A Robust and Efficient Method For Error Detection And Correction In Memories

Jayarani M.A
M.E VLSI Design Scholar
Sri Ramakrishna Engineering College
Coimbatore, India

M.Jagadeeswari, PhD.
Prof and Head, Dept. of M.E VLSI Design
Sri Ramakrishna Engineering College
Coimbatore, India

ABSTRACT
Due to higher integration densities, technology scaling and variation in parameters, the performance failures may occur for every application. The memory applications are also prone to single event upsets and transient errors which may lead to malfunctions. The paper deals with the idea of a novel fault detection and correction technique using EG-LDPC codes with the application mainly focused on memories. The majority logic decoding is used here, since it can correct a large number of errors. Even though the majority decoding consumes more time, it can be overcome by the proposed technique which detects the errors in less cycle time. It can obviously reduce memory access time when the data read process is error free. The use of an additional logic results in a slight area overhead in proposed method when compared to the existing technique, which is overcome by a modified implementation of majority gate. The results obtained are compared with the existing version of the technique.

Keywords
Majority logic decoding; error correction codes (ECCs); Euclidean geometry low-density parity check (EG-LDPC); memory.

1. INTRODUCTION
Memories are the most universal component today. They are prone to errors like soft and transient errors. Some type of embedded memory, such as ROM, SRAM, DRAM, flash memory etc is seen in almost all system chips. Now days, the memory failure rates are increasing due to the impact of technology scaling-smaller dimensions, high integration densities, lower operating voltages etc.[4],[5]. The ability to quickly determine that a bit has flipped is key to high reliability and high availability applications. Some commonly used error detecting techniques are Triple Modular Redundancy (TMR) and Error Correction Codes (ECCs).

The TMR triplicates all the memory parts of the system and to choose the correct data using a voter. This method have disadvantage of large area and complexity overhead of three times. Therefore the ECC became the best way to mitigate soft errors in memory [4].

The most commonly used ECC codes are Single Error Correction (SEC) codes that can correct one bit error in a memory word. Due to consequence of augmenting integration densities, there is an increase in soft errors which points the need for higher error correction capabilities [1], [3]. More advanced ECCs has been proposed for memory applications but even Double Error Correction (DEC) codes with a parallel implementation results in a significant power consumption penalty. The usual multi error correction codes, such as Reed–Solomon (RS) or Bose Chaudhuri–Hocquenghem (BCH) are not suitable for this task due to complex decoding algorithm.

Cyclic block codes have the property of being majority logic (ML) decodable. Therefore cyclic block codes have been identified as more suitable among the ECC codes that meet the requirements of higher error correction capability and low decoding complexity. Euclidean geometry low-density parity check (EG-LDPC) codes, a subgroup of the low-density parity check (LDPC) codes, which belongs to the family of the ML decodable codes, is focused here.

The advantages of ML decoding are that it is very simple to implement and thus it is very practical and has low complexity. The drawback of ML decoding is that, it needs as many cycles as the number of bits in the input signal, which is also the number of taps, N, in the decoder and also same decoding time for both error and error free code words. This is a great impact on the performance of the system, depending on the size of the code.

Another alternative is to first detect if there are errors in the word and only perform the rest of the decoding process when there are errors. This greatly reduces the average power consumption as most words will have no errors. Error detection in a block code can also be implemented by computing the syndrome and checking whether all its bits are zero [15]. By calculating the syndrome, we can implement a fault detector for an ECC is but this also would add an additional complex functional unit. This paper focus on using the MLD circuitry itself as an error detecting module therefore with no additional hardware the read operations could be accelerated.

The remainder of this paper is organized as follows. Section II gives an overview of existing ML decoding solutions. Section III presents the novel ML detector/decoder (MLDD) using EG-LDPC cyclic codes. Section IV discusses the results obtained in respect to speedup, delay and power consumption. Finally, Section V discusses conclusions and future work.

2. MAJORITY LOGIC DECODING (MLD) SOLUTONS
Among the error correction technique one step majority correction is a fast and relatively efficient with low complexity error-correcting technique [6]. One-step-majority correctable ECC codes are limited which include type-I two-dimensional EG-LDPC.

