In-Memory Hamming Error-Correcting Code in Memristor Crossbar

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

03-11-2021 / 06-11-2021

CITATION

Bae, Woorham; Han, Jin-Woo; Yoon, Kyung Jean (2021): In-Memory Hamming Error-Correcting Code in Memristor Crossbar. TechRxiv. Preprint. <https://doi.org/10.36227/techrxiv.16926025.v1>

DOI

[10.36227/techrxiv.16926025.v1](https://doi.org/10.36227/techrxiv.16926025.v1)

In-Memory Hamming Error-Correcting Code in Memristor Crossbar

Woorham Bae, *Member, IEEE*, Jin-Woo Han, *Senior Member, IEEE*, and Kyung Jean Yoon

Abstract— This paper proposes a in-memory Hamming error-correcting code (ECC) in memristor crossbar array (CBA). Based on unique I-V characteristic of complementary resistive switching (CRS) memristor, this work discovers that a combination of three memristors behaves as a stateful exclusive-OR (XOR) logic device. In addition, a two-step (build-up and fire) current-mode CBA driving scheme is proposed to realize a linear increment of the build-up voltage that is proportional to the number of low-resistance state (LRS) memristors in the array. Combining the proposed XOR logic device and the driving scheme, we realize a complete stateful XOR logic, which enables a fully functional in-memory Hamming ECC, including parity bit generation and storage followed by syndrome vector calculation/readout. The proposed technique is verified by simulation program with integrated circuit emphasis (SPICE) simulations, with a Verilog-A CRS memristor model and a commercial 45-nm CMOS process design kit (PDK). The verification results prove that the proposed in-memory ECC perfectly detects error regardless of data patterns and error locations with enough margin.

Index Terms—crossbar memory, error-correcting code, Hamming code, in-memory computing, memristor, stateful logic

I. INTRODUCTION

Memristor crossbar is expected to make significant contributions to future computing system in various ways, such as high-density and low-latency non-volatile memories [1]–[6], neural networks [7]–[14], and process-in-memory (PIM) [15]–[20]. However, device reliability issue is one of the main limiters for their realization into a practical computing system [12]. For example, there is an inherent trade-off between performance and reliability of a memristor device that sacrificing reliability is inevitable to operate the memory at speed. Such issue aggravates further when they come to logic operations in neural network and PIM applications where the access rate increases by orders of magnitude.

Error-correcting codes (ECC) is an essential technique in memory technology that significantly enhances data reliability. Hamming ECC, first proposed in 1950 [21], is one of the most popular ECC and is still widely used to address the reliability

W. Bae was with the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720 USA.

W. Bae is now with Ayar Labs, Santa Clara, CA 95054 USA.

J.-W. Han is with the University Space Research Association, NASA Ames Research Center, Moffett Field, CA 94035 USA.

K. J. Yoon is with Intel Corporation, Santa Clara, CA 95054 USA. (e-mail: kyung.jean.yoon@intel.com)

Fig. 1. Conceptual diagram of Hamming ECC.

issue of the various legacy memories [22]–[26]. Parity bits are stored in memory array along with data bits during the data programming, and are used to tell whether there have been any bit flips, or error, occurred during a data read out. In general, an ECC engine located outside of the memory array calculates parity at programming and decodes the ECC at read out to detect and correct the error so that a clean data is sent to user.

