Residue-Based Code for Reliable Hybrid Memories

Nor Zaidi Haron[1,2] Said Hamdioui[1]
1. Computer Engineering Laboratory, Delft University of Technology, The Netherlands
2. Faculty of Electronic and Computer Engineering, Universiti Teknikal Malaysia Melaka, Malaysia
{N.Z.B.Haron[1,2], S.Hamdioui[1]}@tudelft.nl, zaidi@utem.edu.my[1,2]

Abstract— **Hybrid memories, structured from scaled CMOS and non-CMOS devices, are novel memory architectures that offer trillion-capacity of data storage. In spite of that, the reliability of such memories is questionable because of (i) imprecise and immature fabrication processes and (ii) unreliable devices. This paper introduces the concept of Residue Number System (RNS), mainly used in digital signal processing and communication, to the realization of reliable hybrid memories. An error correction code based on RNS to mitigate cluster faults in hybrid memories is proposed; such code is referred to as Six Moduli Redundant Residue Number System (6M-RRNS) code. The experimental results show that 6M-RRNS code can achieve competitive error correction capability as the conventional RRNS (C-RRNS) and Reed-Solomon (RS) codes, yet at lower cost. E.g., for hybrid memories with word size of B=32 bits, the 6M-RRNS code requires 88 bits to encode the data, whereas C-RRNS and RS codes require 106 and 96 bits, respectively. It means that for a fixed memory size and given correction capability, the total data that can be stored when using 6M-RRNS coding is 20.4% and 9.1% larger as compared with C-RRNS and RS, respectively. Moreover, the speed at which 6M-RRNS decodes the data is 5.6 times faster than when using C-RRNS; hence allowing for higher performance.**

Index Terms — **Reliability, hybrid memories, error correction codes, residue number system**

I. Introduction

Hybrid memories, the future concept of data storage, are foreseen to extend the storage capacity offered by current CMOS-based memories. Structured from non-CMOS nanodevices (e.g., nanowires, single electron junctions, molecules) and nanoscale CMOS transistors, these kind of memories are predicted to provide up to 1 Tbit per centimeter square chip area [1]. Numerous hybrid memory circuits have been suggested such as CMOL memory [1], molecular memory [2], [3], and carbon nanotube memories [4]. In spite of trillion-capacity potential, such memories are more susceptible to non-permanent faults causing reliability challenges. The immature fabrication techniques (e.g., self-assemble, nanoimprint) might introduce latent defects like loose nanowires, poor CMOS/non-CMOS interface pins, etc. These defects are not detected by manufacturing testing, but they will cause faults some time during the operation of the memories. Reducing the operating voltage might save the total power, yet decreasing the noise-to-signal ratio. A small magnitude of particles energy could disturb the internal state of such nanodevices. Not only that, the influence of these faults might disturb adjacent cells in the memory array causing a cluster of faults.

In combating these reliability challenges, researchers have applied fault tolerance techniques to increase the reliability in hybrid memories. Techniques like error correcting codes (ECCs) [1], [5], [6], [7], [8], [9], [10], hardware redundancy [5], and defect-map [10] have been utilized to improve defect and fault tolerance of hybrid memories. Among these techniques ECCs is the most used due to their dynamic correction capability and lower cost compared to the others. The ECCs are such as Hamming [1], [5], Bose-Chaudhuri-Hocquenghem (BCH) [6], [7], Euclidean Geometry (EG) [8], and Low-Density Parity-Check (LDPC) [9]. Nevertheless, these conventional ECCs are based on either weak or random faults but not for cluster faults. Therefore, a new type of error correction is necessited to tolerate cluster faults in having reliable hybrid memories.

This paper presents the concept of *Redundant Residue Number Systems (RRNS)* as an error correction scheme for hybrid memories. It proposes a modified version of RRNS, which is suitable for the realization of reliable hybrid memories; it allows detection and correction of cluster faults. The proposed modified RRNS code is referred to as *Six Moduli Redundant Residue Number System (6M-RRNS)* as it based on six relatively co-prime moduli in generating a set of redundant residues. The experimental results show that 6M-RRNS code achieves competitive results, in terms of reliability improvement as compared to the conventional RRNS (C-RRNS) and Reed-Solomon (RS) codes, but then at lower cost and minimal impact on the performance.

