Fault Secure Memory Design using Difference Set Codes

K.Manikandan
M .Tech VLSI Design Scholar,
Dept. of ECE., Kalasalingam University,
Krishnankoil, virudhunagar Dist, Tamil Nadu, India

G.Thiruselvi
Assistant Professor
Dept. of ECE., Kalasalingam University,
Krishnankoil, virudhunagar Dist, Tamil Nadu, India

ABSTRACT

Modified decoding algorithms for DS codes are proposed that, in addition to error correction, provide error detection when the number of correctable bit errors is exceeded by one. This combined error detection and correction capability of the modified decoder are provide to prevent soft errors from causing data corruption, memories are typically protected with error correction codes (ECCs). Memory applications require low latency encoders and decoders. These codes allow us to design a fault tolerant error-detector unit that detects any error in the received code-vector despite having faults in the detector circuitry. The fault secure detector unit to check the output vector of the encoder and corrector circuitry, and if there is any error in the output of either of these units, that unit has to redo the operation to generate the correct output vector. Using this detect-and-repeat technique, correct potential transient errors in the encoder or corrector output and provide fault tolerant memory system with fault-tolerant supporting circuitry.

Keywords

Error correction codes, low-density parity check (LDPC), memory, majority logic.

1. INTRODUCTION

The fault-secure memory 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.

Memory system design that can tolerate errors in any part of the system, including the storage unit and encoder and corrector circuits using the fault-secure detector. Any single error in the encoder or corrector circuitry can at most corrupt a single codeword bit this by preventing logic sharing between the circuits producing each codeword bit or information bit in the encoder and the corrector respectively .This fault-secure detector can verify the correctness of the encoder and corrector operation. A proposed reliable memory system is shown in Fig. 1 and is described in the following. The information bits are fed into the encoder to encode the information vector, and 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. During memory access operation, the stored codeword's will be accessed from the memory unit.

All the memory words pass through the Additional Error Detection MLDD and any potential error [5] in the memory words will be corrected. Similar to the encoder unit, a fault-secure detector monitors the operation of the corrector unit.

When using the normal decoder that requires 15 iterations (for codeword length 15) to decode a word .This results in a large latency that in a memory application would increase the access time. Additionally, the number of iterations would increase linearly with the length of the coded word. The latency of the decoder can be reduced by using a modified MLDD implementation

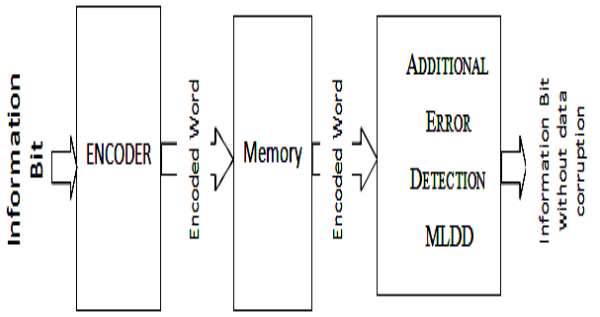


Fig . (1) Memory system with Modified MLDD

2. MlDD

In general, the decoding algorithm is still the same as the one in the plain ML decoder version [1]. The difference is that, instead of decoding all codeword bits by processing the ML decoding during cycles, the proposed method stops intermediately in the third cycle, as illustrated in Fig. 2. If in the first three cycles of the decoding process, the evaluation of the XOR matrix for all is "0," the codeword is determined to be error-free and forwarded directly to the output.If they contain in any of the three cycles at least a "1," the proposed method would continue the whole decoding process in order to eliminate the errors. The control unit manages the detection process. It uses a counter that counts up to three, which distinguishes the first three iterations of the ML decoding

3. MODIFIED MLDD

The Modified MLDD algorithm performs the decoding as in the MLDD with some modifications The Modified MLDD algorithm is illustrated in Fig.4. Modified MLDD algorithm requires additional logic compared to the MLDD algorithm. The corrections are performed during the first n iterations. A counter is used to determine if there have been more than t errors in those iterations and based on the result the corrected or the original register is send to the output. If the majority gate detect any error in codeword the iterations take place

