A Proficient Method of Error Detection Using Ml Algorithm for Memory Applications

Mrs.S.Ellammal M.E [1], M.Sangeetha [2]

1HOD, ECE, Sri vidya college of engineering and technology, Virudhunagar[1]

[2]PG Student, VLSI Design, Sri vidya college of engineering and technology, Virudhunagar[2]

ABSTRACT: As technology scales the size of the memory used in digital circuits are gradually increasing and memory is responsible for storing and retrieving the data which makes its role very essential. The influence of technology mounting smaller dimensions and lower operating voltages have come to a level that consistency of memories is put into risk, not only in dangerous radiation environments like spacecraft and avionics electronics, but also at normal terrestrial environments. Due to higher integration densities, technology scaling and variation in parameters, performance failures may occur. The memory applications are also prone to single event upsets (SEU) and transient errors which may lead to malfunctions. The proposed error-detection method uses ML algorithm and a special class of Low Density Parity Check (LDPC) codes especially Euclidean geometry Codes (EG) which significantly reduces memory access time when there is no error in the data read.

Keywords: Single event upsets (SEUs), Maximum likelihood algorithm (ML algorithm), Low density parity check (LDPC) codes, Euclidean geometry (EG) codes.

I. INTRODUCTION

Memory applications should be protected from all kinds of faults for reliable performance. Soft error occurs when a radiation event causes enough charge disturbances to reverse or flip the data state of a memory cell. The soft error is also often referred to as a Single Event Upset (SEU). If the radiation event is of a very high energy, more than a single bit may be affected, creating Multi Bit Upset (MBU). For reliable communication, errors must be detected and corrected.

Error detection is the way to find out that is a data is correct or incorrect. As a consequence of augmenting integration densities, there is an increase in the number of soft errors, which produces the need for higher error correction capabilities. The usual multi error correction codes, such as Reed–Solomon (RS) or Bose–Chaudhuri–Hocquenghem (BCH) are not suitable for this task. The reason for this is that they use more sophisticated decoding algorithms, like complex algebraic (e.g., floating point operations or logarithms) decoders that can decode in fixed time, and simple graph decoders, that use iterative algorithms (e.g., belief propagation). Both are very complex and increase computational costs. Reed-Muller is one of the methods of multiple error detection in blocks for digital communications signals.

Hamming codes are often used in today's memory systems to correct single error and detect double errors in any memory word. In these memory architectures, only errors in the memory words are tolerated and there is no preparation to tolerate errors.

Various error detection techniques are used to avoid the soft error. Some commonly used mitigation techniques are:
• Triple modular redundancy (TMR);
• Error correction codes (ECCs).

TMR is a special case of the von Neumann method consisting of three versions of the design in parallel, with a majority voter selecting the correct output. As the method suggests, the complexity overhead would be three times plus the complexity of the majority voter and thus increasing the power consumption. For memories, it turned out that ECC codes are the best way to mitigate memory soft errors. The reason is that they can be easily implemented and cost effective. ECC encodes data in such a way that a decoder can identify and correct certain errors in the data. There are 2 basic types of Error Correction Codes: **Block codes**, referred to as (n,k) codes and **Convolution codes**, where the code words produced depend on both the data message and a given number of previously encoded messages.

Among the ECC codes that meet the requirements of higher error correction capability and low decoding complexity, cyclic block codes have been identified as good one, due to their property of being Majority Logic (ML) decodable. The main reason for using ML decoding is that it is very simple to implement and thus it is very practical and has low complexity. LDPC (Low Density Parity Check) codes belong to the family of ML decodable codes. Euclidean geometry Codes (EGC), one specific type of LDPC codes are used here. These codes are one step majority logic decodable.

A. LOW DENSITY PARITY CHECK (LDPC) CODES

Low Density Parity Check (LDPC) Codes are the class of linear block codes which provide near capacity performance on large collection of data transmission channels while simultaneously feasible for implementable decoders. LDPC codes were first proposed by Gallagher. LDPC codes are also known as Gallagher codes. These are linear error correcting codes and it is a method of transmitting a message over a noisy transmission channel and is constructed using a sparse bipartite graph. LDPC codes are defined by a sparse parity-check matrix. This sparse matrix is often randomly generated, subject to the sparsity constraints.

