# High-Performance, Fault-Tolerant Architecture for Reliable Hybrid Nanolectronic Memories

N. Z. Haron, A. M. Darsono, A. A. M. Isa
*Faculty of Electronic and Computer Engineering*
*Universiti Teknikal Malaysia Melaka, Malaysia*
*zaidi@utem.edu.my*

***Abstract*—Although hybrid nanoelectronic memories (hybrid memories) promise scalability potentials such as ultra-scale density and low power consumption, they are expected to suffer from high defect/fault density reducing their reliability. Such defects/faults can impact any part of the memory system including the memory cell array, the encoder and the decoder. This article presents a high-performance, fault-tolerant architecture for hybrid memories; it is based on a combination of two techniques: (i) an error correction scheme that tolerates both random and clustered faults in memory cell array and (ii) an on-line masking incorporated into the decoder to tolerate faults in the decoder. Moreover, the decoding process is optimized for area and performance by reversing the decoding sequence. Experimental results show that the proposed architecture realizes a higher performance and competitive reliability level at a comparable overhead as compared with the state-of-the-art. For example, the architecture decodes 5× faster and provides 0.7% better reliability (assuming 10% fault rate) at the cost of similar area overhead (for 1024-bit memory word) as compared to Reed-Solomon code.**

***Index Terms*—Error correction codes, fault tolerance, hardware redundancy, hybrid nanoelectronic memories.**

## I. Introduction

It is well-acknowledged that CMOS technology is experiencing difficulties to sustain the scalability in producing smaller, higher-capacity, less-power and cheaper memories. One promising solution is to use non-CMOS devices (instead of CMOS or capacitor) as storage elements. This is the main idea behind hybrid nanoelectronic memories (hybrid memories) where the non-CMOS memory cell array can also be stacked on the top of scaled CMOS peripheral circuits. Hybrid memories, such as molecular-based memories [1]–[4] and carbon nanotube-based memories [5]–[8], offer scalability potentials better than that of existing memories. However, they will suffer from significant faults (permanent, intermittent and transient) [9]–[11]. In addition, a single defect/fault might induce *clustered errors* impacting adjacent memory cells and transistors in such circuits [10], [11]. Therefore, designing reliable hybrid memories require not only to protect the non-CMOS memory cell array from clustered faults, but also the CMOS peripheral circuits.

Published work on hybrid memory reliability has applied fault-tolerant schemes such as error correction codes (ECCs) [1], [12]–[17], hardware redundancy [13], [16], [18], reconfiguration [12], [14] and re-execution [15], together with supporting scheme like scrubbing [15], defect map [19], [20] and tagging mechanism [20]. However, most of the published work focuses only on the memory cell array while assuming that the other memory parts are reliable. This assumption is no longer valid for hybrid memories because even at 130nm CMOS technology node, logic circuits have exhibited almost similar fault rate to that of unprotected memories [21]. To the best knowledge of the authors, there is only a single published article [15] that addresses fault tolerance for the entire hybrid memory system; the authors in [15] combine an ECC, re-execution and scrubbing to tolerate faults that induce *random errors* in the memory cell array, encoder and decoder. However, as clustered errors are also expected to occur in hybrid memories [10], [11], an appropriate fault-tolerant scheme to deal not only with random errors but also with clustered errors is required.

This article presents a high-performance, fault-tolerant architecture to tolerate faults that induce clustered and random errors in hybrid memories, thereby improve their reliability. The architecture is based on a combination of two techniques: (i) an error correction scheme (based on Redundant Residue Number System code and double modular redundancy) that tolerates both random and clustered faults in the memory cell array, and (ii) an on-line masking to tolerate faults in the decoder. Moreover, the decoding process is optimized for area and performance by reversing the decoding sequence. Experimental results show that the proposed architecture realizes a higher performance and competitive reliability level at a comparable area overhead as compared with the state-of-the-arts. The main contributions of this article are:

- A fault-tolerant architecture that improves the reliability of a hybrid memory system including the memory cell array and the decoder.
- A high-performance error correction scheme that corrects clustered and random errors in the memory cell array.
- A fault-tolerant decoder that masks transient and intermittent faults in the decoding circuits.
- A reverse decoding process that results in a cost-effective hardware implementation.

The rest of the article is organized as follows. Section II

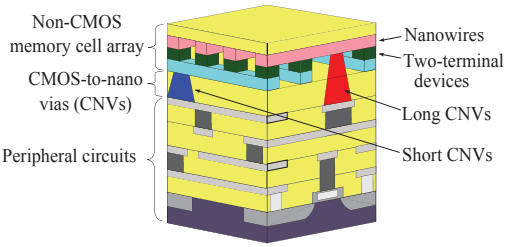



Figure 1: Architecture of CMOL memory

briefly overviews hybrid memory architecture and some of traditional fault-tolerant schemes, which will be used in this work. Section III introduces the memory architecture that tolerates both clustered and random faults in memory cell array using error correction code (ECC) scheme. Section IV improves the architecture by incorporating an on-line masking into the decoder and providing a new decoding process. Section V presents the simulation results for ECC-based memory architecture (presented in Section III). Section VI presents these for the improved architecture and a comparison to the related work. Section VII concludes this article.

## II. Background

This section briefly discusses the architecture of one of the hybrid memories known as *CMOS/Molecular (CMOL)* memories [1], [2], [12]. CMOL memories provide the utmost data storage capacity as huge as 1Tbit/cm$^2$. The description of the architecture includes the structure and operations, as well as the potentials and challenges. Thereafter, three traditional fault-tolerant schemes used in the work including the Redundant Residue Number System code, double modular redundancy, re-execution and Muller C-gates will be covered.

### A. Hybrid memory architecture

Figure 1 shows the architecture of CMOL hybrid memories where a non-CMOS memory cell array is fabricated at the top of CMOS peripheral circuits. The memory cell array consists of nanowires and reconfigurable two-terminal nanodevices. Nanowires fabricated from semiconductor, metal, or carbon nanotubes build up a crossbar-based local interconnect of the memory cell array. Two-terminal nanodevices such as memristor, organic molecules, single electron junctions, memristor, etc. are embedded at each nanowire junction to function as a single memory cell. These non-CMOS devices, nevertheless, are incapable to perform the logical functions; for instance, amplification, inversion, etc. Therefore, nanoscale CMOS is required to function as peripheral circuits such as encoder, decoder, global interconnects, etc. Both circuits are connected using two sets of CMOS-to-nano vias (CNVs). Short CNVs and tall CNVs connect the lower and upper nanowires, respectively, to the peripheral circuits.

