# Difference Set Codes Based Majority Logic Fault Detection

**E.Jebamalar Leavline**
Department of ECE,
Bharathidasan Institute of Technology, Anna University,
Tiruchirappalli – India,
jebi.lee@gmail.com

**F. Jabeena**
Department of ECE,
Bharathidasan Institute of Technology, Anna University,
Tiruchirappalli – India,
jnsrockz@yahoo.com

**M.Dhivyapriya, V.Kalaiyarasi**
Department of ECE,
Bharathidasan Institute of Technology, Anna University,
Tiruchirappalli – India,

## ABSTRACT
Errors in memory applications such as SRAM, is increased now a days due to technology scaling, higher package density and lower voltages even at normal terrestrial environments. There are several techniques to detect and correct the errors. These techniques can only detect single error and double errors due to the low encoding and decoding complexity. It leads to longer decoding time and large power consumption. In majority logic decoding technique, different set cyclic codes are used for multierror detection/correction. DSCC is a part of LDPC (Low Density Parity Check) codes. Since this technique is independent of code size.

## Keywords
Fault Detection, Difference Set Codes, LDPC, SRAM.

## 1. INTRODUCTION
Due to technology scaling smaller dimensions, higher integration densities, and lower operating voltages has come to a level that reliability of memories is put into jeopardy, not only in extreme radiation environments like spacecraft and avionics electronics, but also at normal terrestrial environments [1].

A soft error occurs when a radiation event causes enough of a charge disturbance to reverse or flip the data state of a memory cell, register, latch, or flip-flop. The error is "soft" because the circuit/device itself is not permanently damaged by the radiation—if new data are written to the bit, the device will store it correctly. The soft error is also often referred to as a single event upset (SEU).

While MBUs are usually a small fraction of the total observed SEU rate, their occurrence has implications for memory architecture in systems utilizing error correction [2], [3]. Another type of soft error occurs when the bit that is flipped is in a critical system control register such as that found in field programmable gate arrays (FPGAs) or dynamic random access memory (DRAM) control circuitry, so that the error causes the product to malfunction [4]. This type of soft error, called a single event interrupt (SEFI).

Some commonly used mitigation techniques are:

- Triple modular redundancy (TMR)
- Error correction codes (ECCs).

Some error correcting techniques [5] are not used to correct more than triple errors. DSCC [6] are used to correct more than triple errors.

This paper is organized as follows. Section II describes about the existing majority logic decoding (mld) solutions. Section III explores the details about the ml detector/decoder. Section IV discusses the simulation results and Section V concludes the paper.

## 2. EXISTING MAJORITY LOGIC DECODING (MLD) SOLUTIONS
All 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. A generic schematic of a memory system is depicted in [7] 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 codeword 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.

### 2.1 Plain ML Decoder
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, as illustrated in Fig. 1

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 of the check sum equations from the XOR matrix. In the cycle, the result has reached the final tap, producing the output signal (which is the decoded version of input).

The input x might correspond to wrong data corrupted by a soft error. To handle this situation, the decoder would behave as follows. After the initial step, in which the codeword is loaded into the cyclic shift register, the decoding starts by calculating the parity check equations hardwired in the XOR matrix. The resulting sums {BJ} are then forwarded to the majority gate for evaluating its correctness. If the number of 1's received in {BJ} is

greater than the number of 0's, that would mean that the current bit under decoding is wrong, and a signal to correct it would be triggered. Otherwise, the bit under decoding would be correct and no extra operations would be needed on it. In the next step, the content of the registers are rotated and the above procedure is repeated until all N codeword bits have been processed. Finally, the parity check sums should be zero if the codeword has been correctly decoded. Further details on how this algorithm works can be found in [8].

