



International Journal of Emerging Technologies in Computational and Applied Sciences (IJETCAS)

www.iasir.net

Majority Logic Fault Detection with Difference-Set Codes for Memory Applications

Meera.R and Shajimon K John

Department of Electronics & Communication Engineering

SaintGits College of Engineering

Pathamuttom P.O., Kottayam, Kerala -686 532
India

Abstract: Reliability, Availability and Serviceability are the three important parameters to be satisfied by any application. With further reduction in transistor size that leads to smaller dimensions, higher integration densities, and lower operating voltages, the reliability of memories is put into jeopardy. Single event upsets (SEUs) altering digital circuits are becoming a bigger concern for memory applications.

An error-detection method for difference-set cyclic codes with majority logic decoding is presented. Majority logic decodable codes are suitable for memory applications due to their capability to correct a large number of errors. They are simple to implement and have modular encoders and decoders. However, they require a large decoding time that impacts memory performance.

The proposed fault-detection method significantly reduces memory access time when there is no error in the data read. The technique uses the majority logic decoder itself to detect failures, which makes the area overhead minimal and keeps the extra power consumption low.

Keywords: Majority Fault Detection; Error correction codes; Difference set cyclic codes.

I. Introduction

The impact of technology scaling is affecting the reliability of memory applications not only in extreme radiation environments like spacecraft and avionics electronics, but also at normal terrestrial environments. Especially, SRAM memory failure rates are increasing significantly, therefore posing a major reliability concern for many applications[1]. The reliability, availability, and serviceability (RAS) of the system to perform to customer expectations are a strong function of how the system is designed to respond to hard and soft failures. Soft errors or failures are defined as any upset of a semiconductor device that does not lead to permanent damage. Conversely, hard errors or failures are defined as a permanent change in the physical characteristics of the device [2].

A. Techniques to mitigate Soft Errors

By far, the most effective method of dealing with soft errors in memory components is by employing additional circuitry for error detection and/or correction. In its simplest form, error detection consists of adding a single bit to store the parity (odd or even) of each data word (regardless of word length). Whenever data are retrieved, a check is run comparing the parity of the stored data to its parity bit. If a single error has occurred, the check will reveal that the parity of the data does not match the parity bit. Thus, the parity system allows for the detection of a soft error for a minimal cost in terms of circuit complexity and memory width (only a single bit is added to each word). The two disadvantages of this system are that the detected error cannot be corrected and if a double error has occurred then the check will not reveal that anything is wrong since the parity will match. This is true for any even number of errors. For example, if the data were stored with odd parity, the first error changes the odd parity to even parity (detectable error), but the second error changes the parity back to odd (non detectable error) [3].

In order to address these shortcomings, error detection and correction (EDAC) or error correction codes (ECC) is employed. Typically, error correction is achieved by adding extra bits to each data vector encoding the data so that the “information distance” between any two possible data vectors is, at least, three.

II. Literature Survey

A. Desired ECC Properties

Among the ECC codes that meet the requirements of higher error correction capability and low decoding complexity necessary for mitigating soft errors, cyclic block codes have been identified as good candidates, due to their property of being majority logic (ML) decodable [8]. MLD is based on a number of parity check equations which are orthogonal to each other, so that, at each iteration, each codeword 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. 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. The drawback of ML decoding is that, for a coded word of N -bits, it takes N cycles in the decoding process, posing a big impact on system performance [7]. One way of coping with this problem is to implement parallel encoders and decoders. This solution would enormously increase the complexity and, therefore, the power consumption. As most of the memory reading accesses will have no errors, the decoder is most of the time working for no reason. This has motivated the use of a fault detector module [12] that checks if the codeword contains an error and then triggers the correction mechanism accordingly. In this case, only the faulty code words need correction, and therefore the average read memory access is speeded up, at the expense of an increase in hardware cost and power consumption. A similar proposal has been presented in [16] for the case of flash memories. The simplest way to implement a fault detector for an ECC is by calculating the syndrome, but this generally implies adding another very complex functional unit. [1] explores the idea of using the ML decoder circuitry as a fault detector so that read operations are accelerated with almost no additional hardware cost. The results show that the properties of DSCC-LDPC enable efficient fault detection.