In order to write to and read from a memory cell, sufficient voltages are applied from the CMOS peripheral circuits to the non-CMOS memory cell array [1]. For instance, a positive voltage larger than the threshold voltage of the two-terminal nanodevices used as the memory cells will write logic 1. On the other hand, a negative voltage smaller than the threshold voltage of the two-terminal nanodevices will write logic 0. Read operation is accomplished by applying a voltage at a shorter period than that of writing, and the current flow is sensed and is converted into logic states.

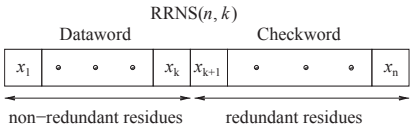



Figure 2: RRNS codeword structure

As aforementioned, hybrid memory is an emerging technology that promises many benefits such as huge data storage capacity [1], [4], [6], low power consumption per unit devices [1], [4], less complex fabrication of the crossbar memory array [1], [6]. However, since hybrid memories are extremely dense and the devices used to build up the circuits are incredibly tiny and sensitive, they are subject to different faults. These faults might emerge from the manufacturing defects, or the operational disturbances due to non-environmental conditions, or the operational disturbances due to environmental conditions. They can occur in any part of the hybrid memories: in the non-CMOS memory array (e.g., missing, impure organic molecules, broken nanowires, electromigration, bit upset, crosstalk, etc.) [2], [4], [6], [7], within the CMOS-to-nano vias (e.g., misalignment, damage, imperfect shape, crack, etc.) [2], [4] or in the CMOS peripheral circuits (e.g., stuck-open, stuck-closed, bit upset, bit latch up, etc.) [21].

Note that the permanent (persistent) faults are due to the manufacturing defects, the intermittent (repetitive) faults are due to non-environmental disturbances and the transient (temporal) faults (also referred to as *soft errors*) are due to environmental disturbances [22].

### B. Traditional fault-tolerant schemes

*1) Redundant residue number system code:* Redundant Residue Number System (RRNS) code comprises two sets of symbol-oriented encoded data that form a codeword as shown in Figure 2 [23]; each symbol is referred to as *residue* $x_i$, where $1 \leq i \leq n$ and $n$ is a positive integer. The first set is referred to as *non-redundant residues* that represent the dataword (input data); whereas, the second set is referred to as *redundant residues* that represent the checkword (for error detection and correction). The non-redundant residues consist of $k$ residues and the redundant residues consist of $(n–k)$ residues, where $k$ and $n$ are positive integers. This code detects $u=n–k$ and corrects $t=\frac{u}{2}=\frac{n-k}{2}$ errors in a codeword.

Each residue of the RRNS code is encoded by performing a modulo operation of an input data $X$ to a set of moduli $m_i$, expressed as $x_i=|X|_{m_i}$ where $1 \leq i \leq n$ [23].

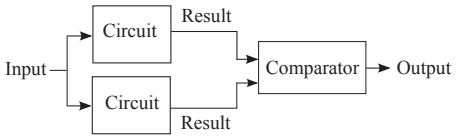
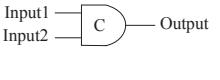



Figure 3: Double modular redundancy configuration




| Input1 | Input2 | Output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | no change |
| 1 | 0 | no change |
| 1 | 1 | 1 |

(a) (b)

Figure 4: Muller C-gate (a) symbol (b) truth table

The bit size of each residue is defined as $b_i = \lfloor \{log_2(m_i - 1) + 1\} \rfloor$ bits; thus, an RRNS codeword has the bit size equal to $\sum_{i=1}^{n} b_i$. RRNS decoding performs *detection*, followed by *correction* if errors are detected in the read data. An error-free RRNS codeword is always within the *operation legitimate range* $LR = 2^d - 1$ when it is decoded, where $d$ is a positive integer denoting the memory word size; the decoded data is sent out of the memory without requiring any correction. An erroneous RRNS codeword will have a value larger than LR; it requires a correction process where an iterative correction is executed to recover a valid data with a maximum of $C_t^n = \frac{n!}{t!(n-t)!}$ iterations [24]. During this phase, $t$ residues are discarded in each iteration and the calculation of $(n-t)$ residues (along with their corresponding parameters) is performed. Any recovered data less than $LR$ is regarded as the valid data and is sent out of the memory. However, if the decoded data is beyond $LR$ after all iterations, an uncorrectable signal will be flagged (to indicate that the decoder cannot correct the erroneous data) and the decoded data is ignored. RRNS has many advantages such as [23], [24]:

- It possesses correction capability of clustered and random errors.
- It can be encoded based on the *low-cost moduli* that enable small area overhead and fast operation.
- Its encoder and decoder sub-units consist of modular circuits that operate in parallel resulting in fast decoding performance.

*2) Double modular redundancy:* Double modular redundancy (DMR) is a hardware redundancy scheme where two identical circuits operate in parallel and their corresponding results are compared, as depicted in Figure 3. Any disagreement from the comparison indicates that faults have occurred. DMR has the advantages of detecting faults occurring in one of its redundant parts and it has a parallel execution allowing high performance operation. In addition, it requires a simple hardware implementation. DMR is a variant of N-modular redundancy (NMR) with majority voter where N is usually an odd integer. NMR has the capability to mask errors due to faults in some of its modules.

The advantages of DMR are as follows:

- It detects faults occurred in one of its redundant parts.
- It executes in parallel allowing high performance operation.
- It requires simple hardware implementation.

*3) Re-execution:* Re-execution is a time redundancy scheme that computes the same data for more than one time. The corresponding results of the multiple executions are stored prior to a comparison.

The advantages of re-execution are as follows:

- It can mask faults of one of its executions.
- It requires small, simple hardware implementation.

*4) Muller C-gate:* Muller C-gate is a simple circuit commonly employed in asynchronous logic applications to mask glitches [25]. Figure 4(a) and (b) shows its symbol and its truth table, respectively. This logic gate only changes its output, to be the same as the input logic value, when the states of all inputs match. On the other hand, the output remains in its previous state if the inputs are dissimilar.

The advantages of Muller C-gate are as follows:

- It can mask glitches (faults) in logic circuits.
- It requires simple hardware implementation.

## III. ECC-based Fault-Tolerant Architecture

This section describes the ECC-based architecture developed to tolerate both random and clustered faults in memory cell array while assuming that no faults occur in the decoder. As shown in Figure 5(a), ECC-based architecture comprises:

- CMOS encoder – it encodes the input data using a modified version of RRNS code referred to as *Double Three-Residue (D3R)* code.
- CMOS decoder – it decodes the D3R code into the output data. The decoding process and the decoding circuit as shown in Figure 5(b) and (c) will be explained later in the section.
- Non-CMOS memory cell array – it stores D3R codewords ($x_1$,$x_2$,$x_3$ and their duplicate $x'_1$,$x'_2$,$x'_3$; see Figure 5(a)).

The rest of this section first will discuss the D3R encoding process, followed by its circuit. Thereafter, the decoding process and circuit will be covered.

