Double Adjacent Error Correction in RRAM Matrix Multiplication using Weighted Checksums

Kenrick Xavier Pinto[1], Krishnaja Kodali[1], Abhishek Das[2], Nur A. Touba[1]
University of Texas at Austin[1], Intel Corporation[2]
kenrickpinto@utexas.edu, krishnajakodali@utexas.edu, abhishekdas@utexas.edu, touba@ece.utexas.edu

***Abstract*—Artificial Intelligence (AI) has permeated various domains but is limited by the bottlenecks imposed by data transfer latency inherent in contemporary memory technologies. Matrix multiplication, crucial for neural network training and inference, can be significantly expedited with a complexity of O(1) using Resistive RAM (RRAM) technology, instead of the conventional complexity of O(n^2). This positions RRAM as a promising candidate for the efficient hardware implementation of machine learning and neural networks through in-memory computation. However, RRAM manufacturing technology remains in its infancy, rendering it susceptible to soft errors, potentially compromising neural network accuracy and reliability. In this paper, we propose a syndrome-based error correction scheme that employs selective weighted checksums to correct double adjacent column errors in RRAM. The error correction is done on the output of the matrix multiplication thus ensuring correct operation for any number of errors in two adjacent columns. The proposed codes have low redundancy and low decoding latency, making it suitable for high throughput applications. This scheme uses a repeating weight based structure that makes it scalable to large RRAM matrix sizes.**

***Index Terms*—Resistive RAM, Error Correcting Codes (ECC), Weighted Checksums, Double Adjacent Error Correction**

I. Introduction

The rapid growth of machine learning and AI applications has created a growing demand for hardware that can accelerate these workloads. This can be achieved by speeding up matrix multiplication operations that form the bedrock of neural network algorithms. The conventional approach towards this has been to read matrices from memory and use a specialized hardware unit to achieve high throughput. However, this approach is limited by data transfer latency of modern memory technologies that aren't able to meet the demands of compute. Resistive RAM (RRAM) technology has tremendous potential in alleviating this bottleneck due to it's inherent crossbar structure that provides in-memory computation capabilities [1]. Storing neural network weights in the RRAM matrix can perform matrix multiplication with O(1) complexity [2].

However, RRAM adoption is made difficult due to it's many challenges. The primary challenge is RRAM's low yield and error prone nature compared to existing technologies [3] [4]. RRAM fabrication technology is at a stage where it's susceptible to a high error rate. These errors fall into two major categories: hard errors and soft errors. Hard errors manifest in the form of stuck-at-0 and stuck-at-1 faults at multiple points in the array. Soft errors on the other hand are more dynamic, caused usually by reading and writing to the RRAM array. There has been considerable amount of work done in the area of hard error correction in RRAM. [5] proposes using various matrix transformations like row-flipping, permutation and redefinition of the weight range to make RRAM more robust to stuck-at faults. Performing all the three transformations results in better recovery but is disruptive and adds additional computational overhead. To attain better coverage in this method, each weights needs to be represented by more than one memristor, which further increases the redundancy. [6] introduces SAFER, a method that effectively partitions data blocks to recover from stuck-at faults, reducing the hardware overhead compared to traditional error recovery techniques. However, the dynamic partitioning and inversion processes used in this paper introduce additional complexity and require careful hardware implementation.

Soft errors, on the other hand, are harder to detect as their occurance is random. There has been limited work on soft error detection and correction in RRAM. [7] proposes X-ABFT, which extends algorithm-based fault tolerance along the time dimension to correct hard and soft errors in RRAM. In this method, the RRAM matrix is divided into sub-arrays with two checksum columns per sub-array, weighted and non-weighted, to help locate the fault's column address. Row address of the faults is identified by applying additional test input vectors and monitoring the RRAM output. Though the proposed method helps correct soft errors, it has a low coverage of two cells per submatrix. Additional X-ABFT requires storing of RRAM output data across multiple clock cycles to accurately locate the fault. In our paper, we propose a decoding scheme that offers increased error coverage to any number of rows within two adjacent columns, while also achieving lower redundancy.

