Highly Efficient Layered Syndrome-based Double Error Correction Utilizing Current Summing in RRAM Cells to Simplify Decoder

Shruti Dutta[1], Sai Charan Rachamadugu Chinni[2], Abhishek Das[3], Dr. Nur A. Touba[4]
Department of Electrical and Computer Engineering, The University of Texas at Austin[1,2,4]
Intel Corporation[3]
shrutidutta@utexas.edu[1], rcsaicharan@utexas.edu[2], abhishekdas@utexas.edu[3], touba@utexas.edu[4]

***Abstract*— Applications involving machine learning and neural networks have become increasingly essential in the AI revolution. Emerging trends in Resistive RAM technologies provide high-speed, low-cost, scalable solutions for such applications. These RRAM cells provide efficient and sophisticated memory hardware structures for machine-learning applications. However, it is difficult to achieve reliable multilevel cell storage capacity in these memory technologies due to the occurrence of soft and hard errors. As these memories can store multi-bits per cell, exploring limited magnitude symbols(multi-bit) error correction in RRAM is important. This paper proposes a new syndrome-based double error correcting code that divides the syndromes into groups and, uses addition and XOR operations to correct double limited magnitude errors in the RRAM cells. The key idea is to use the built-in current summing capability of RRAM cells to perform the addition operations that are used for the error correction thereby greatly reducing the overhead of the decoding logic needed to implement the ECC. This effectively avoids the need for explicit adder hardware in the decoding logic making it smaller and faster than conventional ECC codes with similar error-correcting capability. Experimental results show that the proposed code reduces the number of check symbols and significantly reduces the decoder area and power by using the RRAM cells to perform the addition.**

***Keywords*— Error Correcting Codes(ECC), Double Error Correcting codes (DEC), resistive RAM (RRAM), Orthogonal Latin Square (OLS), syndromes, soft errors**

I. Introduction

The field of machine learning and artificial intelligence is evolving rapidly. The need for efficient hardware solutions to implement these software applications is more than ever. Resistive Random-Access Memory (RRAM) technology is one of the coherent hardware solutions for these use cases. Its cross-bar structure helps in performing array-based applications such as matrix multiplication through current summation. RRAM cells make use of resistance switching materials that can change their resistance level under the application of electric field [1].

However, these memories are susceptible to soft and hard errors. In case of hard errors, which are generally modeled as stuck-at-0 (high resistance state) or stuck-at-1(low resistance state), the RRAM cells become practically unusable as the resistance level cannot be changed under these conditions [2]. Previous work mostly aims at alleviating hard errors which in turn aim at retraining the models. A method to detect stuck cells by extracting test vectors was proposed in [3]. Quiescent-Voltage comparison is used for fault detection followed by neuron re-ordering in [4].

The effect of soft errors cannot be overlooked, however, limited research has been done in this domain. [5] determines the susceptibility of RRAM cells to soft errors.

Limited magnitude errors, a type of soft error, can occur in these cells due to their ability to store multiple bits depending on the resistance of the cell. Drifts in resistance levels can lead to potential errors. Limited magnitude errors can be either symmetric or asymmetric [6]. For example, a 3-bits/cell will generate an asymmetric error of +1 to +7 however the symmetric error will range from +1 to +3. Improvements have been made to nonbinary Hamming codes which help in achieving better decoding complexity however they can be used for correcting single magnitude errors only [7]. For multiple error correction, non-binary majority decodable Orthogonal Latin Square codes, BCH codes have been proposed [8] [9]. The new codes discussed in [10] offer parallel decoding and smaller check bit length.

Techniques for detecting and correcting limited magnitude errors specifically in RRAM cells have not been explored much. [2] proposes a scheme to detect and correct single-column errors, however, there are no codes proposing double-column error correction. In this paper, we propose a scheme for correcting limited magnitude errors in the symbols in double columns. The key idea is to use the built-in current summing capability of RRAM cells to perform the arithmetic operations that are used for the error correction thereby greatly reducing the overhead of the decoding logic needed to implement the ECC. This effectively avoids the need for explicit adder hardware in the decoding logic making it smaller and faster than conventional ECC codes with similar error-correcting capability.

The structure of the paper is as follows: Section II explains the details of RRAM memory technology, and Section III discusses the details of OLS codes and their decoding procedure. The proposed new codes are explained in Section IV followed by the results and conclusion in Section V.

