Asian Journal of Information Science and Technology (AJIST)
and Instumentation Engineering (IJAREEIE)
Vol.2.No.1 2014 pp 19-26.
available at: www.goniv.com
Paper Received :08-03-2014
Paper Accepted:22-03-2013
Paper Reviewed by: 1. R. Venkatakrishnan 2. R. Marimuthu
Editor : Prof. P.Muthukumar

FAULT SECURE MEMORY DESIGN USING DIFFERENCE-SET CODES FOR MEMORY APPLICATIONS

Lakshmanan.V, Vijaya Ganesh.J
NPR College of Engineering and Technology
Dindigul.India

ABSTRACT

Modified decoding algorithms for DS codes are proposed that, in addition to error correction, provide error detection when the number of correctable bit errors is exceeded by one. This combined error detection and correction capability of modified decoder are provide to prevent soft errors from causing data corruption, memories are typically protected with ECCs. Memory applications require low latency encoders and decoders. These codes allow us to design a fault tolerant error-detector unit that detects any error in the received code vector despite having faults in the detector circuitry. The fault secure detector unit to check the output vector of the encoder and corrector circuitry, and if there is any error in the output of either of these units that unit has to redo the operation to generate the correct output vector. Using this detect and repeat technique, correct potential transient errors in the encoder or corrector output and provide fault tolerant memory system with fault tolerant supporting circuitry. The need for fault tolerant systems in terrestrial applications is of growing importance. Unpredictability in the system design, manufacture and operation is of critical importance to the population that these systems affect. Majority logic decodable codes are suited for memory applications due to their capability to correct a large number of errors. However they require a large decoding time that impact memory performance. The 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 over head minimal and keeps the extra power consumption low.

Index Terms— Block codes, difference-set, error correction codes (ECCs), low-density parity check (LDPC), majority logic decoder (MLD), memory, difference-set cyclic codes (DSCCs).

1. INTRODUCTION

Now a days soft errors are makes a major problems in memory applications due to scaling and higher integration densities. These errors not only in extreme radiation environments like space craft and avionics but also at normal terrestrial environments. Especially, SRAM memory failure rates are increasing significantly, therefore posing a major reliability concern for many applications. Some commonly used mitigation techniques are:

• triple modular redundancy (TMR);
• error correction codes (ECCs).

TMR is a special case of the von Neumann method [3] 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 [2]. For terrestrial radiation environments where there is a low soft error rate (SER), codes like single error correction and double error detection (SEC–DED), are a good solution, due to their low encoding and decoding complexity. However, as a consequence of augmenting integration densities, there is an increase

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 [6].

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 candidates, due to their property of being majority logic (ML) decodable [7], [8]. A subgroup of the low-density parity check (LDPC) codes, which belongs to the family of the ML decodable codes, has been researched in [9]–[11]. In this paper, we will focus on one specific type of LDPC codes, namely the difference-set cyclic codes (DSCCs), which his widely used in the Japanese tele text system or FM multiplex broadcasting systems [12]–[14].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 -bits, it takes cycles in the decoding process, posing a big impact on system performance [6].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 [11] 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 [15] 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.

This paper 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. 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

Figure. 1. Memory system schematic with MLD.

(MLDD) using difference-set cyclic codes; Section IV discusses the results obtained for the different versions in respect to effectiveness, performance, and area and power consumption. Finally, Section V discusses conclusions and gives an outlook onto future work.

2. EXISTENT MAJORITY LOGIC DECODING (MLD) SOLUTIONS

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.

MLD was first mentioned in [7] for the Reed–Müller codes. Then, it was extended and generalized in [8] for all types of systematic linear block codes that can be totally orthogonalized on each code word bit .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 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. There are two ways for implementing this type of decoder. The first one is called the Type-IML decoder, which determines, upon XOR combinations of the syndrome, which bits need to be corrected [6]. The second one is the Type-II ML decoder that calculates directly out of the codeword bits the information of correctness of the current bit under decoding [6]. 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. Therefore, this paper focuses only on this one.