LDPC codes achieve a remarkable performance with iterative decoding that is very close to the Shannon limit. Consequently, these codes have become strong competitors to turbo codes for error control in many communication and digital storage systems where high reliability is required. LDPC codes have several advantages, which have made them popular in many communication applications like low density of the encoding matrix, easy iterative decoding and generating large code words that can approach Shannon's limit of coding .

An LDPC code is defined as the null space of a parity check matrix H that has the following properties:
1. Each row has ρ number of 1's.
2. Each column has γ number of 1's.
3. The number of 1's that are common between any two columns (λ) is no greater than 1, i.e., $\lambda = 0$ or 1.
4. Both ρ and γ are small compared to the length of the code and the number of rows in H.

As both ρ and γ are very small compared to the code length and the number of rows in the matrix H, H has a low density of 1's. Hence H is said to be a low density parity check matrix and the code defined by H is said to be a low density parity check code. The density of H (r) is defined to be the ratio of the total number of 1's in H to the total number of entries in H — in this case $r = \rho/n = \gamma/J$, where J is the number of rows in H. This kind of LDPC code is said to be a (γ, ρ)-regular LDPC code. If the weights of all the columns or rows in H are not the same, then it is called an irregular LDPC code.

B. EUCLIDEAN GEOMETRY LOW DENSITY PARITY CHECK (EG-LDPC) CODES

Euclidean geometry is a large family in finite geometries. A geometry in which Euclid's parallel postulate holds is called Euclidean Geometry. The parallel postulate states that if two lines are drawn which intersect a third in such a way that the sum of the inner angles on one side is less than two right angles, and then the two lines inevitably must intersect each other on that side if extended far enough. It's sometimes also called parabolic geometry.

II. PREVIOUS APPROACHES

MLD is based on a number of parity check equations which are orthogonal to each other, so that at each iteration, each code word bit only participates in one parity check equation, except the very first bit which contributes to all equations. For this reason, the majority result of these parity check equations decide the correctness of the current bit under decoding. A generic schematic of a memory system is depicted in Fig. 1 for the usage of an ML decoder. Initially, the data words are encoded and then stored in the memory. When the memory is read, the code word is then fed through the ML decoder before sent to the output for further processing. In this decoding process, the data word is corrected from all bit-flips that it might have suffered while being stored in the memory.

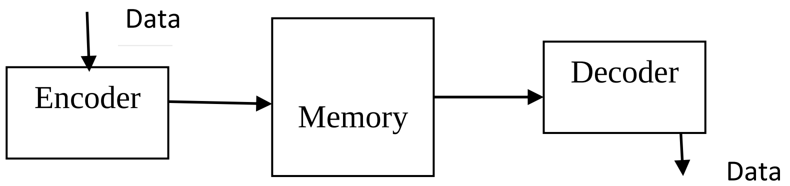

Fig. 1.Simple memory system schematic

There are two ways for implementing this type of decoder. The first one is called the Type-I ML decoder, which determines, upon XOR combinations of the syndrome, which bits need to be corrected. The second one is the Type-II ML decoder that calculates directly out of the code word bits the information of correctness of the current bit under decoding. Both are quite similar but when it comes to implementation, the Type-II uses less area, as it does not calculate the syndrome as an intermediate step.

A. Plain ML Decoder

The ML decoder is a simple and powerful decoder, capable of correcting multiple random bit-flips depending on the numbers of parity check equations.
It consists of four parts:

- A cyclic shift register;
- XOR matrix;
- Majority gate; and
- XOR for correcting the code word bit under decoding, as illustrated in Fig. 2.