The data flow of memory system schematic with MLD is that the word is first encoded and is then written to the memory [2]. After the reading process of the memory it is passed to a majority logic detector block which detects and corrects the errors which occurred while the reading code word.

This type of decoder can be implemented in two ways. The first one is called the Type-I ML decoder, which determines the bits need to be corrected from the XOR combinations of the syndrome, [9]. The Type-II ML decoder that calculates the information of correctness of the current bit under decoding, directly out of the codeword bits [6]. Both are quite similar, but when implementation is considered the Type-II uses less area, since it does not have a syndrome calculation as an intermediate step. For this reason the paper focus on this type II implementation.

2.1 Existent Plain ML Decoder

The existent plain majority decoder have the method of working in which from the received codeword itself the correct values of each bit under decoding can directly found out. This method consists of mainly two steps-
1) Generating a specific set of linear sums of the received vector bits using the xor matrix 2) Determining the majority value of the computed linear sums. It is the majority logic output which determines the correctness of the bit under decoding. If the majority output is ‘1’, then the bit is inverted, otherwise would be kept unchanged.

As described before, the ML decoder is powerful and simple decoder, which has the capability of correcting multiple random bit-flips depending on the number of parity check equations. It consists of four parts: 1) a cyclic shift register; 2) an XOR matrix; 3) a majority gate; and 4) an XOR for correcting the codeword bit under decoding. The circuit implementing a serial one-step majority logic corrector [6], [12] for (15, 7, 5) EG-LDPC code is shown in Figure 1.

The cyclic shift register is initially stored with the input signal x and shifted through all the taps. The results {Bj} of the check sum equations from the XOR matrix is calculated from the intermediate values in each tap. In the Nth cycle, the result would reach the final tap, producing the output signal, which is the decoded version of input [2].

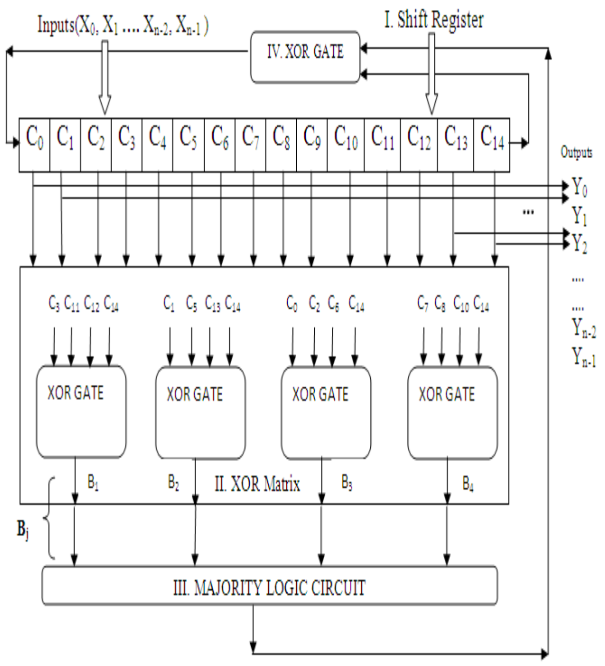
Inputs(X_0, X_1 .... X_{n-2}, X_{n-1})
I. Shift Register
IV. XOR GATE
C_0 C_1 C_2 C_3 C_4 C_5 C_6 C_7 C_8 C_9 C_{10} C_{11} C_{12} C_{13} C_{14}
Outputs
Y_0
Y_1
Y_2
Y_{n-2}
Y_{n-1}
C_3 C_{11} C_{12} C_{14}
C_1 C_5 C_{13} C_{14}
C_0 C_2 C_6 C_{14}
C_7 C_8 C_{10} C_{14}
XOR GATE
XOR GATE
XOR GATE
XOR GATE
B_1
B_2
II. XOR Matrix
B_3
B_4
B_j
III. MAJORITY LOGIC CIRCUIT

Figure 1. Serial one-step majority logic corrector for (15, 7, 5) EG-LDPC code

This is the situation of error free case. The input x might correspond to wrong data corrupted by a soft error or SEUs. The decoder is designed to handle this situation as follows.
From the parity check sum equations hardwired in the xor matrix the decoding starts at the very next moment after the codeword x are loaded into the cyclic shift register. The linear sum outputs $\{B_j\}$ is then forwarded to the majority logic circuit which determines the correctness of the bit under decoding. If the majority of the B_j bits are “1” that is greater than the majority number of zeros then the current bit is erroneous and should be corrected, otherwise it is kept unchanged.

