Secure Transmission for Nano-Memories using EG-LDPC

Siva Sreeramdas
Asst.Prof.,Dept.of ECE,AITS,
Rajampet,Andhra Pradesh, India.

S.Asif Hussain
Principal,JNTUniversity,
Pulivendula,Andhra Pradesh, India

Dr.M.N.Giri Prasad
M.tech(VLSI),AITS,
Rajampet,Andhra, India.

ABSTRACT
Memory cells have been protected from soft errors for more than a decade; due to the increase in soft error rate in logic circuits, the encoder and decoder circuitry around the memory blocks have become susceptible to soft errors as well and must also be protected. Here introducing a new approach to design fault-secure encoder and decoder circuitry for memory designs. The key novel contribution of this paper is identifying and defining a new class of error-correcting codes whose redundancy makes the design of fault-secure detectors (FSD) particularly simple and further quantify the importance of protecting encoder and decoder circuitry against transient errors. By using that Euclidean Geometry Low-Density Parity-Check (EG-LDPC) codes have the fault-secure detector capability. Using some of the smaller EG-LDPC codes, can tolerate bit or nanowire defect rates of 10% and fault rates of 10^{-18} upsets/device/cycle, achieving a FIT rate at or below one for the entire memory system and a memory density of 10^{11} bit/cm^2 with nanowire pitch of 10 nm for memory blocks of 10 Mb or larger. Larger EG-LDPC codes can achieve even higher reliability and lower area overhead.

***Index Terms*—Decoder, encoder, memory, nanotechnology**

1. INTRODUCTION
Nanotechnology provides smaller, faster, and lower energy devices which allow more powerful and compact circuitry; however, these benefits come with a cost—the nanoscale devices may be less reliable. Thermal- and shot-noise estimations [1], [2] alone suggest that the transient fault rate of an individual nanoscale device (e.g., transistor or nanowire) may be orders of magnitude higher than today's devices ands also combinational logic to be susceptible to transient faults in addition to storage cells and communication channels. Therefore, the paradigm of protecting only memory cells and assuming the surrounding circuitries (i.e., encoder and decoder) will never introduce errors is no longer valid. This paper, introduces a fault-tolerant nanoscale memory architecture which tolerates transient faults **both** in the storage unit and in the supporting logic (i.e., encoder, decoder (corrector), and detector circuitries). Particularly, identify a class of error-correcting codes (ECCs) that guarantees the existence of a simple fault-tolerant detector design. This class satisfies a new, restricted definition for ECCs which guarantees that the ECC codeword has an appropriate redundancy structure such that it can detect multiple errors occurring in both the stored codeword in memory and the surrounding circuitries and this type of error-correcting codes, fault-secure detector capable ECCs (FSD-ECC). By introducing fault-secure detection unit to design a fault-tolerant encoder and corrector by monitoring their outputs. If a detector detects an error in either of these units, that unit must repeat the operation to generate the correct output vector. Using this retry technique, we can correct potential transient errors in the encoder and corrector outputs and provide a fully fault-tolerant memory system.

2. RELATED WORK
Traditionally, memory cells were the only circuitry susceptible to transient faults, and all the supporting circuitries around the memory (i.e., encoders and decoders) were assumed to be fault-free. As a result most of prior work designs for fault-tolerant memory systems focused on protecting only the memory cells and also by continuous scaling down feature sizes or use sub lithographic devices, the surrounding circuitries of the memory system will also be susceptible to permanent defects and transient faults [3].One approach to avoid the reliability problem in the surrounding circuitries is to implement these units with more reliable devices (e.g., more reliable CMOS technologies [4], [5]). However, from an area, performance, and power consumption point of view it is beneficial to implement encoders and decoders with scaled feature size or nanotechnology devices. Consequently, it is important to remove the reliability barrier for these logic circuits so they can be implemented with scaled feature size or nanotechnology devices.