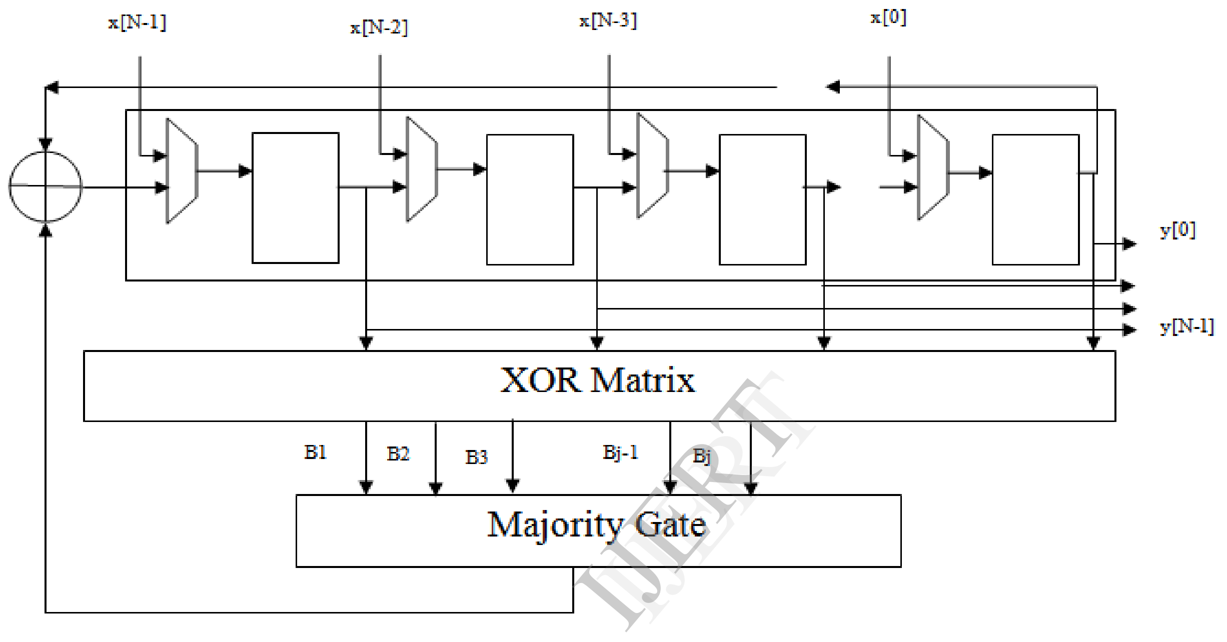

Fig. 2. Majority Logic Detector

In the next step, the content of the registers are rotated and the above procedure is repeated until all code word bits have been processed. Finally, the parity check sums should be zero if the code word has been correctly decoded.

B. Majority Logic Decoder/Detector (MLDD)

Another existing methodology is based on using Difference-set Cyclic Codes (DSCCs). This code is part of the LDPC codes, and, based on their attributes, they have the following properties:

- Ability to correct large number of errors;
- Sparse encoding, decoding and checking circuits synthesizable into simple hardware;
- Modular encoder and decoder blocks that allow an efficient hardware implementation;
- Systematic code structure for clean partition of information and code bits in the memory.

An important thing about the DSCC is that its systematical distribution allows the ML decoder to perform error detection in a simple way, using parity check sums. However, when multiple errors accumulate in a single word, this mechanism may misbehave, as explained in the following.

In general, the decoding algorithm is still the same as the one in the plain ML decoder version. The difference is that, instead of decoding all code word bits by processing the ML decoding during cycles, the proposed method stops intermediately in the third cycle. If in the first three cycles of the decoding process, the evaluation of the XOR matrix for all {*Bj*} is "0", the code word is determined to be error-free and forwarded directly to the output. If the {*Bj*} contain in any of the three cycles at least a "1", this existing method would continue the whole decoding process in order to eliminate the errors.

