EFFICIENT FAULT DETECTION CODES

***M. Palaniappan**
****B. Manikandan**

ABSTRACT

The technology advancements in scaling-smaller dimensions, higher integration densities, and lower operating voltages-has lead to reduction of reliability not only in extreme radiation environments like spacecraft and avionics , but also in terrestrial environments. SRAM memory failure rates are increasing significantly, thereby raising a major reliability problem for many applications. This paper presents an error-detection method for difference-set cyclic codes with majority logic decoding. Majority logic decodable codes are suitable for memory applications due to their capability to correct a large number of errors. However, they require a large decoding time that impacts memory performance. The proposed fault-detection method significantly reduces memory access time when there is no error in the data read. The technique uses the majority logic decoder itself to detect failures, which makes the area overhead minimal and keeps the extra power consumption low. This technique will tend to correct burst errors of any length.

Index Terms : Error correction Schemes (ECC), Majority Logic Decoding (MLD), Low Density Parity Check Codes (LDPC).

***Student, PG-Final year, University College of Engineering, BIT Campus, Trichy**
**** Assistant Professor, University College of Engineering, BIT Campus, Trichy**

I.INTRODUCTION

The general idea for achieving error detection and correction is to add some redundancy (i.e., some extra data) to a message, which receivers can use to check consistency of the delivered message, and to recover data determined to be corrupted. Error-detection and correction schemes can be either systematic or non-systematic: In a systematic scheme, the transmitter sends the original data, and attaches a fixed number of *check bits* (or *parity data*), which are derived from the data bits by some deterministic algorithm. If only error detection is required, a receiver can simply apply the same algorithm to the received data bits and compare its output with the received check bits; if the values do not match, an error has occurred at some point during the transmission. In a system that uses a non-systematic code, the original message is transformed into an encoded message that has at least as many bits as the original message. Error-detecting and correcting codes can be generally distinguished between random-error-detecting / correcting and burst-error-detecting/correcting. Some codes can also be suitable for a mixture of random errors and burst errors.

The various error correction schemes available are Hash function, Repetition codes, Parity bits, Check sums, Cryptographic Hash function, Cyclic redundancy Checks and etc.,.

TMR, a error correcting mechanism having complexity overhead three times plus the complexity of the majority voter and consumes more power. ECC codes are the best way to mitigate memory soft errors. The usual multi error correction codes, such as Reed–Solomon (RS) or Bose–Chaudhuri–Hocquenghem (BCH) are not suitable for this task. The reason for this is that they use more sophisticated decoding algorithms. Cyclic block codes have been identified as good candidates, due to their property of being majority logic (ML) decodable and it is very simple to implement. But the drawback of ML decoding is that, for a coded word of N -bits, it takes N cycles in the decoding process, posing a big impact on system performance. The simplest way to implement a fault detector for an ECC is by calculating the

syndrome, but this generally implies adding another very complex functional unit. Among the ECC codes that meet the requirements of higher error correction capability and low decoding complexity, cyclic block codes have been identified as good candidates, due to their property of being majority logic (ML) decodable [7], [8].

In this paper, we will focus on one specific type of LDPC codes, namely the difference-set cyclic codes (DSCCs), which is widely used in the Japanese tele text system or FM multiplex broadcasting systems [12]-[14]. The main reason for using ML decoding is that it is very simple to implement and thus it is very practical and has low complexity. The drawback of ML decoding is that, for a coded word of N-bits, it takes N cycles in the decoding process. One way of coping with this problem is to implement parallel encoders and decoders. This solution would enormously increase the complexity and, therefore, the power consumption. As most of the memory reading accesses will have no errors, the decoder is most of the time working for no reason. This has motivated the use of a fault detector module [11] that checks if the codeword contains an error and then triggers the correction mechanism accordingly. In this case, only the faulty code words need correction, and therefore the average read memory access is speeded up, at the expense of an increase in hardware cost and power consumption. A similar proposal has been presented in [15] for the case of flash memories. This paper explores the idea of using the ML decoder circuitry as a fault detector so that read operations are accelerated with almost no additional hardware cost. The results show that the properties of DSCC-LDPC enable efficient fault detection.

II. EXISTING ERROR CORRECTION METHODOLOGIES

A. Plain ML Decoder

The ML decoder is a simple and powerful decoder, capable of correcting multiple random bit-flips depending on the number of parity check equations. It consists of four parts: 1) a cyclic

shift register; 2) an XOR matrix; 3) a majority gate; and 4) an XOR for correcting the codeword bit under decoding, as illustrated in Fig. 2. The input signal is initially stored into the cyclic shift register and shifted through all the taps. The intermediate values in each tap are then used to calculate the results of the check sum equations from the XOR matrix. In the cycle, the result has reached the final tap, producing the output signal (which is the decoded version of input). As stated before, input might correspond to wrong data corrupted by a soft error. To handle this situation, the decoder would behave as follows.

