2009 24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems

# Using RRNS Codes for Cluster Faults Tolerance in 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, Univeristi Teknikal Malaysia Melaka, Malaysia*

{*N.Z.B.Haron, S.Hamdioui*}*@tudelft.nl*[1], *zaidi@utem.edu.my*[2]

**_Abstract_**

*Hybrid CMOS/non-CMOS memories, in short hybrid memories, have been lauded as future ultra-capacity data memories. Nonetheless, such memories are going to suffer from high degree of cluster faults, which impact their reliability. This paper proposes two modified Redundant Residue Number Systems (RRNS) based error correcting codes to tolerate cluster faults in hybrid memories, namely (i) Three Non-Redundant Moduli RRNS (3NRM-RRNS) and (ii) Two Non-Redundant Moduli RRNS (2NRM-RRNS). Experimental results and analysis show that 3NRM-RRNS and 2NRM-RRNS possess competitive error correction capability to that of Reed-Solomon (RS) and conventional RRNS (C-RRNS), but at lower cost (reduced code size, lower performance penalty). E.g., for 16-bit memory 2NRM-RRNS provides a bit-wise error correction capability up to $t$=41.5% using 41 bits codeword, whereas RS offers only up to $t$=33.3% using 48 bits and C-RRNS supports up to $t$=31.1% using 61 bits. In addition, 2NRM-RRNS is 5.6 times faster than C-RRNS in recovering a correct data, which in turn results in higher speed decoding performance.*

## 1. Introduction

The remarkable reduction in CMOS transistor geometry and the introduction of non-CMOS nanodevices have made it possible to have ultimate large-scale of hybrid memories for storing digital data. Several research groups from, e.g., academia [1, 2, 3] and industries [4, 5] have come out with their hybrid memories. According to [1], these kind of memories are able to store data up to 1 TBit/cm$^2$. Nonetheless, such memories are likely to suffer from high degree of multiple bit upset due to transient faults. Constant technology scaling of CMOS enables a reduction in voltage and capacitance, on the other hand, lowering the signal-to-noise ratio. Likewise, the charged-based non-CMOS nanodevices (e.g., single electron junctions and molecules), which require low voltage to change their internal state, will have a similar impact to that of nanoscale CMOS. Moreover, because the components are very tiny, the effect of these faults may impact several neighboring components leading to cluster faults.

Research on applying fault tolerance techniques to hybrid memories is growing in order to improve their reliability. Fault tolerance techniques such as hardware redundancy [6, 7], reconfiguration [8], and error correction codes (ECCs) [1, 6]–[11] have been utilized to tolerate faults. Among these fault tolerance techniques ECCs is the most used. ECCs like Hamming [1, 6], Bose-Chaudhuri-Hocquenghem (BCH) [9, 8], and Euclidean Geometry (EG) [10, 11] have been employed as part of fault tolerance techniques for hybrid memories. Nevertheless, these works rely on

1550-5774/09 $26.00 © 2009 IEEE
DOI 10.1109/DFT.2009.37

conventional ECCs, which are based on random faults but not for cluster faults. Hence, a new type of error correction is required to mitigate cluster faults.

*Redundant Residue Number Systems (RRNS)* code is proposed to correct cluster faults [13]. Current applications, that use RRNS to mitigate faults, include digital signal processing [13, 14] and communication [15] but not in memory technology. Furthermore, existing applications employ conventional RRNS, which is subject to high cost. In this paper two modified RRNS codes are proposed to tolerate cluster faults in hybrid memories with low cost overhead.

The rest of the paper is organized as follows. Section 2 reviews the fundamental structure of hybrid memories. Section 3 discusses the basic concept of RRNS codes. Section 4 introduces the two modified versions of RRNS code suitable for cluster faults. Section 5 presents an experimental analysis of the proposed RRNS variants including the comparison with RS and conventional RRNS codes. Lastly, Section 6 draws the conclusion.

