International Journal of Scientific Engineering and Research (IJSER)
www.ijser.in
ISSN (Online): 2347-3878
Volume 1 Issue 1, September 2013

Efficient Fault Detection Majority Logic Correction with in Memory with Difference-Set Codes

N. V. P. Naidu Babu [1], P. M. Francis[2], B. Prasad Kumar[3]

[1]M.Tech, GITAS College, Bibbili, A. P, India

[2]Department of ECE, Head of Department, Assistant Professor, GITAS, College, Bobbili, A. P, India

[3]Department of ECE, Assistant Professor, GITAS, College, Bobbili, A. P, India

Abstract: ***Now a-days on soc applications the major problem is with the on chip memory to be faster and we require without any error correction and disrupt the altering digital circuit are becoming the major concern for memory application. This paper presents an error-detection method for difference-set cyclic codes with majority logic decoding. To correct a large no of correction Majority logic decodable codes are suitable for memory applications. 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.***

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

1. Introduction

The impact of technology scaling smaller dimensions, higher integration densities, and lower operating voltages has come to a lev el th at reliab ility o f memories is pu t in to jeopardy, not only in e xtreme radiation environments like spacec raft and avi onics electronics , but also at normal terrestrial environments [1] , [2]. Es pecially, SRAM memory failure rates are incre asing si gnificantly, th erefore p osing a ma jor reliability co ncern for m any ap plications. Some co mmonly used mitigation techniques are:

- Triple Modular Redundancy (TMR);
- Error Correction Codes (ECCs).

TMR i s a s pecial case o f t he von Ne umann m ethod [3] consisting of th ree versions of the design in parallel, with a majority vot er sel ecting t he cor rect o utput. As t he m ethod suggests, the complexity overhead would be three times plus the complexity of the majority voter and thus increasing the power con sumption. Fo r m emories, it tu rned ou t th at ECC codes are th e best way t o mitig ate memory so ft erro rs [2]. For terrestrial rad iation env ironments wh ere th ere is a lo w soft er ror rate (SER) , co des 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 co nsequence o f aug menting in tegration den sities, th ere is an increase in the number of soft errors, which produces the need for h igher erro r co rrection cap abilities [4], [5]. Th e usual m ulti error co rrection codes, such as Reed– Solomon (RS) or B ose C haudhuri Hoc quenghem (B CH) a re not suitable fo r this task . Th e reason fo r t his is th at th ey u se more sophisticated dec oding algorit hms, like com plex algebraic (e. g., fl oating point o perations or l ogarithms) decoders t hat can decode i n fi xed t ime, and simple grap h decoders, t hat u se iterati ve algorith ms (e.g ., belief propagation). B oth are ve ry com plex and inc rease computational costs [6]. A mong t he EC C codes t hat m eet the requ irements o f higher error co rrection cap ability a nd low dec oding com plexity, cyclic bl ock code s ha ve b een identified as good candidates, due to their property of being majority l ogic (M L) dec odable [7] , [8]. A sub group o f t he low-density pa rity check (L DPC) c odes, which bel ongs t o the family of the ML decoda ble codes, has been researched in [9]–[11]. In this paper, we will focus on one specific type of LDPC codes, n amely th e d ifference-set cyclic co des (DSCCs), which is wid ely u sed in th e Jap anese tele-tex t system o r FM multiplex b roadcasting syste ms [1 2]–[14]. The m ain reaso n for using ML d ecoding is th at it is v ery simple to implement and thus it is very practical and has low complexity. The drawback of M L decoding i s t hat, 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 p roblem is t o im plement parallel en coders and decoders. This sol ution would enormously increase the complexity and, therefore, the power consumption. As most of th e m emory read ing accesses will h ave n o erro rs, th e decoder is most of the time working for no reason. This has motivated the use of a fault detector module [11] that checks if t he co deword c ontains a n er ror a nd t hen t riggers t he correction m echanism accordingl y. In this case, only the faulty code words need correction, and therefore the average read m emory access is speeded up, at the expe nse of a n increase in hardware cost and power consumption. A similar proposal has been presented i n [15] fo r t he case of flash memories. The si mplest way to im plement a fau lt detector for an ECC i s by cal culating t he sy ndrome, but t his generally im plies ad ding a nother very c omplex fu nctional unit. Th is p aper exp lores th e id ea o f using the ML d ecoder circuitry as a fau lt d etector so t hat rea d operati ons are accelerated wi th al most no additional hardwa re cost. The results sh ow that th e pro perties o f DSCC-LDPC en able efficient fa ult det ection. T he rem ainder of t his pa per i s organized as fol lows. Sect ion I I gi ves an o verview of

