RESEARCH ARTICLE OPEN ACCESS

High Performance Error Detection and Correction Technique for Memories Using Difference - Set Codes

Teena Ann Thomas[1], Ashad Kumar A[2]
M-Tech Vlsi Design Scholar[1], Professor[2],
Dept of Ece, Met'school of Engg: Mala Thrissur, Kerala

ABSTRACT
As technology scales, Single Event Upsets (SEU) become more common and affect a large number of memory cells. In order to protect memories against SEUs is to make use advanced error detecting and correcting codes are used. An error detection method for difference – set cyclic codes with majority logic is presented in this paper. Majority logic decodable codes have the capability to correct a large number of errors so these codes are used in memory applications. But they require a large decoding time which affects the memory performance. The proposed fault detection technique resolves this problem. It reduces the memory access time when there is no error in the data read. In this technique the Majority Logic Decoder (MLD) itself used to detect failures, thus area become minimal and keeps the extra power consumption low.
Keywords: Error Correcting Code, block codes, majority logic.

I. 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. 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

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. 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. In this paper, we will focus on one specific type of LDPC codes, namely the difference-set cyclic codes (DSCCs), which is widely used in the Japanese teletext system or FM multiplex Broad casting systems

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 [1]. The drawback of ML decoding is that, for a coded word of bits, it takes cycles in the decoding process, posing a big impact on system performance. One way of coping with this problem is to implement parallel encoders and decoders which lead to the increase in complexity and 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 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.

The rest of this paper is organized as follows, Section II gives an overview of existing ML decoder, Majority Logic Detector/Decoder ; Section III presents the modified High performance ML detector/decoder (MLDD) using difference-set cyclic codes; Section IV discusses the results obtained for the different versions in respect to effectiveness, performance, and area. Finally, Section V concludes this paper.

II. EXISTING SYSTEMS

1. PLAIN MAJORITY LOGIC DECODER

MLD is based on a number of parity check equations which are orthogonal to each other, so, for 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. MLD was first mentioned in [1] for the Reed–Muller codes. Then, it was extended and generalized in [2] for all types of systematic linear block codes that can be totally orthogonalized on each codeword bit.

Fig.1 shows memory system with plain MLD, Initially, the data words are encoded and then stored in the memory.

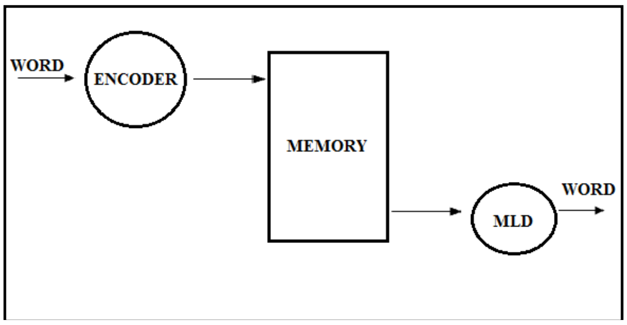

Fig.1 Memory system with plain MLD

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. 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 x 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 y.

If the codeword under decoding is wrong then the decoder 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 $\{B_j\}$ are then forwarded to the majority gate for evaluating its correctness. If the number of 1's received in $\{B_j\}$ 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.

This type of MLD needs as many cycles as the number of bits in the input signal. This is a big impact on the performance of the system, depending on the size of the code. For example, for a codeword of 73 bits, the decoding would take 73 cycles, which would be excessive for most applications.

2. MAJORITY LOGIC DETECTOR/DECODER (MLDD)

A modified version of Plain ML Decoder is the MLDD. This modified MLD is implemented based on the Difference set - cyclic codes, which is a part of LDPC codes [3]. In general, the decoding algorithm is still the same as the one in the plain ML decoder. 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 [4].

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. A detailed schematic of the proposed design is shown in fig.2.

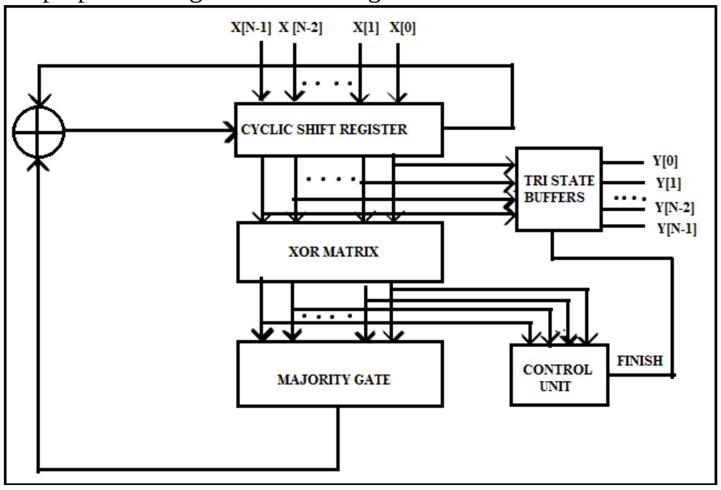

Fig.2 Internal block diagram of MLDD.

