International Journal of Engineering Research and Development
e-ISSN: 2278-067X, **p**-ISSN: 2278-800X, www.ijerd.com
Volume 7, Issue 5 (June 2013), PP.77-81

Efficient Decoding Using Majority Gate

Naveenraj.M[1], Aishwarya.E.J[2],

1PG Scholar, Sri Shakthi Institute of Engineering and Technology, Chinniampalayam, Coimbatore-641062
[2]Assistant professor/ECE, Sri Shakthi Institute of Engineering and Technology, Chinniampalayam, Coimbatore-641062

Abstract:- Normally decoding is done to correct errors present in the codeword by calculating syndrome. syndrome calculation for large size codeword makes decoding more complex**.** This paper presents an error detection and correction schemes using Majority Logic Decoding(MLD) which provides simple decoding process to detect and correct errors, the existing MLD requires number of clock cycles equal to size of codeword to detect error but the proposed decoding significantly reduces the iterations by detecting error within three clock cycles for any size of codeword using control logic .Also the proposed scheme uses detection logic to detect the uncorrectable codeword if error occured is more than actual capacity of system. Hence this makes error detection and correction schemes more efficient.

Keywords:- Block codes, Error correction codes(ECCs),LowDensity Parity Check(LDPC) ,Majority logic Memory.,

I. 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 [1], [2]. 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 [3] 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 [2]. For terrestrial radiation environments where there is a lowsoft 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 in the number of soft errors, which produces the need for higher error correction capabilities [4], [5]. The usual multierror 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 [6]. 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]. A subgroup of the low-density parity check (LDPC) codes, which belongs to the family of the ML decodable codes, has been researched in [9]–[11]. 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 [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 bits, it takes cycles in the decoding process, posing a big impact on system performance [6]. 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 a codewords 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. The simplest way to implement 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

II. EXISTING SYSTEM OVERVIEW

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 [7] for the Reed–Müller codes. Then, it was extended and generalized in [8] for all types of systematic linear block codes that can be totally orthogonalized on each codeword bit. Initially, the data words are encoded and then stored in the memory. When the memory is read, 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.

There are two ways for implementing this type of decoder. The first one is called the Type-I ML decoder, which determines, upon XOR combinations of the syndrome, which bits need to be corrected [6]. The second one is the Type-II ML decoder that calculates directly out of the codeword bits the information of correctness of the current bit under decoding [6]. Both are quite similar but when it comes to implementation, the Type-II uses less area, as it does not calculate the syndrome as an intermediate step. Therefore, this paper focuses only on this one.

Codes exist which are capable of correcting large number of random errors. Such codes are rarely used in practical data transmission systems ,however, because the equipments necessary to realize their capabilities, that is to actually correct the errors, is usually prohibited complex and expensive. The problem of finding simply implemented decoding algorithm is perhaps the outstanding unsolved problem today. Here a new class of random error correcting cyclic codes such as majority logic algorithm is defined. These codes can be decoded with the simplest decoding algorithm .

As VLSI Technology scaling continues and the device dimensions keep shrinking, memories are more and more sensitive to soft errors. Memory cores have significant impact on chip reliability therefore error detection and correction techniques are commonly used for protecting the system against soft errors.

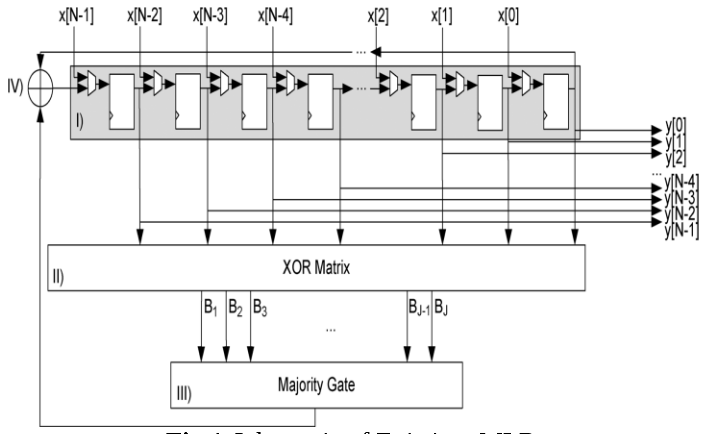

Fig 1 Schematic of Existing MLD

III. PLAIN MLD

As described before, 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. 1. 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.

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.

3.1 Plain MLD with syndrome fault detector

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 codewords are decoded [11]. Since most of the codewords 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.A 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.

IV. PROPOSED SYSTEM

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.

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

The figure1 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 . The additional hardware to perform the error detection is illustrated in Fig. 2 as: i) the control unit which triggers a finish flag when no errors are detected after the third cycle and ii) the output tristate buffers. The output tristate 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 .

4.1 Control and Detection logic

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 (four, 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 detection logic provides the information about whether codeword is correctable or not .consider that system is able to detect up to four bits but codeword contains six bit error means detection logic produces output as uncorrectable codeword, suppose the codeword contains four or less number of bits means it produces output as correctable codeword .This makes the error detection schemes more efficient than existing decoding process.