The rest of the paper is organized as follows. Section II reviews the fundamental concept of hybrid memories. Section III discusses the basic theory of RRNS codes. Section IV introduces the 6M-RRNS code suitable for mitigating high degree of cluster faults. Section V presents an experimental analysis of the proposed 6M-RRNS including the comparison with RS and C-RRNS codes. Finally, Section VI draws the conclusion.

II. Fundamental Concept of Hybrid Memories

Figure 1 shows an example of a generic structure of hybrid memory constructed by integrating non-CMOS circuit on top of CMOS circuit [1]. The non-CMOS circuit consist of two perpendicular planes of nanowires and reconfigurable two-terminal nanodevices, e.g, single electron junction, organic molecule and phase change material [11], that form the memory cell array where data is stored. The reconfigurable two-terminal nanodevices embedded at each nanowire junction function as a single memory cell. Nanoscale CMOS devices build up peripheral circuits, e.g., encoding/decoding,

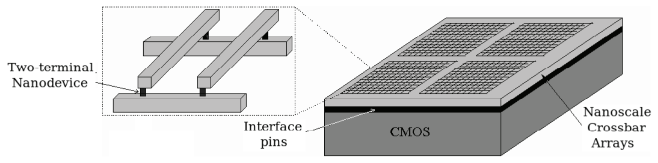


Fig. 1. Generic structure of hybrid CMOS/nanodevices circuits.

sensing global interconnecting, etc.

A specific memory cell can be accessed by activating the two perpendicular nanowires. Immediately after the access, a sufficient voltage (depending on the type of the two-terminal nanodevice being used) is biased to the memory cell (i.e., the two-terminal nanodevice) to change its internal state (resistance) for writing, or to supply appropriate current flow for reading.

Immature bottom-up fabrication technique (e.g., self-assembly) used to fabricate the nanodevices is potential to contribute to latent defects, for instance, loose nanowires and poor nanowire crossbar/CMOS interface, which lead to intermittent faults. Latent defects might escape manufacturing test but will cause reliability problem during operation lifetime of the device. In addition to that, as scaled CMOS and tiny non-CMOS devices operate using low voltage, the probability for these devices to be impacted by transient faults is high. At low voltage, charged-based devices like the one used in hybrid memories tend to have low signal-to-noise ratio [11]. A small energy strike of soft error is adequate to upset the logic state hold by these tiny devices. Also, the impact of intermittent and transient faults might affect several adjacent memory cells causing a cluster of faults. For instance, a loose nanowire will disturb several two-terminal non-CMOS devices connected to it. According to [14], based on the incident angle and the level of a particle flux, many nanoscale CMOS devices may suffer from transient faults.

In order to mitigate these latent defects and faults, and cnsequently to improve the overall reliability error correction circuit is included in hybrid memories. Fig. 2 exhibits the block diagram of the fault tolerance architecture considered in this paper.

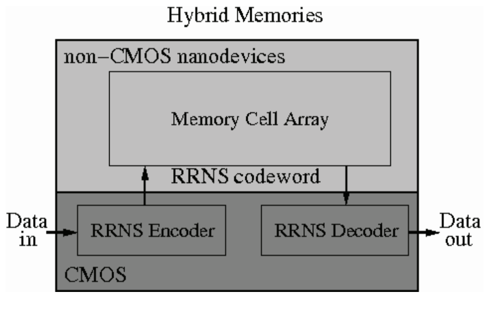


Fig. 2. RRNS Encoder and decoder in hybrid memories.

Prior to writing, input data must be encoded to RRNS codeword. The RRNS codeword is then stored in the memory cell array. When reading, the desired codeword is decoded before data can be read out. The decoding process will ensure read data is fault-free, provided the faults are still within the correction capability of RRNS decoder. In this work, the RRNS decoder are perceived to be ideal, where intermittent and transient faults can occur in RRNS encoder and memory cell array.

III. Redundant Residue Number Systems Code

In this section a concept of RRNS code is presented. This is followed by encoding and decoding procedure of this code.

A. Theory of RRNS code

RRNS code is a derivation of residue number system (RNS), which is usually found in high speed arithmetic operation applications (e.g., digital signal processing, cryptography, communication). RRNS code offers fast and built-in self-checking computation [12]. These advantages open a new direction in fault tolerance area, especially for error correcting codes, to improve the reliability of memory.