3. SYSTEM OVERVIEW
An overview of proposed reliable memory system is shown in Fig. 1 and is described in the following. The information bits are fed into the encoder to encode the information vector, and the fault secure detector of the encoder verifies the validity of the encoded vector. If the detector detects any error, the encoding operation must be redone to generate the correct codeword. The codeword is then stored in the memory. During memory access operation, the stored codewords will be accessed from the memory unit. Codewords are susceptible to transient faults while they are stored in the memory; therefore a corrector unit is designed to correct potential errors in the retrieved codewords. In our design (see Fig. 1) all the memory words pass through the corrector and any potential error in the memory words will be corrected. Similar to the encoder unit, a fault-secure detector monitors the operation of the corrector unit. All the units shown in Fig. 1 are implemented in fault-prone, nanoscale circuitry; the only component which must be implemented in reliable circuitry are two OR gates that accumulate the syndrome bits for the detectors (shown in Fig. 2). Data bits stay in memory for a number of cycles and, during this period, each memory bit can be upset by a transient fault with certain probability. Therefore, transient errors accumulate in the

memory words over time. In order to avoid accumulation of too many errors in any memory word that surpasses the code correction capability, the system must perform memory scrubbing. Memory scrubbing is the process of periodically reading memory words from the memory, correcting any potential errors, and writing them back into the memory (e.g., [6]).

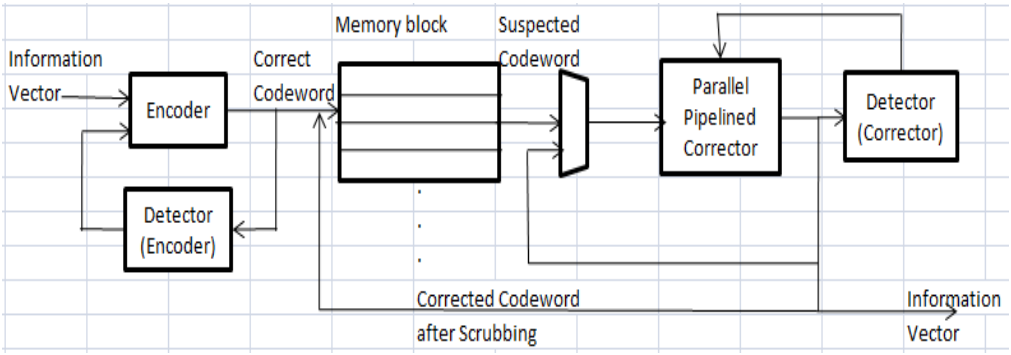

Fig. 1. Overview of proposed system

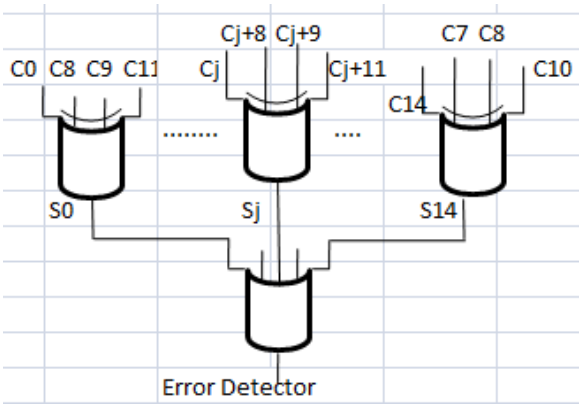

Fig. 2. Fault-secure detector for (15, 7, 5) EG-LDPC code.

4. DESIGN STRUCTURE

The design structure of the encoder, corrector, and detector units of our proposed fault-tolerant memory system is shown below. Before going into the design structure details we start with a brief overview of the sub-lithographic memory architecture model.

4.1. NanoMemory Architecture Model

Here NanoMemory [7],[4]and *NanoPLA [8]* architectures to implement the memory core and the supporting logic, respectively. NanoMemory and NanoPLA are based on nanowire crossbars [9],[10]. the NanoMemory architecture developed in[7],[4] can achieve greater than 10^{11} b/cm^2 density even after including the lithographic-scale address wires and defects. This design uses a nanowire crossbar to store memory bits and a limited number of lithographic scale wires for address and control lines. Fig. 3 shows a schematic overview of this memory structure. The fine crossbar shown in the center of the picture stores one memory bit in each crossbar junction. To be able to write the value of each bit into a junction, the two nanowires crossing that junction must be uniquely selected and an adequate voltage must be applied to them (e.g., [11], [12]). The nanowires can be uniquely selected through the two address decoders located on the two sides of the memory core.