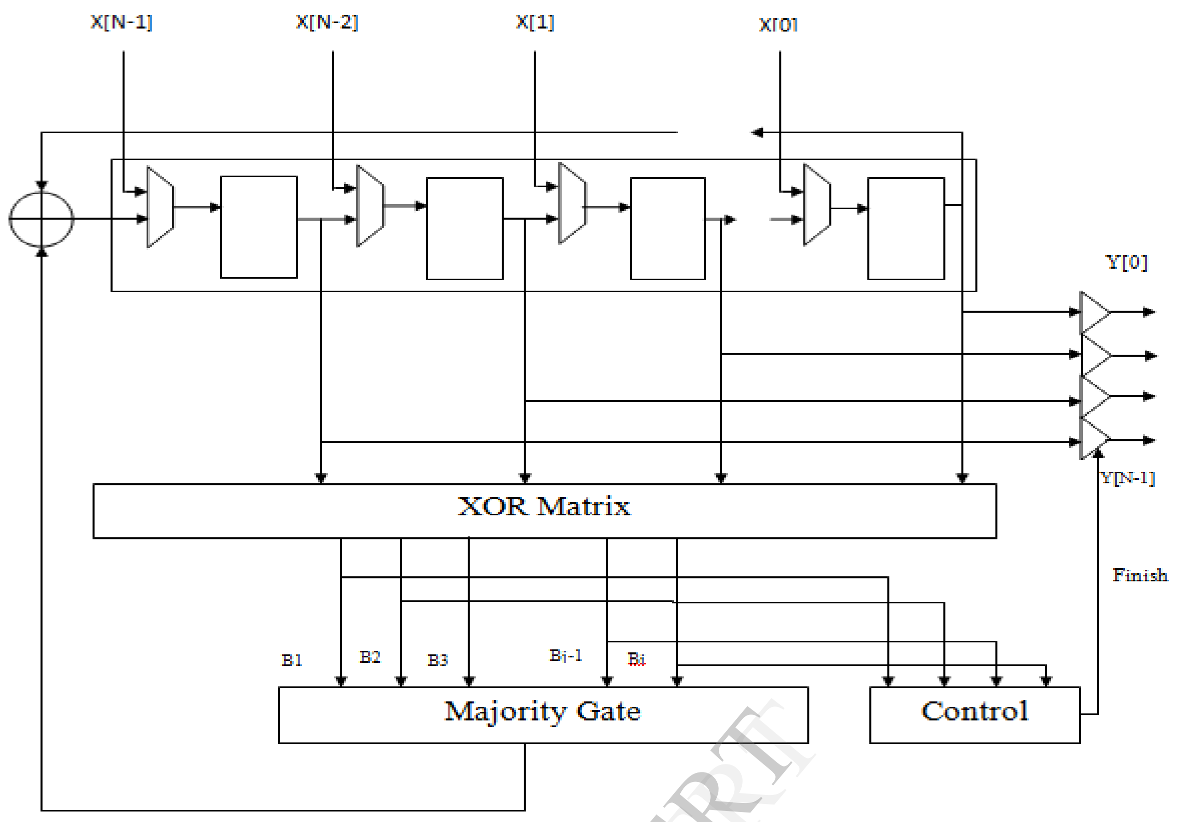

Fig. 3. Improved MLD with control Logic

The additional hardware to perform the error detection is illustrated in above figure as:

- The control unit which triggers a finish flag when no errors are detected after the third cycle and
- The output tristate buffers.

The output tristate buffers are always in high impedance unless the control unit sends the finish signal so that the current values of the shift register are forwarded to the output. The control unit manages the detection process. It uses a counter that counts up to three, which distinguishes the first three iterations of the ML decoding. In these first three iterations, the control unit evaluates the by combining them with the OR1 function. This value is fed into a three-stage shift register, which holds the results of the last three cycles. In the third cycle, the OR2 gate evaluates the content of the detection register. When the result is "0" the FSM sends out the finish signal indicating that the processed word is error-free. In the other case, if the result is "1" the ML decoding process runs until the end.

III.PROPOSED METHODOLOGY

The drawbacks of the existing system are simple parity check logic design is used for detecting errors. In this system, when there is no error was found at first iteration, and then checking was stopped and it will not check for remaining. There was no any condition checking for the error in a word. This may not consider low number of error in a word. But in our proposed work, ML algorithm is used for checking error in a word with the combination of MLD EG-LDPC. This provides controlling mechanism when the bit from word was extracted. High performance rate can be achieved because of this error bits can be reduced. The words which are need to be error checked are arranged bit by bit and it is given to a shift register. The shift register converts the serial bits into parallel bits and it is sent for error checking. By finding the parity bit the errors can be found. The maximum likelihood bit ranges for neighboring bits are found. Then decoding is done using MLD algorithm. After then the error bit is corrected by checking last bit from shift register and bit from majority logic circuit. In this error correction stage, XOR gates are used that changes error bit if there was change in bit. This compare bits from last bit from shift register and bit from majority logic circuit and finally correct error bit. In this way error free output can be obtained. If there is no any error in the word, then it stop cycle checking and goes to next word. In this way millions of error words can be corrected. The block diagram and flow diagram of the proposed methodology are shown below in Fig 4 and Fig 5.