II. RRAM TECHNOLOGY

RRAM resistive technology is a novel non-volatile memory type that has a metal-insulator-metal structure and stores data by modulating the resistance of this insulator material. The RRAM structure as shown in Fig. 1 looks like a typical crossbar with *N* bit-lines and *M* word-lines. These memory cells are capable of storing multi-bit per cell thus offering high density.

The RRAM cells function in the following manner :

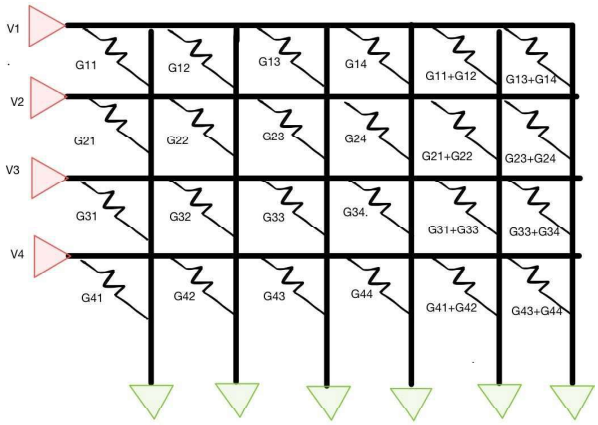

Fig. 1. RRAM crossbar structure

- The current passing(I) from each RRAM cell is given by (1) when a Voltage V is applied (Let G denote the conductance).

$$I = G.V \tag{1}$$

- The arithmetic sum from each RRAM cell gives us the final current. The final current across a bit line is given in (2).

$$I_1 = G_{11}.V_1 + G_{21}.V_2 + G_{31}.V_3 + G_{41}.V_4 \tag{2}$$

- It can be concluded that the dot product of input voltage at the word-lines and the conductance G can give us the value of the final current I.
- This is effectively matrix multiplication in the analog domain. Finally, an ADC converter is used to get the digital output result of the matrix multiplication.

The conductances are usually mapped at discrete levels using integral values which act as the weights of the neural network. As the number of distinct quantized levels is limited, online error correction gets restricted to smaller matrices. Further, using weighted checksum-based decoding would involve integer division which is an expensive operation. Also, in the weighted sum approach, as the number of bits increases, the area and delay overhead increases exponentially as shown in (3). [2] explores limiting selective-checksum decoding technique but only for single-bit errors. Hence, in this paper, a direct syndrome group-wise-based decoding technique for decoding and correcting double-column errors is proposed.

$$\#Syndromes/symbol = 2^b - 1 \tag{3}$$

Consider the following example with a 2x8 RRAM crossbar where each cell stores a 3-bit value. The conductance value G_{y1} is the weighted checksum as shown in (4). As each cell can store from 0-7, the value of $G_{y1,max}$ will be 252 as shown in (5). Total I would be G_{y1} plus G_{y2} is equal to 2 times 252, that is, 504. Hence this would require a 9-bit ADC and as the size of the matrix increases, this number keeps on increasing exponentially.

$$G_{y1} = G_{1,1} + 2.G_{1,2} + 3.G_{1,2}..... + 8.G_{1,8} \tag{4}$$

$$G_{y1,max} = 7.(1 + 2 + 3... + 8) = 252 \tag{5}$$

III. OLS CODES

A Latin square is a matrix of size *m x m*, where m is the order of the matrix, consisting of *m* unique symbols such that no element appears more than once in a given row or column. OLS codes are formed using such mutually orthogonal Latin squares [8]. Unlike Hamming codes, which can be used to correct one data symbol error, OLS codes can correct multiple data errors. In general, to correct t errors, m OLS matrices would be required where k data symbols can be corrected ($k<m^2$). These matrices will require *2tm* check symbols. OLS codes have the ability to correct multiple-data symbols, unlike Hamming codes which can correct single errors. The OLS code's parity check matrix (H) is constructed as shown below in (6).

$$H = \begin{pmatrix} M_1 & \\ M_2 & I_{2tm} \\ M_3 & \end{pmatrix} \tag{6}$$

In the above equation, I_{2tm} is the identity sub-matrix of which represents the parity portion of the parity check matrix. The sub-matrices M_1, M_2 and so on are *m x m* size matrices that satisfy the OLS properties. M_1 and M_2 are shown in (7) and (8).

$$M_1 = \begin{pmatrix} 1 & 1 & & \\ & & 1 & 1 \end{pmatrix} \tag{7}$$

$$M_2 = \begin{pmatrix} 1 & & 1 & \\ & 1 & & 1 \end{pmatrix} \tag{8}$$

OLS codes use the majority decoding property. Corresponding to each data bit, there are *2t* parity bits. For any t-errors, *t-1* syndromes will indicate no change while *t+1* syndromes will have their data bit flipped. A clear majority will indicate whether data bits were flipped or not. XOR is then used with the corresponding data bits to get the correct output.

