THE INTERNATIONAL JOURNAL OF SCIENCE & TECHNOLEDGE

Fault Detection in Low Density Parity Check (LDPC) Codes

S. Deepa
PG SCHOLAR, Department of ECE, Excel Engineering College, Komarapalayam, India
T. Mohankumar
M. E., Assistant Professor, Department of ECE, Excel Engineering College, Komarapalayam, India

Abstract:
Design of less decoding time of system is one of the largest areas of research in VLSI system design. The resent technology 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 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 EGLDPC codes. Extensive simulation results are given to accurately estimate the probability of error detection for different code sizes and numbers of errors

Key words: *Error correction codes, Euclidean geometry low-density parity check (EG-LDPC) codes, majority logic decoding, memory*

1. Introduction

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. 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 were 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 for the difference set of low density parity check (DSLDPC) codes. Low-Density-Parity-Check (LDPC) codes. Error correction code is a technique used for controlling errors in data transmission over unrealizable in communication, errors can occur due to a lot of reasons: noisy channel, scratches on CD or DVD, power surge in electronic circuit. To prevent soft errors from causing data corruption, memories are typically protected with Error Correction Codes (ECCs). Some types of ECCs are Single Error Correction (SEC) Codes that can correct one error in a memory word are commonly used. Parity allows the detection of all single-bit errors (actually, any odd number of wrong bits).

2. Majority Logic Decoding

The One step (MLD) majority logic decoding can be implemented serially using the scheme in which corresponds to the decoder for the EG-LDPC code with First the data block is loaded into the registers. Then the check equations are computed and if a majority of them has a value of one, the last bit is inverted. Then all bits are cyclically shifted. This set of operations constitutes a single iteration: after iterations, the bits are in the same position in which they were loaded. In the process, each bit may be corrected only once. As in Figure 1, the decoding circuitry is simple, but it requires a long decoding time if N is large. If errors can be detected in the first few iterations of MLD, then whenever no errors are detected in those iterations, the decoding can be stopped without completing the rest of the iterations.

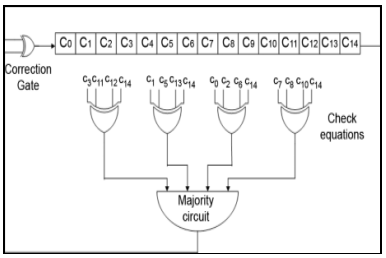

Figure 1: majority logic decoder EG-LPDC code

In the first iteration, errors will be detected when at least one of the check equations is affected by an odd number of bits in error. In the second iteration, as bits are cyclically shifted by one position, errors will affect other equations such that some errors undetected in the first iteration will be detected. As iterations advance, all detectable errors will eventually be detected One step MLD can be implemented serially which corresponds to the decoder for the EG-LDPC code with N=15. First the data block is loaded into the register. Then the check equations are computed and if a majority of them has a value of one, the last bit is inverted. Then all bits are cyclically shifted. This set of operations constitutes a single iteration, after each iteration the bits are in the same position in which they were loaded. In the process, each bit may be corrected only once. As can be seen, the decoding circuitry is simple, but it requires a long decoding time if N is large.

The check equations must have the following properties

- All equations include the variable whose value is stored in the last register (the one marked as c14).
- The rest of the registers are included in at most one of the check equations.

The DS-LDPC codes most errors can be detected in the first three iterations of MLD. Based on simulation results and on a theoretical proof for the case of two errors, the following hypothesis was made. Given a word read from a memory protected with DS-LDPC codes and affected by up to five bit-flips, all errors can be detected in only three decoding cycles The method proposed has been applied to the class of one step MLD EG-LDPC codes. To present the results, the conclusions are presented first in terms of a hypothesis that is then validated by simulation and also partially by a theoretical analysis presented in the appendix. The results obtained can be summarized in the following hypothesis Given a word read from a memory protected with one step MLD EG-LDPC codes, and affected by up to four bit-flips, all errors can be detected in only three decoding cycles. For codes with small words and affected by a small number of bit flips, it is practical to generate and check all possible error combinations. As the code size grows and the number of bit flips increases, it is no longer feasible to exhaustively test all possible combinations. Therefore the simulations are done in two ways, by exhaustively checking all error combinations when it is feasible and by checking randomly generated combinations in the rest of the cases.

N	K	J	tML
15	7	4	2
63	37	8	4
255	175	16	8
1023	781	32	16

Table 1: One step MLD EGLPDC codes

The parameters for some of these codes are given in Table 1 where N is the block size, K the number of information bits, J the number of MLD check equations and tML the number of errors that the code can correct using one step MLD. Therefore the simulations are done in two ways, by exhaustively checking all error combinations when it is feasible and by checking randomly generated combinations in the rest of the cases.

