A Parallel-friendly Majority Gate to Accelerate In-memory Computation

John Reuben
Chair of Computer Science 3 - Hardware Architecture
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
91058 Erlangen, Germany
johnreuben.prabahar@fau.de

Stefan Pechmann
Chair of Communications Electronics
Universität Bayreuth
95447 Bayreuth, Germany
stefan.pechmann@uni-bayreuth.de

***Abstract*—Efforts to combat the 'von Neumann bottleneck' have been strengthened by Resistive RAMs (RRAMs), which enable computation in the memory array. Majority logic can accelerate computation when compared to NAND/NOR/IMPLY logic due to it's expressive power. In this work, we propose a method to compute majority while reading from a transistor-accessed RRAM array. The proposed gate was verified by simulations using a physics-based model (for RRAM) and industry standard model (for CMOS sense amplifier) and, found to tolerate reasonable variations in the RRAMs' resistive states. Together with NOT gate, which is also implemented in-memory, the proposed gate forms a functionally complete Boolean logic, capable of implementing any digital logic. Computing is simplified to a sequence of READ and WRITE operations and does not require any major modifications to the peripheral circuitry of the array. The parallel-friendly nature of the proposed gate is exploited to implement an eight-bit parallel-prefix adder in memory array. The proposed in-memory adder could achieve a latency reduction of 70% and 50% when compared to IMPLY and NAND/NOR logic-based adders, respectively.**

***Index Terms*—Resistive RAM (RRAM), majority logic, majority gate, memristor, 1 Transistor-1 Resistor(1T–1R), von Neumann bottleneck, in-memory computing, compute-in-memory, processing-in-memory, parallel-prefix adder**

I. Introduction

THE movement of data between processing and memory units in present day computing systems is their main performance and energy-efficiency bottleneck, often referred to as the 'von Neumann bottleneck' or 'memory wall'. The emergence of non-volatile memory technologies like Resistive RAM (RRAM) has created opportunities to overcome the memory wall by enabling computing at the residence of data. RRAMs are two terminal devices (usually a Metal-Insulator-Metal structure) capable of storing data as resistance. The change of resistance is due to the formation or rupture of a conductive filament, depending on the direction of the current flow through the structure. The word 'memristor' is also used by researchers to denote such a device, because it is essentially a resistor with memory. Connecting such RRAM devices in a certain manner, or by applying certain voltage patterns, or by modifying the sensing circuitry, basic Boolean gates (NOR, NAND, XOR, IMPLY logic) have been demonstrated in RRAM arrays [1]–[6]. The motivation for such efforts is to perform Boolean operations on data stored in the memory array, without moving them out to a separate processing circuit, thus mitigating the von Neumann bottleneck. Reviews of such in-memory computing approaches are presented in [7], [8]. To construct a memory array using such devices, two configurations are common: 1Transistor–1Resistor (1T–1R) and 1Selector–1Resistor (1S–1R). The 1T–1R configuration uses a transistor as an access device for each cell, isolating the accessed cell from its neighbours in the array. The 1S–1R configuration uses a two-terminal device called a 'selector' which is fabricated in series with the memristive device. The 1S–1R is area-efficient, but suffers from current leakage (sneak–path problem) due to the inability to access a particular cell without interfering with its neighbours [9].

Majority logic, a type of Boolean logic, is defined to be true if more than half of the n inputs are true, where n is odd. Hence, a majority gate is a democratic gate and can be expressed in terms of Boolean AND/OR as $MAJ(a,b,c) = a.b + b.c + a.c$, where a, b, c are Boolean variables. Although majority logic was known since 1960, there has been a revival in using it for computation in many emerging nanotechnologies (spin waves, magnetic Quantum-Dot cellular automata, nano magnetic logic, Single Electron Tunneling). Recent research [10]–[12] has confirmed that majority logic is to be preferred not only because a particular nanotechnology can realize it, but also because of its ability to implement arithmetic-intensive circuits with less gates. It must be emphasized that majority logic did not become the dominant logic to compute because it was more efficient to implement NAND/NOR gate than a majority gate, in CMOS technology. However, with many emerging nanotechnologies, this is not the case anymore, therefore, majority logic needs to be re-evaluated for its computing efficiency. In [13]–[15], majority logic is implemented in RRAM by applying the two inputs of the majority gate as voltages across its terminals, and the initial state of the RRAM (which is also the third input) switches to evaluate majority. Such an approach complicates the peripheral circuitry and is also not parallel-friendly, because two of the three inputs of a majority gate need to be applied as voltages at wordline/bitline (see Fig.1(a)).

In this paper, we propose a majority gate whose structure is conducive for parallel-processing in the memory array. By activating three rows of the array simultaneously, the

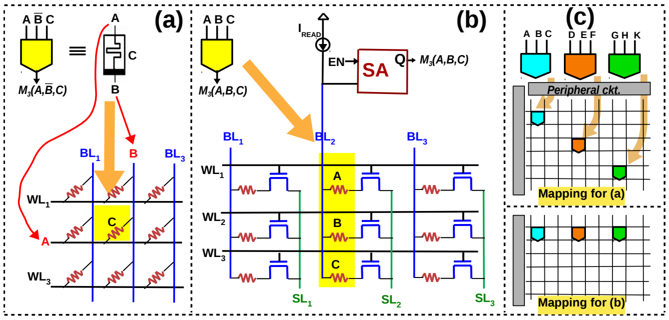

Fig. 1: (a) In-memory majority gate of previous works [13]–[15] (b) Proposed parallel-friendly gate (c) When multiple gates have to be executed in parallel, the majority gates of previous works [13]–[15] have to be mapped diagonally because two gates cannot be executed in the same row/column. This manner of computation complicates **both** the peripheral circuitry and memory controller (inputs of the gates influence row/column decoding). In the proposed method, multiple gates can be mapped to the same set of rows, thereby simplifying the peripheral and the memory controller (inputs of the gates are resistance of memory cells and row/column decoders retain their functionality as in a conventional memory).