A. 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 results of the checksum 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).As stated before, input 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 code word is loaded, as illustrated in Fig. 2.The input signal is initially stored into the cyclic shift register rand shifted through all the taps. The intermediate values in each tap are then used to calculate the into the cyclic shift register, the decoding starts by calculating the parity check equations hardwired in the XOR matrix. The resulting sums are then forwarded to the majority gate for evaluating its correctness. If the number of 1's received in 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 codeword bits have been processed. Finally, the parity check sums should be zero if the code word has been correctly decoded. Further details on how this algorithm works can be found in [6]. The whole algorithm is depicted in Fig. 3. The previous algorithm needs as many cycles as the number of bits in the input signal, which is also the number of taps, in the decoder. This is a big impact on the performance of the system, depending on the size of the code. For example, for a code word of 73 bits, the decoding would take 73 cycles, which would be excessive for most applications.

B. 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 [11]. 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 (see Fig. 4).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. A faulty code word 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 In this way, the performance is improved in exchange of an additional module in the memory system: a matrix of XOR gates to resolve the parity check matrix, where each check bit results into a syndrome equation. This finally results in a quite complex module, with a large amount of additional hardware and power consumption in the system.

3.PROPOSED ML DETECTOR/DECODER

This section presents a modified version of the ML decoder that improves the designs presented before. Starting from the original design of the ML decoder introduced in [8], the proposed ML detector/decoder (MLDD) has been implemented using the difference-set cyclic codes (DSCCs) [16]–[19]. 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

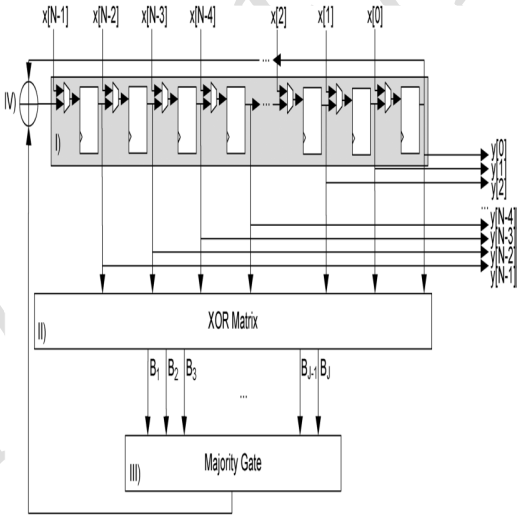

Figure. 2.Schematic of an ML decoder.I) cyclic shift register. II) XOR matrix. III) Majority gate. IV) XOR for correction.

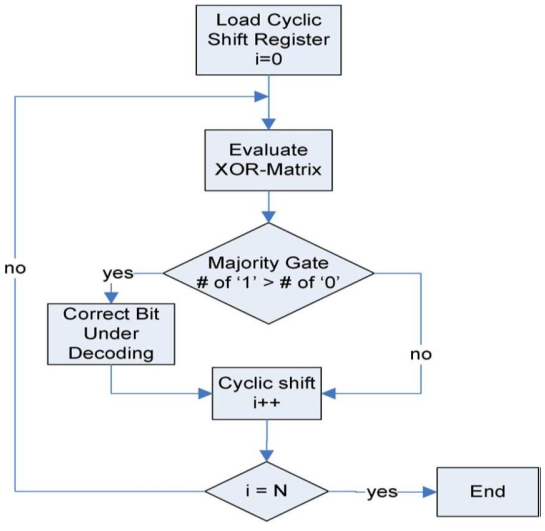

Figure. 3.Flowchart of the ML algorithm.

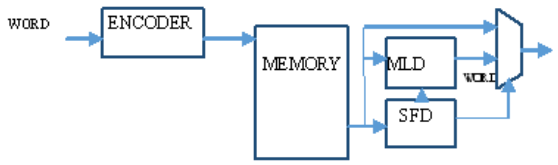

Figure. 4.Memory system schematic of an ML decoder with SFD.

• 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.

