High-Performance Cluster-Fault Tolerance Scheme for Hybrid Nanoelectronic Memories

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

***Abstract*—Error correction codes (ECCs) are common industrial practices for tolerating intermittent and transient faults in semiconductor memories. They have been also proposed for emerging hybrid nanoelectronic memories. However, this solution comes at higher cost in terms of performance penalty and area overhead. This paper proposes a high performance cluster-fault correction scheme for hybrid nanoelectronic memories. The scheme, referred to as Double Three-Residue Code (D3R), is based on combining the advantages of N-tuple Modular Redundancy (NMR) and Redundant Residue Number System (RRNS). Experimental results show that D3R decodes 10 to 28 times faster as compared to RRNS variants and Reed-Solomon (RS) while achieving similar cluster-fault correction capability.**

Keywords-hybrid nanoelectronic memories; fault tolerance; N-tuple modular redundancy; redundant residue number system

I. Introduction

Hybrid technology, which combines CMOS and non-CMOS, is seen as a near-term alternative to existing CMOS technology. One of the earliest and the simplest products of this technology are *hybrid nanoelectronic memories* (hereafter referred to as hybrid memories). These memories are structured by integrating non-CMOS nanodevices memory cells array on top of scaled CMOS peripheral circuits. Various research teams, both from academia [1]–[6] and industries [7]–[9], have been extensively building up their hybrid memory prototypes. Such memories are anticipated to offer huge data storage capacity (up to 1Tbit per cm^2) at lower power consumption. However, these benefits do not come for free. Imprecise top-down fabrication techniques cause latent defects in scaled CMOS circuits, whereas immature bottom-up fabrication techniques introduce latent defects in the non-CMOS circuits. These latent defects, when activated by non-environmental disturbances, might induce intermittent faults. Moreover, the lower voltage at which these devices operate is subject to transient faults (e.g., sensitivity to soft errors and radiations). Therefore, before hybrid memories can be massively produced and commercialized, these critical issues must be resolved. Besides improving material and process development, circuit design technique that can tolerate the issues is a solution to this problem.

Fault-tolerant design techniques applied for hybrid memories have been addressed by many authors [1], [11]–[17]; schemes like error correction codes (ECCs), sparing and reconfiguration have been investigated. Among these schemes, ECCs are the most frequent used due to their dynamic operation, i.e., this scheme operates based on two modes, either (i) detection or (ii) detection followed by correction. In case the read data is error-free, ECCs only perform detection. On the other hand, in case detection has discovered errors in the read data the scheme needs to perform correction. Compared to detection, correction requires more computation power that cause greater performance penalty. Examples of ECCs proposed for hybrid memories are Hamming [1], [12], [13], Bose-Chaudhuri-Hocquenghem (BCH) [10], [11], Low-Density Parity-Check (LDPC) [14], [15] and Redundant Residue Number System (RRNS) code [16], [17]. However, the major drawback of such solutions is the associated high cost either in performance penalty and/or area overhead.

This paper presents a high-performance fault tolerance scheme able to tolerate intermittent and transient cluster faults in hybrid memories. Two established fault tolerance schemes are combined, namely N-tuple Modular Redundancy (NMR) and RRNS code, creating the combined fault tolerance scheme referred to as *Double Three-Residue (D3R)* code. A three-residue RRNS codeword encoded from an input data is replicated once to have another copy of codeword. These two copies of codeword are compared simultaneously during decoding. Experimental results exhibit that D3R provides better time performance besides competitive correction capability and area overhead (which measured in user capacity) as compared to the related ECCs investigated in [16], [17].

The rest of the paper is organized as follows. Section II reviews the basic theory of fault tolerance including RRNS code and N-tuple Modular Redundancy. Section III describes the structure as well as the encoding and decoding procedures of the proposed Double Three-Residue scheme (D3R). Section IV presents experimental results and a comparison to the related scheme. Section V concludes the paper.