resistance of the RRAM cells in a column are in parallel during the READ operation. A Sense Amplifier (SA) which can accurately sense the effective resistance implements a 'in-memory' majority gate. This manner of computing majority enables parallelism and is energy-efficient (both reading and writing is energy-efficient in 1T–1R when compared to 1S–1R arrays due to the absence of sneak paths). To demonstrate the potential of this method to accelerate computation, we consider a parallel-prefix adder and formulate the steps to perform eight-bit addition in a 1T–1R array. The remainder of the paper is organized as follows. Section II-A presents the principle of reading majority from a 1T–1R array. Since the read operation is the crucial aspect of the proposed majority gate, we present the detailed sensing methodology in Section II-B. Further, we study tolerance to variations in resistive states by performing Monte Carlo simulations. In Section III we present the framework to compute in the memory array, using the proposed majority gate. Section IV-A briefly presents parallel-prefix technique and the structure of an eight-bit parallel-prefix adder in terms of majority gates. The adder is then mapped to a 1T–1R array using the proposed in-memory computing technique, in Section IV-B. We compare the proposed eight-bit adder with the state-of-the-art, followed by conclusions in Section V.

II. Majority gate in 1T–1R array

A. Majority gate: Operating principle

Consider an array of RRAM cells arranged in a 1T-1R configuration, as depicted in Fig. 2. Each cell can be individually read/written into by activating the corresponding wordline (WL) and applying appropriate voltage across the cell (BL and SL). To read from a cell, the corresponding WL is activated, a small current is injected into the cell and the voltage across the cell is sensed in a voltage-mode SA $i.e.$ the BL voltage is sensed while the SL is grounded. Now, if three rows are activated simultaneously during read operation (Rows 1 to 3 in Fig. 2), the resistances in column 1 are in parallel (neglecting the parasitic resistance of BL and SL). During read, the access transistor will be in linear region, and hence the transistor's resistance will be $r_{DS} = \frac{1}{\mu_n C_{ox}(\frac{W}{L})(V_{GS}-V_t)} = 544\ \Omega$ [16]. The effective resistance between BL and SL will therefore be $R_{eff} = (R_A + r_{DS})||(R_B + r_{DS})||(R_C + r_{DS}) \approx (\mathrm{R}_A||R_B||R_C)$, if the drain-to-source resistance of transistor (r_{DS}) is small compared to LRS. Table I lists the truth table of 3-input majority gate ($M_3(A,B,C)$) and the effective resistance for all the eight possibilities. To verify the proposed gate on a real RRAM device, we choose the 1T-1R cell from IHP[1]. The 1T–1R structure consists of a NMOS transistor manufactured in IHP's 130 nm CMOS technology, whose drain is connected in series to the RRAM. The RRAM is a $TiN/Hf_{1-x}Al_xO_y/Ti/TiN$ stack integrated between Metal2 and Metal3 in the BEOL of the CMOS process. IHP's 1T–1R cells were modeled using the Stanford-PKU RRAM model following the methodology presented in [16]. The cells have a mean LRS and HRS of 10 KΩ and 133.3 KΩ, respectively. Therefore, the R_{eff} is $\geq$ 8.7 KΩ when two or more cells are in HRS (shaded grey in Table I) and $\leq$ 4.8 KΩ when two or more cells are in LRS. Consequently, a majority gate can be implemented during a READ operation by precisely sensing R_{eff}. As can be deciphered from Table I, the crucial aspect of the proposed gate is to be able to differentiate between R_{eff}^{001} (two LRS and one HRS) and R_{eff}^{110} (two HRS and one LRS). Let's denote the resistance to be differentiated as sensing window,

Sensing window for majority = 8.7 KΩ – 4.8 KΩ = 3.9 KΩ

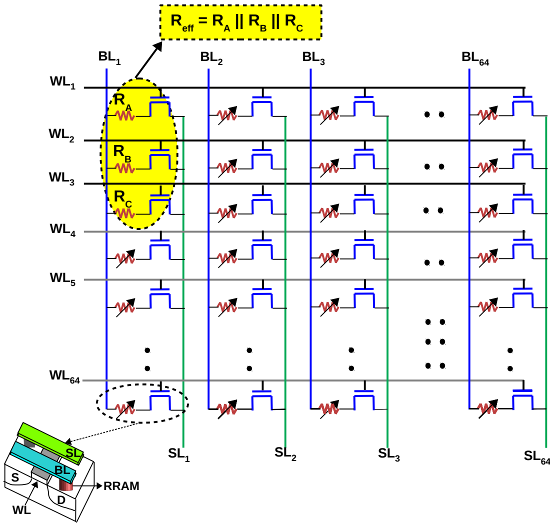

Fig. 2: When three rows are activated (WL_{1-3}) simultaneously in a 1T-1R array, the resistances of the three RRAM devices are in parallel. An 'in-memory' majority gate can be implemented by accurately sensing the effective resistance R_{eff}.

[1] Innovations for High Performance Microelectronics– Leibniz-Institut für innovative Mikroelektronik, Germany

for IHP's cell (resistance window = 13.3).

TABLE I: Precisely sensing R_{eff} results in majority: Logic '0' is LRS (10 KΩ) and logic '1' is HRS (133.3 KΩ)

A	B	C	$M_3(A,B,C)$	R_{eff}	R_{eff}
0	0	0	**0**	$\frac{LRS}{3}$	3.3 KΩ
0	0	1	**0**	$\frac{HRS \cdot LRS}{LRS+2 \cdot HRS}$	4.8 KΩ
0	1	0	**0**	$\frac{HRS \cdot LRS}{LRS+2 \cdot HRS}$	4.8 KΩ
0	1	1	**1**	$\frac{HRS \cdot LRS}{HRS+2 \cdot LRS}$	8.7 KΩ
1	0	0	**0**	$\frac{HRS \cdot LRS}{LRS+2 \cdot HRS}$	4.8 KΩ
1	0	1	**1**	$\frac{HRS \cdot LRS}{HRS+2 \cdot LRS}$	8.7 KΩ
1	1	0	**1**	$\frac{HRS \cdot LRS}{HRS+2 \cdot LRS}$	8.7 KΩ
1	1	1	**1**	$\frac{HRS}{3}$	44.4 KΩ

