Redundant Residue Number System Code for Fault-Tolerant Hybrid Memories

NOR ZAIDI HARON, Delft University of Technology and Universiti Teknikal Malaysia Melaka
SAID HAMDIOUI, Delft University of Technology

Hybrid memories are envisioned as one of the alternatives to existing semiconductor memories. Although offering enormous data storage capacity, low power consumption, and reduced fabrication complexity (at least for the memory cell array), such memories are subject to a high degree of intermittent and transient faults leading to reliability issues. This article examines the use of Conventional Redundant Residue Number System (C-RRNS) error correction code, which has been extensively used in digital signal processing and communication, to detect and correct intermittent and transient cluster faults in hybrid memories. It introduces a modified version of C-RRNS, referred to as 6M-RRNS, to realize the aims at lower area overhead and performance penalty. The experimental results show that 6M-RRNS realizes a competitive error correction capability, provides larger data storage capacity, and offers higher decoding performance as compared to C-RRNS and Reed-Solomon (RS) codes. For instance, for 64-bit hybrid memories at 10% fault rate, 6M-RRNS has 98.95% error correction capability, which is 0.35% better than RS and 0.40% less than C-RRNS. Moreover, when considering 1Tbit memory, 6M-RRNS offers 4.35% more data storage capacity than RS and 11.41% more than C-RRNS. Additionally, it decodes up to 5.25 times faster than C-RRNS.

Categories and Subject Descriptors: B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance

General Terms: Reliability

Additional Key Words and Phrases: Hybrid memories, nanowire, residue number system, fault-tolerance, error correction codes

ACM Reference Format:
Haron, N. Z. and Hamdioui, S. 2011. Redundant residue number system code for fault-tolerant hybrid memories. ACM J. Emerg. Technol. Comput. Syst. 7, 1, Article 4 (January 2011), 19 pages.
DOI = 10.1145/1899390.1899394. http://doi.acm.org/10.1145/1899390.1899394.

1. INTRODUCTION

Hybrid memories, the emerging memory architecture that combines CMOS and non-CMOS devices in a single chip, are envisioned as one of the alternatives to existing semiconductor memories. These memories use non-CMOS devices such as nanowire, carbon nanotubes, molecules, and so on, instead of transistors and/or capacitors to

---

This article is an extended version of the paper "Residue-Based Code for Reliable Hybrid Memories," presented at the IEEE/ACM International Symposium on Nanoscale Architecture, 2009.
Authors' addresses: N. Z. Haron (corresponding author) and S. Hamdioui, Computer Engineering Laboratory, Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology, Mekelweg 4, 2628 CD, Delft, The Netherlands.
N. Z. Haron is on leave from the faculty of Electronics and Computer Engineering, Universiti Teknikal Malaysia Melaka, Hang Tuah Jaya, Durian Tunggal, 76100, Melaka, Malaysia.

DOI 10.1145/1899390.1899394 http://doi.acm.org/10.1145/1899390.1899394

structure the memory cell array. Examples are molecular-based memories [Bullis; CALMEC; Kügeler et al. 2009; Luyken and Hofmann 2003; Reed et al. 2007; Strukov and Likharev 2004; Waiser and Aono 2007; Zettacore], carbon nanotube-based memories [DeHon et al. 2005; Kish and Ajayan 2005; Rispal and Schwalke 2008], and so on. On the other hand, these memories still employ scaled CMOS devices to form the peripheral circuits such as the encoder, the decoder, the interconnects, and so on. Such an architecture realizes an enormous capacity of data storage (up to 1Terabit per cm^2), consumes low power and requires modest fabrication complexity (at least for the memory cell array) [Strukov and Likharev 2004; Strukov 2006; Likharev 2008]. The tiny size of the CMOS and non-CMOS devices as well as the higher density of the memory cell array cause a major challenge in producing reliable hybrid memories. As reported in Mishra and Goldstein [2003], the defect rate of 10% (about six orders of magnitude higher than existing technology) is expected due to the imperfection of the manufacturing process. Moreover, a high degree of intermittent and transient faults is anticipated to occur, which can originate from either process variabilities or operational disturbances [Lincoln 2009; Orailoglu 2007]. Furthermore, it is highly probable that these faults will impact adjacent cells in the memory cell array and transistors in the peripheral circuits [Orailoglu 2007] causing cluster faults.

Considerable research work has applied fault tolerance techniques to deal with the high degree of faults in hybrid memories [Ghosh and Lincoln 2008; Jeffery and Figueiredo 2006; Naeimi and DeHon 2009; Strukov and Likharev 2004, 2007; Sun and Zhang 2007]. The similarity among these efforts is the use of error correction codes (ECCs). ECCs possess concurrent error correction capability at lower cost as compared to the other fault tolerance techniques. For example, Hamming code is used by Strukov and Likharev [2004] and Jeffery and Figueiredo [2006]; however, this ECC can only correct small number of faults. Bose-Chaudhuri-Hocquenghem (BCH) code is used by Strukov and Likharev [2007], Sun and Zhang [2007], and Biswas et al. [2007]; nevertheless, it imposes higher computational complexity and targets random faults. Low Parity Density-Check (LDPC) code is used by Naeimi and DeHon [2009] and Ghosh and Lincoln [2008]; although this ECC provides a better error correction capability than Hamming, it is worse than BCH. Above all, these ECCs target either single or multiple random faults, but not cluster faults; something one has to care about in hybrid memories.

Redundant Residue Number System (RRNS) code, derived from Residue Number System (RNS), has been extensively used in high speed arithmetic operation applications (e.g., digital signal processing, cryptography, communication, etc.) [Sun and Krishna 1992; Szabo and Tanaka 1967; Yang and Hanzo 2001]. This code inherits the advantages of RNS such as parallelism, modularity, and unweighted number system representation resulting in fast and fault tolerance properties. RRNS is suitable to correct cluster faults and has similar error correction capability to that of Reed-Solomon code. Therefore, RRNS could be used to improve the reliability of hybrid memories. However, conventional RRNS code employed in the existing applications is subject to high area overhead and performance penalty. Consequently, if the code is applied to hybrid memories directly without any modification and adaptation, it will reduce the data storage capacity and degrade the memory performance.

This article presents the concept of RRNS code as a fault tolerance technique to improve the reliability of hybrid memories at lower cost. The conventional RRNS is modified by devising the moduli set, which then produces a new version of RRNS code referred to as Six-Moduli RRNS (6M-RRNS). To the best of the authors' knowledge, this article is the first to adapt RRNS code to memory systems. Our simulation and analysis on 16, 32, and 64-bit memories show that 6M-RRNS code is able to provide sufficient reliability improvement at minimal impact on area. The preliminary

work of using C-RRNS and 6M-RRNS codes for hybrid memories was reported by the authors [Haron and Hamdioui 2009]. The main contributions of this article are the following.

— The introduction and the application of Redundant Residue Number System code to correct cluster intermittent and transient faults in hybrid memory systems;
— the modification and the adaptation of conventional RRNS (C-RRNS) in order to achieve cluster fault correction at lower cost. For this, a modified version of C-RRNS, referred to as Six-Moduli RRNS (6M-RRNS) code, is introduced. 6M-RRNS provides a competitive error correction capability, larger user data capacity, and faster decoding latency as compared to C-RRNS and Reed-Solomon codes. The hardware implementation of 64-bit 6M-RRNS encoder and decoder structured based on 32nm CMOS technology shows that they only add 0.005% area overhead to 1cm^2 hybrid memories.

The rest of the article is organized as follows. Section 2 reviews the structure, operation, potentials, and challenges of hybrid memories. Section 3 defines the fundamental theory of RRNS code. Section 4 describes the construction of the error correction codes investigated in this work; this includes the conventional RRNS, Reed-Solomon, and the proposed 6M-RRNS codes. Section 5 presents the experimental set up, simulation results, and quantitative analysis. Section 6 gives the hardware implementation of the encoder and decoder of 6M-RRNS code. Section 7 concludes the article.

2. HYBRID MEMORIES

In this section we briefly review the architecture of one of the hybrid memories, known as CMOS/Molecular (CMOL) memory [Likharev 2008; Strukov and Likharev 2004, 2007]. CMOL memory provides the utmost data storage capacity, as huge as 1Tbit per cm^2. First, we describe the structure and operations of CMOL memory. Thereafter, we discuss their potentials and challenges. Finally, we explain the architecture of the memory incorporated with this ECC scheme. More details about CMOL memory are given in Strukov and Likharev [2004, 2007].

