IOSR Journal of VLSI and Signal Processing (IOSR-JVSP)
Volume 7, Issue 3, Ver. I (May. - Jun. 2017), PP 19-26
e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197
www.iosrjournals.org

Design of Majority Logic Decoder for Error Detection and Correction in Memories

B.Swapna[1], K.Jamal[2]
[1](ECE, GRIET/ JNTUH, INDIA)
[2](ECE, GRIET/ JNTUH, INDIA)

Abstract: *Due to augmenting integration densities, technology scaling and variation in parameters, the performance failures would possibly occur for every application. The memory applications are vulnerable to single event upsets and transient errors which may cause malfunctions. This paper deals with the idea of a totally distinctive fault detection and correction technique using EG-LDPC codes with the applying mainly targeted on reminiscences. The majority logic secret writing is used here, since it will correct associate degree outsize type of errors. Albeit the majority secret writing consumes longer, it will be overcome by the projected technique that detects the errors in less cycle time. It will clearly reduce operation time once the information scan technique is error free. the employment of associate degree additional logic finishes up during a little house overhead in projected methodology once place next to the current technique, that's overcome by a revised implementation of majority gate.*

Key words: *Majority logic decoding; error correction codes (ECCs); Euclidean geometry low-density parity check (EG-LDPC); memory.*

I. Introduction

Memories are the most general component today. They are prone to errors like soft and transient errors. Some type of embedded memory, such as ROM, DRAM, SRAM, flash memory etc. is seen in almost all system chips. Now a days, the memory failure rates are increasing due to the influence of technology scaling-smaller dimensions, lower operating voltages, highintegration densities etc.[4],[5]. As the dimensions and operating voltages of computer electronics are minimized to satisfy the consumer's unquenchable demand for higher density, functionality, and lower power, their sensitivity to radiation increases noticeably. There are a number of radiation effects in semiconductor devices that vary in magnitude from data interruption to permanent damage ranging from parametric shifts to complete device failure.

Some frequently used error detection techniques are Error Correction Codes (ECCs) and Triple Modular Redundancy (TMR). The TMR triplex all the memory parts of the system and to choose the correct data using a voter. This method have disadvantage of complexity overhead of three times and large area. A soft error occurs when a radiation event causes enough of a charge disturbance to flip or reverse the data state of a memory cell, register, latch, or flip-flop. The error is "soft" because the device/circuit itself is not permanently damaged by the radiation if new data are written to the bits; the device will store it correctly. The soft error is also often referred to as a single event upset (SEU).

The TMR threefold 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 [4]. 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 device/circuit itself is not permanently damaged by the radiation. If new data are written to the bit, the device wills storeit correctly. The soft error is also often referred to as a single event upset (SEU) [4].

The most frequently used ECC codes are Single Error Correction (SEC) codes that can correct one bit error in a memory word. Due to consequence of higher integration densities, there is an increase in soft errors which points the need for more error correction capabilities [1],[3]. Therefore, it has become conventional to safeguard memories with the application of error correcting codes (ECC) like single-error-correcting (SEC) Hamming code, single-error-correcting double-error-detecting (SEC-DED) extended-Hamming, or SEC-DED Hsiao codes With multi-bit upsets (MBU) becoming a major contributor to soft errors. Conventional SEC or SEC-DED codes may not be sufficient to meet reliability goals. To mitigate these effects, the use of more powerful ECC and/or memory scrubbing with regular ECC are being suggested.

The general multi error correction codes, such as Reed–Solomon (RS) or Bose Chaudhuri–Hocquenghem (BCH) are not suitable for this task in view of complex decoding algorithm. Cyclic block codes have the property of being majority logic (ML) decodable. Therefore cyclic block codes have been described as more suitable among the ECC codes that meet the requirements of greater error correction capability and low

decoding complexity. Euclidean geometry low-density parity check (EG-LDPC) codes, a subcategory of the low-density parity check (LDPC) codes, which belongs to the family of the ML decodable codes, is focused here. The benefits of ML decoding are that it is very simple to construct and easy to implement thus it is very practical and has low complexity.

