On Correcting Cluster Errors in Nanoelectronic Memories

Nor Zaidi Haron   Said Hamdioui

Delft University of Technology, Computer Engineering Laboratory
Mekelweg 4, 2628 CD, Delft, The Netherlands
{N.Z.B.Haron, S.Hamdioui}@tudelft.nl

Abstract. By combining non-CMOS devices and CMOS devices in a 3D chip, nanoelectronic memories are able to offer extraordinary scalability advantages. However, these extremely dense memories are prone to high magnitude of faults impacting their reliability. This paper presents a fault tolerance scheme, developed from Redundant Residue Number System (RRNS) error correction code, to correct cluster errors in nanoelectronic memories. Three variants of RRNS codes are optimized and experimentally compared with Reed-Solomon (RS) and conventional RRNS (C-RRNS) codes. Experimental results show that the optimized RRNS variants realize better error correction capability while providing larger data storage capacity and faster decoding speed.

1 Introduction

An emerging memory technology that uses non-CMOS devices to build up the memory cell array is extensively being studied [1]–[3]. Referred to as *nanoelectronic memory*, this memory offers extraordinary scaling capability resulting in larger data storage as compared to semiconductor memories. However, such memory is also likely to suffer from high permanent and non-permanent faults [4, 5]. Moreover, because the devices that structure the memories are incredibly tiny and fabricated very densely, such faults might affect adjacent cells (belonging to the same word in the memory cell array) causing *cluster errors*. Therefore, designing reliable nanoelectronic memories requires appropriate fault tolerance scheme.

Coding, such as error correction codes (ECCs), is one of the most known fault tolerance schemes used to improve the reliability of memory systems. In particular, ECCs proposed for nanoelectronic memories include Hamming [2], Bose-Chaudhuri-Hocquenghem (BCH) [6], Low-Density Parity-Check (LDPC) [7] and Redundant Residue Number System (RRNS) [8]. All these proposed ECCs, however, subject to high cost in performance penalty and/or area overhead.

This paper builds upon our previous work [8] that develop an optimized RRNS code for cluster errors correction in nanoelectronic memories. In this work we proposed two new optimized RRNS codes with smaller cost. We show that the optimized ECCs offer larger user data capacity and faster decoding, while providing a competitive error correction as compared to RS and C-RRNS. The rest of the paper is organized as follows. Section 2 reviews two ECCs used for comparison in this paper, i.e., RS and C-RRNS codes. Section 3 introduces the optimized RRNS variants. Section 4 compares the performance of the optimized RRNS variants with RS and C-RRNS codes. Section 5 concludes this paper.

2 Background

This section gives an overview of the appropriate conventional ECCs used for comparison in this work; i.e., RS and C-RRNS.

2.1 Reed-Solomon

Reed-Solomon code is formed by n-symbol codewords, each of which consists of k-symbol dataword and (n–k)-symbol checkword [10] where $n>k$, n and k are positive integers. RS code can correct up to t erroneous symbols in a codeword by appending a checkword of $2t$=(n–k) symbols. In this work *Galois Field* of degree eight (m=8) is used; it means that each RS symbol is represented by eight bits. A d-bit input data is encoded into $\frac{d}{m}$-symbol dataword. In order to ensure that $\frac{d}{m}$-dataword is correctable if corrupted, $2\times\frac{d}{m}$-symbol checkword is appended resulting into a codeword of $3\times\frac{d}{m}$-symbol. Thus, the bit length of the RS codeword is thrice longer than that of the input data. For instance, for a 16-bit input data with m=8, $\frac{16}{8}$=2-symbol are needed for the dataword; and, $2\times\frac{16}{8}$=4-symbol are required for the checkword to correct the 2-symbol dataword. All the required number of symbols and their corresponding number of bits for different codeword length of RS code are shown in Table 1.

2.2 Conventional RRNS

RRNS code has a similar structure and error correction capability as RS code [9]. The k-symbol dataword is referred to as *non-redundant residues*, whereas the (n–k)-symbol checkword is referred to as *redundant residues.* The residues may have *different* length depending on the moduli used to encode the input data into the RRNS codeword (i.e., b=$\lfloor log_2(moduli-1)+1\rfloor$ bits). There are three requirements when developing RRNS code: (i) the moduli are mutually prime positive integers, (ii) the succeeding moduli must be bigger than that of preceding moduli, and (iii) the product of both moduli sets must be larger than a *operating legitimate range*, i.e., 2^d–1 where d is input data length. More detail can be found in [8, 9].