existing ML decoding so lutions; Section III presen ts the novel M L det ector/decoder (MLDD) using di fference-set cyclic codes; Sect ion I V di scusses th e results ob tained for the diffe rent versio ns in res pect to effective ness, performance, and a rea a nd po wer c onsumption. Fi nally, Section V discusses conclusions and gi ves an outlook onto future work.

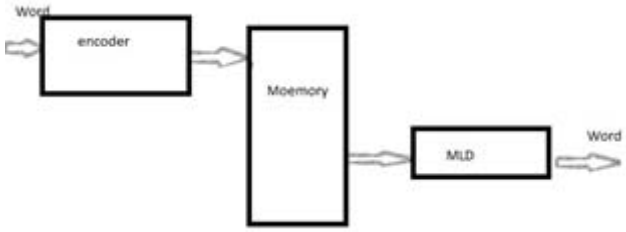

Figure 1: Memory system schematic with MLD

2. Existent Majority Logic Decoding (MLD) Solutions

MLD is based on a number of parity check equations which are orthogonal to each othe r, so that, at each iteration, eac h codeword bit only participates in one parity check equation, except th e very first bit wh ich co ntributes to all eq uations. For t his reason , th e m ajority resu lt o f t hese p arity ch eck equations decide t he c orrectness of t he cu rrent bit un der decoding. M LD wa s fi rst m entioned i n [7] for t he R eed–Müller cod es. Then, it was ex tended and gen eralized i n [8] for al l t ypes of sy stematic l inear block co des t hat can b e totally orthogonalized on each codeword bit.

A generic schematic of a m emory system is depicted in Fig. 1 for th e usage of an ML d ecoder. Initially, th e data word s are encode d and the n store d in the memory. W hen the memory is read , t he co deword is th en fed through th e ML decoder before sent to the output for further pr ocessing. I n this decoding pr ocess, the d ata 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. T he first one is called the Type-IML decoder, which determines, upo n XOR co mbinations of th e syndrome, which bits need to be corrected [6]. T he sec ond one is the Type-II ML decoder that calculates directly out of the cod eword b its th e in formation of correctne ss of the current bit u nder decoding [6]. B oth a re quite si milar 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.

2.1 Plain ML Decoder

As described bef ore, t he ML dec oder is a si mple and powerful dec oder, capa ble o f cor recting 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 u nder decod ing, as illu strated in Fig. 2. Th e input sig nal is in itially sto red in to th e cyclic sh ift reg ister and sh ifted thro ugh all th e taps. The intermediate values in each tap are then used $\{^{B}j\}$ to calcu late t he resu lts o f th e check sum equations from the XOR matrix. In t he N cycle, the res ult has reache d the final tap, producing the output signal Y (which is the decoded version of input x).As stated before, x input might correspond to wrong data corrupted by a so ft er ror. To handle t his si tuation, t he dec oder would behave as fo llows. After th e in itial step , in wh ich the codeword is lo aded in to t he cyclic sh ift reg ister, the decoding start s by calculati ng t he pa rity check e quations hardwired in t he XOR m atrix. Th e resu lting su ms $\{^{B}j\}$ are then fo rwarded to th e m ajority g ate for ev aluating its correctness. If the number of 1's re ceived in is greater than the number of 0's that would mean that the current bit under decoding is wrong and a sign al to c orrect it would be triggered. Ot herwise, the b it under dec oding would b e correct and no extra operations would be needed on it.In the next step, t he co ntent of t he re gisters a re rot ated a nd t he above procedure is rep eated until all N codeword bits have been p rocessed. Fi nally, t he pari ty chec k sums sho uld be zero if the c odeword has been correctly decode d. Furt her details on how this algorithm works can be found in [6]. The whole algorithm is depicted in Fig. 3.The previous algorithm needs as m any cycles as the number of bits in th e i nput signal, which is al so t he number of t aps of, N , i n t he decoder. Thi s i s a bi g im pact on t he per formance of t he system, depending on the size of the code. For example, for a codeword of 73 bi ts, t he decoding would take 73 cy cles, which would be excessive for most applications.

