International Journal of Advanced Information Science and Technology (IJAIST) *ISSN: 2319:268*
Vol.2, No.4, April 2013 *DOI:10.15693/ijaist/2013.v2i4.182-189*

High Performance Error Detection and Correction for Memory Applications using Majority Logic Decoder and Detector

[1] Dhanasekaran S.G
PG Scholar/Department of ECE
Karpagam University, Coimbatore, India

[2] Mahendra Babu G.R
Assistant Professor/Department of ECE
Karpagam University, Coimbatore, India

***Abstract*—The Error-Detection and Correction method for Difference-Set Cyclic Codes with Majority Logic Decoder and Detector is to be present in this project. Majority logic decodable codes are suitable for memory applications due to their capability to correct a large number of errors. They require a large decoding time that impacts memory performance. The fault-detection method significantly reduces memory access time when there is no error in the read data. This technique using the majority logic decoder itself to detect errors, it makes the area overhead minimum and keeps the extra power consumption low.**

Index terms - Low-density Parity Check (LDPC), Error Correction Codes (ECCs), Difference Set-Cyclic Codes, Block codes, Majority logic, Memory.

I. INTRODUCTION

The impact of technology scaling small in dimensions, high combination densities, and low operating voltages which has come to a level that reliable of memories is put to difficulty, not only in excessive radiation environment like spacecraft and avionics electronics, but also at normal terrestrial environments. Especially, SRAM memory failure rates are increasing drastically, so posing a foremost reliable 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 consisting of three versions of the designs in corresponding, with a majority voter select the exact output. As the technique suggest the complexity is overhead would be three times plus the complexity of the majority voter and thus increasing the power consumption [1].

For memories, it turned out that ECC codes are best way to mitigate memory soft errors. For terrestrial radiation environments where there is a lowsoft error rate (SER), codes like single error correction and double error detection (SEC–DED) are a good result, due to their small encoding and decoding complexity. However, as a consequence of augmenting integration densities, there is an increase in the number of soft errors and which produce the require for high error correction capabilities. The usual multierror correction codes, such as codes are Reed–Solomon (RS) or Bose–Chaudhuri– Hocquenghem (BCH) are not suitable in favor of this task. The reason is that they utilize more sophisticated decoding algorithms, like complex algebraic (e.g., floating point operations or logarithms) decoders that can decode in unchanging time, and simple for graph decoders, to use for iterative algorithms. Among the ECC codes that meet the requirements of higher error correction capability and low decode complexity, the cyclic block codes have been recognized as good candidate, suitable to their property of majority logic (ML) decodable [2]. A subgroup of the low-density parity checks (LDPC) codes, which belongs to the family of the ML decodable codes. 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 broadcasting 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 difficulty [1]. The disadvantage of ML decoding is for a coded word of -bits, it takes cycles in the decoding progression, posing a large impact on system output. One way of coping with this problem is to implement parallel encoders and the decoders. This result would especially increases the complexity and the power consumption. The most of the memory read access will without errors, and the decoder is majority of the time functioning without reason. This will have motivate the uses of a error detector module that checks if the codeword contains an error and then triggers the correction mechanism accordingly. In this case, only the faulty code word needs correction, and as a result the average reading memory accessing is speed up, by the expense of increased in hardware cost and the power utilization.

A related proposal have been obtainable in for the case of flash memories. The simplest way to implement a fault detector for an ECC is by calculates syndrome, however this in general implying adds another very complex functional unit [2]. 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 expenses. The results show that the properties of DSCC-LDPC enable efficient fault detection.

II. EXISTENT MAJORITY LOGIC DECODING (MLD) SOLUTION

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 checks

equation, apart from the very first bit which contribute to every equations. For this motive, the majority results of these parity check equations decide the correctness of the current bit under decoding [1] .