### A. D3R encoding process

A Double Three-Residue (D3R) code consists of an RRNS codeword ($C = DW + CW$) and its duplicate ($C' = DW' + CW'$) as depicted in Figure 6. The D3R code is encoded based on the moduli set $m_i = \{2^{\frac{d}{2}} - 1, 2^{\frac{d}{2}+1} - 1, 2^{\frac{d}{2}+1}\}$ [17] where $d$ is a positive integer representing the memory word size. Such a moduli set is considered as the *low-cost moduli*, which results in faster performance and lower area overhead [23]. The first two moduli are used to generate the two-residue dataword $DW = x_1, x_2$ and its duplicate $DW' = x'_1, x'_2$, while the third modulus is used to produce the single-residue checkword $CW = x_1$ and its duplicate $CW' = x'_1$. The bit size of the D3R codeword is $b_{D3R} = 2 \times (\lfloor log_2(m_1 - 1) + 1 \rfloor + \lfloor log_2(m_2 - 1) + 1 \rfloor +$

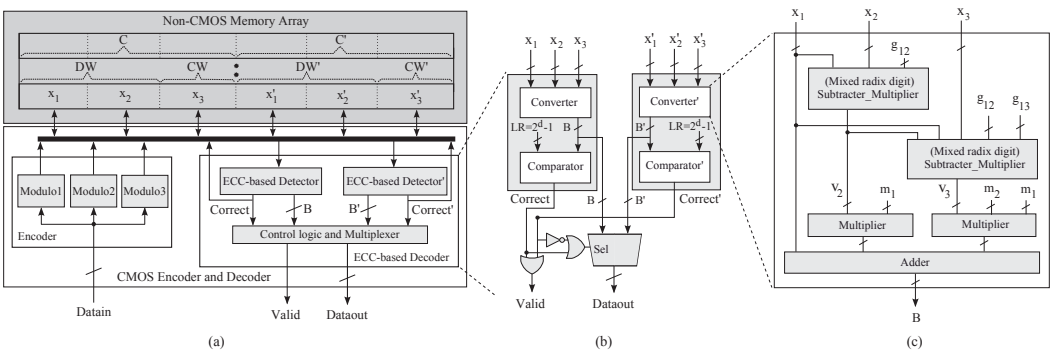



Figure 5: (a) ECC-based architecture (b) units inside the decoder (c) units inside the detector

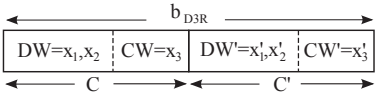



Figure 6: D3R codeword structure

$\lfloor log_2(m_3-1)+1 \rfloor$) where $m_1$, $m_2$ and $m_3$ are the moduli.

Three moduli are used to ensure the detection of a single erroneous residue, i.e., $u$=$n$–$k$=3–2=1. This implies that each D3R codeword part ($C$ and $C'$) is able to detect a single erroneous residue. In case of faults impacting only one of the D3R parts (yet another one is fault-free), this code can tolerate up to three erroneous residues. Such capability is better than that of the error correction capability possessed by RRNS and Reed-Solomon [16], assuming that the codes are composed of an equal number of residues. For example, let us assume an RRNS code consists of six residues ($n$=6), where two residues represent the dataword ($k$=2) and four residues represent the checkword ($n-k$=4). With this setting, RRNS can only correct up to $\frac{n-k}{2}=\frac{6-2}{2}=2$ residues [23], [24].

*D3R encoding circuit*

The bottom-left part of Figure 5(a) shows the block diagram of a D3R encoder constructed of three modulo circuits. These three modulo circuits receive a $d$-bit input word and generate the corresponding residues simultaneously. *Modulo1* and *Modulo2* generate the two-residue dataword $x_1$ and $x_2$, respectively, which then form $DW$=$x_1, x_2$; while *Modulo3* generates a single-residue checkword $CW$=$x_3$. Then, the original codeword $C$=$DW$+$CW$ is duplicated ($C'$=$DW'$+$CW'$) creating a D3R codeword. The duplicate dataword and checkword parts are simply generated by wires and buffers.

*B. D3R decoding process*

As mentioned in Section II-B1, RRNS decoding requires an iterative correction when errors are detected; thus, it impacts the performance of the decoder. However, this is not the case for D3R as it does not discard any residues during the correction phase but swaps between the two codeword parts. The maximum number of swapping is equal to half of the number of residues in a codeword, i.e., $\frac{n}{2}$ (as compared to $C_t^n$ for RRNS). Hence, due to its structure and decoding nature, D3R decoding is faster than that of RRNS. The following steps describe the D3R decoding [17]:

1) *Convert $C$=$x_1$,$x_2$,$x_3$ and $C'$=$x'_1$,$x'_2$,$x'_3$ into binary data $B_1$ and $B'_1$;*
2) *Compare $B$ and $B'$ to the operation legitimate range $LR$=$2^d-1$ where $d$ is the memory word size;*
3) *If ($B$=$B'$)≤$LR$, read out $B$ or $B'$; Break*
   *else if $B$≤$LR$, read out $B$; Break*
   *else if $B'$≤$LR$, read out $B'$; Break*
   *else go to Step 4;*
4) *Check if the maximum number of swapping has been reached. If "yes" go to Step 6, else go to Step 5;*
5) *Swap one of the residues between $C$ and $C'$. Go to Step 1;*
6) *Ignore the output data and invoke a flag indicating that the data is uncorrectable.*

Step 1 performs a conversion of D3R code into binary data. This conversion can be accomplished using either *Chinese Remainder Theorem (CRT)* or *Mixed Radix Conversion (MRC)*. Because CRT is based on large modulo constants that require a complex circuit, MRC is used in this work; the latter conversion algorithm also speeds up the simulation work [23]. MRC is expressed as follows [23]:

$$X = v_1 + v_2 m_1 + v_3 m_2 m_1 + ... + v_n \prod_{i=1}^{n-1} m_i \qquad (1)$$

where $v_1$=$x_1$, $m_i$ are the moduli and $v_i$ are the mixed radix digits calculated as:

$$v_i = \left| \left( ((x_i - v_1) \times g_{1i}) ..... - v_{(i-1)} \right) \times g_{(i-1)i} \right|_{m_i} \qquad (2)$$

where $x_i$=$|X|_{m_i}$ and $g_{(i-u)i}$ are the modular multiplicative inverses of $m_{(i-u)}$ with respect to $m_i$ satisfying $|m_{(i-u)} \times g_{(i-u)i}|_{m_i}$=1; 2≤$i$≤$n$ and 1≤$u$≤$n$−1. Examples of encoding and decoding calculation are available in [16], [24].