2.2 Plain MLD with Syndrome Fault Detector (SFD)

In order t o i mprove t he decoder pe rformance, al ternative designs m ay be u sed. One p ossibility is to ad d a fault detector by ca lculating t he s yndrome, so t hat o nly faul ty codewords a re dec oded [11]. Si nce m ost of t he co dewords will be error-free, no further correction will b e needed, and therefore p erformance will not b e affected . Althou gh th e implementation of an SFD reduces the a verage latency of the decoding proces s, it also adds complexity to th e design (see Fig. 4). T he 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 fau lty co deword is d etected wh en at least o ne of th e syndrome b its is "1 ." Th is trig gers th e MLD to start th e decoding, as explained before. O n t he o ther ha nd, i f t he codeword is error-free, it is fo rwarded directly to the output, thus sa ving the correction cycles. In this way, the performance is i mproved in ex change of an add itional module i n t he m emory sy stem: a matrix of X OR gates t o resolve the parity check matrix, where each check bit results into a synd rome eq uation. This fin ally resu lts in a quit e complex m odule, with a large am ount of a dditional hardware and power consumption in the system.

3. Proposed Ml Detector/Decoder

This section presents a m odified version of the ML decoder that improves the designs presented before. Starting from the original desi gn o f t he M L decoder i ntroduced i n [8], t he proposed M L det ector/decoder (M LDD) has been implemented u sing t he di fference-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

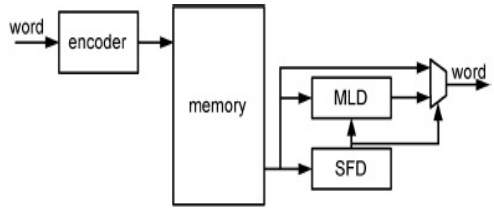

Figure 2: Momory system schemties for ML decoder with SFD

An important thing about the DSCC is that its systematical distribution allows the MLdecoder 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," as shown in Fig. 5(a). This figure shows a bit-flip affecting bit 42 of a codeword N= 73 with length and the related check sum that produces a "1." However, in the case of Fig. 5(b), the codeword 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, in Fig. 5(c), there are three bit-flips which again are detected by the check sum (with a "1"). As a conclusion of these examples, any number of odd bitflips can be directly detected, producing a "1" in the corresponding ^{B}j. The problem is in those cases with

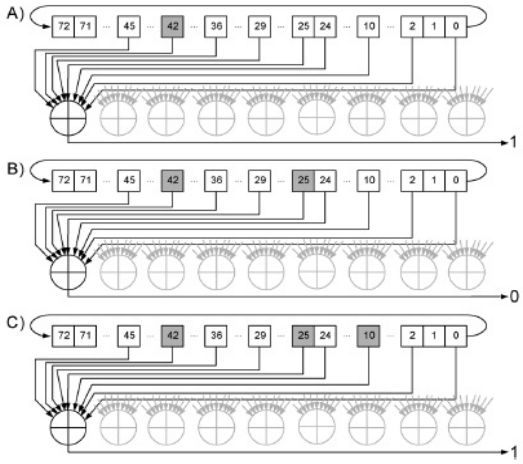

Figure 3: single check equation of a N = ML73 decoder a) one bit flip b) two bit flip c) three bit flip

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 can not 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. The proof of this hypothesis is very complex from the mathematical point of view. Therefore, two alternatives have been used in order to prove it, which are given here.