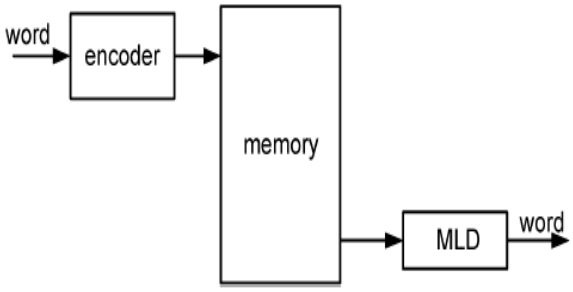

Fig. 1. Block Diagram of the MLD System

MLD was first mentioned for the Reed–Muller codes. Then, it was extended for all types of systematic linear block codes that can be totally orthogonalized on each codeword bit. A generic schematic of a memory system is depicted into Fig.1, as using of an ML decoder. primarily, the data words are encoded and then stored within the memory. After the memory is reads, the code words are fed through the ML decoder before sent to the output for additional processing. During this decoding processing the data words are corrected from all bit-flips that it might have suffered while being stored in the memory.

There are two ways for implementing these type of decoder. The first one is mentioned as the Type-I ML decoder, it determines upon XOR combinations of the syndrome, which bits need to be corrected. 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. Both are quite similar but when it comes to execution Type-II use lesser area, while it is not calculate the syndrome as an intermediate step. Therefore, this paper focuses only on this one.

A. 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 checking equations [1]. It is consisting of four parts: 1) cyclic shift register; 2) XOR matrix; 3) majority gate; and 4) XOR for correcting the codeword bit under decoding. The input signal is initially stored into the cyclic shift register and shifted through the entire taps. The transitional values in each taps 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. As stated before, input might correspond to wrong data corrupted by a soft errors. To handling in this situations the decoder will act as follows. Then after the initial step, where the code word is loaded into a cyclic shift register and the decoding started through calculates the parity check equations hard wired in the XOR matrix.

The resulting sums are then forwarded to the majority gate for evaluating its correctness [9]. If the number of 1's received in is greater than the number of 0's, which would mean that the current bit under decode value is wrong and the signal to correct which would have been triggered. If not, the bits under decoding will be correct and no extra operations will needs on it. Then the next step the contents of the register is rotated and the above procedure is repeated until all codeword bits have processed. As a final point, the parity checks sum must be zero if the codeword has been correctly decoded. The whole algorithm is depicted in Fig.2. 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 system, and depending on the size of the codeword. The example of a code words of 15 bits, then the decoding would takes 15 cycles, as would be extreme for most applications.

B. Plain MLD with Syndrome Fault Detector (SFD)

In order to improve the decoder performance, alternative designs may be used [1]. One possibility is to add a fault detector by calculating syndrome, therefore as only faults code word is decoded. Since most of the code words will be error-free, M no further correction will be required, and thus the result 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.

The SFD is an XOR matrix that calculates the syndrome based on the parity checking matrix. Every parity bits result in to syndrome equation. Thus, the complexities of the syndrome calculator increases with the size of the codes. The error code words are detected while at least one of the syndrome bits is "1." This triggers the MLD to starts the decoding process as explained before. On the otherwise, if the code words are error-free, then it was forward directly into 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 checking matrix and the each check bits result into the syndrome equations. This is finally result into a quite complexity module with a huge amount of extra hardware and the power consumption in the system.

C. ML Decoder/Detector

This section presents a ML decoder that improves the designs existing earlier. Initially from the original design from the ML decoder, the proposed ML detector/decoder (MLDD) has been implemented using the difference-set cyclic codes (DSCCs).The Fig.2 shows the memory system schematics of an MLDD [1].

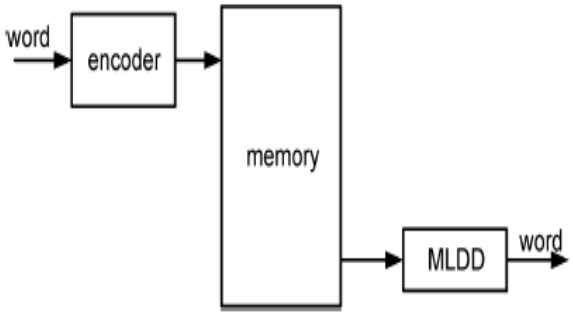

Fig. 2. Memory system Schematics of an MLDD.