LDPC codes have a limited number of 1's in each row and column of the matrix; this limit guarantees limited complexity in their associated detectors and correctors making them fast and light weight.

B. *Difference Set Cyclic Codes (DSCCS)*

DSCC 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 [1].

C. *Existent Majority Logic Decoding (MLD) Solutions*

1) *Plain ML Decoder*

As described before, the ML decoder is a simple and powerful decoder, capable 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 input signal is initially stored into the cyclic shift register and shifted through all the taps. The intermediate values in each tap are then used to calculate the results $\{B_j\}$ of the check sum equations from the XOR matrix. In the N^{th} cycle, the result has reached the final tap, producing the output signal (which is the decoded version of input) [1].

2) *Plain MLD With Syndrome Fault Detector (SFD)*

In order to improve the decoder performance, alternative designs may be used. One possibility is to add a fault detector by calculating the syndrome, so that only faulty code words are decoded [12]. Since most of the code words will be error-free, no further correction will be needed, and therefore performance will not be affected. Although the implementation of an SFD reduces the average latency of the decoding process, it also adds complexity to the design.

III. *Proposed ML Detector / Decoder*

In the proposed Majority Logic Detector/Decoder (MLDD), the data words are encoded using DSCC and then stored in the memory. When the memory is read, the code word is then fed through the ML detector/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.

A. *HYPOTHESIS*

The proposed technique is based on the following hypothesis:

“Given a word read from a memory protected with DSCC codes, and affected by up to five bit-flips, all errors can be detected in only three decoding cycles.”[1]

This is a huge improvement over the simpler case, where N decoding cycles are needed to guarantee that errors are detected. The proof of this hypothesis is very complex from the mathematical point of view. Therefore, two alternatives can be used in order to prove it, which are given here.

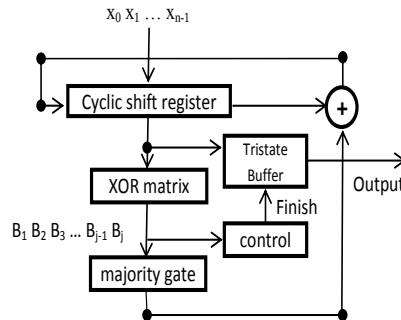
- Through simulation, in which exhaustive experiments have been conducted, to effectively verify that the hypothesis applies (see Section IV of reference [1]).
- Through a simplified mathematical proof for the particular case of two bit-flips affecting a single word (see Appendix of reference [1]).

Let us assume that the hypothesis is true and that only three cycles are needed to detect all errors affecting up to five bits. The input signal is initially stored into the cyclic shift register and shifted through the taps. The intermediate values in each tap are then used to calculate the results $\{B_j\}$ of the check sum equations from the XOR matrix. If in the first three cycles of the decoding process, the evaluation of the XOR matrix for all $\{B_j\}$ is

“0,” the codeword is determined to be error-free and forwarded directly to the output. If the $\{B_j\}$ contain in any of the three cycles at least a “1,” the proposed method would continue the whole decoding process in order to eliminate the errors [1].

B. MLDD DESIGN

Figure 1 Schematic of the proposed MLDD



The figure 1 shows the basic ML decoder with an N-tap shift register, an XOR array to calculate the orthogonal parity check sums and a majority gate for deciding if the current bit under decoding needs to be inverted. The control unit triggers a finish flag when no errors are detected after the third cycle. 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 y .

C. Example Implementation

1) (7, 3) CYCLIC CODE

Consider the set $P = \{0, 2, 3\}$. This is a difference set of order $Q = 2^1$. We have

$$z(X) = 1 + X^2 + X^3 \quad \dots 3.1$$

$$g(X) = [X^7 + 1] / [h(X)] = 1 + X^2 + X^3 + X^4 \quad \dots 3.2$$

This is a (7, 3) cyclic code.

$$z^*(X) = X^3 \cdot z(X^{-1}) = X^3 + X + 1 \quad \dots 3.3$$

$$\begin{aligned} \text{We obtain } w_0(X) &= X^3 \cdot z^*(X) \\ &= X^3 + X^4 + X^6 \end{aligned} \quad \dots 3.4$$