A two-layered syndrome-based error correcting code has been proposed in [8] that can correct limited-magnitude double column errors in the RRAM matrix. The first layer is used to shortlist a set of possible error locations and the second layer identifies the exact error location and the magnitude of correction. The decoding logic performs addition using RRAM cells, thus bringing down the decoder area and power. However, this method can only correct limited magnitude errors, and has a higher redundancy compared to the proposed scheme of double adjacent column error correction.

[9] uses selective checksum to correct any number of errors

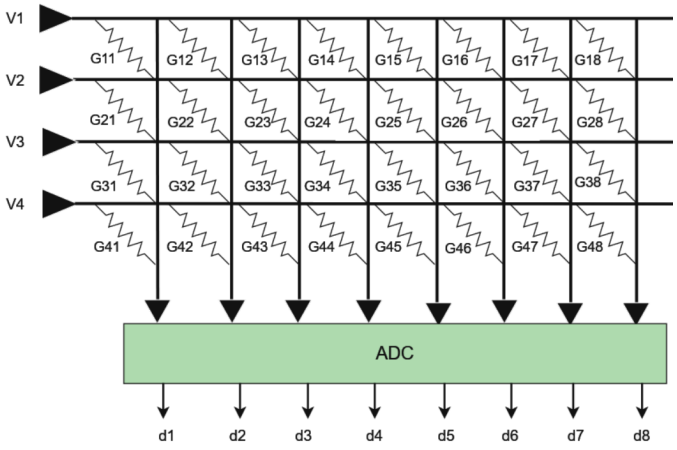

Fig. 1: Structure of a 4x8 RRAM matrix

in a single column in the RRAM matrix. This is accomplished by treating the output of matrix-vector multiplication as a single codeword, reducing multiple errors within a column into a single error in the output current. This approach shifts error correction focus from individual RRAM cells to columns. In this paper, we further extend the coverage to correct double adjacent column errors using weighted selective checksums. When we try to read or write to a column in an RRAM array, there is a possibility of faults being introduced. Instead of being localized to the column in question, they can also propagate to nearby columns, and can affect multiple rows at a time. We propose a scheme to correct errors present either in a single column or double adjacent columns. Our approach uses weighted checksums with repeated weights to obtain a low decoding latency, low redundancy and also simplicity in the decoder design. The rest of the paper is organized as follows. Sec. II discusses the RRAM structure for in-memory computation, Sec. III explains the proposed codes in detail. The decoder design is elaborated in Sec. IV and the results are presented in Sec. V. Finally, Sec. VI concludes the paper.

II. RRAM Structure

Resistive RAM (RRAM) is an emerging non-volatile memory technology which has a metal-insulator-metal stack structure, where the insulator layer exhibits resistive switching behavior. Voltage pulses are applied to RRAM during write/read operations to induce a localized change in the resistance of the insulator layer, thereby altering the cell's state. This resistance level is proportional to the data that needs to be stored. RRAM exhibits high scalability and high-density integration which makes it a popular choice for next-generation non-volatile memory solutions.

RRAM can be used to implement multi-level memory. This means each RRAM cell can store multiple bits of data per cell, for example, in [10] each RRAM cell can store upto 3 bits. It consists of a metal-insulator-metal structure and data is stored by varying the resistance level of the insulator. Figure 1 shows the crossbar structure of RRAM using which it performs weighted sums. For a voltage vector *(V_1, V_2, ..., V_{n-1}, V_n)* applied at the rows, the current vector obtained in the j^{th} column is given by the equation

$$d_j = G_{1j}V_1 + G_{2j}V_2 + ...G_{(n-1)j}V_{n-1} + G_{nj}V_n \quad (1)$$

This is essentially a dot product between the voltage vector and the j^{th} column of the RRAM matrix. In effect, the current vector obtained is a matrix multiplication of the voltage vector and the conductances stored in the RRAM array. We use this property to design an efficient decoding scheme using weighted checksums. The coefficients of the checksum will be stored as additional parity columns in the RRAM array and help us identify and correct double adjacent errors.

