ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 4, April 2015

Design and Implementation of Efficient MLDD for Error Detection and Correction

Ms. Minal M. Jaiswal[1], Prof. Ashish E. Bhande[2]

PG Student, Electronics Department, HVPM's College of Engineering and Technology, Amravati, India [1]

Assistant Professor, Electronics Department, HVPM's College of Engineering and Technology, Amravati, India [2]

Abstract: A small transition delays and little faults create major concern in digital circuits. It Produce greater impact on not only for simple memory but also for most of the memory applications. During encoding and decoding process, the error may occur in the codeword which results in the mismatching or loss of information. Error detection and correction are main issues in the memory which needs to be identified and corrected. The proposed method will identify the error and correct the error in the memory application using Majority Logic Decoder and Detector (MLDD). MLDD corrects the error based on number of parity check equation. This technology reduces the N-iteration to three iteration, if the codeword doesn't contain any fault. It reduces the memory access time when there is no fault in data read. However it reduces the decoding time that increase memory application. Therefore delay is reduced. All the codes for MLDD design are written in VHDL. Modelsim SE 6.3f used for simulation process and the system is implemented on Sparatan-6 - XC6SLX16 - CSG324C FPGA kit.

Keywords: Encode, Error correction, Fault detection, Serial one step MLD, Majority logic decoder/detector, Memory, Soft error, sorting network.

I. INTRODUCTION

Now a day, data communication has become essential part of life and a lot of data is being transferred. Many communication channels are subject to channel noise, and thus errors may be introduced during transmission from the source to a receiver. There also are different ways of hacking, where the intruder modifies the data while communication. So to protect the confidentiality of the data and to retain the correctness of the data, secure communication is very important. There are different methods of implementing the secure communication. Each method has its own advantages and disadvantages. This project is an improvement on most of the existing methods for secure communication.

For reliable communication, errors must be detected and corrected. Some multi error bit correction codes are BCH codes, Reed Solomon codes, but in which the algorithm is very difficult. These codes can correct a large number of errors, but need complex decoders. Among the error correction codes, cyclic block codes have higher error detection capability, low decoding complexity and that are majority logic (ML) decodable. A low-density parity-check (LDPC) code is a linear error correcting code, used to avoid a high decoding complexity. one specific type of low density parity check codes, namely Euclidean Geometry-LDPC codes are used due to their fault secure detector capability, higher reliability and lower area overhead.

Error correction codes are commonly used to protect memories from so-called soft errors, which change the logical value of memory cells without damaging the circuit. As technology scales, memory devices become larger and more powerful error correction codes are needed. To this end, the use of more advanced codes has been recently proposed. These codes can correct a larger number of errors, but generally require complex decoders. To avoid a high decoding complexity, the use of one step majority logic decodable codes was first proposed in for memory applications. One step majority logic decoding can be implemented serially with very simple circuitry, but requires long decoding times. In a memory, this would increase the access time which is an important system parameter. Only a few classes of codes can be decoded using one step majority logic decoding. Among those are some Euclidean geometry low density parity check (EG-LDPC) codes which were used in, and difference set low density parity check (DS-LDPC) codes.

Error Correcting Codes are commonly used on memories. Codeword is parity bits and are appending with the data bits. Codeword is written into and read from the memories. The Hamming codes and Hsiao codes are used to correct the single bit-flip and double bit-flips in the memory, area and performances are high. For multiple errors, codes needed more parity bits in the above methods and hence not efficient. Bose-Chaudhuri-Hocquenghem (BCH)is authoritative random error correcting codes. It is used in the communication system. The demerits of these codes are redundancy requirement and complex decoding so they are not used in high speed memory application. Berlekamp-Massey, Euclidian and weight decoding algorithms require multi-cycles decoding, which is not adequate for embedded memories. Syndrome vectors are simple and power decoder. It detects the error in the codeword and corrects it. The drawback is that, for N-bits it will process N iteration. MLD is simple decoder and complexity. The demerits of the MLD are that

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 4, April 2015

performances of the system are reduced because it takes N-iteration for N-bit codeword.

Particularly, we identify a class of error-correcting codes (ECCs) that guarantees the existence of a simple fault-tolerant detector design. This class satisfies a new, restricted definition for ECCs which guarantees that the ECC codeword has an appropriate redundancy structure such that it can detect multiple errors occurring in both the stored codeword in memory and the surrounding circuitries. We call this type of error-correcting codes, *fault-secure detector capable ECCs* (FSD-ECC). The parity-check Matrix of an FSD-ECC has a particular structure that the decoder circuit, generated from the parity-check Matrix, is Fault-Secure. The ECCs we identify in this class are close to optimal in rate and distance, suggesting we can achieve this property without sacrificing traditional ECC metrics.