B. Sensing methodology

As stated, the methodology to reliably translate R_{eff} into a CMOS-compatible voltage is the crucial aspect of the proposed majority gate. R_{eff}^{001} is 4.8 KΩ and R_{eff}^{110} is 8.7 KΩ, and differentiating such a resistance window ($\approx 3.9K\Omega$) needs a robust SA. It must be noted that this will be exacerbated by the variability exhibited by the RRAM devices. To meet this requirement, a time-based SA recently proposed in [17] was chosen. Different from conventional sensing schemes (voltage-mode and current-mode), the time-based sensing scheme converts the BL voltage (to be sensed) into a time delay and discriminates in time-domain. This sensing scheme was originally proposed to read data from STT-MRAM [17], which have a resistance window of a few KΩ. Therefore, it is ideal for the proposed majority gate. Furthermore, this time-based sensing achieves two to three orders of magnitude improvement in sensing (BER) compared to conventional schemes, in addition to being reference-less [17].

The time-based sensing circuit is essentially a voltage-to-time converter followed by a time-domain comparator (D-flip flop). Voltage-to-time conversion is achieved by the current-starved inverter (transistors M_{1-5}) followed by transistor M_6 and an inverter (Fig. 3). During READ, a current I_{READ} is injected into the 1T-1R cell (corresponding three WLs are activated and SL is grounded). Depending on the effective resistance R_{eff}, the BL reaches an appropriate voltage. In the conceptual waveforms of Fig.3, it is assumed that BL gets charged to 300 mV if R_{eff} is a high resistance (8.7 KΩ) and 200 mV if R_{eff} is a low resistance (4.8 KΩ), for the purpose of illustration. Such a V_{BL} (few hundred mV) limits the current flow through the inverter (transistor M_{1-3}), hence the name current-starved inverter. When EN goes high, the current-starved inverter introduces a delay proportional to V_{BL} $i.e.$ a higher V_{BL} incurs less delay. A V_{BL} of 300 mV incurs less delay and low-to-high transition of EN reaches the input of the Flip-flop (I_{FF}) faster $i.e.$ at T_{HRS}. For a lower V_{BL} of 200 mV, the delay is greater and the low-to-high transition occurs at T_{LRS}. t_{delay} is a chain of inverters programmed to introduce a delay between T_{HRS} and T_{LRS}. EN_{delay}, the EN signal delayed by t_{delay} acts as the edge trigger for the D-FF. When EN_{delay} goes high at T_{DM} (Decision Moment), it latches the signal at I_{FF} and hence the D_{out} is high for high resistance (R_{eff}^{110} = 8.7 KΩ) and low for low resistance (R_{eff}^{001} = 4.8 KΩ). It must be noted that for R_{eff}^{111} = 44.4 KΩ, V_{BL} will be much larger than 300 mV and will result in a transition much before T_{HRS}. Similarly, for R_{eff}^{000} = 3.3 KΩ, V_{BL} will be less than 200 mV and will result in a transition much later than T_{LRS}. Once designed to differentiate between R_{eff}^{110} and R_{eff}^{001}, the time-based SA will output $M_3(A,B,C)$ correctly for all the eight cases. Furthermore, the same SA can be used to read a single bit by using a smaller I_{READ} (and activating a single WL during normal read operation). Hence the proposed gate does not necessitate any modification to the read-out circuit of the regular memory array.

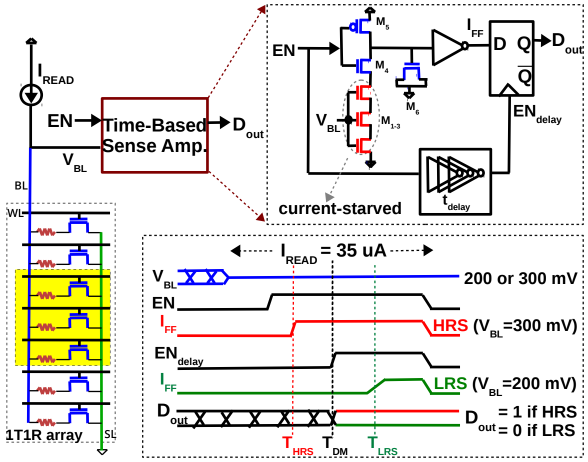

Fig. 3: A small current I_{READ} injected into the cell converts the resistance to a voltage which is fed to the time-based SA. A current-starved inverter transforms this voltage into a proportional delay which is sensed as a CMOS-compatible voltage by the D-FF [17].

The time-based sensing circuit of Fig. 3 was designed in IHP's 130 nm CMOS process, and simulated to verify the functioning of the majority gate. I_{READ} of 35 μA was injected into the 1T-1R cell to sense the BL voltage. For R_{eff}^{001} and R_{eff}^{110}, V_{BL} was 282 mV and 410 mV, respectively. Since the current-starved transistors M_{1-3} are the crucial factor in deciding the delay, they were made large ($\frac{W}{L} = \frac{1.5\mu m}{0.39\mu m}$) to make the circuit less sensitive to CMOS process variations. t_{delay} was set to 3 ns using a chain of inverters with MOS capacitive loads between them. RRAM cells exhibit variability in their programmed resistive states cycle-to-cycle and device-to-device [18]. Therefore the majority gate was evaluated by taking RRAM variations into account. Since majority is computed while reading (memory cell is not switched), the RRAM was replaced with a resistor and variability was incorporated as a Gaussian distribution in that resistor. The impact of process variations was analysed using the statistical model files for the CMOS transistors provided by the foundry. 2000 Monte