An error-correcting code (ECC) or forward error correction (FEC) code is a system of adding redundant data, or parity data, to a message, such that it can be recovered by a receiver even when a number of errors (up to the capability of the code being used) were introduced, either during the process of transmission, or on storage. Since the receiver does not have to ask the sender for retransmission of the data, a back-channel is not required in forward error correction, and it is therefore suitable for simplex communication such as broadcasting. Error-correcting codes are frequently used in lower-layer communication, as well as for reliable storage in media such as CDs, DVDs, hard disks, and RAM.

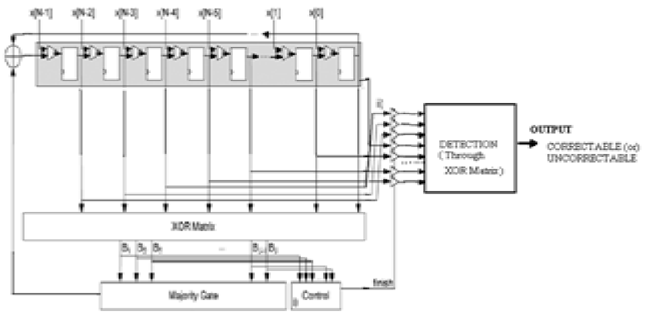

Fig 2 .Proposed MLD scheme

V. SIMULATIONS USING MODELSIM

ModelSim is a powerful simulator that can be used to simulate the behavior and performance of logic circuits. The simulator allows the user to apply inputs to the designed circuit, usually referred to as test vectors, and to observe the outputs generated in response. The user can use the Waveform Editor to represent the input signals as waveforms.

ModelSim is a simulation and debugging environment created by Mentor Graphics. ModelSim allows you to check the syntax and verify the functionality of VHDL programs.

ModelSim VHDL supports both the IEEE 1076-1987 and 1076-1993 VHDL, the 1164-1993 Standard Multivalue Logic System for VHDL Interoperability, and the 1076.2-1996 Standard VHDL Mathematical *Packages* standards. Any design developed with ModelSim will be compatible with any other VHDL system that is compliant with either IEEE Standard 1076-1987 or 1076-1993.

ModelSim Verilog is based on IEEE Std 1364-1995 and a partial implementation of 1364-2001 (see /<install_dir>/modeltech/docs/technotes/vlog_2001.note for implementation details) Standard Hardware Description Language. The Open Verilog International Verilog LRM version 2.0 is also applicable to a large extent. Both PLI (Programming Language Interface) and VCD (Value Change Dump) are supported for ModelSim PE and SE users.

5.1. Simulation Result and Analysis

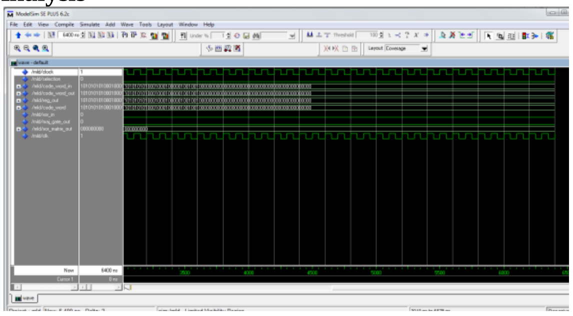

Fig 3 Simulation result of existing MLD

Simulation results of existing MLD shows that the detection of error takes number of cycles equal to size of codeword even if there is no error in the data .This leads to large area and power consumption and results in complexity of decoding process . The codeword of size 73 contains no error but the existing MLD takes 73 iterations to detect the error so that efficiency of decoding process is reduced in this scheme.

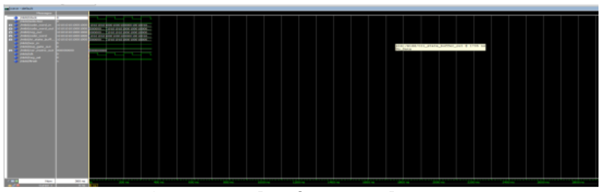

Fig 4 Result of Control Logic

The above simulation result shows that proposed MLD in which the error is detected within three cycles .This significantly reduces area and power consumption of the system. The Error free codeword size is 73 but proposed MLD takes only three iterations to detect error in all 73 bits. The control logic sends finish signal if there is no error in the data after three iterations .Hence proposed MLD has better decoding process than existing MLD.

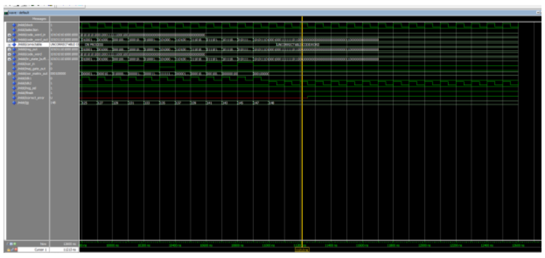

Fig 5 Result of Detection logic

The above simulation result shows that proposed MLD with Detection logic which displays result as uncorrectable codeword because the system is able to detect up to 4 bits only but the codeword contains six bit errors Hence detection logic display produces this output after completion of entire clock cycles

REFERENCES

[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]. 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 nanoscale 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 nanoscale 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 NanoMemory applications," *I*EEE 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," IEEETrans. Device Mater. Reliabil., vol. 10, no. 1, pp. 33–39, Mar. 2010.