Fig. 1 gives a brief explanation on how a Hamming ECC for 4-bit data (d1–d4) works, so-called Hamming (7, 4) ECC. Here, the entire data consists of 7 bits including 3-bit parity and 4-bit data. Hamming ECC necessitates even/odd parity generation during programming, where the count of ‘1’ in the data+parity codes always becomes either even or odd. Therefore, it starts by creating three parity bits (p1–p3) from three different combination of datasets based on exclusive-OR (XOR) logic operation, which is grouped into different circles of Fig. 1(a). In this case, the XOR logic outputs ‘1’ if the number of ‘1’ is odd in any of the three datasets. Therefore, the total number of ‘1’ in a circle of Fig. 1(a), which is comprised of 3-bit data and the corresponding 1-bit parity (e.g., d1, d2, d4, and p1 in the uppermost circle) should always be even. In other words, if we take an XOR of the 4-bit data in a circle during a read out, it

Fig. 2. I-V curve of CRS device model.

should be zero if there has been no bit corruption. Based on the observation, the Hamming ECC takes those XORs as a syndrome vector to exactly point where the error has occurred among the 4-data bits, as shown in Fig. 1(b). Fig. 1(c) and (d) show an example. With d1 through d4 being 1001, p1 through p3 becomes 001. With no error, the syndrome vector calculation must give (0, 0, 0). However, when an error occurs at d1, the XORs of p1 and p2 circles give 1 instead of 0, and lead to the syndrome vector of (1, 1, 0), which informs that the ECC should correct d1 before the data is sent to user. Similarly, the syndrome vector of (1, 0, 1), (0, 1, 1), and (1, 1, 1) point to an error occurred at d2, d3, and d4, respectively.

This work proposes a novel in-memory Hamming (7, 4) ECC in a memristor crossbar. Simulation program with integrated circuit emphasis (SPICE) verifications with a Verilog-A memristor model and commercial CMOS transistor are provided to fully validate the complete steps of Hamming encoding and decoding, including the parity bits and the syndrome vector generation. To the best of the authors' knowledge, this work is the first ever reported in-memory ECC scheme that implements an entire step within a memristor crossbar. [17] and [27] have proposed an in-memory parity generation and an in-memory parity check schemes, respectively, which in fact have inherent limitations to be extended to cover the entire steps.

The remainder of this paper is organized as follows. Section II introduces a complementary resistive switching (CRS) memristor which is adopted in this work [28]. A novel stateful XOR logic implementation based on the CRS memristor is also proposed in Section II. Section III details the circuit techniques for the robust stateful XOR logic, which are followed by SPICE validation results. Section IV presents the verification results from SPICE simulation with the device model and a complete array. Finally, conclusions are provided in Section V.

II. COMPLEMENTARY RESISTIVE SWITCHING (CRS) MEMRISTOR AND XOR LOGIC WITH CRS

[17] proposes and verifies that a current-mode driving scheme enables a linear increment of word-line voltage with respect to the number of low-resistance state (LRS) memristors in the dataset array. Combined with a unique I-V characteristic

Fig. 3. (a) Conceptual I-V curve of CRS-based XOR cell (b) simulated I-V curve of XOR cell, (c) resistance state table of each memristors.

of a CRS memristor and gray code, such linear word-line voltage is used to program parity bits to a CRS memristor, which is a base work of this work. Fig. 2 shows an example of I-V curve of the CRS memristor obtained from SPICE simulation with a Verilog-A model, which is used throughout this work. The device parameters are tunable in the model, but most of the parameters are chosen to mimic the physical device characteristic in [28] such as R_{off}/R_{on} . The unique characteristic of the CRS device originates from switching between the opposite defect configurations which can be read as a high resistance. Assuming that the CRS memristor is initialized to a high-resistance state (HRS), the resistance state goes to the LRS as a conducting path is formed when the applied voltage across the top and bottom electrodes increases to positive direction. However, if the voltage increases further, the device goes to another HRS state (HRS') due to defect migration to the opposite side [17]. On the other hand, once the voltage stops

Fig. 4. Stateful XOR logic in CRS memristor array with current-mode driving scheme. (a) The first step for build voltage up, (b) the second step to program parity bit, (c) simulated voltage transfer curve and linearity measurement for 3-input parity case, (d) simulated voltage transfer and linearity measurement for 4-input syndrome case, (e) illustration of why single-step approach fails.

increasing, the device stores the resistance state where the increment has stopped until a higher voltage is applied, or it is initialized to the HRS by reset operation. The reset happens when a strong negative voltage is applied.