In this work C-RRNS with three-residue dataword and six-residue checkword is used. The latter is required to ensure the protection of the former, i.e., t=3. This code is encoded based on: (i) $\{2^p{-}1, 2^p, 2^p{+}1\}$ *restricted* non-redundant moduli where p is positive integer and (ii) *unrestricted* redundant moduli composed by prime integers larger than those of the restricted non-redundant moduli. For non-redundant-moduli, the p value is selected in such a way that it results in the product of moduli as close as possible to the operating legitimate range. This in turn realizing a cost-effective implementation. Table 1 exhibits the number of the residues (symbols) needed in C-RRNS codeword and their corresponding bit length for different input data.

Table 1. RS and C-RRNS parameters for different memory bandwidth

Types of ECCs	**# of Symbols per Codeword**			**# of Bits per Codeword**		
	16-bit	32-bit	64-bit	16-bit	32-bit	64-bit
RS	6	12	24	48	96	192
C-RRNS	9	9	9	61	106	205

3 Optimized Redundant Residue Number System ECCs

This section introduces the optimized RRNS variants. First, it describes the strategies used for optimization; thereafter, the optimized ECCs.

3.1 Strategies Used for Optimized RRNS ECCs

From Table 1 one can clearly conclude that the bit length of C-RRNS codeword is longer than that of RS codeword regardless of the input data length. However, C-RRNS possesses a better overall error correction capability than RS (will be discussed in Section 4.2). To reduce the bit length of C-RRNS codeword while providing a competitive correction capability, the following strategies are considered:

1. *For non-redundant moduli*: the number of non-redundant moduli is minimized as long as their product is larger than legitimate range. This implies smaller number of redundant moduli for a set error correction $t=\frac{n-k}{2}$.
2. *For redundant moduli*: the redundant moduli with smaller integer value than that of the non-redundant moduli are chosen; yet their product is larger than the operating legitimate range. Note that this strategy violates the second requirement in developing RRNS code as mentioned in Section 2.2 because of the smaller succeeding moduli (redundant moduli) than the preceding (non-redundant moduli). This violation may cause an inconsistency during decoding where a single read codeword might be decoded into *more than one* (ambiguous) output data. However, the third strategy is introduced to solve this problem.
3. *Maximum likelihood decoding (MLD)*: this method is used to determine the authentic output data from the ambiguous data [11]. First, each ambiguous data is encoded resulting into new codeword. Then, Hamming distance between each new codeword and the read codeword (i.e., the codeword that produce the ambiguous data) is calculated. Finally, the ambiguous data of the codeword that has the smallest difference (Hamming distance) is regarded as the authentic output data.

3.2 Six-Moduli RRNS variants (6M-RRNS)

The optimized RRNS code is referred to as Six-Moduli RRNS (6M-RRNS). This code is encoded based on two non-redundant moduli and four redundant moduli. The latter are used to protect the former from faults, i.e., t=2. 6Ma-RRNS have been presented in [8], whereas 6Mb-RRNS and 6Mc-RRNS are proposed in this work. Table 2 gives the three 6M-RRNS variants together with the values of p and their corresponding codeword bit length.

Table 2. Optimized RRNS parameters for different memory bandwidth

Types of ECCs	**Values of p**			**# of Bits per Codeword**		
	16-bit	32-bit	64-bit	16-bit	32-bit	64-bit
6Ma-RRNS	8	16	32	40	88	184
6Mb-RRNS	9	17	33	38	86	182
6Mc-RRNS	10	18	34	36	84	180

1. 6Ma-RRNS is based on non-redundant moduli set $\{2^p{+}1, 2^p\}$ and redundant moduli set $\{2^{p-1}{-}1, 2^{p-2}{-}1, 2^{p-3}{-}1, 2^{p-4}{+}1\}$.
2. 6Mb-RRNS is based on non-redundant moduli set $\{2^p{-}1, 2^{p-1}{+}1\}$ and redundant moduli set $\{2^{p-3}, 2^{p-4}{-}1, 2^{p-5}{+}1, 2^{p-5}{-}1\}$.
3. 6Mc-RRNS is based on non-redundant moduli set $\{2^p, 2^{p-4}{+}1\}$ and redundant moduli set $\{2^{p-5}{+}1, 2^{p-5}{-}1, 2^{p-6}{+}1, 2^{p-7}{-}1\}$.