II. Overview of Fault Tolerance

Fault tolerance is the capability of a system to continue operating accordingly in the event of the failure of some of its components. This capability is attained by introducing redundancy to its components. Redundancy implies replication and an effective fault tolerance depends on the usefulness of the redundant elements. Redundancy can be implemented through the use of *hardware*, *information* or *time* as depicted in Fig. 1. Two sub-schemes belong to hardware redundancy, namely *N-tuple Modular Redundancy (NMR)* and *interwoven logic (IL)*. Two sub-schemes belong to time redundancy, namely *Recomputing With Delay Logic (REDL)* and *Recomputing With Shifted Operand (RESO)*. A sub-scheme belongs to information redundancy, namely *Error Correction Codes (ECCs)*. ECCs address two types of fault distributions: *cluster* and *random*. Cluster faults are suitable to be mitigated using either Redundant Residue Number System (RRNS) or Reed-Solomon (RS) codes [18]–[21]. Random faults are appropriate to be tolerated using either Hamming, Bose-Chaudhuri-Hocquenghem (BCH) or Low-Density Parity-Check (LDPC) [18].

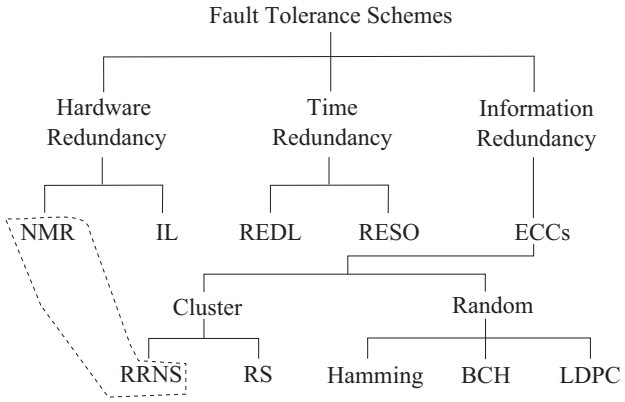

Figure 1. Classification of fault tolerance schemes and ECCs.

As mentioned earlier, two different fault tolerance schemes are combined in this work. These schemes, as highlighted by dotted lines in Fig. 1, are combined to achieve a high-performance cluster-fault tolerance scheme for hybrid memories. The next two subsections discuss the fundamental concept of these two fault tolerance schemes.

A. N-tuple Modular Redundancy

N-tuple Modular Redundancy (NMR) is N copies of identical circuit execute in parallel for which their correct output is defined by a comparator or voting circuit. Duplicated Modular Redundancy (DMR) consists of two identical circuits operate the same function simultaneously as shown in Fig. 2(a). The output from the original circuits and its duplicate are then compared using XOR gates. Any disagreement resulted from the comparison indicates faults have occurred in the circuits.

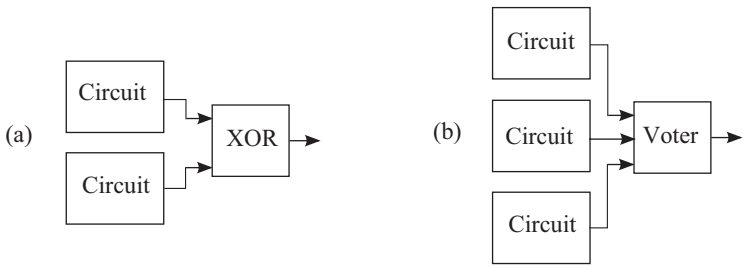

Figure 2. NMR variations (a) DMR (b) TMR.

Triple Modular Redundancy (TMR) consists of three identical circuits operate the same function simultaneously as illustrated in Fig. 2(b). In contrast to the simple XOR gates in DMR, TMR uses majority voting circuit to determine the correct outputs of the circuits. TMR can mask faults occurred in one of the triplicated circuit if another two replicas are fault-free.

The advantages of NMR are as follows:

- it executes in parallel allowing high performance operation.

- it masks fault in one of the copies (with sufficient replication).
- it realizes simple hardware implementation.

B. Redundant Residue Number System Code

Redundant Residue Number System (RRNS) code is devised from Residue Number System (RNS), which have been extensively investigated and applied for communication and high-speed arithmetic applications [19]–[21]. RRNS code comprises of two sets of symbol-oriented encoded data called *residues* x_i, where $1 \leq i \leq n$ and n is positive integer, as shown in Fig. 3. The first set is called as *non-redundant residues* that represent dataword (input data), whereas the second set is called as *redundant residues* that represent checkword (for fault detection and correction). The non-redundant residues consist of k residues, whereas the redundant residues consist of $(n–k)$ residues; k and n are positive integer. This code has fault detection capability defined as $u=n–k$ and fault correction capability equated as $t=\frac{u}{2}$. E.g., one residue and two residues must be appended as the checkword to detect and correct single residue, respectively.