A RRNS codeword is constructed from a set of encoded numbers called *residues*. An input data of d bits will be encoded into n symbols *codeword*, which is divided into two set of residues (see Fig. 3): (i) *non-redundant residues*, x_i, consisting of k symbols *dataword*, and (ii) *redundant residues*, x_j, consisting of $(n-k)$ symbols *checkword (parity)*; $1 \leq i \leq k$ and $k+1 \leq j \leq n$. The bit length of k symbols maybe larger than the d bits input data ($k \geq d$) to have residue number system representation of the d bits. Note that, each residue is in the form of symbols, which resemble the codeword in RS code.

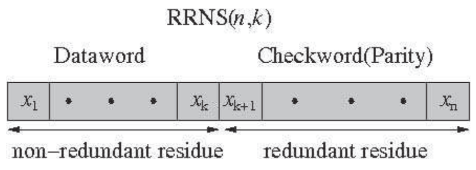


Fig. 3. Structure of RRNS code.

Each non-redundant residue is generated by performing modulo operation on the input data, say X, to a set of non-redundant moduli, m_i. Equivalently, redundant residues are generated by performing the same operation on X but then by using redundant moduli m_j instead of m_i. Note that the modulo operation finds the residue (remainder) of division of one number by another. These operation can be mathematically represented as the following equation [12].

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

The binary representation of x_i and x_j has bit length of $\lceil log_2(m_i) \rceil$ bits and $\lceil log_2(m_j) \rceil$ bits, respectively. Thus, the bit length of a RRNS codeword is the summation of these two bit lengths.

Example 1. An 8 bits input data can be encoded into 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 non-redundant moduli, m_i, and the last two moduli are redundant moduli, m_j. The RRNS codeword for input data $X=234$ will be:

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

Note that the input data is represented by (i) a dataword consisting of three non-redundant residues $0, 6, 5$, which are each represented by three bits ($\lceil log_2(5)\rceil$=$\lceil log_2(7)\rceil$=$\lceil log_2(8)\rceil$= 3), and (ii) a checkword consisting of two redundant residues $8, 4$ with a total length of 8 bits. Therefore, the 8 bits input data is encoded into 17 bits RRNS codeword.

For a RRNS codeword to be consistently decoded (i.e., to prevent different codewords to be decoded into the same output data), the moduli must satisfy three rules; (i) a pair of any two moduli, say m_a and m_b with $a{\neq}b$, must be co-primes such that their greatest common divisor $gcd(m_a, m_b){=}1$; (ii) the integer value for succeeding modulus is greater than the preceding modulus, i.e., $m_1{<}...{<}m_k{<}m_{k+1}{<}...{<}m_n$; and (iii) The product of moduli M_i, is sufficient to represent all numbers in the *legitimate range* of $[0{,}M_i{-}1]$ [12].

In additional to the three rules, the redundant moduli are chosen arbitrarily such that (i) they ensure the desired error correction capability; and (ii) their product is sufficient to represent all the numbers in legitimate range.

Note that each modulus in RRNS code can have different bit length depending on the chosen moduli. Hence, choosing appropriate moduli can reduce the total bit length of codeword. This characteristic is not possessed by RS code, where all symbols have fixed lengths.

B. *RRNS encoding*

RRNS encoding is straightforward; it is based on performing modulo operations on input data to a moduli set as explained before. The resulted residues from modulo operations are obtained simultaneously; i.e., the operations are executed in parallel by corresponding modulo circuits. The residues are then concatenated to be a RRNS codeword before it is stored in memory cells. Similar to RS code, the correction capability of RRNS is defined by $t{=}\frac{(n-k)}{2}$ [12], [13] (see Figure 3). E.g., to correct one erroneous residue in a RRNS codeword, two residues of checkword must be appended to dataword.

C. *RRNS Decoding*

Decoding of RRNS codeword has two phases: (i) detection and (ii) correction of errors. Detection of errors must be applied to any read codeword from the memory. A codeword is valid if its decoded value is within the legitimate range, thus no error correction is needed. In opposition, invalid data is determined when the value of a decoded codeword is larger than legitimate range, hence error correction is needed.

During correction phase a systematic calculation is conducted exhaustively to search for values that are within the legitimate range. This exhaustive search will be performed until the valid data is recovered; it requires C^n_t iterations. In each iteration, t number of residues are discarded, and the calculation is executed based on the remaining $(n{-}t)$ residues. Ideally, from all recovered integer values, there will be at least one unique value fall within the legitimate range, which is the correct data.