III. Proposed Error Correction Methodology

We use weighted checksums to correct double adjacent column errors. For an 8x8 RRAM matrix, we use the following four parity equations:

$$p_1 = d_1 + 2d_3 - d_5 - 2d_7 \quad (2)$$
$$p_2 = d_1 + 2d_2 + d_3 + 2d_4 + d_5 + 2d_6 + d_7 + 2d_8 \quad (3)$$
$$p_3 = 2d_1 + d_2 + 2d_3 + d_4 + 2d_5 + d_6 + 2d_7 + d_8 \quad (4)$$
$$p_4 = d_2 + 2d_4 - d_6 - 2d_8 \quad (5)$$

The parity check matrix for the 8x8 RRAM matrix is:

$$\begin{pmatrix} 1 & 0 & 2 & 0 & -1 & 0 & -2 & 0 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 2 & 1 & 2 & 1 & 2 & 0 & 1 & 0 & 0 \\ 2 & 1 & 2 & 1 & 2 & 1 & 2 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 2 & 0 & -1 & 0 & -2 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Accordingly, the syndrome values calculated by the decoder are defined as follows. Any non-zero syndrome implies the presence of an error.

$$S_1 = p_1 - (d_1 + 2d_3 - d_5 - 2d_7) \quad (6)$$
$$S_2 = p_2 - (d_1 + 2d_2 + d_3 + 2d_4 + d_5 + 2d_6 + d_7 + 2d_8) \quad (7)$$
$$S_3 = p_3 - (2d_1 + d_2 + 2d_3 + d_4 + 2d_5 + d_6 + 2d_7 + d_8) \quad (8)$$
$$S_4 = p_4 - (d_2 + 2d_4 - d_6 - 2d_8) \quad (9)$$

We can obtain the parity values p_1, p_2, p_3 and p_4 using the inherent summing capability of the RRAM crossbar structure. These parity values are represented in additional redundant columns in the RRAM matrix. We assume that each cell in RRAM can store upto 3 bits. This results in the possible conductances per cell ranging from 0 to 7. To correct the proposed errors, we need a unique syndrome for every possible error value which ranges from $\pm$ 0 to $\pm$ 7 in each cell. For double adjacent errors, the range of syndrome values in each row corresponding to these errors is shown in Table I.

Syndrome	Possible errors	Range of error values
S_1	One column	-14 to +14
S_2	Two columns	-21 to +21
S_3	Two columns	-21 to +21
S_4	One column	-14 to +14

TABLE I: Range of syndromes in 8x8 RRAM matrix

Since errors could be both negative and positive, we use 2's complement form to represent negative values. To represent these as conductances without overflow we need atleast 6 bits

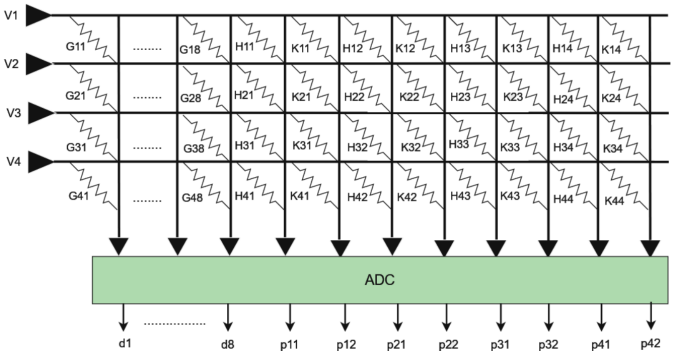

Fig. 2: 4x8 RRAM matrix augmented with parity columns

per row. However, each RRAM cell can only store 3 bits. Therefore, each parity equation is subsequently split across two columns. For example, in Figure 2, the conductance in row *i* for parity p_2 gets split into p_{21} and p_{22} as shown below

$$\{K_{i2}, H_{i2}\} = G_{i1} + 2G_{i2} + ... + 2G_{i8} \quad (10)$$

$$H_{i2} = (G_{i1} + 2G_{i2} + ... + 2G_{i8}) \textit{(mod 8)} \quad (11)$$

$$K_{i2} = ((G_{i1} + 2G_{i2} + ... + 2G_{i8}) \gg 3) \textit{(mod 8)} \quad (12)$$