Cyclically shifting $w_0(X)$ to the right two times and three times we obtain

$$w_1(X) = X + X^5 + X^6 \quad \dots 3.5$$

$$w_2(X) = 1 + X^2 + X^6 \quad \dots 3.6$$

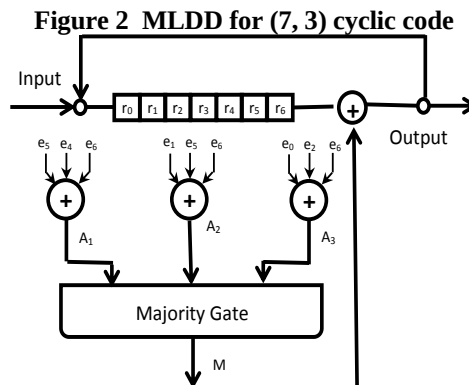
Parity check-sums orthogonal on the bit at position 6, e_6 are

$$A_1 = e_3 + e_4 + e_6 \quad \dots 3.7$$

$$A_2 = e_1 + e_5 + e_6 \quad \dots 3.8$$

$$A_3 = e_0 + e_2 + e_6 \quad \dots 3.9$$

Using the equations 3.7, 3.8 and 3.9, we obtain the detector/decoder based on majority logic decoding for the code as shown in figure 2. The majority gate consists of AND and OR gates that consider all combinations to determine the majority. The gate output M is used to determine if the bit under decoding is right or wrong and to take the necessary action.



In order to compute the syndrome for the code, we consider $h(x)$ to form the parity check matrix as shown:

$$h(X) = 1 + X^2 + X^3 \quad \dots 3.10$$

In systematic form we have

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix} \quad \dots 3.11$$

Let $r = (r_0 \ r_1 \ r_2 \ r_3 \ r_4 \ r_5 \ r_6)$ be the received vector. Syndrome is given by

$$s = (s_0 \ s_1 \ s_2 \ s_3) = r \cdot H^T \quad \dots 3.12$$

Thus we have

$$s_0 = r_0 + r_4 + r_5 \quad \dots 3.13$$

$$s_1 = r_1 + r_5 + r_6 \quad \dots 3.14$$

$$s_2 = r_2 + r_4 + r_5 + r_6 \quad \dots 3.15$$

$$s_3 = r_3 + r_4 + r_6 \quad \dots 3.16$$

2) (21,11) CYCLIC CODE

The set $P = (0, 2, 7, 8, 11)$ is a perfect simple difference set of order $q = 2^2$.

$$\text{Let } z(X) = 1 + X^2 + X^7 + X^8 + X^{11} \quad \dots 3.17$$

The generator polynomial of the difference-set code of length $n = 21$ is

$$g(X) = X^{21} + 1 / H(X) = 1 + X^2 + X^4 + X^6 + X^7 + X^{10} \quad \dots 3.18$$

Thus, the code is a (21, 11) cyclic code.

We have

$$z^*(X) = X^{11}z(X^{-1}) = 1 + X^3 + X^4 + X^9 + X^{11} \quad \dots 3.19$$

$$w_0(X) = X^9 z^*(X) = X^9 + X^{12} + X^{13} + X^{18} + X^{20} \quad \dots 3.20$$

By shifting $w_0(X)$ cyclically to the right 2 times, 7 times, 8 times and 11 times, we obtain