Table I
Residues Set for Each Iteration of D3R Correction

| Number of Iteration | Residues | |
|---|---|---|
| | C | C' |
| 1 | $x_1', x_2, x_3$ | $x_1, x_2', x_3'$ |
| 2 | $x_1, x_2', x_3$ | $x_1', x_2, x_3'$ |
| 3 | $x_1, x_2, x_3'$ | $x_1', x_2', x_3$ |

Step 5 swaps the residues between the two D3R codeword parts. Table I gives the swapped residue combinations for each iteration. For example, in the first iteration the swapping is done in such a way that $x_1$ becomes a part of $C'$ and $x_1'$ a part of $C$. Similar steps are performed to the second and third iterations. The residue to be swapped is selected at random, i.e., no fixed rule.

### C. ECC-based decoder circuit

The bottom-right part of Figure 5(a) depicts the block diagram of the ECC-based decoder, which consists of two *detectors* and a *multiplexer*. As illustrated in Figure 5(b), each ECC-based detector is formed by an RRNS-to-binary *converter* and a *comparator*. The output signals of both detectors are multiplexed prior to read out of the memory; the selection is controlled by the *Correct* and *Correct'* signals.

During the detection phase, Converter converts $C$=$x_1$,$x_2$,$x_3$ into a binary word $B$, while Converter' converts $C'$=$x_1'$,$x_2'$,$x_3'$ into a binary word $B'$. The binary words $B$ and $B'$ are then fit into the comparators; they are compared to the operation legitimate range $LR$= $2^d$−1 (where $d$ is memory word size) to determine the validity of the read codeword. If no errors are detected, then both $B$ and $B'$ will be within the operation legitimate range resulting into high *Correct* and *Correct'* signals. This in turn sets the *Valid* signal to high indicating that the output of the multiplexer is a valid read data. However, if faults affect $B$ ($B'$) while $B'(B)$ is fault-free, then the *Correct* (*Correct'*) signal is set to high so that the correct data $B(B')$ is forwarded to *Dataout* signal; note that in this case the *Valid* signal is set to high indicating that the *Dataout* signal is valid. When both $B$ and $B'$ are faulty (i.e., beyond LR), then a correction action is initiated; the converters of Figure 5(b) will then convert the residue sets of the second and the third column of Table I to $B$ and $B'$, respectively. During each iteration, the value of both of $B$ and $B'$ are checked to determine if they are within the legitimate range (i.e., corrected value); if it is the case, then the corrected value is forwarded to *Dataout*. However, if after all iterations the converted are beyond the legitimate range, then both *Correct* and *Correct'* signals will be set to low; this in turn sets the *Valid* signal to low indicating that the *Dataout* signal is not valid.

Figure 5(c) shows the functional units of the converters that produce the binary data (i.e., the implementation of Eq. 1 and 2). Each converter consists of two *mixed radix digit unit*s (formed by subtracters and multipliers), two *multipliers* and an *adder*. As mentioned in Section III-B, the D3R decoding is based on the Mixed Radix Conversion (MRC) algorithm. Typically, MRC executes an RRNS codeword starting from the most significant residue (MSR) and ending with the least significant residue (LSR). In this article MSR is $x_1$ and LSR is $x_3$. Besides these residues, other parameters are required including the moduli $m_i$ and modular multiplicative inverses $g_{(i-u)i}$ where $i$ and $u$ are integers (see Eq. 1 and 2). These parameters can be pre-calculated and are given in Table II for different memory words. $g_{(i-u)i}$ is the input to the two mixed radix digit units in the converter; see Figure 5(c).

Table II
Multiplicative Inverses for ECC-based Architecture

| Memory word, $d$ | Moduli | | Multiplicative inverses |
|---|---|---|---|
| | $m_{(i-u)}$ | $m_i$ | $g_{(i-u)i}$ |
| 16 | $m_1$=255 | $m_2$=511 | $g_{12}$=509 |
| | $m_1$=255 | $m_3$=512 | $g_{13}$=255 |
| | $m_2$=511 | $m_3$=512 | $g_{23}$=511 |
| 32 | $m_1$=65535 | $m_2$=131071 | $g_{12}$=131069 |
| | $m_1$=65535 | $m_3$=131072 | $g_{13}$=65535 |
| | $m_2$=131071 | $m_3$=131072 | $g_{23}$=131071 |
| 64 | $m_1$=4294967295 | $m_2$=8589934591 | $g_{12}$=8589934589 |
| | $m_1$=4294967295 | $m_3$=8589934592 | $g_{13}$=4294967295 |
| | $m_2$=8589934591 | $m_3$=8589934592 | $g_{23}$=8589934591 |

## IV. Improved-based Fault-Tolerant Architecture

The ECC-based architecture described in the previous section tolerates both random and clustered faults in the memory cell array. In this section, the architecture will be extended so that it can tolerate faults in the decoder as well [18]. According to [26], transient faults in combinational logic nodes closer to the output pins have more impact than other circuit parts. Inspecting Figure 7(a) reveals that the decoder/multiplexer is the closest unit to the output pins. Thus, making this unit fault tolerant will improve the overall reliability of the memory architecture. The bottom-right part of Figure 7(a) illustrates the block diagram of the Improved-based decoder comprising two *modified detectors* and a group of *Muller C-gates*. The decoding is modified both in terms of its circuit implementation and its decoding process. The rest of this section first will discuss the improved decoding implementation; thereafter, the decoding process and the Muller C-gates.

### A. Improved decoding circuit

The decoding process is based on: (i) the multiplication of residues $r_i$ and multiplicative inverses $g_{(i-u)i}$ where $i$ and $u$ are integers, (ii) the subtraction, and (iii) the summation; see Eq. 1 and 2. Making the $g_{(i-u)i}$ values smaller reduces the multiplication complexity. Moreover, there is no need for multiplication if the $g_{(i-u)i}$ values are made equal to 1, and there are only shift operations if the $g_{(i-u)i}$ values are power of 2. These are the basic ideas used to develop the new decoder.