2.1 Structure

Figure 1 shows the structure of CMOL memory, in which a non-CMOS-based memory cell array is fabricated on top of the CMOS-based peripheral circuits [Strukov and Likharev 2004]. The non-CMOS-based memory cell array consists of two perpendicular planes of nanowires with reconfigurable two-terminal nanodevices embedded at each nanowire junction. Nanowires fabricated from semiconductor, metal, or carbon nanotubes build up a matrix-like local interconnect of the memory cell array. Two-terminal non-CMOS devices, such as organic molecules, single electron junctions and phase change materials function as a single memory cell [Likharev 2008]. However, these non-CMOS-based devices are incapable of performing the logical functions, e.g., amplification. Therefore, nanoscale CMOS is required to implement the peripheral circuits like the encoder, the decoder, and the global interconnect. The non-CMOS-based and CMOS-based circuits are connected using two sets of cone-type CMOS-to-nano interface pins. Metals or silicon can be used as the interface pins [Likharev 2008].

2.2 Operations

To write to and read from a specific memory cell, a sufficient voltage is biased from the CMOS-based peripheral circuits to the non-CMOS-based memory cells through the interface pins. Biasing a positive voltage larger than the threshold voltage of the two-terminal nanodevices will write logic 1. Contrarily, supplying a negative voltage

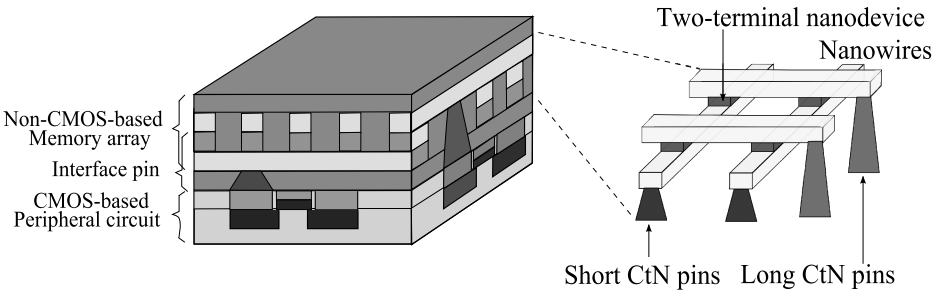

Fig. 1. Generic structure of hybrid memory.

smaller than the threshold voltage of the two-terminal nanodevices will write logic 0. Reading is done by applying a voltage slightly smaller than the threshold voltage of the devices. Note that the bias voltages depend on the type of two-terminal nanodevices used as the memory cells [Likharev 2008].

2.3 Potentials

As aforementioned, hybrid memories provide enormous data storage at low power and reduced fabrication complexity. According to Strukov and Likharev [2004], CMOL hybrid memories can achieve a trillion-bit data storage capacity in a square centimeter memory chip. This is achievable by the use of two-terminal non-CMOS devices as the data storage elements. Moreover, as these devices are very tiny, they require a small number of electrons to represent a digital logic state, resulting in low power consumption. Because the memory cells are constructed by a regular crossbar-based non-CMOS circuit structure, the need to use arbitrary features and a high number of masks as in the existing technology is alleviated. The two-terminal devices can be fabricated using a self-assembled technique, as they pose only one critical dimension [Strukov 2006]; this eliminates the use of a complex lithography process.

2.4 Challenges

Reliability is one of the major challenges facing by the fabrication and production of hybrid memories [Mishra and Goldstein 2003; Strukov and Likharev 2004]. Inherent variation in design, process, and device properties when stimulated by nonenvironmental-related sources (e.g., temperature, noise) produces intermittent faults, which in turn become permanent faults after some time. Moreover, despite reducing the power consumption, non-CMOS and scaled-CMOS devices lose their resistance to environmental disturbances. Consequently, a small magnitude of energy originated from, e.g., cosmic particles, electrical noise, temperature variation, and so on, could disturb the logic state held by such nanodevices. This problem results in transient faults, which are also known as soft errors.

2.5 ECC Scheme for Hybrid Memories

To tolerate these intermittent and transient faults, and subsequently improve the overall reliability of hybrid memories, an ECC scheme is incorporated. Figure 2 exhibits the block diagram of the ECC circuit considered in this work.

During writing, the d-bit input data is encoded to RRNS codeword b_c, where $1 \leq c \leq n$, and n is an integer. The RRNS codeword is then stored in the memory cell array. During reading, the desired codeword is retrieved from the memory cells and is decoded before data can be read out. The decoding process will ensure that the read data

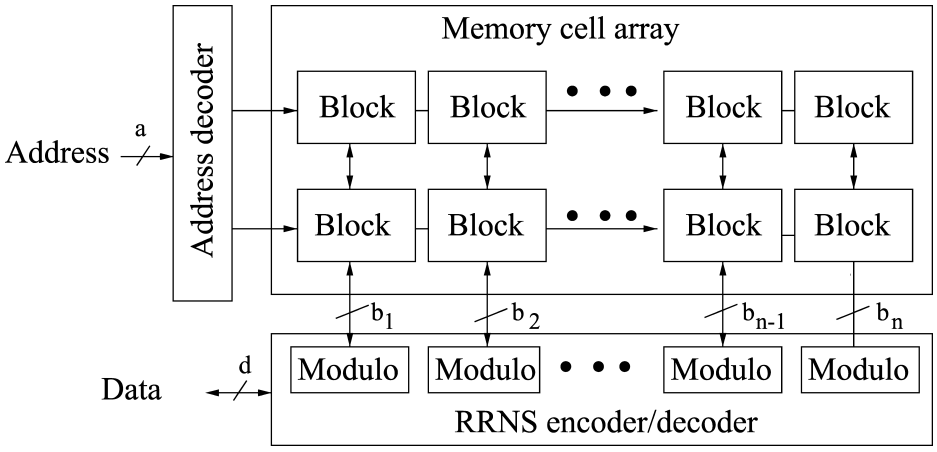

Fig. 2. ECC scheme in CMOL hybrid memory.

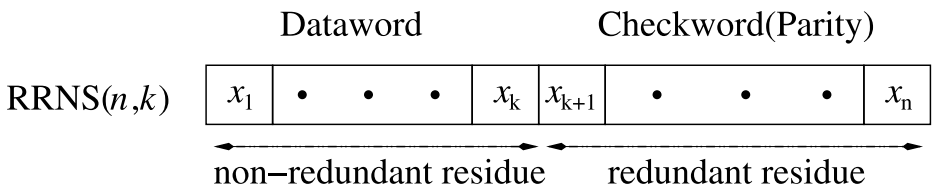

Fig. 3. Structure of RRNS code.

will be fault-free, provided that the faults are still within the correction capability of the RRNS decoder. In this work, the RRNS decoder is assumed to be ideal, where the faults only occur in the RRNS encoder and memory cell array. One solution to overcome this limitation is by using hardware redundancy for the decoder [Haron and Hamdioui 2011].

3. REDUNDANT RESIDUE NUMBER SYSTEMS CODE

In this section we explain the concept of RRNS code including its encoding and decoding procedures. To facilitate the explanation, we give examples for both procedures.

3.1 Theory of RRNS Code

An RRNS codeword is constructed from a set of encoded numbers called *residues*. A d-bit input data will be encoded into an n-symbol *codeword*, which is divided into two sets of residues (see Figure 3): (1) *nonredundant residues* x_i, consisting of a k-symbol *dataword*, and (2) *redundant residues* x_j, consisting of an $(n-k)$-symbol *checkword (parity)*; $1 \leq i \leq k$ and $k+1 \leq j \leq n$. The bit length of k-symbol maybe larger than that of d-bit input data ($k \geq d$) in order to provide the residue number system representation of the d bits. An RRNS codeword can correct up to $t = \frac{(n-k)}{2}$ symbols.