## 2. Hybrid Memories Structure

Fig. 1 shows the generic structure of hybrid CMOS/non-CMOS circuits [1]. The circuits consist of arrays of nanowire (or carbon nanotubes) crossbar fabricated on top of nanoscale CMOS circuit. At each nanowire crosspoint, a two-terminal nanodevice (e.g, single electron junction, organic molecule, or phase change material) is embedded that serve as a single bit memory cell. In such memories, a specific memory cell can be accessed by activating its (local) column and row nanowires (the crossbar 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 to change its internal state (resistance) for writing, or to supply appropriate current flow for reading. However, the state-of-the-art of non-CMOS devices cannot amplify signals as CMOS can do; thus require CMOS to perform the peripheral task (e.g., encoding/decoding, sensing global interconnecting). Cone-type interface pins are used to connect the CMOS circuits to nanoscale crossbar arrays.

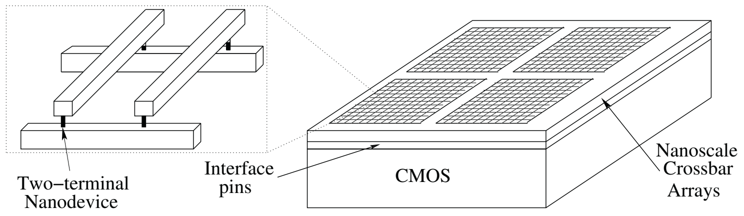


**Figure 1. Generic structure of hybrid CMOS/nanodevices circuits.**

The CMOS circuit and nanoscale crossbar arrays are fabricated using top-down techniques (e.g., extreme ultra violet lithography and nanoimprint). Whilst, two-terminal devices and interface pins are structured using bottom-up technique (e.g., self-assembly). Nevertheless, due to imprecise top-down fabrication technique, defects such as broken/open/short/misalignment/loosen nanowires are probable to occur. The immature bottom-up fabrication techniques may also cause defects like open/close/missing of two-terminal nanodevices and interface pins. These defects might result in hard or intermittent faults leading to low production yield.

In addition, the tiny components are likely to suffer from transient faults due to lower signal-to-noise ratio and parametric variations [1, 8, 12]. When this problem happens, the impacted memory cells may flip the stored data causing reliability problem during the operation of memory-based systems. Moreover, because the nanodevices used to structure the nanoscale crossbar arrays are closely connected, the impact of defects and faults might last up to several contiguous memory cells resulting in cluster faults.

## 3. Redundant Residue Number Systems Codes

RRNS code is a type of *block codes* where dataword is explicitly distinguished from checkword (parity). A RRNS codeword is divided into two parts, namely (i) non-redundant residue, which is represented by $k$ symbols of $x_a$; $1 \leq a \leq k$, and (ii) redundant residue, which is delineated by $(n-k)$ symbols of $x_b$; $k+1 \leq b \leq n$ as shown in Fig. 2 [15]. Here, $n$ and $k$ symbols consist of a number of bits. The non-redundant residues and redundant residues act as dataword and checkword (parity), respectively. Note that a residue is the remainder of a division. RRNS codes possess equivalent error correction capability to RS codes, i.e., $t = \lfloor \frac{n-k}{2} \rfloor$.

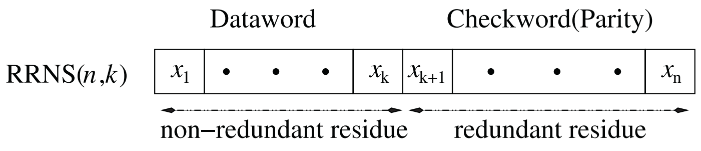


**Figure 2. Structure of RRNS codes with $k$ symbols dataword and $(n-k)$ symbols checkword.**

In producing the residues, a set of non-redundant moduli $m_a$, and a set of redundant moduli $m_b$ are used. These moduli sets must fulfill three rules; they are explained next [13]:

1. The moduli comprise of pairwise relatively prime positive integers, such that the greatest common divisor for any $i^{th}$ and $j^{th}$ $(i \neq j)$ moduli is $gcd(i,j) = 1$ .
2. The integer value for succeeding modulus is greater than the preceding modulus, i.e., $\{m_1 < ... < m_k < m_{k+1} < ... < m_n\}$.
3. The products of the moduli $M_a$ and $M_b$, must be sufficient to represent all numbers in the *legitimate range* of [0,$M_a - 1$] and [0,$M_b - 1$], respectively.

