International Journal of Combined Research & Development (IJCRD)
eISSN:2321-225X;pISSN:2321-2241 Volume: 4; Issue: 1; January -2015

Energy Efficient Layer Decoding Architecture for LDPC Decoder

Jyothi B R
Lecturer
KLS's VDRIT
Haliyal-581329

Sangamesh G Tamburimath
Firmware Development Engineer
BITCOMM Technologies
Noida-201301

Abstract- Low Density Parity-Check (LDPC) codes are best error correcting codes these codes are less complex and easy to construct so LDPC codes are used in communication system, it needs to access large amount of data so it requires large amount of memory access which leads to high energy consumption. To reduce the energy consumption, the memory access is reduced by decreasing the decoding time and complexity using proposed one step Majority logic decodable(MLD) technique for EG-LDPC codes. The turbo-decoding message-passing (TDMP) algorithm is used in the proposed architecture-aware (AA) LDPC code which has a faster convergence rate and hence a throughput advantage in the standard decoding algorithm. This work aims to error detection in the decoding process. The one step Majority Logic Decodable(MLD) method proposed accelerates the logic decoding of Euclidean Geometry Low Density Parity check(EG-LDPC) requirements. In this method error detection is done for first few iterations, if there are no errors it will decode the data directly. The most words within a memory will be error-free, so the typical decoding time is greatly decreased. The outcomes obtained show that the method is effective for EG-LDPC codes of long length. The simulation and synthesis are done using XILINX ISE 13.1, the code is written in verilog. The proposed LDPC decoder architecture requires only 3.30ns time for decoding process and number of LUTs used is only 176 which are less compared to conventional decoder architecture. It requires less memory access. Hence the proposed architecture is Energy Efficient.

Key words- LDPC, EG-LDPC, MLD, Error correcting codes,

1. INTRODUCTION

Low Density Parity Check codes (LDPC) get acquired major consideration as a result of close to Shannon limit performance. They have been used in a number of cellular requirements such as DVB S-2, IEEE 802.16e along with 802.11n because of their exceptional problem improving performance.

The main goal of this work is concentrating on Decoder part in the communication system. At the decoder section LDPC codes are used to decode the data along with Error Correcting Codes (ECC).

Several Error Correcting Codes(ECC) have been developed over decades to perform encoding and decoding of data. They vary in their construction, performance, computation, and implementation complexity. Some well known error correcting codes are convolutional code, Reed Solomon, BCH(Bose- Chaudhuri-Hocquenghem), Turbo and LDPC codes.

Among different error correcting codes LDPC codes are best error correcting codes for many applications because of their best characteristics like low density, less complexity in the decoding process and also it is easy to represent the LDPC codes than other codes.

There are different types of LDPC codes 1) Pseudo random LDPC codes-These are initial LDPC codes. 2) Architecture Aware (AA)-LDPC codes-These are structured codes, whose parity check matrix is built according to specific patterns. Since they support an efficient partial-parallel hardware VLSI implementation, AA-LDPC codes have been adopted in several modern communication standards.

Finite geometries have been used to derive many error correcting codes some examples are

1) Difference Set (DS)-LDPC codes- The first three iterations will detect all errors affecting four or fewer bits, and also errors affecting five bits were additionally always detected.

2) Euclidean Geometry (EG)-LDPC codes- These codes are cyclic. These will detect the errors in the first three iterations only, if no error found it will decode the data directly. Thus it is fitting well to the requirements of modern memory systems.

So in this work Euclidean Geometry LDPC codes were used for the error detection. Majority Logic Decodable(MLD) is the sub class of EG-LDPC codes. Codes in this subclass are also cyclic.

For a code with block length N, majority logic decoding (when implemented serially) requires N iterations, so that as the code size grows, so does the decoding time. In the proposed approach, only the first three iterations are used to detect errors. If there are no errors, then decoding can be stopped without completing the remaining iterations thereby greatly reducing the decoding time and therefore achieving a large speed. The probability of undetected errors was also found to decrease as the code block length increased. For a billion error patterns only a few errors (or sometimes none) were undetected. This may be sufficient for some applications.