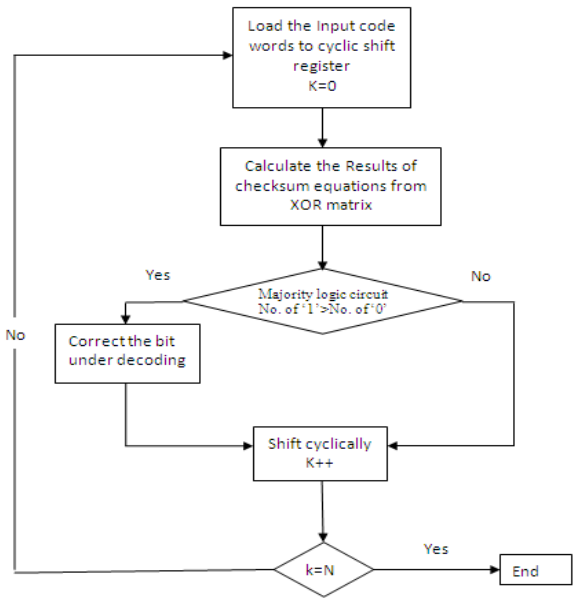
Load the Input code words to cyclic shift register K=0
Calculate the Results of checksum equations from XOR matrix
Yes
No
Majority logic circuit No. of '1'>No. of '0'
No
Correct the bit under decoding
Shift cyclically K++
k=N
Yes
End

Figure 2. Flow diagram of the ML algorithm

The process is repeated and contents of the shift registers are rotated up to the whole N bits of the codeword are processed. When all the parity check sums outputs are zero the codeword is correctly decoded. Further details on how this algorithm works can be found in [6], [12]. The whole algorithm [2] is depicted in Figure 2. The algorithm needs as many cycles as the number of bits in the input signal, which is number of taps, N, in the decoder and also needs same decoding time for both error and error free code words.

3 ML DETECTOR/DECODER

A novel version of the ML decoder for improving performance is presented here. With reference to the original ML decoder, the proposed ML detector/decoder (MLDD) has been implemented using the Euclidean geometry low-density parity check (EG-LDPC) codes. The EG-LDPC codes are based on the structure of Euclidean geometries over a Galois field. Among EG-LDPC codes there is a subclass of codes that is one step majority logic decodable (MLD) [12].The Figure 3 shows the memory system schematic of proposed MLDD [1].

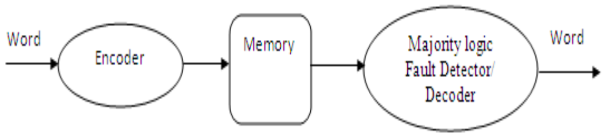

Figure 3. Schematic of a memory system with MLDD

The proof of the hypothesis that all error will be detected in three cycles is very complex from the mathematical point of view. It is practical to generate and check all possible error combinations for codes with small words and affected by a small number of bit flips. When the size of code and the number of bit flips increases, it is difficult to exhaustively test all possible combinations. Therefore the simulations are done in two ways, the error combinations are exhaustively checked when it is feasible and in the rest of the cases the combinations are checked randomly.

Since it is convenient to first describe the chosen design and also for simplicity, let us assume that the hypothesis is true, that only three cycles are needed to detect all errors affecting up to four bits [12] in EG LDPC Codes

.

3.1 Design Structure of encoder

The systematic generator matrix to generate (15, 7, 5) EG-LDPC code is shown in Figure 4 [6]. The encoded vector mainly consists of two parts, the first part consist of information bits and second part is the parity bits, where each parity bit is simply an inner product of information vector and a column of X , from G=[I:X].