4 Experimental Results and Analysis

This section evaluates the 6M-RRNS variants by comparing their performance to RS and C-RRNS. First, it presents the simulation set up followed by the experimental results of error correction capability of these ECCs for 32 and 64-bit dataword memory. Thereafter, it give the analysis of three different aspects of the results including: (i) the required codeword length of each ECC, (ii) the ratio of correctable bits over the required codeword length, and (iii) the decoding latency of RRNS codes.

4.1 Simulation Setup

The RRNS variant codes, RS code, memories and fault injection were described using MATLAB script. All codes were set to the desired t to protect their corresponding codeword from faults. For RRNS decoding, an algorithm called *Mixed Radix Conversion* was used. For RS code, MATLAB built-in RS encoding and decoding functions were used [12]. Clustered faults were randomly increased from two bits up to 35 bits for 32-bit memory and two bits up to 68 bits for 64-bit memory. Fault rates from 1% to 10% were applied during the experiments.

4.2 Overall Results

Figure 1 shows the simulation results of C-RRNS, 6M-RRNS variants and RS codes for 32 and 64-bit dataword memory. Overall, C-RRNS provides the best error correction capability followed by 6M-RRNS variants, whereas RS scores the worst. As the input data increases the difference between all investigated ECCs becomes marginal. For example, the difference between C-RRNS and 6Mb-RRNS is 0.8% for 32-bit, yet it is only 0.4% for 64-bit.

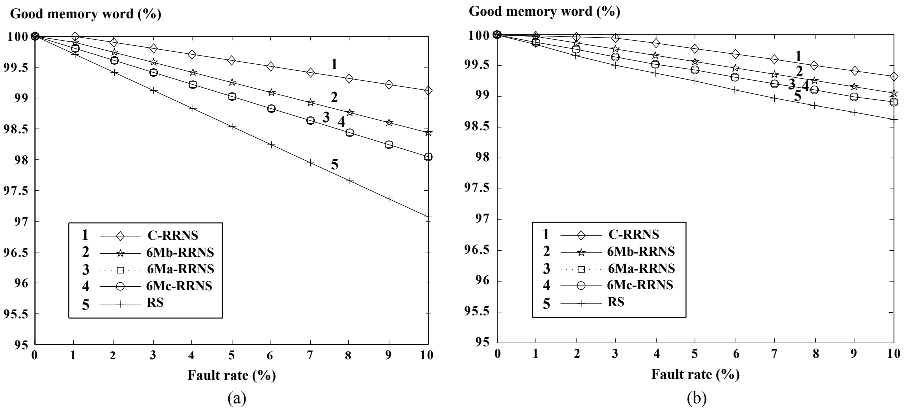

Fig. 1. Simulation results for (a) 32-bit memory (b) 64-bit memory

It is clear from the above that the 6M-RRNS variants can realize a competitive error correction capability; especially for large memory word size, which is expected to be the case for nanoelectronic memories. In next section, we will explain their advantages over RS and C-RRNS in terms of user data capacity and decoding speed.

4.3 ECCs versus Codewords Length

Figure 2 shows the required codeword length (in terms of bits) of the investigated ECCs for all considered input data length. The bit length of the ECCs are also given in Table 1 and 2. The figure shows that all 6M-RRNS variants have shorter codeword length as compared to RS and C-RRNS. For example, for a 64-bit dataword memory 6Mc-RRNS realizes a codeword which is 6.25% and 12.20% shorter than that of RS and C-RRNS codes, respectively. Overall, 6Mc-RRNS requires the shortest codeword bit length among the optimized codes.

Having shorter codeword length provides more capacity of user data storage for a given memory size. For instance, if we assume a memory size of 1Tbit, then capacity (in terms of memory words) is calculated by dividing $1T=2^{40}$ bits by codeword length B_c, i.e., $C=\frac{2^{40}}{B_c}$. For example, for a 64-bit dataword memory, the capacity of stored data encoded into 6Mc-RRNS is 6.11Gwords. It means that 6Mc-RRNS provides 6.25% more storage than RS and 12.20% more storage as compared to C-RRNS.

4.4 Decoding Latency of RRNS Codes

Another advantage of having shorter codeword length for the optimized RRNS variants is faster decoding. This is because the correction procedure in RRNS is an iterative process, which is proportional to the number of residues in the codeword, i.e., $C_t^n=\frac{n!}{t!(n-t)!}$ [9]. Regardless of codeword length, C-RRNS requires nine residues to protect its three-symbol dataword, i.e., t=3. This code stops when it recovers the correct data (data less then the operating legitimate range). Hence on the average, this code needs maximum of $\frac{C_3^9}{2}=\frac{1}{2}\times\frac{9!}{3!(9-3)!}=42$ iterations. However, 6M-RRNS variants require six residues to protect their two-symbol dataword. In addition, these codes always need an extra MLD step (see Section 3).