In some cases, however, more than one recovered values are within the legitimate range. This ambiguity can be solved using *maximum likelihood decoding* scheme [16], which will determine the actual correct data between the recovered values. The idea behind the scheme is the closest Hamming distance between among the values fallen within the legitimate range and the read codeword.

Two algorithms can be used in the decoding process: (i) *Chinese Remainder Theorem (CRT)* and (ii) *Mixed-Radix Conversion* [12]. In this work, MRC is used because it deals with smaller integers; this facilitates the simulation work. MRC is based on the following equation.

$$X_r = \sum_{s=1}^{n} v_s w_s \tag{2}$$

where v_s is calculated as

$$v_1 = |X|_{m_1} = x_1 \tag{3}$$

$$v_s = \left|\left(\left((x_s - v_1) \times m_{1s}\right) ..... - v_{(s-1)}\right) \times m_{(s-1)s}\right|_{m_s} \tag{4}$$

where $x_s{=}|X|_{m_s}$ and $m_{(s-n)s}$ is the multiplicative inverse of $m_{(s-n)}$ with respect to m_s defined as $|m_{(s-n)}m^{-1}_{(s-n)}|_{m_s}{=}1$; $s \in \{2, 3, ..., n\}$ and $n \in \{1, 2, 3, ..., n-1\}$. Whereas w_s is

$$w_1 = 1, \qquad w_s = \prod_{s=2}^{n-1} m_s \tag{5}$$

Example 2. Assume that the fourth residue of the codeword in Example 1 (i.e., 0) is corrupted during the storing, which results in $x{=}\{4, 3, 2, \mathbf{8}, 3\}$. Decoding the codeword using MRC requires the calculation of multiplicative inverses. E.g., $m_{12}{=}|m_1^{-1}|_{m_2}{=}3$ because $|m_1{\times}m_1^{-1}|_{m_2}{=}|5{\times}3|_7{=}1$. In a similar way, the other multiplicative inverses are calculated: $m_{13}{=}5$, $m_{23}{=}7$, $m_{14}{=}2$, $m_{24}{=}4$, $m_{34}{=}8$, $m_{15}{=}9$, $m_{25}{=}8$, $m_{35}{=}7$, $m_{45}{=}5$. The legitimate range for this codeword is $M_i{=}5{\times}7{\times}8{=}280$.

Based MRC equations above, the validity of the codeword is checked.

$v_1{=}4, \qquad v_2{=}|(3-4)(3)|_7{=}4,$
$v_3{=}|((2-4) \times 5 - 4) \times 7|_8{=}6,$
$v_4{=}|(((8-4) \times 2 - 4) \times 4) - 4) \times 8|_9{=}8,$
$v_5{=}|((((3-4) \times 9 - 4) \times 8) - 4) \times 7 - 8) \times 5|_{11}{=}4$

$$\begin{aligned} X_r &= \textstyle\sum_{s=1}^{n} v_s w_s \\ &= (4 \times 1) + (4 \times 5) + (6 \times 5 \times 7) + (8 \times 5 \times 7 \times 8) \\ &\quad +(4 \times 5 \times 7 \times 8 \times 9) \end{aligned}$$

$X_r = 12554 > 280$ (error detected)

Error correction is invoked since the codeword is invalid. A systematic iterative calculation that discards a residue in each iteration is performed. There are C^5_1 iterations since $t{=}\frac{5-3}{2}$ (see Section III.C). This exhaustive search produces the integer values as shown in Table II. Since only $X_{r'_4}{<}280$, the correct data $X{=}234$ is recovered.

TABLE I

CODEWORD PARAMETERS FOR 16- AND 32 BITS RS, C-RRNS, AND 6M-RRNS CODES.

Data	ECC Types	Moduli Sets, m_a: Non-Redundant	Moduli Sets, m_a: Redundant	Number of Residues/Symbols: Dataword	Number of Residues/Symbols: Checkword	Number of Residues/Symbols: Codeword	*Bits/Codeword
16 bits	RS	–	–	4	8	12	48
	C-RRNS	64,63,65	67,71,73,79,83,89	3	6	9	61
	6M-RRNS	257,256	127,63,31,17	2	4	6	40
32 bits	RS	–	–	4	8	12	96
	C-RRNS	2047,2048,2049	2051,2053,2057,2059,2063,2069	3	6	9	106
	6M-RRNS	65537,65536	32767,16383,8191,4097	2	4	6	88

*Bits/Codeword=$\lceil \log_2 m_a \rceil$; 1≤a≤n

TABLE II

RESULT OF CORRECTING SINGLE ERRONEOUS RESIDUE.