The encoder circuit [6] to compute the parity bits of the (15, 7, 5) EG-LDPC code is shown in Figure 5. In this figure, the information vectors are (i_0,….i_6) and will be copied to (c_0,..,c_6) bits of the encoded vector, c. The rest of encoded vector (c_7…c_{14}), that is the parity bits are the linear sums (XOR) of the information bits.

$$
\begin{array}{c|ccccccccccccccc}
 & C_0 & C_1 & C_2 & C_3 & C_4 & C_5 & C_6 & C_7 & C_8 & C_9 & C_{10} & C_{11} & C_{12} & C_{13} & C_{14} \\
\hline
i_0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 \\
i_1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\
i_2 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
i_3 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\
i_4 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
i_5 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \\
i_6 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
\hline
 & \multicolumn{7}{c}{\mathrm{I}} & \multicolumn{8}{c}{\mathrm{X}}
\end{array}
$$

Figure 4. Generator matrix for the (15, 7, 5) EG-LDPC code

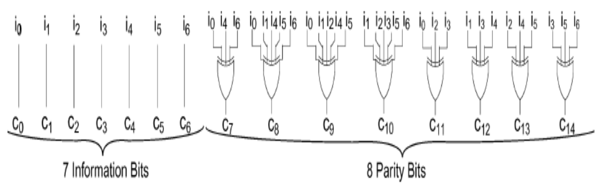

Figure 5. Structure of an encoder circuit for the (15, 7, 5) EG-LDPC code

3.2 Proposed MLDD structure

In general, the proposed version uses the same decoding algorithm as the one in plain ML decoder version. The advantage is that, proposed method stops intermediately in the third cycle when there is no error in data read, [2] as illustrated in Figure. 6, instead of decoding it for the whole codeword size of N. The xor matrix is evaluated for the first three cycles of the decoding process, and when all the outputs {Bj} is “0,”the codeword is determined to be error-free and forwarded directly to the output. On other hand, the proposed method would continue the whole decoding process to eliminate the errors [2] if the {Bj} contain at least a “1” in any of the three cycles.

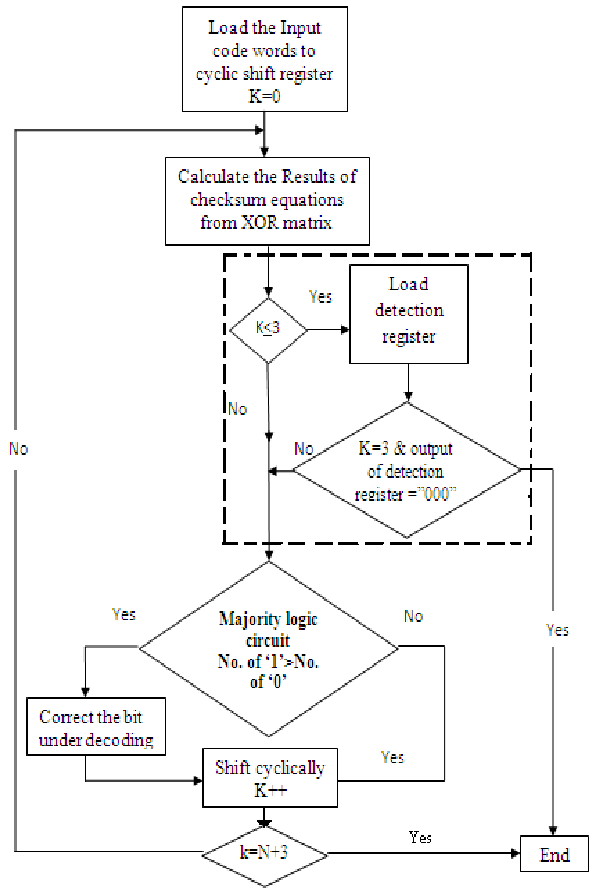
Load the Input code words to cyclic shift register K=0
Calculate the Results of checksum equations from XOR matrix
K≤3
Yes
No
Load detection register
K=3 & output of detection register ="000"
No
Yes
No
Majority logic circuit No. of '1'>No. of '0'
Yes
No
Correct the bit under decoding
Shift cyclically K++
Yes
k=N+3
Yes
End

Figure 6. Flow diagram of the MLDD algorithm.