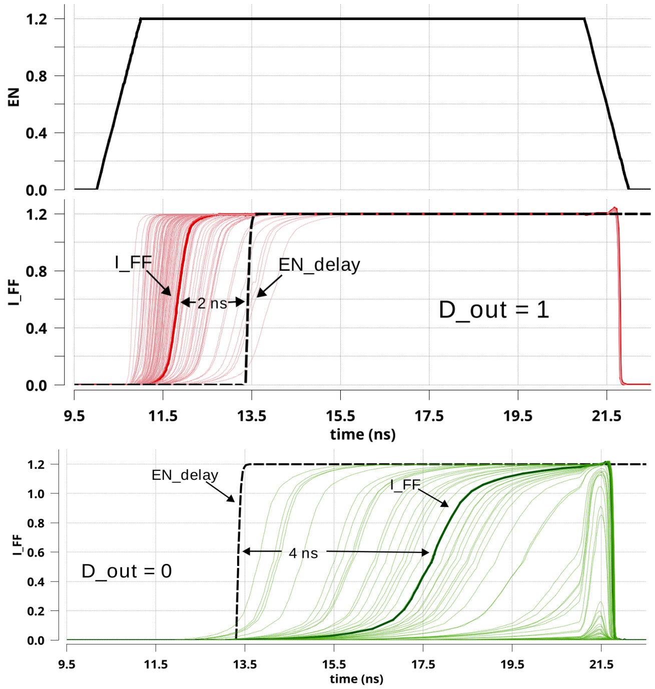

Fig. 4: Sample output of the time-based SA. At 13.5 ns, the EN_{delay} goes high deciding the output. Only 100 MC simulations are plotted (shaded light) with single typical case highlighted dark.

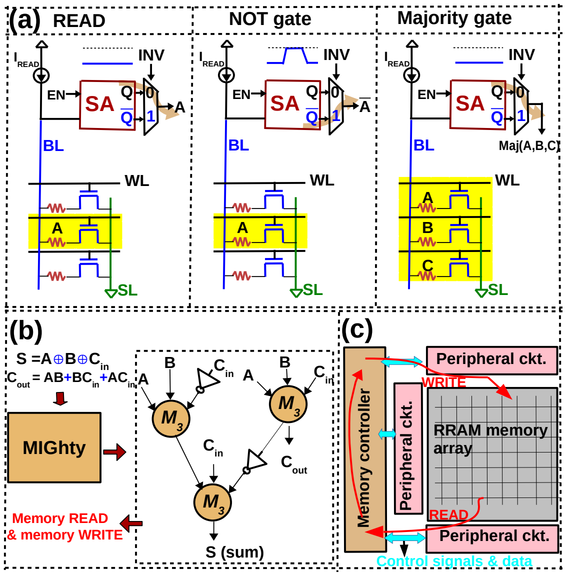

Fig. 5: (a) NOT operation implemented with a 2:1 Mux at the output of the time-based SA; all logic operations are essentially READ operations (b) 1-bit full adder expressed as Majority-Inverter-Graph using MIGhty synthesis tool [19], where M_3 represents 3-input majority operation (c) With majority/NOT gate computed as READ, multiple levels of logic can be executed by writing the data back to the memory, simplifying computing to READ and WRITE operations.

Carlo simulations were performed where the resistance of the RRAM was Gaussian distributed with a standard deviation, σ = 10% of mean RRAM resistance $i.e$ σ_{LRS} = 1 KΩ and σ_{HRS} = 13.33 KΩ. With combined effects of RRAM variability **and** process variability (in transistors of SA), the Bit Error Rate (BER) was found to be 5.4%. Sample wave-forms are plotted in Fig. 4. Further failure analysis of the majority gate (incorrect sensing of R_{eff}^{001} and R_{eff}^{110}) revealed that it occurred only when RRAM variability was more than 2σ from mean LRS/HRS (It must be noted that 95% of resistances fall within 2σ from the mean, in a Gaussian distribution).

III. Framework to compute in 1T–1R array

A. Functional completeness and memory controller

As shown in Fig. 5-(a), NOT operation can be implemented in a 1T–1R array by simply latching $\overline{Q}$ from the output of the time-based SA during READ (D-Flip flop of Fig.3 outputs Q and $\overline{Q}$). This is accomplished by using a control signal INV which is low during READ and majority operation (Q is latched) and goes high only during NOT operation ($\overline{Q}$ is latched). Majority together with NOT is functionally complete $i.e$ any Boolean logic can be expressed in terms of majority and NOT gates [19]. In [19], the authors present Majority-Inverter Graph (MIG), a new logic manipulation structure consisting of three-input majority nodes and regular/inverted edges. Fig.5-(b) is the MIG of a 1-bit full adder obtained by MIGhty (MIG synthesis tool) and, any Boolean logic can be synthesised in terms of majority and NOT gates in a similar manner. Since both majority and NOT gates are implemented as READ, multiple levels of gates can be cascaded by writing the read data back to the array. In essence, ‘computing’ is simplified to a sequence of READ and WRITE operations, orchestrated by the memory controller, as depicted in Fig.5-(c).

The memory controller of a regular memory (be it DRAM-based or NVM-based) is responsible for orchestrating the READ and WRITE operation by issuing the control signals to the peripheral circuitry of the array. In addition, the memory controller must be augmented with additional capability to execute majority and NOT operation. Since both majority and NOT operations are READ operations in this logic family, the controller does not require any major alterations. To execute a majority operation, an additional control signal called MAJ is needed, which is set to logic ‘1’ during majority operation[2] and, the address of the first row (out of three rows in which majority is to be performed) is placed on the row decoder. It must be noted that majority operation is executed on three contiguous bits of data in a column and the triple row decoder of section III-B will not only select the row corresponding to the address placed on the row decoder, but also the next two rows if MAJ is ‘1’. The column address is placed on the column decoder to select the particular column in which majority is executed and the SA is activated to get the output. The NOT operation is the same as the READ operation with the only exception being the controller issues the control signal INV, which goes high to invert the read data at the output of

[2] this signal acts as an additional input to the row decoder, Fig. 6

the SA (Fig. 5-(a)). The control signals activated during logic operations are summarized in Table II.