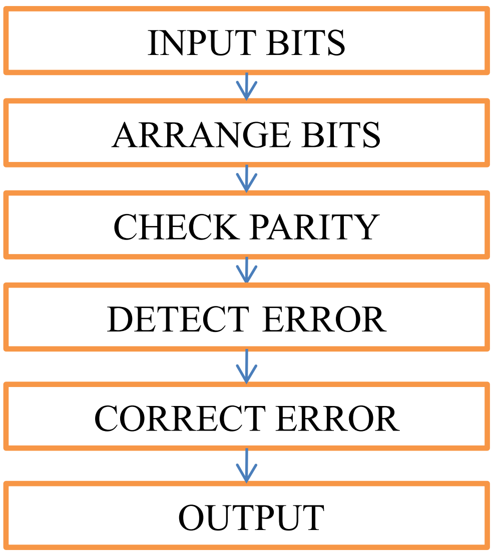

Fig. 4. Block diagram

The input bits are arranged in a bit by bit fashion and then the parity is checked using the check parity block. With the parity bit obtained from the above step the error is detected and decoded using ML algorithm. Then the error is corrected using the correct error block and the proper output is recovered.

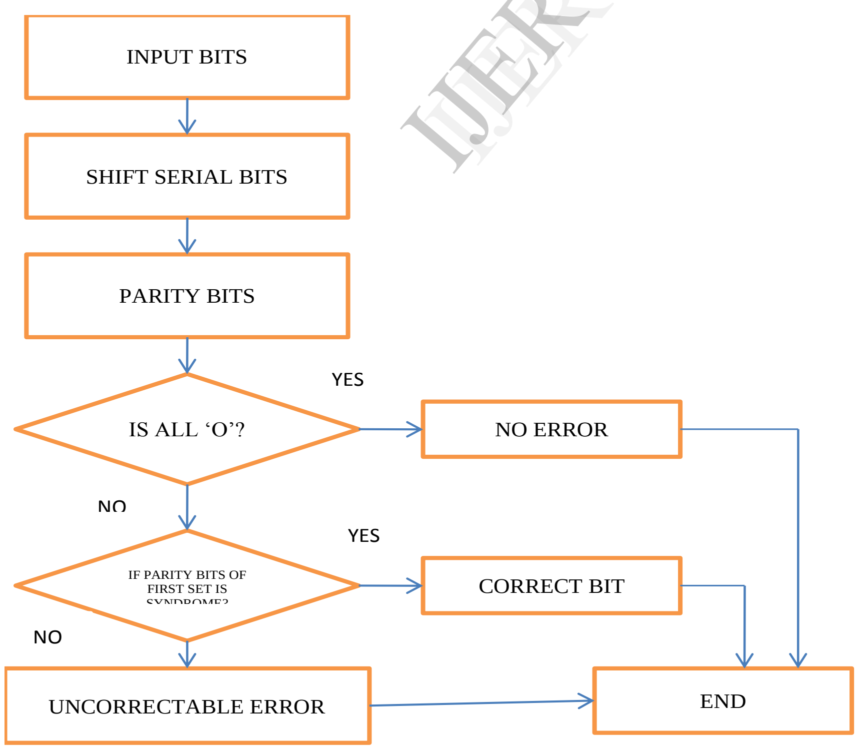

Fig. 5. Flow diagram

VI.EXPERIMEMTAL RESULTS

The design is being simulated using Xilinx 9.2. Until now the first two modules of arranging the input bits and checking the parity has been completed. The simulation results are shown below.