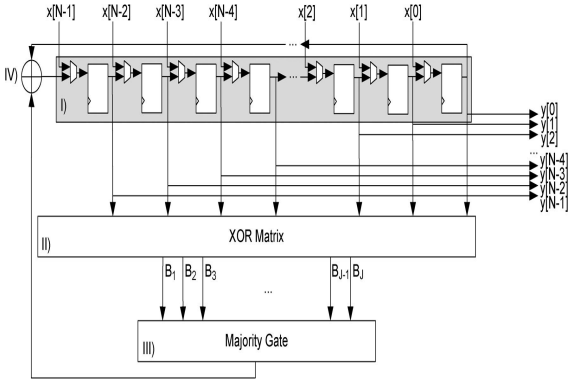


Fig. 1. Schematic of an ML decoder. I) cyclic shift register. II) XOR matrix. III) Majority gate. IV) XOR for correction

## 2.2 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 codewords are decoded [9]. Since most of the codewords 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 fig.4 in [7].

The SFD is an XOR matrix that calculates the syndrome based on the parity check matrix. Each parity bit results in a syndrome equation. Therefore, the complexity of the syndrome calculator increases with the size of the code.Afaulty codeword is detected when at least one of the syndrome bits is "1." This triggers the MLD to start the decoding, as explained before. On the other hand, if the codeword is error-free, it is forwarded directly to the output, thus saving the correction cycles.

# 3. ML DETECTOR/DECODER

The ML detector/decoder (MLDD) has been implemented using the 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.

The use of a simple error detector based on parity check sums does not seem feasible, since it cannot handle "false negatives" (wrong data that is not detected). However, the alternative would be to derive all data to the decoding process

(i.e., to decode every single word that is read in order to check its correctness), as explained in previous sections, with a large performance overhead.

This proposal 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. This is a huge improvement over the simpler case, where N cycles are needed to guarantee that errors are detected. The encoding process is similar to that illustrated in [9].

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 codeword bits by processing the ML decoding during N cycles, the proposed method stops intermediately in the third cycle, as illustrated in Fig. 2.

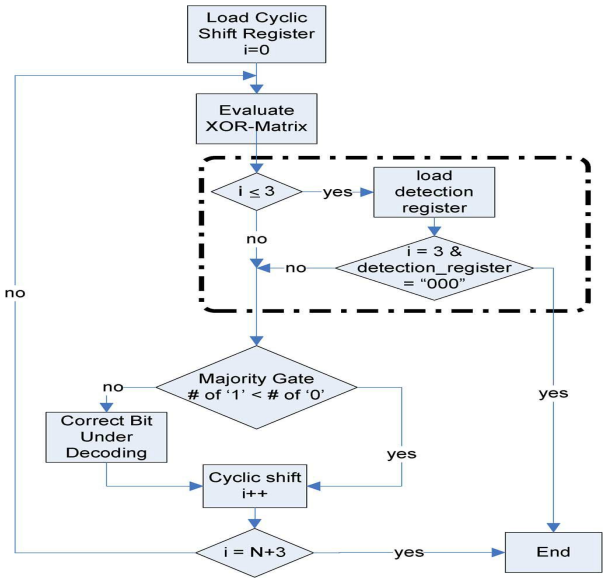


Fig. 2. Flow diagram of the MLDD algorithm.

If in the first three cycles of the decoding process, the evaluation of the XOR matrix for all is "0," the codeword is determined to be error-free and forwarded directly to the output. If the 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.

A detailed schematic of the proposed design is shown in Fig. 3.

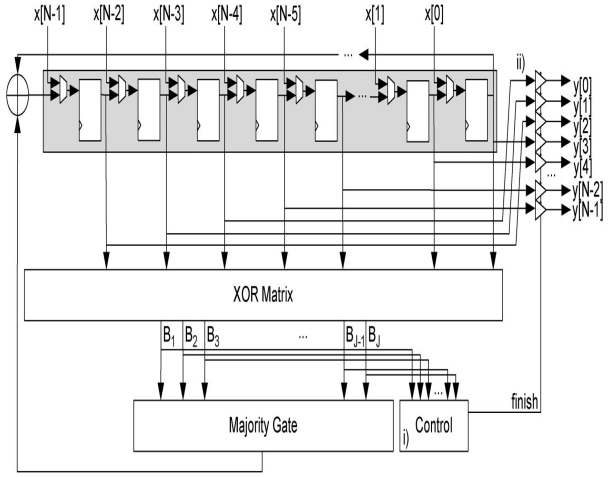


Fig. 3. Schematic of the proposed MLDD. i) Control unit. ii) Output tristate buffers