Iteration, i	1	2	3	4	5
Discarded Residue, x_i	x_1	x_2	x_3	x_4	x_5
Recovered Data, $X_{r'_i}$	1466	3594	2159	234	2474

IV. RRNS CODES FOR HYBRID MEMORIES

A. Conventional RRNS Code

Conventional RRNS code termed as C-RRNS is based on $\{2^n-1, 2^n, 2^n+1\}$ *restricted* non-redundant moduli, which can be realized in small and fast hardware [17]. *Unrestricted* redundant moduli are commonly appended to the non-redundant moduli in generating a RRNS codeword. Unrestricted redundant moduli are selected from any number of pairwise relatively prime positive integer, which have larger integer values than those of the restricted non-redundant moduli. Here, unrestricted means that the selected moduli are not based on a set of generic equation but they can be any values satisfying the RRNS criteria.

For 16 bits memory word, the integer value for restricted non-redundant residues $\{2^n-1, 2^n, 2^n+1\}$ must be selected such that their product is at least $2^{16}-1$. To satisfy this criterion, the minimum value of n=6 is chosen to alleviate the area and performance overhead; this results into the non-redundant moduli m_i=$\{64, 63, 65\}$, with legitimate range of M_i=262080; see Table I. The redundant moduli is chosen to be m_j=$\{67, 71, 73, 79, 83, 89\}$ where the product of the redundant moduli is obviously more than $2^{16}-1$. The six-residue checkword is needed to ensure the protection of three-residue dataword, i.e., t=3. The length of this codeword is $\sum_{s=1}^{n} \lceil log_2 m_s \rceil$=61 bits where m_s are the moduli used to generate the codeword.

For 32 bits memory word, the minimum value of n that produces the required legitimate range is n=11, resulting into m_i=$\{2047, 2048, 2049\}$. For the same reason as for 16 bits memory word, the redundant moduli is chosen to be m_j=$\{2051, 2053, 2057, 2059, 2063, 2069\}$. The bit length of this codeword is 106 bits.

B. Modified RRNS Code

RS code is also used as ECC for memories (e.g., flash memories). Table I shows the required number of bits for 16 and 32 bits memory word when RS coding is used. Note that the length of RS codeword is shorter than that of C-RRNS in both cases. However, RS code has lower correction capability than C-RRNS.

To reduce the length of C-RRNS codeword while providing a competitive correction capability, a modified version of C-RRNS will be introduced. The basic ideas behind this are:

- For non-redundant moduli
  1) The number of moduli set can be kept small, provided that their product is at least equal to the legitimate range (see Section III.A),
  2) Smaller number of moduli set requires smaller number of redundant moduli for error correction (see Section III.B).
- For redundant moduli
  1) The integer values for all redundant moduli can be made smaller than that of non-redundant moduli as long as the product is sufficiently larger than legitimate range.

The modified version of C-RRNS is referred to as *Six-Moduli RRNS (6M-RRNS)*; it is based on six residues: (i) two are non-redundant moduli and (ii) four are redundant moduli. The non-redundant moduli set is $\{2^n+1, 2^n\}$, while redundant moduli set is $\{2^{n-1}-1, 2^{n-2}-1, 2^{n-3}-1, 2^{n-4}+1\}$. Note that in case of 6M-RRNS both moduli are *restricted*, while for C-RRNS this is only the case for the non-redundant moduli. These generalized six moduli are adopted to realize simple hardware implementation.

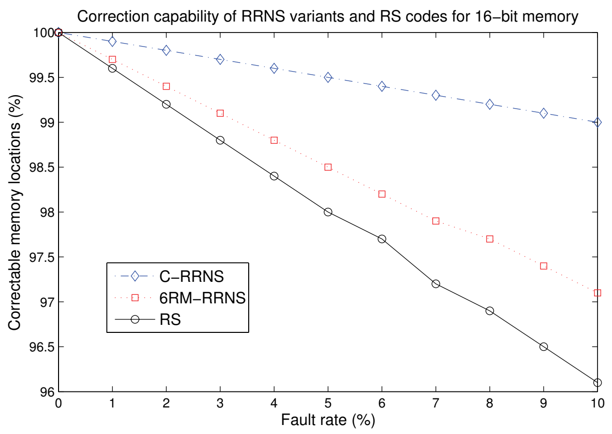
For 16 bits memory word, the smallest number n that satisfies the requirement that the product of non-redundant moduli $\{2^n+1, 2^n\}$ is larger than the legitimate range is n=8. This results in m_i=$\{257, 256\}$ and m_j=$\{127, 63, 31, 17\}$. Although the moduli consist of smaller integer values than that of redundant moduli, the product M_j=4216527 is clearly more than legitimate range of 16 bits. The benefit of using two non-redundant moduli in 6M-RRNS code instead of three as in C-RRNS is the reduction of the total length of the codeword.

