ISSN 2348 – 9928
IJAICT Volume -1, Issue-5, September 2014 Doi:01.0401/ijaict.2014.01.18 Published Online 05 (05) 2014

ENHANCED DECODING ALGORITHM FOR ERROR DETECTION AND CORRECTION IN SRAM

Ms. E.Hemha Chandra
PG Scholar,
PPG Institute of Technology,
Coimbatore, Tamilnadu, India.

Mr. S.Baskar
Assistant Professor,
Angel College of Engineering and Technology,
Coimbatore, Tamilnadu, India.

Abstract— As expertise scales, Multiple Cell Upsets (MCUs) become more common and affect a larger number of cells. In order to guard memories alongside of MCUs as well as SEUs is to make use of sophisticated Error detecting and correcting codes that can accurate more than one error per word. A sub-group of the low-density parity checks (LDPC) codes, which be-longs to the family of the Majority logic decoding has been newly projected for memory application and Difference set codes are one example of these codes which contributes for error detection and correction.ML decodable Codes are appropriate for memory applications due to their ability to correct a large number of errors. In this paper, the anticipated scheme for fault-detection and correction method significantly makes area overhead minimal and to reduce the decoding time through DC codes than the existing technique and it shows potential option for memory applications. HDL accomplishment and synthesis consequences are included, showing that the proposed techniques can be proficiently implemented.

Keywords— Difference Set Codes, Error Correction Codes, Majority Logic Decoding, Memory, Multiple Cells Upsets (Mcus).

I. INTRODUCTION

In recent centaury, the need for efficient and reliable data transmission and storage system has been significantly highlighted. RADIATION-INDUCED soft errors are one of the major issues for Memory reliability. To prevent soft errors from causing data corruption, memories are typically protected with error correction codes (ECCs). The most commonly used codes can correct one error and detect two errors per memory word are known as single-error- correction double-error- detection (SEC-DED) codes. Their main advantages are that they require few additional bits per word and that the decoding process is simple.

A SEC-DED code enforces a minimum distance of four between any two coded words by having a distance of four any word that suffers a double error would be in the worst case at a distance of two from any valid coded word. Therefore, it cannot be mistaken for a single error and miscorrected. The same approach is used for codes that can correct two errors; in this case, Double Error Correction Triple Error-Detection (DEC-TED) codes are used. However, this increases the decoder complexity substantially. Further, in a hierarchical approach that combines a Hamming code and a Bose–Chaudhuri–Hocquenghem code was proposed to minimize the Latency. The use of Euclidean geometry (EG) codes has also been considered for memory protection, The particular EG codes studied are one-step Maximum likelihood decodable and therefore, these decoders can be implemented with low cost. Other codes that are one-step Maximum likelihood decodable are difference-set (DS) codes. Their use for memory protection has also been studied recently showing that the properties of the codes can be exploited to reduce the decoding time significantly.

The combination of a simple decoder and reduced decoding time makes DS codes an attractive option for memory protection. 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 Maximum likelihood (ML) decodable. A sub-group of the low-density parity check (LDPC) codes, which be-longs to the family of the ML decodable codes, has been re-searched in. In this paper, we will focus on one specific type of LDPC codes, namely the difference-set cyclic codes (DSCCs), which are widely used in the Japanese teletext system or FM multiplex broadcasting system. In contrast with these works, several research groups have aimed to improve the performance of decoding algorithm for memory applications.

In this work one modification of the current algorithm and a new low complexity high performance algorithm is proposed. In phase I decoding algorithm has been analyzed for the existing approach, and in future the proposed work is carried out in CRC, mod-2 arithmetic and Benes network.

The remainder of this paper is organized as follows. Section II gives an overview of existing ML decoding solutions; Section III presents the Existing ML difference-set cyclic codes

algorithm; Section IV Proposed two dimensional Two dimensional modulo sum algorithm Section V the results obtained for the different versions in respect to effectiveness, performance, and area and power consumption. Finally, Section V discusses conclusions and gives an outlook onto future work.

1.1 Motivation

There-fore, it is necessary to develop an improved decoding algorithm without introducing the computation complexity and increasing implementation cost compared to the conventional algorithms.