Each of these H_{ij} and K_{ij} are 3 bits each and can be stored within a single cell. In an RRAM array having n rows, the current observed in each of these columns is shown below

$$p_{21} = \Sigma_{i=1}^{n} H_{i2} * V_i \quad (13)$$

$$p_{21} = \Sigma_{i=1}^{n} (G_{i1} + 2G_{i2} + .. + 2G_{i8}) \textit{(mod 8)} * V_i \quad (14)$$

$$p_{22} = \Sigma_{i=1}^{n} K_{i2} * V_i \quad (15)$$

$$p_{22} = \Sigma_{i=1}^{n} ((G_{i1} + 2G_{i2} + .. + 2G_{i8}) \gg 3) \textit{(mod 8)} * V_i \quad (16)$$

To obtain the parity equations p_1 through p_4, the decoder combines the two components and recreates the original parity as shown in Figure 3. For example,

$$p_2 = p_{21} + (p_{22} \ll 3) \quad (17)$$

$$p_2 = \Sigma_{i=1}^{n} (H_{i2} + (K_{i2} \ll 3)) * V_i \quad (18)$$

$$p_2 = \Sigma_{i=1}^{n} \{K_{i2}, H_{i2}\} * V_i \quad (19)$$

$$p_2 = \Sigma_{i=1}^{n} (G_{i1} + 2G_{i2} + ... + 2G_{i8}) * V_i \quad (20)$$

$$p_2 = \Sigma_{i=1}^{n} G_{i1} V_i + \Sigma_{i=1}^{n} 2G_{i2} V_i + ... + \Sigma_{i=1}^{n} 2G_{i8} V_i \quad (21)$$

$$p_2 = d_1 + 2d_2 + d_3 + 2d_4 + d_5 + 2d_6 + d_7 + 2d_8 \quad (22)$$

Similar results can be proven for other equations.

Under conditions of no error, each of the four syndromes S_1, S_2, S_3 and S_4 are all zero and a non-zero syndrome implies the existing of one or more errors. Moreover, in Sec. IV, we show that a regular and repeating checksum structure makes

Sno.	Error Possibility	Syndrome Values
1	No error	All syndromes are zero
2	Single data column	Either S_1 or S_4 is zero
3	Single parity column	Three syndromes are zero
4	Two adjacent data columns	S_1 and S_4 are non zero, At most one of S_2 and S_3 is zero
5	Two adjacent parity columns	Two syndromes are zero
6	Last data column and first parity column	All syndromes are non-zero

TABLE II: Possible error cases in the proposed scheme

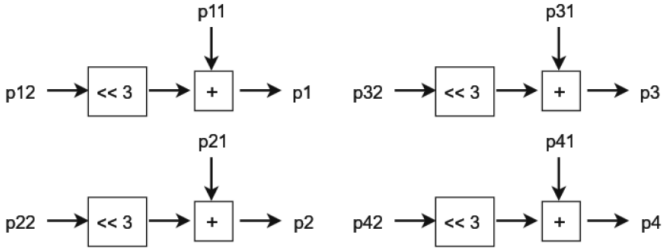

Fig. 3: Parity logic for 8x8 RRAM Matrix

the decoder design simple and modular. The proposed scheme uses 8 parity columns for 8 data columns (50% redundancy). We use four parity equations for all sizes of the RRAM array. Due to this, the redundancy values scales well and are shown in Table III. For sizes 8x8 and 16x16, we need two columns per parity equation and for 32x32 and 64x64, we need two parity columns for p_2 and p_3 but three parity columns for p_1 and p_4 as described later. The redundancy in the proposed scheme is better than the single column correcting majority voting code in [9] and the double column limited magnitude error correction in [8]. The difference between the redundancy of the proposed code and the hamming code reduces for higher order RRAM matrices.