A detailed schematic of the proposed design for 15 bit code word is shown in Figure 7.

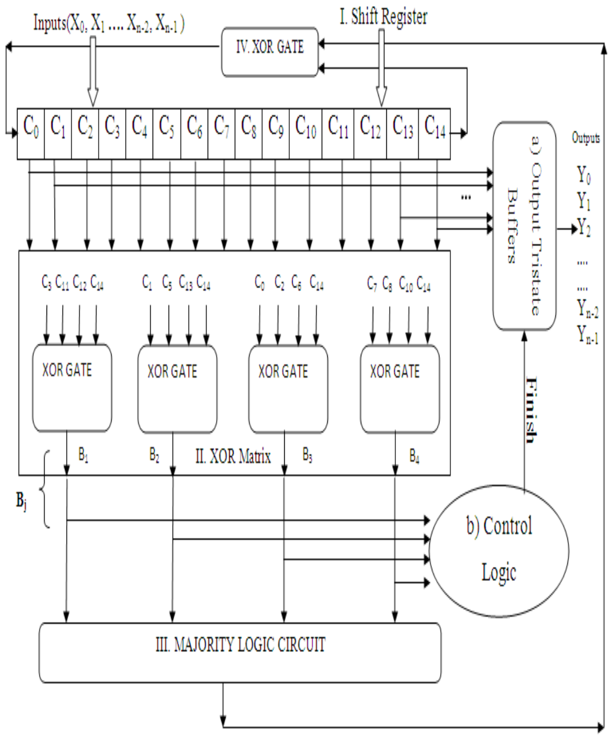

Figure 7. Schematic of the proposed MLDD for 15 bit code word.
a) Control logic b) Output tristate buffers

A detailed schematic of the proposed design for 15 bit code word is shown in Figure 7. The figure shows the basic ML decoder with a 15-tap shift register, an XOR array to calculate the orthogonal parity check sums and a majority logic circuit which will decide whether the current bit under decoding is erroneous and the need for its inversion. The plain ML decoder [2] shown in Figure 1 is also having the same schematic structure up to this stage. The additional hardware [2] intended for fault detection illustrated in Figure 8 are: a) the control logic unit and b) the output tristate buffers. The control unit triggers a finish flag when there is no errors are detected in data read. The output tristate buffers are always in high impedance state until the control unit sends the finish signal so that the current values are forwarded to the output y from the shift register.

The control logic schematic [2] is illustrated in Figure 8. The detection process is managed by the control unit. For distinguishing the first three iterations of the ML decoding, a counter is used here which counts up to three cycles.
The control unit evaluates the output from xor matrix B_j by giving it as input to the OR 1 gate. This output value is fed to

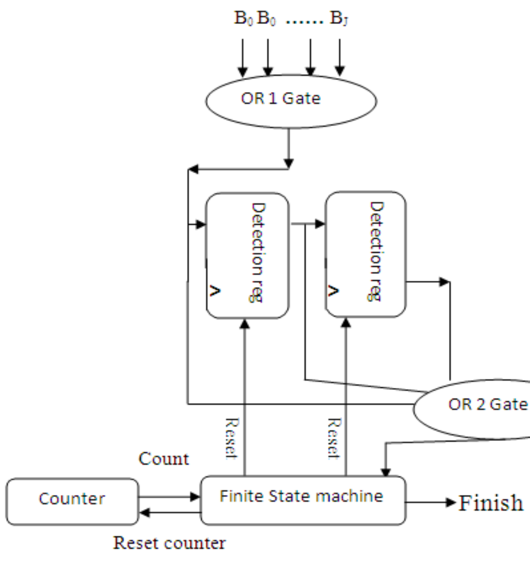

Figure 8. Schematic of the control unit.

two shift registers which has the results of the previous stages stored in it. The values are shifted accordingly. The third coming input is directly forwarded to the OR 2 gate and finally all are evaluated in the third cycle in the OR 2 gate. If the result is "0," a finish signal is send by the FSM which indicates that the processed word is error-free. The ML decoding process runs until the end, if the result is "1".