We use the fault-secure detection unit to design a fault-tolerant encoder and corrector by monitoring their outputs. If a detector detects an error in either of these units, that unit must repeat the operation to generate the correct output vector. Using this retry technique, we can correct potential transient errors in the encoder and corrector outputs and provide a fully fault-tolerant memory system. A method was recently proposed to accelerate a serial implementation of majority logic decoding of EG-LDPC codes. The design behind the method is to use the first iterations of majority logic decoding to detect if the word decoded contains errors.

If it is found there are no errors, then decoding process can be stopped. Decoding time is much more reduced because of stopping the iterations before fully completing. For a code with block length N, majority logic decoding which is implemented serially requires N iterations, so that the sizes of the code increase, so the decoding time also increase. In the proposed system, the errors are detected in parallel and in pipelining method. The detection of errors requires only single iteration where most of the errors are detected. The delay time is reduced for this proposed method is low compared to the prior technique.

II. LITERATURE REVIEW

[1]. Pedro Reviriego, Juan A. Maestro, and Mark F. Flanagan presented, Error Detection in Majority Logic Decoding of Euclidean Geometry Low Density Parity Check (EG-LDPC) Codes. A method was proposed to accelerate the logic decoding of difference set low density parity check codes. The detection of errors during the first iterations of serial one step Majority Logic Decoding of EG-LDPC codes has been studied.

The objective was to reduce the decoding time by stopping the decoding process when no errors are detected. The simulation results show that all tested combinations of errors affecting up to four bits are detected in the first three iterations of decoding. These results extend the ones recently presented for DS-LDPC codes, making the modified one step majority logic decoding more attractive for memory applications. The designer now has a larger choice of word lengths and error correction capabilities.

[2]. P. Kalai Mani, V. Vishnu Prasath presented, Majority Logic Decoding Of Euclidean Geometry Low Density Parity Check (EG-LDPC) Codes. Error detection in memory applications was proposed to accelerate the majority logic decoding of difference set low density parity check codes. This is useful as majority logic decoding can be implemented serially with simple hardware but requires a large decoding time. For memory applications, this increases the memory access time. The method detects whether a word has errors in the first iterations of majority logic decoding, and when there are no errors the decoding ends without completing the rest of the iterations. Since most words in a memory will be error free, the average decoding time is greatly reduced. In this brief, the application of a similar technique to a class of Euclidean geometry low density parity check (EG-LDPC) codes that are one step majority logic decodable. The results obtained show that the method is also effective for EG-LDPC codes.

[3]. M.Pramodh kumar, S.Murali mohan presented, Serial One-Step Majority Logic Decoder for EG-LDPC Code. In this brief, the detection of errors during the first iterations of serial one step Majority Logic Decoding of EG-LDPC codes has been studied. The objective was to reduce the decoding time by stopping the decoding process when no errors are detected. The simulation results show that all tested combinations of errors affecting up to four bits are detected in the first three iterations of decoding. These results extend the ones recently presented for DS-LDPC codes, making the modified one step majority logic decoding more attractive for memory applications. The designer now has a larger choice of word lengths and error correction capabilities.

[4]. Adline Priya presented, Low Power Error Correcting Codes Using Majority Logic Decoding. Moreover, the decoder architecture for LDPC codes are designed. And the simulation results for encoder, decoder, memory and detector are obtained. And also the majority logic decoder is implemented serially.

[5]. Senbagapriya.S. presented, An Efficient Enhanced Majority Logic Fault Detection with Euclidean Geometry Low Density Parity Check (EG-LDPC) Codes for Memory Applications. In this paper, the detection of errors during first iterations of serial one step Majority Logic Decoding of EG-LDPC codes has been presented. The simulation results show that the one step MLD would takes 15cycles to decode a codeword of 15-bits, which would be excessive for most applications. The MLD design requires small area but requires large decoding time and can able to detect two or few errors. Hence, memory access time increases. Another method, called MLDD can detect up to five bit-flips and consumes the area of majority gate. The proposed enhanced MLDD have the capability of detecting more than five bit flips and also reduces the area

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 4, April 2015

of majority gate by the use of sorting network. Finally, the decoding cycle slightly will increase compared to MLDD approach. These two designs are under progress.

[6]. M. Sakthivel, M. Karthick Raja, K R. Ragupathy and K. Sathis Kumar presented, Performance comparision of EG-LDPC codes with maximum likelihood algorithm over non binery LDPC codes. The power consumed by the components used for the construction of NB-LDPC codes and EG-LDPC codes with ML algorithm was simulated using Xilinx Power Estimator series7 XPE 2013. The power is calculated by loading number of flip-flops and slice LUT registers used. The comparison of Junction Temperature VS power and power consumed by elements for NB-LDPC and EG-LDPC with ML algorithm.

