Appearing in *IEEE Int'l Symp. on Defect and Fault Tolerance in VLSI Systems* (DFTS 2007), Sept. 26–28, 2007

Fault Secure Encoder and Decoder for Memory Applications

Helia Naeimi
Computer Science
California Institute of Technology
helia@caltech.edu

André DeHon
Electrical and System Engineering
University of Pennsylvania
andre@acm.org

Abstract

We introduce a reliable memory system that can tolerate multiple transient errors in the memory words as well as transient errors in the encoder and decoder (corrector) circuitry. The key novel development is the fault-secure detector (FSD) error-correcting code (ECC) definition and associated circuitry that can detect errors in the received encoded vector despite experiencing multiple transient faults in its circuitry. The structure of the detector is general enough that it can be used for any ECC that follows our FSD-ECC definition. We prove that two known classes of Low-Density Parity-Check Codes have the FSD-ECC property: Euclidean Geometry and Projective Geometry codes. We identify a specific FSD-LDPC code that can tolerate up to 33 *errors in each memory word or supporting logic that requires only 30% area overhead for memory blocks of 10 Kbits or larger. Larger codes can achieve even higher reliability and lower area overhead. We quantify the importance of protecting encoder and decoder (corrector) circuitry and illustrate a scenario where the system failure rate (FIT) is dominated by the failure rate of the encoder and decoder.*

1: Introduction

Memory systems are protected against transient upsets of data bits using ECCs. Hamming codes are often used in today's memory systems to correct single error and detect double errors in any memory word. In these memory architectures, only errors in the memory words are tolerated and there is no preparation to tolerate errors in the supporting logic (*i.e.* encoder and corrector).

However combinational logic has already started showing susceptibility to soft errors, and therefore the encoder and decoder (corrector) units will no longer be immune from the transient faults. Furthermore, memory system designed with nanotechnology devices are expected to experience even higher transient fault rate [3] [5]; therefore, protecting the memory system support logic implemented with nanotechnology devices is even more important. Here we proposed a fault tolerant memory system that tolerates multiple errors in each memory word as well as multiple errors in the encoder and corrector units.

We illustrate using *Euclidean Geometry* codes and *Projective Geometry* codes to design the above fault-tolerant memory system, due to their well-suited characteristics for this application, which include: 1) Memory applications require low latency encoders and decoders. 2) These codes allow us to design a fault tolerant error-detector unit that detects any error in the received code-vector despite having faults in the detector circuitry; in other words we can design a *fault-secure* detector for these codes (Section 4).

We use the fault secure detector unit to check the output vector of the encoder and corrector circuitry, and if there is any error in the output of either of these units, that unit has to redo the operation to generate the correct output vector. Using this detect-and-repeat technique we can correct potential transient errors in the encoder or corrector output and provide fault-tolerant memory system with fault-tolerant supporting circuitry.

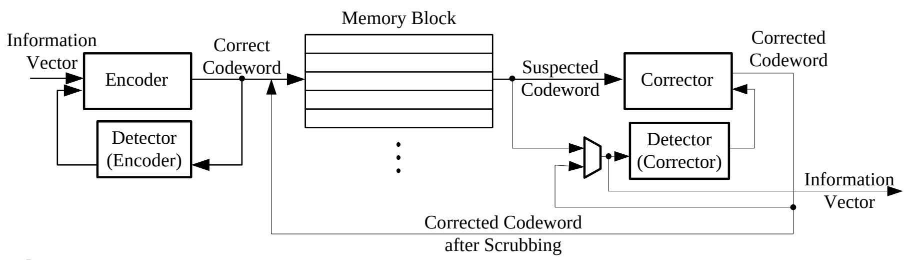


Figure 1. The overview of our proposed fault tolerant memory architecture

In the rest of this paper we first show an overview of our fault-tolerant memory system in Section 2, and a brief overview of ECCs in Section 3. Section 4 presents a stronger ECC definition that can have a *Fault Secure Detector*. Section 5 reviews Euclidean Geometry and Projective Geometry codes and prove that these codes have *Fault Secure Detectors*. The technique and an efficient implementation of one-step majority logic corrector is demonstrated in Section 6, and the reliable systematic encoder is presented in Section 7. The reliability of the complete fault-tolerant memory system and the improvement achieved by tolerating faults in the supporting logic is analyzed in Section 8.