Fig. 3 illustrates the basic block diagram of RRNS encoder/decoder in a memory system. A $d$ bits input data is encoded into a set of $i$ bits of residues (RRNS codeword) by modulo converters in parallel. Note that $i$ is different for each residue depending on the moduli used; the length is defined as $i = \lceil \log_2 m_l \rceil$; $1 \leq l \leq n$. The codeword is then stored in blocks of memory cell array. When reading the selected codeword is decoded to check its validity. If it is valid, the decoded data is read and sent out. However, if it is invalid, a correction procedure is performed for recovering the correct data before it is sent out. The following subsections explain the encoding and decoding of RRNS code.

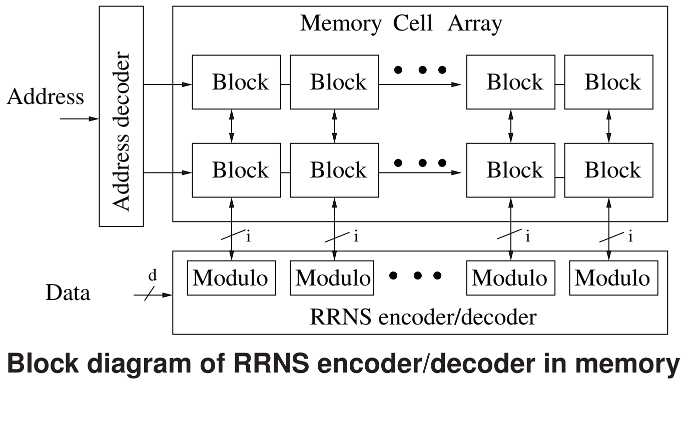


**Figure 3. Block diagram of RRNS encoder/decoder in memory system.**

## 3.1. Encoding of RRNS code

Encoding of RRNS code is straightforward by performing modulo operation of input data to the chosen moduli set. Example 1 explains the encoding calculation, which uses decimal number instead of binary for easy understanding and readability.

**Example 1.** A 8 bits RRNS codeword will be generated based on the moduli set $\{m_1, m_2, m_3, m_4, m_5\}=\{5, 7, 8, 9, 11\}$, where the first three moduli are non-redundant moduli $m_a$, and the last two moduli are the redundant moduli $m_b$. The RRNS codeword for input data $X$=125 will be as follow,

$$x_{k_i} = \{|125|_5, |125|_7, |125|_8, |125|_9, |125|_{11}\}$$
$$x_{k_i} = \{0, 6, 5, 8, 4\}$$

Note that the RRNS dataword is represented by three non-redundant of integers with small values $(0, 6, 5)$ instead of an integers with a larger values ($X$=125). The integers with small values enable fast computation to be performed during decoding. Note that the considered RRNS codeword in this example has two redundant residues, which means that it is able to correct one erroneous residue ($t=\frac{n-k}{2}=\frac{5-3}{2}=1$), or detect two erroneous residues ($n-k=5-3=2$).

## 3.2. Decoding of RRNS code

Decoding of RRNS codeword is executed first by detecting the validity of a codeword and then by correcting the erroneous residues, if any. During detection the codeword is compared to a predetermined value $M_a$. The codeword is regarded as valid (no correction is required) if its value is less than $M_a$. On the other hand, it is regarded as invalid if its value is equal or larger than $M_a$. Error correction is required in the second case, where numerous steps are executed to search for the correct data. Two algorithms can be used, either *Chinese Remainder Theorem (CRT)* or *Mixed-Radix Conversion (MRC)* [13]. CRT and MRC use large and small integers, respectively, in calculating the data. Thus, in this paper MRC is used due to less complex calculation. Note that, the accomplishment of correcting the error depends on the correction capability; i.e., the number of errors to be corrected must be less or equal to the capability of the RRNS code. The error detection and correction procedures are described as follows:

1. Compute/decode the read codeword to $X_n$ (data) using all $n$ residues.
2. If $X_n \leq M_a$, no error thus output the $X_n$.
3. If $X_n > M_a$, errors are detected. Error correction procedure starts by discarding $t$ number of residues where $t$ is error correction capability of the RRNS code. Compute $X_z$ from the remaining $z=(n-t)$ residues, and compare to the product of the remaining moduli $M_z$. Note that the minimum $M_z$ will be $M_b$; that is the reason $M_b$ must be larger than the legitimate range. This procedure is a trial and error step and must be repeated for maximum $C_t^n$ times, each with difference combination of remaining residues.
4. If any of the $X_z$ falls within the $M_z$, then $X_z$ is the corrected version of input data.
5. If all of the $X_z$ fall beyond the $M_z$, then RRNS codes cannot correct the erroneous codeword.

**Mixed-Radix Conversion (MRC).** MRC is based on the following equation [13].

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

where $v_s$ is calculated as

$$v_1 = |X|_{m_1} = x_1, \qquad v_s = \left|\Big(((x_s - v_1) \times m_{1s}) ..... - v_{(s-1)}\Big) \times m_{(s-1)s}\right|_{m_s} \tag{2}$$

where $x_s{=}|X|_{m_s}$ and $m_{(s-h)s}$ is the multiplicative inverse of $m_{(s-h)}$ with respect to $m_s$ defined as $|m_{(s-h)}{\times}m_{(s-h)}^{-1}|_{m_s}{=}1$ for $s{>}h$, $2{\leq}s{\leq}n$, and $1{\leq}h{\leq}n$-1. Whereas $w_s$ is

$$w_1 = 1, \qquad w_s = \prod_{s=2}^{n-1} m_s \tag{3}$$

**Example 2.** Assume that the third residue of the codeword of Example 1 (i.e., 5) is corrupted during the storing, which results in $x{=}\{0, 6, \mathbf{1}, 8, 4\}$. 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, which result in $m_{13}{=}5$, $m_{23}{=}7$, $m_{14}{=}2$, $m_{24}{=}4$, $m_{34}{=}8$, $m_{15}{=}9$, $m_{25}{=}8$, $m_{35}{=}7$, $m_{45}{=}5$. The legitimate range for this codeword is $M_a = 5{\times}7{\times}8{=}280$.

- Step 1 of the decoding procedure is performed by using Equations 1–3,

  $v_1{=}|125|_5{=}0, \qquad v_2{=}|(6-0)(3)|_7{=}4$

  $v_3 = |((1-0)(5)-4)(7)|_8 = 7$

  $v_4{=}|(((8-0)(2)-4)(4)-7)(8)|_9{=}4$

  $v_5{=}|((((4-0)(9)-4)(8)-2)(7)-6(5)|_{11}{=}5$

  $X_n{=}(0 \times 1) + (4 \times 5) + (2 \times 5 \times 7) + (7 \times 5 \times 7 \times 8) + (5 \times 5 \times 7 \times 8 \times 9)$

  $X_n{=}13985 > 280$ (error detected)

- Step 3 is executed since $X_n{>}M_a$ where $X_z$ is calculated using Equations 1–3. The integers as shown in Table 1 are obtained.

**Table 1. Result of correcting single erroneous residue.**

| Iteration, $c$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Discarded Residue, $x_c$ | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ |
| Recovered Data, $X_{z_c}$ | 2897 | 2105 | 125 | 1665 | 1385 |

- Since $X_{z_3}{<}280$, the correct data, $X{=}125$ is recovered.

Note that the number of discarded residues is equivalent to error correction capability $t$, where in this example it is one residue. For RRNS with higher error correction capability, e.g., $t{=}3$, each iteration will discard three residues in recovering the correct data and the maximum number of iteration will be $C_3^9{=}15$.