The residues x_i and x_j are generated by performing modulo operation on the input data X to a set of nonredundant moduli m_i and redundant moduli m_j, respectively; X represents the integer value of d-bit input data. These operations can be mathematically represented with the following equations [Szabo and Tanaka 1967].

$$x_i = |X|_{m_i}, \quad x_j = |X|_{m_j}. \tag{1}$$

The bit length of x_i is $b_i = \lfloor log_2(m_i - 1) + 1 \rfloor$ bits and that of x_j is $b_j = \lfloor log_2(m_j - 1) + 1 \rfloor$ bits. For example, for $m_i = 8$, the bit length of x_i is $\lfloor log_2(8 - 1) + 1 \rfloor = 3$ bits. This is true because for $m_i = 8$, $0 \leq x_i \leq 7$, each of which is represented by 3 bits. Summing b_i and b_j will result in the total bit length of the RRNS codeword b_c.

For an RRNS codeword to be consistently decoded (to prevent a codeword from being decoded into more than one output data), the nonredundant residues m_i must satisfy three requirements [Sun and Krishna 1992; Szabo and Tanaka 1967].

(1) A pair of any two moduli, say m_a and m_b with $a \neq b$, must be relatively prime positive integers such that their greatest common divisor $gcd(m_a, m_b) = 1$;
(2) the integer value of the succeeding modulus is greater than the preceding modulus, i.e., $m_1 < ... < m_k < m_{k+1} < ... < m_n$; and
(3) the product of nonredundant moduli $M_i = \prod_{i=1}^{k} m_i$, is sufficient to represent all numbers in the operating legitimate range of input data $M_{op} = [0, 2^d - 1]$ for d-bit input data.

In additional to the three requirements, the redundant residues m_j are chosen arbitrarily such that:

(1) they ensure the desired error correction capability; and
(2) their product $M_j = \prod_{j=k+1}^{n} m_j$, is sufficient to represent all the numbers in M_{op}.

Note that for a memory with d-input data words, both M_i and M_j have to be larger that the operating legitimate range $M_{op} = 2^d - 1$.

3.2 RRNS Encoding

RRNS encoding is based on modulo operation of the input data to a moduli set as explained in Section 3.1. The residues resulting from the modulo operations are obtained simultaneously; the operations are executed in parallel by the corresponding modulo circuits. They are then concatenated to create an RRNS codeword before being stored in memory cells. Alternatively, the redundant residues can be generated from the nonredundant residues using a scheme known as *Base Extension (BEX)* [Szabo and Tanaka 1967]. However, this scheme requires an iterative calculation, which makes the encoding slower.

Example 1. An 8-bit input data of a memory system will be encoded into an RRNS codeword based on the moduli set, e.g., $\{m_1, m_2, m_3, m_4, m_5\} = \{5, 7, 8, 9, 11\}$. The first three moduli are nonredundant moduli m_i, and the last two moduli are redundant moduli m_j. This RRNS code has the correction capability of $t = \frac{5-3}{2} = 1$. Note that the operating legitimate range M_{op} for this RRNS codeword is $[0, 2^8 - 1 = 255]$. Therefore, the RRNS codeword for input data $X = 234$ is:

$$x_c = \{|234|_5, |234|_7, |234|_8, |234|_9, |234|_{11}\}$$
$$x_c = \{4, 3, 2, 0, 3\}.$$

The total bit length of x_c is 17 bits; this is because the first three moduli require three bits each, while the fourth and the fifth modulus require 4 bits each.

3.3 RRNS Decoding

RRNS decoding starts by validating the read codeword from the memory array. The codeword is regarded as valid if its decoded value is within the operating legitimate range M_{op}; so, no error correction is needed. On the other hand, it is regarded as invalid if its value is equal to or larger than M_{op}; hence, a correction phase is needed.

During the correction phase, an iterative calculation is executed. In each iteration, t number of residues will be discarded. Subsequently, the decoding procedure will calculate the decoded value using the remaining $(n - t)$ residues. This exhaustive calculation will be performed until the valid data is recovered for maximum $C_t^n = \frac{n!}{t!(n-t)!}$ times of iterations, where n is the number of residues and t is the error correction capability (see Section 3.1). Ideally, whenever the erroneous residues are discarded, the correct data will be recovered and read out. However, when no data less than M_{op} is recovered after the maximum iteration, the decoding cannot correct the erroneous residues in the RRNS codeword. Consequently, an uncorrectable output signal can be invoked so that the output data can be ignored or can be further processed for a system equipped with multilevel fault tolerance.

3.4 Decoding Algorithms

Two algorithms can be used in the decoding process, either (1) *Chinese Remainder Theorem (CRT)* or (2) *Mixed-Radix Conversion (MRC)* [Szabo and Tanaka 1967].

CRT is the basic theory to convert the residue number system into binary or decimal. It is based on the following equation [Szabo and Tanaka 1967].

$$X = \left| \sum_{c=1}^{n} x_c \times M_c \times |M_c^{-1}|_{m_c} \right|_M, \tag{2}$$

where x_c are the residues, $M = \prod_{c=1}^{n} m_c$ is theoretical RNS legitimate range, $M_c = \frac{M}{m_c}$ are modular multiplicatives, and $|M_c^{-1}|_{m_c}$ are the modular multiplicative inverse of $|M_c|_{m_c}$, satisfying $|M_c \times M_c^{-1}|_{m_c} = 1$. MRC is based on the following equations [Szabo and Tanaka 1967].

$$X = x_1 + (v_2 \times m_1) + (v_3 \times m_1 \times m_2) + \ldots + \left(v_n \times \prod_{c=1}^{n-1} m_c \right), \tag{3}$$

where x_1 is the first residue, m_c are the moduli, and v_c are mixed radix digits calculated as:

$$v_c = \left| \left(\left((x_c - v_1) \times g_{1c} \right) ..... - v_{(c-1)} \right) \times g_{(c-1)c} \right|_{m_c}, \tag{4}$$

where $x_c = |X|_{m_c}$ and $g_{(c-u)c}$ are the multiplicative inverses of $m_{(c-u)}$ with respect to m_c defined as $|m_{(c-u)} \times g_{(c-u)}|_{m_c} = 1$; $2 \le c \le n$ and $1 \le u \le n - 1$.

Example 2. Assume that the fourth residue of the codeword in Example 1 is corrupted during the storing and that resulted in $x_c = \{4, 3, 2, \mathbf{8}, 3\}$. First, we show the decoding process using CRT and thereafter MRC.

Before that, parameters such as legitimate ranges, multiplicatives, multiplicative inverses, and mixed radix digits are required to accomplish the decoding. These parameters can be precalculated; for instance, $M = \prod_{c=1}^{n} m_c = 27720$ and $M_o p = 2^d - 1 = 255$. The other parameters as given in Table I.

Table I. The Appropriate Parameters for CRT and MRC

	Parameter for CRT		Parameter for MRC
Moduli, m_c	Multiplicatives, M_c	Multiplicative Inverses, M_c^{-1}	Mixed Radix Digits, v_c
5	5544	4	4
7	3960	3	4
8	3465	1	6
9	3080	5	8
11	2520	1	4

Table II. Calculated Data During Correcting Phase

Iteration, g	1	2	3	4	5
Discarded Residue, x_g	x_1	x_2	x_3	x_4	x_5
Recovered Data, X'_g	1466	3594	2159	234	2474

Substituting the appropriate parameters from Table I into Equation (2) will result in decoded data based on CRT:

$$X = \left| \sum_{c=1}^{n} x_c \times M_c \times |M_c^{-1}|_{m_c} \right|_M$$

$$X = \Big| (4 \times 5544 \times 4) + (3 \times 3960 \times 3) + (2 \times 3465 \times 1) + (8 \times 3080 \times 5) + (3 \times 2520 \times 1) \Big|_{27720}$$

$X = 12554 > 255$: error is detected since this is larger than the legitimate operating range.

The same result will be obtained when substituting the appropriate parameters from Table I into Equation (3). The decoded data based on MRC is:

$$X = x_1 + (v_2 \times m_1) + (v_3 \times m_1 \times m_2) + (v_4 \times m_1 \times m_2 \times m_3) + (v_5 \times m_1 \times m_2 \times m_3 \times m_4)$$

$$X = 4 + (4 \times 5) + (6 \times 5 \times 7) + (8 \times 5 \times 7 \times 8) + (4 \times 5 \times 7 \times 8 \times 9)$$

$X = 12554 > 255$: error is detected since this is larger than the legitimate operating range.

The error correction procedure is invoked because the codeword is invalid. A systematic iterative calculation that discards a residue in each iteration is performed for a maximum $C_1^5 = 5$ iterations (see Section 3.3). This exhaustive search produces the integer values as shown in Table II. It recovers the correct value when x_4 is discarded, i.e, results in $X = 234$, which is within the legitimate operating range.