• Through simulation, in which exhaustive experiments have been conducted, to effectively verify that the hypothesis applies (see Section IV).

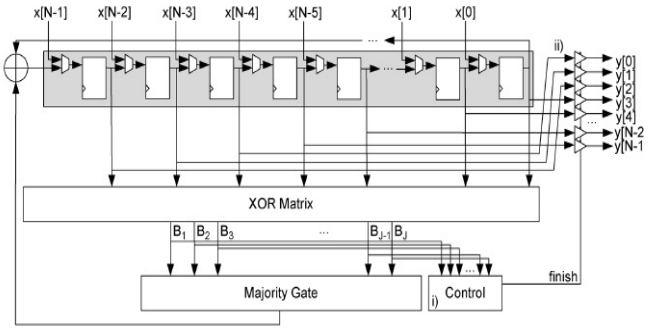

Figure 4: Proposed MLDD schematic 1) control unit 2) tri state buffer

• Through a simplified mathematical proof for the particular case of two bit-flips affecting a single word (see Appendix).

For simplicity, and since it is convenient to first describe the chosen design, let us assume that the hypothesis is true and that only three cycles are needed to detect all errors affecting up to five bits (this will be confirmed in Section IV). 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 codewords bits by processing the ML decoding during cycles, the proposed method stops intermediately in the third cycle, as illustrated in Fig. 6.

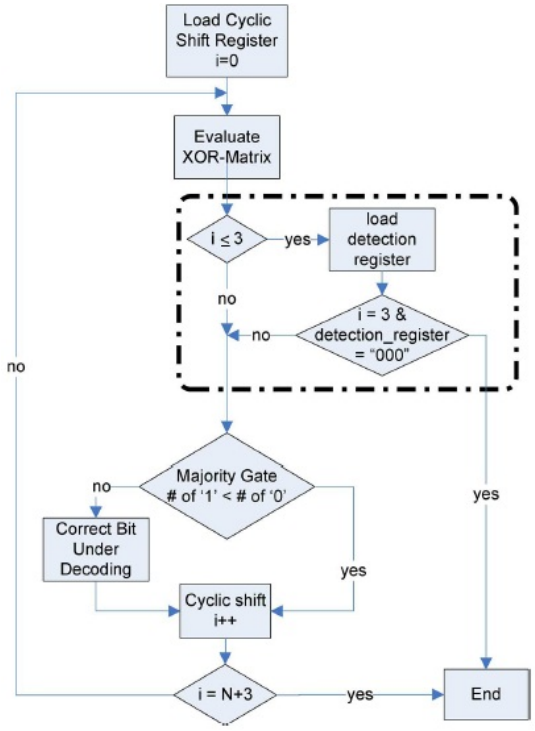

Figure 5: flowchart of MLDD algorithm

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. 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. In the third cycle, the OR 2 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 one in Figure 1, adding the control logic in the MLDD module.

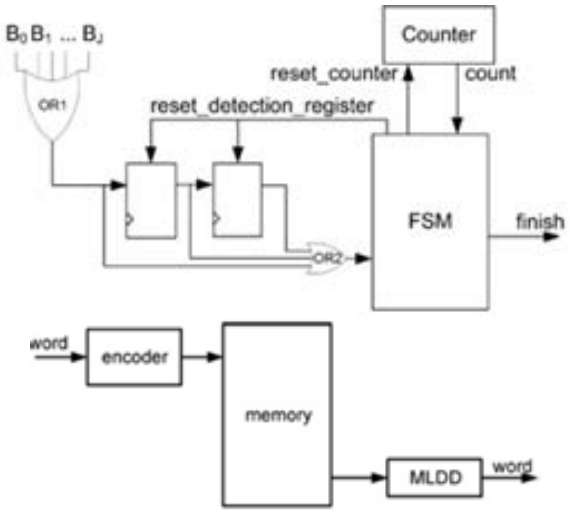

Figure 6: the proposed MLDD memory design

4. Results

The experimental results to measure the effectiveness, performance and area of the proposed technique will be presented.