1.2 Contribution

The review of existing decoding algorithms for DSCCs codes and a proposed improved decoding algorithm, which can achieve better decoding performance without requiring any additional computation complexity or hardware overhead compared to the conventional ones. Further the low complexity switch network and a novel efficient control signals generation for a reconfigurable DCSSc (combined random LDPC and DSCCs) have been analyzed. The proposed architecture can lead to significant reductions in hardware complexity. However there is a between performance tradeoff between area and complexity in the choice of soft-decision and hard-decision algorithms. The hard-decision algorithms in comparison to the soft-decision algorithms are considerably less complex but their performance is not as good as soft-decision algorithms. The proposed method reduces the low complexity property of hard-decision algorithms and the good performance properties of soft-decision algorithms are preserved. Several error detecting algorithms are proposed and experimental results are compared these inurn reduce the complexity of hardware.

II. PROFILE ABOUT ML DECODING

Existing version of the ML decoder that improves the designs presented before. Starting from the original design of the ML decoder introduced in section 2. The Existing ML detector/decoder (MLDD) has been implemented using the difference-set cyclic codes (DSCCs) .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.
- Modular encoder and decoder blocks that allow an efficient Hardware implementation for systematic code structure for clean partition of information and code bits in the memory.

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. 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 existing method stops intermediately in the third cycle, as illustrated in Fig.1. 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 the 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.[1]

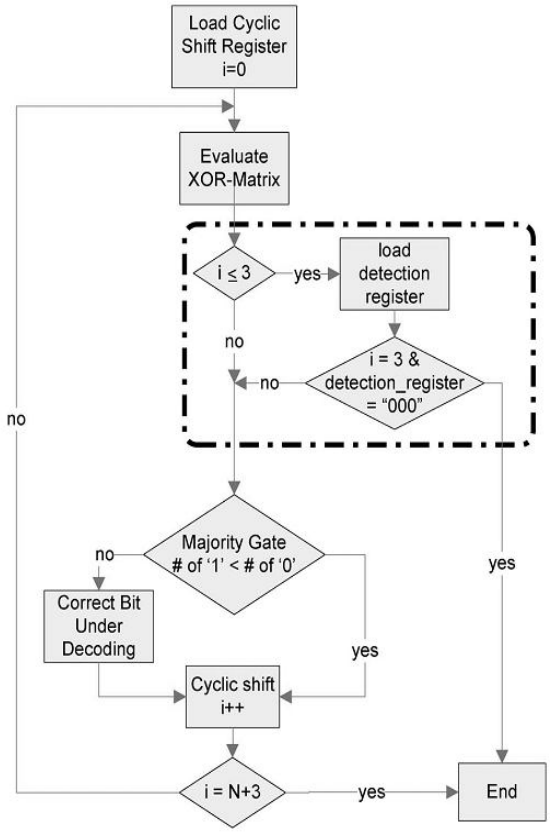

Fig 1: Flow diagram of the MLDD algorithm.

III. EXISTING ALGORITHM OF MLDD

The data N is divided into k segments each of m bits. The segments (S_t) are added using ones complemented arithmetic to get the sum and corresponding output is complemented to get the Two dimensional modulo sum, and then segmented Two dimensional modulo sum is sent along with data segments. All the received segments (S_r) are added using ones complemented arithmetic to get the sum. Then the sum (S_s) has to be complemented, if the result is zero, the received data is accepted; otherwise rejected. (As shown in Algorithm.1.)

[1].	Initialize H, G
[2].	Initialize P, I
[3].	If ($H=\mathbf{P}^T\|\mathbf{I}$) then
[4].	H<=valid
[5].	Else
[6].	H<=invalid
[7].	End if
[8].	If ($G=\mathbf{I}\|\mathbf{P}$) then
[9].	G<=valid
[10].	Else
[11].	G<=invalid
[12].	End if
[13].	Initialize
[14].	c=m*g
[15].	If (code=valid)
[16].	Then $c*H^t$ terms to zero else
[17].	$C*H^t$ terms to be error
[18].	End if
[19].	Data segmentation
[20].	For: 1: N do
[21].	Segment the N
[22].	S_t (N) <=s1(m) mod 2 addition s2 (m)
[23].	Invert the Two dimensional modulo sum S_t(N)
[24].	S_r(N)<= s1(m) mod 2 addition s2 (m)

Algorithm 1. Code construction and two dimensional two dimensional modulo sum equations.

The ML decoding algorithm is a hard-decision message-passing algorithm for LDPC codes. A binary (hard) decision about each received bit is made by the detector and this is passed to the decoder. For the ML decoding algorithm the messages passed along the Tanner graph edges are also binary: a bit node sends a message declaring if it is a one or a zero, and each check node sends a message to each connected bit node, declaring what value the bit is based on the information available to the check node.[3] The check node determines that its Two dimensional modulo sum equation is satisfied if the modulo-2 sum of the incoming bit values is zero. If the majority of the messages received by a bit node are differ-ent from its received value the bit node changes (flips) its current value.

