Dynamic LDPC Codes for Nanoscale Memory with Varying Fault Arrival Rates

Shalini Ghosh, Patrick D. Lincoln
SRI International, Menlo Park, California

Abstract—Modern state-of-the-art nanodevices exhibit remarkable electronic properties, but the current assembly techniques yield very high defect and fault rates. Static errors can be addressed at fabrication time by testing and reconfiguration, but soft errors are problematic since their arrival rates are expected to vary over the lifetime of a part. Usual designs consider error correcting codes that tolerate the maximum failure rate expected over the entire lifetime. In this paper, we propose using a special variant of low-density parity codes (LDPCs) — Euclidean Geometry LDPC (EG-LDPC) codes — to enable dynamic changes in the level of fault tolerance. EG-LDPC codes have high error correcting ability (for large words they can approach the optimal Shannon limit) and they are sparse (circuit implementation requires small fan-in). In addition, a special property of EG-LDPC codes enables us to dynamically adjust the error correcting capacity for improved system performance (e.g., lower power consumption) during periods of expected low fault arrival rate. We present a system architecture for nanomemory based on nanoPLA building blocks using EG-LDPCs, where the encoder/decoder could also have faults, and analyze the fault detection and correction capabilities considering dynamic fault tolerance.

I. INTRODUCTION

In this paper we focus on soft error correction for nanoscale memory, using nanoPLA blocks and simple nanogates (e.g., majority gate, NAND/NOR gates) as architectural components [1]. Static errors can be handled using testing and reconfiguration [2]. But for nanomemories with high fault rates, a new type of soft error correction is desired since conventional ECC techniques are not directly applicable.

The encoders and decoders for conventional ECC codes, e.g., Hamming and Hsiao codes, have low encoding and decoding complexity, but also have relatively low error correcting capacity (e.g., Hamming is single error correcting, double error detecting). To achieve higher error correcting capability, codes like Reed-Solomon or BCH require more sophisticated decoding algorithms, which would need either (a) complex algebraic decoders that can decode in fixed time — the designs for these complex operations (e.g., floating point operations, logarithms) would be difficult to implement using nanoscale PLAs, which favor simple regular designs, or (b) simpler graphical decoders, that use iterative algorithms (e.g., belief propagation) — these typically need more computation time, and hence would not be fast enough for at-speed ECC operations for nanomemory [3].

For these reasons, we desire an error correcting system that has (1) high error detecting and correcting ability, to tolerate relatively high soft error rates, and (2) sparse and regular coding, decoding and checker circuits, so that they can be synthesized using simple nanoscale hardware. Additional properties that are desirable for some applications include (a) modular encoder and decoder blocks, which can simplify and shrink hardware design; (b) systematic code structure, which will cleanly partition the information and code bits in the memory; and (c) dynamic error correcting capability, to enable engineering the trade-off between error correction and system performance.

We propose the use of a variant of a particular type of low-density parity check (LDPC) code [4], Euclidean Geometry (EG) LDPC [3], which is built using special structures of finite Euclidean Geometry. Various types of EG-LDPC codes have some of the desirable properties listed earlier, e.g., type-I codes are systematic, type-II codes have encoding and parity check matrices with regular modular structure, Gallager codes have properties that enable their error correction rate to be changed dynamically. The sparseness (enabling low fan-in circuit implementation) and low computational overhead of decoding EG-LDPC codes make them easy to implement using nanoscale hardware.

II. BACKGROUND

We first outline concepts that will be useful in understanding our main design and architecture for ECC in a nanomemory based on EG-LDPC codes.

LDPC codes: LDPC codes have several advantages, which have made them popular in many communication applications: (1) low density of the encoding matrix, (2) easy iterative decoding, (3) generating large code words that can approach Shannon’s limit of coding [3]. A $(\gamma, \rho)$-regular LDPC code is defined as the null space of a parity check matrix $H$ that has the following properties [3]: (1) each row has $\rho$ number of 1’s; (2) each column has $\gamma$ number of 1’s; (3) the number of 1’s that are common between any two columns ($\lambda$) is no greater than 1, i.e., $\lambda = 0$ or 1; and (4) both $\rho$ and $\gamma$ are small compared to the length of the code and the number of rows in $H$. As both $\rho$ and $\gamma$ are very small compared to the code length and the number of rows in the matrix $H$, $H$ has a low density of 1’s. Hence $H$ is said to be a low-density parity check matrix and the code defined by $H$ is said to be a low-density parity check code. The density of $H$ ($r$) is defined to be the ratio of the total number of 1’s in $H$ to the total number of entries in $H$ — in this case $r = \rho/n = \gamma/J$, where $J$ is the number of rows in $H$. If the weights of all the columns or rows in $H$ are not the same, the code is called an irregular LDPC code.