In a similar way the required n for 32 bits memory word can be calculated. This results in n=16; hence m_i=$\{65537, 65536\}$ and m_j=$\{32767, 16383, 8191, 4097\}$.

V. EVALUATION AND ANALYSIS

A. Simulation Setup

To evaluate the 6M-RRNS code, a comparison with C-RRNS and RS was performed; such comparison is relevant because both codes are designed for correcting multiple bit



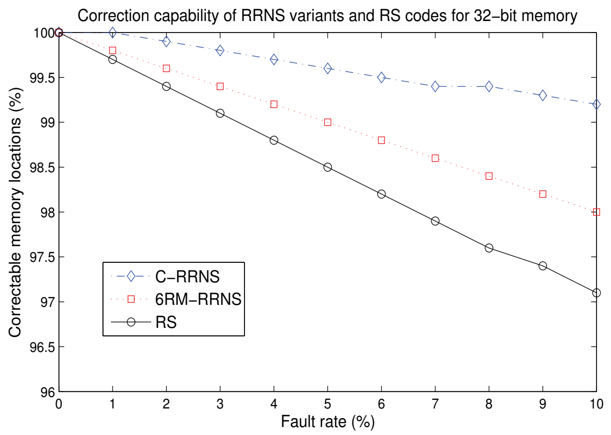


Fig. 4. Simulation results for 16 and 32 bits memory word.

TABLE III

COMPARISON ON BIT LENGTH FOR 16-AND 32 BITS RS, C-RRNS, AND 6M-RRNS CODES.

Data	ECC Type	Bit Length				
		Dataword	Checkword	Codeword	(Codeword Differences)(%)	
		b_{k_i}	$b_{(n-k)_i}$	b_{n_i}	(compared to RS)	(compared to C-RRNS)
16 bits	RS	16	32	48	–	-21.3
	C-RRNS	19	42	61	+27.1	–
	6M-RRNS	17	23	40	-16.7	-34.4
32 bits	RS	32	64	96	–	-9.4
	C-RRNS	34	72	106	+10.4	–
	6M-RRNS	33	55	88	-8.3	-17

clustered errors. For these codes, the error correction capability is equal to half the number of residues (symbols) per checkword.

As mentioned in Section III.A, M_i is taken as a reference to determine the validity of a codeword. In this work, nonetheless, M_i is set up to M_{16}=2^{16}−1 and M_{32}=2^{32}−1 for 16 and 32 bits word memory, respectively. These are considered to alleviate data that larger than M_{16} and M_{32} to be wrongly decoded. E.g., for a 32 bits memory word. when M_{32} is taken as reference, then any decoded data with a value from 2^{32} to 2047×2048×2049=8589932543 (i.e., the product of non-redundant moduli), will be considered as erroneous data.

The RRNS variants and RS codes, 4K×16 bits word and 4K×32 bits word memories, and fault injection were described using MATLAB script. All codes were set to the corresponding t to protect the codeword from faults. For RRNS decoding process, MRC was implemented. For RS code, MATLAB built-in RS encoding and decoding functions were used [18]. An appropriate adjustment for polynomial generator was done to encode and decode 16 and 32 bits memory word.

Faults were uniformly injected at each memory location. The faults were increased from single bit up to 20 clustered bits per codeword for 16 bits word memory data, and single bit up to 35 clustered bits per codeword for 32 bits memory word data. Various fault rate, from 1% to 10%, were applied during the experiments. The size of total memory capacity, whether to use 4096 locations or more, is not so important because faults were uniformly distributed.

B. Results

Table III shows the number of bits that represent the codeword for each code. Overall, 6M-RRNS has the smallest number of codeword bit length follows by RS and C-RRNS codes. For 16 bits memory word data, 6M-RRNS realizes a codewords which are 16.7% and 34.4% shorter as compared to RS and C-RRNS codes, respectively. Whilst for 32 bits memory word data, 6M-RRNS realizes a codewords which are 8.3% and 17% shorter as compared to RS and C-RRNS codes, respectively.