These algorithms are really important due to their simple implementation. Their binary structure, binary memories and limited wiring, makes them remarkable in hardware implementation especially in the situations where only hard-decision values are available at the receiver. Gallager's algorithm A (GA) is, for instance, a hard-decision decoder in the set of Majority Based algorithms with Alphabet A = (-1, 1) In addition, the out-going message of a check node is the product of its extrinsic incoming messages. The MB^w algorithms are studied in depth in, by the use of density evolution another example of hard-decision algorithms is ML decoding algorithm.

In this algorithm a flipping function is defined that counts the number of unsatisfied syndrome bits (check nodes) in which each variable node participates. Each variable node has a binary buffer to store a hard decision value; the content of this buffer will be flipped if the corresponding output of the flipping function is more than a certain threshold.[2] Decoding continues until all check node equations are satisfied or until maximum number of iterations is reached. Maximum Likelihood estimation (MLE) is an important tool in determining the actual probabilities of the assumed model of communication. In reality, a communication channel can be quite complex and a model becomes necessary to simplify calculations at decoder side.

The model should closely approximate the complex communication channel. There exist a myriad of standard statistical models that can be employed for this task; Gaussian, Binomial, Exponential, Geometric, Poisson, etc., A standard communication model is chosen based on empirical data. Each model mentioned above has unique parameter that characterizes them. Suppose a binomial model is chosen (based on observation of data) for the error events over a particular channel, it is essential to determine the probability (p) of the binomial model.

If a Gaussian model (normal distribution) is chosen for a particular channel then estimating μ (mean) and σ^2 (variance) are necessary so that they can be applied while computing the conditional probability of p(y received | x sent) Similarly estimating lambda is a necessity for a Poisson distribution model. Maximum likelihood estimation is a method to determine these unknown parameters associated with the corresponding chosen models of the communication channel. (As shown in Algorithm.2)

[1]. For i=1 to m do
[2]. St=Sr
[3]. End for
[4]. Repeat
[5]. For i=1 to m do
[6]. If (St=Sr) then
[7]. All the values known
[8]. Finished else
[9]. (St≠Sr)
[10]. Er belongs to Ss
[11]. Flip Er corresponds to Ss
[12]. Finished
[13]. End if
[14]. End for

Algorithm 2. ML decoding

$$P(y \text{ received} | x \text{ send}) = (1-p)^{n-d}.p^{d}$$

Where ,
d=the hamming distance between the received and the sent codeword's.
n= number of bit sent.
p= error probability of the BSC.
1-p = reliability of BSC.

IV. PROPOSED CYCLIC REDUNDANCY CHECK

A cyclic redundancy check is the error detecting method for memory and digital networks and storage devices to detect accidental changes to raw data. Blocks of data entering these systems get a short check value attached, based on the remainder of a polynomial division of their contents.

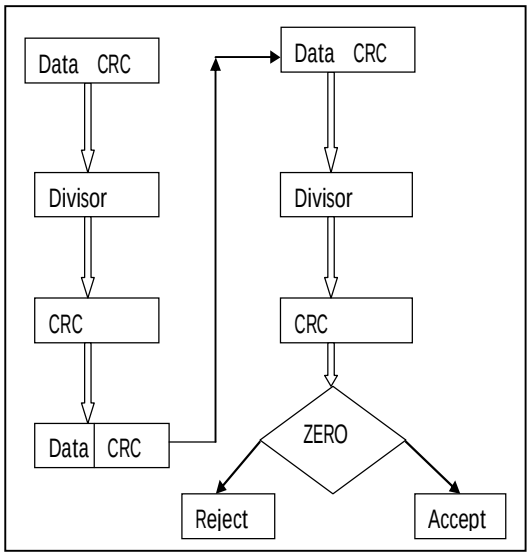

Fig2 : Flowchart of the CRC design flow

On retrieval the calculation is repeated, and corrective action can be taken against presumed data corruption if the check values do not match. As shown in fig.2,The CRC is based on polynomial arithmetic, in particular, on computing the remainder of dividing one polynomial in GF (2) (Galois field with two elements) To develop a hardware circuit for computing the CRC checksum, we reduce the polynomial division process to its essentials. The process employs a shift register, which we denote by CRC. This is of length r (the degree of G) bits, not as you might expect.

When the subtractions (exclusive or's) are done, it is not necessary to represent the high-order bit, because the high-order bits of G and the quantity it is being subtracted. CRCs are specifically designed to protect against common types of errors on communication channels, where they can provide quick and reasonable assurance of the integrity of content delivered. However, they are not suitable for protecting against intentional alteration of data.