To explain how the new $g_{(i-u)i}$ values are generated, let us assume the moduli set $m_i$ for 16-bit memory word (see Table II for $d$=16); herein, $m_i$=$\{255, 511, 512\}$ where

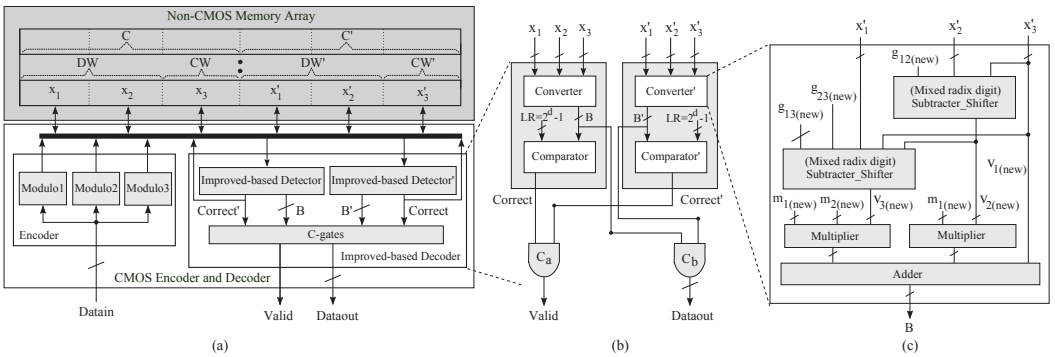



Figure 7: (a) Improved-based architecture (b) units inside the decoder (c) units inside the detector

Table III
Multiplicative Inverses for Improved-based Architecture

| Memory word, $d$ | Moduli | | Multiplicative inverses |
|---|---|---|---|
| | $m_{(i-u)(new)}$ | $m_{i(new)}$ | $g_{(i-u)i(new)}$ |
| 16 | $m_{1(new)}$=512 | $m_{2(new)}$=511 | $g_{12(new)}$=1 |
| | $m_{1(new)}$=512 | $m_{3(new)}$=255 | $g_{13(new)}$=128 |
| | $m_{2(new)}$=511 | $m_{3(new)}$=255 | $g_{23(new)}$=1 |
| 32 | $m_{1(new)}$=131072 | $m_{2(new)}$=131071 | $g_{12(new)}$=1 |
| | $m_{1(new)}$=131072 | $m_{3(new)}$=65535 | $g_{13(new)}$=32768 |
| | $m_{2(new)}$=131071 | $m_{3(new)}$=65535 | $g_{23(new)}$=1 |
| 64 | $m_{1(new)}$=8589934592 | $m_{2(new)}$=8589934591 | $g_{12(new)}$=1 |
| | $m_{1(new)}$=8589934592 | $m_{3(new)}$=4294967295 | $g_{13(new)}$=2147483648 |
| | $m_{2(new)}$=8589934591 | $m_{3(new)}$=4294967295 | $g_{23(new)}$=1 |

255 is the *most significant moduli (MSM)* and 512 is the *least significant moduli (LSM)*. In the new design, however, these moduli are put in the opposite sequence where 512 becomes the MSM and 255 the LSM; hence $m_{i(new)}$=$\{512, 511, 255\}$. With these values, the new multiplicative inverses will be calculated using the equation $|m_i \times g_{(i-u)i}|_{m_{(i-u)}}$=1; see Eq. 2. Note that this calculation is based on a trial and error [23], i.e., increasing the $g_{(i-u)i}$ value by one in each attempt. Substituting $m_{1(new)}$=512, $m_{2(new)}$=511 and $g_{12(new)}$=1 result in $|512\times1|_{511}$=1, meaning that the $g_{12(new)}$=1 is the desired value. This is also the case for $m_{2(new)}$=511 and $m_{3(new)}$=255, where $g_{23(new)}$=1 is the desired value. However, substituting $m_{1(new)}$=512 and $m_{3(new)}$=255 with $g_{13(new)}$=1 result in $|512\times1|_{255}$=2; in this case $g_{13(new)}$=1 does not satisfy the RRNS uniqueness requirement [23]. Yet, after several attempts the desired multiplicative inverse is obtained, i.e., $g_{13(new)}$=128, which is a power of 2.

It is interesting to note that the two new modular multiplicative inverses $g_{12(new)}$=$g_{23(new)}$=1 regardless of the memory word size, and $g_{13(new)}$=$2^{\frac{d}{2}-1}$ where $d$ is the memory word size. This significantly reduces the decoding implementation as compared to the conventional design. Table III gives the new modular multiplicative inverses for different memory word sizes.

### B. Improved decoding process

In addition to simplifying the decoding circuit, the decoding process has been modified in order to improve the overall memory reliability. For the decoding process of Fig 5(a), each detector requires up to three iterations to recover the read data in case of a fault; see Table I. In the improved design, each detector will execute up to six iterations; i.e., each detector will execute the RRNS-to-binary conversion for all six residue sets shown in Table I. Note that the execution of a next iteration is needed only if the obtained converted data is larger than the legitimate range. It is clear that the new decoder requires up to twice as many iterations as the former decoder, which in turn impacts the performance. To compensate for this performance loss, the decoding process is modified and made cost-efficient using a *reversing decoding* process; this is explained next.

As mentioned before, the new decoding circuit reverses the MSM and LSM as compared to that of ECC-based architecture. Therefore, the decoding process has to be reversed as well. For each residue set (i.e., codeword) of Table I, the conversion to binary starts at the most right residue followed by the residue to the left. For example, for the original codeword part $C$ the conversion starts first at $x_3$, then $x_2$ and finally $x_1$. This is exactly the reverse order of the operation performed with the decoder of ECC-based architecture.

In order to provide more insight into the modified decoding procedure, an example of encoding and decoding for 16-bit input data is given. The multiplicative inverses $g_{(i-u)i(new)}$ are given in Table III and the operation legitimate range is $LR$=$2^{16}$−1=65535.

**Encoding:**

- Assume an input data $X$=65535.
- The D3R original codeword part $C$ will be:

  $C$=$\{|65535|_{255}, |65535|_{511}, |65535|_{512}\}$=$\{0, 127, 511\}$
- Then, $C$ is duplicated to $C'$ and both parts are combined to produce $D3R$=$\{0, 127, 511, 0, 127, 511\}$; the D3R codeword is stored in the memory cell array.

**Decoding:**

- Assume that faults induce clustered errors that corrupt all three residues of $C'$ during storage resulting into $D3R$=$\{0, 127, 511, \mathbf{3}, \mathbf{255}, \mathbf{31}\}$. When reading, both $C$ and $C'$ are retrieved for decoding.

- Decoding starts by converting both $D3R$ codeword parts simultaneously. This process has two steps:
  - Calculate the mixed-radix digits $v_i$ and $v'_i$
    For $C$, the $v_i$ values are:

$$v_{1(new)} = x_3 = 511$$

$$v_{2(new)} = \left|(x_2 - v_{1(new)}) \times g_{12(new)}\right|_{m_{2(new)}} = \left|(127 - 511) \times 1\right|_{511} = 127$$

$$v_{3(new)} = \left|\left((x_1 - v_{1(new)}) \times g_{13(new)} - v_{2(new)}\right) \times g_{23(new)}\right|_{m_{3(new)}} = \left|\left((0 - 511) \times 128 - 127\right) \times 1\right|_{255} = 0$$

    For $C'$, the $v'_i$ values are:

$$v'_{1(new)} = x'_3 = 31$$

$$v'_{2(new)} = \left|(x'_2 - v'_{1(new)}) \times g_{12(new)}\right|_{m_{2(new)}} = \left|(255 - 31) \times 1\right|_{511} = 224$$

$$v'_{3(new)} = \left|\left((x'_1 - v'_{1(new)}) \times g_{13(new)} - v'_{2(new)}\right) \times g_{23(new)}\right|_{m_{3(new)}} = \left|\left((3 - 31) \times 128 - 224\right) \times 1\right|_{255} = 17$$