TABLE II: Control signals for memory and logic operations

Operation	WL	BL	SL	EN(SA)	INV	MAJ
READ	single row activated	to read ckt.	grounded	1	0	0
NOT	single row activated	to read ckt.	grounded	1	1	0
Majority	three rows activated	to read ckt.	grounded	1	0	1
WRITE '0'	single row activated	V_{SET}	grounded	0	0	0
WRITE '1'	single row activated	grounded	V_{RESET}	0	0	0

B. Triple-row decoder design

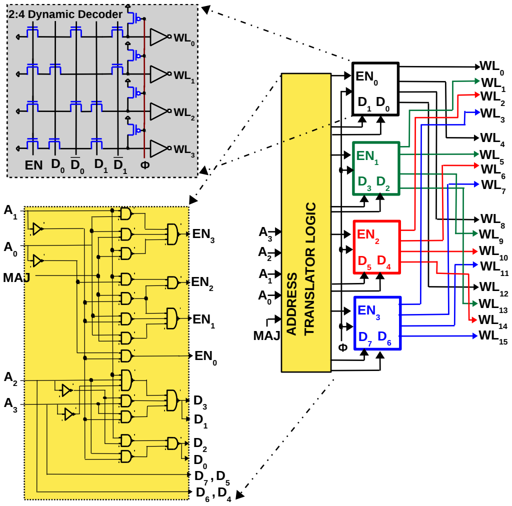

Fig. 6: Triple-row decoding is achieved by interleaving multiple single-row decoders. When control signal MAJ is logic '0' (READ/WRITE/NOT), WL_i corresponding to row address $A_3A_2A_1A_0$ is selected. When MAJ is logic '1' (majority), WL_i, WL_{i+1}, WL_{i+2} are selected.

A conventional decoder for a 1T–1R array can select one row at a time, while the proposed majority gate needs three rows to be selected simultaneously. Moreover, the row-decoder must be versatile to switch between single-row activation and triple-row activation seamlessly. This is because, as stated in the previous section, one must be able to read/write a single bit of the array (READ/WRITE/NOT) as well as read three bits in a column (majority). To this end, we propose a robust row decoder which is designed by interleaving multiple single-row decoders. As depicted in Fig.6, a 4:16 triple-row decoder can be designed by interleaving four 2:4 dynamic $NAND$ decoders[3]. Since single-row decoding must co-exist with triple-row decoding, an address translator circuit is used to switch between the two modes using MAJ as a control signal. For example, to select a single row WL_5, the address is $A_3A_2A_1A_0$ = '0101' and MAJ = '0'. For these inputs, the address translator outputs $EN_3EN_2EN_1EN_0$ = '0010' and $D_7D_6D_5D_4D_3D_2D_1D_0$ = 'XXXX01XX' (green decoder in Fig. 6 is enabled and it's second row is selected, thereby activating WL_5). But, for the same row address $A_3A_2A_1A_0$ = '0101' and MAJ = '1', the address translator outputs $EN_3EN_2EN_1EN_0$ = '1110' and $D_7D_6D_5D_4D_3D_2D_1D_0$ = '010101XX' (blue, red and green decoders are enabled and second row of each of them is selected, thereby activating WL_5, WL_6 and WL_7). The address translator inputs MAJ and $A_3A_2A_1A_0$ and generates $D_7D_6D_5D_4D_3D_2D_1D_0$ and $EN_3EN_2EN_1EN_0$ to achieve this desired functionality for all the 16 cases. With the address translator logic (88 transistors), the triple-row decoder requires 200 transistors, while a regular 4:16 dynamic decoder (only single row activation) requires 136 transistors, a 47% increase in the row-decoder area. The address translator does not add any significant latency to the decoding process. The decoder was designed in 130 nm IHP process and its functionality was verified and decoding latency was found to be 496 ps.

C. Area of time-based Sense Amplifier

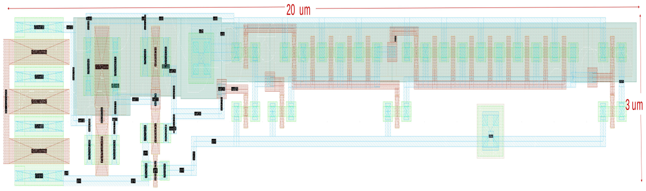

Fig. 7: Layout of time-based SA.

In this work, the primary motivation for pioneering a parallel-friendly gate was to exploit it to accelerate addition, by executing gates in parallel. It must be emphasized that the main drawback of RRAM based in-memory adders is their latency – numerous cycles of Boolean operations (NAND, NOR, IMPLY) are needed to perform addition, when compared to CMOS. To evaluate the number of gates that can be executed in parallel, we evaluated the area of the time-based SA. The time-based SA of [17] could sense the BL voltage without an op-amp, and, this was an important reason for adopting it for our majority gate (conventional SAs use operational amplifier, which consume huge silicon area). The layout of the time-based SA of Fig.3 is drawn in Fig.7 and occupies an area of 20 × 3 = 60 μm^2. It must be noted that this area estimate does not include the area of the delay element since it is shared by all the SA in the array. (t_{delay} in Fig.3 is implemented as series of inverters with MOS capacitive load between them). From [20], the layout of a single 1T–1R cell occupies 450 nm × 450 nm = 0.2 μm^2 in 130 nm (12.4 F^2). If the SA is stacked along its height of 3 μm, eight columns can share a SA. This means that the number of majority gates that can be executed in parallel in an array is the number of columns divided by a factor of 8 $i.e.$ 32 gates can be executed simultaneously in a 256×256 array, 8 gates in a 64×64 array $etc.$

[3] a dynamic decoder uses a precharge signal ϕ, which when low, all WL are driven to '0'. When ϕ goes high, WL_i corresponding to D_1D_0 goes high, provided EN is '1'