A. Non-binary OLS Codes

This decoding technique is set over finite rings [11]. This code uses the same binary OLS H matrix *(n, k)* for *t*-errors. These conditions are valid as the non-binary H-matrix satisfies the RC constraint and the minimum weight of the column is also *2t*, thus keeping the *2t+1* distance constant. The same binary OLS code encoder, decoder, and syndrome generator can be re-used just by adding additional circuitry for the rings

(XOR gates over GF(2^b)).

The decoder makes use of the majority decoding logic, quite like the one used in Binary OLS codes [8]. Instead of using traditional n-2^n decoders, it counts the number of 1's in a binary digit.

B. One-step decodable limited magnitude OLS codes [12]

The codes in [11] are different from the Non-binary OLS codes due to the following changes:

- XOR and AND operations are replaced with the decimal arithmetic operation, SUM and MULTIPLICATION.
- The parity check matrix is repeated for more data symbols such that more than one data symbol will have *2t* independent sources.

According to the second point, repetitions can be infinite. However, this is limited by the factor that none of the data symbols should generate the same syndrome. If this happens then it is not possible to distinguish the erroneous bits.
The H-matrix for (8,4) OLS code is shown in Fig. 2.

H=

d0	d1	d2	d3	d4	d5	d6	d7	p0	p1	p2	p3
1	1	0	0	2	2	0	0	1	0	0	0
0	0	1	1	0	0	1	1	0	1	0	0
1	0	1	0	2	0	2	0	0	0	1	0
0	1	0	1	0	2	0	2	0	0	0	1

Fig. 2. H-matrix for limited magnitude errors for SEC

C. Layered OLS ECC for DEC

This scheme described in [10] consists of two layers and is valid for data bits rather than symbols. Single Error correcting OLS codes, that form the first layer find the possible error location. The second layer is made using the following two conditions :

- The added columns should be unique so they are not repeated.
- The sum of two columns of the entire H-matrix shouldn't repeat with any other two columns.

The first condition helps in ensuring that the syndrome produced by double errors is non-zero. The second condition helps in finalizing the location of the short-listed error locations. The two groups are formed by considering *m* data symbols where m=$k^{1/2}$ make up the first group. The remaining data bits belong to group 2.

In this paper, this scheme is extended to symbols by using SUM instead of OR gates and continuing to use the XOR gates.

IV. Proposed Scheme

The proposed approach aims for Double Error Correction in RRAM cells using the arithmetic properties of the same. The goal is to correct *n*-bit data symbols using *n*-bit check symbols for limited magnitude errors. This scheme consists of two ECC layers. As mentioned in the previous section, the first layer helps in getting a shortlist of possible error locations for single errors while the second layer helps in getting to the final error location. As these are a type of arithmetic code, after constructing the H-matrix using binary 1s and 0s, the matrix can be repeated in such a way that the sum of any two columns should not be equal. For example, as shown in Fig. 3, the H-matrix is constructed using a repetition factor of 2. Hence, correcting 32 data symbols will require only one additional check symbol compared to 16 data symbols.

H=

1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	2	2	2	2	0
1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	2	0	0	0	2	0	0	0	2	0	0	0	2	0	0	0	0
0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	2	0	0	0	2	0	0	0	2	0	0	0	2	0	0	0
0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	2	0	0	0	2	0	0	0	2	0	0	0	2	0	0
0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	2	0	0	0	2	0	0	0	2	0	0	0	2	0
1	1	0	1	0	1	0	0	0	0	1	1	1	0	1	1	2	2	0	2	0	0	0	0	2	2	2	0	2	2	0	2	2
1	0	1	0	1	0	0	1	1	0	0	1	1	0	1	2	0	2	0	2	0	0	0	0	2	2	2	0	2	2	0	2	2
0	1	1	0	0	0	1	0	1	0	1	0	1	1	1	0	0	2	2	0	0	0	2	0	2	0	2	0	2	2	2	2	0
0	0	0	1	1	0	1	0	0	1	0	1	1	1	0	0	0	0	2	2	0	2	0	0	2	0	2	0	2	2	2	0	0
0	0	0	0	1	0	1	1	0	0	0	1	1	0	0	0	0	1	0	2	2	0	0	0	0	1	0	2	2	2	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Fig. 3. H-matrix of proposed scheme for 32 data symbols