[7] K .Manikandan, G. Thiruselvi presented, Fault secure memory design using difference set codes. Modified Majority Logic detector/decoder (MLDD) code algorithms for difference set codes for memory applications have been proposed. A fault-detection mechanism, Modified MLDD, has been presented based on MLDD decoding using the DSCCs .The proposed technique is able to detect any pattern of up to more than five bit-flips in the three to nine cycles depending on the codeword length of the decoding process. This improves the performance of the design with respect to the traditional MLD approach. This is useful to avoid silent data corruption that can cause catastrophic failures in critical systems. By combining with MLDD techniques, the modified MLDD algorithms can be implemented very efficiently in terms of efficiency with a low latency. This makes them attractive for memory applications. The proposed scheme can be extended by requiring a larger of the majority logic check equations to take a value of one to perform a correction. This would increase the error detection capabilities at the expense of the error-correction capabilities.

[8] R. Meenaakshi Sundhari, C. Sundarrasu, M.Karthikkumar presented, an efficient majority logic fault detection to reduce the accessing time for memory applications. A fault detection technique, majority logic detector and decoder, has been presented based on majority logic decoding using the quasi cyclic LDPC codes. Exhaustive simulation test output shows that the proposed system is able to detect any pattern of up to five bit-flips in the first three cycles of the decoding, which improves the performance of the design with respect to the traditional majority logic decoding approach. In the same way, the majority logic detector and decoder in which error detector module has been proposed in a way that is independent of the code size. This makes its area overhead quite reduced compared with other traditional approaches such as the syndrome fault calculation.

[9] Anu Jose, M. Revathy presented, VLSI implementation of EG-LDPC codes using maximum likelihood decoding. The detection of errors during the first iterations of serial one step Majority Logic Decoding of EG-LDPC codes have been studied. The main objective of the work is to reduce the decoding time by stopping the decoding process when no errors are detected. The simulation results now shows that all the tested combinations of errors affecting up to four bits are detected and corrected in the first three iterations of decoding. These results extends to ones recently presented for DS-LDPC codes, making the modified one step Majority logic decoding more attractive for memory applications.

[10] D.Subalakshmi, P. S. Raghavendran presented, Error identification and correction for memory application using majority logic decoder and detector. An error detection mechanism, called Majority Logic Decoder and Detector has been proposed based on Majority Logic Decoder technique. The simulation results are explained about the detection and correction of the error using the proposed MLDD method. In this paper, the better performance is achieved by MLDD method compared to MLD and MLD with syndrome vector. The main advantage of the MLDD technique which is designed independent of the size of the codeword. This helps to reduce the area when compared to other technique. The further scope is to eliminate the silent error corruption. If the input has more than four bit error in the codeword, then the MLDD process is not exactly suitable to correct the codeword. In such case, silent fault corruption may occur. To reduce such fault, one more detection logic can be implemented after the completion of 73 iteration.

In order to overcome the drawback of MLD method, majority logic decoder/detector was proposed, in which the majority logic decoder itself act as a fault detector. In general, the decoding algorithm is still the same as the majority logic decoder. The difference is that instead of decoding all codeword bits, the MLDD method stops intermediately in the third cycle, which can able to detect up to five bit flips in three decoding cycles. So the number of decoding cycles can be reduced to get improved performance.

Syndrome vector method overcomes the demerits of Majority Logic Decoder (MLD) method. The faulty codeword are decoder, by adding the fault detector which calculates the syndrome value. This will not affect the performances of the system because most of the codeword are error-free. The main drawback of this system increases the complexity to the design. Based on parity check equation, the XOR matrix calculates the syndrome value. This increases the complexity of the syndrome value vector based on the size of the codeword. An error in the codeword is identified when the syndrome vector value is '1', then the ML decoder is used to correct the wrong codeword. Otherwise it forwards the codeword to the output, without correcting cycles. In this method, the performance is improved the performances of the system but additional module which increases the complexity to the design. Further, it increases the power complexity and reduces the performances of the system. It will increase the power consumption. Syndrome vector is oldest

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 4, April 2015

technology, which is used to detect the error in the codeword. Syndrome decoder is linear decoder. Hamming code is one of the example of syndrome decoder. Thus the proposed MLDD method overcomes the demerits of above existing method.

III. PROPOSED WORK

A. General schematic of MLDD

The general schematic of MLDD is shown in figure 1. Which consist of Encoder, Memory and MLDD.