4.1 Effectiveness

Here, the hypothesis that any error pattern affecting up to five bits in a word can be detected in just three cycles of the decoding process will be verified. Additionally, the detection of errors affecting a larger number of bits is also briefly discussed. As stated in previous sections, an odd number of errors will not pose any problem to a traditional parity check detector, but an even number will. Therefore, this is the scenario that has been studied. Several word widths have been considered in order to perform the experiments. The details are shown in Table I, where, for each size, the number of data and parity bits are stated. Given a size, all combinations of two and four bit-flips on a word have been calculated, in order to study all of the possible cases. The number of combinations can be seen in Table I for different values of with double and quadruple errors.

Table 1: Data word length

N	*Data bits*	*Parity bits*
73	48	35
273	191	82
1057	833	244

As expected, increasing the code length implies an exponential growth of the number of combinations, and therefore, of the computational time. An important final comment is that the area overhead of the MLDD actually *decreases* with respect to the plain MLD version. For large values of, both areas are practically the same. 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. One of the problems to make the MLDD module independent of

has been the mapping of the in termediate delay line values to the output signals. The reason is that this module behaves in two different ways depending if the processed word is erroneous or correct. If it is correct, its output is driven after the third cycle, what means that the word has been shifted three positions in the line register.

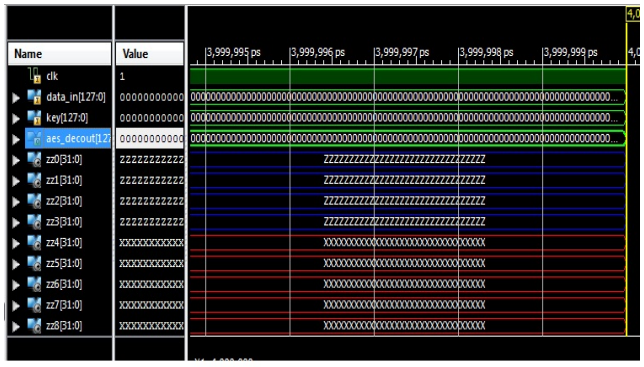

Figure 7: the error identification of data

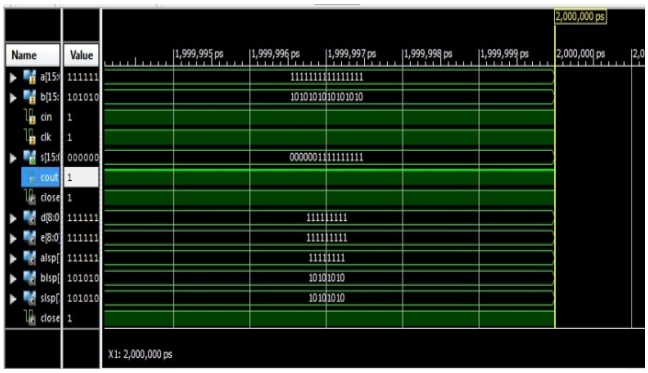

Figure 8: Describe the adjustment of the error and data retrieve

If it is wrong, the word has to be fully decoded, what implies being shifted positions. So, both scenarios end up with the output values at different positions of the shift register. Then some kind of multiplexing logic would be needed to reorder the bits before mapping them to the output. However, the area of this logic would grow with linearly. In order to avoid this, it has been decided to make three extra shift movements in the case of a wrong word, in order to align its bits with those of a correct word. After this, the output bits are coherent in all situations, not needing multiplexing logic. The penalty for this solution is three extra cycles to decode words with errors, which usually has a negligible impact on performance

5. 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.

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] 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 Nano Memory applications," *IEEE Trans. Very Large Scale Integr.(VLSI) Syst.*, vol. 17, no. 4, pp. 473–486, Apr. 2009.