2: System Overview

In this section we outline the memory system which can tolerate errors in any part of the system, including the storage unit, encoder, corrector, and detector circuitry. Let E be the maximum number of error bits that the code can correct and D be the maximum number of error bits that it can detect, and in one error combination that strikes the system let e_e, e_m, and e_c be the number of errors in encoder, memory word, and corrector. In existing designs, the system would guarantee error correction as long as $e_m \leq E$ and $e_e = e_c = 0$. In contrast, here we guarantee that the system can correct any error combination as long as $e_m \leq E$, $e_e + e_{de} \leq D$, and $e_m + e_c + e_{dc} \leq D$, where e_{de} and e_{dc} are the number of errors in two separate detectors, monitoring the encoder and corrector units. This design is feasible when the following two fundamental properties are satisfied: 1) Any single error in the encoder or corrector circuitry can only corrupt a single codeword digit (*i.e.* cannot propagate to multiple codeword digits). 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.

An overview of our proposed reliable memory system is shown in Figure 1, and is as described below: 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. Later during operation, the stored codeword will be retrieved from the memory unit. Since the codeword is susceptible to transient faults while it is stored in the memory, the retrieved codeword must be fed into the checker to detect any potential error and possibly to the corrector to recover any erroneous bits. The fault secure detector of the corrector unit guarantees the correctness of the corrector unit operations with the *detect-and-repeat* technique similar to the encoder.

Transient errors accumulate in the memory words over time. In order to avoid accumulation of too many errors in the memory words that surpasses the code correction capability, the system has to perform memory *scrubbing*. Memory scrubbing is periodically reading memory words from the memory, correcting any potential errors and writing them back into the memory [9]. The effect of memory scrubbing and the details of each of the above units that make the fault tolerant memory structure, will be explain in the following sections.

3: Linear Block Error Correcting Codes

This section provides a brief introduction on linear block ECCs. Let $i = (i_0, i_1, ..., i_{k-1})$ be k-bit information vector that will be encoded into n-bit codeword, $c = (c_0, c_1, ..., c_{n-1})$. For linear codes the encoding operation essentially performs the following vector-matrix multiplication: $c = i \times G$, where G is a $k \times n$ generator matrix. The checking or detecting operation is the following vector-matrix multiplication: $s = c \times H^T$, where H is an $(n-k) \times n$ *Parity-Check* matrix, and the $(n-k)$-bit vector s is called *syndrome* vector. A syndrome vector is zero if c is a valid codeword, and non-zero if c is an erroneous codeword.

A code is a *systematic code* if any codeword consists of the original k-bit information vector followed by $n - k$ parity-bits [8]. With this definition, the generator matrix of a systematic code must have the following structure: $G = [I : X]$, where I is a $k \times k$ identity matrix and X is a $k \times (n-k)$ matrix that generates the parity-bits. The advantage of using systematic codes is that there is no need for a decoder circuitry 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 *Cyclic code* if for any codeword c, all the cyclic shifts of c is still a valid codeword. A code is cyclic *iff* the rows of its parity-check matrix and generator matrix are the cyclic shifts of their first rows.

4: ECC with Fault Secure Detector

Single-error fault-tolerant decoder and encoder circuits for *Reed-Solomon* codes have been suggested in [1]. In this work the encoder is protected with parity-prediction and parity-checker. The decoder is protected by adding a code checker (detector) block and a hamming-distance counter block to count the number of error bits at the output of the decoder. If the code checker detects a non-codeword, then the error in the decoder is detected. If the code checker detects a codeword but the hamming-distance counter indicate a non-zero error, then an error is also detected.