The objective of the method is to use the first three iterations of majority logic decoding to detect if the word being decoded contains errors. If there are no errors, without completing the remaining iterations, immediately decoding can bestopped therefore greatly reducing the decoding time. For a code with block length n, majority logic decoding using serial implementation requires equal number of iterations, means decoding time is directly proportional to code size, so as the code size grows, so does the decoding time.

In the proposed method we are using initial three iterations to detect errors thereby achieving a large speed increase for codeword if it is error free. For DS-LDPC codes, all error combinations of up to five errors can be detected only in the first three iterations. Additionally, errors affecting more than five bits were detected with a probability close to one. The probability of undetected errors was also found to decrease as the code word length increased. For a million error patterns only a few errors or sometimes none were undetected. This may be acceptable for most of the applications. Another advantage of this method is that it requires a little additional circuitry as the decoding circuitry is also used for error detection.

II. Majority Logic Decoding (Mld) Solutions

Majority logic decoder is mainly based on number of parity check equations which are orthogonal to each other so that for each iteration, each codeword bit participates in only one parity check equation, except the very first bit which contributes to all equations. With this reason, the majority result of these parity check equations decide the correctness of the current bit under decoding [6]. MLD was first mentioned in for the Reed–Müller codes. Then, it was extended and derived for all types of systematic linear block codes that can be totally orthogonalized on each codeword bit.

A generic schematic of a memory for the usage of an ML decoder is shown below. Initially, the data words are encoded and then stored in the memory [2]. When the memory is read, the codeword is then passed through the ML decoder before sent to the output for further processing. In this decoding process, the codeword 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 known as Type-I ML decoder, which determines, upon XOR combinations of the syndrome, [9] which bits need to be corrected. The second one is the Type-II ML decoder that calculates directly out of the codeword bits [6]. These two types are quite similar, when implementation is considered the second type uses less area, since it does not have a syndrome calculation as an intermediate step. For this reason the paper focused on this Type-II implementation.

2.1 Existent Plain ML Decoder

The existent plain majority logic decoder have the method of working in which from the received codeword itself the correct values of each bit under decoding can directly found out.

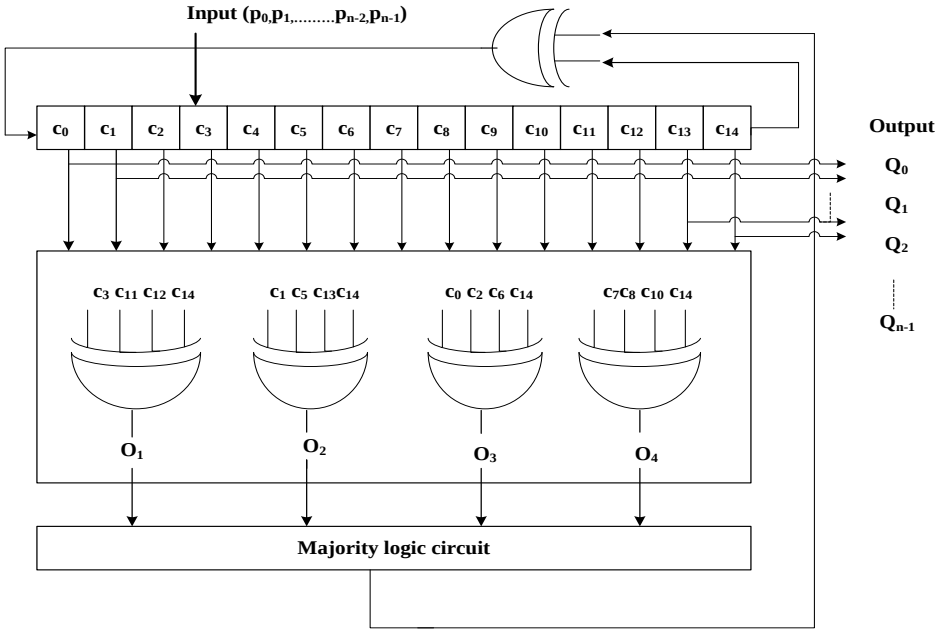

Fig 1. Serial one-step majority logic corrector for (15, 7, 5) EG-LDPC code

This method mainly consists of two steps.