Fig. 4 shows the simulation results for C-RRNS, 6M-RRNS and RS codes. The correction capability of all codes reduces as the fault rate becomes higher. At 10% fault rate all the codes ensure that more than 96% and 97% of the 16 and 32 bits word memory locations, respectively, are reliable. Both RRNS variants perform better than RS code in correcting faults for irrespectively of faults rate. For 16 bits memory word, at 10% fault rate, C-RRNS code can correct 99% of the total faulty locations, whereas 6M-RRNS and RS code can correct 97.1% 96.1%, respectively, of the faulty locations. For 32 bits word memory and at the same fault rate, all codes perform slightly higher; C-RRNS with 99.2%, 6M-RRNS with 98%, and RS with 97.1%.

C. Analysis

Although C-RRNS code is able to correct the largest number of erroneous bits as shown in Fig. 4(b), 6M-RRNS code is the best if we look at different perspectives. These include ratio of correction capability over generated codeword length, ratio of correction capability over fixed codeword length, capacity of data storage over fixed memory size, and decoding latency of RRNS codes; they are analyzed in the followings.

First, consider the ratio of correction capability over generated codeword length. For 16 bits memory word coded into 6M-RRNS, the maximum number of erroneous bits it can correct is 17 out of 40 bits codeword, which means 42.5% of each memory word will be corrected. RS and C-RRNS possess only 16/48=33.3% and 19/61=31.1%, respectively. For 32 bits memory word, 6M-RRNS, RS, and C-RRNS codes ensure the correction of 37.5%, 33.3%, and 32% of each memory word, respectively. Thus, 6M-RRNS provides the highest bit-wise error correction capability in this case.

Second, consider of the capability of the error correction codes when assuming a fixed length of codeword. In this case, RS and C-RRNS codes will be to 40 bits codeword (i.e., by taking off the last two- and three-residue checkword, respectively) to have similar bit length to 6M-RRNS. These reductions decrease the error correction capability to at most t=$\lfloor\frac{10-4}{2}\rfloor$=3 for RS, t=$\lfloor\frac{6-3}{2}\rfloor$=1 for C-RRNS, and t=$\lfloor\frac{7-3}{2}\rfloor$=2 for 3NRM-RRNS codes. The maximum 16 bits word memory data that can be corrected for RS is now 3×4=12 bits, C-RRNS is 1×7=7 bits, and 3NRM-RRNS is 9+8=17 bits. Yet, all 16 bits word data encoded into 2NRM-RRNS are still protected by the four-residue checkword. In a similar way, one can find that for 32 bits memory word when considering a codeword of 88 bits in size, then the maximum bit-wise correction capability will be 3×8=24 bits, C-RRNS is 1×12=12 bits, and 3NRM-RRNS is 17+16=33 bits. Hence, 6M-RRNS is the best to realize a higher error correction capability when fixed length of codeword is considered.

Third, consider the capacity of the data storage given a fixed memory chip size. If the input data is coded with the three different considered schemes, then 6M-RRNS will realize the highest data storage because of its compact codeword. It is reported in one that [1] 1Tbit hybrid memory can be fabricated in a centimeter square. Considering such memory size for 16 bits memory word, 6M-RRNS will be able to store 20% and 52.5% more data than RS and C-RRNS, respectively. This is because 6M-RRNS allows the storage of $\#C_{6M}$=1T/40 codewords in the memory, while the use of C-RNNS allows for the storage of $\#C_C$=1T/61 codewords. Note that the size of a codeword for 6M-RRNS is 40 bits and that for C-RRNS is 61 bits. The difference in storage is then calculated as $\frac{\#C_{6M}-\#C_C}{\#C_C}$=52.5%. Redoing the calculation for 1T organized into 32 bits memory word results in 9.1% and 20.4% larger data for 6M-RRNS as compared to RS and C-RRNS, respectively. Therefore, 6M-RRNS offers the biggest data storage when considering a fixed memory chip size.

Fourth, consider the decoding time for RRNS codes. Since 6M-RRNS code hold at most two error correction capability, the code requires maximum C_2^6=15 iterations during decoding procedure. However, for C-RRNS code the procedure needs maximum C_3^9=84 iterations. Hence, 6M-RRNS performs 5.6 times faster decoding than C-RRNS.