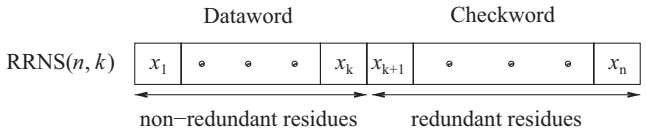

Figure 3. RRNS code structure.

Each residue in RRNS code is generated (encoded) by performing modulo operation of an input data X to a set of moduli m_i, i.e., $x_i=|X|_{m_i}$ where $1 \leq i \leq n$. The bit length of each residue is defined as $b_i=\lceil log_2(m_i) \rceil$ bits; thus, an RRNS code has the bit length equal to $\sum_{i=1}^{n} b_i$. RRNS decoding performs *detection* followed by *correction*, if faults are detected in the read data. Detection validates a read RRNS codewords, whereas correction recovers the detected erroneous RRNS codewords and produce the correct output data. Error-free read data is read out of the memory without requires any correction if it is within the legitimate range. Otherwise, an iterative operation is executed to correct the erroneous read data for a maximum of $C_t^n=\frac{n!}{t!(n-t)!}$ times. During this phase, t number of residues are discarded in each iteration and the calculation of $(n-t)$ number of residues (and their corresponding parameters) is performed to recover a data within the legitimate range. Any data falls within the legitimate range is regarded as the valid data and is read out of the memory. However, if no data is corrected (i.e., all data are beyond the legitimate range) after the maximum iteration, an uncorrectable signal will be flagged to indicate that the decoder cannot correct the erroneous data.

The advantages of RRNS code are as follows [19]–[21]:

- it possesses cluster faults correction capability.
- it can be generated based on the *low-cost moduli*, which render to small area overhead and fast operation.
- its encoder and decoder sub-units consist of modular circuitries that operate in parallel; thus, speeds up the performance.

III. Proposed D3R Code

This section describes the structure, the encoding and the decoding process of the proposed D3R scheme. As aforementioned this scheme combines the concept and advantages of NMR and RRNS schemes; hence, it realizes a synergistic high-performance error correction scheme.

A. D3R Code Structure

As illustrated in Fig. 4, a Double Three-Residue (D3R) code consists of an RRNS codeword ($C=DW+CW$) and its duplicate ($C'=DW'+CW'$). The D3R code is encoded based on the moduli set $\{2^{\frac{d}{2}}-1,\ 2^{\frac{d}{2}+1}-1,\ 2^{\frac{d}{2}+1}\}$ where d is the width of the memory word. Three moduli set is used to ensure the detection of a single faulty residue, i.e., $u=n–k=3–2=1$. Moreover, the moduli set is in the form of *low-cost moduli*, which facilitate in improving the performance [20]. The first two moduli are employed to generate two-residue dataword DW and its duplicate DW', while the third modulus is used to produce the checkword CW and its duplicate CW'. Table I gives the moduli sets for 16, 32 and 64-bit memory word for this code. The bit length of dataword and checkword of D3R code is $b_{D3R}=2\times(\lceil log_2(m_1) \rceil+\lceil log_2(m_2) \rceil+\lceil log_3(m_3) \rceil)$ where m_1, m_2 and m_3 are the moduli used.

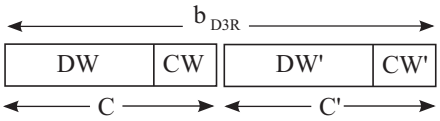

Figure 4. D3R code structure.

Table I
MODULI SET FOR D3R CODE.