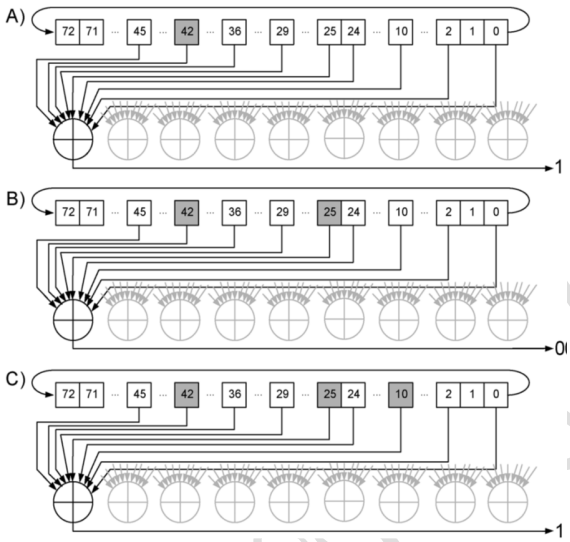

Figure. 5.Single check equation of a □ _ __ ML decoder. (a) One bit-flip. (b) Two bit-flips. (c) Three bit-flips.

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 (see [6] for more details).However, when multiple errors accumulate in a single word, this mechanism may misbehave, as explained in the following. In the simplest error situation, when there is a bit-flip in a code word, the corresponding parity check sum will be "1," as shown in Fig. 5(a). This figure shows a bit-flip affecting bit 42 of a codeword with length and the related check sum that produces a "1." However, in the case of Fig. 5(b), the code word is affected by two bit-flips in bit 42 and bit 25, which participate in the same parity check equation. So, the check sum is zero as the parity does not change. Finally, in Fig. 5(c), there are three bit-flips which again are detected by the check sum (with a "1").As a conclusion of these examples, any number of odd bit flip scan be directly detected, producing a "1" in the corresponding .The problem is in those cases with an even numbers of bit-flips, where the parity check equation would not detect the error. In this situation, 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 .Since performance is important for most applications, we have chosen an intermediate solution, which provides a good reliability with a small delay penalty for scenarios where up to five bit-flips may be expected (the impact of situations with more than five bit-flips will be analyzed in Section IV-A). This proposal is one of the main contributions of this paper, and it 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 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 have been 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).
• Through a simplified mathematical proof for the particular case of two bit-flips affecting a single word
For simplicity, and since it is convenient to first describe the chosen design, let us assume that the hypothesis is true and that only three cycles are needed to detect all errors affecting up to five bits (this will be confirmed in Section IV).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 cycles, the proposed method stops intermediately in the third cycle, as illustrated in Fig. 6. If in the first three cycles of the decoding process, the evaluation of the XOR matrix for all is "0," the code word 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. 7.

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. Those components are the same as the ones for the plain ML decoder shown in Fig. 2. The additional hardware to perform the error detection is illustrated in Fig. 7 as i) the control unit which triggers a finish flag when no errors are detected after the third cycle and ii) the output.

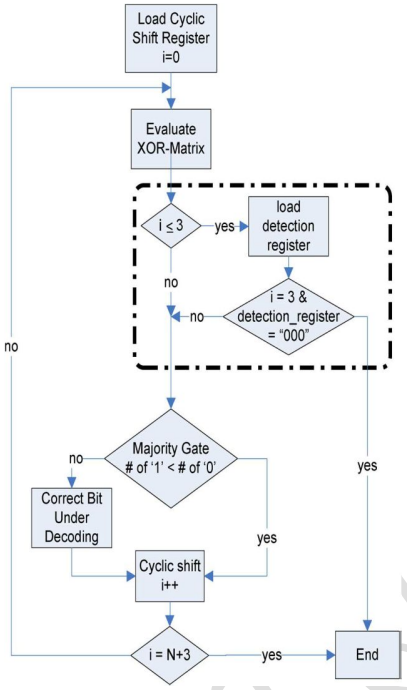

Figure. 6.Flow diagram of the MLDD algorithm.

Tristate 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. The control schematic is illustrated in Fig. 8. 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.

This clearly provides a performance improvement respect to the traditional method. Most of the words would only take three cycles (five, if we consider the other two for input/output) and only those with errors (which should be a minority) would need to perform the whole decoding process. More information about performance details will be provided in the next sections. The schematic for this memory system is very similar to the one in Fig. 1, adding the control logic in the MLDD module.

