ABHINAV
NATIONAL MONTHLY REFEREED JOURNAL OF REASEARCH IN SCIENCE & TECHNOLOGY
www.abhinavjournal.com

FAULT SECURE ENCODER AND DECODER FOR NANO MEMORY APPLICATIONS

Tajuddin Sayyad

Student M.Tech., Qubq College of Engineering & Technology, India
Email: imtiyaz_nimra@yahoo.co.in

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. We introduce a new approach to design fault-se- cure 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 de- sign of fault-secure detectors (FSD) particularly simple. We fur- ther quantify the importance of protecting encoder and decoder circuitry against transient errors, illustrating a scenario where the system failure rate (FIT) is dominated by the failure rate of the en- coder and decoder. We prove that Euclidean Geometry Low-Den- sity Parity-Check (EG-LDPC) codes have the fault-secure detector capability. Using some of the smaller EG-LDPC codes, we can tol- erate bit or nanowire defect rates of 10% and fault rates of upsets/device/cycle, achieving a FIT rate at or below one for the entire memory system and a memory density of bit/cm 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.

INTRODUCTION AND MOTIVATION

NANOTECHNOLOGY provides smaller, faster, and lower energy devices which allow more powerful and compact circuitry; however, these benefits come with a cost—the Nano scale devices may be less reliable. Thermal- and shot-noise estimations, alone suggest that the transient fault rate of an individual nano scale device (e.g., transistor or Nano wire) may be orders of magnitude higher than today's devices. As a result, we can expect 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only memory cells and assuming the surrounding circuitries (i.e., encoder and decoder) will never introduce errors is no longer valid.

In this paper, we introduce a fault-tolerant Nano scale 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, we 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.

The novel contributions of this paper include the following:

1. A mathematical definition of ECCs which have simple FSD which do not requiring the addition of further redundancies in order to achieve the fault-secure property;
2. Identification and proof that an existing LDPC code (EGLDPC) has the FSD property;
3. a detailed ECC encoder, decoder, and corrector design that can be built out of fault-prone circuits when protected by this fault-secure detector also implemented in fault-prone circuits and guarded with a simple OR gate built out of reliable circuitry.

To further show the practical viability of these codes, we work through the engineering design of a Nano scale memory system based on these encoders and decoders including the following:

- Memory banking strategies and scrubbing;
- Reliability analysis;
- Unified ECC scheme for both permanent memory bit defects and transient upsets.

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. However, as we continue 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.

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). 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. Almost all of the proposed fault tolerant encoders and decoders so far, use the conventional fault tolerant scheme (e.g., logic replication or concurrent parity prediction) to protect the encoder and corrector circuitry. That is, they add additional logic to check the correctness of the circuit calculation. In contrast, the technique introduced in this work exploits the existing structure of the ECC to guarantee the fault-secure property of the detector unit without adding redundant computations. The work presented, is an example of the scheme using redundancy to generate fault tolerant encoder.

This project develops a fault-secure encoder unit using a concurrent parity prediction scheme. Like the general parity-prediction technique, concurrently generates (predicts) the parity-bits of the encoder outputs (encoded bits) from the encoder inputs (information bits).

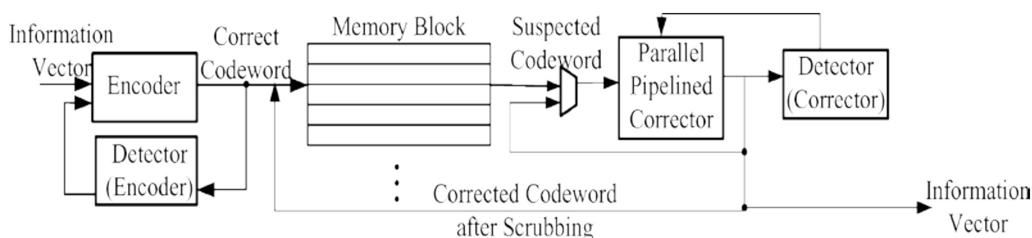
The predicted parity bits are then compared against the actual parity function of the encoder output (encoded bits) to check the correctness of the encoder unit. The parity predictor circuit implementation is further optimized for each ECC to make a more compact design.