Memory Word Width, d	Moduli		
	m_1	m_2	m_3
16	255	511	512
32	65535	131071	131072
64	4294967295	8589934591	8589934592

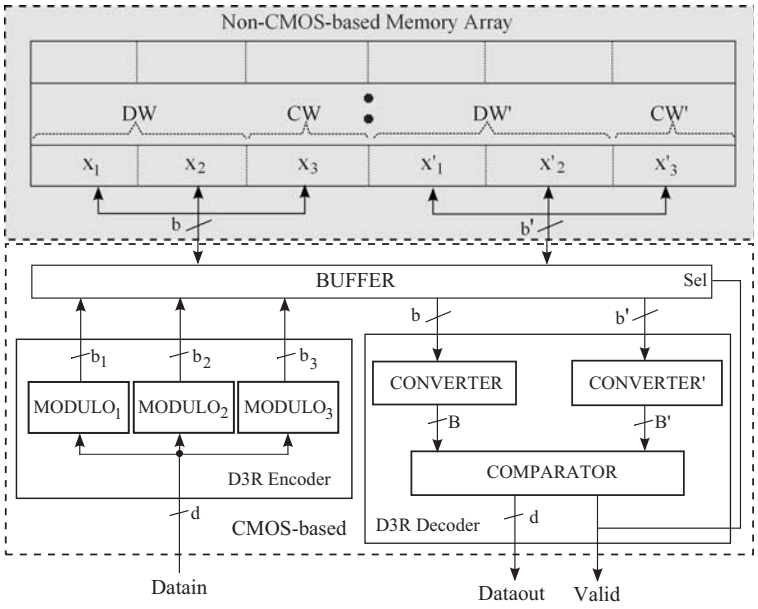

Figure 5. Block diagram of D3R encoder and decoder in a hybrid memory.

B. D3R Code Encoding

The bottom left part of Fig. 5 depicts the block diagram of a D3R encoder constructed of three modulo circuits. These three modulo circuits receive a d-bit input word and generate the corresponding residues *simultaneously*. $MODULO_1$ and $MODULO_2$ generate two residues x_1 and x_2, respectively, which then form $DW{=}x_1, x_2$. Their duplicates, i.e, x'_1 and x'_2 form $DW'{=}x'_1, x'_2$. At the same time, $MODULO_3$ generates a single-residue checkword $CW{=}x_3$ (and its duplicate $CW'{=}\ x'_3$). Thereafter, D3R codewords ($C{=}x_1,x_2,x_3$ and $C'{=}x'_1,x'_2,x'_3$) are stored in the memory array. By using the equation in Section III-A, the bit length of D3R codeword can be determined. E.g. for 64-bit memory word, each residue has bit length b_1=32 bits, b_2=33 bits and b_3=33 bits. Thus, a 64-bit memory word encoded into D3R code has bit length b_{D3R}=196 bits.

C. D3R Code Decoding

The bottom right part of Fig. 5 illustrates the block diagram of D3R decoder. The decoder consists of two main circuits: *converter* and *comparator*. The first circuit is formed by two RNS-to-binary converters, denoted as *CONVERTER* and *CONVERTER'*, which operate in parallel. *CONVERTER* converts $C{=}x_1,x_2,x_3$ into a binary word B, while *CONVERTER'* converts $C'{=}x'_1,x'_2,x'_3$ into a binary word B'. The binary words B and B' are fit into a comparator; they are compared to the legitimate range $LR{=}2^d{-}1$ to determines the validity of the read codeword. If no fault occurred, then $B{=}B'$ will be within the legitimate range. If the fault affected one of the codewords, then either B or B' will be out of the legitimate range. The *Sel* signal in the figure selects the swapping iteration, which is one of the important steps in the decoding process as will discussed next.

Table II
RESIDUE SET FOR EACH ITERATION OF D3R CORRECTION PHASE.

Iteration	Residues	
	C	C'
1	x'_1, x_2, x_3	x_1, x'_2, x'_3
2	x_1, x'_2, x_3	x'_1, x_2, x'_3
3	x_1, x_2, x'_3	x'_1, x'_2, x_3

As mentioned in Section II-B, RRNS decoding discards t number of residues during the correction phase (where the remaining $(n-t)$ residues are used to compare with the legitimate range). This operation repeats for a maximum of $C_t^n=\frac{n!}{t!(n-t)!}$ times; thus, it impacts the performance of the decoder. However, this is not the case for D3R where it does not discard any residues during correction phase but swaps between the two datawords. The maximum number of swapping is equal to the number of symbols in a codeword, i.e., n. It works as follows:

Step 1: Convert C=x_1,x_2,x_3 and C'=x'_1,x'_2,x'_3 into binary data B_1 and B'_1, each;

Step 2: Compare B and B' to a predefined legitimate range LR=2^d-1;

Step 3: If (B=B')<LR or B<LR, read out B. If B'<LR, read out B'. Break, else go to Step 4;

Step 4: Check if the maximum number of swapping has been reached. If "yes" go to Step 6, else go to Step 5;

Step 5: Swap one of the residue between C and C'. Go to Step 1;

Step 6: Ignore the output data and invoke the uncorrectable control signal.