After the initial step, in which the codeword is loaded into the cyclic shift register, the decoding starts by calculating the parity check equations hardwired in the XOR matrix. The resulting sums are then forwarded to the majority gate for evaluating its correctness. If the number of 1's received in is greater than the number of 0's that would mean that the current bit under decoding is wrong and a signal to correct it would be triggered. Otherwise, the bit under decoding would be correct and no extra operations would be needed on it. In the next step, the content of the registers are rotated and the above procedure is repeated until all codeword bits have been processed. Finally, the parity check sums should be zero if the codeword has been correctly decoded. Further details on how this algorithm works can be found in [6]. The whole algorithm is depicted in Fig. 3. The previous algorithm needs as many cycles as the number of bits in the input signal, which is also the number of taps, in the decoder. This is a big impact on the performance of the system, depending on the size of the code. For example, for a codeword of 73 bits, the decoding would take 73 cycles, which would be excessive for most applications.

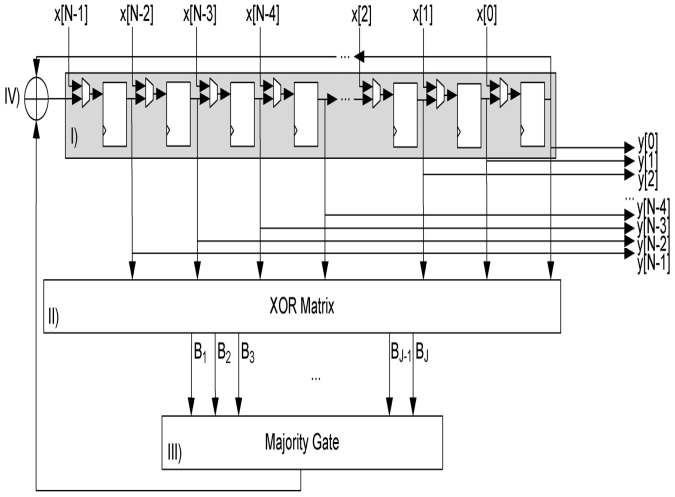


Fig.1 Schematic of plain ML decoder

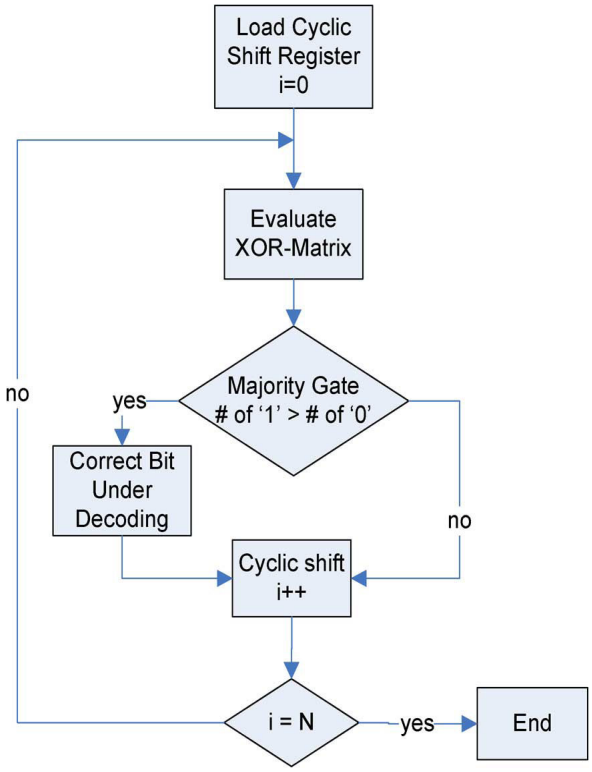


Fig.2 Flow diagram of MLD

B. Plain MLD with Syndrome Fault Detector (SFD)

In order to improve the decoder performance, alternative designs may be used. One possibility is to add a fault detector by calculating the syndrome, so that only faulty code words are decoded [11]. Since most of the code words will be error-free, no further correction will be needed, and therefore performance will not be affected. Although the implementation of an SFD reduces the average latency of the decoding process, it also adds complexity to the design (see Fig. 4). The SFD is an XOR matrix that calculates the syndrome based on the parity check matrix. Each parity bit results in a syndrome equation. Therefore, the complexity of the syndrome calculator increases with the size of the code.