1) Generating a specific set of linear sums of the received vector bits using the XOR matrix.
2) Finding the majority value of the computed linear sums.

It is the majority logic output which determines the correctness of the bit under decoding. If the majority output is '1', then the bit is inverted, otherwise would be kept unchanged. As described before, the ML decoder is simple and powerful decoder, which has the capability of correcting multiple random bit-flips depending on the number of parity check equations. It consists of four parts:

1) Cyclic shift register;
2) XOR matrix;
3) Majority gate; and
4) XOR for correcting the codeword bit under decoding.

The circuit implementing a serial one-step majority logic decoder [6],[12] for (15, 7, 5) EG-LDPC code is shown in Fig. 1.

The cyclic shift register is primarily stored with the input signal p, and shifted through all the taps or bits. The results $\{O_j\}$ of the check sum equations through the XOR matrix is calculated from the intermediate values in each tap. In the N^{th} iteration, the result would reach the final tap, producing the output signal, which is the decoded version of input [2]. This is the situation of error free case. The input p might correspond to wrong data corrupted by a soft error. The decoder is designed to handle this situation as follows.

Using the parity check sum equations hardwired in the XOR matrix the decoding starts at the very next moment after the codeword p are loaded into the cyclic shift register. The linear sum outputs {Oj} is then forwarded to the majority logic circuit which determines the correctness of the bit under decoding. If the majority of the Oj bits are "1" that is greater than the majority number of zeros then the current bit is erroneous and should be corrected, otherwise it is kept unchanged.

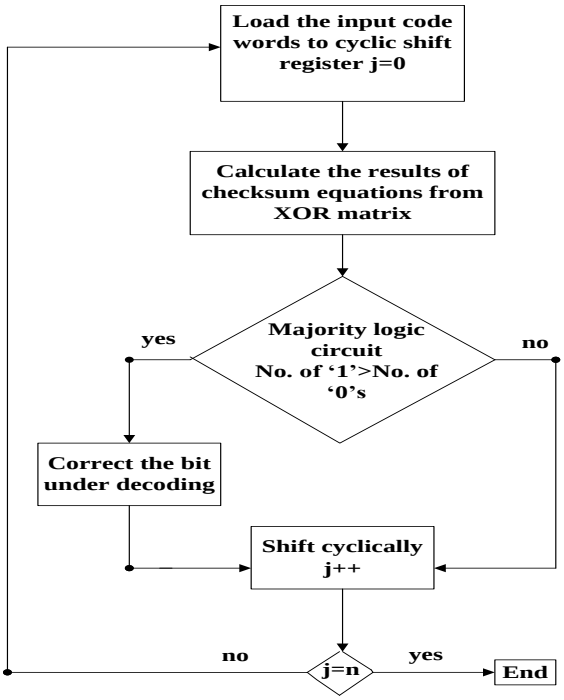

Fig 2. Flow diagram of the ML algorithm

The process is repeated and values of shift registers are rotated up to the whole N bits of the codeword are processed. When the entire parity check sum outputs are zero the codeword is correctly decoded. Further details on how this algorithm works can be found in [6], [12].

Flow diagram of the ML algorithm is shown in Fig 2. The Algorithm needs as many cycles as the number of bits in the input signal, which is number of taps, N, in the decoder and also needs same decoding time for both error and error free code words.

III. Ml Detector/Decoder

One step MLD can be implemented serially using the scheme in which corresponds to the decoder for the EG-LDPC code with N=15. First the data block is loaded into the registers. Then the check equations are resolved and if a majority of them has a value of one, the last bit is inverted. Then all bits are cyclically shifted. This set ofoperations constitutes one clock cycle or iteration. After N iterations, the bits are in the same position in which they were loaded. In the process, each bit may be corrected in one clock cycle.