$$w_1(X) = X + X^{11} + X^{14} + X^{15} + X^{20} \quad \dots 3.21$$

$$w_2(X) = X^4 + X^6 + X^{16} + X^{19} + X^{20} \quad \dots 3.22$$

$$w_3(X) = 1 + X^5 + X^7 + X^{17} + X^{20} \quad \dots 3.23$$

$$w_4(X) = X^2 + X^3 + X^8 + X^{10} + X^{20} \quad \dots 3.24$$

Five parity checksums orthogonal on e_{20} are

$$A_1 = S_9 = e_9 + e_{12} + e_{13} + e_{18} + e_{20} \quad \dots 3.25$$

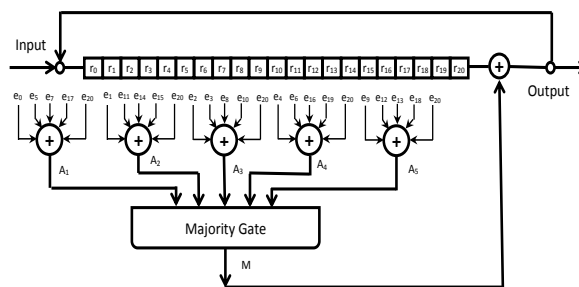
$$A_2 = S_1 = e_1 + e_{11} + e_{14} + e_{15} + e_{20} \quad \dots 3.26$$

$$A_3 = S_4 = S_6 = e_4 + e_6 + e_{16} + e_{19} + e_{20} \quad \dots 3.27$$

$$A_4 = S_0 + S_5 + S_7 = e_0 + e_5 + e_7 + e_{17} + e_{20} \quad \dots 3.28$$

$$A_5 = S_2 + S_3 + S_8 = e_2 + e_3 + e_8 + e_{10} + e_{20} \quad \dots 3.29$$

Figure 3 MLDD for (21, 11) cyclic code



Using above equations and majority gate, we design the majority logic decoder/detector. To design the majority gate for the (21, 11) cyclic code, we consider all combinations of $A_1 A_2 A_3 A_4 A_5$ that results in a majority 1 and obtain the simplified expression for the same.

As before, to calculate the syndrome we use the parity check matrix.

Let $r = (r_0 \ r_1 \ r_2 \ r_3 \ r_4 \ r_5 \ r_6 \ r_7 \ r_8 \ r_9 \ r_{10} \ r_{11} \ r_{12} \ r_{13} \ r_{14} \ r_{15} \ r_{16} \ r_{17} \ r_{18} \ r_{19} \ r_{20})$ be the received vector. The syndrome $s = (s_0 \ s_1 \ s_2 \ s_3 \ s_4 \ s_5 \ s_6 \ s_7 \ s_8 \ s_9)$ is given by

$$s_0 = r_0 + r_{10} + r_{13} + r_{14} + r_{19} \quad \dots 3.30$$

$$s_1 = r_1 + r_{11} + r_{14} + r_{15} + r_{20} \quad \dots 3.31$$

$$s_2 = r_2 + r_{10} + r_{12} + r_{13} + r_{14} + r_{15} + r_{16} + r_{19} \quad \dots 3.32$$

$$s_3 = r_3 + r_{11} + r_{13} + r_{14} + r_{15} + r_{16} + r_{17} + r_{20} \quad \dots 3.33$$

$$s_4 = r_4 + r_{10} + r_{12} + r_{13} + r_{15} + r_{16} + r_{17} + r_{18} + r_{19} \quad \dots 3.34$$

$$s_5 = r_5 + r_{11} + r_{13} + r_{14} + r_{16} + r_{17} + r_{18} + r_{19} + r_{20} \quad \dots 3.35$$

$$s_6 = r_6 + r_{10} + r_{12} + r_{13} + r_{15} + r_{17} + r_{18} + r_{20} \quad \dots 3.36$$