VI. Conclusion

In this paper the concept of Redundant Residue Number Systems (RRNS) is introduced for fault tolerance hybrid memories. A modified RRNS code has been proposed while targeting higher degree of cluster faults. The proposed version, referred to as 6M-RRNS, is based on six redundant residues in generating the RRNS codeword. The experimental results show that 6M-RRNS code has shorter codeword than well-known Reed-Solomon (RS) and conventional RRNS codes (C-RRNS). Hence, allowing for more data storage for a given fixed memory size. Moreover, the 6M-RRNS has a competitive error correction capability as compared to RS and C-RRNS. Future work is to enhance the fault tolerance capability, e.g., by combining RRNS codes to other techniques like N-tuple modular redundancy and scrubbing.

References

[1] D. B. Strukov and K. K. Likharev, "Prospects for terabit-scale nanoelectronic memories", *Nanotechnology*, vol. 16, pp. 137–148, 2005.

[2] Zettacore^{TM}. *ZettaCore^{TM}memory*. http://www.zettacore.com/

[3] K. Bullis, "Ultradense Molecular Memory: Researchers develop a large-scale array of nanoscale memory circuits", *MIT Technology Review*, $http://www.technologyreview.com/Nanotech/18100/$

[4] 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 Letters*, vol. 29, iss. 12, pp. 1349–1352, Dec 2008.

[5] C.M. Jeffery, A. Basagalar, and R. J. O Figueiredo, "Dynamic sparing and error correcting techniques for fault tolerance in nanoscale memory structures", *in Proc. of IEEE Conf. on Nanotechnology*, pp. 168–170, Aug. 2004.

[6] D. B. Strukov and K. K. Likharev, "Architectures for defect-tolerant nanoelectronic crossbar memories", *Nanotechnology*, vol. 7, pp. 151–167, 2007.

[7] F. Sun and T. Zhang, "Two Fault Tolerance Design Approaches for Hybrid CMOS/Nanodevice Digital Memories", *in Proc. of IEEE Int'l Work'p on Defect and Fault Tolerant Nanoscale Architectures*, 2006.

[8] H. Naeimi and A. DeHon, "Fault Tolerant Nano-Memory with Fault Secure Encoder", *in Proc. Int'l Conf. on Nano-Networks (Nanonets 2007)*, Sept. 2007.

[9] S. Ghosh and P.D. Lincoln, "Dynamic Low-Density Parity Check Codes for Fault-tolerant Nanoscale Memory Fault-tolerant Nanoscale Memory", online at $http://www.csl.sri.com/users/shalini/ldpc.pdf$.

[10] S. Biswas, T. S. Metodi, F. T. Chong, and R. Kastner, " A Pageable, Defect-Tolerant Nanoscale Memory System", *in Proc. of IEEE Int'l Symposium on Nanoscale Architecture*, pp. 85–92, 2007.

[11] K. K. Likharev, "Hybrid CMOS/Nanoelectronic Circuits: Opportunities and Challenges", *J. of Nanoelectronics and Optoelectronics*, vol. 3, pp. 203–230, 2008.

[12] N. Szabo and R. Tanaka. *Residue Arithmetic and its Application to Computer Technology*. MC-Graw-Hill. New York. 1967.

[13] J. -D. Sun and H. Krishna, "A coding theory approach to error control in redundant residue number system - Part II: multiple error detection and correction", *IEEE Trans. on Circuits and Systems*, pp. vol. 39, 18–34, Jan 1992.

[14] R. C. Baumann, "Soft Error in Advanced Semiconductor Devices–Part 1: The Three Radiation Sources", *IEEE Trans. on Device and Materials Reliability*, Vol. 1, No. 1, pp. 17–22, March 2001.

[15] L. Yang and Lajos Hanzo, "Coding theory and performance of redundant residue number system codes", $http://www-mobile.ecs.soton.ac.uk/$

[16] V. T. Goh and M. U. Siddiqi, "Multiple Error Detection and Correction based on Redundant Residue Number Systems", *IEEE Trans. on Communications*, vol. 56, no. 3, pp. 325–330, March 2008.

[17] F. Barsini and P. Maestrini, "Error correcting properties of redundant residue number systems", *IEEE Transactions of Computers*, vol. 2, no. 2, pp. 915–923, 1973.

[18] MathWorks^{TM}. Reed-Solomon Decoder Simulation. $http://www.mathworks.com/matlabcentral$