Table II exhibits the residue sets after the swapping process for each iteration. E.g., the first residues between C and C' are swapped during the first iteration in such a way that x_1 is part of C' and x'_1 is part of C. A similar step is performed to the second and third residues in the subsequent iterations. The order to select the residue to be swapped is random, i.e., no fixed rule is set.

Step 1 performs a conversion of D3R code into binary numbers. This conversion can be accomplished using either *Mixed Radix Conversion (MRC)* or *Chinese Remainder Theorem (CRT)* [20]. In this work, MRC is used as it deals with smaller integer; hence, it speeding up simulation work. MRC is based on the following equation [20].

$$X = x_1 + \sum_{i=2}^{n} v_i \prod_{j=1}^{n-1} m_j \tag{1}$$

where x_1 is the first residue, m_j are the moduli and v_i are mixed radix digits calculated as

$$v_i = \Big|\big(((x_i - v_1) \times g_{1i}) ..... - v_{(i-1)}\big) \times g_{(i-1)i}\Big|_{m_i} \tag{2}$$

where x_i=$|X|_{m_i}$ and $g_{(i-u)i}$ are the multiplicative inverses of $m_{(i-u)}$ with respect to m_i defined as $|m_{(i-u)}g_{(i-u)}|_{m_i}$=1; $2\leq i\leq n$ and $1\leq u\leq n-1$.

IV. EXPERIMENTAL EVALUATION AND ANALYSIS

This section presents the experimental evaluation and analysis of the proposed and existing considered ECCs. Four attributes are compared among the schemes, namely error correction performance, decoding time performance, codeword length and hardware overhead.

A. Simulation Setup

The encoder and decoder of the considered ECCs (D3R, RS and RRNS variants [16]), memories as well as the fault injection were described using MATLAB script. C-RRNS code is based on moduli set $\{2^p-1, 2^p, 2^p+1, a_1, a_2, ..., a_6\}$ where p=6, 11, and 22 for 16, 32, and 64-bit memory word, respectively, whereas a_1 to a_6 are arbitrary integers that satisfy RRNS rules (see Section II-B). To protect the three-residue dataword, the error correction capability of C-RRNS is set to t=$\frac{(n-k)}{2}$=$\frac{9-3}{2}$=3. 6M-RRNS code is based on moduli set $\{2^p, 2^p+1, 2^{p-1}-1, 2^{p-2}-1, 2^{p-3}-1, 2^{p-4}+1\}$ where p=8, 16, 32 for 16, 32, and 64-bit memory word, respectively [16]. To protect the two-residue dataword, its error correction capability is set to t=$\frac{(n-k)}{2}$=$\frac{6-2}{2}$=2.

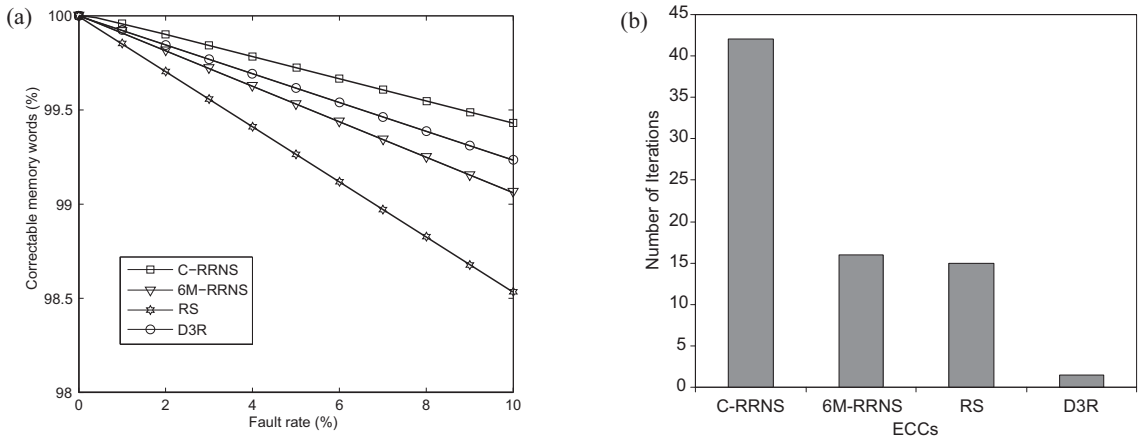

Figure 6. Performance of (a) error correction capability (b) decoding iterations.