$Gsum_1$,$Gsum_2$,$Gxor_1$ and $Gxor_2$ refers to the partial sum of the syndrome groups. Here the groups are determined by the order of the matrix. For an H-matrix of order *m*, in that case, $Gsum_1$ corresponds to the sum of *2m* syndromes. For example, for 16 data symbols, *m* is 4, hence $Gsum_1$ is the sum corresponding to S_0 - S_7 sums and $Gsum_2$ corresponds to S_8 - S_{12} sum. Similarly, $Gxor_1$ refers to a multi-bit xor of S_0 - S_7 and $Gxor_2$ is multi-bit xor of S_8 - S_{12}. The equations for $Gsum_1$, $Gsum_2$, $Gxor_1$ and $Gxor_2$ are given in equations (9)-(12).

$$G_{sum1} = S_0 + S_1 + S_2 ..... + S_{m-1} \quad (9)$$

$$G_{sum2} = S_m + S_{m+1} + S_{m+2} ..... + S_{2m-1} \quad (10)$$

$$G_{xor1} = S_0 \oplus S_1 \oplus S_2 ..... \oplus S_{m-1} \quad (11)$$

$$G_{xor2} = S_m \oplus S_{m+1} \oplus S_{m+2} ..... \oplus S_{2m-1} \quad (12)$$

The decoding scheme is described below:

- If $Gsum_1$=$Gsum_2$=$Gxor_1$=$Gxor_2$=0 $\Rightarrow$ No error
- If $Gsum_1$=$Gsum_2$ and $Gxor_1$=$Gxor_2$=$Gsum_2$ $\Rightarrow$ Single error condition or Error in multi-bit in distinct groups if syndrome values are non-zero. In this case, a simple majority voting logic can be used.
- $Gxor_1$ = 0 and $Gxor_2 \neq 0$ $\Rightarrow$ same magnitude error in distinct groups. (In this case, the group refers to *m* columns of the H-matrix having symmetry. For e.g for 16 data symbols, these groups are d_0 - $d-3$,d_4 - d_7 and so on). Hence, in this case one error belongs to d_0 - d_3 while the other is either in d_4 - d_7 or d_8 - d_11 one or the other.
- $Gxor_1 \neq 0$ and $Gxor_2$ = 0 $\Rightarrow$ same magnitude error in same group,
- $Gsum_1 \neq Gxor_1$ $\Rightarrow$ multi-bit error in different groups
- $Gsum_2 \neq Gxor_2$ $\Rightarrow$ multi-bit error in the same group

Data Symbols	OLS MLD [11]			Proposed Code			
	Check Symbols	Area(um^2)	Latency(ns)	Check Symbols	Area(um^2)	Power	Latency(ns)
16	16	672.1	0.12	15	387.89	0.988	0.13
32	28	1362.1	0.19	16	762.13	1.857	0.15
64	32	2838	0.23	25	1578.23	2.21	0.22
128	64	5755	0.39	26	3412.45	3.96	0.31
256	64	11984	0.39	45	6921.54	8.21	0.35

Table 1. DEC decoder performance parameters for RRAM cells

Apart from the above cases, 2 additional check bits are required to distinguish between a single error and a double error in parity. The two check bits basically compute the parity of each group of parities to detect errors in the parity bits.

Consider a simple example, with reference to the H-matrix in Fig. 3, let's say data symbols d_0 and d_{16} are in error. Let's assume there's a +1 magnitude error in both of these data symbols. In the parity cells, we would only be storing the lower three bits, to account for overflows from computed parity. In order to calculate the syndromes, we use the arithmetic sum along with the weights multiplied by the error magnitude *E* gives the syndrome. The syndromes S_0,S_4,S_8,S_9 will be 3 and S_{13} will be 2. As this is DEC, the possible combinations of errors from S_0 can be deduced to d_0/d_1/d_2/d_3 and d_{16}/d_{17}/d_{18}/d_{19}. From S_4, the only possible combination satisfying the above conditions are d_0 and d_{16}. S_{13} is 2 which gives the insight that there's a +1 error in d_{16} which leads to the fact that d_0 has a +1 error as well.