Firstly, as there is no authentication an attacker can edit content and recomputed the CRC without the substitution being detected. When stored alongside the data, CRCs and cryptographic hash functions by themselves do not protect against intentional modification of data. Any application that requires protection against such attacks must use cryptographic authentication mechanisms, such as message authentication codes or digital signatures (which are commonly based on cryptographic hash functions).

To compute an n-bit binary CRC, line the bits are representing the input in a row, and position the (n+1)-bit pattern representing the CRC's divisor (called a "polynomial") underneath the left-hand end of the row. Start with the message to be encoded: This is first padded with zeroes corresponding to the bit length n of the CRC. Here is the first calculation for computing a 3-bit CRC. If the input bit above the leftmost divisor bit is 0, do nothing. If the input bit above the leftmost divisor bit is 1, the divisor is XORed into the input (in other words, the input bit above each 1-bit in the divisor is toggled).

The divisor is then shifted one bit to the right, and the process is repeated until the divisor reaches the right-hand end of the input row Since the leftmost divisor bit zeroed every input bit it touched, when this process ends the only bits in the input row that can be nonzero are the n bits at the right-hand end of the row. These n bits are the remainder of the division step, and will also be the value of the CRC function (unless the chosen CRC specification calls for some post processing).

The validity of a received message can easily be verified by performing the above calculation again, this time with the check value added instead of zeroes. The remainder should equal zero if there are no detectable errors. The selection of

generator polynomial is the most important part of implementing the CRC algorithm. The polynomial must be chosen to maximize the error-detecting capabilities while minimizing overall collision probabilities. The most important attribute of the polynomial is its length (largest degree (exponent) +1 of any one term in the polynomial), because of its direct influence on the length of the computed check value. (As shown in the Algorithm.3.)

The most commonly used polynomial lengths are:

- 9 bits (CRC-8)
- 17 bits (CRC-16)
- 33 bits (CRC-32)
- 65 bits (CRC-64)

Initialize CRC register -0-bits.

IN= m bit.

If (Higher order CRC = 1),

>> CRC =m together left 1 position,

XOR the result with Low-order r bits of G.

Else

>> CRC and m left 1 position.

Algorithm 3. CRC Flow

If there are more message bits, go back to get the next one. It might seem that the subtraction should be done first, and then the shift. It would be done that way if the CRC register held the entire generator polynomial, which in bit form is a bit. Instead, the CRC register holds only the low-order r bits of G, so the shift is done first, to align things properly. By making use CRC in memory error detection it in turn increases the basic parameters but it helps to reduce the hardware complexity through LFSR- linear feedback shift register.

V. RESULTS

5.1 Memory

The memory read access delay of the plain MLD is directly dependent on the code size, i.e., a code with length 72 needs 72 cycles, etc. Then, two extra cycles need to be added for I/O. On the other hand, the memory read access delay of the proposed MLDD is only dependent on the word error rate (WER). If there are more errors, then more words need to be fully decoded. In all the above methods they made use of two dimensional parity equations it increase the memory through extra overhead due to the addition of redundant bit (parity).[4] It explores the idea of two dimensional Two dimensional modulo sum which in turn reduce the memory consumption of the device. Redundancy has been reduced through this algorithm (As shown in Table.I)

TABLE 1. Memory Results of The Device

Logic utilization	Proposed method checksum	Proposed method CRC	Available resource
Number of slices	16	25	46560
I/O Buffers	71	64	4896

5.2 Area

The previous subsection showed that the performance of the Proposed design MLDD is much faster than the plain MLD version, but slightly lower than the design with syndrome calculator (SFD).As mentioned several times,[5] this is compensated with a clear savings in area.

The conclusions on the area results are given as follows.

- The MLD design requires little area compared with the other two designs. However, as shown before, the performance results are not very good.
- The SFD version, which had the best performance, needs more area than the MLD does, ranging from 25.40% to 294.94% depending on. Notice that the increment of Area grows quicker than does.
- The MLDD version has a very similar performance to SFD, However it requires a much lower area overhead, ranging From 10.16% to 0.43%.

In all the methods they used two dimensional parity check equations and increase the occupancy of gate it has to be declined through two dimensional modulo sum algorithmic flow in order to save the area. These conclusions can be extrapolated to power. The over-head introduced by MLDD is very small, contrary to the SFD case.