For RS code, Galois Field of degree eight denoted as GF(2^8) is used where each symbol consists of an 8-bit data. An appropriate number of symbols is set to have similar correction capability to that of D3R and RRNS variants. E.g., a 16-bit input word is encoded into six symbol (two symbols for dataword and four symbols for checkword) to realize t=2.

Varied-length of cluster faults were randomly injected to the memory depending on fault rate. The cluster faults can impact either single, two, three or four residues/symbols that form the codeword. Various fault rates from 1% to 10% were applied during the experiments.

B. Error Correction Performance

Figure 6(a) gives the error correction performance represented by the percentage of correctable memory words for 64-bit word memory for all considered ECCs. Regardless of the fault rates, C-RRNS provides the best correction performance among the considered ECCs followed by D3R, 6M-RRNS and RS. However, the difference between C-RRNS and D3R is negligible, e.g., at 10% fault rate it is only 0.2%. Moreover, the larger the memory word, the smaller the difference; e.g., for 256-bit word memory (not in the figure) the difference is less than 0.1%.

Based on the set parameters, the comparison of error correction performance among the ECCs is given as follows:

- D3R - it ensures up to three erroneous residues are correctable *if faults corrupt one dataword copy C while another copy C' is fault-free or vice-versa*. This is also applied to any combination of corrupted residues shown in Table II. However, if a residue and its copy, let say x_1 and x_1' are corrupted, D3R cannot correct them even though the other four residues are fault-free. In general, D3R can correct up to $t{+}1{=}\frac{n-k}{2}{+}1$ residues.
- C-RRNS - it always ensures up to three erroneous residues are correctable *regardless of which residues in the codeword are corrupted*. Moreover, its varied-length residues are longer than that of the other three ECCs; hence, they assist in providing the highest error correction capability. In general, C-RRNS can correct up to $t{=}\frac{n-k}{2}$ residues.
- 6M-RRNS - it always ensures up to two erroneous residues are correctable *regardless of which residues in the codeword are corrupted*. Furthermore, the extra likelihood decoding step helps in recovering some fault cases [16]. In general, 6M-RRNS can correct up to $t{=}\frac{n-k}{2}$ residues.
- RS - it always ensures up to two symbols are correctable *regardless of which symbols in the codeword are corrupted.* Nevertheless, its fixed-length symbols are shorter than that of the other three ECCs; thus, they result in the lowest error correction performance. In general, RS can correct up to $t{=}\frac{n-k}{2}$ symbols.

Therefore, D3R is able to provide similar or better fault correction performance than the conventional ECCs like RRNS and RS. More interestingly, for the considered example, D3R is able to correct one residue more than theoretical error correction capability of RRNS for certain cases.

C. Decoding Time Performance

Based on the set parameters, the comparison of the decoding performance among the considered ECCs is given as follows (see Fig. 6(b)):

- D3R - to protect its two-residue dataword, D3R requires n=6 that are swapped during the detection phase (see Table II); therefore, the maximum iteration is 3 iterations. The iteration stops when the fault has been detected and the correct