Another advantage of the proposed method is that it requires very little additional circuitry as the decoding circuitry is also used for error detection. The additional area required to implement the scheme was only around 1% for large word sizes.

1.1 PROBLEM DEFINATION

- The earlier LDPC decoder consumes more energy for long length code.
- Decoding complexity is more.

High power consumption is one of the bottlenecks for LDPC decoder. However how to further reduce its energy consumption is a challenging design problem.

1.2 OBJECTIVES

- Reducing the complexity of the decoder.
- Finding the errors in the first 3 iterations.
- Increase the speed of the decoding process.

2. LITERATURE SURVEY

LDPC codes are invented in early 1960's by Robert G Gallager[1]. It is one of the most attractive methods in the field of communication for the transmission of data in the noisy channel in a easy way.

Any linear code incorporates a representation as a code associated to bipartite graph then that code is referred to as a low density parity check (LDPC) code.

The crucial innovation was Gallager's introduction of iterative decoding algorithms (or message-passing decoders) which he showed to achieving a significant fraction of channel capacity at low intricacy. Except for the papers by means of Zyablov and Pinsker [2], Margulis [3], and Tanner [4] the field then laid unknown for next 30 years. Interest in LDPC codes leads to the discovery of turbo codes and LDPC requirements were independently rediscovered by equally MacKay and Neal [5] and Wiberg [6].

VLSI architectures for low density parity-check (LDPC) decoders amenable to low- voltage and low-power functioning. First, a highly-parallel decoder buildings with low routing overhead can be described. Second, an efficient method is proposed to detect early convergence on the iterative decoder and terminate the computations, thereby reducing dynamic electric power. With early termination, the prototype is efficient at decoding with 10.4pJ/bit/iteration, while performing within 3 dB on the Shannon limit at a BER of 10 sufficient reasons for 3.3Gb/s total throughput. 7pJ/bit/iteration while maintaining an overall throughput of 648 Mb/s, due to highly-parallel architecture.[19].

In a recent paper, a method 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, we study 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. Extensive simulation results are given to accurately estimate the probability of error detection for different code sizes and numbers of errors[25].

3. IMPLEMENTATION OF EG-LDPC DECODER

Flow chart briefs about various steps involved in the implementation for the LDPC decoder. Each of these steps is explained in the following section. The Flow chart is shown in the fig 3.1

Implementation of LDPC decoder is described in the following section

1. Generation of parity check matrix for the given code

Parity check matrix can be generated using the equation

$$H(x)=(1+x^{N}) / G(x)$$

Where N—total number of bits

$$H(x)=(1+x^{63}) / (1+x^{n}+\ldots\ldots+x^{26})$$

While representing the H-matrix we will make it into two matrices

1) Parity matrix
2) Identity matrix

$$H=[P^{T}|I_{n-k}]$$

Where P is the given parity matrix and I is the identity matrix

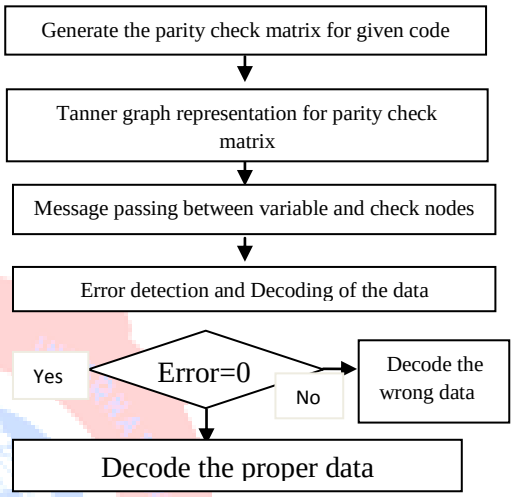

Fig 3.1 Flow chart of LDPC decoder implementation

Ex:- If we are performing 63 bit code rate parity matrix will be upto 0 to 36 columns and Identity matrix will be 37 to 63 columns.