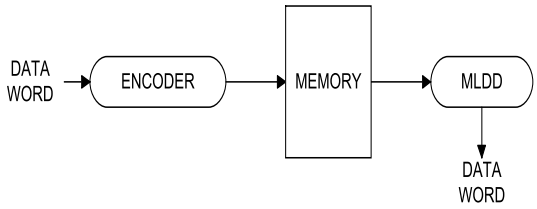

Fig. 1 General schematic of MLDD

Initially, the data words are encoded and then stored in the memory. When the memory is read, the codeword is then fed through the MLDD 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.

B. Overall MLDD system

In this section we present overall MLDD system in our proposed work. In this system we present the detail of encoder, memory, serial corrector, parallel corrector, and pipeline and parallel corrector and detector units of our proposed fault-tolerant memory system.

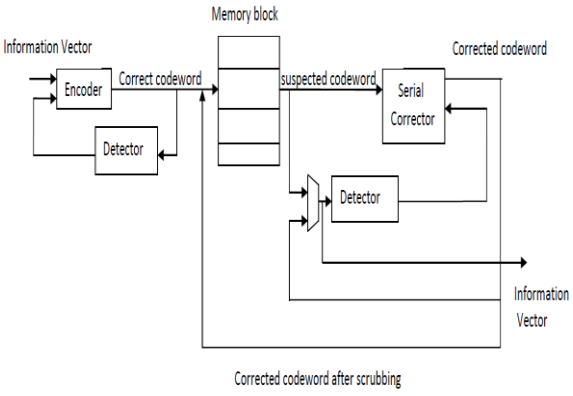

Fig.2 MLDD system with serial corrector

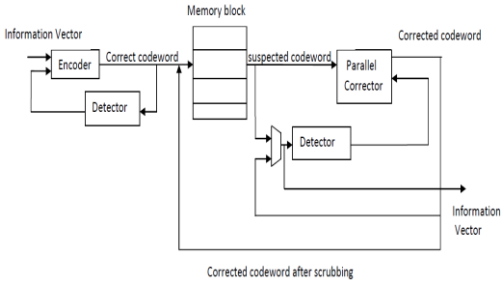

Fig.3 MLDD system with parallel corrector

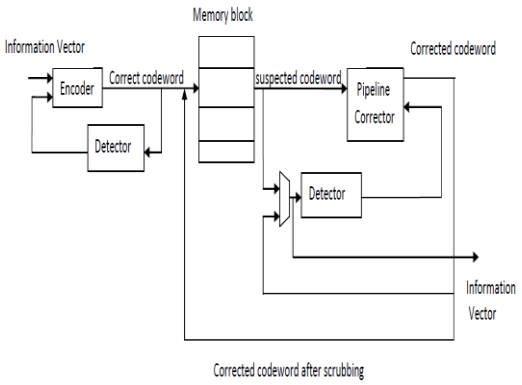

Fig.4 MLDD system with pipeline corrector

a. Encoder:

The information bits are fed into the encoder to encode the information vector. This section provides a brief introduction on linear block ECC's. Let I = (i0, i1, ..., i k−1) be k-bit information vector that will be encoded into n-bit codeword, C= (c0, c1, ..., c n− 1). For linear codes the encoding operation essentially performs the following vector-matrix Multiplication.

$C = I \times G$

Where, G is a k × n generator matrix.

A code is a systematic code if any codeword consists of the original k-bit information vector followed by (n – k) parity-bits. With this definition, the generator matrix of a systematic code must have the following structure.

$G = [I: X]$

Where, I is a k × k identity matrix and X is a k×(n−k) matrix that generates the parity-bits.

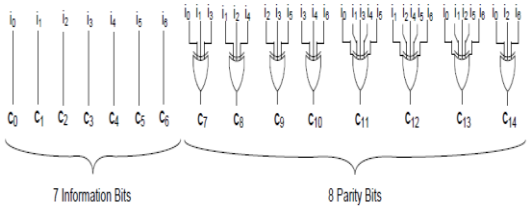

Fig. 1.1 Structure of an Encoder Circuit

Figure 1.1 shows the encoder circuit to compute the parity bits of the (15, 7, 5) EG-LDPC code. In this figure I = (i0, i1,…..i6) is the information vector and will be copied to C=(c0,……c6) bits of the encoded vector C, and the rest of encoded vector, the parity bits, are linear sums (XOR) of the information bits.

b. Fault Secure Detector:

The fault secure detector of the encoder verifies the validity of the encoded vector. If the detector detects any error, the encoding operation must be redone to generate the correct codeword. The codeword is then stored in the memory. A code is said to be cyclic code if for any codeword c, all the cyclic shifts of C is still a valid codeword. A code is cyclic if the rows of its parity-check matrix and generator matrix are the cyclic shifts of their first rows. The checking or detecting operation is the following vector-matrix multiplication.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 4, April 2015

$S = C \times H^T$
Where, H is an (n−k) ×n Parity-Check matrix. The (n − k)-bit vector S is called syndrome vector.
A syndrome vector is zero if C is a valid codeword and non-zero if C is an erroneous codeword.