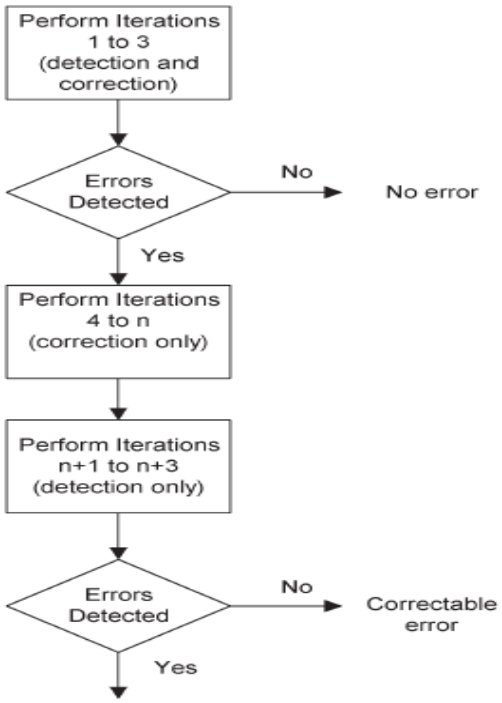


Fig. 2.Flow diagram of the MLDD algorithm

depends upon the length of the codeword's. The process stops after three iteration if the codeword length less than ten and six iteration for codeword less than twenty, if the codeword greater than 20 then nine iteration performed

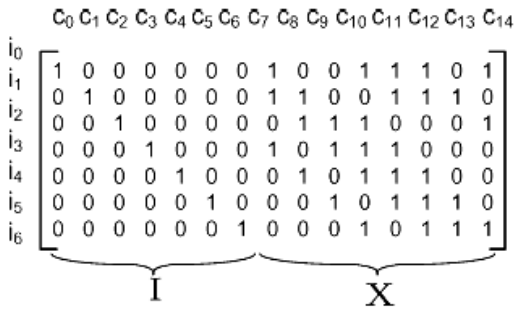


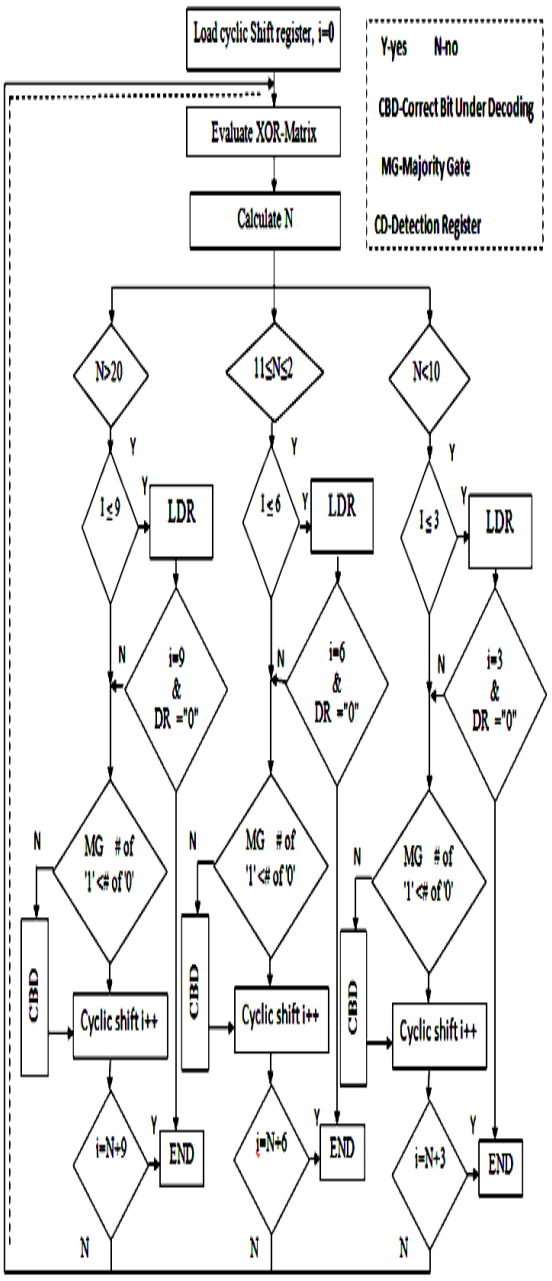
Fig. 3 Generator matrix for the (15, 7, 5) EG-LDPC

Modified MLDD detect more than five bit-flips and high efficiency compare to Majority logic decoder. The control unit manages the detection process. It uses a counter that counts up to three, which distinguishes the first three iterations of the ML decoding. 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

4. Encoder Design

Encoder design [3] here used to encodes a k-bit information vector i into n-bit codeword c In this figure i = (i0... i6) is the information vector and will be copied to (c0... c6) bits of the encoded vector, c.



An n-bit codeword, which encodes a -bit information vector, is generated by multiplying the k-bit information vector with a k × n bit generator matrix.