H-matrix:

$$\begin{bmatrix} 0 & - & - & 36 & 37 & - & - & 63 \\ - & & & & & & & \\ - & & & & & & & \\ - & & & & & & & \\ 63 & - & - & 36 & 37 & - & - & 63 \end{bmatrix}$$

This represents parity matrix — This represents Identity matrix

Parity matrix:

$$\begin{array}{c} 0 \\ - \\ - \\ - \\ - \\ - \\ - \\ \\ 63 \end{array} \begin{bmatrix} 37 & - & - & - & 63 \\ 1 & 0 & & & 0 \\ 0 & 1 & - & & - \\ 0 & 0 & - & - & \\ & & & - & \\ & & & - & \\ & 1 & & & - \\ 0 & 0 & & & 0 \end{bmatrix}$$

Identity Matrix:

$$\begin{array}{c} 0 \\ - \\ - \\ 25th \\ 26th \\ 27th \\ - \\ \\ 63 \end{array} \begin{bmatrix} 0 & 0 & 0 & - & - \\ - & & & & \\ - & & & & \\ 1 & 0 & 0 & & \\ 0 & 1 & 0 & & \\ 0 & 0 & 1 & & \\ - & - & - & & \\ 0 & 0 & 0 & - & - \end{bmatrix}$$

If the value of 25th row first column in the identity matrix is high then in the next 26th row it will be shifted down to 2nd column and so on.

2. Tanner graph representation for the parity check matrix

Based on the Parity matrix generated Tanner graph is drawn by considering all the columns are variable node and all the rows are check nodes.

Then variable nodes and check nodes are connected depending upon the number of 1's in the matrix like if there is 1 in the 1st row and 1st column position then those two nodes are connected to each other like that the process repeats.

Tanner graph representation is shown in fig 3.2

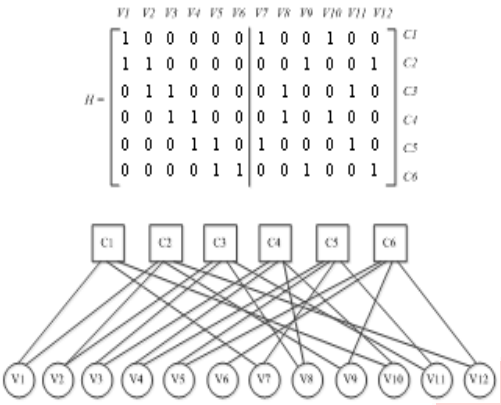

Fig 3.2 Tanner graph representation

3. Message passing between variable node and check node

Message passing between variable node and check node is done according to tanner graph. The decoding can be done by message passing between the nodes through number of iterations. The number of iterations can be fixed by the user. Here the initial message stored in the variable node is first sent to the check node then the check node will update the message and send it back to the variable node this way the process continues till the fixed number of iterations. After completing the iterations the message will decode.

To reduce the complexity of the decoding the min sum algorithm is used in the message passing technique.

The messages updatation in the variable node and check node is done using following equations

Variable node process

$$Q_{m,n}^{(q)} = \lambda n + \sum_{i\in[Mn\setminus m]} R_{i,n}^{(q)}$$

Where $Q_{m,n}^{(q)}$ is the msg from variable to check node

λn is the intrinsic LLR of the variable node n

$R_{m,n}^{(q)}$ Is the msg from check node to variable node

Soft output for the variable node is given by

$$\Lambda_n^{(q)} = \lambda n + \sum_{i\in\{Mn\}} R_{i,n}^{(q)}$$

Check node process

$$-\operatorname{sgn}(R_{m,n}^{(q+1)}) = \prod_{j\in\{Nm\setminus n\}} -\operatorname{sgn}(Q_{m,j}^{(q)})$$

$$\left|R_{m.n}^{(q+1)}\right| = \phi^{-1}\left\{\sum_{j\in\{Nm\setminus n\}} \phi\left(\left|Q_{m,j}^{(q)}\right|\right)\right\}$$

Where $\phi(x) = \phi^{-1}(x) = -\ln\left(\tanh\left(\frac{x}{2}\right)\right)$

The min sum algorithm equation is given by

$$\left|R_{m,n}^{(q+1)}\right| = \min_{j\in\{Nm\setminus n\}} \left|\Gamma_{m,j}^{(q+1)}\right|$$

4. Error detection and decoding of the data

The decoding of EG-LDPC codes are done along with the error detection in the decoder part.