4. RRNS CODES FOR HYBRID MEMORIES

In this section we first give an overview of Reed-Solomon and Conventional RRNS code, which will be experimentally compared in this work. Thereafter, we introduce the adapted RRNS variant suitable for improving the reliability of hybrid memories at lower cost.

4.1 Conventional RRNS Code

In this work, the conventional RRNS (C-RRNS) code consists of nine residues, where there are (1) three restricted nonredundant moduli, and (2) six unrestricted redundant moduli. The restricted nonredundant moduli set is based on $\{2^p - 1, 2^p, 2^p + 1\}$, where p is a positive integer; this moduli set is adopted to realize simple and fast

Table III. Moduli and p for C-RRNS and 6M-RRNS Codes

ECC Types	Moduli Set	p value		
		16-bit	32-bit	64-bit
C-RRNS	$2^p-1, 2^p, 2^p+1, q_1, q_2, q_3, q_4, q_5, q_6$	6	11	22
6M-RRNS	$2^p+1, 2^p, 2^{p-1}-1, 2^{p-2}-1, 2^{p-3}-1, 2^{p-4}+1$	8	16	32

Table IV. Bit Length of C-RRNS, 6M-RRNS and RS ($GF(2^8)$) Codes

ECC Types	Bit length of Codeword		
	16-bit	32-bit	64-bit
C-RRNS	61	106	205
RS	48	96	192
6M-RRNS	40	88	184

hardware implementation [Barsi and Maestrini 1973; Wang et al. 2003]. Unrestricted redundant moduli are commonly appended to the nonredundant moduli in order to provide the fault detection and the correction capability. The restricted nonredundant moduli are selected in such a way that, besides satisfying the RRNS rules, the product of the selected moduli should be as close as possible to the operating legitimate range to optimize the area overhead and performance penalty. This selection is accomplished by choosing the minimum value of p. On the other hand, the number of the unrestricted redundant moduli q_υ, where $1 \leq \upsilon \leq 2t$, can be any integer larger than the nonredundant moduli as long they conform the RRNS rules. In this work, prime integers larger than those of the restricted nonredundant moduli are chosen to be the redundant moduli.

For a 16-bit memory word, the integer value for restricted nonredundant residues $\{2^p-1, 2^p, 2^p+1\}$ must be selected such that their product is at least equal to the operating legitimate range $M_{op} = 2^{16}-1$. To satisfy this criterion, the minimum value of $p = 6$ is chosen (see Table III). This results in nonredundant moduli $m_i = \{63, 64, 65\}$; note that the theoretical RRNS legitimate range $M_i = 63 \times 64 \times 65 = 262080 > M_{op}$. The chosen redundant moduli are $m_j = \{67, 71, 73, 79, 83, 89\}$, where their product is obviously more than M_{op}. The six-residue checkword is needed to protect the three-residue dataword, $t = 3$. The codeword length is $b_c = \sum_{i=1}^{k} \lfloor log_2(m_i - 1) + 1 \rfloor + \sum_{j=k+1}^{n} \lfloor log_2(m_j - 1) + 1 \rfloor = 61$ bits. The first row of Table IV shows the required bit length for different data lengths encoded into C-RRNS.

4.2 Reed-Solomon Code

Reed-Solomon (RS) code encodes d-bit input data into an n-symbol codeword, which consists of a k-symbol dataword and an $(n-k)$-symbol checkword [Lin and Costello 2004]. The symbols in RS code are encoded by b bits each and are represented by finite field elements (also known as Galois Field). All symbols have the same bit length depending on the Galois Field elements. RS code can correct up to t faulty symbols in a codeword by appending a checkword of $2t = (n-k)$ symbols. This code has been employed in high performance SRAM, flash memories, and compact disks [Neuberger et al. 2003].

In this work we use Galois Field of degree eight ($m = 8$); it means that each RS symbol is represented by eight bits. A d-bit input data is encoded into a $\frac{d}{m}$-symbol dataword. In order to ensure that the $\frac{d}{m}$-dataword is correctable if corrupted, a $2 \times \frac{d}{m}$-symbol checkword is appended, resulting in a codeword of $3 \times \frac{d}{m}$-symbols. Thus, the bit length of the RS codeword is three times longer than that of the input data. For instance, for a 16-bit input data with $m = 8$, a codeword of $\frac{16}{8} = 2$-symbol dataword and

$2 \times \frac{16}{8}$ = 4-symbol checkword is needed to correct the 2-symbol dataword. The required bit length for different data lengths encoded into RS code is shown in Table IV.

4.3 Modified RRNS Code

As shown in Table IV, the bit length of C-RRNS is longer than that of RS for all considered data lengths. Yet, C-RRNS code has better correction capability than RS (more detail in Section 5). Therefore, it is beneficial to modify C-RRNS to reduce the length of a C-RRNS codeword, while retaining its correction capability at a competitive level. Modified RRNS is introduced to achieve this; it is based on the following strategies [Haron and Hamdioui 2009].

(1) *For non-redundant moduli*. We minimize the number of nonredundant moduli as long as their product is larger than the legitimate operating range. This results in shorter checkwords, hence shorter codewords.
(2) *For redundant moduli*. We choose redundant moduli with smaller integer values than that of the nonredundant moduli; yet the chosen moduli have product larger than the operating legitimate range. Note that this strategy violates the second rule of RRNS coding as mentioned in Section 3.1; the succeeding moduli (redundant moduli) become smaller than the preceding (nonredundant moduli). This violation may cause an inconsistency during decoding, where a single read codeword might be decoded into more than one (ambiguous) output data. Maximum likelihood decoding is introduced to resolve this problem.
(3) *Maximum likelihood decoding (MLD)*. We use this method to distinguish the authentic output data from the ambiguous data. More details on this method will be given in Section 4.4.

The modified version of C-RRNS is referred to as *Six-Moduli RRNS (6M-RRNS)*. This code is comprised of six residues, where (1) two are nonredundant moduli and (2) four are redundant moduli. Based on our modification strategies mentioned in the preceding, the nonredundant moduli set is $\{2^p, 2^p + 1\}$, while the redundant moduli set is $\{2^{p-1} - 1, 2^{p-2} - 1, 2^{p-3} - 1, 2^{p-4} + 1\}$, where p is an integer as shown in Table III. Note that for 6M-RRNS both moduli sets are restricted, while for C-RRNS this is only the case for the nonredundant moduli. Therefore, this restricted moduli set will result in a simpler hardware implementation than that of C-RRNS.

For a 16-bit memory word, the smallest p that satisfies the requirement that the product of nonredundant moduli is larger than $M_{op} = 2^{16} - 1$ is $p = 8$ (see Table III). This results in $m_i = \{257, 256\}$, $M_i = 65792$, $m_j = \{127, 63, 31, 17\}$, and $M_j = 4216527$. Although m_j consists of smaller integer values than that of m_i, their product M_j is clearly more than $M_{op} = 2^{16} - 1$. In a similar way, $p = 16$ for a 32-bit and $p = 32$ for a 64-bit memory word are chosen.

4.4 Maximum Likelihood Decoding

Because we intentionally use the second strategy mentioned in Section 4.3, some of the read codeword might be decoded into more than one (ambiguous) decoded data. According to our simulation, less than 10% of the decoded data will suffer from this ambiguity. To resolve this ambiguity, a maximum likelihood decoding (MLD) scheme, adopted from Goh and Siddiqi [2008], is employed. The concept behind the scheme is to find the closest Hamming distance between the residues of the ambiguous decoded data and the read codeword that causes the ambiguity. First, each ambiguous decoded data is encoded, resulting in a new codeword. Then, the Hamming distance between each new codeword and the read codeword (the codeword that produces the ambiguous

data) is calculated [Goh and Siddiqi 2008]. Finally, the ambiguous data for the codeword that has the smallest difference (Hamming distance) is regarded as the authentic output data. As the moduli set used for 6M-RRNS satisfies the first and third RRNS requirement, MLD ensures the decoding correctness. The following example clarifies the MLD scheme.