The encoded vector consists of information bits followed by parity bits, where each parity bit is simply an inner product of information vector and a column of x , from G=[I : X]

Fig. 3 shows the systematic generator matrix to generate (15, 7, 5) code word. The encoded vector consists of information bits followed by parity bits, where each parity bit is simply an inner product of information vector and a column of , from . Fig.6 shows the encoder circuit to compute the parity bits of the (15, 7, 5) EG-LDPC code. In this Fig. 6 i = (i0... i6) is the information vector and will be copied to (c0... c6) bits of the encoded vector, c by the information vector and will be copied to bits of the encoded vector, , and the rest of encoded

vector, the parity bits, are linear sums (XOR) of the information bits. EG-LDPC codes are not systematic and the information bits must be decoded from the encoded vector, which is not desirable for fault-tolerant approach

5. Corrector

One-step majority logic correction [3] is the procedure that identifies the correct value of an each bit in the codeword directly from the received codeword the majority value indicates the correctness of the code-bit under consideration; if the majority value is 1, the bit is inverted, otherwise it is kept unchanged.

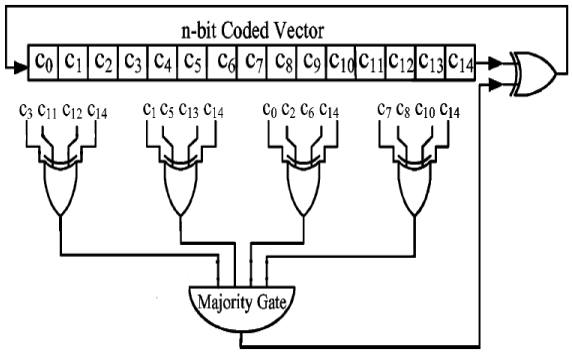


Fig.5 Serial one-step majority logic corrector for 15-bit codeword's

The circuit implementing serial one-step majority logic Corrector for (15, 7) codeword is shown in Fig. 5. One-step majority-logic correction is a fast and relatively compact error-correcting technique

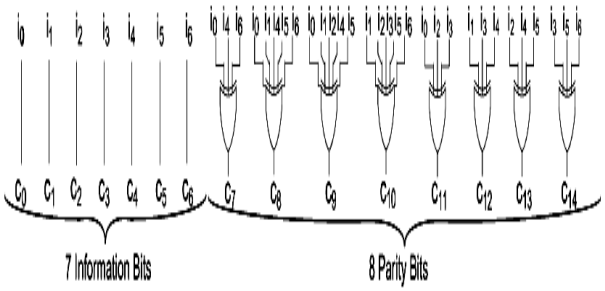


Fig. 6 Encoder Circuit

A compact implementation for the majority gate is by using Sorting Networks[3] .The binary Sorting Networks is used to do the sort operation of the second step efficiently. An -input sorting network is the structure that sorts a set of bits, using 2-bit sorter building blocks. Fig. 7 (a) 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 see Fig. 7 (b) 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.

6. Conclusion

In this paper, 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.

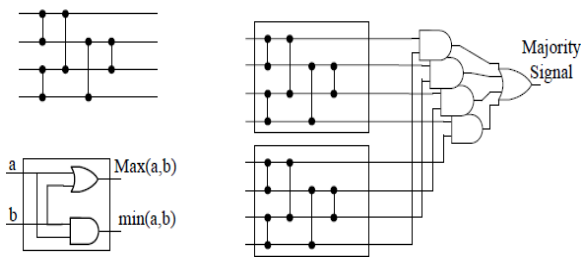


Fig 7 (a) Four-input sorting network; each vertical line shows a one-input comparator. (b) One comparator structure. (C) Eight-input majority gate using sorting network

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.

7. REFERENCES

[1] Efficient Majority Logic Fault Detection With Difference-Set Codes for Memory Applications Shih-Fu Liu, Pedro Reviriego, Member, IEEE, and Juan Antonio Maestro, Member, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 20, NO. 1, JANUARY 2012

[2] P. Ankolekar, S. Rosner, R. Isaac, and J. Bredow, "Multi-bit error correction methods for latency-contrained flash memory systems," IEEE Trans. Device Mater. Reliabil., vol. 10, no. 1, pp. 33–39, Mar. 2010.

[3] 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.

[4] 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.

[5] 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.

[6] S. Lin and D. J. Costello, Error Control Coding, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2004.

[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] Heng Tang et al. Codes on finite geometries. IEEE Transaction on Information Theory, 51(2):572–596, 2005.

[9] G. C. Cardarilli et al. Concurrent error detection in reed-solomon encoders and decoders. IEEE Trans. VLSI, 15:842–826, 2007.

[10] Shu Lin and Daniel J. Costello. Error Control Coding. Prentice Hall, second edition, 2004.

[11] S. Hareland et al. Impact of CMOS process scaling and SOI on the soft error rates of logic processes. In Procedings of Symposium on VLSI Digest of Technology Papers, pages 73–74, 2001.

[12] R. Horan et al. Idempotents, mattson-solomon polynomials and binary ldpc codes. IEE Proceedings of Communication, 153(2):256–262, 2006.

[13] J. Kim et al. Error rate in current-controled logic processors with shot noise. Fluctuation and Noise Letters, 4(1):83–86, 2004.

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