IV. Decoder Design

The main motivation behind using repeated weights in our parity checksums is to make the equations symmetric. This makes the decoder design simple and opens up many optimization possibilities. We can reuse the same hardware to correct multiple columns. The proposed scheme has six distinct error possibilities as shown in Table II.

As can be seen, the first five categories are easily distinguished by counting the number of syndromes that are zero. This is achieved by using four comparators to check if each syndrome is zero or not. These four comparator outputs are then sent to an adder which returns a 2-bit output. Once we determine the type of error present, we need to perform correction only for cases 2 and 4 where data columns are in error. Parity column errors can be filtered out using the counter output and don't need any correction. Case 6 is a special case with a distinct set of possible syndromes that doesn't overlap with the other cases and can be easily detected. The syndromes corresponding to scenarios 2, 4 and 6 are shown in Table IV. The detection and correction logic for all possible single and double adjacent column errors are also shown. Additionally, note the presence of many recurring equations in the decoding logic ($2S_1 + S_4$, $S_1 + 2S_4$ etc) that can be easily implemented by using shift, add and complement operations and reused wherever necessary.

To extend the proposed error correction scheme to higher order RRAM matrices, and preserve the essence of the decoding logic, we define a few rules on the coefficients used. Any set of coefficients following these rules have the same syndrome behaviour as described before.

Number of Data Columns	Single Column Correction				Double Column Correction		Double Adjacent Column Correction	
	Majority Voting Code in [9]		Hamming Code in [9]		Limited Magnitude Correction in [8]		Proposed Code	
	Parity columns	Redundancy	Parity columns	Redundancy	Parity Columns	Redundancy	Parity columns	Redundancy
8	-	-	-	-	-	-	8	50%
16	7	30%	5	24%	15	48.3%	8	33.3%
32	9	22%	6	16%	16	33.3%	10	23.8%
64	12	16%	7	10%	25	28%	10	13.5%

TABLE III: Comparison of redundancy for various sizes of the RRAM matrix

Error in Column(s)	S_1	S_2	S_3	S_4	Detection Logic	Correction Logic
d_1	e_1	e_1	$2e_1$	0	$S_1 = S_2$, $S_4 = 0$	S_2
d_2	0	$2e_2$	e_2	e_2	$S_4 = S_3$, $S_1 = 0$	S_3
d_3	$2e_3$	e_3	$2e_3$	0	$S_1 = S_3$, $S_4 = 0$	S_2
d_4	0	$2e_4$	e_4	$2e_4$	$S_4 = S_2$, $S_1 = 0$	S_3
d_5	$-e_5$	e_5	$2e_5$	0	$S_1 = -S_2$, $S_4 = 0$	S_2
d_6	0	$2e_6$	e_6	$-e_6$	$S_4 = -S_3$, $S_1 = 0$	S_3
d_7	$-2e_7$	e_7	$2e_7$	0	$S_1 = -S_3$, $S_4 = 0$	S_2
d_8	0	$2e_8$	e_8	$-2e_8$	$S_4 = -S_2$, $S_1 = 0$	S_3
d_1, d_2	e_1	$e_1 + 2e_2$	$2e_1 + e_2$	e_2	$(2S_1 + S_4) = S_3$, $(S_1 + 2S_4) = S_2$	S_1, S_4
d_2, d_3	$2e_3$	$e_3 + 2e_2$	$2e_3 + e_2$	e_2	$(S_1 + S_4) = S_3$, $(S_1 + 4S_4) = 2S_2$	S_4, $S_1 \gg 1$
d_3, d_4	$2e_3$	$e_3 + 2e_4$	$2e_3 + e_4$	$2e_4$	$(2S_1 + S_4) = 2S_3$, $(S_1 + 2S_4) = 2S_2$	$S_1 \gg 1, S_4 \gg 1$
d_4, d_5	$-e_5$	$e_5 + 2e_4$	$2e_5 + e_4$	$2e_4$	$(S_4 - 4S_1) = 2S_3$, $(S_4 - S_1) = S_2$	$S_4 \gg 1$, $-S_1$
d_5, d_6	$-e_5$	$e_5 + 2e_6$	$2e_5 + e_6$	$-e_6$	$-(2S_1 + S_4) = S_3$, $-(S_1 + 2S_4) = S_2$	$-S_1$, $-S_4$
d_6, d_7	$-2e_7$	$e_7 + 2e_6$	$2e_7 + e_6$	$-e_6$	$-(S_1 + S_4) = S_3$, $-(S_1 + 4S_4) = 2S_2$	$-S_4$, $-S_1 \gg 1$
d_7, d_8	$-2e_7$	$e_7 + 2e_8$	$2e_7 + e_8$	$-2e_8$	$-(2S_1 + S_4) = 2S_3$, $-(S_1 + 2S_4) = 2S_2$	$-S_1 \gg 1, -S_4 \gg 1$
d_8, p_1	x	$2e_8$	e_8	$-2e_8$	$S_2 = -S_4$, $S_2 = 2S_3$	S_3