System Overview

In this section, we outline our memory system design that can tolerate errors in any part of the system, including the storage unit and encoder and corrector circuits using the fault-secure detector. For a particular ECC used for memory protection, let be the maximum number of error bits that the code can correct and be the maximum number of error bits that it can detect, and in one error combination that strikes the system, let , , and be the number of errors in encoder, a memory word, and corrector, and let and be the number of errors in the two separate detectors monitoring the encoder and corrector units. In conventional designs, the system would guarantee error correction as long as and. In contrast, here we guarantee that the system can correct any error combination as long as, and.

This design is feasible when the following two fundamental properties are satisfied:

1. Any single error in the encoder or corrector circuitry can at most corrupt a single codeword bit (i.e., no single error can propagate to multiple codeword bits);
2. There is a fault secure detector that can detect any combination of errors in the received codeword along with errors in the detector circuit. This fault-secure detector can verify the correctness of the encoder and corrector operation.



The first property is easily satisfied by preventing logic sharing between the circuits producing each codeword bit or information bit in the encoder and the corrector respectively. In Section IV, we define the requirements for a code to satisfy the second property. An overview of our proposed reliable memory system is shown in Fig 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 code words will be accessed from the memory unit. Code words 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 code words. In our design 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. are implemented in fault-prone, Nano scale circuitry; the only component which must be implemented in reliable circuitry are two OR gates that accumulate the syndrome bits for the detectors.

ECCs with Fault Secure Detector

In this section, we present our novel, restricted ECC definition for our fault-secure detector capable codes. Before starting the details of our new definition we briefly review basic linear ECCs.

A. Error- Correcting Code Reviews:

Let i = (i_0, $i_1 \ldots i_{k-1}$) be the k-bit information vector that will be encoded into an n-bit code word, C = (C_0, $C_1 \ldots C_{K-1}$). For linear codes, the encoding operation essentially performs the following vector-matrix multiplication:

$$C = i. G$$

Where G id a kxn generator matrix. The validity of a received encoded vector can be checked with the parity-check matrix, which is an (n-k) x n binary matrix named H. The checking or detecting operation is basically summarized as the following vector-matrix multiplication:

$$S = C. H^T$$

The (n-k) bit vector S is called the SYNDROME vector. A syndrome vector is zero if c is a valid code word, and non zero if C is an erroneous code word. Each code is uniquely specified by its generator matrix or parity-check matrix.

A code is a systematic code if every codeword consists of the original -bit information vector followed by n – k parity bits. With this definition, the generator matrix of a systematic code must have the following structure:

$$G = [I : X]$$

where I is a k x k identity matrix and X is a k x (n-k) matrix that generates the parity-bits. The advantage of using systematic codes is that there is no need for a decoder circuit to extract the information bits. The information bits are simply available in the first k bits of any encoded vector.

A code is said to be a cyclic code if for any codeword C, all the cyclic shifts of the codeword are still valid code words. A code is cyclic iff the rows of its parity-check matrix and generator matrix are the cyclic shifts of their first rows.

The minimum distance of an ECC, , is the minimum number of code bits that are different between any two code words. The maximum number of errors that an ECC can detect is d-1, and the maximum number that it corrects is [d/2]. Any ECC is represented with a triple (n, k, d), representing code length, information bit length, and minimum distance, respectively.

FSD-ECC Definition

The restricted ECC definition which guarantees a FSD-ECC is as follows.

Definition I: Let C be an ECC with minimum distance d. C is FSD-ECC if it can detect any combination of overall d-1 or fewer errors in the received codeword and in the detector circuitry.

Theorem I: Let be an ECC, with minimum distance d. C is FSD-ECC if any error vector of weight 0<e<d-1, has syndrome vector of weight at least d-e.

Note: The following proof depends on the fact that any single error in the detector circuitry can corrupt at most one output (one syndrome bit). This can be easily satisfied for any circuit by implementing the circuit in such a way that no logic element is shared among multiple output bits; therefore, any single error in the circuit corrupts at most one output (one syndrome bit).

Proof: The core of a detector circuitry is a multiplier that implements the vector-matrix multiply of the received vector and the parity-check matrix to generate the syndrome vector. Now if e errors strike the received codeword the syndrome weight of the error pattern is at least from the assumption. Furthermore, the maximum number of tolerable errors in the whole system is and errors already exist in the encoded vector, therefore the maximum number of errors that can strike in the detector circuitry is . From the previous note, these many errors can corrupt at most syndrome bit, which in worst case leaves at least one nonzero syndrome bit and therefore detects the errors

The difference between FSD-ECC and normal ECC is simply the demand on syndrome weight. That is, for error vector of weight, a normal ECC demands nonzero syndrome weight while FSD-ECC demands syndrome weight of e > 0, a normal ECC demands nonzero syndrome weight while FSD-ECC demand syndrome weight of > d-e.