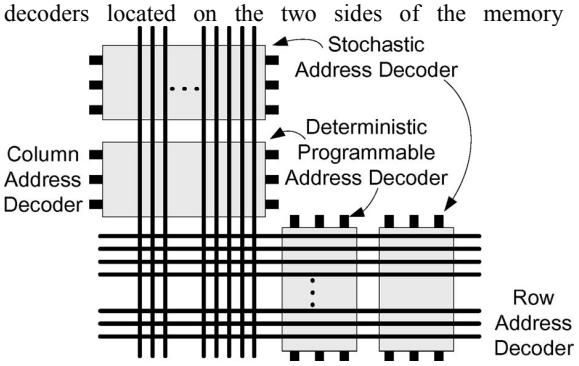

Fig.3. Structure of NanoMemory core

4.2. Banked Memory

Large memories are conventionally organized as sets of smaller memory blocks called banks. The reason for breaking a large memory into smaller banks is to trade off

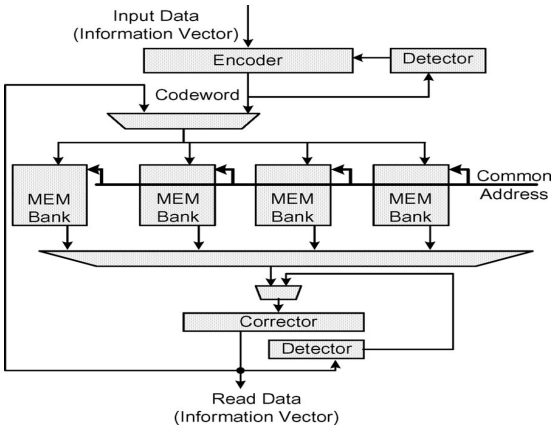

Fig. 4. Banked memory organization

Overall memory density for access speed and reliability. Excessively small bank sizes will incur a large area overhead for memory drivers and receivers. Large memory banks require long rows and columns which results in high capacitance wires that consequently increases the delay. Furthermore long wires are more susceptible to breaks and bridging defects. Therefore excessively large memory banks have high defect rate and low performance. The memory system overview shown in Fig. 1 can be generalized to multiple banks as shown in Fig.4, Fig.5 shows a banked memory organization.

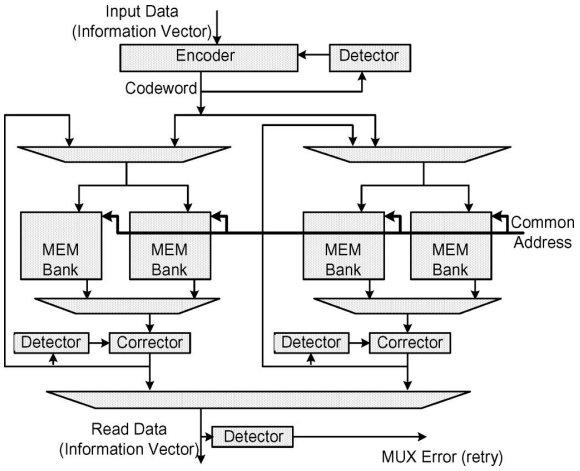

Fig. 5. Banked memory organization with cluster size of 2.

4.3. Fault Secure Detector

The core of the detector operation is to generate the syndrome vector, which is basically implementing the following vector-matrix multiplication on the received encoded vector c and parity-check matrix H: s = c×HT, and therefore each bit of the syndrome vector is the product of the following vector-vector multiply: si = c · hi T , where hi T is the transposed of the ith row of the parity-check matrix. The above product is a linear binary sum over digits of c where the corresponding digit in hi is 1. This binary sum is implemented with an xor gate. Since the row weight of the parity-check matrix is P , to generate one digit of the syndrome vector we need a P-input xor gate, or (P− 1) 2-input xor gates in a tree structure. For the whole detector, it takes n(P − 1) 2-input xor gates. An error is detected if any of the syndrome bits has a nonzero value. The final error detection signal is implemented by an or function of all the syndrome put of this n-input or gate is the error detector signal.