Example 3. Assume a 16-bit memory word with 6M-RRNS coding; the nonredundant moduli are $m_i = \{257, 256\}$ and the redundant moduli are $m_i = \{127, 63, 31, 17\}$. The first residue of the codeword $x = \{221, 0, 72, 18, 9, 2\}$ (represents $X = 9216$) is corrupted during storing and is read as $x' = \{\mathbf{0}, 0, 72, 18, 9, 2\}$. During decoding, the erroneous residue is detected and then is corrected. After the maximum iteration of the correction has been reached, there are two decoded data with value less than $M_{op} = 2^{16} - 1$; $X'_1 = 9216$ when m_1 and m_2 are discarded, and $X'_2 = 257$ when m_3 and m_6 are discarded. This ambiguity requires MLD to determine the authentic output data as follows.

First, X'_1 and X'_2 are each modulo to the moduli set producing x'_1 and x'_2.

$$x'_1 = \{|9216|_{257}, |9216|_{256}, |9216|_{127}, |9216|_{63}, |9216|_{31}, |9216|_{17}\}$$
$$x'_1 = \{\mathbf{221}, 0, 72, 18, 9, 2\}$$
$$x'_2 = \{|257|_{257}, |257|_{256}, |257|_{127}, |257|_{63}, |257|_{31}, |257|_{17}\}$$
$$x'_2 = \{0, \mathbf{1}, \mathbf{3}, \mathbf{5}, 9, 2\}$$

Then, the resulting residues x'_1 and x'_2 are compared with the the read codeword x'. This comparison reveals that x'_1 has Hamming distance $d_{min} = 1$ (differs in the first residue) while x'_2 has $d_{min} = 3$ (differs in second, third, and fourth residues). Therefore, the authentic output data is $X'_1 = 9216$.

5. EXPERIMENTAL EVALUATION AND ANALYSIS

In this section we evaluate and analyze the performance of the proposed 6M-RRNS code by comparing it to RS and C-RRNS. We start with the simulation setup. Then the simulation results will be presented. Thereafter, we present an analysis of five different aspects including, (1) the codeword length required by the considered codes, (2) the data storage capacity for a fixed memory size, (3) the decoding latency of RRNS variants, (4) the ratio of error correction capability for the required codeword length, and (5) the correction capability when considering all codes have the same codeword length.

5.1 Simulation Setup

The RRNS variants, RS codes, memories, and fault injection were described using MATLAB script. All codes were set to the corresponding t to protect the codeword from faults. For the RRNS decoding process, MRC was implemented because it requires simple design, and is easier to optimize than CRT. The operating legitimate range M_{op} is set up to $M_{op} = 2^d - 1$, where $d = 16, 32, 64$. For RS code, MATLAB built-in RS encoding and decoding functions were used [MathWorks]. An appropriate adjustment for the polynomial generator was done to encode and decode 16, 32, and 64-bit RS memory words.

Cluster faults were uniformly injected at each memory word (the stored codeword). The faults were increased from a single bit up to 20 clustered bits per codeword for 16-bit word memory, single bit up to 35 clustered bits per codeword for 32-bit memory words, and single bit up to 68 bits per codeword for 64-bit memory words. Various fault rates from 1% to 10% were applied during the experiments.

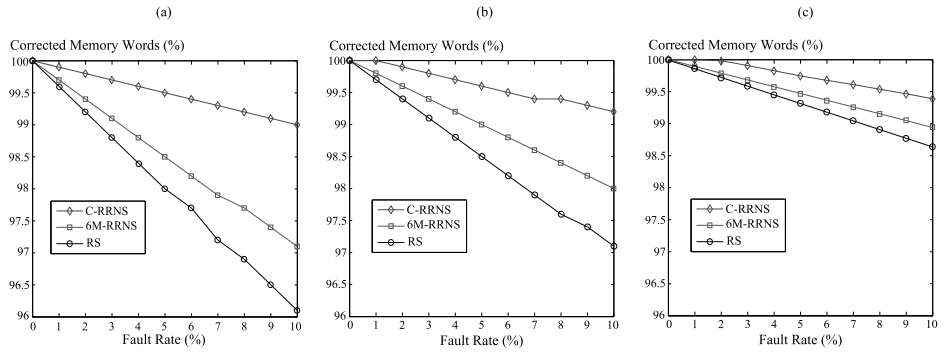

Fig. 4. Simulation results for different lengths of memory words.

5.2 Simulation Results

Figure 4 shows the simulation results for C-RRNS, 6M-RRNS, and RS codes. Overall, C-RRNS provides the best error correction capability, followed by 6M-RRNS, whereas RS scores the worst. For example, for 16-bit memory at 10% fault rate, C-RRNS code can correct at least 2% more than the other two codes. However, as the memory word increases, the difference among all investigated ECCs becomes marginal. For example, for 64-bit memory at 10% fault rate, the difference between C-RRNS and 6M-RRNS is only 0.40%.

It is clear from the preceding that the modified version of 6M-RRNS can realize a competitive error correction capability, especially for large memory word size, which is expected to be the case for hybrid memories. Moreover, the advantage of 6M-RRNS as compared to RS and C-RRNS is lower overhead and time performance, as will be explained in next subsections.

5.3 Analysis

First, consider the generated codeword length (in terms of number of bits) of the considered codes. Figure 5 shows that 6M-RRNS has the shortest codeword length as compared to RS and C-RRNS codes. The bit lengths of the ECCs are also given in Table IV. For example, for a 64-bit memory word 6M-RRNS realizes a codeword that is 4.17% and 10.24% shorter than that of the RS and C-RRNS codes, respectively. This implies larger user data storage capacity and faster decoding.

Second, consider the capacity of the user data storage for a fixed memory chip size. We assume that a memory size of 1Tbit is used to store user data encoded in all of the considered codes. For this, the capacity (in terms of memory words) is calculated by dividing $1T = 2^{40}$ bits by codeword length B_c of the considered ECCs, that is, $C = \frac{2^{40}}{B_c}$. Then the ratio between 6M-RRNS and C-RRNS as well as between 6M-RRNS and RS is calculated. Considering 16-bit memory words, RS is able to store 1T/48 data words, C-RRNS 1T/61 data words, and 6M-RRNS 1T/40 data words. Hence, as presented in Table V, 6M-RRNS is able to store 20.00% and 52.50% more data than RS and C-RRNS, respectively. The difference in user data capacity for 1T organized into 32-bit and 64-bit memory words is shown in the third and fourth columns in Table V, respectively. Although the differences decrease as the input data width increases, they are still significant. Overall, 6M-RRNS offers the biggest data storage when considering a fixed memory chip size.

Third, consider the decoding time for RRNS codes. Since 6M-RRNS code is able to correct at most two erroneous residues, the code requires a maximum of $C_2^6 = \frac{6!}{2!(6-2)!} =$

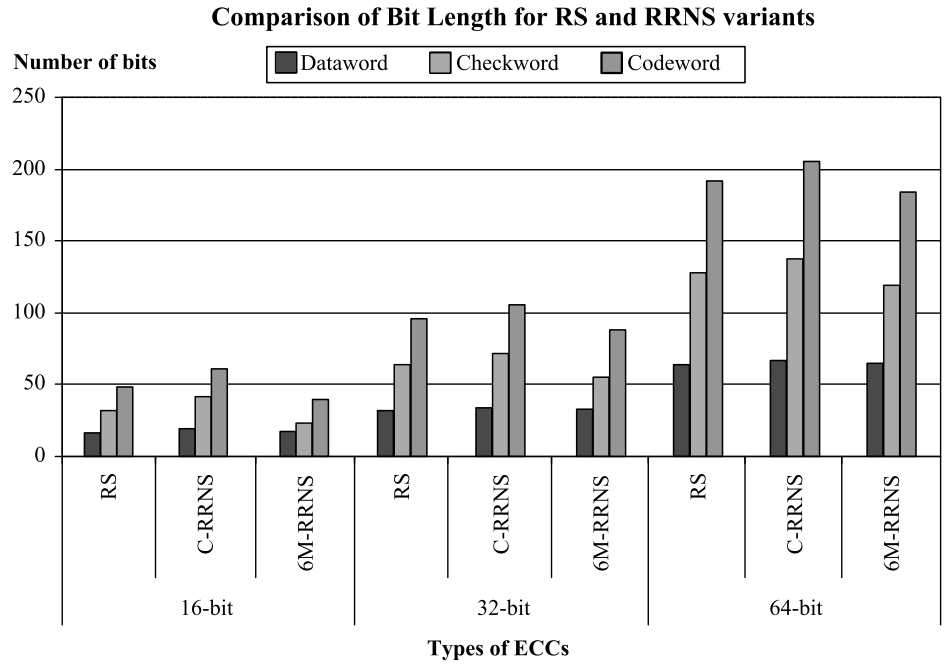

Fig. 5. Codeword length for each code.