The majority logic gate is implemented by using the conventional majority logic decoding mechanism. That is two level logic [6].If during the memory read access an error is detected, the xor gate will correct it, by inverting the current bit under decoding.

4 MODIFIED MLDD

Since we are using a separate module for fault detection, there will be a slight area overhead. This area overhead can be overcome by using sorting network in the majority gate.
The EG LDPC code used here is only for 15 bits, it have only outputs four outputs from xor matrix. Therefore the above structure of sorting network in Figure 9 (a) can be used. It takes only four input bits and the vertical lines shown here is comparator Figure 9 (b) which has two inputs and it will compare and larger bit is given to the top output and smaller to bottom respectively [6].

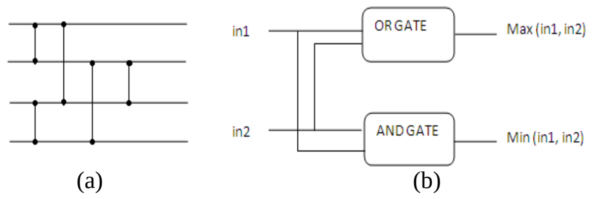

Figure 9. (a)Sorting network-four input (b) Schematic of one comparator

This clearly provides a performance improvement respect to the traditional method which is the existing MLD. [2]. The proposed method mostly would only take three cycles for decoding(five, if we consider the other two for input/output) since most of the words would be error free and would need to perform the whole decoding process only for those words with errors (which should be a minority).

5 EXPERIMENTAL RESULTS

In this section the simulations results of the proposed Majority Logic Decoder/Detector (MLDD) and the existing method Plain MLD is presented. The front end design of the architecture, its simulation, synthesis and comparison are done using XILINX ISE Design Suite 12.3. The target device is Spartan 3E-XC3S500E. The designs are coded in VHDL language. A codeword of size 15 is chosen here for designing.

5.1 Simulation Results

The proposed MLDD and existing MLD techniques are simulated for both error free and erroneous conditions the during memory access, and the results are shown below in figure 10 and 11.

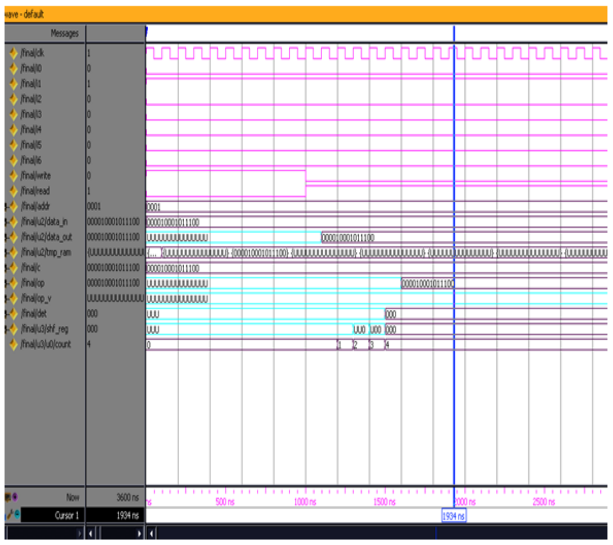

Figure 10 Proposed technique MLDD without fault in memory

5.2 Analysis of Memory Read Access Delay

For the plain MLD, the memory read access delay is directly dependent on the code size, i.e., a code with length 15 needs 15 cycles etc. Then, for I/O two extra cycles needed. On the other hand, for proposed MLDD the memory read access delay is only dependent on the word error rate (WER).