4.4. Encoder

An n-bit codeword, which encodes a n-bit information Vector i is generated by multiplying the n-bit information vector with a kXn bit generator matrix; i.e., c=i.G. EG-LDPC codes are not systematic and the information bits must be decoded from the encoded vector, which is not desirable for our fault-tolerant approach due to the further complication and delay that it adds to the operation. However, these codes are cyclic codes. We used the procedure to convert the cyclic generator matrices to systematic generator matrices for all the EG-LDPC codes under consideration shown in Fig.6.The encoded vector consists of information bits followed by parity bits, where each parity bit is simply an inner product of information vector and a column of X, from G=[I:X]

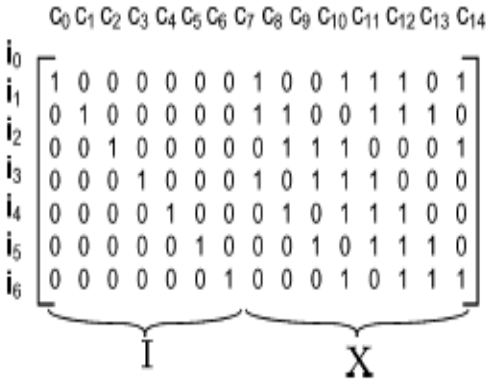

Fig. 6. Generator matrix for the (15, 7, 5) EG-LDPC in systematic format; note the identity matrix in the left columns.

4.5. Corrector

One-step majority-logic correction is a fast and relatively compact error-correcting technique [13]. There is a limited class of ECCs that are one-step-majority correctable which include type-I two-dimensional EG-LDPC.

4.5.1. One-Step Majority-Logic Corrector

One-step majority logic correction is the procedure that identifies the correct value of a each bit in the codeword directly from the received codeword; this is in contrast to the general message-passing error correction strategy (e.g., [14]) which may demand multiple iterations of error diagnosis and trial correction. Avoiding iteration makes the correction latency both small and deterministic. This technique can be implemented serially to provide a compact implementation or in parallel to minimize correction latency. This method consists of two parts: 1) generating a specific set of linear sums of the received vector bits and 2) finding the majority value of the computed linear sums. The majority value indicates the correctness of the code-bit under consideration; if the majority value is 1, the bit is inverted, otherwise it is kept unchanged. The circuit implementing a serial one-step majority logic corrector for (15, 7, 5) EG-LDPC code is shown in Fig. 7.

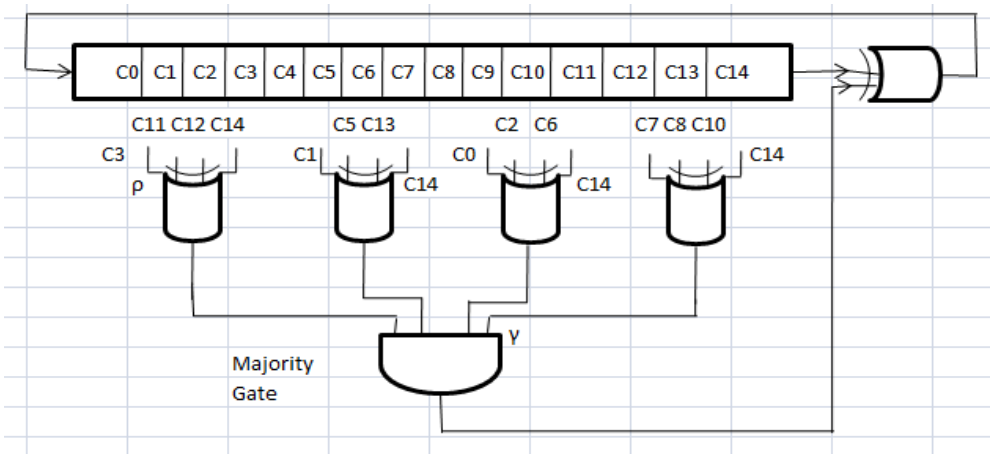

Fig. 7. Serial one-step majority logic corrector structure to correct last bit (bit 14th) of 15-bit (15, 7, 5) EG-LDPC code.

4.5.2. Majority Circuit Implementation

Here majority circuit implementation gate use Sorting Networks the majority gate has application in many other error-correcting codes, and this compact implementation can improve many other applications. We use binary *Sorting Networks* [15] to do the sort operation of the second step efficiently. An -input sorting network is the

Structure that sorts a set of n bits, using 2-bit sorter building blocks. Fig. 8(a) shows a 4-input sorting network. Each of the vertical lines represents one comparator which compares two bits and assigns the larger one to the top output and the smaller one to the bottom [see Fig. 8(b)]. The four-input sorting network, has five comparator blocks, where each block consists of two two-input gates; overall the four-input sorting network consists of ten two-input gates in total.