This figure 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. Those components are the same as the ones for the plain ML decoder. The additional hardware to perform the error detection is illustrated in Fig. 4 are the control unit and tri state buffers. The control unit which triggers a finish flag when no errors are detected after the third cycle. The output tri state 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 OR_1 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 OR_2 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. This clearly provides a performance improvement respect to the traditional method. But the area becomes high.

III. PROPOSED SYSTEM

High performance Majority Logic Detector/ Decoder (MLDD) is proposed here, this high performance MLDD is the improved version of existing MLDD in terms of area and performance. In this MLDD, decoder itself is used to detect and correct errors, without increasing the circuit complexity. The proposed MLDD is implemented using the difference set cyclic codes (DSCCs).

3.1 BLOCK DIAGRAM

Basic block diagram of High Performance Majority Logic Detector / Decoder (MLDD) is shown in Fig 3.

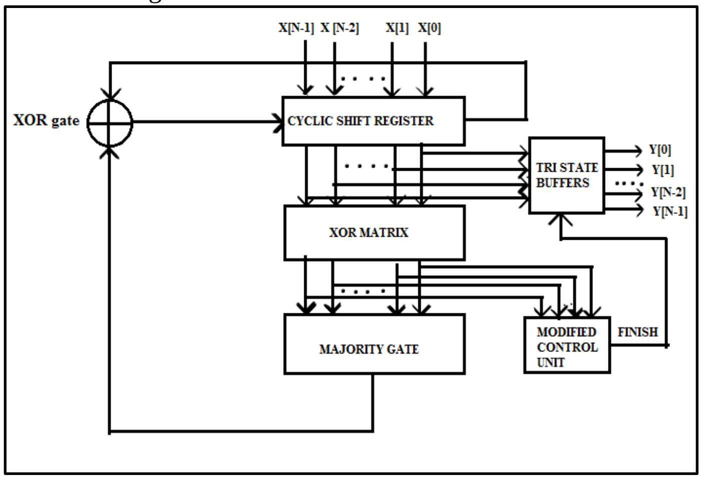

Fig.3. Internal block diagram of high Performance MLDD.

The main components of High Performance MLDD includes,

- Cyclic shift register.
- XOR matrix.
- Majority gate.
- XOR gate.
- Control unit without FSM.
- Tri – state buffers.

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. 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. The Fig 3 shows the basic ML decoder with cyclic 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. Those components are the same as the ones for the plain ML decoder. The additional hardware to perform the error detection is illustrated in Fig 3 as: i) the control unit which triggers a finish flag when no errors are detected after the third cycle and ii) the output tri - state buffers. The output tri - state 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. The flow diagram of MLDD algorithm is shown in Fig. 4.

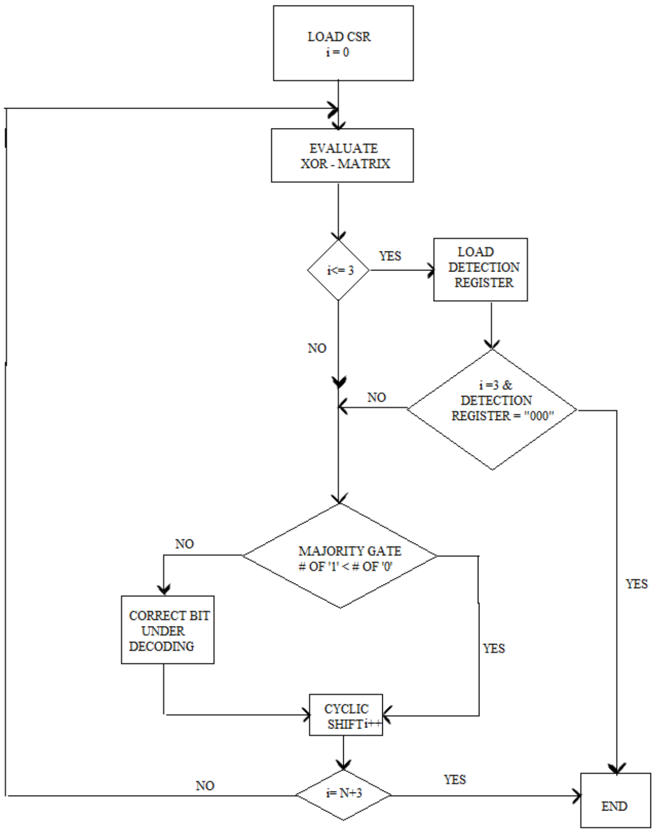

Fig.4. Flow chart of MLDD algorithm.