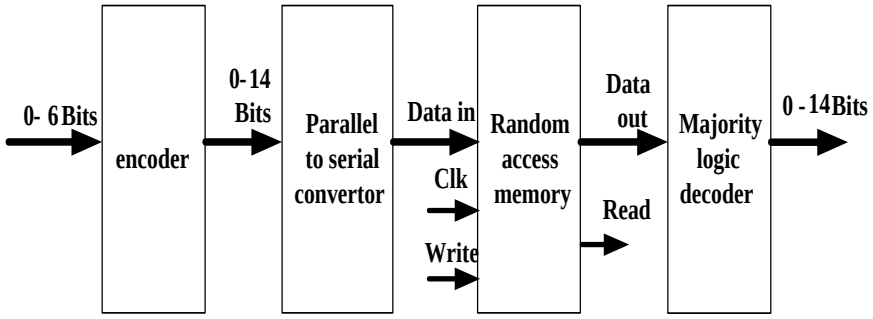

Fig 3. Schematic of a memory system with MLDD

As can be seen, the decoding circuitry is simple, but it requires a long decoding time if N is large. Figure 3 Shows The Memory System Schematic Of Proposed MLDD.

3.1 Design Structure of encoder

The systematic generator matrix to generate (15, 7, 5) EGLDPC code is shown in Fig 4[6].

$$G = \begin{bmatrix} i_0 \\ i_1 \\ i_2 \\ i_3 \\ i_4 \\ i_5 \\ i_6 \end{bmatrix} = \underbrace{\begin{bmatrix} 1&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0 \\ 0&0&1&0&0&0&0 \\ 0&0&0&1&0&0&0 \\ 0&0&0&0&1&0&0 \\ 0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&1 \end{bmatrix}}_{I_{7x7}\ \text{Matrix}} \underbrace{\begin{bmatrix} 1&0&0&0&1&0&1&1 \\ 1&1&0&0&1&1&1&0 \\ 0&1&1&0&0&1&1&1 \\ 1&0&1&1&1&0&0&0 \\ 0&1&0&1&1&1&0&0 \\ 0&0&1&0&1&1&1&0 \\ 0&0&0&1&0&1&1&1 \end{bmatrix}}_{P\ \text{Matrix}}$$

Fig 4. Generator matrix for the (15, 7, 5) EG-LDPC code.

The encoded vector consists of mainly two parts, the first part consist of information bits and second part is the parity bits, where each parity bit is simply an inner product of information vector and a column of X , from G=[I:X].

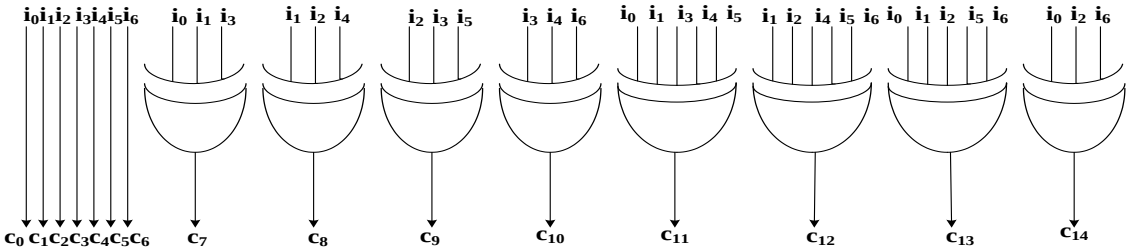

Fig 5. Structure of an encoder circuit for the (15, 7, 5) EG-LDPC code

The encoder circuit [6] to derive the parity bits of the (15, 7, 5) EG-LDPC code is shown in Figure 5. In this figure, the information vectors are (i0,….i6) and will be copied to (c0,..,c6) bits of the encoded vector, c. The rest of encoded vector bits (c7…c14), that is the parity bits are the linear sums (XOR) of the information bits.

IV. Proposed MLDD Structure

The advantage is that, proposed method stops intermediately in the third clock cycle when there is no error in data read [2] as illustrated in Figure.7, instead of decoding it for the whole codeword size of N. The XOR matrix is evaluated for the first three cycles of the decoding process, and when all the outputs {Oj} are "0,"the codeword is determined to be error-free and forwarded directly to the output. On the other hand, the proposed method would continue the whole decoding process to eliminate the errors [2] if the {Oj} contain at least a "1" in any of the three cycles.