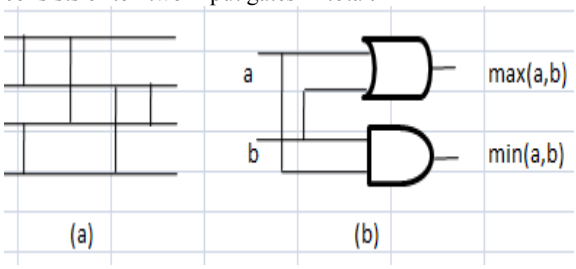

Fig. 8. (a) Four-input sorting network; each vertical line shows a one-input comparator. (b) One comparator structure.

4.5.3. Serial Corrector

As mentioned earlier, the same one-step majority-logic corrector can be used to correct all the n bits of the received codeword of a cyclic code. To correct each code-bit, the received encoded vector is cyclic shifted and fed into to the XOR gates as shown in Fig. 7. The serial majority corrector takes cycles to correct an erroneous codeword. If the fault rate is low, the corrector block is used infrequently; since the common case is error-free codewords, the latency of the corrector will not have a severe impact on the average memory read latency. The serial corrector must be placed off the normal memory read path. This is shown in Fig. 9. The memory words retrieved from the memory unit are checked by detector unit. If the detector detects an error, the memory word is sent to the corrector unit to be corrected, which has the latency of the detector plus the round latency of the corrector.

4.5.4. Parallel Corrector

For high error rates, the corrector is used more frequently and its latency can impact the system performance. Therefore we can implement a parallel one-step majority corrector which is essentially copies of the single one-step majority-logic corrector. Fig. 1 shows a system integration using the parallel corrector. All the memory words are pipelined through the parallel corrector.

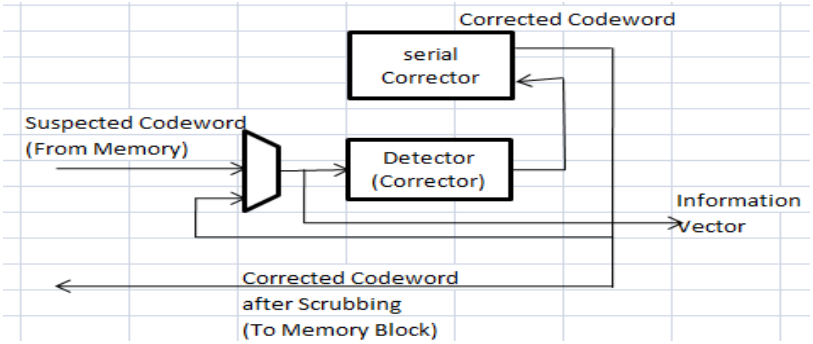

Fig. 9. Partial system overview with serial corrector.

This way the corrected memory words are generated every cycle. The detector in the parallel case monitors the operation of the corrector, if the output of the corrector is erroneous; the detector signals the corrector to repeat the operation. Note that faults detected in a nominally corrected memory word arise solely from faults in the detector and corrector circuitry and not from faults in the memory word. Since detector and corrector circuitry are relatively small compared to the memory system, the failure rate of these units is relatively low.

5. SYSTEM ANALYSIS

5.1. Performance Analysis

Here high scrubbing rates will decrease the time spent correcting data at the cost of additional cycles lost to scrubbing. As you can see in Figure 1, when there is no error in the memory word, the memory words are pipelined through the detector and therefore we can read one word per cycle without any throughput loss. If we select our memory word size and scrubbing rate appropriately, the vast majority of read operations can take this fast path.

however, when the detector observers an error, the memory word must pass through the corrector to be corrected. The total number of cycles that will be lost is equal to the latency of the corrector and the detector for our serial corrector design, the main latency is due to the corrector. For larger codes, where the serialized corrector can take a long time, multiple copies of the corrector can be used to reduce the throughput loss. longer scrubbing intervals increase the number of errors accumulated in the memory and therefore more retrieved memory words have to go through the correction operation.