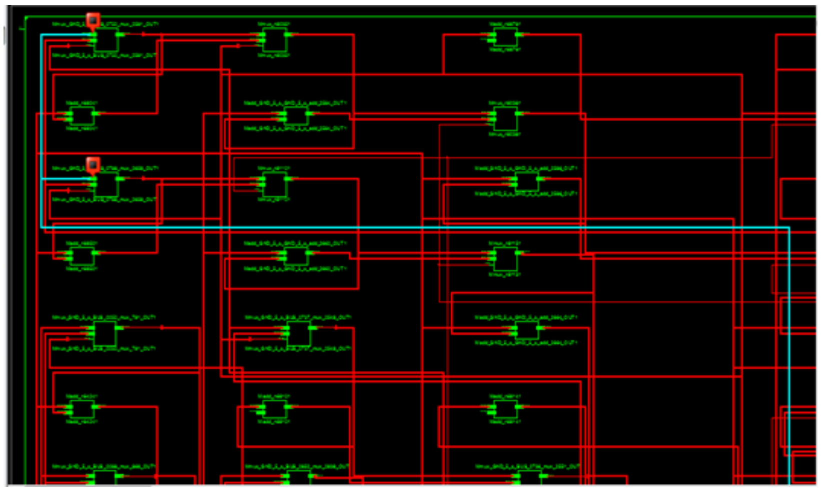

Fig. 6. Schematic showing the arrangement of bits

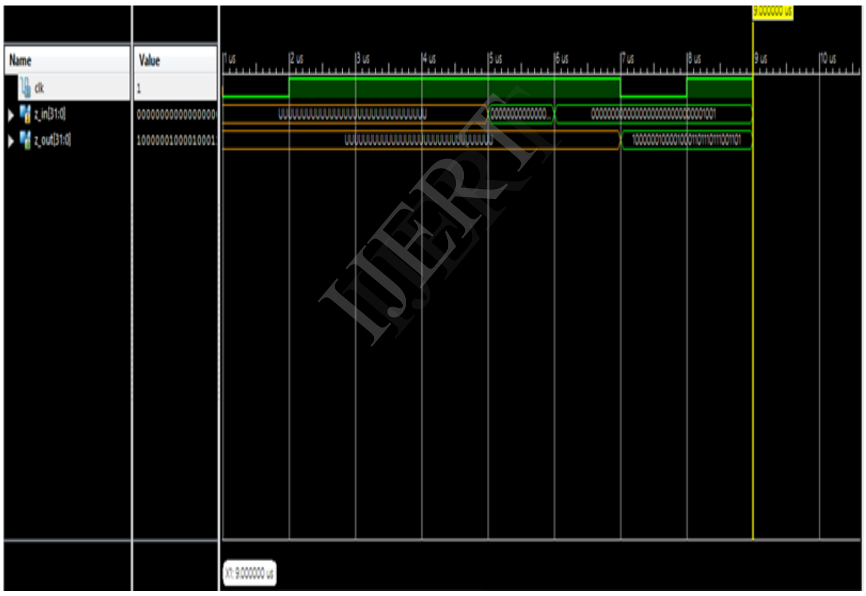

Fig. 7. Graph showing the input bits and the resultant parity bits

VII.CONCLUSION

In this paper, an error-detection and correction mechanism has been presented based on ML algorithm using the EG- LDPC codes. This was made by providing a controlling mechanism in the stage of parity decoding. This improves the performance of the design with respect to the traditional other approaches .Using this mechanism, the decoding and detection of errors can be done simultaneously. This circuit can be implemented in various tools and the results can be compared.

ACKNOWLEDGMENT

I express my sincere gratitude to my guide Mrs.S.Ellammal for guiding my research work throughout the process and i extend my thanks to Sri Vidya college of Engineering for providing the required lab facilities and promoting this research work.

.

REFERENCES

- H. Naeimi and A. DeHon, "Fault secure encoder and decoder for memory applications," in Proc. IEEE Int. Symp. Defect Fault Toler. VLSI Syst., 2012, pp. 409–417.
- S. Liu, P. Reviriego, and J. 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.
- R. Naseer and J. Draper, "DEC ECC design to improve memory reliability in sub-100 nm technologies," Proc. IEEE ICECS, pp. 586–589, 2011.
- M. A. Bajura, Y. Boulghassoul, R. Naseer, S. DasGupta, A. F.Witulski, J. Sondeen, S. D. Stansberry, J. Draper, L. W. Massengill, and J. N. Damoulakis, "Models and algorithmic limits for an ECC-based approach to hardening sub-100-nm SRAMs," IEEE Trans. Nucl. Sci., vol. 54, no. 4, pp. 935–945, Aug. 2010.
- S. Ghosh and P. D. Lincoln, "Dynamic low-density parity check codes for fault-tolerant nano-scale memory," presented at the Foundations Nanosci. (FNANO), Snowbird, Utah, 2009.