EG-LDPC codes: In recent work [5], Euclidean Geometry (EG) constructions over $GF(2^s)$ have been used to construct
of the memory can be switched off, too. For that, we propose to selectively enable modules of the ECC circuit at different times of the operation, so that \( \gamma \) is higher during the initial and final phases and low during the middle phase of operation. The corresponding encoder can be modified accordingly, if reconfigurable nanoPLA hardware is used. This capability of changing the error correcting capacity of the encoder and decoder gives the required dynamic error correcting capability, and resulting power savings.

However, significant power savings are possible only if parts of the memory can be switched off, too. For that, we propose a memory repacking scheme in our memory bank architecture.

### Architecture: Figure 1 shows the overall system architecture of the nanomemory with EG-LDPC ECC. During a write operation, the incoming word to be stored in memory is encoded by the encoder and the code word is stored in memory. During a read operation, a code word is retrieved from the memory, checked by the checker unit, and finally the majority logic unit decodes the syndrome and does the error correction.

The controller unit controls the error detection and correction capability of the ECC unit. The following paragraphs describe these different components and their implementation using nanoPLA components.

![Fig. 1. Overall architecture of ECC nanomemory.](image)

### Checker and Encoder: Each submatrix \( H_i \) in the equation for \( H \) can be expressed in the form \( P_i H_1 \), where \( P_i \) is a permutation matrix. The \( i^{th} \) unit in the checker circuit, which corresponds to the parity check submatrix \( H_i \), consists of a block of XOR gates implementing \( H_1 \) and a reconfigurable permutation block configured to implement \( P_i^T \). If the system needs to operate at an error correcting level of \( \gamma/2 \), encoder units 1 to \( \gamma \) are enabled by a controller and the remaining units are disabled. The design of the checker can be implemented using the same module \( H_1 \) in each unit, along with a corresponding permuting array (mixer) — this makes the design regular and modular, both of which are characteristics facilitating implementation using nanoPLA components. The basic block \( H_1 \) is essentially a set of n-input XOR gates, which can be realized in nanoPLA using a nested tree configuration of the 2-input XOR gates. The permutation array is a reconfigurable switching circuit, which can be implemented using a nanoPLA cross-bar architecture [1]. Once the checker circuit is changed by selectively turning off some of its blocks, the corresponding encoder circuit can be modified by reconfiguring the nanoPLA implementing the encoder.

### Memory: When the ECC requirements are less, the \( k \) value of the code can be increased (thereby decreasing \( \gamma \)). In terms of the memory, changing \( k \) amounts to varying the number of information bits stored per word of memory. When \( k \) becomes higher with a lower ECC requirement, more information bits can be packed into a memory code word. If the information bits are properly repacked, all the information bits in a memory bank can be stored in a lesser number of memory blocks, thereby enabling a few memory blocks to be powered off for power savings. The repacking architecture is designed using a

### NanoPLA architecture: In this work, we refer to the nanochip architecture based on Programmable Logic Arrays (PLAs), using a two-plane PLA with silicon nanowires [1]. The defects that occur in this kind of architecture are mainly of two types — (1) wire, where the wire is either functional or defective and (2) non-programmable crosspoint, where the crosspoint cannot be programmed into the on state, or the crosspoint may be shorted into an on state.