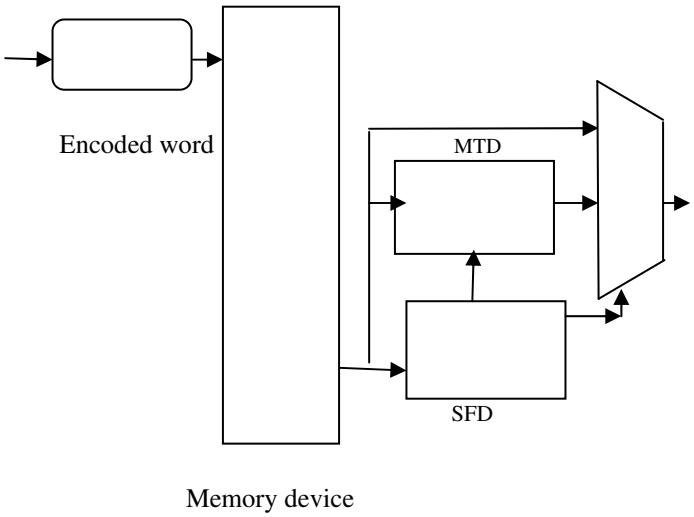


Fig.3 Plain ML decoder with SFD

A faulty code word is detected when at least one of the syndrome bits is "1." This triggers the MLD to start the decoding, as explained before. On the other hand, if the codeword is error-free, it is forwarded directly to the output, thus saving the correction cycles. In this way, the performance is improved in exchange of an additional module in the memory system: a matrix of XOR gates to resolve the parity check matrix, where each check bit results into a syndrome equation. This finally results in a quite complex module, with a large amount of additional hardware and power consumption in the system.

III. PROPOSED METHODOLOGY

This section presents a modified version of the ML decoder that improves the designs presented before. Starting from the original design of the ML decoder introduced in [8], the proposed ML detector/decoder (MLDD) has been implemented using the difference-set cyclic codes (DSCCs) [16]–[19]. This code is part of the LDPC codes, and, based on their attributes, they have the following properties:

- ability to correct large number of errors;
- sparse encoding,
- decoding and checking circuits synthesizable into simple hardware;
- modular encoder and decoder blocks that allow an efficient hardware implementation;
- systematic code structure for clean partition of information and code bits in the memory.

An important thing about the DSCC is that its systematical distribution allows the ML decoder to perform error detection in a simple way, using parity check sums (see [6] for more details). However, when multiple errors accumulate in a single word, this mechanism may misbehave, as explained in the following. In the simplest error situation, when there is a bit-flip in a codeword, the corresponding parity check sum will be "1," .However, in the case of the code word is affected by two bit-flips in bit 42 and bit 25, which participate in the same parity check equation. So, the check sum is zero as the parity does not change. Finally if there is three bit-flips which again are detected by the check sum (with a "1"). As a conclusion of these examples, any number of odd bit flips can be directly detected, producing a "1" in the corresponding equation.