FSD-ECC Example: Euckidean Geometry Codes

A. Euclidean Geometry Code Review

This section reviews the construction of Euclidean Geometry codes based on the lines and points of the corresponding finite geometries [27]. Euclidean Geometry codes are also called EG-LDPC codes based on the fact that they are low-density parity-check (LDPC) codes [14]. LDPC codes have a limited number of 1's in each row and column of the matrix; this limit guarantees limited complexity in their associated detectors and correctors making them fast and light weight [9].

Let EG be a Euclidean Geometry with points and J lines. EG is a finite geometry that is shown to have the following fundamental structural properties:

1. every line consists of points;
2. any two points are connected by exactly one line;
3. every point is intersected by lines;
4. Two lines intersect in exactly one point or they are parallel; i.e., they do not intersect.

Let H be a Jxn binary matrix, whose rows and columns corresponds to lines and points in an EG Euclidean geometry, respectively, where h_{ij} = 1 if and only if the th line of EG contains the th point of EG, and h_{ij}=0 otherwise. A row in H displays the points on a specific line of and has weight ρ. A column H in displays the lines that intersect at a specific point in EG and has weight γ. The rows of are called the incidence vectors of the lines in EG, and the columns of H are called the intersecting vectors of the points in EG. Therefore, H is the

incidence matrix of the lines in over the points in EG. It is shown in that H is a LDPC matrix, and therefore the code is an LDPC code.

A special subclass of EG-LDPC codes, type-I 2-D EG-LDPC, is considered here. It is shown in [15] that type-I 2-D EG-LDPC have the following parameters for any positive integer **t > 2**:

- information bits, $K = 2^{2t} - 3^{t}$;
- length, $n = 2^{2t-1}$;
- minimum distance, $d_{min} = 2^{t} + 1$;
- dimensions of the parity-check matrix, nxn;
- row weight of the parity-check matrix, $\rho = 2^{t}$;
- column weight of the parity-check matrix, $\gamma = 2^{t}$.

It is important to note that the rows of H are not necessarily linearly independent, and therefore the number of rows do not necessarily represents the rank of the H matrix. The rank of H is which makes the code of this matrix (n,k) linear code.

B. FSD-ECC Proof for EG-LDPC

In this section, we prove that EG-LDPC codes have the FSD-ECC property.

Theorem II: Type-I 2-D EG-LDPC codes are FSD-ECC.

Proof: Let C be an EG-LDPC code with column weight γ and minimum distance .We have to show that any error vector of weight $0 < e < d-1$ corrupting the received encoded vector has syndrome vector of weight at least d-e.

TABLE I
EG-LDPCs AND UPPER AND LOWER BOUNDS ON CODE LENGTH

Hamming bound	EG-LDPC	Gilbert-Varshamov bound
(14,7,5)	(15,7,5)	(17,7,5)
(58,37,9)	(63,37,9)	(67,37,9)
(222,175,17)	(255,175,17)	(255,175,17)

Now a specific bit in the syndrome vector will be one if and only if the parity-check sum corresponding to this syndrome vector has an odd number of error bits present in it. Looking from the Euclidean geometry perspective, each error bit corresponds to a point in the geometry and each bit in the syndrome vector corresponds to a line. Consequently, we are interested in obtaining a lower bound on the number of lines that pass through an odd number of error points.

We further lower bound this quantity by the number of lines that pass through exactly one of the error points. Based on the definition of the Euclidean geometry, lines pass through each point; so error points potentially impact lines. Also at most one line connects two points.

Therefore, looking at the e error points, there are at most (e/2) lines between pairs of error points. Hence, the number of lines passing through a collection of these points is lower bounded by γe-(e/2).

Out of this number, at most lines connect two or more points of the error points. Summarizing all this, the number of lines passing through exactly one error point, which gives us the lower bound on the syndrome vector weight, is at least ***γe-2(e/2)***.