### Dynamic ECC: Soft error rates vary over the lifetime of complex electronic components, due to changes in the expected fault rate. If one designs ECCs to only tolerate the maximum failure rate expected over the entire lifetime of a device, it can result in wasteful overhead for tolerating too many faults during the bulk of the component’s lifetime. In these cases, we wish to enable the dynamic control of fault tolerance, which may allow one to shut down a subset of ECC circuits to save power when error rate is low.
memory bank, which is a useful architecture in the presence of high error rates [6]. In memory banks, the read/write access to multiple memory blocks is controlled via a memory controller unit. In memory units with ECC, typically there is a scrubbing logic that periodically reads memory words, corrects them if they have errors and writes them back — this maintains the integrity of the memory. We modify the scrubbing logic to perform repacking. The repacking unit has a repacking buffer, and it uses the decoder and encoder circuits of the memory ECC. When the repacking controller receives notification from the CPU to modify the ECC and repack the memory, the repacking controller reads memory rows one by one from all the memory units in the bank except the active memory unit, from which the CPU is reading data [6] — for the active unit, its request is processed only if there is no current memory request from the CPU. The repacking controller stores the data from the rows of the different memory units into the repacking buffer, re-segments the data according to the new $k$ size and writes them back to the necessary number of memory blocks. The new ECC requirement and $k$ are chosen such that the data gets repacked into fewer memory blocks, thereby making it possible to switch off one or more memory blocks.

**Decoder and Controller:** The majority-logic (ML) decoder requires two key hardware units: XOR and majority (MAJ). A 3-input majority gate MAJ(A,B,C) can be implemented using the AND-OR planes of the nanoPLA, since MAJ(A,B,C) = AB + BC + AC. An $n$-input MAJ gate can be similarly implemented using AND/OR gates, or equivalently using the AND-OR planes of a nanoPLA. Like the encoder circuit, the decoder is reconfigured when the ECC is modified, to handle dynamic coding. This reconfiguration can be handled easily since the MAJ decoder is implemented using reconfigurable nanoPLA.

The control signals from the memory repacking controller circuit (used to disable memory blocks in the memory bank architecture) are also used for selectively enabling and disabling modules of the checker, and for reconfiguring the encoder and decoder components. Since we need only one controller for a complete memory bank, this module can be implemented using micro-level circuits.

**IV. Analysis and Experiments**

We analyze the effects of faults and dynamic coding in the various components of the proposed model.

**Fault Tolerance:** Let $\epsilon_e$, $\epsilon_m$, and $\epsilon_c$ be random variables signifying the number of errors in the encoder, memory, and checker blocks, respectively. When $\gamma$ components are enabled by the controller, the overall error correcting capacity of the ECC memory system (with ML decoding) is $\gamma/2$. Therefore, the majority-logic decoder will be able to correct errors as long as $\epsilon_e + \epsilon_m + \epsilon_c \leq \gamma/2$. Note that the MAJ decoder is assumed to be fault free, which can be ensured by using self-checking in the MAJ logic.

Let us now analyze each component of the above equation in detail. When $\gamma$ units are enabled by the controller, the code is a $(\gamma, \rho)$-regular Gallager code according to Section III, i.e., the check matrix unit $H_1$ has a single 1 in each column and $\rho$ 1’s in each row. Consequently, all XOR gates realizing the $H_1$ matrices are $\rho$-input XOR gates. In our worst-case analysis, we will consider the effects of errors in the XOR gates to be additive. Let $p_e$ and $p_c$ be the errors in one bit location of an XOR gate in the encoder and checker, respectively, and $p_m$ be the probability of a nanoPLA memory junction losing its charge. Assuming errors to be i.i.d. and the total number of code bits to be $n$, we get

$$\text{Prob}(e \text{ out of } n \text{ code bits have errors}) = \binom{n}{e} (p)^e (1-p)^{n-e},$$

where putting $p = p_m$ gives us the error probability in the memory, while selecting $p = p_e$ (or $p = p_c$) gives us the error probability in the encoder (or checker). Considering the overall error function and Equation (1), we see that the distribution of errors in the overall ECC memory system is the sum of binomial random variables. Therefore, the probability of the majority decoder not having any error is the cumulative distribution of the convolution of binomial distributions.

![Figure 2](image-url) -- Probability of correct detection as a function of the number of dynamic EG-LDPC checker units ($\gamma$). The three plots correspond to different values of the memory size ($n = 8, 16, 32$), corresponding to different values of the $m$ parameter ($m = 3, 4, 5$) of the EG-LDPC code.

Figure 2 shows how the probability of error-free operation of a memory system with EG-LDPC error correction changes...
along with different values of $\gamma$ (the number of stages in the dynamic error control), for $p_e, p_c = 1\%$ (error rates for encoder and checker) and $p_m = 2\%$ (error rate for memory).