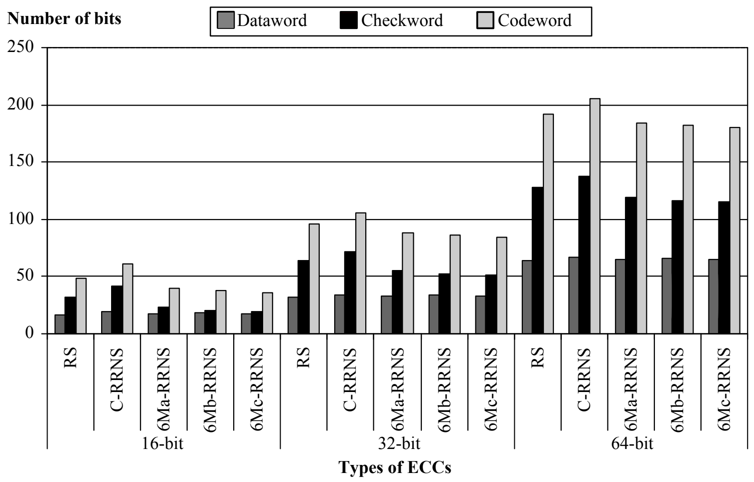

Fig. 2. Comparison of bit length of RS and RRNS variants codes

Therefore, they require maximum $C_2^6 + MLD = \frac{6!}{2!(6-2)!} + 1 = 16$ iterations during decoding procedure. Thus, 6M-RRNS decodes 2.6 times faster than C-RRNS.

5 Conclusion

In this paper the optimized variants of RRNS to correct cluster errors in nanoelectronic memories have been introduced and experimented. Three strategies were used to have the optimized ECCs such as: (i) smaller number of non-redundant moduli, (ii) smaller integer value for redundant moduli, and (iii) maximum likelihood decoding. Experimental simulation and analysis show that all optimized RRNS codes are able to provide competitive correction capability with better data storage as compared to well-known RS and C-RRNS codes. Among the optimized RRNS variants, 6Mc-RRNS is the best as it provides the shortest codeword and thus the largest data storage. Furthermore, all 6M-RRNS variants operate times faster than C-RRNS. This investigation has proven that the optimized RRNS offer an attractive solution for reliability improvement in developing nanoelectronic memories at low cost.

References

1. The International Technology Roadmap for Semiconductors 2009. Available: http://www.itrs.net/Links/2009ITRS/Home2009.htm
2. D. B. Strukov and K. K. Likharev, "Prospects for Terabit-scale Nanoelectronic memories", *J. Nanoscience and Nanotechnology*, vol. 16, no. 1, pp. 137–148, 2005.
3. A. DeHon, S. C. Goldstein, P. J. Kuekes, and P. Lincoln, "Nonphotolithographic Nanoscale Memory Density Prospects", *IEEE Trans. on Nanotechnology*, vol. 4, no. 2, pp. 215–228, 2005.
4. M. Mishra and S. C. Goldstein, "Defect Tolerance at the End of the Roadmap", in *Proc. of International Test Conference*, vol. 1, pp. 1201–1211, 2003.
5. M. Butts, A. DeHon, and S. Goldstein, "Molecular Electronics: Devices, Systems and Tools for Gigagate, Gigachips", in *Proc. of IEEE/ACM International Conference on Computer-aided Design*, pp. 433-440, 2002.
6. D. B. Strukov and K. K. Likharev, "Defect-Tolerant Architectures for Nanoelectronics Crossbar Memories", *J. Nanoscience and Nanotechnology*, vol. 7, no. 1, pp. 151–167, 2007.
7. H. Naeimi and A. DeHon, "Fault Secure Encoder and Decoder for NanoMemory Applications", *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol.17, no.4, pp. 473–486, 2009.
8. N. Z. Haron and S. Hamdioui, "Residue-based Code for Reliable Hybrid Memories", in *Proc. of IEEE/ACM International Symposium on Nanoscale Architectures*, pp. 27–32, 2009.
9. F. Barsi and P. Maestrini, "Error Correcting Properties of Redundant Residue Number Systems", *IEEE Trans. of Computers*, vol. 22, no. 3, pp. 307–315, 1973.
10. S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*. 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 2004.
11. 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, 2008.
12. MathWorks^{TM}. *Reed-Solomon Decoder Simulation.* Available: http://www.mathworks.com/matlabcentral