## 4. Modified Redundant Residue Number Systems Codes

*Conventional RRNS code* termed as C-RRNS is based on three restricted non-redundant moduli $m_a{=}\{2^f{,}2^f{-}1{,}2^f{+}1\}$ where $f$ is a positive integer. Unrestricted $m_b$ are commonly appended to the $m_a$ in generating a RRNS codeword. The $m_b$ are selected from any number of pairwise relatively prime positive integer. The value of integers for $m_b$ must be larger than that of $m_a$. In order to represent all numbers in the legitimate range of 16 bits word $[0, 2^{16}{-}1]$, the smallest $f{=}6$ is chosen to produce $m_a{=}\{64, 63, 65\}$ and $M_a{=}\prod_{a=1}^{3} m_a{=}262080$. To ensure the three residues dataword can be corrected if corrupted, six $m_b{=}\{67, 71, 73, 79, 83, 89\}$ are arbitrarily chosen, which clearly generate $M_b$ larger than the legitimate range. Hence, the codeword is now consists of $k{+}(n-k){=}3{+}6{=}9$ residues and possess $t{\leq}\lfloor\frac{n-k}{2}\rfloor{\leq}\lfloor\frac{9-3}{2}\rfloor{\leq}3$.

**Table 2. Moduli sets used by C-RRNS, 3NRM-RRNS, and 2NRM-RRNS codes.**

| ECC | Moduli | |
|---|---|---|
| Types | Non-Redundant, $m_k$ | Redundant, $m_{(n-k)}$ |
| C-RRNS | $\{2^f, 2^f-1, 2^f+1\}_{f=6} = \{64,63,65\}$ | $\{67,71,73,79,83,89\}$ |
| 3NRM-RRNS | $\{2^f, 2^f-1, 2^f+1\}_{f=6} = \{64,63,65\}$ | $\{31,29,23,19,17,11\}$ |
| 2NRM-RRNS | $\{2^f+1, 2^f\}_{f=8} = \{257,256\}$ | $\{61,59,55,53\}$ |

Because the redundant moduli consist of integers with larger values than that of non-redundant moduli, the C-RRNS code produces longer codeword bit length than RS code. To reduce the length of the codeword while providing high error correction capability, two modified RRNS codes are proposed in this work. The idea behind this is to use integers with smaller value as the redundant moduli, provided that the product of the moduli is larger than the legitimate range (see Section 3).

### 4.1. Three Non-Redundant Moduli RRNS (3NRM-RRNS) Code.

The 3NRM-RRNS code uses the same three restricted non-redundant moduli $m_a$=$\{64, 63, 65\}$ as the C-RRNS code. Nevertheless, the unrestricted redundant moduli $m_b$ for 3NRM-RRNS are based on integers with smaller value that of $m_a$ as shown in Table 2. To ensure the three residues dataword can be corrected if corrupted, six arbitrary integers $m_b$=$\{31, 29, 23, 19, 17, 11\}$ are chosen. Similar to C-RRNS, the codeword for 2NRM-RRNS comprises of nine residues, which contribute to $t \leq 3$. These integers are the minimum values that fulfill the first and third rule of generating RRNS explained in Section 3; they are co-prime to each other and their product $M_b = \prod_{b=1}^{6} m_b = 73465381 > 2^{16}-1$. Nonetheless, the use of $m_b$ with smaller value that that of $m_a$ violates the second rule, but this violation can be resolved by using *maximum likelihood decoding (MLD)* method proposed in [16] (explain later in Subsection 4.3).

### 4.2. Two Non-Redundant Moduli RRNS (2NRM-RRNS) Code.

The 2NRM-RRNS code utilizes two restricted non-redundant moduli $m_a$=$\{2^f$+1,$2^f\}$ instead of three $m_a$ as in C-RRNS and 3NRM-RRNS (see Table 2). For 16 bits data, the smallest $f$ satisfying the legitimate range is $f$=8, which produces $m_a$=$\{257, 256\}$ and $M_a = \prod_{a=1}^{2} m_a = 65792 > 2^{16}-1$. To ensure the two residues dataword can be corrected if corrupted, four arbitrary integers with smaller values than that of $m_a$ are selected for unrestricted moduli $m_b$. For this, $m_b$=$\{61, 59, 55, 53\}$ and its $M_b = \prod_{b=1}^{4} m_b = 10491085 > 2^{16}-1$. Equivalent to 3NRM-RRNS, this code also fulfill the first and third rule of RRNS, but violates the second rule. The same MLD method is used to solve the violation.