One step MLD can be implemented serially using the plan shown in Fig.3.3 which corresponds towards decoder for the EG-LDPC signal with N=15. First the information block is loaded into this registers. Then the check equations are computed and if most them has a value of a single, the last bit is inverted. Then all bits are cyclically altered. This set of operations takes its single iteration: after N iterations, the bits are in the same position by which they were loaded. In accomplishing this, each bit may be corrected only once. As can be seen, this decoding circuitry is simple, but it really requires a long decoding time period if N is large.

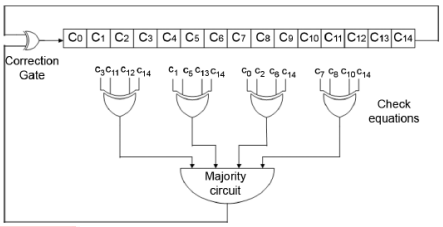

Fig.3.3 Serial one step majority logic decoder for the (15,7) EG-LDPC code

1) All equations include this variable whose value is stored within the last few register (the one marked as c_{14}).

2) The rest of this registers are included in at most one of many check equations.

If errors are detected in the first few iterations of MLD, then no errors are detected with other iterations, the decoding can be stopped without completing all of other iterations. In the first iteration, errors will be detected when at least one of the check equations is affected with an odd number of parts in error. In the next iteration, as bits are cyclically altered by one position, errors may affect other equations such that some errors undetected from the first iteration will be detected. As iterations advance, all detectable errors will eventually be detected.

Finally if there are no errors then the data will decode properly if there is error found during iterations then data will corrupt and it will decode the wrong data.

3.1 Error detection in decoding of EG-LDPC codes

One particular step majority logic decoding can be implemented serially with very basic circuitry, but requires long decoding times. In a recollection, this would increase the access time which can be an important system parameter. Just 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 of used in, and Difference Set Low Density Parity Check (DS-LDPC) codes.

A procedure to help accelerate a serial implementation connected with majority logic decoding of DS-LDPC codes. The idea behind the method is by using the first iterations of majority logic decoding to detect if word being decoded contains errors. If there are no errors, then decoding can be stopped without completing the remaining iterations, therefore greatly reducing this decoding time. For a signal with block length N, majority logic decoding (when implemented serially) requires N iterations, in addition to being the code size grows, thus does the decoding time. Within the proposed approach, only the initial three iterations are used to help detect errors, thereby achieving a sizable speed increase when N is large. In DS-LDPC codes, all error combinations of approximately five errors can be detected from the first three iterations. Also, errors affecting over five bits were detected that has a probability very close to one particular. The probability of undetected errors to the code was also found decrease since block length increased. Regarding a billion error patterns only a few errors (or sometimes none) had been undetected. This may be sufficient for many applications.

N	K	J	t_{ML}
15	7	4	2
63	37	8	4
255	175	16	8
1023	781	32	16

Another advantage of the proposed method is it requires very little additional circuitry since the decoding circuitry is also used for error detection. For example, the excess area required to implement this scheme was only around 1% pertaining to large word sizes.

The method proposed relies upon the properties of DS-LDPC codes and so it is not directly applicable to additional code classes. In the following, a similar approach for EG-LDPC codes is presented.

Finite geometries are actually used to derive many error-correcting codes. One example is EG-LDPC codes that are based on the structure of Euclidean geometries over a Galois field. Among EG-LDPC codes we have a subclass of codes that is one step majority logic decodable (MLD). Codes with this subclass are also cyclic. The actual parameters for some of these codes are shown in Table 3.1, where N would be the block size, K the quantity of information bits, j the quantity of MLD check equations and t_{ML} the quantity of errors that the code can correct using one step MLD.

Table 3.1 One Step MLD EG-LDPC Unique Codes

The DS-LDPC codes most errors can be detected in the first a few iterations of MLD. Based on simulation results and on a theoretical proof for case of two errors, this hypothesis was made.
"Given a word read from a memory protected with DS-LDPC codes, and affected by up to help five bit-flips, all errors can be detected in only three decoding cycles".