c. Memory block:
Data bits stay in memory for a number of cycles and, during this period, each memory bit can be upset by a transient fault with certain probability. Therefore, transient errors accumulate in the memory words over time. In order to avoid accumulation of too many errors in any memory word that surpasses the code correction capability, the system must perform memory *scrubbing*. Memory scrubbing is the process of periodically reading memory words from the memory, correcting any potential errors, and writing them back into the memory. To perform the periodic scrubbing operation, the normal memory access operation is stopped and the memory performs the scrub operation.

d. Corrector:
During memory access operation, the stored code words will be accessed from the memory unit. Code words are susceptible to transient faults while they are stored in the memory. Therefore a corrector unit is designed to correct potential errors in the retrieved code words. In our design all the memory words pass through the corrector and any potential error in the memory words will be corrected. Similar to the encoder unit, a fault secure detector monitors the operation of the corrector unit.

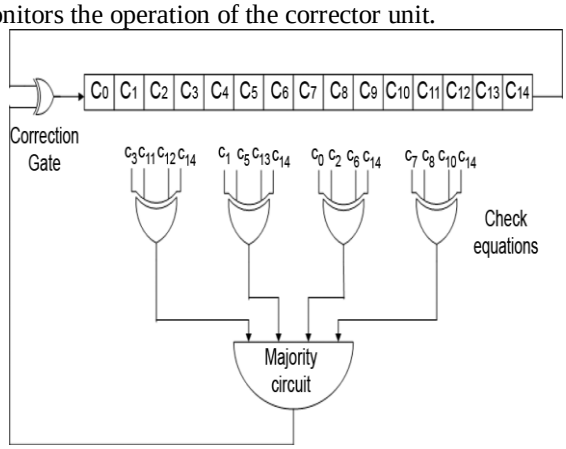

Fig.1.2 Serial one step majority logic decoder

To detect the errors serially the MLDD technique uses Serial One Step Majority Logic Decoder. The serial one step majority logic decoder is depicted in fig.1.2. In this decoder 15 bit data is first stored in the cyclic shift register. Then the inputs are assigned to the XOR gates. Since there is 15 bit data the XOR gates required are four. The bit to be detected should be given as one of the inputs for all the XOR gates. The outputs of the XOR gates are the check sum equations. The check sum equations consist of binary data. Then the Majority circuit outputs the data which is in major number. If the output of the majority circuit is '1' then the corresponding bit has the error else the bit is error free.

The output of the Majority circuit is given as one of the input to the correction gate. Another input to the correction gate is the bit which is under test. So the corrected bit is stored into the shift register where first cyclic shift occurs. This entire process is called as one iteration.

Majority circuit implementation: Here majority circuit implementation gate use Sorting Networks the majority gate has application in many other error-correcting codes, and this compact implementation can improve many other applications. We use binary *Sorting Networks* [15] to do the sort operation of the second step efficiently. An –input sorting network is the Structure that sorts a set of n bits, using 2-bit sorter building blocks. Fig.1.3 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. 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.

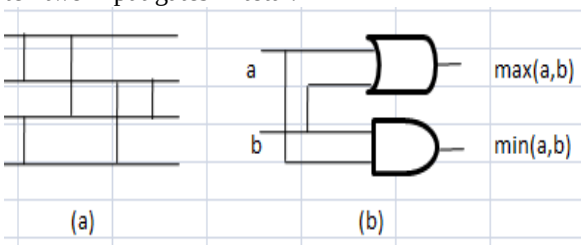

Fig.1.3 Four-input sorting network; each vertical line shows a one-input comparator. (b) One comparator structure.

Serial Corrector: As mentioned earlier, the same one-step majority-logic corrector can be used to correct all the n bits of the received codeword of a cyclic code. To correct each code-bit, the received encoded vector is cyclic shifted and fed into to the XOR gates as shown in fig.1.2. The serial majority corrector takes n cycles to correct an erroneous codeword. If the fault rate is low, the corrector block is used infrequently; since the common case is error-free codewords, the latency of the corrector will not have a severe impact on the average memory read latency. The serial corrector must be placed off the normal memory read path. This is shown in Fig.2. The memory words retrieved from the memory unit are checked by detector unit. If the detector detects an error, the memory word is sent to the corrector unit to be corrected, which has the latency of the detector plus the n round latency of the corrector.

