE Trans. Device Mater.Reliabil. , vol. 5, no.3, pp. 301–316, Sep. 2005.

[10] C. W. Slayman, "Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations,"IEEE Trans. Device Mater. Reliabil., vol. 5, no. 3, pp. 397–404, Sep. 2005.

[11] Y Kato and T. Morita, "Error correction circuit using difference-set cyclic code," Proceedings of the ASP-DAC Asia and South Pacific Design Automation Conference, 2003.

[12] Costas A. Argyrides, Pedro Reviriego, Dhiraj K. Pradhan and Juan Antonio Maestro "Matrix-Based Codes for Adjacent Error Correction" IEEE Transaction on Nuclear Science (Vol. 57 No.4), August 2010.

[13] Pedro Reviriego, Juan A. Maestro, and Mark F. Flanagan, "Error Detection in Majority Logic Decoding of Euclidean Geometry Low Density Parity Check (EG-LDPC) Codes,"IEEE Transactions On Very Large Scale Integration (Vlsi) Systems 1.

[14] R. J. McEliece, The Theory of Information and Coding. Cambridge,U.K.: Cambridge University Press, 2002.

[15] R. Lucas, M. P. C. Fossorier, Yu Kou, Shu Lin, "Iterative decoding of one-step majority logic deductible codes based on belief propagation", IEEE Transactions on Communications (Volume:4 Issue: 6), June 2000.

[16] Y. Kou, S. Lin, M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results", IEEE Transactions on Information Theory (Volume:47 , Issue: 7), Nov 2011.