Here we propose a multiple-error fault tolerant decoder and encoder that is general enough for any decoder and encoder implementation and for any kind of ECC that satisfies the restricted ECC definition. The restricted ECC definition which guarantees a fault-secure-detector capable ECC (FSD-ECC) is as follows: 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 C be an ECC, with minimum distance d. C is FSD-ECC iff any error vector of weight $e \leq d - 1$, 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 type of circuitry 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 corrupt 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 code-vector the syndrome weight of the error pattern is at least $d - e$ from the assumption. Furthermore, the maximum number of tolerable errors in the whole system is $d - 1$ and e errors already exist in the encoded vector, therefore the maximum number of errors that can strike in the detector circuitry is $d - 1 - e$. From the

Table 1. The parameters of EG- and PG-LDPC, for any $s \geq 2$

Parameter	EG-LDPC	PG-LDPC
Information bits (k)	$2^{2s} - 3^s$	$2^{2s} + 2^s - 3^s$
Length (n)	$2^{2s} - 1$	$2^{2s} + 2^s + 1$
Minimum distance (d)	$2^s + 1$	$2^s + 2$
Dimensions of the Parity-Check matrix	$n \times n$	$n \times n$
Row weight of the Parity-Check matrix (ρ)	2^s	$2^s + 1$
Column weight of the Parity-Check matrix (γ)	2^s	$2^s + 1$

above note, this many errors can corrupt at most $d - 1 - e$ syndrome bit, which in worst case leaves at least one non-zero syndrome bit and therefore detects the errors. **Q.E.D**

The difference between FSD-ECC and normal ECC is the demand on syndrome weight: *i.e.*, a normal ECC demands non-zero syndrome weight while FSD-ECC demands $\geq d - e$.

5: Euclidean Geometry Codes

Two types of ECCs with FSD-ECC property are subclasses of *Euclidean Geometry* codes and *Projective Geometry* codes [10]. This section briefly reviews these two codes. We start by showing the construction of Euclidean geometry codes. Let **EG** be a Euclidean Geometry with n points and J lines, then **EG** is a finite geometry such that:

- Every line consists of ρ points.
- Any two points are connected by exactly one line.
- Every point is intersected by γ lines.
- Two lines intersect in exactly one point or they are parallel, *i.e.* don't intersect.

Let H be a $J \times n$ binary matrix, whose rows and columns correspond to lines and points in **EG** geometry, respectively, where $h_{i,j} = 1$ if and only if the ith line of **EG** contains the jth point of **EG**, and $h_{i,j} = 0$ otherwise. A row in H displays the points on a specific line of **EG** and has weight ρ. A column in H displays the lines that intersect at a specific point in **EG** and has weight γ. The rows of H are called the incidence vectors of the lines in **EG**. Therefore H is the incidence matrix of the lines in **EG** over the points in **EG**. [7] shows construction algorithm for these codes and prove that H is a Low-Density Parity-Check matrix, and therefore the code is an LDPC code, and it is called EG-LDPC code henceforth.

A special subclass of EG-LDPC code, type-I 2-dimensional EG-LDPC [10], is considered here. The parameters of these codes are summarized in Table 1. An important point is that the rows of H are not all linearly independent. The rank of H is $(n - k)$ which makes the code of this matrix an (n, k) linear code. The parity-check matrix H of an **EG** euclidean geometry, can be formed by taking the incidence vector of a line in **EG** and its $n - 2$ cyclic shifts as rows, therefore this code is a *cyclic code.* The type-I 2-dimensional Projective Geometry codes have similar structure to EG-LDPC codes, with the specifications shown in Table 1.

5.1 Euclidean Geometry Codes and FSD-ECC

Theorem II: *Any ECC of type-I 2-dimensional EG-LDPC or PG-LDPC is FSD-ECC.*

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

A specific bit in the syndrome vector is 1 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. Now 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, γ lines pass through each point; so e error points potentially impact γe lines. Also at most one line connects two points. Therefore, looking at the e error points, there are at most $\binom{e}{2}$ lines between any two error points. Hence the number of lines passing through a collection of these e points is lower bounded by $\gamma e - \binom{e}{2}$. Out of this number, at most $\binom{e}{2}$ lines connect two or more points of the error points. Summarizing all this, the number of lines passing through exactly one of the error points is at least $\gamma e - 2\binom{e}{2}$. Note from the code properties (Table 1) that $d = \gamma + 1$, we can derive the following inequality:

$$\gamma e - 2\binom{e}{2} \quad = e(\gamma + 1 - e) = e(d - e) \geq d - e$$

Which prove that the weight of the syndrome vector is at most $d - e$ which is the required condition of Theorem (I). Therefore EG- and PG-LDPC are FSD-ECC. **Q.E.D.**