An important final comment is that the area overhead of the MLDD actually decreases with respect to the plain MLD version through two dimensional modulo sum algorithm. For large values of, both areas are practically the same.[6]

The reason for this is that the error detector in the MLDD has been designed to be independent of the size code the opposite situation occurs, with the SFD technique, which uses syndrome calculation to perform error detection: its complexity grows quickly when the code size increases.

TABLE 2. delay Results of The Device

	Existing method		Proposed method	
Cell: in>out	Gate delay	Net delay	Gate delay	Net delay
IBUF	1.218	1.108	1.218	0.995
LUT's	0.704	0.668	0.704	0.622
	0.704	0.455	0.704	0.455
	0.704	0.424	0.704	0.424
	0.704	0.420	0.704	0.420
OBUF	3.272		3.272	
Total	10.381ns(7.306ns logic, 3.075ns route,70.4% logic,26.6% route)		10.222ns (7.306ns logic, 2.916ns route,71.5% logic, 28.5% route)	

5.3 Timing Summary of existing algorithm

Minimum period: 2.309ns
(Maximum Frequency: 433.088MHz)
Minimum input arrival time before clock: No path found
Maximum output required time after clock: 5.693ns
Maximum combinational path delay: 10.381ns

As it is shown in the above results(Tabel 1,Table 2) due to the addition of redundant bit it increase the area and memory It has to be overcome by two dimensional Two dimensional modulo sum.[7,8]

5.4 Power

The power consumption of the existing method has in turn enhance the on-chip power due to high increase in static power, it has to be overcome by two dimensional modulo sum.(As shown in Fig.3 and Fig.4).[9]

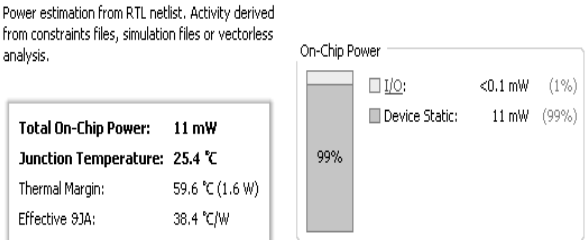

Fig 3: Two dimensional modulo sum.

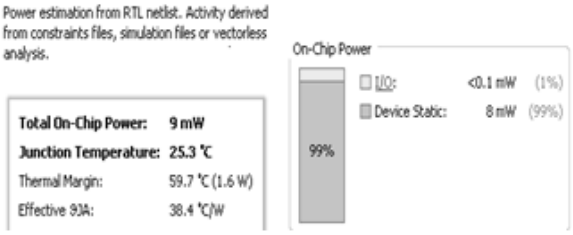

Fig 4: CRC

VI. CONCLUSION

In this paper, an algorithmic scheme has derived from the existing technique which effectively helps to correct errors caused by Multiple Cell Upsets (MCUs) as well as SEU in memories. Identification of the errors in an MCU has to analyze by placing data in the memory, thus providing additional error correction capabilities. Modified algorithmic methodology helps to correct burst errors than the existing method and helps to reduce the LUTs level. Additionally, It helps to accelerate the decoding and effectively reduced the area and memory occupied by the present MLDD, than the previously proposed algorithms for DS codes. Thus the results show that the method is also effective in reducing LUTs, Power and delay when MCUs are present. The proposed scheme has been validated by simulation using a large number of error combinations and implemented to evaluate its cost in terms of circuit area and speed.

ACKNOWLEDGMENT

The authors would like to thank PPG Institute of Technology and Angel College of Engineering and Technology for their Valuable support.

References

[1] 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

[2] J. L. Massey, "Threshold Decoding. Cambridge", MA: M IT Press, 1963.

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

[4] C. W. Slay man, "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] 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

[6] G.Torrens,B.Alorda,S.Barceló,J.L.Rosselló,S.A.Bota,andJSegura,"Desig n hardening of nanometre SRAMs through transistor width modulation and multi- combination," IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 57, no. 4, pp. 280–284, Apr. 2010

[7] E. Ibe, H. Taniguchi, Y. Yahagi, K. Shimbo, and T. Toba, "Impact of scaling on neutron-induced soft error rate in SRAMs from a 250 nm to a

22nm design rule," IEEE Trans. Electron Devices, vol. 57, no. 7, pp. 1527–1538, Jul. 2010.

[8] S. Liu, P. Rev iriego, and J. A. Maestro, "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, Jan. 2012.

[9] "Multiple Cell Upset Correction in Memories Using Difference Set Codes" by Pedro Reverie, Member, IEE E, Mark F. Flanagan, Senior Member, IEEE, Shih-Fu Liu, and Juan Antonio Maestro, Member, IEEE, Jun 2012