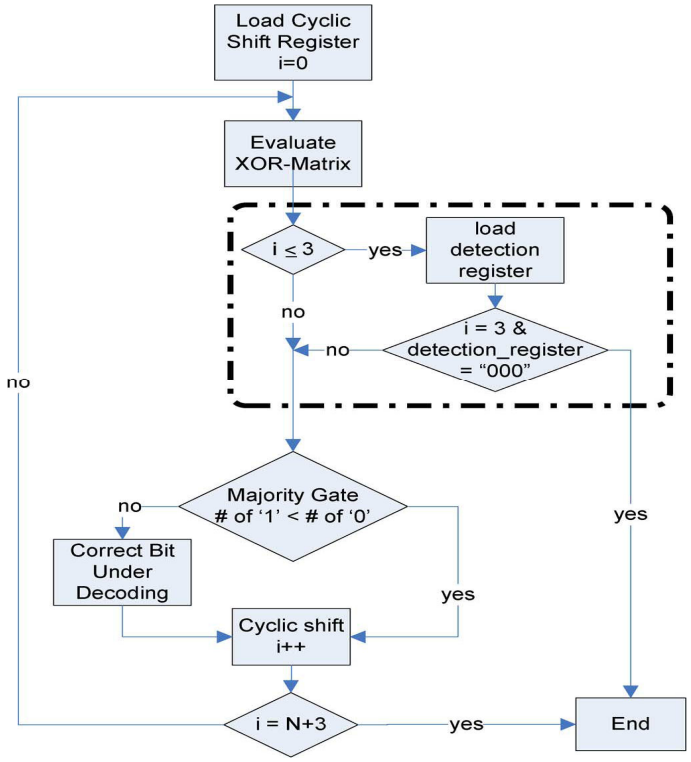


Fig.4 Flow diagram of proposed method

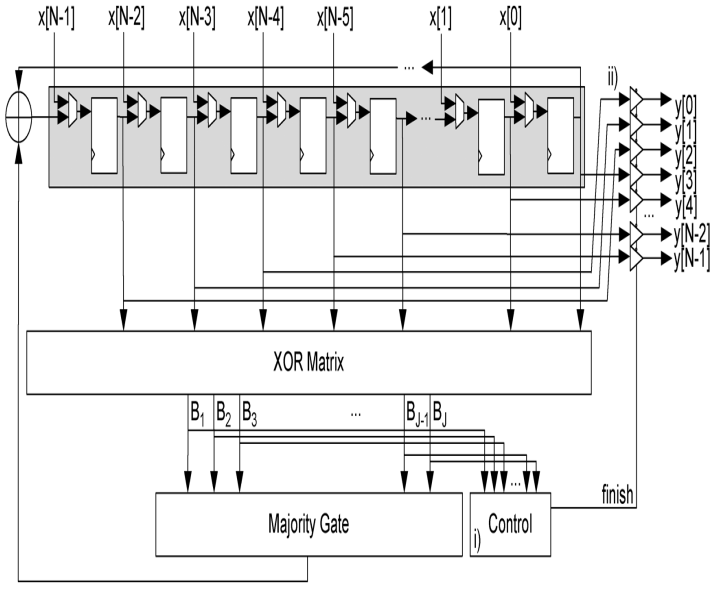


Fig.5 Schematic of MLDD

The problem is in those cases with an even numbers of bit-flips, where the parity check equation would not detect the error. In this situation, the use of a simple error detector based on parity check sums does not seem feasible, since it cannot handle “false negatives” (wrong data that is not detected).

However, the alternative would be to derive all data to the decoding process (i.e., to decode every single word that is read in order to check its correctness), as explained in previous sections, with a large performance overhead. Since performance is important for most applications, we have chosen an intermediate solution, which provides a good reliability with a small delay penalty for scenarios where up to five bit-flips may be expected (the impact of situations with more than five bit-flips will be analyzed in Section IV-A).

This proposal is one of the main contributions of this paper, and it is based on the following hypothesis:

Given a word read from a memory protected with DSCC codes, and affected by up to five bit-flips, all errors can be detected in only three decoding cycles. This is a huge improvement over the simpler case, where decoding cycles are needed to guarantee that errors are detected.

In general, the decoding algorithm is still the same as the one in the plain ML decoder version. 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. 6. 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 it contains 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. A detailed schematic of the proposed design is shown in Fig. 7. The figure shows the basic ML decoder with an -tap shift register, an XOR array to calculate the orthogonal parity check sums and a

majority gate for deciding if the current bit under decoding needs to be inverted. Those components are the same as the ones for the plain ML decoder shown in Fig. 2.

The additional hardware to perform the error detection is illustrated in Fig. 6 as: i) the control unit which triggers a finish flag when no errors are detected after the third cycle. The output tri state buffers are always in high impedance unless the control unit sends the finish signal so that the current values of the shift register are forwarded to the output. The control schematic is illustrated in Fig. 8. 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. In these first three iterations, the control unit evaluates the by combining them with the OR1 function. This value is fed into a three-stage shift register, which holds the results of the last three cycles.

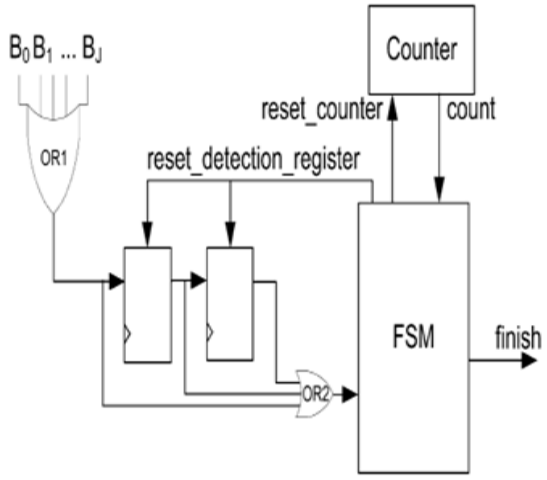
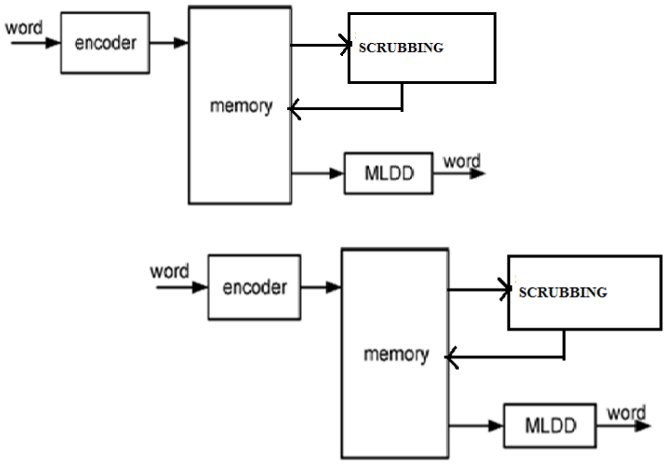