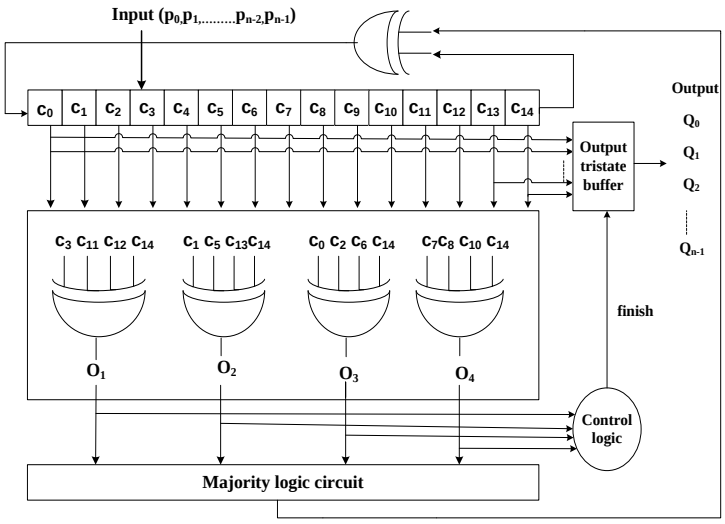

Fig 6. Schematic of the proposed MLDD for 15 bit code word.

A detailed schematic of the proposed design for 15 bit code word is shown in Fig 6. The figure shows the basic ML decoder with a 15-tap shift register, an XOR array to calculate the parity check sums and a ML circuit which will decide whether the current bit under decoding is erroneous and the need for its inversion. A detailed flow diagram of MLDD algorithm is shown in fig 7.

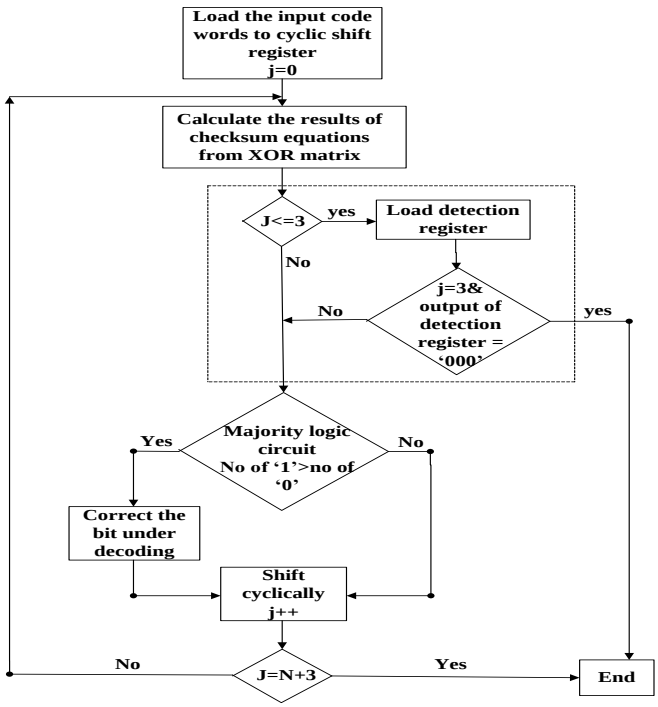

Fig 7.Flow diagram of the MLDD algorithm.

The plain ML decoder [2] shown in Figure 1 is also having the same schematic structure up to this stage. The additional hardware [2] planned for fault detection illustrated in Figure 8 those are:
a) The control logic unit and
b) The output tristate buffers.

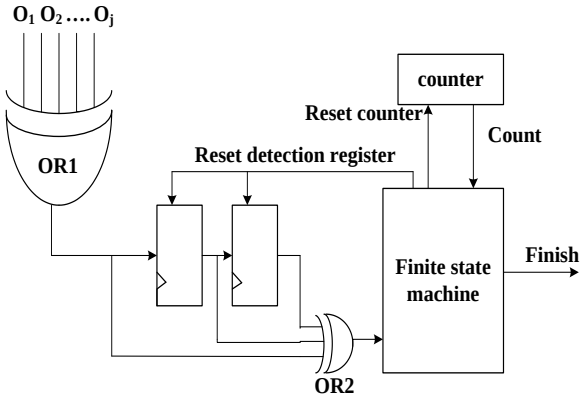

Fig.8 Schematic of the control unit.