  - Calculate the corresponding binary data $B$ and $B'$
    For $C$, the $B$ value is:

$$B = v_{1(new)} + (v_{2(new)} \times m_{1(new)}) + (v_{3(new)} \times m_{1(new)} \times m_{2(new)}) = 511 + (127 \times 512) + (0 \times 512 \times 511) = 65535$$

    For $C'$, the $B'$ value is:

$$B' = v'_{1(new)} + (v'_{2(new)} \times m_{1(new)}) + (v'_{3(new)} \times m_{1(new)} \times m_{2(new)}) = 31 + (224 \times 512) + (17 \times 512 \times 511) = 4562463$$

- Compare both $B$ and $B'$ to the legitimate range
  - $B$=$LR$, while $B'$>$LR$
- Read out $B$=65535 and break (no correction is required).

### C. Fault-tolerant decoding circuit

The C-gates are used to replace the multiplexer in the ECC-based architecture, as shown in Figure 7(a) and (b). The inputs of the C-gates are connected to the outputs of the two detectors. The reason for using C-gates instead of the multiplexer is that these asynchronous logic gates are able to mask short-period glitches (due to intermittent and transient faults) produced by the detectors. In practice, the probability of two glitches occuring simultaneously at two pins carrying the same signal is very low. For example, the probability that glitches occur simultaneously at one output of a 64-bit *Detector* and one output of a 64-bit *Detector'* is $(\frac{1}{64})^2$=2.44×10$^{-4}$. This is where C-gates show their superiority because even if there are many short-period glitches, as long as they occur at different times and/or at different pins, the output data is still unchanged.

## V. Evaluation of ECC-based Architecture

This section presents the experimental evaluation and analysis of the proposed ECC-based architecture. A comparison to existing ECCs is carried out. Three attributes are compared among the ECCs: fault tolerance capability, codeword size and decoding time performance.

### A. Simulation setup

The simulation model consists of 4K×64-bit memory system. All parts of the memory (i.e., encoder, decoder, memory cell array, fault injection, etc.) were built using MATLAB script. The intention of the-to-be-evaluated D3R architecture is to recover the read data when faulty as long as three residues from the six stored in the memory cell array are fault-free. For comparison with the existing work, three symbol-based ECCs are considered:

- C-RRNS [24] – This ECC always consists of three residues representing the dataword. For the correction capability of these three residues, C-RRNS has to make use of six residues representing the checkword. Therefore, the moduli set of $\{2^p-1, 2^p, 2^p+1, a_1, a_2, ..., a_6\}$ will be used; where $p$ is the minimum integer satisfying the RRNS requirements and $a_i$ is a prime number larger than $2^p$+1 [24]. The first three moduli are used to encode the dataword and the last six moduli to encode the checkword. For the experiment, the $p$ values are $p$=6, 11, and 22 for 16, 32, and 64-bit memory word, respectively.
- RS [27] – This ECC consists of two symbols representing the dataword. For the correction of these two symbols, RS has to make use of four symbols representing the checkword. Galois Field of degree $q$ denoted as GF($2^q$) is used where each symbol consists of a q-bit data satisfying RS requirement [27]. For the experiment, the $q$ values are $q$=8, 16, and 32 for 16, 32, and 64-bit memory word, respectively.
- 6M-RRNS [16] – This ECC consists of two residues representing the dataword. For the correction capability of these two residues, 6M-RRNS has to make use of four residues representing the checkword. Therefore, the moduli set of $\{2^p, 2^p+1, 2^{p-1}-1, 2^{p-2}-1, 2^{p-3}-1, 2^{p-4}+1\}$ will be used; where $p$ is the minimum integer satisfying two of three RRNS requirements [16]. The first two moduli are used to encode the dataword and the last four moduli to encode the checkword. For the experiment, the $p$ values are $p$=8, 16, and 32 for 16, 32, and 64-bit memory word, respectively.

Faults were randomly injected into the memory cell array with fault rates ranging from 1% to 10%. The faults flip a number of adjacent bits (representing the clustered errors) that form the codewords of the considered ECCs.

### B. Fault tolerance capability and codeword size

Figure 8 shows the capability of the considered ECCs to tolerate faults that induce clustered and random errors

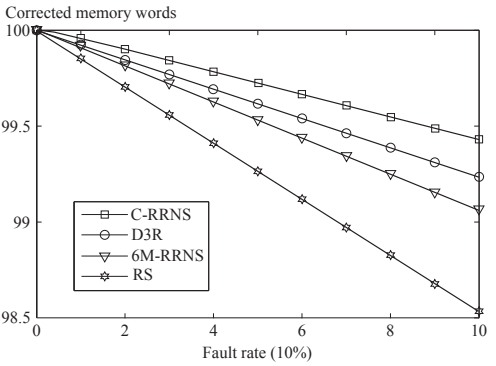



Figure 8: Fault tolerance capability for 64-bit memory

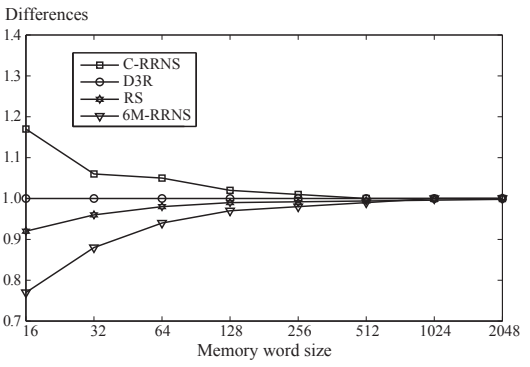



Figure 9: Codeword size of the considered ECCs

in the memory cell array. This capability is represented by the percentage of corrected memory words. Regardless of the fault rates, C-RRNS provides the best correction capability among the considered ECCs followed by D3R, 6M-RRNS and RS. However, the difference between C-RRNS and D3R capability is negligible, e.g., it is only $0.2\%$ at $10\%$ fault rate. The difference becomes even smaller for memories with larger words [16]. Next, the capability of each ECC will be elaborated.

D3R code ensures the correction of up to three erroneous residues *if faults do not affect a residue and its duplicate at the same time*; i.e., if at least one of the codeword shown in Table I is not affected, then the data can be recovered. C-RRNS code always ensures the correction of up to three erroneous residues *regardless of the erroneous residue combination*. However, the bit size of its varied-size residues is longer than those of the other three ECCs. RS code always ensures the correction of up to two erroneous symbols *regardless of the erroneous symbol combination*. Nevertheless, the bit size of its fixed-size symbols is shorter than those of the other three ECCs. 6M-RRNS code always ensures the correction of up to two erroneous residues *regardless of the erroneous residue combination* [16]. Note that in general, D3R can recover up to $t{+}1{=}\frac{n-k}{2}{+}1$ residues, while C-RRNS, RS and 6M-RRNS each can only correct up to $\frac{n-k}{2}$ residues.