From the code properties introduced in this Section and knowing that ***d= γ+1***, we can derive the following inequality:

$$|s(c_e)| \geq \gamma e - 2\binom{e}{2} = e(\gamma + 1 - e) = e(d - e)$$
$$\geq d - e \quad \text{when} \quad e > 0$$

The previous inequality says that the weight of the syndrome vector of a codeword with errors is at least when which is the required condition of Theorem (I). Therefore, EG-LDPC is FSD-ECC.

C. Efficiency of EG-LDPC

It is important to compare the rate of the EG-LDPC code with other codes to understand if the interesting properties of low-density and FSD-ECC come at the expense of lower code rates. We compare the code rates of the EG-LDPC codes that we use here with an achievable code rate upper bound (Gilbert-Varshamov bound) and a lower bound (Hamming bound). Table I shows the upper and lower bounds on the code overhead, for each of the used EG-LDPC.

Design Structure

In this section, we provide the design structure of the encoder, corrector, and detector units of our proposed fault-tolerant memory system. We also present the implementation of these units on a sub-lithographic, Nano wire-based substrate. Before going into the design structure details we start with a brief overview of the sub-lithographic memory architecture model.

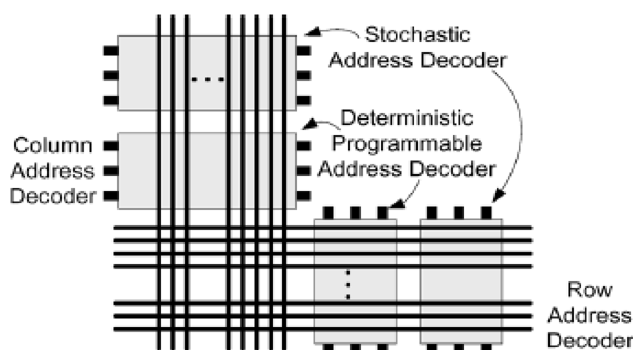


Fig. 2. Structure of NanoMemory core. Please note that the address decoders are for memory access and are not related to the memory ECC decoders.

A. Nano Memory Architecture Model

We use the Nano Memory and Nano PLA architectures to implement the memory core and the supporting logic, respectively. Nano Memory and Nano PLA are based on nanowire crossbars.

Fig. 2 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. The nanowires can be uniquely selected through the two address decoders located on the two sides of the memory core. The detail of the NanoMemory structure is presented. For our design, we revise the original NanoMemory structure introduced. Instead of using a lithographic- scale interface to read and write into the memory core, we use a nanowire-based interface. The reason that we can remove the lithographic-scale interface is that all the blocks interfacing with the memory core (encoder, corrector and detectors) are implemented with nanowire-based crossbars. So we use a nanowire-based DEMUX to connect the memory core to the supporting logic blocks. The detail of the DEMUX structure is available.

B. 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 and parity-check matrix H:

$$S = C.H^T$$

Therefore each bit of the syndrome vector is the product of with one row of the parity-check matrix. This product is a linear binary sum over digits of C where the corresponding digit in the matrix row is 1. This binary sum is implemented with an XOR gate. Fig. 3 shows the detector circuit for the EG-LDPC code. Since the row weight of the parity-check matrix is , to generate one digit of the syndrome vector we need a ρ-input XOR gate, or (ρ-1) 2-input XOR gates.

For the whole detector, it takes n (ρ-1) 2-input XOR gates. Table II illustrates this quantity for some of the smaller EG-LDPC codes. Note that implementing each syndrome bit with a separate XOR gate satisfies the assumption of Theorem I of no logic sharing in detector circuit implementation.

This n-input wire-OR is much smaller than implementing the entire n x (ρ-1) 2-input XORs at the lithographic scale. The area of each detector is computed using the model of Nano PLA and NanoMemory accounting for all the supporting lithographic wires and reported in Table III.

Fig. 4 shows the implementation of the detector on a Nano PLA substrate. The framed block in Fig. shows a -input XOR gate, implementing a -level XOR tree in spiral form. The solid boxes display the restoration planes and the white boxes display the wired-OR planes of Nano PLA architecture model. In Fig. 4, the signals rotate counter clock-wise, and each round of signal generates the XOR functions of one level of the XOR-tree. The final output then gates a robust lithographic-scale wire.