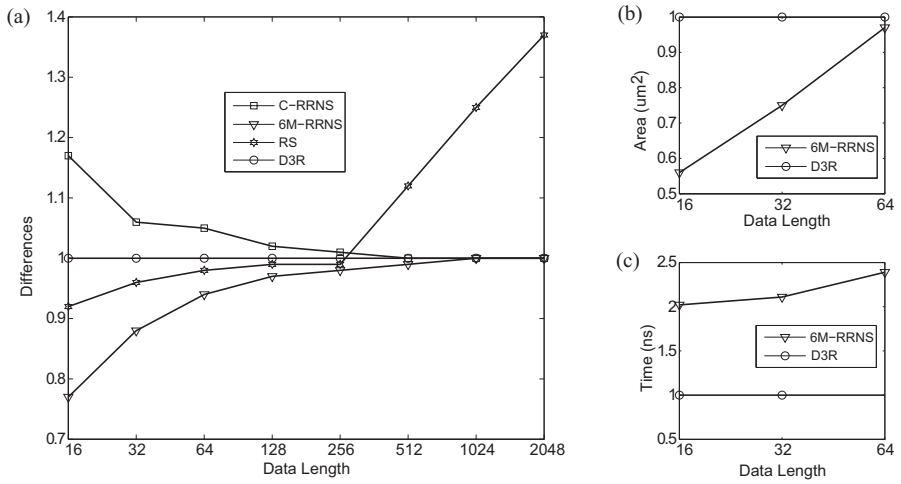

Figure 7. ECCs' (a) codeword length (b) area overhead (c) time overhead.

value is determined; so, on average D3R requires 1.5 iterations to produce the correct output data. In general, D3R iterates for a maximum of $\frac{n}{2}$ times.

- C-RRNS - to protect its three-residue dataword, C-RRNS requires n=9 residues; hence, the maximum of iteration of this code is $C_3^9=\frac{9!}{3!(9-3)!}$=84 iterations. As in D3R, C-RRNS stops when it has recovered the correct output data; thus, on average this code requires 42 iterations. In general, C-RRNS iterates for a maximum of C_t^n times.
- 6M-RRNS - to protect its two-residue dataword, 6M-RRNS requires n=6 residues; thus, the maximum of iteration of this code is $C_2^6=\frac{6!}{2!(6-2)!}$=15 iterations. Moreover, this code must going through all iterations and an additional decoding likelihood step [16]. In general, C-RRNS iterates for a maximum of C_t^n+1 times.
- RS - to protect its two-symbol dataword, RS needs n=6 residues; therefore, the maximum of iteration of this code is $C_2^6=\frac{6!}{2!(6-2)!}$=15 iterations. In general, C-RRNS iterates for a maximum of C_t^n times.

Hence for the considered example, the proposed D3R decodes on the average $\frac{42}{1.5}\approx$28, $\frac{16}{1.5}\approx$10 and $\frac{15}{1.5}$=10 times faster than C-RRNS, 6M-RRNS and RS, respectively. It is worth to note that the higher the considered error correction capability, the higher the performance improvement of D3R as compared to the other schemes.

D. Codeword Length

Figure 7(a) shows the normalized differences in codeword length of the considered ECCs for memory word size up to 2048 bits. For this analysis, D3R is used as a reference, i.e., it is set to 1. E.g., for a memory word of 16 bits, the codeword length of 6M-RRNS and RS is 22% and 8% shorter than that of D3R, respectively; while that of C-RRNS is about 18% longer than that of D3R. It is interesting to note that for realistic word size for hybrid memories (typically $>$ 256 bits), the codeword length of D3R is almost the same as that of the other three ECCs. In addition, D3R outperforms RS as the memory size becomes larger. RS requires bigger degree of Galois Field to ensure the legitimate range, i.e, GF(2^9) for 512 bits, GF(2^{10}) for 1024 bits and GF(2^{11}) for 2048 bits. For RRNS variants and D3R (in fact it belongs to RRNS variants), larger moduli are required that satisfy the rules.

E. Hardware Overhead

Figure 7(b) gives the normalized total area overhead of the encoder and decoder for D3R and 6M-RRNS ECCs; again, D3R is set as the reference (i.e., 1) for this comparison. It shows that the difference of the area overhead between the two ECCs becomes insignificant as the data length increases. E.g. for 16-bit data, the encoder and decoder of 6M-RRNS is 45% smaller than D3R but for 64-bit data the difference is about 3%. D3R requires larger encoder and decoder because the parameters (e.g., moduli, multiplicative inverses) used are based on bigger integer (thus larger circuits) than that of 6M-RRNS. Figure 7(c) illustrates the normalized total time overhead of the encoder and decoder for D3R and 6M-RRNS. It shows that the encoder and decoder of D3R are faster than that of 6M-RRNS regardless of the data length. Moreover, their difference grows as the data length becomes greater. This synthesis results prove the decoding performance analysis given in Section IV-C.

Note that the area and time overhead for C-RRNS and RS are not considered here because the moduli for the former is based on arbitrary integers that require complex hardware, whereas the latter has different decoding scheme and hardware from D3R. Their hardware implementation will be considered as a future work.