The control unit generates a finish flag when there is no errors are detected in data read. The output tristate buffers are remains in high impedance state until the control unit sends the finish signal so that the current values are transferred to the output y from the shift register. The control unit shown in fig.8 [2] manages the detection process.

The detection process is controlled by the control unit. For distinguishing the first three iterations of the ML decoding, a counter is used here which counts up to three cycles. The control unit evaluates the output from XOR matrix Oj by giving it as input to the OR 1 gate. This output value is fed to two shift registers which has the results of the previous stages stored in it. The values are shifted accordingly. The third coming input is directly transfered to theOR 2 gate and finally all are evaluated in the third cycle in the OR 2 gate. If the result is "0," a finish signal is send by the finite state machine which indicates that the processed word is error-free. The ML decoding process continues until the end, if the result is "1".

The majority logic gate is implemented by using the conventional majority logic decoding mechanism that is two level logic [6].In case during the memory read access an error is detected, the XOR gate will correct it, by inverting the current bit under decoding.

4.1. Modified MLDD

This clearly provides a performance improvement respect to the existing MLD. The proposed method mostly would only take three cycles for decoding(five, if we consider the other two for input/output)since most of the words would be error free and would need to perform the whole decoding process only for those words with errors (which should be a minority).

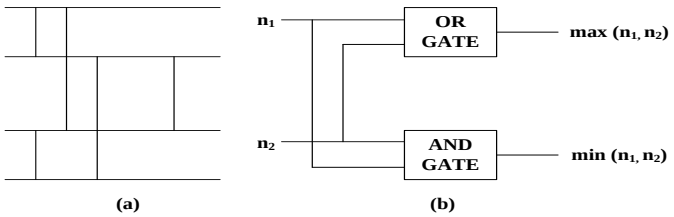

Fig 9. (a)Sorting network-four input
(b) Schematic of one comparator

Results

In this section the simulations results of the proposed Majority Logic Decoder/Detector (MLDD) shown.

RTL Schematic

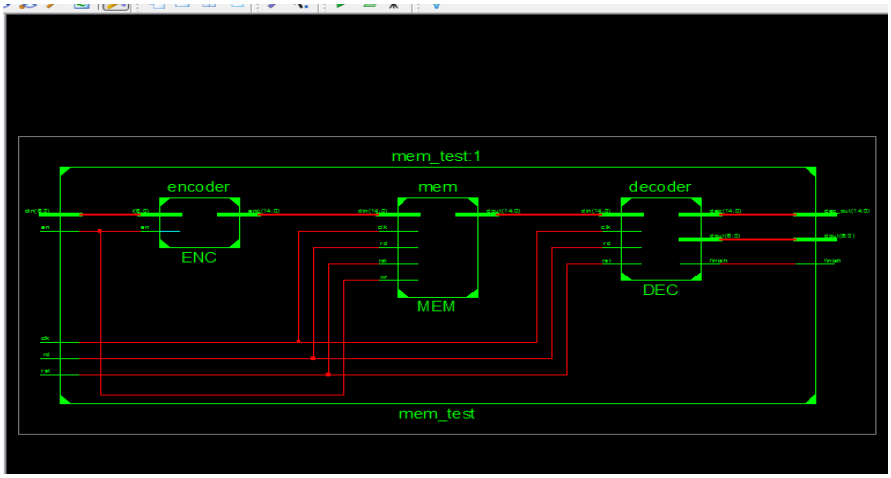

Fig 10. RTL schematic of proposed MLDD

The figure shows the Rtl schematic of proposed method.

Simulation Results

The proposed MLDD technique is simulated for error free condition during memory access, and the results are shown below in figure 11.