As discussed in Section I, the binary XOR logic operation is the key for realizing the parity generation and the Hamming ECC. Although it is proven that the gray code and the multi-bit operation by utilizing the intermediate states between HRS/LRS and LRS/HRS' enables parity bit generation without using XOR logic in [17], however a further in-memory processing is not possible due to the complexity of decoding the gray-coded resistance. Therefore, to realize a complete in-memory ECC, this work does not rely on the multi-bit operation but utilizes the stateful in-memory XOR logic. Fig. 3(a) shows a conceptual explanation of the XOR logic with the CRS memristors. Given that a linear incremental voltage with respect to the number of '1's (=LRS memristors) in the dataset is available from the current-mode scheme, an I-V curve of 'M' shape or an R-V curve of 'W' shape can serve as an XOR logic. In addition, of course the curve should keep the resistance state to store the parities and the syndromes, in other words it should be a stateful logic. In Fig. 3(a), the blue line shows a simplified I-V curve of a single CRS device. If two CRS devices are stacked in series, its I-V curve must be a stretched version of a single CRS's in V-axis, as the red line in Fig. 3(a). If the stacked devices are connected in parallel with a single device, the total current is the sum of each current, where we obtain the M-shape I-V curve. For the rest of the paper, this cell is referred to an XOR memristor. The concept is fully verified with the SPICE

simulation with the CRS device model, which is shown in Fig. 3(b). Neglecting all intermediate states for simplicity, which are indeed included in the SPICE model, the XOR memristor has five resistance states: three HRS states (HRS_1, HRS_2, HRS_3) and two LRS states (LRS_1, LRS_2). Fig. 3(c) shows a state table of each resistance states for better understanding. Once initialized by a strong negative reset voltage, all three memristors are in the HRS state. If a positive voltage of 1 V is applied, R1 is flipped to LRS, but R2 and R3 stay at HRS because the voltage stresses are only 0.5 V due to the stacking. If the voltage increases to ~ 1.5 V, then it flips R1 to HRS' but it is still not enough to flip R2 and R3, leading to the overall resistance to HRS_2 . Once the voltage reaches 2 V, then R2 and R3 become LRS so the overall resistance goes to LRS_2 . If the voltage keeps increasing, R2 and R3 also reach HRS'.

III. VALIDATION OF IN-MEMORY XOR WITH CURRENT-MODE DRIVING SCHEME

In the previous Section, we discussed how a 'M'-shaped I-V curve is obtained and it can be used as an XOR logic. To fully implement XOR, the I-V curve should be translated to current versus digital input, which means the XOR memristor should be preceded by a digital-to-analog converter (DAC) which translate a digital data (i.e. how many '1's in the array) to an analog voltage. Fig. 4(a) and (b) shows a circuit diagram of CRS array which explains how the proposed current-mode driving DAC works. Four data memristors (R_{D1} - R_{D4}) and an XOR memristor share a word-line. When the array is configured to the XOR operation, the data memristors are

Fig. 5. Circuit-level illustration of each experiment steps. (a) Write phase, (b) parity generation phase, (c) syndrome program phase, (d) syndrome readout phase.