### 4.3. Maximum Likelihood Decoding.

Some input data encoded into 2NRM-RRNS and 3NRM-RRNS may posses more than one value that fall within the legitimate value. This is because the modified RRNS codes violate the second rule in generating residue number system (see Section 3). Nevertheless, this ambiguity is resolved by using maximum likelihood decoding method as proposed in [16]. The ambiguous numbers are re-calculated by performing modulo operation to the chosen moduli. The resulted residues are then compared to the read residues. The ambiguous number, which residues have the smallest difference (Hamming distance) to the read residues is regarded as the valid data.

## 5. Experimental and Analysis

To validate the capability of correcting errors both modified RRNS codes were compared to RS and conventional RRNS codes. The comparison between RRNS and RS codes is relevant because both codes are designed for correcting cluster errors and they have equal error correction capability.

## 5.1. Simulation Setup

Because 16 bits word memory data is considered in this work, the valid data (i.e., legitimate range) is between [0,$2^{16}$-1]. However, as mentioned in Section 3.2, $M_a$ and $M_z$ are used as references to determine the occurrence of error in a codeword. In this work $M_a$ and $M_z$ have bigger number than the valid data and must be are replaced by $2^{16}-1$=65535. This adjustment is done to avoid decoded data that larger than 65535 to be falsely decoded. E.g., for 3NRM-RRNS codes if $M_a$ is taken as the reference, invalid decoded data from 65536 to 262079 will be considered as valid data.

The RRNS variants and RS codes, 4K$\times$16-bit memory, and fault injection were described using MATLAB script. All codes were set to the corresponding $t$ to protect the dataword from transient faults. For RRNS decoding process, MRC was chosen because the algorithm uses smaller number value as compared to CRT to alleviate long simulation time. For RS code, built-in RS encoding and decoding MATLAB functions were used [17]. An appropriate adjustment in polynomial generator was done to encode and decode 16 bits word memory.

Faults were uniformly injected to the memories where the number of fault were increased from single bits to 20 bits faults per codeword. These faults flip the impacted memory bits from 0 to 1 and vice versa. Various fault rates were set from 1% up to 10% of the total memory capacity.

## 5.2. Simulation Results

Table 3 shows the total number of bits that represent the codeword for each code. RS, C-RRNS, 3NRM-RRNS, and 2NRM-RRNS are represented by $48, 61, 48$, and $41$ bits. These numbers are the total number of bits required to represent dataword and checkword. E.g., the 2NRM-RRNS requires $g$=9+8+6+6+6+6=41 bits. C-RRNS produces the longest codeword, whereas 2NRM-RRNS results in the shortest codeword among all codes. C-RRNS, 3NRM-RRNS, and 2NRM-RRNS produce bit length difference of 27.1% longer, equal, and 14.6% shorter, respectively, when compare to RS. 3NRM-RRNS and 2NRM-RRNS are shorter by 21.3% and 32.8%, each, when compared to C-RRNS.

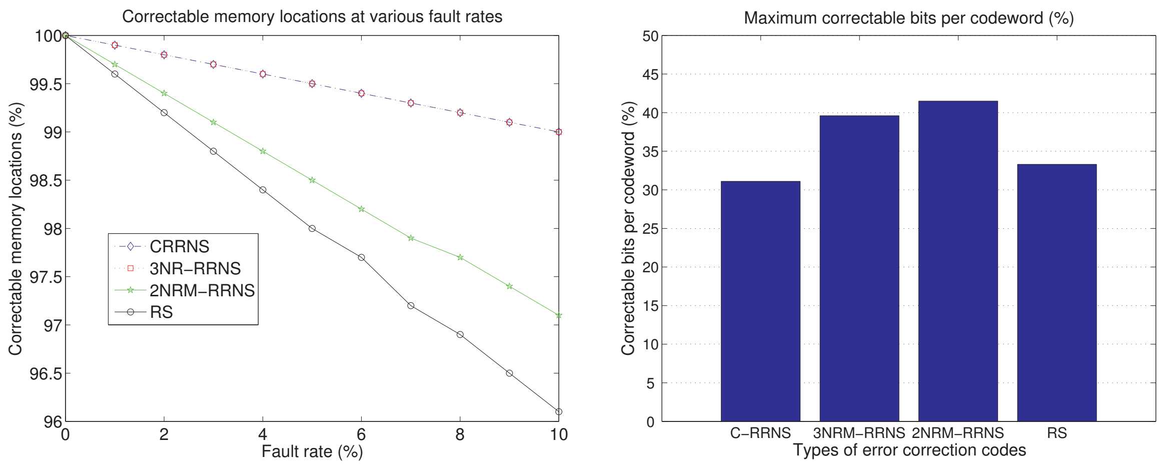
**Table 3. 16 bits word Reed Solomon and Redundant Residue Number Systems variant codes.**