Table V. Difference (%) in User Data Capacity Among the ECCs for 1TBit Memory

6M-RRNS	**Difference (%) in capacity**		
vs.	16-bit	32-bit	64-bit
RS	20.00%	9.09%	4.35%
C-RRNS	52.50%	20.45%	11.41%

15 iterations during the decoding procedure. Moreover, the likelihood decoding step is required to determine the authentic output of the ambiguous data (see Section 4.4). On the other hand, C-RRNS code needs a maximum of $C_3^9 = \frac{9!}{3!(9-3)!} = 84$ iterations. This means that 6M-RRNS variants decode up to 5.25 times faster than C-RRNS. Note that RS has different decoding steps than RRNS variants [Lin and Costello 2004]; thus, it is not considered in this case.

Fourth, consider the ratio of error correction capability for the required codeword length. Although 6M-RRNS requires shorter codewords, Figure 6 shows that this code realizes better correctable bits per codeword length (in bits). The number of correctable bits is $B_t = \sum_{s=1}^{t} B_s$, where $t = \frac{n-k}{2}$ is the error correction capability in terms of symbols and B_s is the number of bits per symbol s. The ratio is calculated by dividing B_t by codeword length B_c, i.e., $t_B = \frac{B_t}{B_c}$. For 16-bit input data encoded into 6M-RRNS, the maximum number of erroneous bits it can correct is 17 out of 40 bits per codeword (42.50%). RS and C-RRNS can correct only 16/48 = 33.33% and 19/61 = 31.10%, respectively. Note that the differences decrease as the input data width becomes larger. Overall, 6M-RRNS provides the highest bit-wise error correction capability in this case.

Fifth, consider the correction capability when assuming that all codes have the same codeword length. Let's take C-RRNS with codeword length of 61 bits for 16-bit data as the reference. Given the budget of 61 bits, RS (48 bits) can have one additional symbol, whereas 6M-RRNS (40 bits) can have two additional residues. This is because

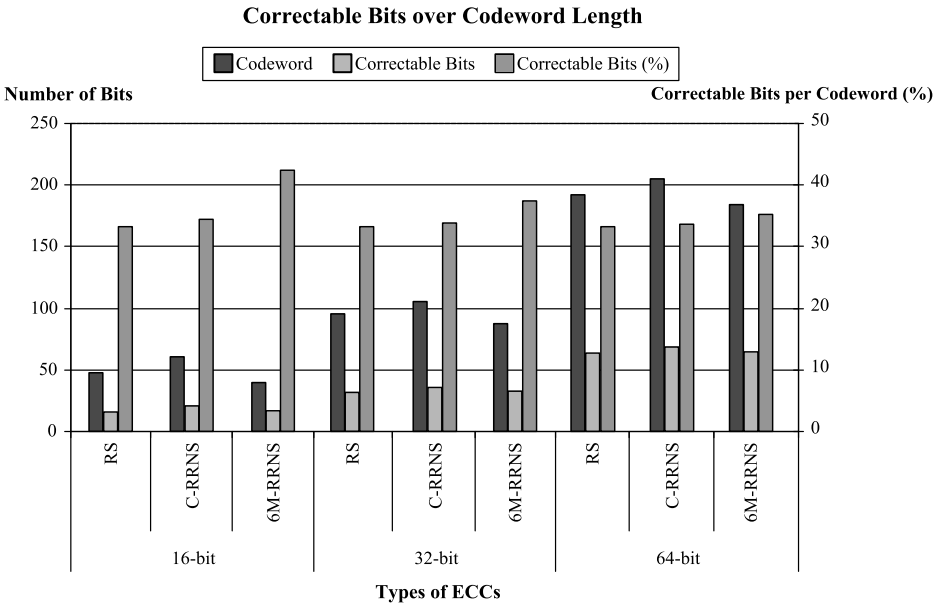

Fig. 6. Error correction capability over codeword length.

RS requires 8 bits for each symbol, while RRNS can use any prime integer moduli with bit length of 8 bits each. Therefore, RS will consist of a codeword of five symbols, resulting in a correction capability of $t = \lfloor \frac{7-2}{2} \rfloor = 2$ symbols. For 6M-RRNS, the code will consist of a checkword of six residues that can correct $t = \lfloor \frac{8-2}{2} \rfloor = 3$ symbols. This capability also applies for larger data lengths. Therefore, 6M-RRNS offers the best error correction in this case.

6. ENCODER AND DECODER IMPLEMENTATION

Having shown that 6M-RRNS possesses advantages over C-RRNS and RS codes, the question is now what is the cost of implementing the encoder and decoder in terms of the area and performance penalties. We designed the circuits using Very High Speed Integrated Circuit Hardware Design Language (VHDL) and synthesized the circuits in 90nm CMOS technology. Moreover, we use the scaling factor from [ITRS] to estimate the area and time overhead for 32nm CMOS technology. Xilinx ISE and Synopsys Design Compiler tools are used during the design phase.

6.1 Encoder Implementation

Figure 7(a) shows the block diagram of an RRNS encoder, which consists of d-bit data at the input side and varied-length n-residue at the output side. This encoder is formed by n numbers of modulo circuits that run in parallel. Figure 7(b), (c), and (d) illustrate the functional units of the modulo circuitry embedded inside the encoder. Each figure represents the circuit that generates the residues based on moduli set in the form of 2^p, $2^p - 1$, and $2^p + 1$.

The circuit that generates the residue based on 2^p is comprised of a buffer as shown in Figure 7(b). The p-bit outputs of buffer *BUFF* are connected to the corresponding

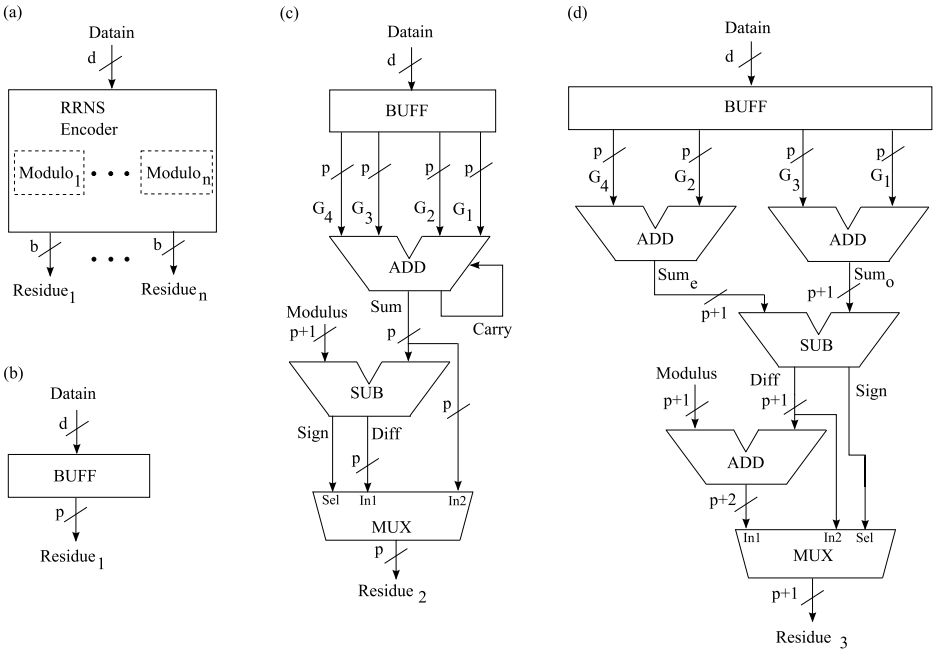

Fig. 7. (a) Block diagram of RRNS encoder (b) functional units of modulo 2^p circuit (b) functional units of modulo $2^p - 1$ circuit (c) functional units of modulo $2^p + 1$ circuit; for $p = 4$.

p least significant bits of the input data. The remaining $d - p$ most significant inputs are left unconnected. Any input data *Datain* asserted into the buffer will produce its corresponding *Residue1*.