This code is part of LDPC codes based on its attributes, and they contain the following properties:

- To correct the large numbers of error;
- Sparse encoding, decoding and checking circuits synthesizable into simple hardware;
- Modular encoding and decoding blocks that allows an efficient hardware implementation;
- Logical code structures for clean partitioning 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 [9]. The problem is in those cases with an even numbers of the bit-flips, wherever the parity checking equations will 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". But, the alternative should be derive all data to the decoding process, with a large performance overhead. Since performance is important for most of applications and we would have selecting the intermediate solutions, which would provided a high reliability with a small delay penalty for scenarios where up to five bit-flips may be expected. The ML Decoding Process is control by the control unit shown in Fig.3.

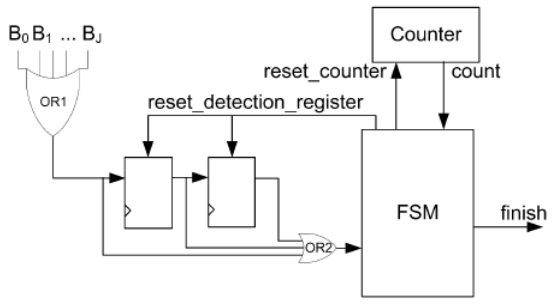

Fig. 3. Schematics of control unit

This proposal is one of the main contributions of this article, and it will be based on the following assumption: Given a word read from a memory protected with the DSCC codes, and they affected by capable of five bit-flips, the entire errors can be detects only in three decoding cycles. This is a huge improvement over the simpler case,

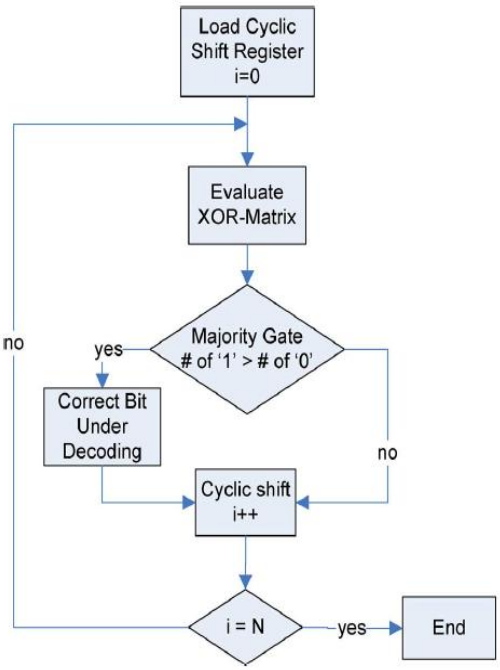

Fig. 4. Flow Diagram for ML Algorithm

where decoding cycles are needed to guarantee that errors are detected. The flow diagram of ML algorithm is shown in Fig.4. The proof of this hypothesis is very complex from the mathematical approach. As a result two alternatives have used inorder to proves it, which are specified here.

- During simulation in which comprehensive experiments have been conducted, to effectively verify that the hypothesis applies.
- A simplified mathematical proofs of the particular case of two bit-flips affecting a single word.

For simplicity, and since it is convenient to first describe the preferred designs, lets us assumes that the assumption is true and that only three cycles are needed to detect all errors affecting up to five bits. In general, the decoding algorithm is still the same as the one in the plain ML decoding version. The variation is that instead of the decoding all codeword bits by processing the ML decoding during cycles, the proposed method stops intermediate. 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 they 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 basic ML decoder with an tapped shift registers is an XOR array to be calculates the orthogonal parity check sums and a majority gate for deciding if the current bit under decoding needs to be inverted. i) the control unit which triggers a finish flag when no errors are detected after the third cycle and ii) the output tristate buffer. The output of tristate buffer is 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 detecting the process. They used a counters that would be counting upto three bit-flips, which compares 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' functions. Its values are fed into the three stage shift registers, it holds the outcome of the final three cycles. During the third cycle the OR'2' gate evaluate the content of the detection in registers. once 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 Point. This is clearly provides a performance of improvement respected to the conventional method. It is very easy to implement the design using VHDL code for MLDD improvement.