V. Results

The proposed codes were synthesized on Synopsys Design Compiler using NCSU FreePDk45 library for *k*=16, 32, 64, 128, and 256 for +1 error magnitude. Table 1. shows the comparison of check symbols used in DEC OLS MLD and proposed codes for asymmetric +1 error magnitude capable of storing 3 bits per cell. The proposed decoding scheme in RRAM cells for double-error correction makes use of the inherent property of RRAM cells being able to provide arithmetic sum. Hence the calculations of $Gsum_1$ and $Gsum_2$ do not require extra adders, thus effectively reducing the overall decoder area. Furthermore, due to the repetition factor(weights), the number of check symbols required for 16 data symbols and 32 data symbols only differ by 1. Hence, this novel decoding scheme is especially effective when the H-matrix can be used by keeping the order of the matrix the same as the previous data symbol's H-matrix. As shown in Table 1, 16 data symbols and 32 data symbols have almost the same number of check symbols.

When compared against DEC OLS MLD (Orthogonal Latin Square Majority Logic Decoding) [11] for limited magnitude errors where each cell can hold up to 3-bits, we can see that as the data symbol size increases, the number of check symbols decreases by a maximum of 60 percent. For example for 128 symbols, the number of check symbols reduces from 64 to 26, and for 256 symbols the number of check symbols reduces from 64 to 45 (approximately 30 percent). The decoder area for the proposed code is significantly smaller when compared with traditional OLSMLD. For 256 symbols, the area reduction is around 45 percent as expected earlier and the reduction in check symbols.

VI. Conclusion

This paper proposes Double Error Correcting Codes in RRAM cells for limited magnitude errors that are capable of storing multi-bit values. The proposed scheme uses the inherent arithmetic summation properties of the RRAM cells to reduce the decoder area substantially. The repetition of H-matrix columns with selected weight values also leads to an overall reduction in the decoder area.

VII Acknowledgements

This research was supported in part by the National Science Foundation under Grant No. CCF-2113914.

References

[1] F. Zahoor, T. Z. A. Zulkifli, and F. A. Khanday, "Resistive random access memory (rram): an overview of materials, switching mechanism, performance, multilevel cell (mlc) storage, modeling, and applications," *Nanoscale Research Letters*, vol. 15, 2020.

[2] A. Das and N. A. Touba, "Selective checksum based on-line error correction for rram based matrix operations," in *2020 IEEE 38th VLSI Test Symposium (VTS)*, 2020, pp. 1–6.

[3] M. Liu, L. Xia, Y. Wang, and K. Chakrabarty, "Fault tolerance for rram-based matrix operations," in *2018 IEEE International Test Conference (ITC)*, 2018, pp. 1–10.

[4] L. Xia, M. Liu, X. Ning, K. Chakrabarty, and Y. Wang, "Fault-tolerant training with on-line fault detection for rram-based neural computing systems," in *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2017, pp. 1–6.

[5] G. Li, S. K. S. Hari, M. Sullivan, T. Tsai, K. Pattabiraman, J. Emer, and S. W. Keckler, "Understanding error propagation in deep learning neural network (dnn) accelerators and applications," in *SC17: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–12.

[6] T. Klove, B. Bose, and N. Elarief, "Systematic, single limited magnitude error correcting codes for flash memories," *IEEE Transactions on Information Theory*, vol. 57, no. 7, pp. 4477–4487, 2011.

[7] A. Das and N. A. Touba, "Efficient non-binary hamming codes for limited magnitude errors in mlc pcms," in *2018 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, 2018, pp. 1–6.

[8] M. Y. Hsiao, D. C. Bossen, and R. T. Chien, "Orthogonal latin square codes," *Ibm Journal of Research and Development*, vol. 14, pp. 390–394, 1970.

[9] D. Strukov, "The area and latency tradeoffs of binary bit-parallel bch decoders for prospective nanoelectronic memories," in *2006 Fortieth Asilomar Conference on Signals, Systems and Computers*, 2006, pp. 1183–1187.

[10] A. Das and N. A. Touba, "Layered-ecc: A class of double error correcting codes for high density memory systems," in *2019 IEEE 37th VLSI Test Symposium (VTS)*, 2019, pp. 1–6.

[11] K. Namba and F. Lombardi, "Non-binary orthogonal latin square codes for a multilevel phase charge memory (pcm)," *IEEE Transactions on Computers*, vol. 64, no. 7, pp. 2092–2097, 2015.

[12] A. Das and N. A. Touba, "Efficient one-step decodable limited magnitude error correcting codes for multilevel cell main memories," *IEEE Transactions on Nanotechnology*, vol. 18, pp. 575–583, 2019.