International Journal of Scientific & Engineering Research, Volume 4, Issue 5, May-2013
ISSN 2229-5518

CORRECTIONS TO FAULT SECURE OF MAJORITY LOGIC DECODER AND DETECTOR FOR MEMORY APPLICATIONS

Viji.D
PG Scholar – Embedded Systems
Prist University, Thanjuvr - India

Mr.T.Sathees Kumar
AP/ECE
Prist University, Thanjuvr – India

Abstract : Nowadays, single event upsets (SEUs) altering digital circuits are becoming a bigger concern for memory 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.

Index Terms—Block codes, difference-set, error correction codes (ECCs), low-density parity check (LDPC), majority logic, memory.

INTRODUCTION

THE impact of technology scaling smaller dimensions, higher integration densities, and lower operating voltages has come to a level that reliability of memories is put into jeopardy, not only in extreme radiation environments like spacecraft and avionics electronics, but also at normal terrestrial environments. Especially, SRAM memory failure rates are increasing significantly, therefore posing a major reliability concern for many applications. Some commonly used mitigation techniques are:

• triple modular redundancy (TMR);
• error correction codes (ECCs).

TMR is a special case of the von Neumann method consisting of three versions of the design in parallel, with a majority voter selecting the correct output. As the method suggests, the complexity overhead would be three times plus the complexity of the majority voter and thus increasing the power consumption. For memories, it turned out that ECC codes are the best way to mitigate memory soft errors.

For terrestrial radiation environments where there is a low soft error rate (SER), codes like single error correction and double error detection (SEC–DED), are a good solution, due to their low encoding and decoding complexity. However, as a consequence of augmenting integration densities, there is an increase number of soft errors, which produces the need for higher error correction capabilities. 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, like complex algebraic (e.g., floating point operations or logarithms) decoders that can decode in fixed time, and simple graph decoders, that use iterative

algorithms (e.g., belief propagation). Both are very complex and increase computational costs.

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. A subgroup of the low-density parity check (LDPC) codes, which belongs to the family of the ML decodable codes, has been researched in. 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 teletext system or FM multiplex broadcasting systems. 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 -bits, it takes cycles in the decoding process, posing a big impact on system performance.

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 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 for the case of flash memories.

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. 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.

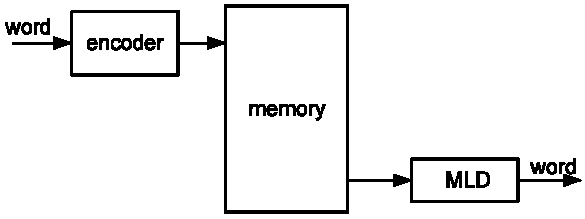

Fig: Memory system schematic with MLD.

EXISTENT MAJORITY LOGIC DECODER

MLD is based on a number of parity check equations which are orthogonal to each other, so that, at each iteration, each codeword bit only participates in one parity check equation, except the very first bit which contributes to all equations. For this reason, the majority result of these parity check equations decide the correctness of the current bit under decoding. MLD was first mentioned in for the Reed–Müller codes. Then, it was extended and generalized in for all types of systematic linear block codes that can be totally orthogonalized on each codeword bit.

A generic schematic of a memory system is depicted in Figure for the usage of an ML decoder. Initially, the data words are encoded and then stored in the memory. When the memory is read, the codeword is then fed through the ML decoder 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.

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.

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 this unit. The whole algorithm is depicted in Figure 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.

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. 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.

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. faulty codeword 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.

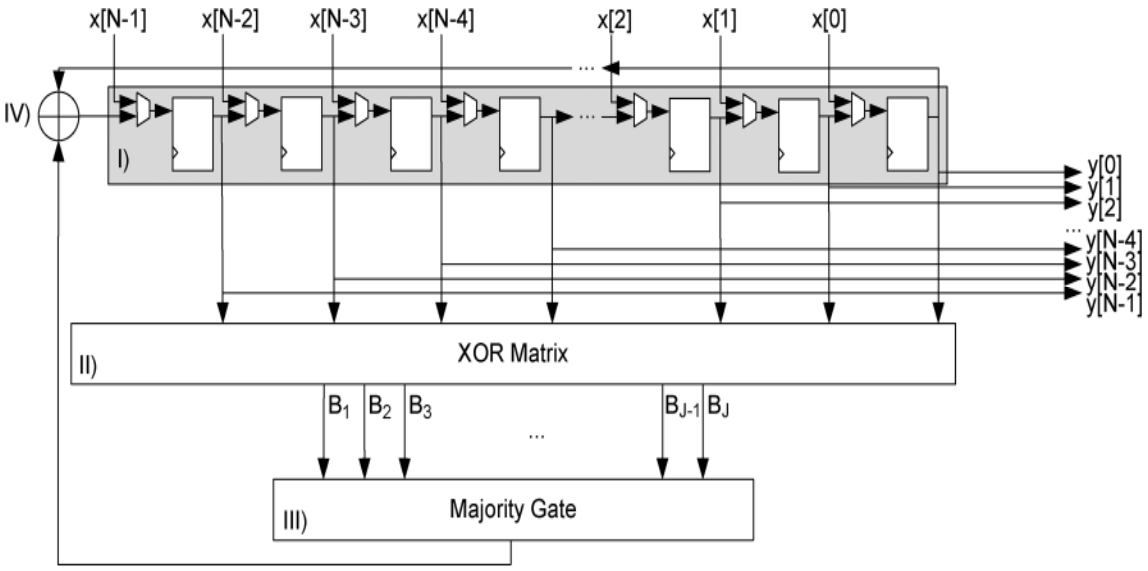

Fig: Schematic of an ML decoder. I) cyclic shift register. II) XOR matrix. III) Majority gate. IV) XOR for correction

.