III. Proposed Parallel Majority Logic Decoder and Detector (MLDD) Solutions

A. Parallel MLDD

The decoding process of the MLD can also be implemented in a parallel manner. The Fig. 5. shows the block diagram of parallel MLDD using sorting networks. Like this, the decoding is speeded-up showing a similar speed to the serial ML decoder. Its schematic diagram implementation is illustrated in Fig.6. Herein, all codeword bits are decoded at the same time by having for each codeword bit a parity check equation, a MG and a correction unit. Here encoder is encoding the output to the shift registers, and the shift register shifted the each parity bits, then shifted values send to the

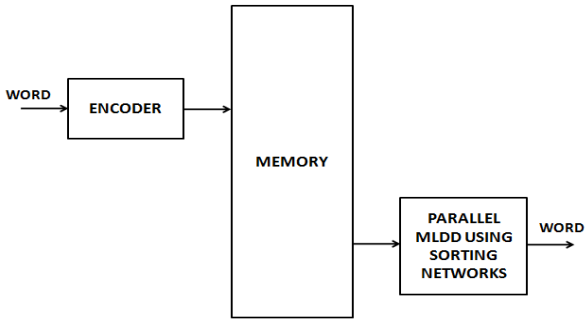

Fig. 5. Block Diagram of Parallel MLDD using Sorting Networks

Parity check equations using XOR function for 15 parity checks, and it gives to the majority gate for reducing the gates using sorting network, then if any error is detected goes to XOR operations, so error is now corrected using Parity check sums.

B. Majority Gate (MG)

The MG is responsible for evaluating the parity checksums and deciding upon the correctness of the current BUD. Traditionally, the implementation complexity of the MG could grow exponentially with the number of inputs added. To tackle this complexity issue, presented a MG based on sorting networks.

A modified version to handle eight parity checksums of MG based on sorting networks has been implemented as depicted in Fig. 7. These Majority Gate is used to number of parity check sums and it is gives the output to the shift registers.