| ECC | Number of bits | | | Differences (%) | |
|---|---|---|---|---|---|
| Types | Dataword | Checkword | Codeword | Compared to RS | Compared to C-RRNS |
| RS | 4,4,4,4 | 4,4,4,4,4,4,4,4 | 48 | – | -21.3 |
| C-RRNS | 6,6,7 | 7,7,7,7,7,7 | 61 | +27.1 | – |
| 3NRM-RRNS | 6,6,7 | 5,5,5,5,5,4 | 48 | 0 | -21.3 |
| 2NRM-RRNS | 9,8 | 6,6,6,6 | 41 | -14.6 | -32.8 |

Fig. 4(a) shows the simulation results for the RRNS variant and RS. Overall, the correction capability of all codes reduces as the fault rates become higher. However, all considered codes are able to protect more than 96% of the memory locations for at 10% fault rate. All three RRNS variants perform better than RS in correcting faults for all fault rates. E.g., at 10% fault rate, C-RRNS and 3NRM-RRNS are able to protect highest number of memory locations, which is 99% follow by 2NRM-RRNS (97.1%) and RS (96.1%).

## 5.3. Analysis

Although C-RRNS and 3NRM-RRNS are able to correct the largest memory locations as shown in Fig. 4(a), 2NRM-RRNS is the best in certain points of view. These include (i) correction capability-per-generated codeword length, (ii) correction capability-per-fixed codeword length, (iii)



(a) Correctable memory locations at various fault rates (b) Maximum correctable bits per codeword

**Figure 4. Simulation result and analysis**

data memory size-per-fixed data storage capacity, and (iv) decoding time complexity.

First, with regard to the correction capability-per-generated codeword length. For 16 bits input data encoded into 2NRM-RRNS the maximum number of erroneous bits it can correct is 17 out of 41 bits. This means that 17/41=41.5% of each codeword will be corrected as shown in Fig. 4(b). However, 3NRM-RRNS, RS, and C-RRNS possess only 19/48=39.6%, 16/48=33.3%, and 19/61=31.1%, respectively. Thus, 2NRM-RRNS provides the best correction capability follow by 3NRM-RRNS, RS, and C-RRNS.

Second, with regard to correction capability-per-fixed codeword length. Taking the codeword length of 2NRM-RRNS as a reference, RS and C-RRNS will be encoded into 40 bits (i.e., by taking off the last two- and three residues checkword, respectively), while 3NRM-RRNS will be represented by 39 bits (i.e., by removing the last two residues checkword). These reductions decrease the error correction capability to $t \leq \lfloor \frac{10-4}{2} \rfloor \leq 3$ for RS, $t \leq \lfloor \frac{6-3}{2} \rfloor \leq 1$ for C-RRNS, and $t \leq \lfloor \frac{7-3}{2} \rfloor \leq 2$ for 3NRM-RRNS codes. The maximum 16 bits word memory data that can be corrected by RS is now $4+4+4=12$ bits ($t \times$number of bits), by C-RRNS is 7 bits, and by 3NRM-RRNS is $6+7=13$ bits. Yet, 2NRM-RRNS can still protect $9+8=17$ bits by the four residues checkword. Hence, 2NRM-RRNS is able to offer the highest correction capability follow by 3NRM-RRNS, RS, and C-RRNS.