As shown in the figure, for an 8-bit memory, $\gamma = 6$ gives almost 100% probability of correct operation; the same high reliability is obtained using $\gamma = 8$ for a 16-bit memory and $\gamma = 12$ for a 32-bit memory. Figure 3 shows the plot for a 64-bit memory (corresponding to $m = 6$), for a higher error rate (8%) — using $\gamma = 32$ in this case gives > 99% probability of correct operation. Note that in all these figures, the size of the memory is $n = 2^m$, for a $(m, 2^n)$ EG-LDPC code.

**Dynamic Coding:** Along the bathtub curve, the probability of failure of components varies — high in the beginning, low during normal operations, and finally high again. When $m = 5$ and $p = 6\%$, 99% error correction requires $\gamma = 15$. For the same circuit if $p$ decreases to 1%, the required $\gamma$ for 99% error correction decreases to 5, implying that one can get the same error correcting ability with lesser number of active hardware modules under low error conditions.

**Area Overhead:** The different components of the memory LDPC ECC system implemented using nanocomponents are (1) encoder, (2) memory, (3) checker, and (4) decoder. Note that the controller is designed using micro-level circuitry for the whole memory bank; hence, we do not consider its area in our current analysis.

An upper bound on the number of 2-input XOR gates required for the encoder and checker circuits is $(n-k)(k-1) + J(\rho-1)$. The decoder has $n \gamma$-input MAJ gates, which can be synthesized using 2-input AND/OR gates or directly with a nanoPLA. In either case, each of the $n \gamma$-input MAJ gates will need on the order of $(2^{n-1})$ 2-input AND gates and $(\gamma-1)$ 2-input OR gates. Let us consider that a memory unit is composed of 6 nanotransistors, while 2-input OR, AND, and XOR gates are composed of 4, 4, and 8 nanotransistors respectively. For a 2-dimensional $(64, 40)$ EG-Gallager code with $\gamma = 6, \rho = 6, J = 64$, constructed based on $EG(2, 2^4)$, the area of the encoder, checker, and decoder circuits is 44K transistors. The corresponding area for a 64-bit memory with 1K rows is 384K transistors. Table I shows the area overhead of having LDPC ECC in the memory, where area is measured in terms of number of transistors. As the size of the memory block increases, the overhead of the parity bits and the ECC logic converges to $(1-\text{code rate}) = (n-k)/n = 37.5\%$, which indicates that for large memory blocks the additional overhead due to the ECC encoder, checker, and decoder circuits is low (e.g., for 16K memory, it is 0.5%).

<table>
<thead>
<tr>
<th>Memory size</th>
<th>Memory area</th>
<th>Info area</th>
<th>Parity ECC area</th>
<th>Overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>384K</td>
<td>240K</td>
<td>188K</td>
<td>49%</td>
</tr>
<tr>
<td>2K</td>
<td>768K</td>
<td>480K</td>
<td>332K</td>
<td>43%</td>
</tr>
<tr>
<td>4K</td>
<td>1536K</td>
<td>960K</td>
<td>620K</td>
<td>40%</td>
</tr>
<tr>
<td>8K</td>
<td>3072K</td>
<td>1920K</td>
<td>1196K</td>
<td>39%</td>
</tr>
<tr>
<td>16K</td>
<td>6144K</td>
<td>3840K</td>
<td>2348K</td>
<td>38%</td>
</tr>
</tbody>
</table>

TABLE I

**Area overhead for LDPC ECC in nanomemory**

V. CONCLUSIONS AND RELATED WORK

We have presented and analyzed the properties of dynamic error correction using EG-LDPC codes, a particular class of error correcting codes whose special properties enables efficient coding/decoding in hardware for varying levels of faults, using dynamic fault tolerance. There are interesting extensions to this work, e.g., designing block-level error correcting codes, exploiting fault localization.

EG-LDPC codes applied to nanomemories were first introduced in [7]. Detailed follow-up work on useful properties of EG-LDPC codes (e.g., dynamic error correction) have been outlined in [8]. A related approach to using EG-LDPC codes in fault-secure encoders and decoders was proposed in [9], but it does not handle dynamic error correction for variable rate faults. LDPC codes based on other finite geometries (e.g., projective geometries) have also been studied [10], [5]. In our work, we use a property of EG-LDPC codes that allows simpler decoding logic (involving majority gates) to be used, which can be implemented in nanohardware using different techniques [11], [12], [13].

VI. ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation, under Grants CCF-0726843 and CNS-0749931. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

REFERENCES