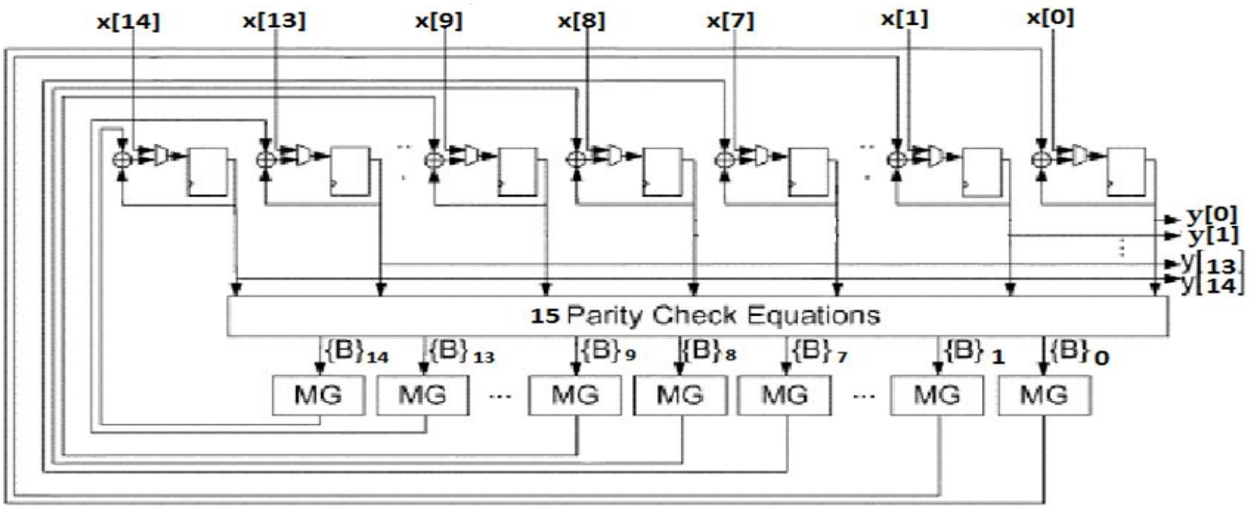

Fig. 6. Schematic Diagram of Parallel MLDD Circuit

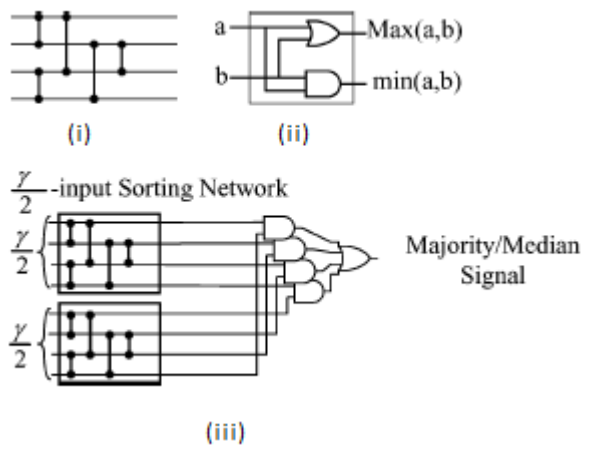

Fig.7. (i) Basic Comparator Structure, (ii) Comparator using Sorting Network, (iii) 8 bit Majority Gate using Sorting Network.

C. Synthesis Results

This subsection is divided into two parts. The first one discusses and compares the resource occupation and the implementation efficiency of all three decoder structures. The second part gives an evaluation of the performance matrics obtained from the timing synthesis.

D. Resource usage

The synthesis report of the resource occupation of the three decoders has been documented in next Chapter. The table also includes the relative values comparing the MLDD and Parallel MLDD next to the absolute values of the resource usage. At first it has to be mentioned that the MLD decoders had to be modified for fault injection with FLIPPER resulting in an overhead of Flip-Flops (FF) / Latches. This overhead will have little impact onto the overall slice usage. The reason for this is because both MLDs have sufficient combinational logic complexity. The ASIC results for both OS-MLDs have been retrieved without the extra FFs required for the FLIPPER platform. The remaining overhead is explained by the employed DSCC which is an original 15 word length is shortened.

However, internally it calculates with 15 bits and besides, the serial MLD requires FFs for its finite state machine (FSM). This is also the reason why the parallel version requires fewer FFs than the serial one. When comparing again the combinational logic of the Serial MLD to the parallel MLD the Speedup time is increased, the delay time is decreased. So the implementation of the combinational logic in Parallel MLD is much more efficient.. The locator needs to check if the syndrome matches a value that corresponds to a single error.

E. Timing performance

The maximum clock frequency reported from the timing synthesis for the three implemented decoders are summarized in next section and including also the throughput of each design. The serial MLD requires 15 cycles for decoding one codeword and hence its throughput is proportionally low. On the other hand, the parallel MLD have the same throughput of one third. The timing synthesis report reveals that the serial MLD have high maximum clock frequency results.

Based on the parallel implementation of combinational logic of the MLD, the parallel version has a maximum clock frequency which is more than four times higher compared to the serial version. For the cases of high-throughput requirements or where the resource occupation is a soft design constraint, the parallel MLD is a good option since it is has a throughput of 14 times better than the serial version and can be implemented with much higher clock frequency.