5.2 Fault Secure Detector Implementation

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: $s = c \cdot H^T$, and therefore each bit of the syndrome vector is the product of c with one row of the parity-check matrix. This product is a linear binary sum over digits of c where the corresponding digit in matrix row is 1. This binary sum is implemented with an XOR gate. Since the row weight of the parity-check matrix is ρ, to generate one digit of the syndrome vector we need ρ-input XOR gate, or $(\rho - 1)$ 2-input XOR gates. For the whole detector, it takes $\gamma(\rho - 1)$ 2-input XOR gates. Figure 5(a) illustrates this quantity for some of EG- and PG-LDPC codes.

An error is detected if any of the syndrome bit has non-zero value. The final error detection signal is implemented by an OR function of all the syndrome bits. The output of this n-input OR gate is the error detector signal. In order to avoid single point of failure, we have to implement the OR gate with reliable substrate, *e.g.* in a system with sub-lithographic *nanowire* substrate [2], the OR gate is implemented with reliable lithographic technology.

6: One-Step Majority Logic Corrector

One-step majority-logic correction is a fast and relatively compact error-correcting technique [7]. There are a few ECCs known to be one-step-majority correctable, including type-I 2-dimensional EG- and PG-LDPC. This method corrects a codeword one code-bit at a time. For EG- and PG-LDPC codes, the one-step majority-logic corrector corrects up to $\gamma/2$ error bits in the received encoded vector, by computing γ *parity-check* sums of ρ code-bits each. Each parity-check sum is implemented with a ρ-input XOR function (Figure 2(a)). The majority value of the parity-check sums are then evaluated, with γ-input majority gate. If the majority value is 1 then the code-bit under consideration holds an erroneous value and has to be inverted. For cyclic codes, including EG- and PG-LDPC, a single serial majority corrector circuit can be used for all the code-bits, where the received encoded vector is cyclic shifted and fed into the XOR gates, to correct each code bit.

Assuming our building blocks are 2-input gates, to generate γ number of ρ-input parity-check sums takes $\gamma \times (\rho - 1)$ 2-input XOR gates. Since the size of γ-input majority gate could grow exponentially with γ, we propose a novel majority gate implementation using *Sorting Networks* [6]. A majority function of γ binary digits is simply the median of the

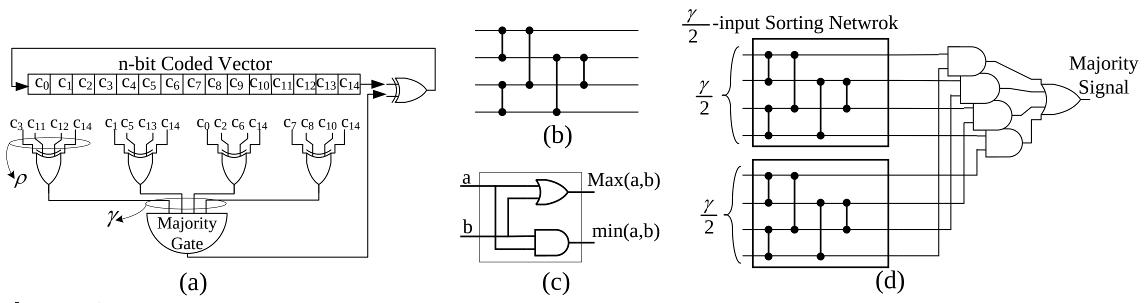
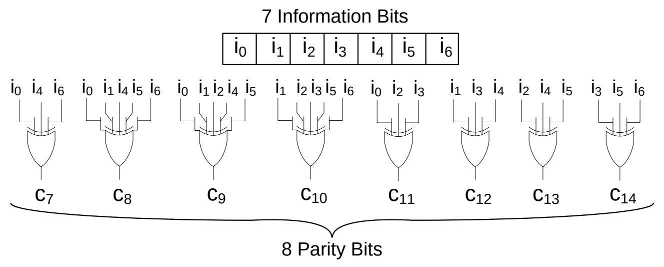


Figure 2. (a) The one-step majority logic corrector for $(15, 7)$ EG-LDPC code. (b) A 4-input sorting network: Each of the vertical lines shows a 2-bit sorter. (c) A 2-bit sorter: Each comparator finds the maximum value of two binary inputs with an OR gate and the minimum value with an AND gate. (d) An 8-input majority gate.