I *Parallel corrector:* The corrector is used more frequently and its latency can impact the system performance. Therefore we can implement a parallel one-step majority corrector which is essentially n copies of the single one-step majority-logic corrector. Fig.1.4 shows a system integration using the parallel corrector. The logic blocks are same for the parallel MLDD as in the serial MLDD. But required is more number of majority gates

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 4, April 2015

and correction gates, each gate is assigned for a single bit which is shown in fig.1.4. The memory schematic for parallel processing MLDD is shown in fig.2. In Parallel schematic each bit of the code word fed for error detection and correction consist of its parity check equation, correction gate and majority gate.

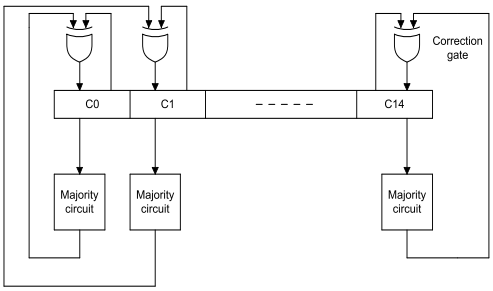

Fig.1.4 Parallel corrector design

Pipeline parallel corrector: The pipelining process is done for the proposed Parallel processing technique by adding registers as shown in fig.1.5. So that the delay is reduced compare to parallel processing. All the memory words are pipelined through the parallel corrector. This way the corrected memory words are generated every cycle. The detector in the parallel case monitors the operation of the corrector; if the output of the corrector is erroneous, the detector signals the corrector to repeat the operation.

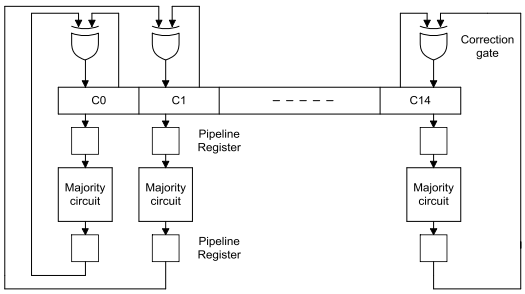

Fig.1.5 Pipeline corrector design

IV SIMULATION RESULT

The architecture is implemented using spartan6 family and XC6LX16 device in Xilinx 14.5.The proposed system is written in VHDL language and synthesized in Xilinx 14.5 and stimulated using Modelsim SE 6.3f. The results are shown in following figures. First set the clock and reset. Then gives the 7 bit input i.e. (c0.....c6) which is information bit and then we got 15 bit codeword i.e. (c0.....c14).

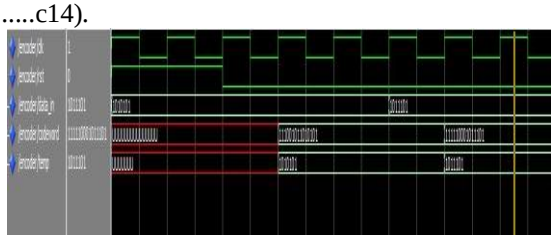

Fig.3.1 Simulation result for Encoder

In simulation process clock and reset bit is set then gives the 15 bit codeword which is received from the encoder. If the syndrome vector output is 0000 then the received codeword is error free otherwise in received codeword error is generated.

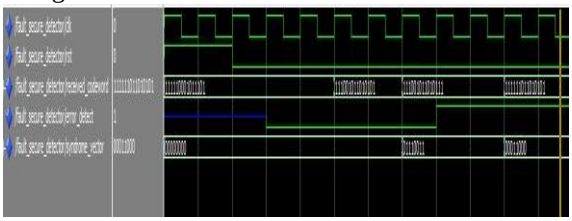

Fig.3.2 Simulation result for Fault secure detector

Memory in which chip select (CS), write enable (WE), output enable (OE) bits are used. First set the clock, for write operation select chip select bit and write enable bit is 1 also set output enable bit 0. Then write 15 bit codeword in address 0000 to 1111. For read operation select chip select bit and output enable bit is 1 also select write enable bit is 0. Give any address i.e. 0000 to 1111. The data_out shows the 15 bit codeword which stored in the given address.

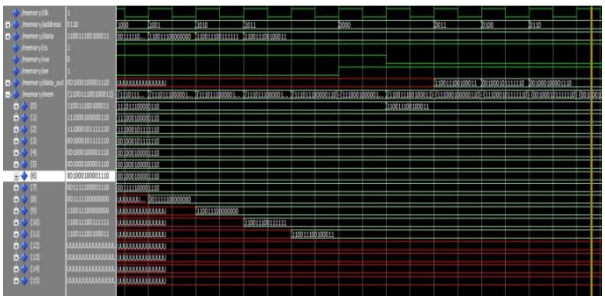

Fig.3.3 Simulation result for Memory

In serial corrector first set clock and reset then give the suspected codeword (gives 15 bit codeword one by one from c14.....c0). If the suspected codeword is error free then the majority_gate_out is 0 and find the corrected information vector. In suspected codeword error is generated then the majority_gate_out is 1and the serial corrector corrects that error and gives the correct information vector. Serial corrector detects and corrects up to two bit error. In serial corrector 15 cycles used for write the operation and 15 cycles used for read operation.