One can conclude that D3R provides a comparable correction capability as C-RRNS, and at a lower overall codeword size; this will result in a smaller area overhead. Figure 9 depicts the codeword size of the considered ECCs normalized to D3R. It clearly shows that for memory word size lower than 256 bits, the D3R codeword size is smaller than that of C-RRNS; e.g., for a 64-bit memory, the difference is about $6\%$. For very large memory word size ($>$512-bit), the difference in the codeword size becomes negligible.

### C. Decoding performance

All the considered ECCs require an iterative correction process during decoding. D3R code ensures the correction of three residues by performing up to $C_1^3{=}\frac{3!}{1!(3-1)!}{=}3$ iterations. Note that although D3R has six residues, the correction process executes in parallel. C-RRNS code ensures the correction of three residues by performing up to $C_3^9{=}\frac{9!}{3!(9-3)!}{=}84$ iterations [24]. RS code ensures the correction of two symbols by performing up to $C_2^6{=}\frac{6!}{2!(6-2)!}{=}15$ iterations [27]. 6M-RRNS ensures the correction of three residues by performing up to $C_2^6{=}\frac{6!}{2!(6-2)!}{=}15$ iterations and an additional likelihood decoding step [16]. Note that in general, D3R requires a maximum of $C_{\frac{t}{2}}^n$ iterations, C-RRNS and RS require a maximum of $C_t^n$ iterations, and 6M-RRNS requires a maximum of $C_t^n{+}1$ iterations. Therefore, the decoding performance of D3R is much better than those of the other three ECCs; i.e., it is $\frac{84}{3}{=}28\times$ faster than C-RRNS, and $5\times$ faster than RS and 6M-RRNS. Furthermore, these differences increase for higher error correction capability.

## VI. Evaluation of Improved-based Architecture

This section gives the evaluation of the Improved-based architecture and compares it first to the ECC-based architecture; thereafter, to the related work.

As mentioned before, the ECC-based architecture is based on the conventional MRC algorithm and the decoding parameters given in Table II; see also Figure 5 for the circuit. In contrast, the Improved-based architecture is based on the modified MRC algorithm and the decoding parameters presented in Table III; see also Figure 7 for the circuit. The implementation of the decoders of both architectures was done using VHDL on Xilinx ISE and Synopsys Design Compiler tools based on 90nm CMOS technology.

The reliability evaluation was carried out using Matlab simulation. Besides injecting faults into the memory cell array as mentioned in Section V, the faults were also injected to the decoders with the ratio of 1:10 (decoder:memory cell array). This ratio is set based on the soft error rates between SRAM bit and logic for 90nm CMOS technology reported in [21].

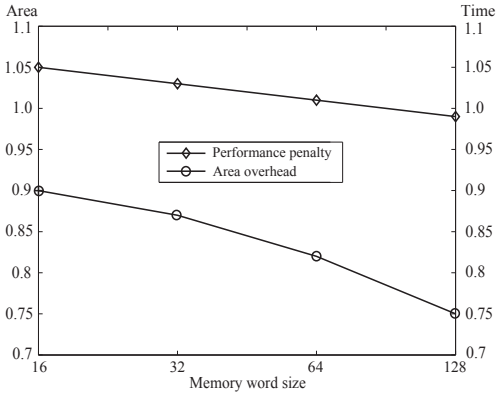



Figure 10: Area overhead and performance penalty of Improved-based decoder

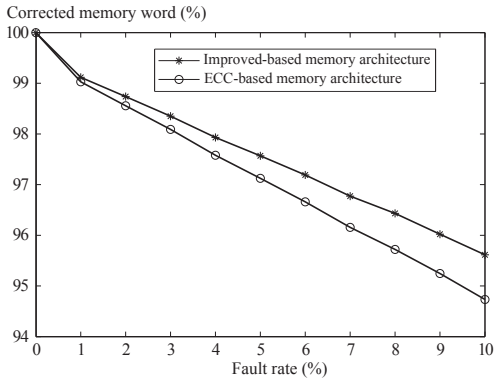



Figure 11: Reliability of hybrid memory using both architectures

### A. Comparison to ECC-based architecture

For comparison to the ECC-based architecture, three attributes are considered: area overhead, performance penalty and memory system reliability. Figure 10 illustrates the area overhead and performance penalty of the Improved-based decoder normalized to the ECC-based decoder. It clearly shows that the required area overhead is smaller and the reduction becomes significant for larger memory words. For example, for a word size of 128 bits, the Improved-based decoder consumes 25% less area than that of the ECC-based decoder. The figure also indicates that for smaller word size ($\leq$64 bits), the Improved-based decoder is slightly slower ($\leq$5%) than the ECC-based decoder. For larger word size, the performance is anticipated to be better than the ECC-based decoder.

Figure 11 shows the simulation results of memory system reliability for the two proposed architectures for a 64-bit memory word. Clearly, the Improved-based decoder provides better system reliability than the ECC-based decoder irrespective of the fault rate. The difference becomes larger at higher fault rate, e.g., 10$\times$ greater at 10% fault rate as compared to 1% fault rate. Thus, the Improved-based decoder is able to improve the memory system reliability especially at higher fault rates.

Table IV
Comparison of the ECCs Proposed in [15] and in This Article

| **EG-LDPC** [15] | **D3R** | **Differences (D3R/EG-LDPC)** | |
|---|---|---|---|
| $(n, k, t)$ | $(n, k, t)$ | Fault tolerance capability | Required codeword size |
| 63,37,4 | 100,32,50 | $\frac{50}{100}/\frac{4}{63}$=8.3 | 100/63=1.59 |
| 255,175,8 | 388,128,194 | $\frac{194}{388}/\frac{8}{255}$=16.7 | 388/255=1.52 |

### B. Comparison to related work

As mentioned in Section I, no published work has addressed the same problem for hybrid memories as considered in this article except [15]. However, the error correction scheme used in [15] targets only *random* errors using Euclidean Geometry Low-Density Parity-Check (EG-LDPC) code. Contrarily, the ECC proposed in this work targets both *random* and *clustered* errors using Double Three-Residue (D3R). Despite these differences, the results from [15] will be used to compare the following attributes: error correction capability and area overhead of the memory cell array.

Table IV summarizes the parameters used to develop both EG-LDPC and D3R codes. The parameters $n, k, t$ denote the codeword size, the dataword (memory word) size and the error correction capability, respectively. Note that the memory size for D3R is chosen to be the nearest to the one reported in [15] for fair comparison.