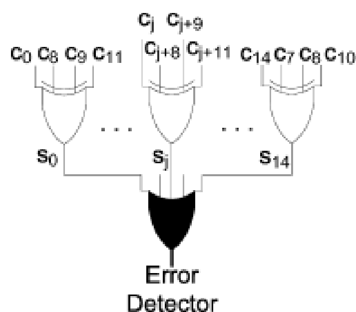


Fig. 3. Fault-secure detector for (15, 7, 5) EG-LDPC code. All the gates except the last OR gate are implemeneted with fault-prone nanoscale circuitry. The last OR gate is implemented with more reliable lithography technique.

This lithographic-scale wire generates a wired-OR function of all the *n ρ-input* XORs and is the final output of the detector circuit. The XOR gate is the main building of the encoder and corrector as well.

TABLE II
DETECTOR, ENCODER, AND CORRECTOR CIRCUIT AREA IN THE NUMBER OF 2-INPUT GATES

Code	(15,7,5)	(63,37,9)	(255,175,17)
Detector	45	501	3825
Encoder	22	355	6577
Serial Corrector	19	83	331
Parallel Corrector	285	5229	84405

C. Encoder

An -bit codeword , which encodes a -bit information vector is generated by multiplying the -bit information vector with a bit generator matrix ; i.e., . 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 presented to convert the cyclic generator matrices to systematic generator matrices for all the EG-LDPC codes under consideration.

	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}
i_0	1	0	0	0	0	0	0	1	0	0	1	1	1	0	1
i_1	0	1	0	0	0	0	0	1	1	0	0	1	1	1	0
i_2	0	0	1	0	0	0	0	0	1	1	1	0	0	0	1
i_3	0	0	0	1	0	0	0	1	0	1	1	1	0	0	0
i_4	0	0	0	0	1	0	0	0	1	0	1	1	1	0	0
i_5	0	0	0	0	0	1	0	0	0	1	0	1	1	1	0
i_6	0	0	0	0	0	0	1	0	0	0	1	0	1	1	1

I X

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

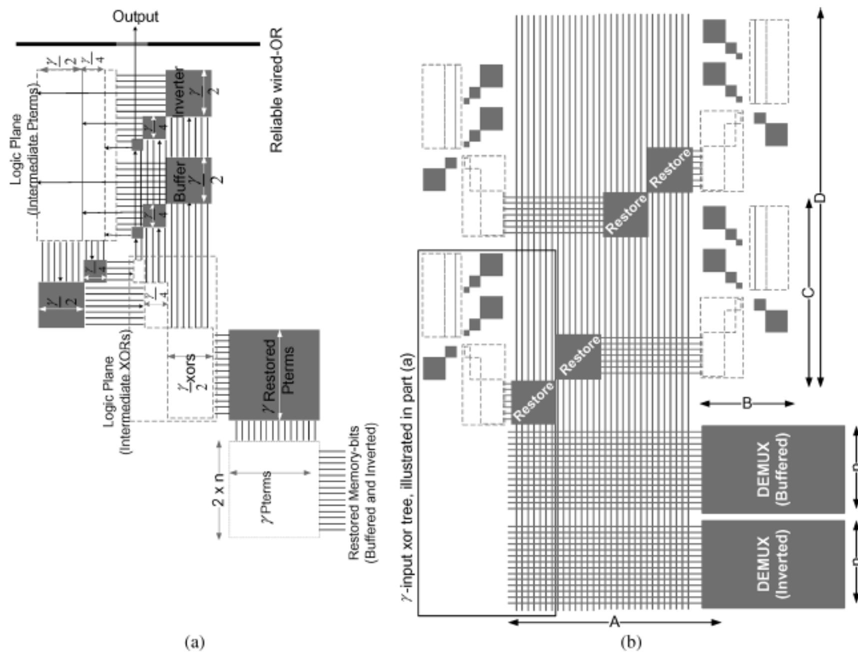


Fig. 4. (a) γ-input XOR tree implemented on NanoPLA structure. (b) Detector circuit implemented on NanoPLA: The parameters in the figure are $A = n \times \gamma$, $B = 2 \times Pl(\gamma - 1)$, $C = Pl(2 \times \gamma - 2) + 2 \times Pl(\gamma - 1)$, and $D = n/2 \times C$, where $Pl(x)$, is the width of a NanoPLA plane with x nanowire, including the area of the supporting lithographic scale wires.