The figure shows the basic ML decoder with an -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. Here,

i) the control unit which triggers a finish flag when no errors are detected after the third cycle and

ii) 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 schematic is illustrated in [7]. The correction and detection process will be same as in [9]. with small modification

## 4. SIMULATION RESULTS

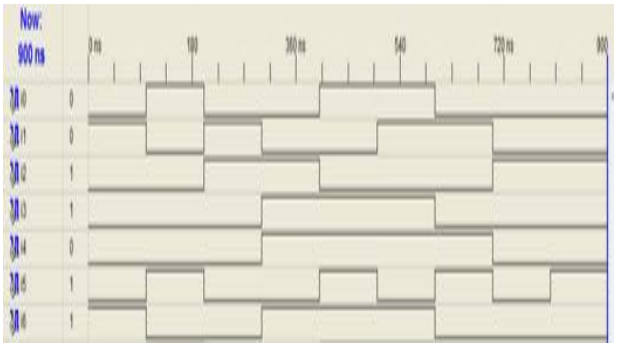

Fig. 4(a). Encoder input

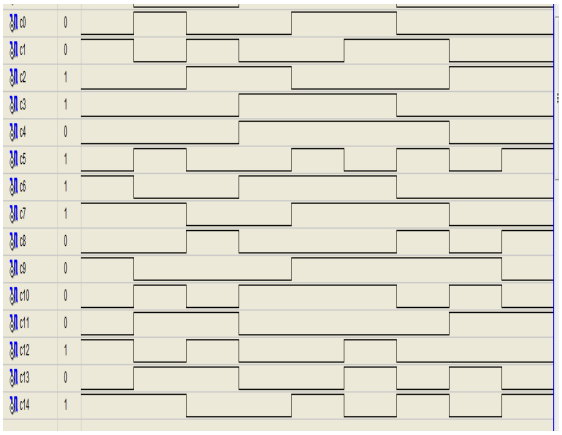

Fig. 4(b).Encoder output

If the encoder input is 0110000, then the encoded output is 011000001001110 (refer fig.4(a) and 4(b)).

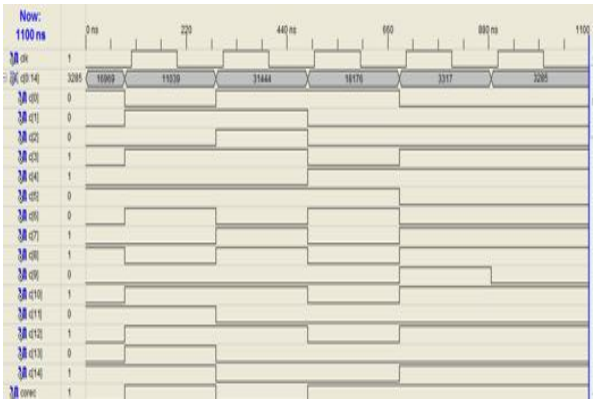

Fig. 5(a).Cyclic shift register input

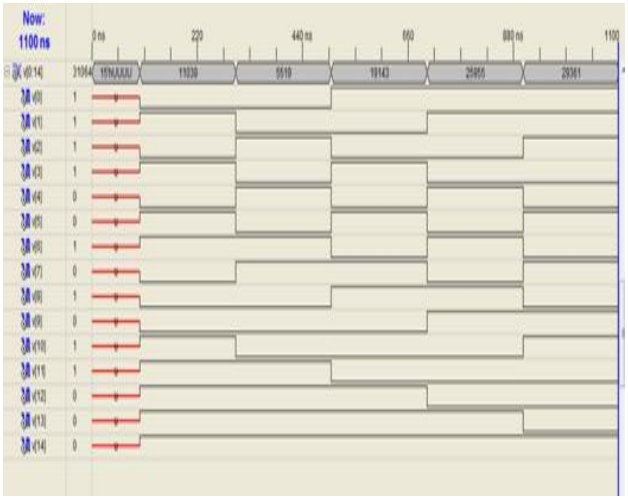

Fig.5(b).Cyclic shift register output

The shift occurs during positive clock only. If cyclic shift register input is 010101100011111(from $c_0$ to $c_{14}$) ,correct bit is 0 then, the output is 001010110001111.