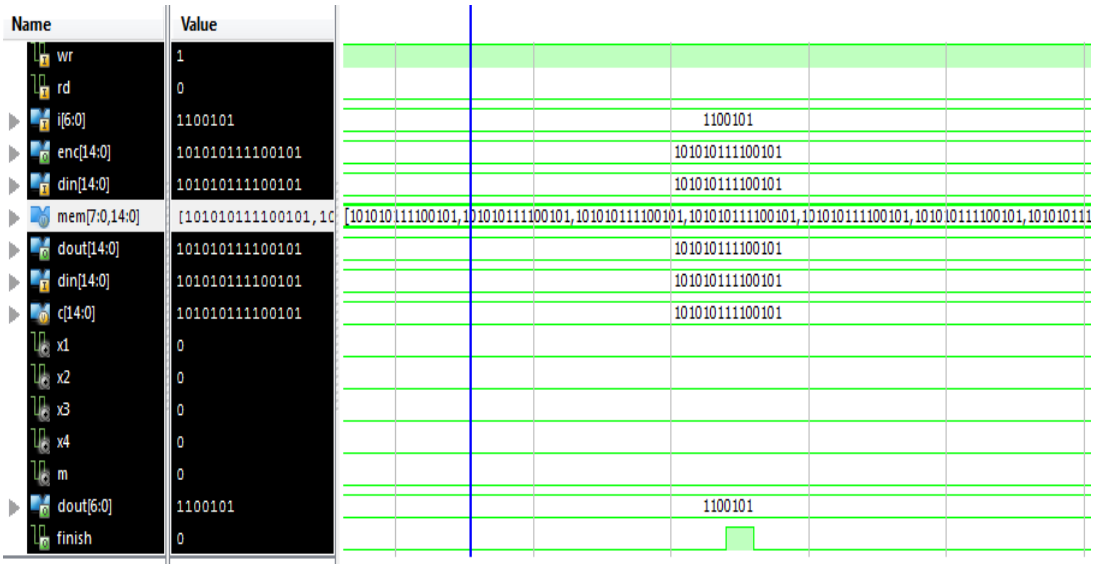

Fig 11.proposed technique MLDD without fault in memory

V. Conclusion

It has found that for error detection and correction (for codeword of 15), when comparing to the existing technique, a speed up of about 1100 ns is obtained when there is no errors in data read access. It's because the fault detection needs only three cycles and after the detection of an error free condition, the codeword is passed to the output without further corrections. 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 about 4% low power consumption than the existing MLD technique, since the proposed design detects the faults in just three cycles. Therefore a large no. of clock cycles (here 12 clock cycles) are saved and hence considerable reduction in power is achieved. MLDD error detector is designed as it is independent of the code word size and inference about area is that for large values of code word size, the area overhead of the MLDD actually decreases with respect to the plain MLD technique. i.e., for large values of code word size both areas are practically the same. Therefore the proposed MLDD will be an efficient design for fault detection and correction.

References

[1]. R. Naseer and J. Draper, "DEC ECC design to improve memory reliability in sub-100 nm technologies," in Proc. IEEE ICECS, 2008, pp.586–589.

[2]. Shih-Fu Liu, PedroReviŋgo, and Juan Antonio Meastro"Efficient majority fault detection with difference set codes for memory applications", IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 20, no. 1, pp. 148– 156, Jan. 2012.

[3]. M.A. Bajuraet al., "Models and algorithmic limits for an ECC- based approach to hardening sub-100-nm SRAMs," IEEE Trans. Nucl. Sci., vol. 54, no. 4, pp. 935– 945, Aug. 2007

[4]. 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.

[5]. 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.

[6]. 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.

[7]. 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.

[8]. B. Vasic and S. K. Chilappagari, "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, Nov. 2007.

[9]. S. Lin and D. J. Costello, *Error Control Coding*, 2nded Englewood Cliffs, NJ: Prentice-Hall, 2004.

[10]. T. Kuroda, M. Takada, T. Isobe, and O. Yamada, "Transmission scheme of high-capacity FM multiplex broadcasting system," *IEEE Trans. Broadcasting*, vol. 42, no. 3, pp. 245–250, Sep. 1996.

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

[12]. Pedro Reviriego,Juan A. Maestro, and Mark F. Flanagan, "Error Detection in Majority Logic Decoding of Euclidean Geometry Low Density Parity Check (EG-LDPC) Codes," *IEEE Transactions On Very Large Scale Integration (Vlsi) Systems* 1

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

[14]. T Shibuya and K. Sakaniwa, "Construction of cyclic codes suitable for iterative decoding via generating idempotents," *IEICE Trans. Fundamentals*, vol. E86-A, no. 4, pp. 928–939, 2003.