Third, with regard to memory chip size for fixed data storage capacity where as reported in [1] 1Tbit hybrid memory can be fabricated in a centimeter square, assuming that no ECC capability is used. The same data capacity encoded into 2NRM-RRNS will result in $41/16=2.6\times$ larger than the original size. Nevertheless, even larger memory size will be required when input data is encoded into 3NRM-RRNS, RS, and C-RRNS; each by $3\times$, $3\times$, and $3.8\times$. Thus, 2NRM-RRNS results in smallest memory size (or provides better data storage) follow by 3NRM-RRNS, RS, and C-RRNS.

Fourth, with regard to decoding time for RRNS codes. Since 2NRM-RRNS holds at most two error correction capability the code requires $C_2^6=15$ iterations to recover a correct data (i.e., $C_t^n$ rule in Section 3.2). However, for C-RRNS and 3NRM-RRNS the procedure needs $C_3^9=84$ iterations. Therefore, 2NRM-RRNS is 5.6 times faster than C-RRNS and 3NRM-RRNS, which in turn results in higher speed decoding operation.

## 6. Conclusion

In this paper two modified Redundant Residue Number Systems codes have been proposed to tolerate high rate of cluster faults in hybrid memories. The modified RRNS codes, referred to as Three Non-Redundant Moduli RRNS (3NRM-RRNS) and Two Non-Redundant Moduli RRNS (2NRM-RRNS), are based on three and two non-redundant moduli sets, respectively. On the contrary to conventional RRNS (C-RRNS), the redundant moduli for 3NRM-RRNS and 2NRM-RRNS consist of integers with value smaller than that of non-redundant moduli. The simulation results for 16 bits word memory show that both modified RRNS codes have competitive error correction capability to that of Reed-Solomon (RS) and C-RRNS, but at reduced code size and lower performance penalty. This work has proved that the modified RRNS codes are able to provide sufficient error correction capability for reliability improvement in hybrid memories. Future work is to further reduce the bits that represent a RRNS codeword to realize lower cost.

## References

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

[2] A. DeHon, S. C. Goldstein, P. Kuekes, and P. Lincoln, “Nonphotolithographic nanoscale memory density prospects”, *IEEE Transactions on Nanotechnology*, vol. 4, no. 2, pp: 215–228, March 2005.

[3] 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, no. 12, pp. 1349–1352, Dec 2008.

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

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

[6] C.M. Jeffery, A. Basagalar, and R. J. O Figueiredo, “Dynamic sparing and error correcting techniques for fault tolerance in nanoscale memory structures”, *Proceedings IEEE Conference on Nanotechnology*, pp. 168–170, Aug. 2004.

[7] S. Biswas, T. S. Metodi, F. T. Chong, and R. Kastner, “Combining static and dynamic defect-tolerance techniques for nanoscale memory systems”, *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*, pp. 773–778, 2007.

[8] F. Sun and T. Zhang, “Defect and Transient Fault Tolerant System Design for Hybrid CMOS/Nanodevice Digital Memories”, *IEEE Transactions on Nanotechnology*, vol. 6, no. 3, pp. 341-351, May 2007.

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

[10] H. Naeimi and A. DeHon, “Fault Tolerant Nano-Memory with Fault Secure Encoder”, *Proceedings of International Conference on Nano-Networks*, Sept. 2007.

[11] S. Ghosh and P.D. Lincoln, “Dynamic Low-Density Parity Check Codes for Fault-tolerant Nanoscale Memory Fault-tolerant Nanoscale Memory”, *Proceedings of Foundation of Nanoscience*, April 2007.

[12] K. K. Likharev, “Hybrid CMOS/Nanoelectronic Circuits: Opportunities and Challenges”, *Journal of Nanoelectronics and Optoelectronics*, vol. 3, pp. 203–230, 2008.

[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 Transactions on Circuits and Systems*, pp. vol. 39, 18–34, Jan 1992.

[14] 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.

[15] L. Yang and Lajos Hanzo, “Redundant Residue Number System Based Error Correction Codes”, *Proceedings of IEEE Vehicular Technology Conference*, pp. 1472–1476, Oct 2001.

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

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