driven by PMOS transistors whose gates are biased to an analog voltage. When a data memristor is HRS, the transistor operates in triode region due to the low drain-source voltage (V_{DS}) by the large IR drop by the HRS memristor. In other words, the current of the path is constrained by the HRS memristor. On the other hand, when a data memristor is LRS, the biased transistor behaves as a current source because the IR drop across the memristor is low. As a result, the current flowing through the memristor is constrained by the saturation current of the transistor, which means the word-line voltage has little impact on the current. It allows a linear increase of the total current from the memristor array with respect to the number of LRS cells in the array. Due to the nature of parallel resistance, a typical voltage-mode scheme like [29] cannot provide such linearity. The linear current is translated to word-line voltage by a parallel build-up resistor R_{BU} . Note that during the build-up phase, the XOR path is disconnected so the current solely flows to the build-up resistor as shown in Fig. 4(a). The bleeder current ($I_{BLEED,P}$) is used to shift the current curve (y-intercept) by introducing a DC offset current to the build-up resistor. Once the word-line is fully charged, the bit-line of the XOR memristor is connected to V_{SS} as shown in Fig. 4(b), and simultaneously the build-up voltage programs the XOR. The program voltage versus the number of LRS cells in the dataset array is shown in Fig. 4(c) and (d), where the 3-input case (parity bit program) and the 4-input case (syndrome vector program) are given, respectively. Differential non-linearity (DNL) plots normalized to least-significant bit (LSB) are also depicted in Fig. 4(c) and (d), which prove the linearity (max DNL < 0.3 LSB) of the proposed current-mode DAC. The two-step (build-up and fire) scheme is required to resolve the voltage saturation issue of the one-step scheme in [17], which is

explained in Fig. 4(e). If the current directly drives the XOR without the build-up step and the resistor, the current-to-voltage transformation cannot be linear. For example, the voltage saturates at $\sim V_2$ even though $R_{D1}-R_{D4}$ are all LRS, because once the resistance enters the LRS region the IR drop cannot be higher.

IV. EXPERIMENTAL PROOF OF IN-MEMORY HAMMING ECC

A. Experimental testbench design

The proposed ECC is verified with SPICE simulation with a commercial 45-nm CMOS process design kit (PDK) and the Verilog-A memristor model. Note that a specific device shown in Fig. 2 is used in this experiment, however the scheme is not sensitive to the device parameters as long as the on/off ratio is comparable or higher, because the transistor design engineering provides enough flexibility. Fig. 5 shows the circuit diagram of the crossbar array and the experiment steps. The array consists of four data memristors and four XOR memristors, and they all share the same word-line. Three XOR memristors are to generate and to store the 3-bit parity for the Hamming ECC, but the last XOR memristor is to calculate the syndrome vector. In the first phase of the operation, 4-bit data is sequentially written to $R_{D1}-R_{D4}$. The word-line is driven by V_{SS} ($= 0V$) during this phase, and the bit-line of the selected cell is driven by V_{WRITE} , whose value depends on the binary datum to be written to the cell ($-1.0V$ to program '1' or LRS, $+2.5V$ to program '0' or HRS).

After programming the data memristors with the desired pattern, the parity generation phase starts. During this phase, the word-line switch to V_{SS} stays disconnected whereas the current-mode scheme drives the bit-line of the selected dataset while the unselected one stays unconnected. This step is

Fig. 6. Simulation result when the data is 0011 and the error is injected to d1.

repeated by three times to program 3 parity bits to R_{P1} , R_{P2} , and R_{P3} , while the selection is based on the parity generation matrix from Fig. 1. For example, for p1 generation, the vector is (1, 1, 0, 1) which means R_{D1} , R_{D2} , and R_{D4} are driven by the transistors but the bottom electrode of R_{D3} is floated, so it is the 3-bit XOR logic discussed in Section III. As described in the Section III, during the parity generation step, the bleeder current is also activated to bring the transfer curve to the optimum for the given XOR memristor.

In the next step, the syndrome vector is sequentially programmed to the last XOR memristor (R_{SYN}), and R_{SYN} readout happens for checking error. In fact, the programming-syndrome phase is divided into two sub-phases. Because the syndrome vector does not have to be stored since it will not be used in further in-memory operations unlike the parity bits, it is wasteful to have dedicated memristors for each syndrome elements. Therefore, in this experiment, a single shared XOR memristor is used so each syndrome elements are sequentially processed. As a result, the first sub-phase should be resetting the R_{SYN} to erase the previously stored bit. After the reset, the 4-bit stateful XOR logic happens, which is shown in Fig. 6(c). During this sub-phase, the dataset is also selected

from the Hamming ECC criteria shown in Fig. 1. For example, for s1 calculation, R_{D1} , R_{D2} , R_{D4} , and R_{P1} are selected while the others are floated. Because a different offset should be applied to this 4-bit XOR over the 3-bit XOR, an additional bleeder path is activated to provide an adjusted shift to the curve.