$$s_7 = r_7 + r_{10} + r_{11} + r_{16} + r_{18} \quad \dots 3.37$$

$$s_8 = r_8 + r_{11} + r_{12} + r_{17} + r_{19} \quad \dots 3.38$$

$$s_9 = r_9 + r_{12} + r_{13} + r_{18} + r_{20} \quad \dots 3.39$$

IV. RESULTS

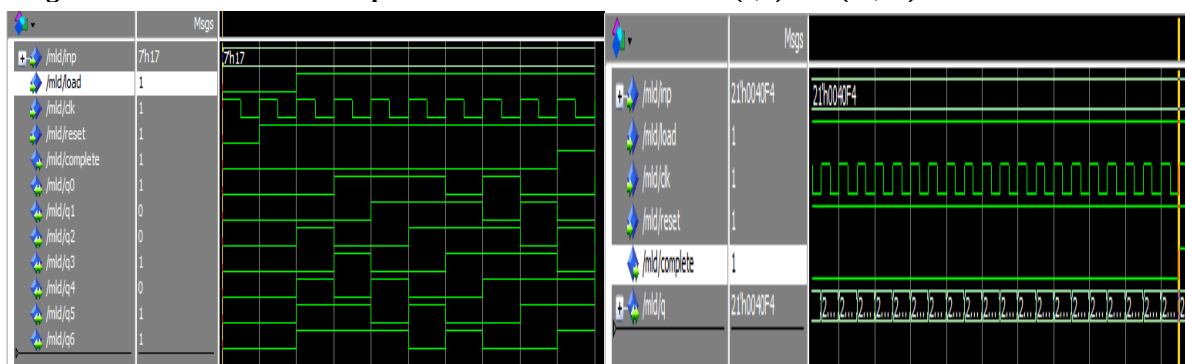
The plain Majority Logic Decoder, Majority Logic Detector / Decoder and Syndrome Fault Detector for both the example codes mentioned before were coded in VHDL and simulated successfully. The tool Leonardo Spectrum offered by Mentor Graphics was used for synthesizing the codes.

A. SIMULATION RESULTS

1) Plain Majority Logic Decoder

The input is provided to the cyclic shift register and after load signal goes high we find that 7 cycles and 21 cycles are needed to decode the code word irrespective of whether the code word is right or wrong for both the codes respectively. The 'Complete' signal signifies when the process is complete and output is available.

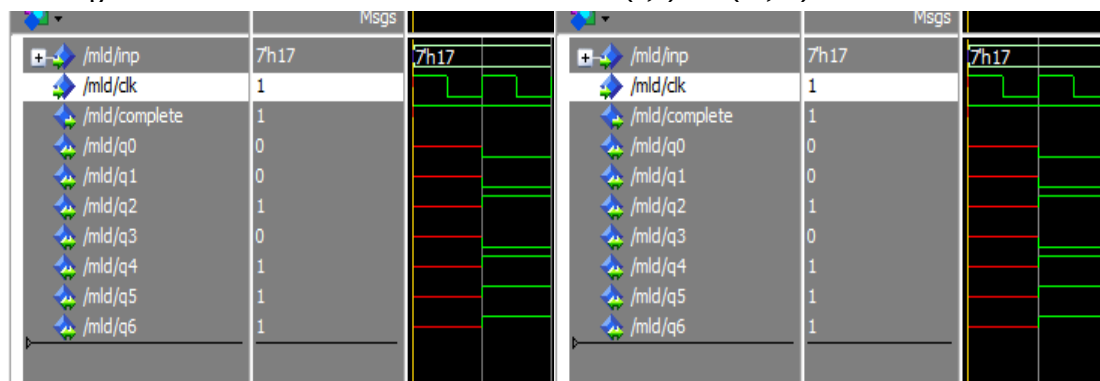
Figure 4 Simulation result for plain ML decoder when correct (7,3) and (21,11) code word is entered



2) SYNDROME FAULT DETECTOR

The input is provided to the SFD and output is available in the first cycle itself when the code word is error free. In case of error like plain MLD 7 cycles and 21 cycles are needed for both codes respectively.

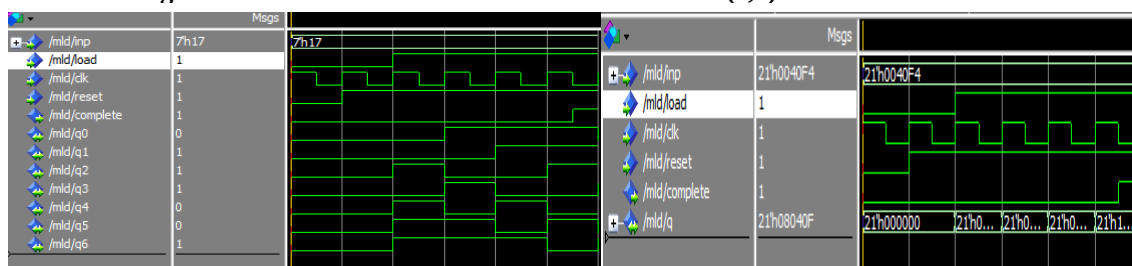
Figure 5 Simulation result for SFD when correct (7,3) and (21,11) code word is entered



3) MAJORITY LOGIC DECODER/DETECTOR

The input is provided to the cyclic shift register and after load signal goes high we find that only 3 cycles are needed to decode the code word if it is error free and 7 cycles and 21 cycles respectively are needed in case of an error. The 'Complete' signal signifies when the process is complete and output is available.