V. Conclusion

This paper has presented a high-performance fault tolerance scheme targeting intermittent and transient cluster faults in hybrid memories. The fault tolerant scheme, referred to as Double Three-Residue (D3R) code, combines two established schemes namely (i) N-tuple Modular Redundancy and (ii) Redundant Residue Number System (RRNS) code. The advantages of parallel execution of NMR and cluster fault correction of RRNS code are exploited in realizing the D3R scheme. Experimental results and analysis show that D3R is 10 to 28 times faster that of RRNS variants and Reed Solomon while achieving competitive error correction capability. Hardware implementation of the encoder and decoder indicates that D3R incurs comparable area and better performance to that of one RRNS variant.

Currently, an on-line test scheme is being explored as another fault tolerance scheme to improve the reliability of hybrid memories. An assessment on the fault coverage and associated cost will be analyzed.

References

[1] D. B. Strukov and K. K. Likharev, "Prospects for Terabit-Scale Nanoelectronic Memories", Journal Nanoscience and Nanotechnology, vol. 16, 2005, pp. 137–148.

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

[3] A. DeHon, "Nonphotolithographic Nanoscale Memory Density Prospects", IEEE Transactions on Nanotechnology, vol. 4, no. 2, 2005, pp. 215–228.

[4] M. A. Reed, J. Chen, A. M. Rawlett, D. W. Price and J. M. Tour, "Molecular Random Access Memory Cell", Applied Physics Letters, vol. 78, no. 23, 2001, pp. 3735–3737.

[5] R. J. Luyken and F. Hofmann, "Concept for Hybrid CMOS-Molecular Non-Volatile Memories", Journal Nanoscience and Nanotechnology, vol. 14, 2003, pp. 273–276.

[6] C. K*ü*geler, M. Meier, R. Rosezin, S. Gilles and R. Waser, "High Density 3D Memory Architecture Based on the Resistive Switching Effect", Journal Solid-State Electronics, vol. 53, no. 12, 2009, pp. 1287–1292.

[7] Zettacore™, ZettaCore™memory, http://www.zettacore.com/

[8] California Molecular Electronics Corporation, (CALMEC®), Molecular Electronic Technology®. http://www.calmec.com/

[9] K. Bullis, "Ultradense Molecular Memory: Researchers Develop a Large-Scale Array of Nanoscale Memory Circuits", MIT Technology Review, available online at http://www.technologyreview.com/Nanotech/18100/

[10] D. B. Strukov and K. K. Likharev, "Defect-Tolerant Architectures for Nanoelectronics Crossbar Memories", Journal Nanoscience and Nanotechnology, vol. 7, 2007, pp. 151–167.

[11] 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, 2007, pp. 341–351.

[12] S. Biswas, T. S. Metodi, F. T. Chong and R. Kastner, "A Pageable, Defect-Tolerant Nanoscale Memory System", Proc. IEEE International Symposium on Nanoscale Architecture, 2007, pp. 85–92.

[13] C. M. Jeffery and R. J. O. Figueiredo, "Hierarchical Fault Tolerance for Nanoscale Memories", IEEE Transactions on Nanotechnology, vol. 5, no. 4, 2006, pp. 407–414.

[14] H. Naeimi and A. DeHon, "Fault Secure Encoder and Decoder for NanoMemory Applications", IEEE Transactions on Very Large Scale Integration Systems, vol. 17, no. 4, 2009, pp. 473–486.

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

[16] N. Z. Haron and S. Hamdioui, "Residue-based Code for Reliable Hybrid Memories", Proc. IEEE/ACM International Symposium on Nanoscale Architectures, 2009, pp. 27–32.

[17] N. Z. Haron and S. Hamdioui, "Using RRNS Codes for Cluster Faults Tolerance in Hybrid Memories", Proc. of IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems, 2009, pp. 85–93.

[18] S. Lin and D. J. Costello, Error Control Coding: Fundamentals and Applications, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 2004.

[19] L. Yang and L. Hanzo, "Coding Theory and Performance of Redundant Residue Number System Codes", available online at http://www-mobile.ecs.soton.ac.uk/

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

[21] F. Barsi and P. Maestrini, "Error Correcting Properties of Redundant Residue Number Systems", IEEE Transactions of Computers, vol. C-22, no. 3, pp. 307–315, 1973.