3.1.1 Cyclic Shift Register

Cyclic shift register is used to shift the data cyclically. For an N- bit input data, decoder requires N tap shift register. The cyclic shift register consists of two modules as follows:

- D flip-flop
- Multiplexer

The input code word bits are stored in the cyclic shift register and shifted to all the registers. It circulates all code word bits of the register around both MSB and LSB ends with no loss of information. The cyclic shift register is designed in parallel in serial out fashion. Initially the selection pin of each mux is kept low and the data is loaded into the flip flops and then selection pin is kept high, during this time input signal is shifted. And input of the first flip flop is xor'ed result of the LSB and output of majority gate.

3.1.2 XOR Matrix

XOR matrix is formed by a polynomial which is based on parity check sum equations. An example of calculating parity check equation is as follows. C Consider a code of length 6: **x**=(x1,x2,x3,x4,x5,x6)

Suppose that
$$\begin{cases} x1+ x2+ x3+ x4 =0 \\ x2+ x3+ x5 =0 \\ x1+ x3+ x6=0 \end{cases}$$

Assign any values to x1, x2, x3, solve for x4,x5,x6

- Parity-check equations

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

$Hx^T = 0$.

Where, H is called a parity-check matrix of the code.

Consider an (n, k) cyclic code C with parity-check matrix H. The row space of H is an (n, n-k) cyclic code of C_d . Then for V in C and W in C_d

$$W \cdot V = w_0v_0 + w_1v_1 + . \ . \ . + w_{n-1}v_{n-1} = 0 \qquad (3.1)$$

The equality is called a parity-check equation.

Suppose the received vector r = v + e

where

$$e = (e_0, e_1, . \ . \ , e_{n-1}) \qquad (3.2)$$

For any vector w in the dual code C_d , we can form the following linear sum of the received digits:

$$A = w \cdot r = w_0r_0 + w_1r_1 + . \ . \ . + w_{n-1}r_{n-1} \qquad (3.3)$$

This is called a parity-check sum or simply check sum. A checksum is a simple type of redundancy check that is used to detect errors in data. If r is a code vector in C, this parity-check sum, A, must be zero; however, if r is not a code vector in C, then A may not be zero.

3.1.3 Majority gate

Majority gate evaluates the result of parity check sum such that If the number of 1's received in $\{B_j\}$ is greater than the number of 0's , which means the current bit under decoding is wrong, and a signal to correct it would be triggered. Signal for correction is developed by the control unit. The output of majority gate is the majority result of the parity check sum equations. Type – 2 ML decoder is used.

3.1.4 Control unit

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. The block diagram of control unit is shown in Fig.5. It consist a counter and a decision block. Counter counts up to three. Decision block consist of one OR gate, three stage shift register and a NOR gate. Result of the parity check sum equation is given to the OR gate and the result of the OR gate is shifted through the three stage shift register and counter stops counting after third count then the content in the shift register is fed to the NOR gate. Output of the NOR gate is high only when all input are low that means code word is error free then output of the NOR gate is given to tri – state buffer. Thus the decision block sends out the finish signal, active high signal to the tri – state buffer. Tri – state buffer is initially at high impedance state and after receiving the finish signal tri – state buffer provide the code word stored in the cyclic shift register directly to the output. In other case, if the output of NOR gate is "0," tri – state buffer remains in the high impedance state and the ML decoding process runs until the end.

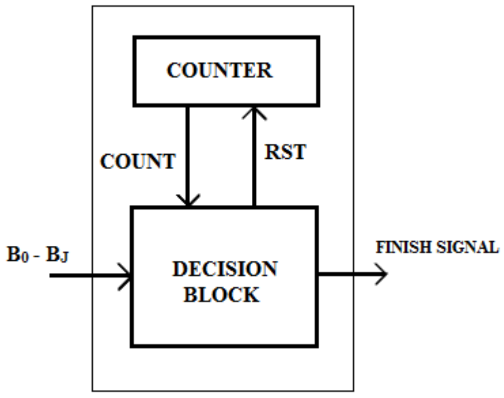

Fig 5.Block diagram of control unit.

3.1.5 Tri – state buffer

Tri- state buffer has three states, active high, active low and high impedance. Initially tri-state buffer is at high impedance state, when control unit sends finish signal then the current value presented in the shift register are forwarded to the output via tri-state buffers. Active high tri – state buffer is used here.

IV. EXPERIMENTAL RESULTS

The proposed High performance MLDD and existing systems simulated using ModelSim 6.3f and implemented using Xilinx 11.1. After implementation area is calculated based on number of gates.