The circuit that generates the residue based on $2^p - 1$ is comprised of a buffer, an adder, a subtracter, and a multiplexer, as depicted in Figure 7(c). Buffer *BUFF* splits the input data *Datain* into an appropriate number of intermediate data groups *Gg*, where g is a positive integer. These are added together producing *Sum*. This temporary addition might produce carry bits, which must be added again to the sum. Subsequently, *SUB* and *MUX* circuits will select whether to take *Sum* as the output *Residue2*, or to take the difference $Diff = Sum - Modulus$ as the output *Residue2*. The former will be read out if *Sum* is less than the *Modulus*; otherwise, the latter will be selected.

The circuit that generates the residue based on $2^p + 1$ consists of a buffer, two adders, a subtracter, and a multiplexer, as illustrated in Figure 7(d). Buffer *BUFF* splits the input data *Datain* into an appropriate number of intermediate data groups *Gg*, where g is a positive integer. Afterwards, each of the two adders performs the addition among the intermediate data in odd and even groups, respectively. Thereafter, a subtracter performs $Diff = Sum_o - Sum_e$. Finally, the *ADD* and *MUX* circuits execute their operations to produce the final result: *Residue3*. If *Diff* is positive, the number is taken as *Residue3*, otherwise it takes $Sum = Modulus + Diff$. If the latter is selected, only the $p + 1$ least significant bits are taken as the residue and the $p + 2$-bit is ignored.

Note that the intermediate data for producing the residues based on $2^p - 1$ and $2^p + 1$ from input data is a fundamental element in a modulo operation. Without this data, an incorrect modulo might be produced. A more complete explanation can be found in Szabo and Tanaka [1967].

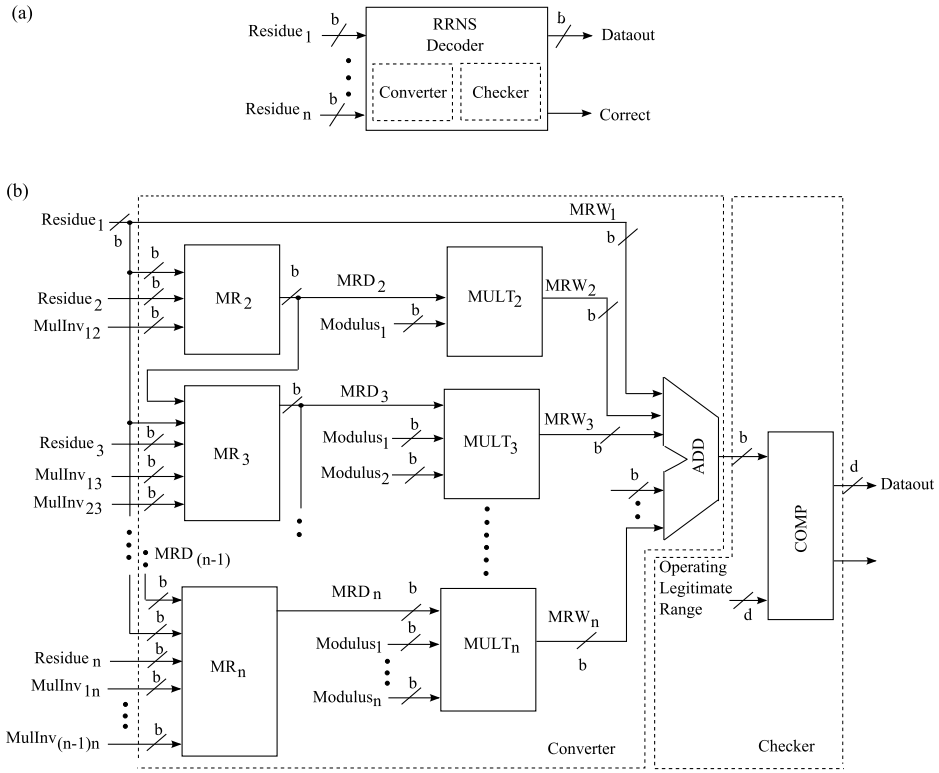

Fig. 8. (a) Block diagram of RRNS decoder (b) functional unit of the detection circuit.

6.2 Decoder Implementation

Figure 8(a) illustrates the block diagram units of an RRNS decoder with n-residue at the input side and d-bit output data as well as a control signal, *Correct*, at the output side. The decoder is designed to perform two operations: (1) to convert the RRNS codeword (represented by a set of residues) into binary output data, and (2) to check its validity (to determine any occurrence of error). The converter is based on a mixed-radix conversion scheme as explained in Section 3.4. The checker compares the binary data converted from an RRNS codeword with a predefined operating legitimate value. This circuit will invoke a software-based correction procedure if faults have been detected.

Figure 8(b) depicts the functional units of the converter and checker. The converter consists of a combination of circuits that produce mixed radix digits MR_c (where c is a positive integer), multipliers, and an adder, whereas the checking circuit is a comparator. Generally, the MR_c circuit consists of the modulo circuit, similar to that of the encoder, but with more inputs and components. Each MR_c circuit has inputs including residues $Residue_n$, mixed radix digits from the previous state $MRD_{(c-1)}$, and multiplicative inverses $MulInv_{(c-u)c}$, where $2 \leq c \leq n$ and $2 \leq u \leq n-1$. For example, MR_2 has inputs $Residue_1$, $Residue_2$, and $MulInv_{12}$, whereas MR_3 has inputs $Residue_1$, MRD_2, $Residue_3$, $MulInv_{13}$, and $MulInv_{23}$. Note that the MR_c at the higher level has more inputs compared to the MR_c at the lower level. The output of MR_c, together with the predefined moduli $Modulus_c$, becomes an input to multiplier $MULT_c$. Similar to MR_c, each multiplier also has inputs that vary depending on the level of the circuit. For

Table VI. Synthesis Results for 90nm CMOS Technology

Data width	Circuits	Area Overhead (μm^2)	Time Overhead(ns)
16-bit	Encoder	1988.38	10.67
	Decoder	3999.34	42.81
	Overall	5987.72	53.48
32-bit	Encoder	4338.03	15.81
	Decoder	11216.22	67.62
	Overall	15554.25	83.43
64-bit	Encoder	10696.90	33.71
	Decoder	31876.19	118.43
	Overall	42573.09	152.14

Table VII. Estimated Results for 32nm CMOS Technology

Data width	Circuits	Area Overhead (μm^2)	Time Overhead(ns)
16-bit	Encoder	248.54	2.13
	Decoder	499.92	8.56
	Overall	748.47	10.70
32-bit	Encoder	542.25	3.16
	Decoder	1402.03	13.52
	Overall	1944.28	16.68
64-bit	Encoder	1337.11	6.74
	Decoder	3984.52	23.68
	Overall	5321.63	29.97

example, $MULT_1$ has only two inputs: MRD_2 and $Modulus_1$; whereas $MULT_2$ has three inputs: MRD_3, $Modulus_1$ and $Modulus_2$. The outputs of the multipliers MRW_c are added, producing the binary output data. This binary data (the sum) is compared to the legitimate operating range to ensure its validity. The binary data is valid if it is less than the operating legitimate range and can be read out of the memory; otherwise it is invalid and is ignored. The control signal *Correct*, indicates the validity of the binary data where it is reset if the binary data is valid and set when invalid. When the *Correct* signal sets, a software-based correction procedure will be activated. Note that the MRW_1 is equal to $Residue_1$; thus, no additional circuit is needed except wires to connect $Residue_1$ to the adder.

6.3 Implementation Results

Table VI shows the synthesis results for the area occupied by the proposed 6M-RRNS encoder and decoder (measured in square micrometers), and their critical combinational path delay (measured in nanoseconds) synthesized in 90nm CMOS technology. Moreover, we estimate our design for 32nm technology. According to the International Technology Roadmap for Semiconductors, the scaling factors for silicon area and combinational datapath delay between 90nm and 32nm CMOS technology are approximately 8 and 5, respectively [ITRS].