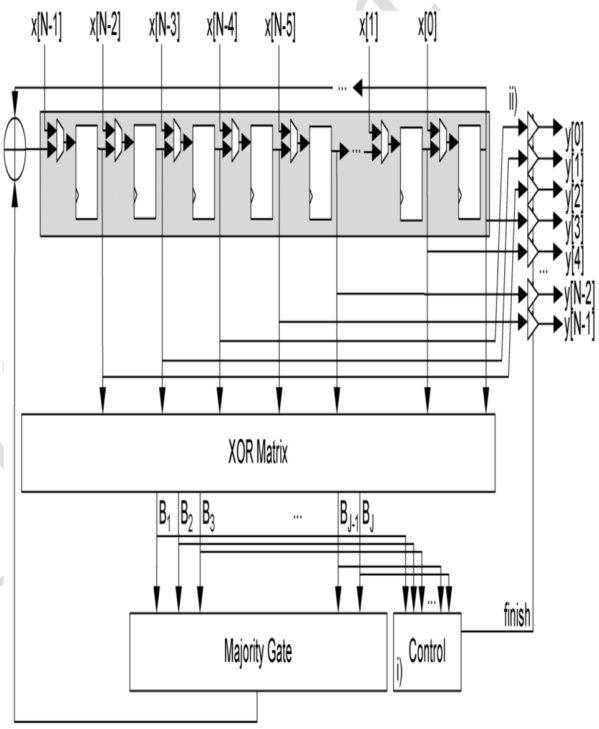

Figure. 7.Schematic of the proposed MLDD.i) Control unit. ii) Output tristate buffers.

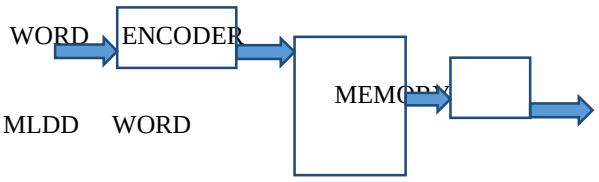

Figure. 8.Memory system schematic of an MLDD.

4.CORRECTOR

One-step majority logic correction is the procedure that identifies the correct value of an each bit in the codeword directly from the received codeword the majority value indicates the correctness of the

code-bit under consideration; if the majority value is 1, the bit is inverted, otherwise it is kept unchanged.

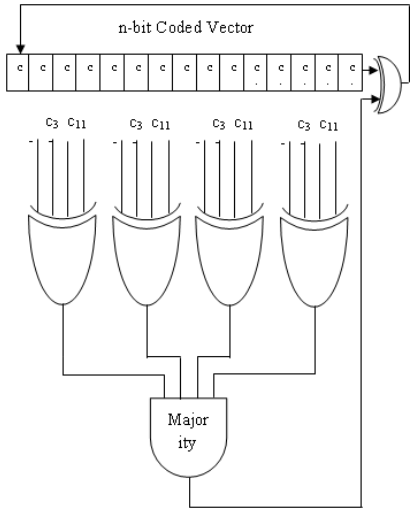

Figure 9 Majority Logic Corrector for 15-Bit Codeword's

A compact implementation for the majority gate is by using Sorting Networks. The binary Sorting Networks is used to do the sort operation of the second step efficiently. An -input sorting network is the structure that sorts a set of bits, using 2-bit sorter building blocks. Fig. 6.5 (a) Shows a 4-input sorting network. Each of the vertical lines represents one comparator which compares two bits and assigns the larger one to the top output and the smaller one to the bottom see Fig. 6.5 (b) the four-input sorting network, has five comparator blocks, where each block consists of two two-input gates; overall the four-input sorting network consists of ten two-input gates in total.

5. SORTING NETWORK

Sorting network is used to sort the two or more inputs. By using the sorting network accessing time is reduced by sorting the inputs. From the below diagram each vertical line indicates a comparator, which compares the two bits and assigns the larger one to the top output and smaller one to the bottom. From that we conclude that without using the sorting network the XOR matrix output is directly applied to the majority gate so the accessing time is large to obtain the output. In our proposed we use sorting network in the modified control unit it separate the maximum level output and minimum level output So the accessing time is reduced.