After programming R_{SYN} from the 4-bit stateful XOR operation, the read voltage ($V_{READ}=0.3V$) is applied to the bit-line of R_{SYN} and the word-line is connected back to V_{SS} , as shown in Fig. 6(d). The read current is compared to the reference current (I_{TH}) to determine the syndrome is 0 or 1. The syndrome step is repeated by 3 times to achieve the complete syndrome vector (s1, s2, s3).

To fully validate the ECC functionality, an additional step is included in the testbench. As long as the stateful XOR logic functions correctly, there is no error in the data and the parity bits (d1–d4, p1–p3). If so, the syndrome vector is always (0, 0, 0). So, an error injection step is added after the syndrome phase, and then another syndrome phase is executed to see if the syndrome vector points to the correct error location.

B. Experimental simulation result

Fig. 6 shows the test result from a case with the 4-bit data of

Fig. 7. Syndrome current measurement when the error is injected to d2, d3, and d4, respectively.

0011 and the intentional error injection to the d1. The first and second plots show the bit-line-voltage waveforms of R_{D1} – R_{D4} , R_{P1} – R_{P3} and R_{SYN} , respectively. The third plot shows the shared word-line-voltage waveform. The last waveform shows the current flow of the R_{SYN} , from which the syndrome vector is obtained during the syndrome readout phase. In the first and the third plots, during the first phase (0~4ns), we can see R_{D1} – R_{D4} are written sequentially while the word-line is tied to V_{SS} (0 V) and the bit-lines are driven by -1.0 V or 2.5 V depending on the data polarity. In the second phase (4~7ns), the word-line voltage is established based on R_{D1} – R_{D4} and fired to program correct parity bits to R_{P1} – R_{P3} . In this particular case, the build-up voltages are 1.63V, 1.08V, and 1.08V, respectively, which are mapped to HRS_2 , LRS_1 , and LRS_1 of Fig. 3(b), respectively. From 7ns to 16ns, the syndrome reset, program, and readout are repeated by three times, and each operation are relevant to the syndrome vector elements s1, s2, and s3. During the programming steps, HRS_2 -corresponding voltages (1.55~1.67V) are applied to R_{SYN} . As a result, because R_{SYN} is HRS_2 for all three sub-phases, the read current is always low (~80 μ A), leading to the syndrome vector of (0, 0, 0). After that, the d1 error injection phase starts at 16ns, where the BL1 is driven by +2.5V to bring the RD1 back to the HRS state. From 17ns to 26ns, the array repeats the syndrome phase with the

injected error. Because the d1 is flipped, the syndrome vector should be (1, 1, 0) by the definition of the Hamming ECC, as shown in Fig. 1. In the test, the syndrome programming voltages are 1.13V (LRS_1), 0.94V (LRS_1), and 1.55V (HRS_2), respectively. As a result, the correct syndrome vector of (1, 1, 0) is evidently obtained from the measured current of (495 μ A, 406 μ A, 74 μ A).

Whereas Fig. 6 details the multiple waveforms when the error is injected to d1, Fig. 7 focuses on validating the syndrome vector when the error is injected to d2, d3, and d4, respectively, by observing only the syndrome current. For all cases, the syndrome vector is always (0, 0, 0) before the error injection which verifies the correct parity generation by the in-memory stateful XOR logic. On the other hand, the vector becomes (1, 0, 1), (0, 1, 1), and (1, 1, 1) after the error is injected to d2, d3, and d4, respectively. It proves the correct Hamming ECC function as well as the stateful XOR logic.