D. Energy for in-memory operations

To assess the energy required for computation, we first calculate the energy required for each logic operation. We calculate the energy for a single majority operation, as

$$E_{MAJ} = V_{DD} \int_{0}^{t_{READ}} I_{READ} \cdot dt + V_{DD} \int_{0}^{t_{READ}} I_{SA} \cdot dt \tag{1}$$

where I_{READ} is the current injected into the 1T–1R cell (see Fig. 3), I_{SA} is the current consumed by the time-based SA and t_{READ} is the READ cycle duration. It must be noted that in Eq. 1, t_{READ} was 20 ns and I_{READ} was 35 μA in our simulations in IHP's 130 nm CMOS process. The energy for a single majority operation, E_{MAJ} = 1.98 pJ. The energy for the NOT operation is the same as the energy to read a single bit, and it was calculated to be E_{NOT} = 1.24 pJ. E_{NOT} is smaller than E_{MAJ} because, I_{READ} was smaller (22 μA) for NOT and READ, where a single bit is read. The energy to write a bit, $E_{WRITE} = V_{cell} \times \int_0^{t_{WRITE}} I_{WRITE}.dt$, where t_{WRITE} was 100 ns in our simulation (although switching time is $\leq$ 10 ns for these devices, t_{WRITE} was set to 100 ns to account for worst-case scenarios). E_{WRITE} was calculated to be 11 pJ.

IV. Eight-bit adder in 1T–1R array

A. Parallel-prefix adder using majority logic

Parallel-prefix (PP) adders are a family of adders originally proposed to overcome the latency incurred by the rippling of carry in CMOS-based adders. The regular structure of the memory array and the proposed parallel-friendly majority gate can be combined to implement PP adders in the memory array. PP adders have a 'carry-generate block' followed by a 'sum-generate block' (Fig. 8). The 'carry-generate block' can generate the carry 'ahead' and is known to reduce the latency to O(log n), for n-bit adders. Kogge-Stone, Ladner-Fischer, Brent-Kung and the like, are examples of PP adders. For this work, we choose Ladner-Fischer since it minimizes logical depth at the cost of fan-out (fan-out of a gate translates to WRITE, as will be elaborated in section IV-B). Since majority gate is the basic building block for many emerging nanotechnologies, prior works [11], [12] have formulated such PP adders in terms of majority gates. The carry-generate and sum-generate blocks for an eight-bit adder in majority logic are derived from [11], [12] (Fig. 8). For an eight-bit adder, the logical depth is six levels of majority gates and one level of NOT gates, and at most eight gates are needed simultaneously in each level.

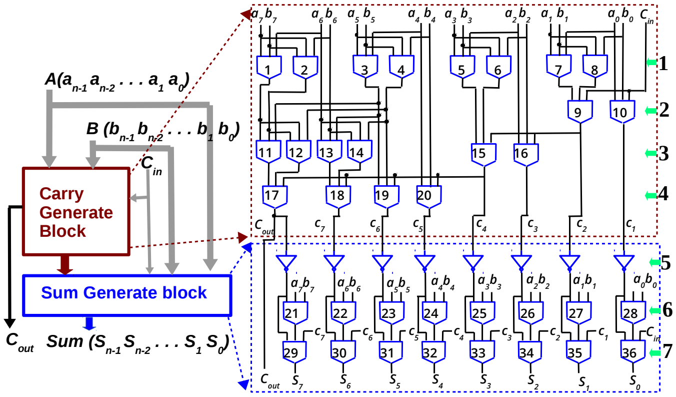
A(a_{n-1} a_{n-2} . . . a_1 a_0)
B (b_{n-1} b_{n-2} . . . b_1 b_0)
C_{in}
Carry Generate Block
Sum Generate block
C_{out}
Sum (S_{n-1} S_{n-2} . . . S_1 S_0)
a_7 b_7 a_6 b_6 a_5 b_5 a_4 b_4 a_3 b_3 a_2 b_2 a_1 b_1 a_0 b_0 C_{in}
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
C_{out} c_7 c_6 c_5 c_4 c_3 c_2 c_1
21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
C_{out} S_7 S_6 S_5 S_4 S_3 S_2 S_1 S_0
1
2
3
4
5
6
7

Fig. 8: Eight-bit PP adder (Ladner-Fischer)expressed as 7 levels of majority and NOT gates [11], [12]. Majority gates 1–20 constitute carry generate block and 21–36 constitute sum generate block.

B. Mapping of the eight-bit LF adder to 1T–1R array

In this section, we map the eight-bit Ladner-Fischer adder structure of Fig. 8 to a 1T–1R array, using the proposed logic family, and elaborate the sequence of operations. Since the proposed gates are not stateful[4], the output of the majority gate (voltage) needs to be written into the array as inputs to the next logic level. We assume a 5×65 processing area (to store the intermediate results of the computation), which is initialized to logic '0', $i.e.$, all cells are in LRS. Further, we assume that the two numbers to be added ($a_7a_6a_5a_4a_3a_2a_1a_0$, $b_7b_6b_5b_4b_3b_2b_1b_0$ and C_{in}) are arranged in the processing area as depicted in Fig. 9. To minimize latency, we map the adder in a way such that all the majority gates in a logic level (see Fig. 8) are executed simultaneously in a READ operation (see Fig. 9). In a 1T–1R array, HRS→ LRS transition (SET process when the conductive filament is created) is accomplished by applying two pulses simultaneously to the WL and BL, while SL is grounded. LRS→ HRS transition (RESET process when the filament is ruptured) is accomplished by applying two pulses simultaneously to the WL and SL, while BL is grounded. This is because a voltage of opposite polarity is needed across the RRAM cell to break the filament. Hence, SET and RESET cannot be performed on the same row simultaneously. Therefore, writing multiple bits to a row is usually done in two steps, $i.e$, to write '1010', first '_0_0' is written by SET process and then '1_1_' is written by RESET process. In our mapping, multiple bits can be written in a single cycle since the 5×65 processing array is initialized to '0'. The contents of the array during the seven levels are depicted in Fig. 9. As enumerated in Fig.9, two eight-bit numbers can be added by a sequence of read and write operations, requiring a total of 19 steps (6 Majority, 2 NOT and 11 write cycles). The proposed approach is one of the fastest implementation of eight-bit adder in RRAM array, with only one other work [2] reporting a lower latency (Table III).