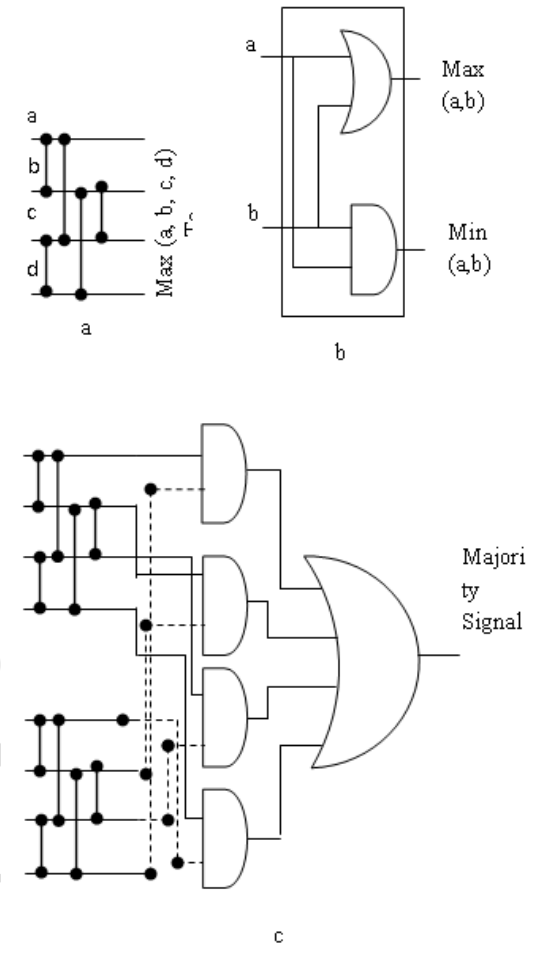

Figure 10 (a) Four-Input Sorting Network Each (b) One Comparator Structure
(c) Eight-Input Majority Gate Using Sorting Network

6.RESULTS

A. Memory

The memory read access delay of the plain MLD is directly dependent on the code size. i.e., a code with length 72 needs 72 cycles. Then two extra cycles need to be added for I/O. On the other hand, the memory read access delay of the proposed Modified MLDD is only dependent on the word error rate(WER).there are more errors, then more words need to be fully decoded.

Figure 11 Error Detection by Plain MLD Method

B. Area

In the proposed MLDD there is an extra circuitry of control logic which consists of shift register and or gates. The given below table shows the comparison of total estimated power consumption.

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 modified MLDD using sorting network.

7. CONCLUSION

In this paper, the detection of errors during first iterations of serial one step Majority Logic Decoding of DSCCs-LDPC codes has been presented. The simulation results show that the one step MLD would takes 15 cycles to decode a code word of 15-bits, which would be excessive for most applications. The MLD design requires small area but requires large decoding time and can be able to detect two or few errors.Hence,memory access time increases another method, called MLDD can detect upto five bit-flips and consumes the area of majority gate.

The proposed modified MLDD have the capability to detect the presence of errors in just 3 cycles even for multiple bit flips.It has found that for error detection and correction (for code word of 15), when comparing to the existing technique, a speed up of about 1100 ns is obtained when there is no errors in the data read access. 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 the capability of detecting more than five bit flips and also reduces the area of majority gate by the use of sorting network.

REFERENCES

[1] 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 Transactions On Very Large Scale Integration (VLSI) Systems, vol. 20, no. 1, January 2012

[2] 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.

[3] 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.

[4] 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.

[5] G. C. Cardarilli et al. Concurrent error detection in reed-solomon encoders and decoders. IEEE Trans. VLSI, 15:842–826, 2007.

[6] R. Horan et al. Idempotents, mattson-solomon polynomials and binary ldpc codes. IEE Proceedings of Communication, 153(2):256–262, 2006.

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

[8] 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.

[9] 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.

[10] Heng Tang et al. Codes on finite geometries. IEEE Transaction on Information Theory, 51(2):572–596, 2005.

[11] Shu Lin and Daniel J. Costello. Error Control Coding. Prentice Hall, second edition, 2004.

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

[13] J. Kim et al. Error rate in current-controled logic processors with shot noise. Fluctuation and Noise Letters, 4(1):83–86, 2004.

[14] S. Hareland et al. Impact of CMOS process scaling and SOI on the soft error rates of logic processes. In Procedings of Symposium on VLSI Digest of Technology Papers, pages 73–74, 2001.