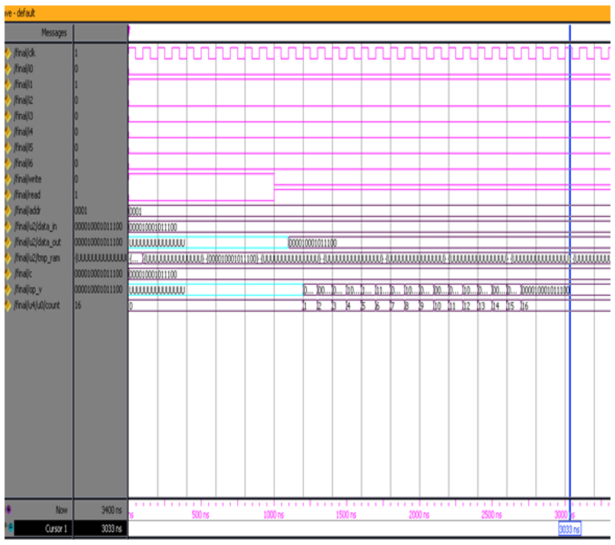

Figure 11 Existing technique MLD without fault in memory read access.

When number of errors is more, then words to fully decode would be more. A summary of the performance of the two different designs is given in Table I.

Table I Comparison of no. of cycles needed for proposed and existing designs

Technique	No. Of Cycles Needed			
	I/O	Error Detection	Cycle at Which the Output Obtained After Detection Process	
			WithoutError	With Error
Plain MLD (Existing)	2	N (E.g N=15)	N+2 (E.g. N=15+2)	N+2 (E.g.N=15+2)
Proposed MLDD	2	3	3+2=5	N+5 (E.g.N=15+5)

The proposed method MLDD designs is been compared with the existent method plain ML decoder (MLD). For the detection of errors, MLD always needs N+2 cycles in all cases. The proposed design just requires three cycles to detect any error (plus two of I/O).

If an error is detected, all of the techniques need to run the whole decoding process. For MLD this represents N+2 cycles (N decoding cycles plus two I/O cycles).The proposed MLDD also have same procedure, but instead of N+2 cycles, three extra cycles are needed (for a total of N+2). In order to simplify the multiplexing logic, these three extra cycles have been added to the process. It has only a negligible impact on performance, but it provides significant savings in area when code size increases.

The Table II gives a comparison of the MLD and MLDD techniques on the speed up of the proposed MLDD for Error free and erroneous Codeword. It gives the idea of how much speed-up can be obtained in an ideal situation.

Table II Speedup comparison for error free code words

Technique	Time At Which The Output Obtained For Error Free Codeword (ns)
Proposed MLDD	1934
Existing MLD	3033

A great speed up can be achieved when the code word does not suffer from errors, (here 1099ns). When the code word does not suffer from errors, it can come out in the next cycle itself without further shifting. When code size increases the speed up will be even larger and when an occurs the extra time period of small fraction of nanoseconds are negligible since most of the situations the memory read access does not make errors.

5.3 Delay Comparison

Proposed MLDD have less delay when compared to existing MLD, since the proposed design detects the faults in just three cycles. The delay comparison is shown in table III.

Technique	Delay (ns)		
	Gate Delay	Net Delay	Total Delay
Proposed MLDD	3.983	8.595	12.578
Existing MLD	3.836	9.17	14.006

Table III Delay comparison

When the code word does not suffer from errors, it can come out in the next 4th cycle itself without further shifting. Therefore this is a great advantage for MLDD in terms of delay and performance.

5.4 Analysis of Power

The concept of low power consumption can be achieved in case of the proposed technique since the proposed design detects the faults in just three cycles. Xilinx X Power Analyzer is used to estimate the total power (dynamic power + quiescent power) for the proposed MLDD and the existing wok of MLD is calculated. Table IV shows the comparison of estimated power.

When the code word does not suffer from errors, it can come out in next 4th cycle itself without further shifting for corrections. Therefore a large no. of clock cycles (here 12 clock cycles) are saved and hence considerable reduction in power is achieved. In the case that an error is detected, both the techniques need to launch the whole decoding process. But in this case also the extra power consumption for the proposed technique is almost negligible.

Table IV Comparison of total estimated power consumption

Technique	Total Power Consumption
Proposed MLDD	47 mW
Existing MLD	49 mW

In the proposed MLDD, there is an extra circuitry of control logic which consists of three shift registers and an xor gate. Therefore there will be a slight area overhead when compared to existing MLD because of this detection logic.