The proposed method naturally enables parallel-prefix addition by 'reading' majority simultaneously from columns of data. Therefore, the number of steps to compute eight-bit addition, in a RRAM array is shortened, as summarized in Table III. For our eight-bit adder, the energy consumption, calculated from simulations, was 631 pJ (36 majority, 8 NOT and 50 WRITE operations). In the Table III, we have not compared the energy for computation since they are either

[4] In memristive logic, a logic family is said to be stateful if both the input and output of a computation are represented as resistance of the RRAM/memristor [7]

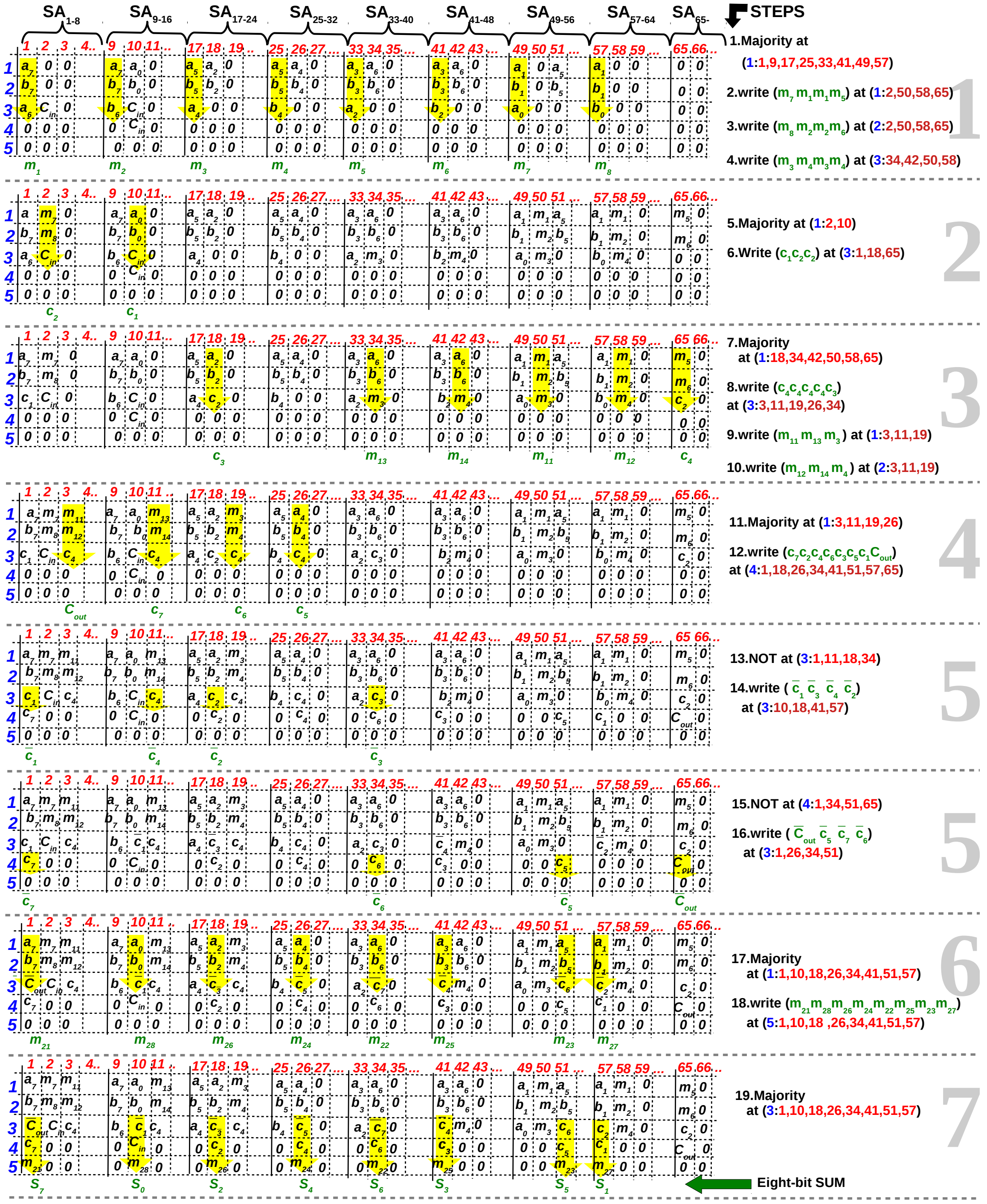
SA$_{1-8}$
SA$_{9-16}$
SA$_{17-24}$
SA$_{25-32}$
SA$_{33-40}$
SA$_{41-48}$
SA$_{49-56}$
SA$_{57-64}$
SA$_{65-}$
STEPS
1.Majority at (1:1,9,17,25,33,41,49,57)
2.write (m_7 $m_1$$m_1$$m_5$) at (1:2,50,58,65)
3.write (m_8 $m_2$$m_2$$m_6$) at (2:2,50,58,65)
4.write (m_3 $m_4$$m_3$$m_4$) at (3:34,42,50,58)
1
5.Majority at (1:2,10)
6.Write ($c_1$$c_2$$c_2$) at (3:1,18,65)
2
7.Majority at (1:18,34,42,50,58,65)
8.write ($c_4$$c_4$$c_4$$c_4$$c_3$) at (3:3,11,19,26,34)
9.write (m_{11} m_{13} m_3) at (1:3,11,19)
10.write (m_{12} m_{14} m_4) at (2:3,11,19)
3
11.Majority at (1:3,11,19,26)
12.write ($c_7$$c_2$$c_4$$c_6$$c_3$$c_5$$c_1$$C_{out}$) at (4:1,18,26,34,41,51,57,65)
4
13.NOT at (3:1,11,18,34)
14.write ($\bar{c}_1$ $\bar{c}_3$ $\bar{c}_4$ $\bar{c}_2$) at (3:10,18,41,57)
5
15.NOT at (4:1,34,51,65)
16.write ($\bar{C}_{out}$ $\bar{c}_5$ $\bar{c}_7$ $\bar{c}_6$) at (3:1,26,34,51)
5
17.Majority at (1:1,10,18,26,34,41,51,57)
18.write ($m_{21}$$m_{28}$$m_{26}$$m_{24}$$m_{22}$$m_{25}$$m_{23}$$m_{27}$) at (5:1,10,18 ,26,34,41,51,57)
6
19.Majority at (3:1,10,18,26,34,41,51,57)
7
Eight-bit SUM
S_7 S_0 S_2 S_4 S_6 S_3 S_5 S_1