5.2. Area and Reliability Analysis

Analyze the area overhead of the fault tolerant memory system using the banking structure as described in the above section4. Fig. 10 plots the total area per bit for different memory sizes, when the upset rate is 10^{-28} errors/device/cycle. The area of the memory banks are computed following the area model provided in [4]. The area of the supporting units (encoder, corrector, and detector) is computed using the area model of *NanoPLA*.

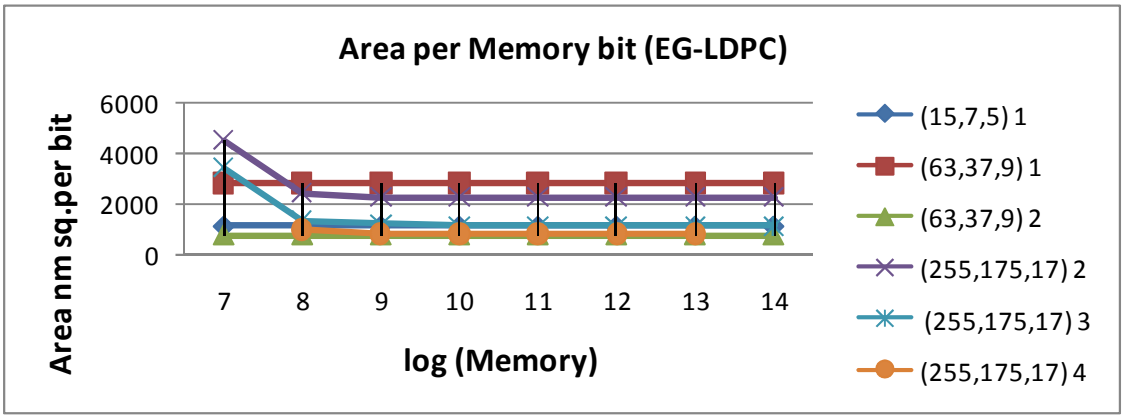

Fig. 10. Area of the memory system versus the memory size

Fig. 11 plots the reliability of the systems that satisfy the throughput loss limit and reliability limit while achieving the minimum area overhead. All of the previous design points are for memory size of 10^{12} bits. For these Calculations we assume a memory unit with the following parameters: lithographic wire pitch of 105 nm (45 nm node [1]), nanowire pitch of 10 nm, defect rate of 0.01 per memory junction, and memory bank size of bits.

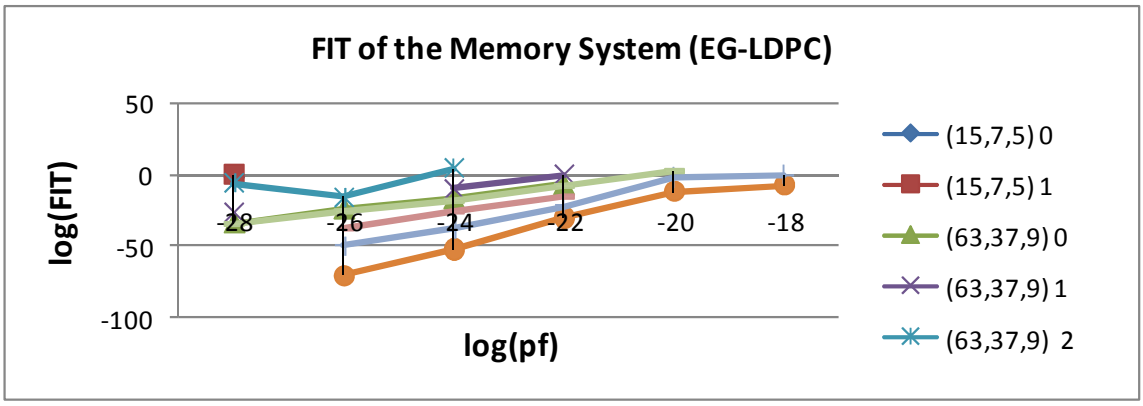

Fig. 11. FIT of EG-LDPC codes for a system with 10^{12} memory bits, memory bank size of 1 Mb, system frequency of 1 GHz, and the defect rate of 1%

6. CONCLUSION

This paper work, presented a fully fault-tolerant memory system that is capable of tolerating errors not only in the memory bits but also in the supporting logic including the ECC encoder and corrector. Euclidean Geometry codes are part of a new subset of ECCs that have FSDs. Using these FSDs we design a fault-tolerant encoder and corrector, where the fault-secure detector monitors their operation. We also presented a unified approach to tolerate permanent defects and transient faults. This unified approach reduces the area overhead.