Basic Idea To Implement The Eg-Ldpc Algorithm:

Fig. 5 shows the systematic generator matrix to generate EG-LDPC code. 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 shows the encoder circuit to compute the parity bits of the EG-LDPC code. In this figure i = (i_0... i_6) is the information vector and will be copied to C_0...C_6 bits of the encoded vector, , and the rest of encoded vector, the parity bits, are linear sums (XOR) of the information bits.

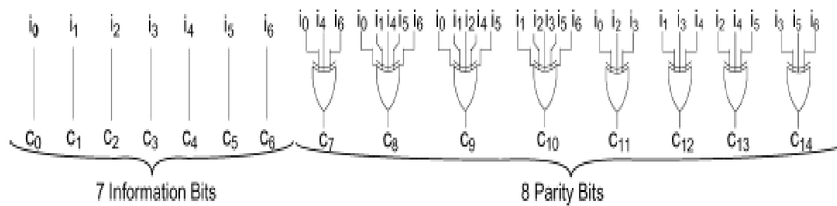


Fig. 6. Structure of an encoder circuit for the (15, 7, 5) EG-LDPC code; i_0 to i_6 are 7-bit information vector. Each of the XOR gates generate one parity bit of the encoded vector. The codeword consists of seven information bits followed by eight parity bits.

If the building block is two-input gates then the encoder circuitry takes 22 two-input XOR gates. Table II shows the area of the encoder circuits for each EG-LDPC codes under consideration based on their generator matrices. Once the XOR functions are known, the encoder structure is very similar to the detector structure shown in Fig. 4, except it consists

of *(n-k)* XOR gates of varying numbers of inputs. Each nanowire-based XOR gate has structure similar to the XOR tree shown in Fig. 4.

D. Corrector

One-step majority-logic correction is a fast and relatively compact error-correcting technique. There is a limited class of ECCs that are one-step-majority correctable which include type-I two-dimensional EG-LDPC. In this section, we present a brief review of this correcting technique. Then we show the one-step majority-logic corrector for EG-LDPC codes.

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 theory behind the one-step majority corrector and the proof that EG-LDPC codes have this property are available. Here we overview the structure of such correctors for EG-LDPC codes.

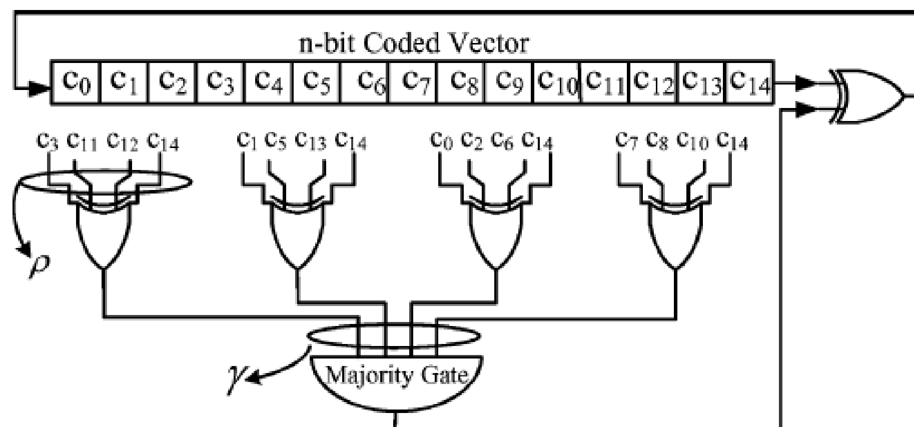


Fig. 7. Serial one-step majority logic corrector structure to correct last bit (bit 14th) of 15-bit (15,7,5) EG-LDPC code.

These steps correct a potential error in one code bit let's say, e.g., e_{n-1}

1. Generate γ parity-check sums by computing the inner product of the received vector and the appropriate rows of parity-check matrix.
2. The γ check sums are fed into a majority gate. The output of the majority gate corrects the bit by e_{n-1} inverting the value of e_{n-1} if the output of majority gate is "1".

The circuit implementing a serial one-step majority logic corrector for (15, 7, 5) EG-LDPC code is shown in Fig. 7. This circuit generates γ parity-check sums with γ XOR gates and then computes the majority value of the parity-check sums. Since each parity-check sum is

computed using a row of the parity check matrix and the row density of EG-LDPC codes are ρ, each XOR gate that computes the linear sum has ρ inputs. The single XOR gate on the right of Fig. 7 corrects the code bit e_{n-1} using the output of the majority gate. Once the code bit is corrected the codeword is cyclic shifted and code bit e_{n-2} is placed at position and will be corrected. The whole codeword can be corrected in n rounds.