Then this proposed technique was implemented with verilog and synthesized, showing in which for codes with large obstruct sizes the overhead is low. This is because the recent majority logic decoding circuitry is reused to perform error detection and only some extra control logic is needed.

For codes with small words and affected by only a few bit flips, it is practical to build and check all possible mistake combinations. As the code size grows and how many bits flips increases, it is no longer feasible to extensively test all possible combinations. Thus the simulations are done within two ways, by exhaustively examining all error combinations when it can be feasible and by checking at random generated combinations in the other countries in the cases.

The results for these exhaustive checks are shown within Table 3.2. These results prove the hypothesis for those codes with smaller word measurement. For N=255 around three errors have been extensively tested while for N=1023 merely single and double error combination have been exhaustively tested.

Table 3.2 Undetected Errors in Exhaustive Checking

N	1error	2 errors	3 errors	4 errors
15	0	0	0	0
63	0	0	0	0
255	0	0	0	-
1023	0	0	-	-

To check the results of the radical checks for larger codes and quantity of errors, simulations using random mistake patterns have also been used. In all the findings, one billion error combinations are tested.

It can be viewed that for errors affecting over four bits there is a small number of error combinations that will not necessarily be detected in the first three iterations. This number lowers with word size and also with the number of errors. The decrease with this word size can be explained as follows: the larger the word dimensions, the larger the number of MLD check equations (see Table 3.1) and therefore it is more unlikely that errors occur in the same equation. As for this number of errors, a related reasoning applies the more problems occur, the larger the probability that an odd quantity of errors occurs in no less than one equation. Finally it must be noted the probabilities of undetected errors vary for an even and an odd quantity of errors as in the latter case, one of the errors must occur in a very bit which is not examined by any equation. The simulation results presented suggest that all errors affecting three and four bits could be detected in the first a few iterations. For errors

affecting an increased number of bits, there is often a small probability of not getting detected in those iterations. Intended for large word sizes, the likelihood are sufficiently small to be acceptable in several applications.

In summary, the first three iterations will detect all errors affecting four or less bits, and almost every different detectable error affecting more bits. This is a slightly worse performance than regarding DS-LDPC codes where problems affecting five bits were moreover always detected. However, the majority logic circuitry is very simple for EG-LDPC codes, as the number of equations is a power connected with two and an approach based on sorting networks can be employed to reduce the cost with the majority logic voting. In EG-LDPC codes have block lengths near a power of two, thus fitting well towards requirements of modern memory systems. This may mean that in most cases it may be more convenient to use an EG-LDPC code and hold a word size compatible with existing designs (power of two) compared to using a DS-LDPC code requiring another word size or a shortened version of that code. When using word size which is a power of two, there is a bit which is not as used by the EG-LDPC code (see table 4.1). This bit can be taken for a parity covering all bits in the word that would detect all errors affecting an odd quantity of bits. In that case, the planning using the EG-LDPC would furthermore detect all errors affecting all 5 or fewer bits.

Two types of errors:

- Soft error: This affects in the memory data.
- Hard error: This affects the hardware components

Some useful formulae for LDPC calculations:

1) For getting number of errors in code rate 2^{2s-1}=<code rate>
2) Code length=$2^{2s}+2^{s}+1$
3) Message bits=$2^{2s}+2^{s}+3^{s}$
4) Parity check bits=$3^{s}+1$
5) Minimum distance(d)=$2^{s}+2$.

4.SIMULATION AND RESULTS

The synthesis and simulations are done using XILINX ISE 13.1 version. The code is written in verilog.

The implementation of LDPC decoder is done in the following flow

1. Generation of parity check matrix.
2. Message passing between variable node to check node based on parity check matrix.
3. Decoding of given data along with error detection using MLD technique.

The main goal of this work is concentrating only on decoder part of the communication system so here the encoder input is assumed with different possible combinations. The implementation of the decoder result is simulated in the following way.