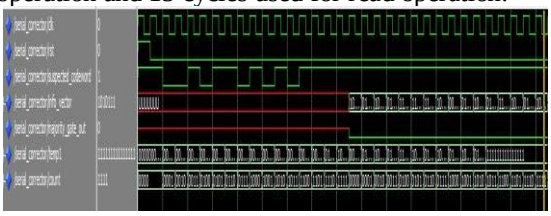

Fig.3.4 (a) Simulation result for serial corrector without error

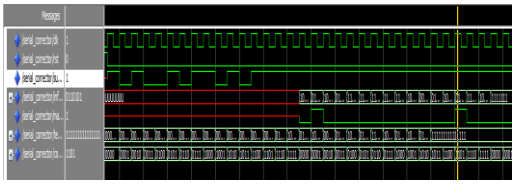

Fig.3.4 (b) Simulation result for serial corrector with error

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 4, April 2015

In parallel corrector first set clock and reset then gives the 15 bit suspected codeword. If the suspected codeword is error free then the majority_gate_out is 000000000000000 and the corrected information vector is shown. Otherwise the majority_gate_out is shows 1 of that bit errors are generated. Parallel corrector detects and corrects up to two bit error. In parallel corrector there are only two cycles are used, one for write operation and second for read the operation.

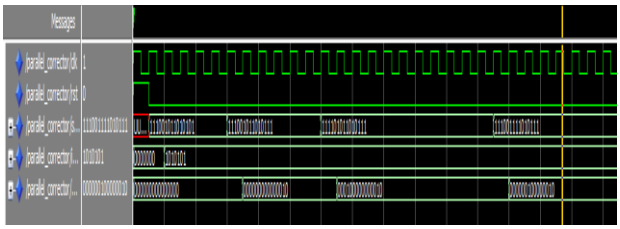

Fig.3.5 Simulation result for parallel corrector

In pipeline corrector first set the clock and reset also the pipeline_en bit is 1. Pipeline corrector performs the same operation of a parallel corrector. Only difference is that in pipeline corrector we used pipeline register which is reduced the delay compare to the parallel corrector.

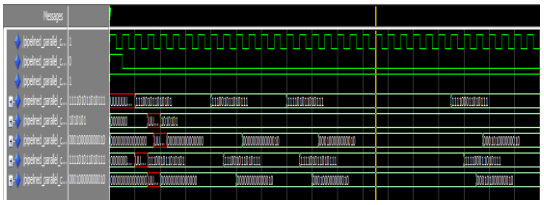

Fig. 3.6 Simulation result for pipeline corrector

For hardware implementation we used Nexys3 Xilinx Spartan-6 - XC6SLX16 – CSG324C FPGA kit is used. The Spartan-6 is optimized for high performance logic, and offers more than 50% higher capacity, higher performance, and more resources as compared to the Nexys2's Spartan-3 500E FPGA.

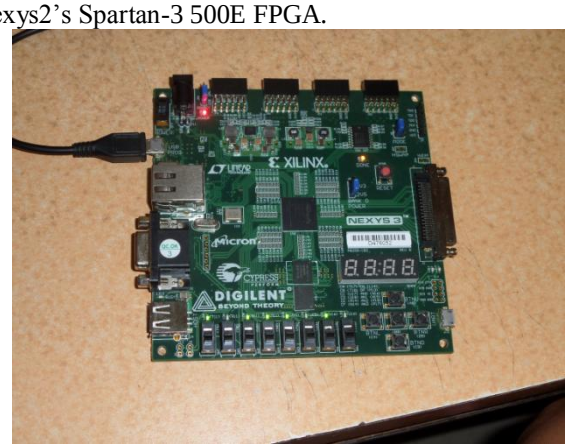

Fig.3.7 Hardware implementation on FPGA

For the hardware implementation we used clock, push buttons and LEDs on FPGA kit. A single 100MHz CMOS oscillator is used and two push buttons are used one for reset and second one for pipeline enable. Impact tool targets the FPGA device by initializing the chain and then programs the selected device by loading the generated bit file through USB (UART) programming cable. Output shows on LED which is shows the corrected information vector.