If implemented in flat, two-level logic, a majority gate could take exponential area. The two-level majority gate is implemented by computing all the ([γ+1/2] product terms that have ([γ+1/2] ON inputs and one ([γ+1/2] -input OR-term. For example, the majority of 3 inputs a, b, c, is computed with product terms and one 3-input OR-terms as follows:

Majority (a, b, c) = ab+ ac+ bc

In the next section we presents a novel, compact implementation of majority circuits.

2) Majority Circuit Implementation: Here we present a compact implementation for the majority gate using Sorting Networks. The majority gate has application in many other error-correcting codes, and this compact implementation can improve many other applications.

A majority function of binary digits is simply the median of the digits (where we define the median of an even number of digits as the smallest digit).

To find the median of the inputs, we do the following:

1. divide the γ inputs into two halves with size γ/2 ;
2. sort each of the halves;
3. the median is 1 if for i = 1,2,...,γ/2 the ith element of one half and the $(\gamma/2 + 1-i)^{th}$ element of the other half are both 1.

To check the condition in the third step, we use γ/2 two-input AND gates followed by a γ/2 -input OR gate. Fig. 9 shows the circuit implementing the above technique to find the median value of 8 bits. It has two -input (four-input) sorting networks followed by combinational circuitry, consisting of four two-input AND gates and a four-input OR gate, which can be implemented with three two-input OR gates. Therefore in total an eight-input majority gate implemented with sorting networks take 27 two-input gates; in contrast, the two-level implementation of this majority gate takes five-input AND gates and one 56-input OR gate.

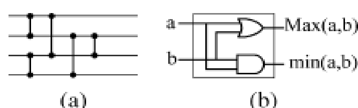


Fig. 8. (a) Four-input sorting network; each vertical line shows a one-input comparator. (b) One comparator structure.

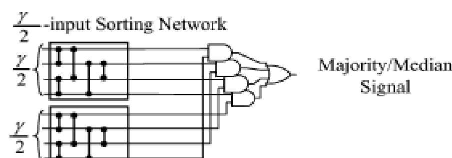


Fig. 9. Eight-input majority gate using sorting network.

ABHINAV
NATIONAL MONTHLY REFEREED JOURNAL OF REASEARCH IN SCIENCE & TECHNOLOGY
www.abhinavjournal.com

3) Serial Corrector:

As mentioned earlier, the same one-step majority-logic corrector can be used to correct all the 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 *n* 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 code words, the latency of the corrector will not have a severe impact on the average memory read latency.

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 ***n*** 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. 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.

Assuming our building blocks are two-input gates, number of -input parity-check sums will require γ x (ρ-1) two-input XOR gates. The size of the majority gate is defined by the sorting network implementation. Table II shows the overall area of a serial one-step majority-logic corrector in the number of two-input gates for the codes under consideration. The parallel implementation consists of exactly copies of the serial one-step majority- logic corrector.

Generating the linear binary sums (XORs) of the one-step majority sum is the same as Fig. 4. The majority gate is simply computed following the structure shown in Fig. 9 using the nano wire-based substrate.

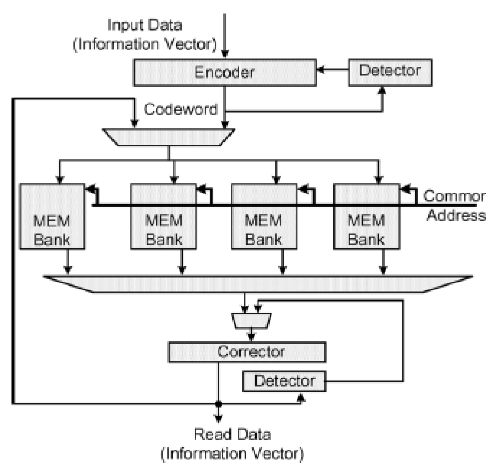


Fig. 11. Banked memory organization, with single global corrector.