1. Generation of parity check matrix.

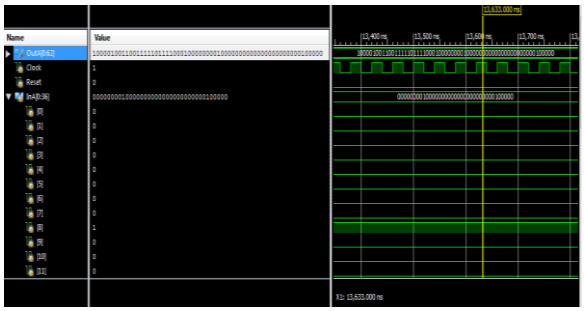

Fig 4.1 Simulation output for Parity check matrix generation

2. Message passing between variable node to check node based on parity check matrix.

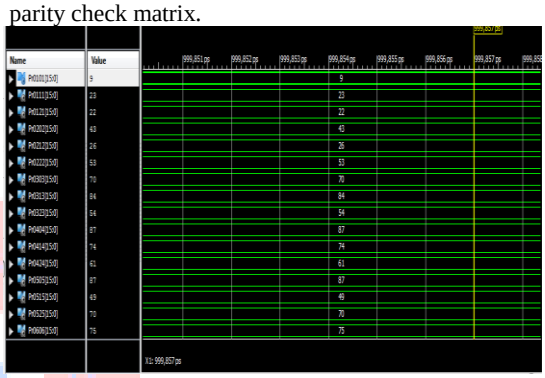

Fig 4.2 Simulation output for the messages updated in the variable node

3. Decoding of given data along with error detection using MLD technique.

The given data is decoded after 65 iterations because here 66 fixed iterations are taken. It will check the errors in the first 3 iterations only if there is no error the output will decode properly. The input given to the LDPC decoder is “898925385058615299” (unsigned decimal).The decoder decodes the given input properly and

the output which is same as input with no errors. The simulation result for decoder output without errors is as shown in Fig 4.2.

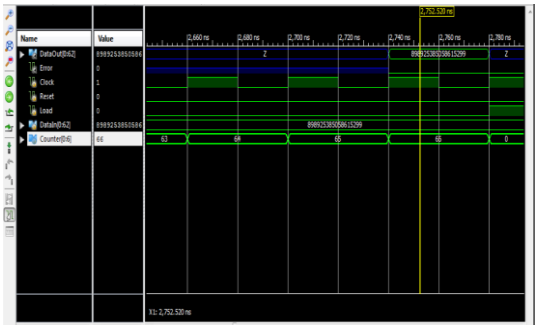

Fig 4.2 Simulation result for decoder output without error

In the simulation window, output represented by blue colour shows the pre output status till 65 iterations and once 65 iterations are done, data gets decoded in the 66 iteration and output appears in green colour as in the simulation window. The counter represents the number of iterations. In the window if the line representing the error contains '0' means there is no error or else if it is '1' the error exist.

If an error is found while decoding, the decoder will not decode the data correctly. The output will not be same as input. The input given to the LDPC decoder is "4440514323093127183". While decoding an error is found. So the decoder will not decode the data correctly and we get an error output i.e. "4440514323093127182". The simulation result for decoder output with error is as shown in Fig 4.3.

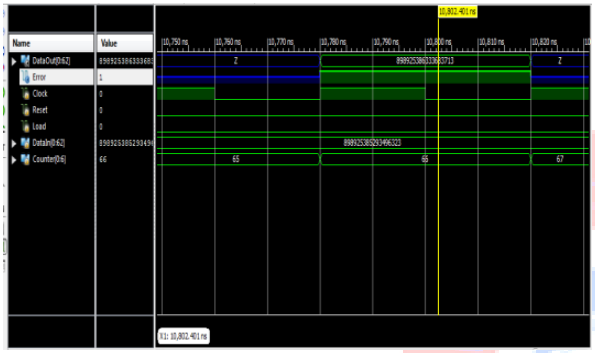

Fig 4.3 Simulation result for decoder output with error

Table 4.1 Comparison between conventional and proposed decoder

	Conventional decoder	Proposed decoder
No of slice LUTs used	8799	176
Timing summary	9.664ns	3.30ns

From the comparison Table 4.1 the proposed decoder uses the less number of LUT slice i.e 176 and requires only 3.30ns time which is very less compared to the conventional decoder. Hence it requires less power of the CPU. By this analysis the conclusion is made that the proposed decoder architecture is energy efficient.