$$
\begin{array}{c|ccccccccccccccc}
 & c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 & c_7 & c_8 & c_9 & c_{10} & c_{11} & c_{12} & c_{13} & c_{14} \\
\hline
i_0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
i_1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
i_2 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
i_3 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
i_4 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
i_5 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
i_6 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}
$$

(a)



(b)

Figure 3. $(15, 7)$ code: (a) Generator matrix in systematic format (b) Encoder circuit

digits. To find the median of γ inputs, the inputs can be sorted using a γ-input sorting network. Figure 2(b) shows a 4-input sorting network. Each vertical line represents one comparator, which sorts two bits (Figure 2(c)). We can further minimize the area of the circuit. Instead of sorting the whole γ inputs, the inputs are divided into two parts, and each part is sorted using $\gamma/2$-input sorting networks. Then the median is 1 if for $i = 1, 2, ..., \gamma/2$ the ith element of one half and the $(\gamma/2 + 1 - i)$th element of the other half are both 1. This condition is checked with a logic circuit as shown Figure 2(d) for 8-input median function.

The serial majority corrector takes n cycles to correct an erroneous codeword. However the corrector block is used only when an error is detected in the memory-word, which is not frequent. Therefore since the common case is error-free codewords and only small fraction of codewords demand the correction operation, the latency of the corrector will not have severe impact on the average memory read latency. Figure 5(a) shows the number of 2-input gates in this serial majority corrector for different codes.

7: Encoder Unit

An n-bit codeword c, which encodes a k-bit information vector i is generated by multiplying the k-bit information vector with $k \times n$ bit generator matrix G, *i.e.* $c = i \cdot G$.

From Section 5 we know that the EG- and PG-LDPC codes are cyclic codes. However this cyclic code generation does not generate a systematic code 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. The generator matrix of any cyclic code can be converted into systematic form ($G = [I : X]$) as shown in [8] and [7].

Figure 3(a) shows the systematic generator matrix of $(15, 7)$ code. The systematic encoded vector, which is generated by the inner product of the information vector and the generator matrix, 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 $G = [I : X]$. Therefore each parity bit is a binary sum (equivalent to an XOR function) of a set of information bits represented by 1 entries in the corresponding column. Figure 3(b) shows the encoder circuit to compute the parity bits of $(15, 7)$ EG-LDPC code, with generator matrix shown in Figure 3(a). 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, c. This circuit takes twenty-two 2-input XOR gate.

In general the vector-matrix multiplication of $c = i \cdot X$ takes, $(n-k)$ XOR gates. The fan-in of the gates depends on the generator matrix of the code. We implemented the procedure developed in [4] and generated all the cyclic generator matrices of the EG- and PG-LDPC codes under consideration. We further transformed the developed cyclic generator matrix to systematic form using the procedure in [8]. From those systematic matrices we computed the exact number of 2-input XOR gates. This information is summarized in Figure 5(a).

8: Reliability and Area Analysis

In this section we compute the system reliability, and report the FIT rate of the system for various device fault rates. We also analyze the impact of the supporting logic failure and illustrate the importance of protecting logic for high fault rates.

Let P_f be the probability that a transistor loses its correct value at a cycle. We assume that this probability has i. i. d. and random distribution over system devices. Recall e_e, e_m, e_c, e_{de}, and e_{dc} are the numbers of errors occur in encoder, memory unit, corrector, and the detectors of encoder and corrector, respectively. Let n_e, n_c, and n_d be the size of the circuitry involved in an operation on a single code bit in the encoder, corrector, or detector, respectively. Then the probability that a code bit is erroneous in any of these units is: $P_{bit\_circuit} = 1 - (1 - P_f)^x$, where x has one of the values: n_e, n_c, or n_d, and the probability that a memory bit is erroneous in scrubbing interval of s cycles is: $P_{bit\_mem} = 1 - (1 - P_f)^{6s}$, where 6 is the number of transistor in one SRAM-cell. Each unit or a memory word experience e errors among n bit of the codeword with the probability:

$$P_{unit} = \binom{n}{e} {P_{bit}}^{e} (1 - P_{bit})^{n-e} \tag{1}$$

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

As explained in Section 4, errors in the encoder unit are detected by its following detector, and is 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. Therefore we define the first reliability condition as:

Cond. I) $\quad e_e + e_{de} < d$

which states that the total number of errors in the encoder unit and the following detector unit must be smaller than the minimum distance of the code. The detector of the corrector is also capable of detecting up to $d - 1$ errors accumulated from memory unit, corrector unit and the second detector unit. Therefore the second reliability condition is defined as:

Cond. II) $\quad e_m + e_c + e_{dc} < d$

Furthermore the corrector can recover a memory word with up to $\gamma/2$ errors from the memory unit, therefore the third reliability condition is formulated as:

Cond. III) $\quad e_m \leq \gamma/2$

Satisfying the three above conditions guarantees that the memory system operates with no undetectable or uncorrectable errors. We calculated the probability of each of the above conditions employing Equation (1), for all the various EG- and PG-LDPC codes. Figure 4(a) shows the failure rate of the memory system protected with these codes in *FIT* (Failure In Time, errors per 10^9 hours). In this example the scrubbing interval is 10 minutes, meaning less than 0.2% of memory cycles are spent on scrubbing, about about 0.5% of memory cycles are spent on correction in the average; therefore, the total throughput loss is less than 1%.

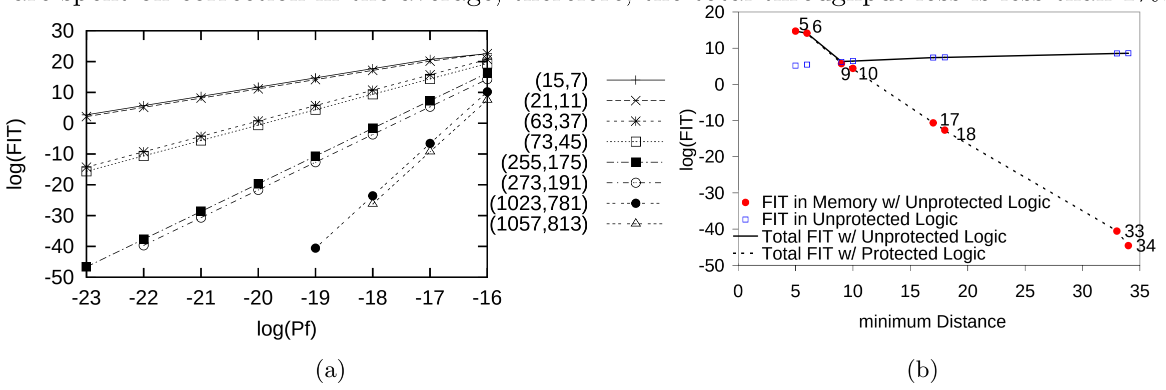


Figure 4. (a) The FIT of 1 Gbit, 10 GHz memory system, with scrubbing interval of 10 minutes (b) The impact of supporting logic failure on system FIT for the same memory at $P_f = 10^{-19}$

It is important to understand the impact of protecting supporting logic on the system FIT. This is illustrated with the graph presented in Figure 4(b), which shows the FIT of different parts of the system: memory bank and supporting logic. The FIT in the supporting logic is without fault secure detector (*i.e.* any error in the supporting logic results an erroneous output, with worst-case analysis). Obviously the FIT of the whole system with no logic protection is the sum of the above two FIT's, illustrated with a solid line. As you can see, for codes with minimum distance > 9 the FIT of the system with no logic protection is dominated by the FIT of the logic. Beyond this point, using codes with higher reliability will not decrease the FIT of the whole system; while the larger codes do decrease the FIT of memory bank, the high FIT of unprotected logic determine the system FIT. To achieve higher reliability the logic must also be protected. The FIT of such system with fault secure logic is illustrated with the dashed line, which follows closely the FIT of the memory bank.