E. 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 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 organization of Nano Memory is not different from the conventional memory organization, except that the overhead

per bank is larger due to the scale difference between the size of a memory bit (a single wire crossing) and the support structures (e.g., micro scale wires for addressing and bootstrapping).

A. Analysis

We assume the fault probability of each device at each cycle (P_F) has i.i.d. and random distribution over the devices of the memory system. Recall e_e and e_{de} are the nominal number of errors that occurs in encoder and encoder detector during memory write operation at one instance. Similarly, e_m ,e_c , e_{dc} and are the number of errors that occur in a memory word and its corresponding corrector and detector. Let n_c ,n_e ,n_d and be the size of the circuitry involved in an operation on a single code bit in the encoder, corrector, or detector, respectively. This is the size of the logic cone of a single output of each of the above units. For example, in a detector each logic cone is a -input XOR gate generating a single bit of the syndrome vector

Let a nominal unit have a logic cone size of x (x may be one of the values n_e , n_c , n_d). With worst-case analysis the output of the logic cone fails when any of the devices in the logic cone fails. So when at least one of the *x* devices inside the cone is erroneous, the output of the logic cone, which is a code bit, would be erroneous. Therefore, the probability that a code bit is erroneous in any of the above units is $P_{bit_circuit} = 1-(1-P_f)^x$. Similarly the probability that a memory-bit is erroneous in scrubbing interval of cycles is

$$P_{bit_mem} = 1-(1-P_f)^{xs}$$

where *x* is the number of devices contained in one memory cell. When using 6 T-SRAM cells, $x=6$. When using a Nano Memory, the memory bit is essentially a single nano wire cross point. However, since accessing each memory bit requires

reading the signal value through a pair of nano wires, the correctness of each memory bit depends on the correctness of two nano wires. Therefore, for the Nano Memory design, $x=2$. Each unit or a memory-word experience error among *n* bits of the codeword with the probability

$$P_{unit} = (^{n}{}_{e})\, P_{bit}\, (1 - P_{bit})^{n-e}$$

which is simply a binomial distribution; is the code-length, P_{bit} is either P_{bit_mem} or $P_{bit_circuit}$, and e is e_e, e_c, e_m, e_{de}, or e_{dc} .

As explained in the above Section, errors in the encoder unit are detected by its following detector, and are corrected by repeating the encoding operation to generate a correct encoded vector. The detector can detect up to d-1 errors overall in these two units. Where is the code distance. With worst-case assumptions, the detector fails to detect the errors if there are more than d-1 errors.

SUMMARY

In this paper, we 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. We used Euclidean Geometry codes. We proved that these 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 monitor their operation. We also presented a unified approach to tolerate permanent defects and transient faults. This unified approach reduces the area overhead. Without this technique to tolerate errors in the ECC logic, we would required reliable (and conse- quently lithographic scale) encoders and decoders. Accounting for all the above area overhead factors, all the codes considered here achieve memory density of 20 to 100 Gb/nm^2, for large enough memory ($\geq$0.1 Gb).

ACKNOWLEDGMENT

The authors would like to thank Dr. S. Ghosh for her valu- able reference to EG-LDPCs. This material is based upon work supported by the Department of the Navy, Office of Naval Re- search. Any opinions, findings, and conclusions or recommen- dations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foun- dation or the Office of Naval Research

REFERENCES

1. ITRS, “International technology roadmap for semiconductors,” 2005. [Online]. Available: http://www.itrs.net/Links/2005ITRS/ Home2005.htm
2. 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.
3. 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.
4. DeHon, “Deterministic addressing of nanoscale devices assembled at sublithographic pitches,” IEEE Trans. Nanotechnol., vol. 4, no. 6, pp. 681-687, 2005.
5. DeHon, “Nanowire-based programmable architectures,” ACM J. Emerging Technol. Comput. Syst., vol. 1, no. 2, pp. 109-162, 2005.
6. 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.
7. DeHon and M. J. Wilson, “Nanowire-based sublithographic programmable logic arrays,” in Proc. Int. Symp. Field-Program. Gate Ar- rays, Feb. 2004, pp. 123-132.

8. M. Forshaw, R. Stadler, D. Crawley, and K. Nikolic´, "A short review of nanoelectronic architectures," Nanotechnology, vol. 15, pp. S220-S223, 2004.
9. R. G. Gallager, Low-Density Parity-Check Codes. Cambridge, MA:MIT Press, 1963.
10. 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,