Figure 6 Simulation result for MLDD when correct (7,3) code word is entered



B. Synthesis Results

The previous subsection showed that the performance of the proposed design MLDD is much faster than the plain MLD version, but slightly lower than the design with syndrome calculator (SFD). The three designs have been synthesized using Leonardo Spectrum for N values of 7 and 21 with scl05u library. The results are depicted in number of equivalent gates in Table 1.

Table I Synthesis results of the three designs

	N = 7	N = 21
MLD	228	555
SFD	249	695
Overhead	9.2%	25.22%
MLDD	361	679
Overhead	58.33%	22.34%

The conclusions on the area results are given as follows:

- The MLD design requires little area compared with the other two designs. However, as seen before, the performance results are not very good as all code words require N cycles for decoding.
- The SFD version, which had the best performance, needs more area than the MLD does. Notice that the increment of area grows quicker than N does.
- The MLDD version has a very similar performance to SFD, however it requires a much lower area overhead as the N value increases. The error detector in the MLDD has been designed to be independent of the size of the code, N. The opposite situation occurs, with the SFD technique, which uses syndrome calculation to perform error detection: its complexity grows quickly when the code size increases.

V. Conclusion

In this work, a fault-detection mechanism, MLDD, has been presented based on ML decoding using the DSCCs. Two example codes were simulated and test results show that the proposed technique is able to detect bit-flips in the first three cycles of the decoding process. This improves the performance of the design with respect to the traditional MLD approach.

The MLDD error detector module has been designed in a way that is independent of the code size. This makes its area overhead quite reduced compared with other traditional approaches such as the syndrome calculation (SFD).

The ML decoder circuitry is used here as a fault detector so that read operations are accelerated with almost no additional hardware cost. The results show that the properties of DSCC-LDPC enable efficient fault detection.

References

- [1] Shih-Fu Liu, Pedro Reviriego, and Juan Antonio Maestro, "Efficient majority logic fault detection with difference-set codes for memory applications", IEEE Transactions on Very Large Scale Integration (VLSI) systems, vol. 20, no. 1, January 2012.
- [2] 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.
- [3] 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.
- [4] J. von Neumann, "Probabilistic logics and synthesis of reliable organisms from unreliable components," Automata Studies, pp. 43–98, 1956.
- [5] 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.
- [6] 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.
- [7] S. Lin and D. J. Costello, Error Control Coding, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2004.
- [8] I. S. Reed, "A class of multiple-error-correcting codes and the decoding scheme," IRE Trans. Inf. Theory, vol. IT-4, pp. 38–49, 1954.
- [9] J. L. Massey, Threshold Decoding. Cambridge, MA: MIT Press, 1963.
- [10] 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.
- [11] B. Vasic and S. K. Chilappagari, "An information theoretical framework 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.
- [12] 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.
- [13] Y. Kato and T. Morita, "Error correction circuit using difference-set cyclic code," in Proc. ASP-DAC, 2003, pp. 585–586.
- [14] 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.
- [15] O. Yamada, "Development of an error-correction method for data packet multiplexed with TV signals," IEEE Trans. Commun., vol.COM-35, no. 1, pp. 21–31, Jan. 1987.
- [16] P. Ankolekar, S. Rosner, R. Isaac, and J. Bredow, "Multi-bit error correction methods for latency-constrained flash memory systems," IEEE Trans. Device Mater. Reliabil., vol. 10, no. 1, pp. 33–39, Mar. 2010.

VI. Acknowledgments

I would like to express my gratitude to Prof.Susan Abe, HOD, Electronics and Communication Engineering Department, SAINTGITS college of Engineering and Prof.Shajimon.K.John, my project guide and co-ordinator for their guidance and support.