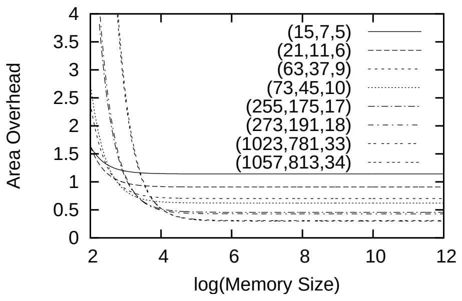
Figure 5(a) shows the area of each part of the memory system in the number of 2-input gate counts. The more accurate area computation, which plots the area overhead in the number of transistor counts, is plotted in Figure 5(b). In this analysis, memory cells are 6-transistor **SRAM** cells. The 2-input XOR, OR, and AND gates each contains 8, 4, and 4 transistors. This figure shows that the area overhead converges to the corresponding code overhead ($(n-k)/k$) for memory bank larger than 10 Kbit; the area overhead of the supporting logic is amortized out over large number of memory cells and therefore the total area overhead is dominated by the code overhead.

9: Summary

In this paper we have developed a memory system that can tolerate and correct errors not only in the storage unit but also in the supporting circuitry. We define the Fault-Secure Detector capable ECC (FSD-ECC) concept which allows us to implement memory systems with protected supporting circuitry. We proved that Euclidean and Projective Geometry codes are FSD-ECC, and presented a compact implementation for this system with complete area overhead analysis. We have computed the complete system reliability. This system can work on highly unreliable substrate. Using the specific code identified, it can tolerate up

Type	s	(n , k)	γ	e	Cor. Area	Det. Area	Enc. Area
EG	2	(15,7)	4	2	19	45	22
PG	2	(21,11)	5	2	32	48	47
EG	3	(63,37)	8	4	83	501	355
PG	3	(73,45)	9	4	108	584	643
EG	4	(255,175)	16	8	331	3825	6577
PG	4	(273,191)	17	8	376	4368	7429
EG	5	(1023,781)	32	16	1263	31713	93823
PG	5	(1057,813)	33	16	1358	33824	98730

(a)



(b)

Figure 5. (a) The number of 2-input gates required for various parts of the system (b) The area overhead in the number of transistor counts for various codes and memory sizes

to 16 errors in each memory-word, and 33 errors in the memory word and supporting logic overall. This is achieved while the area overhead of the system for large enough memory blocks ($\geq$ 10Kbits) is about 30%. Higher reliability is also achievable with larger EG- or PG-LDPC codes. We illustrated the impact of unprotected supporting logic failure on the system FIT and noted the importance of protecting logic to improve system reliability (Section 8).

10: Acknowledgement

We would like to thank Dr. Shalini Gohsh for her valuable reference to EG-LDPCs, and Benjamin Gojman for the discussion leading to the optimized majority gate implementation.

This research was funded in part by National Science Foundation Grant CCF-0403674 and the Defense Advanced Research Projects Agency under ONR contract N00014-01-0651. This material is based upon work supported by the Department of the Navy, Office of Naval Research. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or the Office of Naval Research.

References

[1] G. C. Cardarilli et al. Concurrent error detection in reed-solomon encoders and decoders. *IEEE Trans. VLSI*, 15:842–826, 2007.

[2] André DeHon. Nanowire-Based Programmable Architectures. *ACM Journal on Emerging Technologies in Computing Systems*, 1(2):109–162, 2005.

[3] S. Hareland et al. Impact of CMOS process scaling and SOI on the soft error rates of logic processes. In *Procedings of Symposium on VLSI Digest of Technology Papers*, pages 73–74, 2001.

[4] R. Horan et al. Idempotents, mattson-solomon polynomials and binary LDPC codes. *IEE Proceedings of Communication*, 153(2):256–262, 2006.

[5] J. Kim et al. Error rate in current-controled logic processors with shot noise. *Fluctuation and Noise Letters*, 4(1):83–86, 2004.

[6] Donald E. Knuth. *The Art of Computer Programming*. Addison Wesley, second edition, 2000.

[7] Shu Lin and Daniel J. Costello. *Error Control Coding*. Prentice Hall, second edition, 2004.

[8] R. J. McEliece. *The Theory of Information and Coding*. Cambridge University Press, 2002.

[9] Abdallah Saleh et al. Reliability of scrubbing recovery-techniques for memory systems. *IEEE Transaction on Reliability*, 39(1):114–122, 1990.

[10] Heng Tang et al. Codes on finite geometries. *IEEE Transaction on Information Theory*, 51(2):572–596, 2005.