5.5. Analysis of Area Utilization

In the proposed MLDD, there is an extra circuitry of control logic which consists of shift registers and or gates. Table v shows the comparison of total estimated power consumption.

Table V Comparison of equivalent gate count requirements for various techniques

Technique	Total Equivalent gate count requirement
Existent MLD	3197
MLDD	3322
Modified MLDD	2229

Therefore there will be a slight area overhead when compared to existing MLD because of this detection logic. But this is overcome by MLDD using sorting network.

6. CONCLUSION

The paper focuses on the design of a Majority Logic Decoder/Detector (MLDD) for fault detection along with correction of fault, suitable for memory applications, with reduced fault detection time.

From the simulation results, (A codeword of size 15 is chosen here for designing), when compared to the existing MLD, The proposed MLDD has comparatively less delay of 12.578 ns and can detect the presence of errors in just 3 cycles even for multiple bit flips.

It has found that for error detection and correction (for codeword of 15), when comparing to the existing technique, a speed up of about 1100 ns is obtained when there is no errors in data read access. It's because the fault detection needs only three cycles and after the detection of an error free condition, the codeword is passed to the output without further corrections. This is a great saving of time since most of the situations the memory read access does not make errors. Therefore there is a considerable reduction in the memory access time.

The proposed MLDD have about 4% low power consumption than the existing MLD technique, since the proposed design detects the faults in just three cycles. Therefore a large no. of clock cycles (here 12 clock cycles) are saved and hence considerable reduction in power is achieved.

MLDD error detector is designed as it is independent of the code word size and inference about area is that for large values of code word size, the area overhead of the MLDD actually decreases with respect to the plain MLD technique. i.e., for large values of code word size both areas are practically the same. Therefore the proposed MLDD will be an efficient design for fault detection and correction.

.

7 . ACKNOWLEDGMENTS

The authors would like to thank All India Council for Technical Education (AICTE), India, to financially support this work under the Grant 8023/BOR/RID/RPS-48/2009-10. The authors would also like to thank the Management and Principal of Sri Ramakrishna Engineering College, Coimbatore for providing excellent computing facilities and encouragement

.8. REFERENCES

[1] R. Naseer and J. Draper, "DEC ECC design to improve memory reliability in sub-100 nm technologies," in *Proc. IEEE ICECS*, 2008, pp.586–589.

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

[3] M.A. Bajura *et al.*, "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. 2007

[4] 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.

[5] 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.

[6] 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.

[7] S. Ghosh and P. D. Lincoln, "Low-density parity check codes for error correction in nanoscale memory," SRI Comput. Sci. Lab. Tech. Rep. CSL-0703, 2007.

[8] B. Vasic and S. K. Chilappagari, "An information theoretical frame work for analysis and design of nanoscale fault-tolerant memories based on low-density parity-check codes," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 11, pp. 2438–2446, Nov. 2007.

[9] S. Lin and D. J. Costello, *Error Control Coding*,2nd ed Englewood Cliffs, NJ: Prentice-Hall, 2004.

[10] T. Kuroda, M. Takada, T. Isobe, and O. Yamada, "Transmission scheme of high-capacity FM multiplex broadcasting system," *IEEE Trans. Broadcasting*, vol. 42, no. 3, pp. 245–250, Sep. 1996.

[11] Y Kato and T. Morita, "Error correction circuit using difference-set cyclic code," in *Proc. ASP-DAC*, 2003, pp. 585-586.

[12] 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 (will be published).

[13] C.Tjhai, M. Tomlinson, M. Ambroze, and M. Ahmed, "Cyclotomic idempotent- based binary cyclic codes," *Electron. Lett.*, vol. 41, no. 6, Mar. 2005.

[14] T Shibuya and K. Sakaniwa, "Construction of cyclic codes suitable for iterative decoding via generating idempotents," *IEICE Trans. Fundamentals*, vol. E86-A, no. 4, pp. 928–939, 2003.