4.1 SIMULATED RESULTS

4.1.1 Plain MLD

Fig. 6 shows the simulated result of plain MLD. This is designed for a code word of size 21 – bit. In this single bit error detection and correction is done. It requires 21 cycles for detecting and correcting a code word affected by an error. This reduces the overall performance of the system associated with the memory.

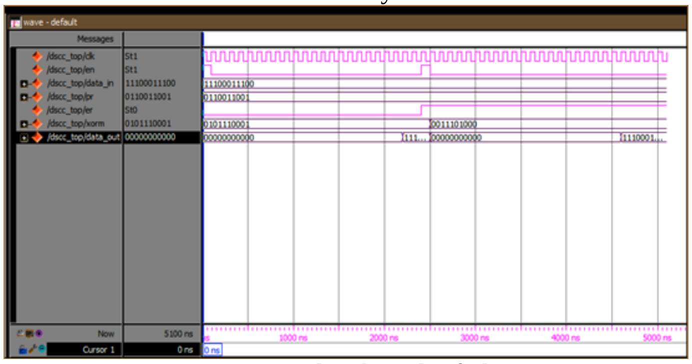

Fig. 6. Simulated result of plain MLD.

4.1.2 Majority Logic Detector/Decoder.

MLDD is simulated and it detected and corrected multi bit errors. Only three decoding cycle is required for detecting a code word is affected by error or not. For correcting a code word affected by error it requires 26 cycles.

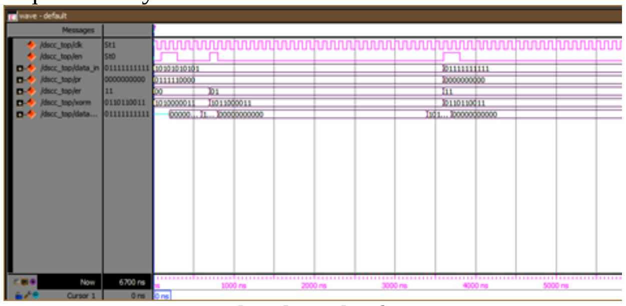

Fig 7.Simulated result of MLDD.

4.1.3 High Performance MLDD

Simulated result of high performance of MLDD is shown in Fig. 8. This high performance MLDD is designed for 21 – bit code word. It required only 3 cycles for detection of an error. Thus it improves the overall performance of the system associated with memory using high performance MLDD technique.

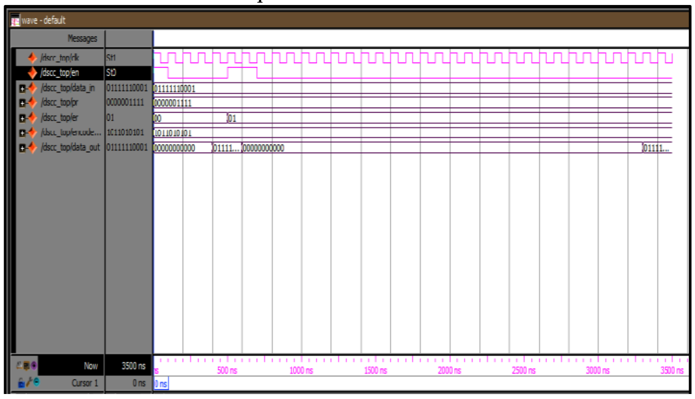

Fig 8. Simulated result of High Performance MLDD.

4.2 Comparison between existing and proposed systems in terms of area

Table1. Provides the number of LUT's and flip flops required for the existing and proposed systems. From the table it is clear that area of the proposed system is less than that of existing MLDD.

TABLE 1
AREA

MODEL	MLD	MLDD	HIGH PERFORMANCE MLDD
No: Of Sliced Ff	26	68	37
No: Of 4 Input LUT	56	100	73
No:Of Bonded IOB	45	46	46

V. CONCLUSION

A new technique for error detection and correction based on Difference Set Cyclic Code (DSCC) with majority logic decoding was proposed. This can be applied for system with high soft error rate. This technique can be able to detect any error pattern in the first three cycles of the decoding process. And correction of detected code word requires N + 5 cycles. Using this technique detection and correction of multi bit errors can be done. This improves the performance of the systems associated with memories employing this design. On other hand, the MLDD error detector module has been designed in a way that is independent code size. This makes its area overhead quite reduced compared with other traditional approaches.

REFERENCES

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

[2] 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.

[3] E. J.Weldon, Jr., "Difference-set cyclic codes," *Bell Syst. Tech. J.*, vol.45, pp. 1045–1055, 1966.

[4] Shih – Fu Liu, Pedro Reviriego, Juan Antonio Maestro, "Efficient majority logic fault detection with Difference – set Codes for memory applications", *IEEE Trans. VLSI Syst.*, vol. 20, no 1, pp. 148 -156, Jan 2012.

[5] 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.

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

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