The third column of Table IV gives the ratio of the corrected bits of the two ECCs. For example, for 32-bit memory, D3R can correct up to 50% of bits that form its six-residue codeword, while EC-LDPC can correct only 6% [15]; this results in an improved ratio of a factor of 8.3. The improvement becomes even more significant for larger memory words.

The fourth column of Table IV presents the ratio of the codeword size of D3R and EG-LDPC; it clearly shows that the improvement in terms of error correction capability of D3R comes at the cost of 50% larger codeword size as compared to EG-LDPC.

## VII. Conclusion

This article presented a high-performance, fault-tolerant architecture for reliability improvement of hybrid memory system. The architecture addresses faults that induce random and clustered errors in the memory cell array and the decoder. The fault-tolerant architecture combines two techniques: (i) an error correction scheme that tolerates both clustered and random faults in memory cell array, and (ii) an on-line masking based on asynchronous logic gates to tolerate faults in the decoder. In addition, the area and performance of the decoding circuit is optimized by reversing the decoding process.

Experimental evaluation shows that the proposed architecture provides 5$\times$ faster decoding performance, a comparable memory cell array area overhead and competitive

reliability as compared to the existing symbol-based ECCs such as Reed-Solomon and Redundant Residue Number System codes. It is worth mentioning that faults in the encoder are not considered in this work. Investigation on new fault tolerance schemes to combine with the architecture could be performed in the future. In summary, the proposed architecture offers an attractive solution for achieving a fast, reliable hybrid nanoelectronic memory.

## REFERENCES


[1] D.B. Strukov and K.K. Likharev, "Prospects for terabit-scale nanoelectronic memories", *J. Nanoscience and Nanotechnology*, vol. 16, no. 1, 2005, pp. 137–148.

[2] K.K. Likharev, "Hybrid cmos/nanoelectronic circuits: opportunities and challenges", *J. of Nanoelectronics and Optoelectronics*, vol. 3, no. 3, 2008, pp. 203–230.

[3] R. Waiser and M. Aono, "Nanoionics-based resistive switching memories", *Nature Materials*, vol. 6, 2007, pp. 833–840.

[4] R.J. Luyken and F. Hofmann, "Concept for hybrid cmos-molecular non-volatile memories", *J. Nanoscience and Nanotechnology*, vol. 14, no. 2, 2003, pp. 273–276.

[5] L.B. Kish and P.M. Ajayan, "TerraByte flash memory with carbon nanotubes", *Applied Physics Letters*, vol. 86, no. 9, 2005, pp. 1–2.

[6] A. DeHon, S.C. Goldstein, P.J. Kuekes and P. Lincoln, "Nonphotolithographic nanoscale memory density prospects", *IEEE Trans. on Nanotechnology*, vol. 4, no. 2, 2005, pp. 215–228.

[7] A. DeHon, "Nanowire-based programmable architectures," *ACM J. Emerging Technology in Computing System*, vol. 1, no. 2, 2005, pp. 109-162.

[8] L. Rispal and U. Schwalke, "Large-Scale in situ fabrication of voltage-programmable dual-layer high-*kappa* dielectric carbon nanotube memory devices with high on/off ratio", *IEEE Electron Device Lett.*, vol. 29, no. 12, Dec. 2008, pp. 1349–1352.

[9] M. Mishra and S.C. Goldstein, "Defect tolerance at the end of the roadmap", in *Proc. of International Test Conference*, vol. 1, 2003, pp. 1201–1211.

[10] P. Lincoln, "Challenges in scalable fault tolerance", in *Proc. of IEEE/ACM International Symposium on Nanoscale Architectures*, 2009, pp. 13–14.

[11] A. Orailoglu, "Nanoelectronic architectures: reliable computation on defective devices", in *Digest of Workshop on Dependable and Secure Nanocomputing*, 2007.

[12] D.B. Strukov and K.K. Likharev, "Defect-tolerant architectures for nanoelectronics crossbar memories", *J. Nanoscience and Nanotechnology*, vol. 7, no. 1, 2007, pp. 151–167.

[13] C.M. Jeffery and R.J.O. Figueiredo, "Hierarchical fault tolerance for nanoscale memories", *IEEE Trans. on Nanotechnology*, vol. 5, no. 4, 2006, pp. 407–414.

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

[15] H. Naeimi and A. DeHon, "Fault secure encoder and decoder for nanomemory applications", *IEEE Trans. on Very Large Scale Integration Systems*, vol. 17, no. 4, 2009, pp. 473–486.

[16] N.Z. Haron and S. Hamdioui, "Redundant residue number system code for fault-tolerant hybrid memories", *ACM J. of Emerging Technologies in Computing Systems*, vol. 7, no. 1, article 4, 2011, pp. 1–19.

[17] N.Z. Haron and S. Hamdioui, "High-performance cluster-fault tolerance scheme for hybrid nanoelectronic memories", in *Proc. of IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*, 2010, pp. 144–151.

[18] N.Z. Haron and S. Hamdioui, "Cost-efficient fault-tolerant decoder for hybrid nanoelectronic memories system", in *Proc. of Design, Automation and Test in Europe*, 2011, pp. 265–268.

[19] S. Biswas, T.S. Metodi, F.T. Chong and R. Kastner, "A pageable, defect-tolerant nanoscale memory system", in *Proc. of IEEE International Symposium on Nanoscale Architecture*, 2007, pp. 85–92.

[20] S. Srivastava, A. Melouki and B.M. Al-Hashimi, "Defect tolerance in hybrid nano/cmos architecture using tagging mechanism", in *Proc. of IEEE International Symposium on Nanoscale Architectures*, 2009, pp. 43–46.

[21] R.C. Baumann, "Radiation-induced soft errors in advanced semiconductor technologies" *IEEE Trans. on Device and Materials Reliability* vol. 5, no. 3, 2005, pp. 305–316.

[22] N. Jha and S. Gupta, *Testing of digital systems*. Cambridge: Cambridge University Press, 2003.

[23] N. Szabo and R. Tanaka, *Residue arithmetic and its application to computer technology*. New York: McGraw-Hill, 1967.

[24] F. Barsi and P. Maestrini, "Error correcting properties of redundant residue number systems", *IEEE Trans. of Computers*, vol. 22, no. 3, 1973, pp. 307–315.

[25] D.E. Muller and W.S. Bartky, "A theory of asynchronous circuits", in *Proc. of International Symposium Theory of Switching*, 1959, pp. 204-243.

[26] K. Mohanram and N.A. Touba, "Cost-effective approach for reducing soft error failure rate in logic circuits", in *Proc. of International Test Conference*, 2003, pp. 893–901.

[27] L. Hanzo, T.H. Liew and B.L. Yeap, *Turbo coding, turbo equalisation and space-time coding for transmission over fading channels*. West Sussex: John Wiley & Sons, 2002.