Fig.6 Control unit of MLDD

In the third cycle, the OR2 gate evaluates the content of the detection register. When the result is "0," the FSM sends out the finish signal indicating that the processed word is error-free. In the other case, if the result is "1," the ML decoding process runs until the end. This clearly provides a performance improvement respect to the traditional method. Most of the words would only take three cycles (five, if we consider the other two for input/output) and only

those with errors (which should be a minority) would need to perform the whole decoding process. The schematic for this memory system is very similar to the one in Fig. 1, adding the control logic in the MLDD module.



FAULT SECURE MEMORY SYSTEM

Scrubbing of memory enables the periodic error detection and correction procedure to be repeated, thereby preventing error propagation to unaffected locations. It also increases the probability of reading the data without errors, which takes only five cycles for MLDD decoding ensuring safety and timely information.

IV.SIMULATION RESULTS

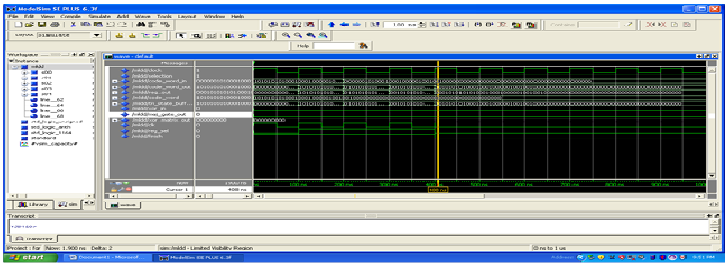


Fig.7 Model sim wave editor showing error detection within 3 cycles.

V. CONCLUSION

In this paper, a fault-detection mechanism, MLDD, has been presented based on ML decoding using the DSCCs. Exhaustive simulation test results show that the proposed technique is able to detect any pattern of up to five bit-flips in the first three cycles of the decoding process. This improves the performance of the design with respect to the traditional MLD approach. On the other hand, the MLDD error detector module has been designed 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 calculation (SFD). In addition, a scrubbing technique is included with the proposed MLDD scheme for preventing error propagation and reducing iterations needed for decoding.

VI. REFERENCES

[1] Shih Fu Liu, Pedro reviriego & Juan Antonio Maestro, "Efficient Logic Fault detection with Difference –Set cyclic codes for Memory Applications," *IEEE Trans. VLSI systems.*, vol.2, no. 1, Jan. 2012.

[2] 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.

[3] J. von Neumann " Probabilistic logics and synthesis of reliable organisms from unreliable components" *Automata Studies*, pp. 43–98, 1956.

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

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

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

[7] I. S. Reed, "A class of multiple-error-correcting codes and the decoding scheme" *IRE Trans. Inf. Theory*, vol. IT-4, pp. 38–49, 1954.

[8] J. L. Massey, *Threshold Decoding*. Cambridge, MA: MIT Press, 1963.

[9] S. Ghosh and P. D. Lincoln, " Low-density parity check codes for error correction in Nano scale memory " SRI Comput. Sci. Lab. Tech. Rep. CSL-0703, 2007.

[10] B. Vasic and S. K. Chilappagari, " An information theoretical framework for analysis and design of Nano scale fault-tolerant memories based on low-density parity-check codes " *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 11, pp. 2438–2446, Nov. 2007.

[11] H. Naeimi and A. DeHon, "Fault secure encoder and decoder for Nano Memory applications" *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 17, no. 4, pp. 473–486, Apr. 2009.

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

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

[14] O. Yamada, "Development of an error-correction method for data packet multiplexed with TV signals" *IEEE Trans. Commun.*, vol. COM-35, no. 1, pp. 21–31, Jan. 1987.

[15] 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.

[16] E. J.Weldon, Jr., " Difference-set cyclic codes" *Bell Syst. Tech. J.*, vol. 45, pp. 1045–1055, 1966.

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

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