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

Table 2: Undetected Error in Exhaustive Checking

The results for the exhaustive checks are shown in Table 2. These results prove the hypothesis for the codes with smaller word size (15 and 63). For N=255 up to three errors have been exhaustively tested while for N=1023 only single and double error combinations have been exhaustively tested. The results for errors affecting more than four bits are shown in Table 3, since for

errors affecting up to four bits there were no undetected errors. It can be observed that for errors affecting more than four bits there is a small number of error combinations that will not be detected in the first three iterations.
This number decreases with word size and also with the number of errors. The decrease with the word size can be explained as follows: the larger the word size, the larger the number of MLD check equations (see Table 3) and therefore it is more unlikely that errors occur in the same equation

N	5 Error	6 Error	7 Error	8 Error	9 Error	10 Error	11 Error	12 Error
63	5672	5422	1079	1174	823	817	537	549
255	23	10	0	0	0	1	0	0
1023	0	1	0	0	0	0	0	0

Table 3: Undetected Errors with One Billion Random Error Combinations

3. Majority Logic Decoder and Detector (MLDD)

A novel version of the ML decoder for improving performance with reference to the original ML decoder, the proposed ML Detector/Decoder (MLDD) has been implemented using the Euclidean geometry Low-Density Parity Check (EG-LDPC) codes. The EG-LDPC codes are based on the structure of Euclidean geometries over a Galois field. Among EG-LDPC codes there is a subclass of codes that is one step Majority Logic Decodable (MLD). The memory system schematic of proposed MLDD is shown in Figure 2.

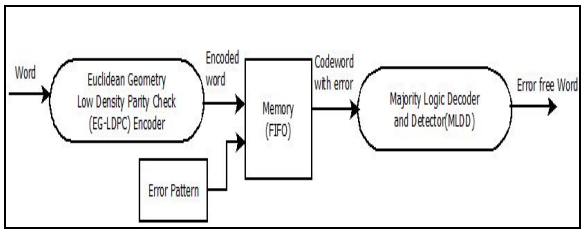

Figure 2: majority logic decoder and detector (MLDD)

The proof of the hypothesis that all error will be detected in three cycles is very complex from the mathematical point of view. It is practical to generate and check all possible error combinations for codes with small words and affected by a small number of bit flips. When the size of code and the number of bit flips increases, it is difficult to exhaustively test all possible combinations. Therefore the simulations are done in two ways, the error combinations are exhaustively checked when it is feasible and in the rest of the cases the combinations are checked randomly. Since it is convenient to first describe the chosen design and also for simplicity, let us assume that the hypothesis is true, that only three cycles are needed to detect all errors affecting up to four bits in EG LDPC Codes.

4. Majority Logic Decoder

The ML decoder is powerful and simple decoder, which has the capability of correcting multiple random bit-flips depending on the number of parity check equations.

It consists of four parts: a cyclic shift registers, an XOR matrix, a majority gate, an XOR for correcting the codeword bit under decoding. The cyclic shift register is initially stored with the input signal x and shifted through all the taps. The results {Bj} of the check sum equations from the XOR matrix is calculated from the intermediate values in each tap.

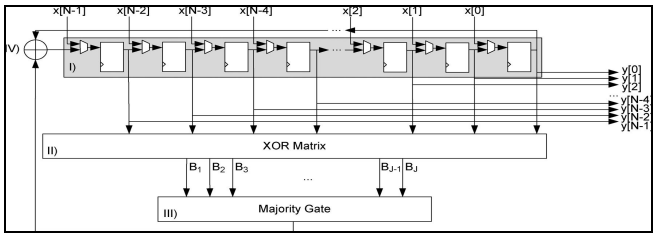

Figure 3: majority decode

In the Nth cycle, the result would reach the final tap, producing the output signal, which is the decoded version of input. The input x might correspond to wrong data corrupted by a soft error or SEUs. The decoder is designed to handle this situation as follows. From the parity check sum equations hardwired in the XOR matrix the decoding starts at the very next moment after the codeword x are loaded into the cyclic shift register. The linear sum outputs {Bj} is then forwarded to the majority logic circuit which determines the correctness of the bit under decoding. If the majority of the Bj 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. The circuit implementing a serial one-step majority logic corrector for (15, 7, 5) EG-LDPC code is shown in Figure 2.

5. Majority Logic Decoder and Detector (MLDD)

In general, the proposed version uses the same decoding algorithm as the one in plain ML decoder version. The advantage is that, proposed method stops intermediately in the third cycle when there is no error in data read, the whole code word size of N. The XOR matrix is evaluated for the first three cycles of the decoding process, and when all the outputs {Bj} is "0,"the codeword is determined to be error-free and forwarded directly to the output. the whole decoding process to eliminate the errors if the {Bj} contain at least a "1" in any of the three cycles. The input x might correspond to wrong data corrupted by a soft error or SEUs. The decoder is designed to handle this situation as follows. From the parity check sum equations hardwired in the XOR matrix the decoding starts at the very next moment after the codeword x are loaded into the cyclic shift register. The linear sum outputs {Bj} is then forwarded to the majority logic circuit which determines the correctness of the bit under decoding