Fig. 8 shows the syndrome current over all 4-bit input data cases, with the error injected to d1. No matter which data is written initially, the correct syndrome vector for d1 error of (1, 1, 0) is obtained after the error injection. Also, the vector is always (0, 0, 0) before the error injection. In summary, the results given in Fig. 6 and Fig. 7 verify the ECC function that

Fig. 8. Simulation result when an error is injected to d1 from different data patterns.

points to where the error happens, and the result in Fig. 8 verifies that it functions robustly with enough margin regardless of the stored data.

V. CONCLUSION

This paper presents a novel in-memory Hamming ECC technique in memristor crossbar. A stateful XOR logic based on CRS memristors is proposed to enable the in-memory parity bit generation and syndrome vector calculation. In addition, a two-step (build-up and fire), current-mode bit-line driving scheme allows a linear digital-to-analog conversion within a crossbar, which helps retaining enough voltage margin for robust operation. Complete array-level SPICE simulations prove that the proposed ECC technique functions perfectly with enough margin, across all possible data input and error injection combinations.

REFERENCES

- [1] A. Kawahara et al., "An 8 Mb multi-layered cross-point ReRAM macro with 443 MB/s write throughput," in *ISSCC Dig. Tech. Papers*, Feb. 2012, pp. 432–433.
- [2] T.-Y. Liu et al., "A 130-mm² 2-layer 32-Gb ReRAM memory device in 24-nm technology," *IEEE J. Solid-State Circuits*, vol. 49, no. 1, pp. 140–153, Jan. 2014.
- [3] W. Bae, K. J. Yoon, C. S. Hwang, and D.-K. Jeong, "Extension of Two-Port Sneak Current Cancellation Scheme to 3-D Vertical RRAM Crossbar Array," *IEEE Trans. Electron Devices*, vol. 64, no. 4, pp. 1591–1596, April 2017.
- [4] K. J. Yoon et al., "Double-layer-stacked one diode-one resistive switching memory crossbar array with an extremely high rectification ratio of 10⁹," *Adv. Electron. Mater.*, vol. 3, no. 7, p. 1700152, Jul. 2017.
- [5] S. Yu and P. Y. Chen, "Emerging memory technologies: Recent trends and prospects," *IEEE Solid-State Circuits Mag.*, vol. 8, no. 2, pp. 43–56, Mar. 2016.
- [6] R. Fackenthal et al., "A 16Gb ReRAM with 200MB/s write and 1GB/s read in 27nm technology," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2014, pp. 338–339.
- [7] F. Alibart, E. Zamanidoost, and D. B. Strukov, "Pattern classification by memristive crossbar circuits using ex situ and in situ training," *Nature Commun.*, vol. 4, p. 2072, May 2013.
- [8] Z. Wang et al., "Fully memristive neural networks for pattern classification with unsupervised learning," *Nature Electron.*, vol. 1, pp. 137–145, Feb. 2018.
- [9] D. S. Jeong, K. M. Kim, S. Kim, B. J. Choi, and C. S. Hwang, "Memristors for Energy-Efficient New Computing Paradigms," *Adv. Electron. Mater.*, vol. 2, no. 9, 1600090, Sep. 2017.
- [10] H.-J. Yoo, "Intelligence on Silicon: From Deep-Neural-Network Accelerators to Brain Mimicking AI-SoCs," in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2019, pp. 20–26.
- [11] M. Prezioso et al., "Training and operation of an integrated neuromorphic network based on metal-oxide memristors," *Nature*, vol. 521, no. 7550, pp. 61–64, 2015.
- [12] P. Yao et al., "Fully hardware-implemented memristor convolutional neural network," *Nature*, vol. 577, no. 7792, pp. 641–646, Jan. 2020.
- [13] W. Bae, and K.J. Yoon, "Weight Update Generation Circuit Utilizing Phase Noise of Integrated Complementary Metal–Oxide–Semiconductor Ring Oscillator for Memristor Crossbar Array Neural Network-Based