5.CONCLUSION & FUTURE SCOPE

Conclusion

The EG-LDPC decoder with less complexity and increased decoding speed is implemented and the errors also successfully detected if exists. The detection of errors throughout the first iterations of serial one step Majority Logic Decoding of EG-LDPC codes has been done. The objective was to reduce the decoding time by halting the decoding process when simply no errors are detected in the first few iterations. The simulation outcomes show that all tested combinations of errors affecting around four bits are detected in the first three iterations of decoding.

The simulation results presented suggest that all errors affecting three and four bits would be detected in the first three iterations. For errors affecting a larger number of bits, there is a small probability of not being detected in those iterations. For large word sizes, the probabilities are sufficiently small to be acceptable in many applications.

According to the synthesis report obtained for the proposed system the time required for decoding process is "3.30ns" which is very less compared to conventional decoder and the number of LUTs used is "176" for the proposed system, which is also less compared to conventional decoder. By this analysis conclusion is made that the proposed system uses very less number of LUTs so the memory access for the system is reduced and the decoding time is also less so the complexity of the decoder is reduced hence it requires less power of the CPU so the proposed layer decoding architecture is Energy efficient.

Future Scope

Future work on decoding can be done for a larger choice of word lengths and error correction capabilities. Future work also includes extending the theoretical analysis for the cases of three and four errors. More generally, determining the necessary number of iterations to detect errors affecting certain number of bits seems to become an interesting problem. A general solution to that problem would enable a fine-grained trade off between decoding time and error discovery capability.

6. REFERENCES

[1] R. G. Gallager, Low-Density Parity-Check Codes. Cambridge, MA: MIT Press, 1963. Available at http://justice.mit.edu/people/gallager.html.

[2] V. Zyablov and M. Pinsker, "Estimation of the error-correction complexity of Gallager low-density codes," Probl. Pered. Inform., vol. 11, pp. 23–26, Jan. 1975.

[3] G. A. Margulis, "Explicit construction of graphs without short cyclesand low density codes," Combinatorica, vol. 2, no. 1, 1982, pp. 71–78.

[4] R. Tanner, "Arecursive approach to lowcomplexity codes," IEEE Trans. Inform. Theory, vol. IT-27, pp. 533–547, Sept. 1981.

[5] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," Electron. Lett., vol. 32, pp. 1645–1646, Aug. 1996.

[6] N. Wiberg, "Codes and decoding on general graphs," Dissertation no. 440, Dept. Elect. Eng. Linköping Univ., Linköping , Sweden, 1996.

[7] N. Sourlas, "Spin-glass models as error-correcting codes," Nature_journal, , pp. 693–695, 1989.

[8] I. Kanter and D. Saad, "Error-correcting codes that nearly saturate shannon's bound," Physical Review Letter., vol. 83, pp. 2660–2663, 1999.

[9] M. Luby, M. Mitzenmacher, A. Shokrollahi, D. Spielman, and V. Stemann, "Practical loss-resilient codes," in Proc. 29th Annual ACM Symp. Theory of Computing, 1997,

[10] M. Luby,M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Analysis of low density codes and improved designs using irregular graphs," in Proc. 30th Annu. ACM Symp. Theory of Computing, 1998, pp. 249–258.

[11] Kiran K. Gunnam, Gwan S. Choi, and Mark B. Yeary 2007 "A Parallel VLSI Architecture for Layered Decoding for Array LDPC Codes".

[12] Jun Lin IJin Sha', Zhongfeng Wang2 and Li u' "An improved min-sum based column-layered decoding algorithm for Ldpc codes"-2009

[13] Zhiqiang Cui1, Zhongfeng Wang2, Senior Member, IEEE, and Xinmiao Zhang3 "Reduced-Complexity Column-Layered Decoding and Implementation for LDPC Codes" 2010.

[14] Mohammad M. Mansour, Member, IEEE, and Naresh R. Shanbhag, Fellow, IEEE "A 640-Mb/s 2048-Bit Programmable LDPC Decoder Chip" 2006.

[15] 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" Jan 2013.