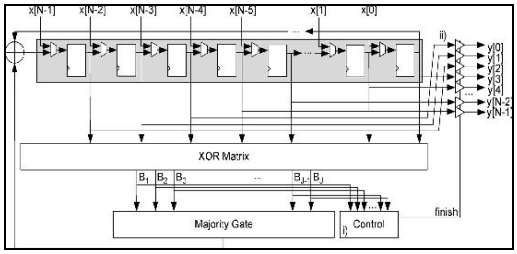

Figure 4: MLDD structure

If the majority of the Bj 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.A detailed schematic of the proposed design for 15 bit code word is The figure shows the basic ML decoder with a 15-tap shift register, an XOR array to calculate the orthogonal parity check sums and a majority logic circuit which will decide whether the current bit under decoding is erroneous and the need for its inversion. The plain ML decoder shown in Figure 4 is also having the same schematic structure up to this stage. The algorithm for MLDD method is to detect the error and correct. The flow chart for MLDD method as shown in Figure.5 Iteration from 1 to 3 will be performed for detection of the error in the codeword.

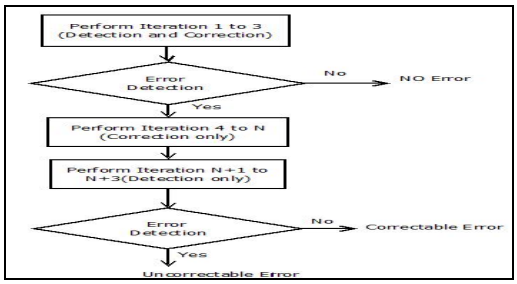

Figure 5: flow chart of MLDD method

If the codeword is error free then output will be obtained in three iteration. Iteration from 4 to N will be preformed to correct the error in the codeword. Performances N+1 to N+3 iteration are to detect the error in codeword. If the error is detected then it will corrected. Otherwise retransmission will be processed.

6. Result

Simulation result of error detection logic is show in Figure 6. Then the simulation result presented suggest that all errors affecting three and four bits would be detected in the first three iterations for errors affecting a larger of bit 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.

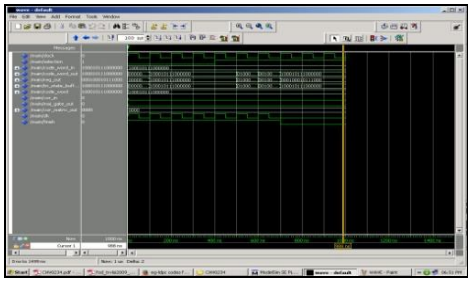

Figure 6: Error Detection

7. Summary and Conclusion

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.

Future work includes extending the theoretical analysis to the cases of three and four errors detected. More generally, determining the required number of iterations to detect errors affecting a given number of bits seems to be an interesting problem. A general solution to that problem would enable a fine-grained tradeoff between decoding time and error detection capability.

8. References

1. R.C. Baumann, (2005) "Radiation induced soft errors in semiconductor Technologies," IEEE Transm .Reliab. vol.5, no. 3,p p. 301316.
2. M.A.Bajura, Y.Boulghassoul, (2007) "Models and algorithmic limits for an ECC-based approachto hardening sub-100-nm SRAMs," IEEE Trans. Nucl. Sci., vol.54, no. 4, pp. 935–945.
3. S. Ghosh and P. D. Lincoln,(2007) "Dynamic low-density parity check codesfor fault tolerant nano scale memory," presented at the Foundations Nanosci. (FNANO), Snowbird, Utah.
4. S. Lin and D. J. Costello (2004) Error Control Coding, 2nd ed. Engle woo Cliffs, NJ: Prentice-Hall.
5. S. Liu, P. Reviriego, and J. Maestro,(2012) "Efficient majority logic fault Detection with difference-set codes for memory applications," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 20, no. 1, pp. 148–156.
6. H. Naeimi and A. DeHon, (2007) "Fault secure encoder and decoder for memory applications," in Proc. IEEE Int. Symp. Defect Fault Toler.VLSI Syst.,pp. 409–417.
7. H. Naeimi and A. DeHon, (2009) "Fault secure encoder and decoder for nano memory applications," IEEE Trans. Very Large Scale Integr.(VLSI) Syst., vol. 17, no. 4,pp. 473–486.
8. R. Naseer and J. Draper, (2008) "DEC ECC design to improve memory reliability in sub-100 nm technologies," Proc. IEEE ICECS, pp. 586–589.
9. Pedro Reviriego, Juan A. Maestro, And Mark F.Flanaga, (2013). "Error Detection in Majority Logic Decoding of Euclidean Geometry Low Density Parity Check (EG-LDPC) Codes", IEEE Transitions On Vlsi. Vol.21, No.1.
10. H. Tang, J. Xu, S. Lin, and K. A. S. Abdel-Ghaffar,(2005) "Codes on finite geometries," IEEE Trans. Inf. Theory, vol. 51, no. 2, pp. 572–596.
11. B. Vasic and S. K. Chilappagari,(2007) "An information theoretical frame Work for analysis and design of nanoscale fault-tolerant memoriesbased on Low density parity-check codes," IEEE Trans. Circuits Syst.IReg.Papers,vol54,no.11,pp.2438–2446.