IV. RESULTS AND DISCUSSION

In this project we have developed a software to detect the Majority Logic Fault error in Memory applications. An important final comment is that the area overhead of the MLDD actually decreases with with respect to the plain MLD version. For large values of both areas are practically the same value. The reason of this is that the error detector in the MLDD has been designed to be independent of the size code. The opposite situation occuring with these SFD technique that uses syndrome calculation to perform error detection: its complexity grows quickly when the code size increases. One of the problems to make the MLDD module independent of has been the mapping of the intermediate delay line values to the output signals.

The reason is that this module behaves in two different ways depending if the processed word is error or correct. Then if it is correct its outputs are driven after third cycle and that the word has been shifting three positions in the line registers. Then if it is not correct value the word has to be fully decoded, what implies being shifted positions. Then some kind of multiplexing logic would be needed to reorder the bits before mapping them to the output.

However, the area of this logic would grow with linearly. After this, the output bits are coherent in all situations, not needing multiplexing logic. We have developed a vhdl codlings using CADENCE software(ncsim:09.20-p007) and Xilinx software. Comparing the failure rates for the decoders, the serial OS-MLD would seem to be the one showing the best soft error resilience and the parallel MLD the worst one. However, these failure rates do not include the resource usage of these designs. However, these failure rates do not include the resource usage of these designs.

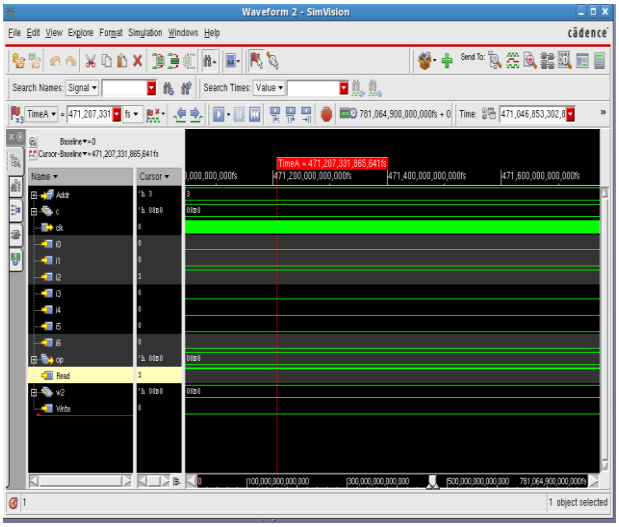

Fig. 8. Simulation Output MLDD without Error

The Fig. 8 shows the simulated output waveform of MLDD without error.

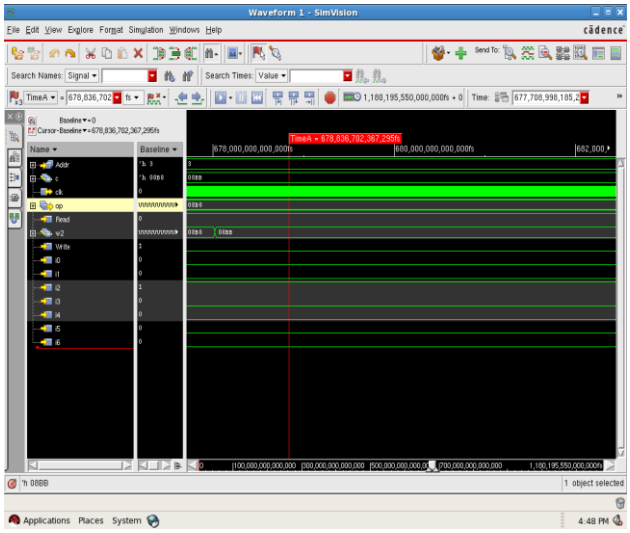

Fig. 9. Simulation Output MLDD with Error orrection

The difference sets of code lengths are calculated as the equation (1), Here given a size N, all combinations of four bit-flips on a word will be calculating, in order to study all of the possible case. Therefore the number of combinations is given by,

$$\binom{N}{M} = \frac{N!}{M!(N-M)!} \qquad (1)$$

where M is the number of bit-flips shown in Table I.