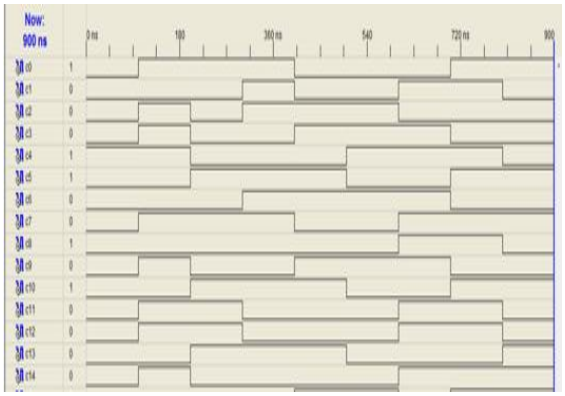

Fig.6(a).XOR matrix input

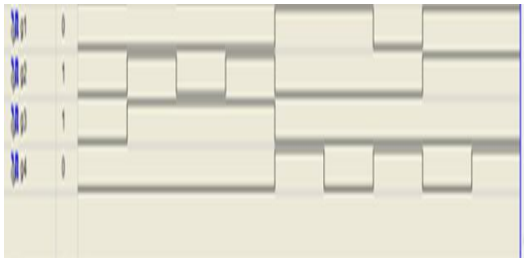

Fig.6(b).XOR matrix output

If XOR matrix input is 101110010101101 (from $c_0$ to $c_{14}$), then XOR matrix output is 0110 (refer fig.6(a) and 6(b)).

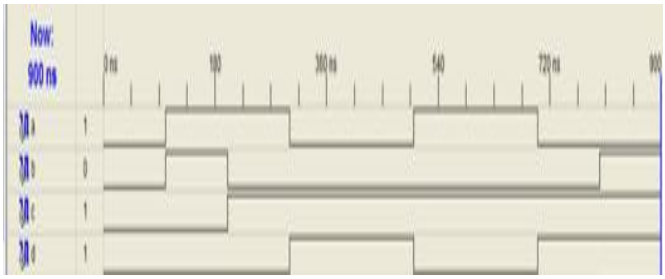

Fig.7(a).Majority gate input

Fig.7(b).Majority gate output

If majority gate input contain more than two 1's, then the output is 1 otherwise 0 (refer fig.7(a) and 7(b)).

## 5. CONCLUSION

This paper presented an study on ML detector/decoder. Errors in memory applications such as SRAM, is increased now a days due to technology scaling, higher package density and lower voltages even at normal terrestrial environments. In majority logic decoding technique, different set cyclic codes are used for multierror detection/correction. DSCC is a part of LDPC (Low Density Parity Check) codes. Since this technique is independent of code size. The simulation results are presented.

## REFERENCES


[1] 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.

[2] S. Satoh, Y. Tosaka, and S. A. Wender, "Geometric effect of multiple-bit soft errors induced by cosmic ray neutrons on DRAM's," IEEE Electron Device Lett., vol. 21, no. 6, pp. 310–312, Jun. 2000.

[3] J. Maiz, S. Hareland, K. Zhang, and P. Armstrong, "Characterization of multi-bit soft error events in advanced SRAMs," in Int. Electro Devices Meeting (IEDM) Tech. Dig., Washington, DC, Dec. 2003, pp. 21.4.1–21.4.4.

[4] R. Koga, S. H. Penzin, K. B. Crawford, and W. R. Crain, "Single Event Functional Interrupt (SEFI) sensitivity in microcircuits," in Proc. 4th Radiation and Effects Components and Systems (RADECS), Cannes, France, Sep. 1997, pp. 311–318.

[5] P. Ankolekar, S. Rosner, R. Isaac, and J. Bredow, "Multi-bit error correction methods for latency-contrained flash memory systems," IEEE Trans. Device Mater. Reliabil., vol. 10, no. 1, pp. 33–39, Mar. 2010.

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

[7] Efficient Majority Logic Fault Detection With Difference-Set Codes for Memory Applications Shih-Fu Liu, Pedro Reviriego, Member, IEEE, and Juan Antonio Maestro, Member, IEEE

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

[9] 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