Table VII summarizes the estimated area overhead and time penalty of the 6M-RRNS encoder and decoder for 32nm technology. For example, the area overhead of the encoder and decoder occupies much less than 1% of the total 1cm^2 CMOL memory chip [Strukov and Likharev 2007]. For example, for 64-bit memory with estimated 32nm technology, the encoder and decoder area is 5321.63μm^2, which is 0.005% of the 1cm^2 area of a CMOL memory chip. In general, for $h \times 64$-bit the encoder and decoder adds $h \times 5.32 \times 10^{-5}$ overhead to 1cm^2 hybrid memories, where h is an integer. For example, for 128-bit the area overhead for the encoder and decoder can be estimated roughly as $\frac{128}{64} \times 5.32 \times 10^{-5} = 1.06 \times 10^{-4}$cm^2. However, high time penalty is the price that has to be paid because of the implementation of the 6M-RRNS encoder and decoder. To

overcome this limitation, design optimizations, for example by using multiple access ports or by parallelizing the MRC-based RRNS decoding, can be implemented.

Theoretically, these area overhead and time penalties are still better than C-RRNS because both ECCs are based on a similar encoding and decoding algorithm. As mentioned in Section 4.1, C-RRNS uses unrestricted moduli to generate the checkword. This has been proven to incur larger cost as compared to that of restricted moduli [Barsi and Maestrini 1973; Wang et al. 2003], which is used by 6M-RRNS for both parts. However, further experimentation is needed to evaluate the difference in area between these two RRNS variants.

7. CONCLUSION

In this article we presented the concept of Redundant Residue Number Systems (RRNS) code to improve the reliability of hybrid memories. After giving the fundamental concept of RRNS code, we introduced a modified RRNS code to tolerate a high degree of cluster faults. The modified version, referred to as 6M-RRNS, consists of six residues, whereas conventional RRNS (C-RRNS) code comprises of nine residues. Our experimental results and quantitative analysis show that 6M-RRNS code provides competitive error correction capability as compared to C-RRNS and Reed-Solomon (RS) codes. Furthermore, it offers shorter codeword bit length, allowing for more data storage as compared to the other two codes. Moreover, 6M-RRNS decodes faster than C-RRNS due to the smaller number of residues. It is worth mentioning that the decoder is assumed to be fault-free where the faults only impact the encoder and memory cell array.

Future work will investigate new fault tolerance techniques to tolerate intermittent and transient faults that impact not only the memory cell array but also the other components of hybrid memories, i.e., CMOS-to-nano interface pins and peripheral circuits. To achieve these aims, techniques like interleaving and hardware redundancy, together with error correction code, will be explored.

REFERENCES

BARSI, F. AND MAESTRINI, P. 1973. Error correcting properties of redundant residue number systems. *IEEE Trans. Comput. 22,* 3, 307–315.

BISWAS, S., METODI, T. S., CHONG, F. T., AND KASTNER, R. 2007. A pageable, defect-tolerant nanoscale memory system. In *Proceedings of IEEE International Symposium on Nanoscale Architecture*. 85–92.

BULLIS, K. Ultradense molecular memory: Researchers develop a large-scale array of nanoscale memory circuits. http://www.technologyreview.com/nanotech/18100/.

CALMEC. Molecular electronic technology. http://www.calmec.com/.

DEHON, A., GOLDSTEIN, S. C., KUEKES, P., AND LINCOLN, P. 2005. Nonphotolithographic nanoscale memory density prospects. *IEEE Trans. Nanotechnol. 4,* 2, 215–228.

GHOSH, S. AND LINCOLN, P. D. 2008. Dynamic low-density parity check codes for fault-tolerant nanoscale memory. http://www.csl.sri.com/users/shalini/ldpc.pdf.

GOH, V. T. AND SIDDIQI, M. U. 2008. Multiple error detection and correction based on redundant residue number systems. *IEEE Trans. Comm. 56,* 3, 325–330.

HARON, N. Z. AND HAMDIOUI, S. 2009. Residue-based code for reliable hybrid memories. In *Proceedings of IEEE International Symposium on Nanoscale Architectures*. 27–32.

HARON, N. Z. AND HAMDIOUI, S. 2011. Cost-efficient fault-tolerant decoder for hybrid nanoelectronic memories. In *Proceedings of Design, Automation and Test in Europe Conference (DATE)*. To appear.

ITRS. The International Technology Roadmap for Semiconductors. http://www.itrs.net/links/2009itrs/home2009.htm.

JEFFERY, C. AND FIGUEIREDO, R. J. O. 2006. Hierarchical fault tolerance for nanoscale memories. *IEEE Trans. Nanotechnol. 5,* 4, 407–414.

KISH, L. B. AND AJAYAN, P. M. 2005. Terrabyte flash memory with carbon nanotubes. *Appl. Phys. Lett. 86,* 9, 1–2.

KÜGELER, C., MEIER, M., ROSEZIN, R., GILLES, S., AND WASER, R. 2009. High density 3D memory architecture based on the resistive switching effect. *J. Solid-State Electron. 53,* 12, 1287–1292.

LIKHAREV, K. K. 2008. Hybrid CMOS/nanoelectronic circuits: Opportunities and challenges. *J. Nanoelectron. Optoelectron. 3,* 3, 203–230.

LIN, S. AND COSTELLO, D. J. 2004. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Upper Saddle River, NJ.

LINCOLN, P. 2009. Challenges in scalable fault tolerance. In *Proceedings of IEEE/ACM International Symposium on Nanoscale Architectures*. 13–14.

LUYKEN, R. J. AND HOFMANN, F. 2003. Concept for hybrid CMOS-molecular non-volatile memories. *J. Nanosci. Nanotechnol. 14,* 2, 273–276.

MATHWORKS. Reed-Solomon decoder simulation. http://www.mathworks.com/matlabcentral.

MISHRA, M. AND GOLDSTEIN, S. C. 2003. Defect tolerance at the end of the roadmap. In *Proceedings of International Test Conference*. IEEE Computer Society, 1201–1210.

NAEIMI, H. AND DEHON, A. 2009. Fault secure encoder and decoder for nanomemory applications. *IEEE Trans. VLSI Syst. 17,* 4, 473–486.

NEUBERGER, G., LIMA, F. D., CARRO, L., AND RIES, R. 2003. A multiple bit upset tolerant SRAM memory. *ACM Trans. Design Autom. Electron. Syst. 8,* 4, 577–595.

ORAILOGLU, A. 2007. Nanoelectronic architectures: Reliable computation on defective devices. In *Digest of Workshop on Dependable and Secure Nanocomputing*.

REED, M. A., CHEN, J., RAWLETT, A. M., PRICE, D. W., AND TOUR, J. M. 2007. Molecular random access memory cell. *Appl. Phys. Lett. 78,* 23, 3735–3737.

RISPAL, L. AND SCHWALKE, U. 2008. 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. 29,* 412, 1349–1352.

STRUKOV, D. B. 2006. Digital architectures for hybrid CMOS/nanodevice circuits. Ph.D. thesis, Stony Brook University, NY.

STRUKOV, D. B. AND LIKHAREV, K. K. 2004. Prospects for terabit-scale nanoelectronic memories. *J. Nanosci. Nanotechnol. 16,* 1, 137–148.

STRUKOV, D. B. AND LIKHAREV, K. K. 2007. Defect-tolerant architectures for nanoelectronics crossbar memories. *J. Nanosci. Nanotechnol. 7*, 151–167.

SUN, F. AND ZHANG, T. 2007. Defect and transient fault-tolerant system design for hybrid CMOS/nanodevice digital memories. *IEEE Trans. Nanotechnol. 6,* 3, 341–351.

SUN, J. D. AND KRISHNA, H. 1992. A coding theory approach to error control in redundant residue number systems - Part II: Multiple error detection and correction. *IEEE Trans. Circuits Syst. 39,* 1, 18–34.

SZABO, N. AND TANAKA, R. 1967. *Residue Arithmetic and its Application to Computer Technology*. MC-Graw-Hill, New York.

WAISER, R. AND AONO, M. 2007. Nanoionics-based resistive switching memories. *Nature Materials 6,* 11, 833–840.

WANG, W., SWAMY, M. N. S., AHMAD, M. O., AND WANG, Y. 2003. A study of the residue-to-binary converters for the three-moduli sets. *IEEE Trans. Circuits Syst. I, Fundam. Theory Appl. 50,* 2, 235–243.

YANG, L.-L. AND HANZO, L. 2001. Redundant residue number system based error correction codes. In *Proceedings of the 54th Vehicular Technology Conference*. Vol. 3. 1472–1476.

ZETTACORE. Zettacore Memory. http://www.zettacore.com/.

Received March 2010; revised August 2010; accepted October 2010