TABLE IV: Possible syndromes and correction logic for 8x8 RRAM matrix

Rule 1 *Coefficients in p_1 should contain odd data columns and be distinct to each other*

Rule 2 *Coefficients in p_4 should contain even data columns and be distinct to each other*

Rule 3 *Coefficients in p_2 and p_3 should alternate between 1 and 2 for data columns, in distinct order*

Rule 4 *Number of columns per parity equation is determined by the allowable range of syndrome values*

Applying these rules for a 16x16 RRAM matrix, we can generate the four parity equations as follows

$$p_1 = d_1 + 2d_3 - d_5 - 2d_7 + 3d_9 + 4d_{11} - 3d_{13} - 4d_{15} \quad (23)$$

$$p_2 = d_1 + 2d_2 + d_3 + 2d_4 + ... + d_{13} + 2d_{14} + d_{15} + 2d_{16} \quad (24)$$

$$p_3 = 2d_1 + d_2 + 2d_3 + d_4 + ... + 2d_{13} + d_{14} + 2d_{15} + d_{16} \quad (25)$$

$$p_4 = d_2 + 2d_4 - d_6 - 2d_8 + 3d_{10} + 4d_{12} - 3d_{14} - 4d_{16} \quad (26)$$

Since the maximum coefficient in p_1 and p_4 is 4 , the range of syndromes for 16x16 RRAM matrix is shown in Table V. Note that when we use alternating 2 and 1 as coefficients, the range of syndromes will always be -21 to +21 for p_2 and p_3 regardless of RRAM size and will require only two columns.

Syndrome	Possible errors	Range of error values
S_1	One column	-28 to +28
S_2	Two columns	-21 to +21
S_3	Two columns	-21 to +21
S_4	One column	-28 to +28

TABLE V: Range of syndromes for 16x16 RRAM Matrix

The decoding scheme for the 16x16 RRAM also uses 8 parity columns (2 columns for each parity equation). Similarly, for 32x32 and 64x64 RRAM sizes, we use 2 columns for p_2 and p_3 and 3 columns for p_1 and p_4, giving us a total of 10 extra parity columns. Note that the redundancy scales logarithmically with increasing RRAM size. For these equations, we can clearly observe that the five possible error configurations can be detected using the same mechanism as before. Moreover, any coefficient set created using these rules follows the same pattern. For example, in equations (22) and (25), we could have chosen powers of 2 i.e +4, +8, -4 and -8 as the last four coefficients. This would lead to a higher range of syndromes, resulting in 7 bits needed to represent equations (22) and (25) i.e 3 columns. The increased redundancy can be traded-off for lower decoding latency achieved through usage of powers of 2, thus limiting the decoder complexity. There is a direct trade-off between the redundancy and latency of the decoder based on the choice of coefficients. Optimal coefficients should be selected with careful consideration of these inherent trade-offs. Additionally, the initial part of each equation is preserved as we increase the size. Hence, a decoder used for the 8x8 RRAM can be used modularly to correct the 16x16 RRAM matrix, with additional hardware designed to handle the remaining cases. Thus, the proposed scheme lends itself amenable to scalable decoder design.