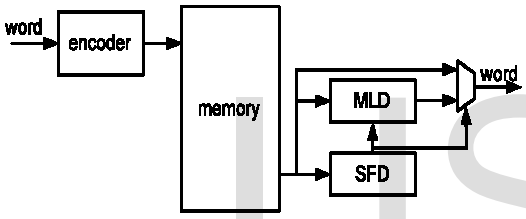

Fig: Memory system schematic of an ML decoder with SFD.

PROPOSED MLDD

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, the proposed ML detector/decoder (MLDD) has been implemented using the difference-set cyclic codes (DSCCs).

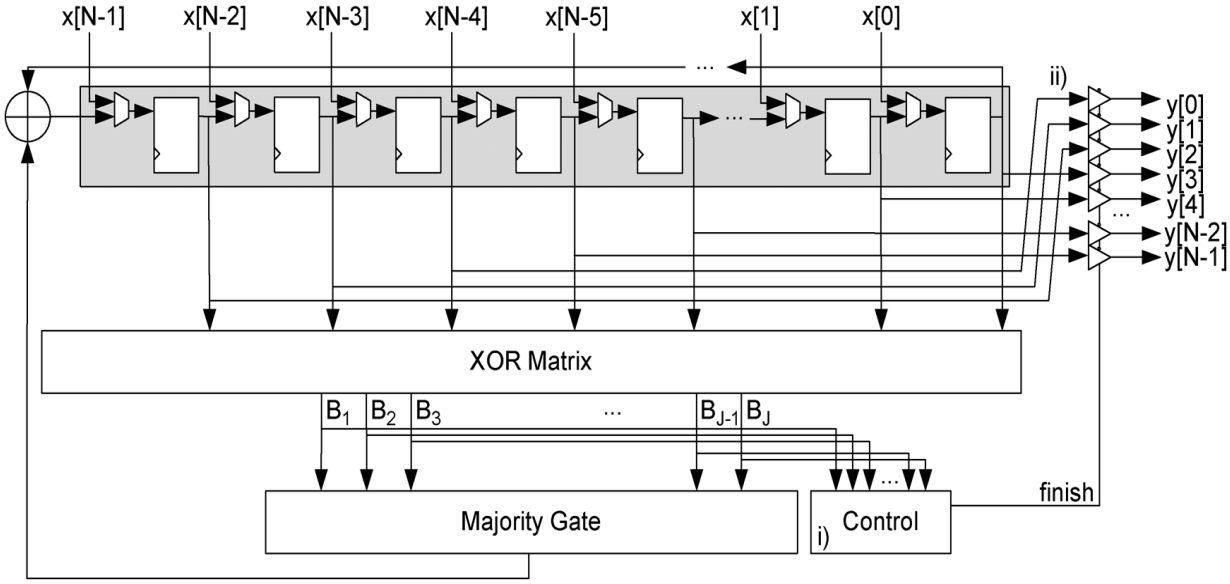

Fig: Schematic of the proposed MLDD. i) Control unit. ii) Output tristate buffers.

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.

A detailed schematic of the proposed design is shown. 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.

The additional hardware to perform the error detection is illustrated in Figure as: i) the control unit which triggers a finish flag when no errors are detected after the third cycle and ii) the output. The figure shows Flow diagram of the MLDD algorithm. Tri state buffers. 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 .

Fig.: Memory system schematic of an MLDD.

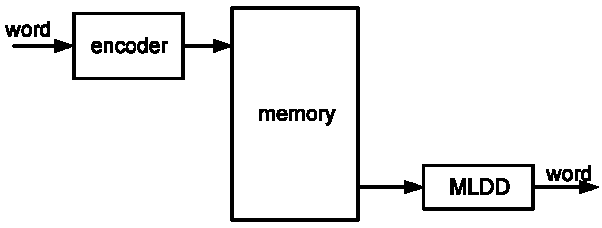

Fig : Schematic of the control unit.

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. 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. More information about performance details will be provided in the next sections. The schematic for this memory system (is very similar to the one in Figure, adding the control logic in the MLDD module.

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 theoretical proof of the proposed MLDD scheme for the case of double errors has also been presented. The extension of this proof to the case of four errors would confirm the validity of the MLDD approach for a more general case, something that has only been done through simulation in the paper. This is, therefore, an interesting problem for future research. The application of the proposed technique to memories that use scrubbing is also an interesting topic and was in fact the original motivation that led to the MLDD scheme.

REFERENCE :

[1] 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.

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

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

[4] 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.

[5] R. C. Baumann, "Radiation-induced soft errors in advanced semicon- ductor technologies," IEEE Trans. Device Mater. Reliabil., vol. 5, no. 3, pp. 301–316, Sep. 2005.

[6] J. von Neumann, "Probabilistic logics and synthesis of reliable organ- isms from unreliable components," Automata Studies, pp. 43–98, 1956.