Design	**Propagation Delay**
Serial	54.51ns
Parallel	3.634ns
Pipeline	3.017nn

Table 3.1 Comparision table

IV. CONCLUSION

In this paper, we have presented a complete MLDD system in which serial, parallel and pipeline corrector are designed. The majority logic decoder/detector outcome that this method is simple and fast, which facilitate the efficient intend for more secured systems. Majority logic decoder/detector can be detect the number of errors and correct it. MLDD have the capability of reduces the area of majority gate by using sorting network. The objective was to reduce the decoding time means the speed is increases. The simulation result shows that all tested combinations of error affecting up to two bits are detected and corrected. The errors are detected by using serial, parallel and pipeline method. For simulation process serial corrector used 15 cycles for write operation and 15 cycles for read operation. Also the parallel and pipeline corrector in which two cycles are used one for write operation and second one for read operation. Therefore the delay time is reduced. The results are obtained by using Xilinx 14.5 and Modelsim SE 6.3f. The hardware implementation designed on Spartan-6 FPGA kit.

Future work, overall system computes on ASIC (application specific integration circuit) or SoC (system on chip). Because the speed of ASIC is faster than FPGA. This gives enoromous opportunity for speed optimizations. SOC can be designed to work at particular frequency and FPGA has the frequency limit like Spartan cannot run more the 500 MHz. SOC contains all the modules including PLL, Data converters & processors etc. FPGA can be programmed to work as a chip for the particular application & may not be having all the modules inside of FPGA itself.

ACKNOWLEDGMENT

We are very grateful to our HVPM College of Engineering and Technology to support and my guide **Prof. Ashish Bhande** and other faculty also associates of ENTC department who are directly & indirectly helped me for this paper.

REFERENCES

[1]. P. Kalai Mani, V. Vishnu Prasath, "Majority Logic Decoding Of Euclidean Geometry Low Density Parity Check (EG-LDPC) Code," International Journal of Innovative Research in Computer and Communication Engineering, Vol.2, Special Issue 1, March 2014.

[2]. Adline Priya, "Low Power Error Correcting Codes Using Majority Logic Decoding," International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622, March 2014.

ISSN (Online) 2278-1021
ISSN (Print) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 4, Issue 4, April 2015

[3]. Anu Jose, M.Revathy, "VLSI implementation of EG-LDPC codes using maximum likelihood decoding," International Journal of Science, Engineering and Technology Research (IJSETR), Volume 3, Issue 4, April 2014.
[4]. M. Sakthivel, M. Karthick Raja , K R. Ragupathy and K. Sathis Kumar, "Performance comparision of EG-LDPC codes with maximum likelihood algorithm over non binery LDPC codes.," International Journal of Computational Science and Information Technology (IJCSITY) Vol.2, No.2, May 2014.
[5]. M. Pramodh Kumar, S. Murali Mohan, "Serial one-step majority logic decoder for EG-LDPC code," IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 1 Issue 6, August 2014.
[6]. 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 Trans. Very Large Scale Integration (VLSI) Systems, Vol. 21, No. 1, January 2013.
[7]. D. Subalakshmi, Major. P. S. Raghavendran, "Error Identification and Correction for Memory Application using Majority Logic Decoder and Detector," International Journal of Computer Applications (0975 – 8887) Volume 64– No.10, February 2013.
[8]. R. Meenaakshi Sundhari, C .Sundarrasu, M. Karthikkumar, "An Efficient Majority Logic Fault Detection to reduce the Accessing time for Memory Applications," International Journal of Scientific and Research Publications, Volume 3, Issue 3, March 2013.
[9]. Senbagapriya.S, "An Efficient Enhanced Majority Logic Fault Detection with Euclidean Geometry Low Density Parity Check (EG-LDPC) Codes for Memory Applications," International Journal of Engineering Science and Innovative Technology (IJESIT) Volume 2, Issue 6, November 2013.
[10]. S. Liu, P. Reviriego, and J. Maestro, "Efficient majority logic fault detection with difference-set codes for memory applications," IEEE Trans. Very Large Scale Integer. (VLSI) Syst., vol. 20, no. 1, pp. 148–156, Jan. 2012.
[11]. K .Manikandan, G. Thiruselvi, " Fault Secure Memory Design using Difference Set Codes," Special Issue of International Journal of Computer Applications (0975 – 8887) on International Conference on Electronics, Communication and Information Systems (ICECI 12).
[12]. Vijaykumar K. and Dr. K. Ashok Babu, "Fault tolerant nano-memory with fault secure encoder and decoder," Vol. 3 No. 1 Jan 2011.
[13]. S. Ghosh and P. D. Lincoln, "Low-density parity check codes for error correction in nanoscale memory," SRI Computer Science Lab., Menlo Park, CA, Tech. Rep. CSL-0703, 2007.
[14]. S. Lin and D. J. Costello, Error Control Coding, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2004.
[15]. J. Bhaskar, A VHDL Primer, 3rd edition, PEARSON, Prentice Hall, 2006.
[16]. http://www.xilinx.com.
[17]. Nexys3™ Board Reference Manual, Revision: April 10, 2013.