TABLE I

DSCC LENGTHS

N Bits	Data bits	Parity Bits
15	7	8

The number of combinations can be seen in Table II for different values of N with double and quadruple errors. Here we used in seven data bits and the eight parity bits.

The number of combinations are listed in table II. As expected, increasing the code length implies an exponential growth of the number of combinations of the computational time. All combinations in Table II have been simulates, then the results can be seen in Tables III. Here the exising method and the Proposed method is compared.

TABLE II

NUMBER OF COMBINATIONS FOR BIT-FLIPS WITH DIFFERENT CODE LENGTHS

PARAMETERS	Existing Method	Proposed Method
M	4	4
N	15	15
No. of Combinations	4896	13824

Table III shows the results for the case of four bit-flips. These results confirm that with only one decoding cycles after the detection method is covering more than 90% of the error patterns for all. The second cycle of MLDD increases the percentage of detection and after the third one, 100% of the errors are detected. The performance of the proposed design MLDD is much faster than the plain MLD version shown in Table III.

TABLE III

SPEED-UP OF THE PROPOSED MLDD FOR ERROR FREE CODEWORDS

PARAMETERS	Existing Method	Proposed Method
N	15	15
ML Detection	5	5
Speed (ns)	7.189	2.872

To study this, the two designs have been implemented in VHDL and synthesized, for different values of N, using the Cadence and the Modelsim library. The obtained results are depicted, in number of equivalent gates, listed in Table IV.

TABLE IV
SYNTHESIS RESULT OF THE DIFFERENT CODE LENGTHS

PARAMETERS	Existing Method	Proposed Method
N	15	15
MLDD	47	47
Area Overhead (%)	27	12.5

In a real situation, a fraction of the words would having bit-flips. These fraction is representing from by the WER. Since MLDD needs five cycles to handle correct words and N+5 for erroneous words, the average performance would be ,

MLDD Performance = (1-WER).5+WER.(N=5) (2)

Using these expression and the performance of the three techniques has been studied for different values of the WER and it is used to calculate the area overhead for different code lengths.

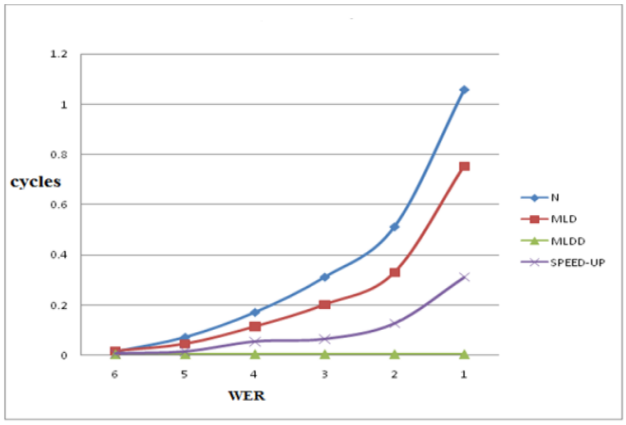

Fig. 10. Speed-Up of Proposed MLDD

This differences are even smaller for larger values of an WER. The WER compares Speed-Up for Existing and Proposed system shown in Fig. 10. It us used for the best performance is given.

V. CONCLUSION

In this paper, a error detection mechanism, they Parallel MLDD has been presented based on the ML decoding used to the DSCCs. The exhaustive simulation test, the result shows that the proposed technique is able to detect any pattern of up to five bit-flips in the first three cycles of the decoded process. This will improving the performances of the design with respect to the traditional MLD approach. Otherwise, the Parallel MLDD error detector module has been designed in a way that is independent of the codeword size. This would makes its area overhead is quite reduced compared with other traditional approaches such as the syndrome calculation (SFD). In addition, a theoretical proof of the proposed MLDD scheme for the case of double errors has to be presented. The expansion of the proofs to the case of four errors would confirm the validity of the MLDD approach for a more general cases and something that only has been done throughout simulation in this paper. This is an interesting problem for future researches. The applications of the proposed techniques to memories that use scrubbing is also an interesting topic and was in fact the original motivation that led to the Parallel MLDD scheme.