7. ACKNOWLEDGMENT

This work is supported by Annamacharya Institute of Technology and Sciences. And thanks for valuable references provided by the authors. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect their views.

8. REFERENCES

[1] M. Forshaw, R. Stadler, D. Crawley, and K. Nikolic´, "A short review of nanoelectronic architectures," *Nanotechnology*, vol. 15, pp.S220–S223, 2004.

[2] J. Kim and L. Kish, "Error rate in current-controlled logic processors with shot noise," *Fluctuation Noise Lett.*, vol. 4, no. 1, pp. 83–86, 2004.

[3] S. Hareland, J. Maiz, M. Alavi, K. Mistry, S. Walsta, and C. Dai, "Impact of CMOS process scaling and SOI on the soft error rates of logic processes," in *Proc. Symp. VLSI*, 2001, pp. 73–74.

[4] A. DeHon, S. C. Goldstein, P. J. Kuekes, and P. Lincoln, "Non-photolithographic nanoscale memory density prospects," *IEEE Trans.Nanotechnol.*, vol. 4, no. 2, pp. 215–228, Feb. 2005.

[5] F. Sun and T. Zhang, "Defect and transient fault-tolerant system design for hybrid CMOS/nanodevice digital memories," *IEEE Trans. Nanotechnol.*,vol. 6, no. 3, pp. 341–351, Jun. 2007.

[6] A. Saleh, J. Serrano, and J. Patel, "Reliability of scrubbing recoverytechniques for memory systems," *IEEE Trans. Reliab.*, vol. 39, no. 1,pp. 114–122, Jan. 1990.

[7] A. DeHon, "Deterministic addressing of nanoscale devices assembled at sublithographic pitches," *IEEE Trans. Nanotechnol.*, vol. 4, no. 6,pp. 681–687, 2005.

[8] A. DeHon, "Nanowire-based programmable architectures," *ACM J. Emerging Technol. Comput. Syst.*, vol. 1, no. 2, pp. 109–162, 2005.

[9] Y. Chen, G.-Y. Jung, D. A. A. Ohlberg, X. Li, D. R. Stewart, J.O. Jeppesen, K. A. Nielsen, J. F. Stoddart, and R. S. Williams,

"Nanoscale molecular-switch crossbar circuits," *Nanotechnology*, vol. 14, pp. 462–468, 2003.

[10] J. E. Green, J. W. Choi, A. Boukai, Y. Bunimovich, E. Johnston- Halperin, E. DeIonno, Y. Luo, B. A. Sheriff, K. Xu, Y. S. Shin,H.-R. Tseng, J. F. Stoddart, and J. R. Heath, "A 160-kilobit molecular electronic memory patterned at 10^{11} bits per square centimeter,"*Nature*, vol. 445, pp. 414–417, Jan. 25, 2007

[11] Y. Chen, D. A. A. Ohlberg, X. Li, D. R. Stewart, R. S. Williams,J. O. Jeppesen, K. A. Nielsen, J. F. Stoddart, D. L. Olynick, and E.Anderson, "Nanoscale molecular-switch devices fabricated by imprint lithography," *Appl. Phys. Lett.*, vol. 82, no. 10, pp. 1610–1612, 2003.

[12] D. R. Stewart, D. A. A. Ohlberg, P. A. Beck, Y. Chen, R. S.Williams, J.O. Jeppesen, K. A. Nielsen, and J. F. Stoddart, "Molecule-independent electrical switching in pt/organic monolayer/ti devices," *Nanoletters*,vol. 4, no. 1, pp. 133–136, 2004.

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

[14] M. Sipser and D. Spielman, "Expander codes," *IEEE Trans. Inf.Theory*, vol. 42, no. 6, pp. 1710–1722, Nov. 1996

[15] D. E. Knuth, *The Art of Computer Programming*, 2nd ed. Reading,MA: Addison Wesley, 2000.