- Stochastic Learning,” *Advanced Intelligent Systems*, vol. 2, no. 5, p.2000011, May 2020.
- [14] O. Krestinskaya, A. P. James, and L. O. Chua, “Neuromemristive Circuits for Edge Computing: A Review,” *IEEE Trans. Neural Netw. Learn. Syst.*, in press.
 - [15] F. Su et al., “A 462 GOPs/J RRAM-based nonvolatile intelligent processor for energy harvesting IoE system featuring nonvolatile logics and processing-in-memory,” in *Proc. Symp. VLSI Circuits*, Jun. 2017, pp. 260–261.
 - [16] C.-X. Xue et al., “A 1Mb Multibit ReRAM Computing-In-Memory Macro with 14.6ns Parallel MAC Computing Time for CNN Based AI Edge Processors,” in *IEEE ISSCC Dig. Tech. Papers*, Feb. 2019, pp. 388–390.
 - [17] K. J. Yoon, J.-W. Han, and W. Bae, “A Novel Stateful Logic Device and Circuit for In-Memory Parity Programming in Crossbar Memory,” *Adv. Electron. Mater.*, vol. 6, no. 12, p. 2000672, Dec. 2020.
 - [18] D. Ielmini, and H.S.P. Wong, “In-memory computing with resistive switching devices,” *Nature Electron.*, vol. 1, no. 6, pp. 333–343, June 2018.
 - [19] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, “Memory devices and applications for in-memory computing,” *Nature Nanotechnol.*, vol. 15, pp. 1–16, Mar. 2020.
 - [20] K. M. Kim and R. S. Williams, “A family of stateful memristor gates for complete cascading logic,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 66, no. 11, pp.4348–4355, Nov.2019.
 - [21] R. W. Hamming, “Error detecting and error correcting codes,” *Bell Syst. Tech. J.*, vol. 29, pp. 147–160, 1950.
 - [22] R. Naseer and J. Draper, “DEC ECC design to improve memory reliability in sub-100 nm technologies,” in *Proc. IEEE ICECS*, 2008, pp. 586–589.
 - [23] K. Furutani, K. Arimoto, H. Miyamoto, T. Kobayashi, K. Yasuda, and K. Mashiko, “A built-in hamming code ECC circuit for DRAM’s,” *IEEE J. Solid-State Circuits*, vol. 24, pp. 50–56, Feb. 1989.
 - [24] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, “Use ECP, not ECC, for hard failures in resistive memories,” in *Proc. Int. Symp. Comput. Archit.*, 2010, pp. 141–152.
 - [25] M. Lv, H. Sun, J. Xin and N. Zheng, “Efficient Repair Analysis Algorithm Exploration for Memory with Redundancy and In-Memory ECC,” *IEEE Transactions on Computers*, in press.
 - [26] P. Ankolekar, S. Rosner, R. Isaac, and J. Bredow, “Multi-bit error correction methods for latency-constrained flash memory systems,” *IEEE Trans. Device Mater. Reliabil.*, vol. 10, no. 1, pp. 33–39, Mar. 2010.
 - [27] X. Sun, T. Zhang, C. Cheng, X. Yan, Y. Cai, Y. Yang, and R. Huang, “A memristor-based in-memory computing network for Hamming code error correction,” *IEEE Electron Device Lett.*, vol. 40, no. 7, pp. 1080–1083, Jul. 2019.
 - [28] K. J. Yoon et al., “Evolution of the shape of the conducting channel in complementary resistive switching transition metal oxides,” *Nanoscale*, vol. 6, no. 4, pp. 2161–2169, 2014.
 - [29] Z. Sun, E. Ambrosi, A. Bricalli, and D. Ielmini, “Logic computing with stateful neural networks of resistive switches,” *Adv. Mater.*, vol. 30, no. 38, Sep. 2018, Art. no. 1802554.