V. RESULTS

The proposed error correcting code was implemented on Synopsys Design Compiler using NCSU FreePDK45 library for RRAM sizes of 8x8, 16x16, 32x32 and 64x64. The redundancy, latency, area and power consumption of the decoder logic are described in Table VI. The decoder logic involves checking of several conditions to identify error locations and calculate the correction vector, which requires multiple adder units, resulting in non-trivial area overhead. Area overhead can be reduced through use of look-up tables [8] using existing

RRAM Matrix Size	Double Adjacent Column Correcting Code (Proposed Code)			
	Redundancy	Latency (ns)	Area (um^2)	Power (mW)
8x8	50%	1.7	5500	0.90
16x16	33.3%	2.31	15774	2.07
32x32	23.8%	3.48	34153	5.20
64x64	13.5%	4.10	81810	11.30

TABLE VI: Results for proposed codes for different matrix sizes

RRAM structures or some other on-chip memory. Based on the category of error present (which is filtered early on), only a certain part of the decoder is active at a time. This results in low dynamic power consumption as can be seen from the table.

VI. Conclusion

In this paper, we proposed a decoding scheme to correct double adjacent column errors in RRAM. RRAM dot product capabilities are utilized to perform parity additions with minimal overhead. We defined rules to construct decoding schemes for higher order RRAM arrays. The repeating and regular structure of these equations leads to modular and scalable decoder designs with low redundancy.

VII. Acknowledgments

This research was supported as part of the National Science Foundation under Grant No: CCF-2113914

References

[1] Yan, B., Li, B., Qiao, X., Xue, C.X., Chang, M.F., Chen, Y. and Li, H., 2019. Resistive memory-based in-memory computing: from device and large-scale integration system perspectives. Advanced Intelligent Systems, 1(7), p.1900068.

[2] Sun, Z. and Huang, R., 2021. Time complexity of in-memory matrix-vector multiplication. IEEE Transactions on Circuits and Systems II: Express Briefs, 68(8), pp.2785-2789.

[3] Liu, M., Xia, L., Wang, Y. and Chakrabarty, K., 2018, May. Design of fault-tolerant neuromorphic computing systems. In 2018 IEEE 23rd European Test Symposium (ETS) (pp. 1-9).

[4] Xia, L., Huangfu, W., Tang, T., Yin, X., Chakrabarty, K., Xie, Y., Wang, Y. and Yang, H., 2017. Stuck-at fault tolerance in RRAM computing systems. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 8(1), pp.102-115.

[5] Zhang, B., Uysal, N., Fan, D. and Ewetz, R., 2019. Handling stuck-at-fault defects using matrix transformation for robust inference of dnns. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 39(10), pp.2448-2460.

[6] Seong, N.H., Woo, D.H., Srinivasan, V., Rivers, J.A. and Lee, H.H.S., 2010, December. SAFER: Stuck-at-fault error recovery for memories. In 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (pp. 115-124).

[7] Liu, M., Xia, L., Wang, Y. and Chakrabarty, K., 2020. Algorithmic fault detection for RRAM-based matrix operations. ACM Transactions on Design Automation of Electronic Systems (TODAES), 25(3), pp.1-31.

[8] Dutta, S., Chinni, S.C.R., Das, A. and Touba, N.A., 2023, October. Highly Efficient Layered Syndrome-based Double Error Correction Utilizing Current Summing in RRAM Cells to Simplify Decoder. In 2023 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT) (pp. 1-4).

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

[10] Le, B.Q., Grossi, A., Vianello, E., Wu, T., Lama, G., Beigne, E., Wong, H.S.P. and Mitra, S., 2018. Resistive RAM with multiple bits per cell: Array-level demonstration of 3 bits per cell. IEEE Transactions on Electron Devices, 66(1), pp.641-646.