ACKNOWLEDGEMENT

The authors would like to thank some of their anonymous friends for their helpful suggestions in the development of these ideas.

REFERENCES

[1]. S. Liu, J. A. Maestro, and P. Reviriego, (2012) "Efficient Majority Logic Fault Detection With Difference-Set Codes for Memory Applications", IEEE Trans. Very Large Scale Integration Systems, Vol. 20, No.1.

[2]. Pedro Reviriego, Mark F. Flanagan, Shih-Fu Liu, and Juan Antonio Maestro, *Member,(2012),"*Error-Detection Enhanced Decoding of Difference Set Codes for Memory Applications", IEEE Trans. Device Mater. Reliability, Vol. 12, No. 2, pp. 397–404.

[3]. P. Ankolekar, J. Bredow, R. Isaac, and S. Rosner, (2010) "Multi-bit Error Correction Methods for Latency-contrained flash memory systems", IEEE Trans. Device Mater. Reliability, Vol. 10, No. 1, pp.33–39.

[4]. M. A. Bajura et al, (2007) "Models and Algorithmic Limits for an ECC Based Approach to Hardening Sub-100-nm SRAMs", IEEE Trans. Nucl. Science, Vol. 54, No. 4.

[5]. M. Basoglu, M. Erez, L. Kaplan, I. Lee, M. Sullivan, D. H. Yoon, and (2011), "Survey of Error and Fault Detection Mechanisms", The University of Texas at Austin 1 University Station (C0803) Austin, Texas 78712-0240,TR-LPH-2011-002.

[6]. R. C. Baumann and Fellow, (2005) "Radiation-Induced Soft Errors in Advanced Semiconductor Technologies", IEEE Trans. Device Mater. Reliability, Vol. 5, No. 3.

[7]. S. K. Chilappagari, and B. Vasic, (2007) "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.

[8]. J. Draper, and R. Naseer, (2008) "DEC ECC Design to Improve Memory Reliability in Sub-100 nm Technologies", in Proc. IEEE ICECS, pp.586-589.
[9]. A. DeHon and H. Naeimi, (2009) "Fault Secure Encoder and Decoder for NanoMemory Applications", IEEE Trans. Very Large Scale Integration Systems, Vol. 17, No. 4.
[10]. S. Ghosh and P. D. Lincoln, (2007) "Low Density Parity Check Code for Error Correction in Nanoscale Memory", SRI Comput. Sci. Lab. Tech. Rep. CSL-0703.
[11]. J. von. Neumann, (1956) "Probabilistic Logics and Synthesis of Reliable Organisms from Unreliable Components", Automata Studies, pp. 43-98.
[12]. C.W.Slayman, (2005) "Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations", IEEE Trans. Device Mater. Reliability, Vol. 5, No. 3, pp. 397–404.

AUTHORS PROFILE

Dhanasekaran S.G received the B.E. Electrical and Electronics Engineering in Sri Muthukumaran Institute of Technology, Chennai, Affiliated to Anna University Chennai and M.E. VLSI Design Final year pursuing in Electronics and Communication Engineering from Karpagam University, Coimbatore, India in 2010 and 2013, respectively. He is currently researching in the field of Digital Design and Low Power CMOS VLSI Design.

Mahendra Babu. G.R received the B.E. degree in Electronics and Communication Engineering from Trichy Engineering College, Trichy and M.E. degree in Embedded System Technology from Anna University of Technology, Coimbatore in 2005 and 2010 respectively. He is currently an Assistant Professor in the Department of Electronics and Communication Engineering, Karpagam University, Coimbatore, India. He has published many papers in national and international conferences. His current research interest includes engineering topics in VLSI and Embedded Systems

.