Fig. 9: Mapping of the logic levels 1 to 7 of Fig.8 to 1T–1R array. All the majority gates in a level are executed in parallel (shaded yellow). m_i represent the output of the i^{th} majority gate and c_i is the carry generated during parallel-prefix addition (denoted green since it is read as a 'voltage' and then written into the array).

not reported [2] or reported for another RRAM technology [22]. Depending on the RRAM technology in which the adder is implemented/simulated, the energy will differ (switching energy depends on HRS, LRS and switching times which varies from few ns to even 1 μs). Therefore, it would be unfair to compare the energy of computation across different RRAM technologies. However, the latency can be a good measure of energy comparison since, for each logic primitive, we mention what is the operation performed in each step. It is reasonable to expect the proposed adder to require a large array area (5×65) since it executes multiple gates in parallel. However, it must be emphasized that for a holistic area comparison between adders, both the array area (memory cells) and the increased peripheral circuit area must be considered [7].

TABLE III: Comparison of eight-bit adders in RRAM array

Primitive	Array	Latency	Area	Comment/Ref
IMPLY	1S-1R	58 steps	72 cells	Each step is IMPLY operation [21]
NOR	1S-1R	38 steps	19×22	Each step has one or more NOR operations [22]
Majority	1S-1R	48* steps	8×3	Each step is majority (Fig.1 (a)) or READ [22]
OR/AND	1S-1R	37 steps	64 cells	Each step has one or more OR/AND operation [23]
ORNOR	1S-1R	31 steps	54 cells	Each step has one or more ORNOR/IMPLY [24]
Majority+NOT	1T-1R	19 steps	5×65	Each step is majority/NOT or WRITE (this work)
XOR**	1T-1R	16 steps	three 1×8	Each step is XOR/READ [2]

* Latency is calculated as 24 RM_3 (Resistive Majority) instructions in [22], where each RM_3 consists of a READ followed by majority of Fig.1 (a)

** XOR gate of [2] is not parallel-friendly and consequently multiple gates cannot be executed in parallel in the array (to circumvent this, [2] has used multiple arrays). Furthermore, XOR is not functionally complete and has to be used in conjunction with other gates to implement other arithmetic circuits. In contrast, majority+NOT is functionally complete and can be implemented with minimal peripheral overhead in the proposed method.

V. Conclusion

A memristive logic family formulates a functionally complete Boolean logic with a memristive device (RRAM/PCM/STT-MRAM) as the primary switching device. The proposed method of implementing a majority and NOT gate in a 1T–1R array forms a new memristive logic family. The majority gate can be implemented in a 1T–1R array without necessitating any major change in the peripheral circuit (except the row decoder which needs to be modified to activate three rows simultaneously). Majority logic can be combined with parallel-prefix techniques to design fast adders, and the proposed gate can be used to implement them in memory arrays, with minimum latency.

Acknowledgment

John Reuben was partially supported by Emerging Talents Initiative (ETI) of Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU).

References

[1] B. Chen, F. Cai, J. Zhou, W. Ma, P. Sheridan, and W. D. Lu, "Efficient in-memory computing architecture based on crossbar arrays," in *2015 IEEE International Electron Devices Meeting (IEDM)*, Dec 2015, pp. 17.5.1–17.5.4.

[2] Z. Wang, Y. Li, Y. Su, Y. Zhou, L. Cheng, T. Chang, K. Xue, S. M. Sze, and X. Miao, "Efficient implementation of boolean and full-adder functions with 1t1r rrams for beyond von neumann in-memory computing," *IEEE Transactions on Electron Devices*, vol. 65, no. 10, pp. 4659–4666, Oct 2018.

[3] S. Hu, Y. Li, L. Cheng, Z. Wang, T. Chang, S. M. Sze, and X. Miao, "Reconfigurable boolean logic in memristive crossbar: The principle and implementation," *IEEE Electron Device Letters*, vol. 40, no. 2, pp. 200–203, Feb 2019.

[4] B. C. Jang, Y. Nam, B. J. Koo, J. Choi, S. G. Im, S.-H. K. Park, and S.-Y. Choi, "Memristive logic-in-memory integrated circuits for energy-efficient flexible electronics," *Advanced Functional Materials*, vol. 28, no. 2, p. 1704725, 2018.

[5] W. Chen, W. Lin, L. Lai, S. Li, C. Hsu, H. Lin, H. Lee, J. Su, Y. Xie, S. Sheu, and M. Chang, "A 16mb dual-mode reram macro with sub-14ns computing-in-memory and memory functions enabled by self-write termination scheme," in *2017 IEEE International Electron Devices Meeting (IEDM)*, Dec 2017, pp. 28.2.1–28.2.4.

[6] H. Li, Z. Chen, W. Ma, B. Gao, P. Huang, L. Liu, X. Liu, and J. Kang, "Nonvolatile logic and in situ data transfer demonstrated in crossbar resistive ram array," *IEEE Electron Device Letters*, vol. 36, no. 11, pp. 1142–1145, Nov 2015, doi: 10.1109/LED.2015.2481439.

[7] J. Reuben, R. Ben-Hur, N. Wald, N. Talati, A. Ali, P.-E. Gaillardon, and S. Kvatinsky, "Memristive logic: A framework for evaluation and comparison," in *Power And Timing Modeling, Optimization and Simulation (PATMOS)*, September 2017, pp. 1–8.

[8] D. Ielmini and H.-S. P. Wong, "In-memory computing with resistive switching devices," *Nature Electronics*, vol. 1, pp. 333 – 343, 2018.

[9] N. Talati, R. Ben-Hur, N. Wald, A. Haj-Ali, J. Reuben, and S. Kvatinsky, *mMPU—A Real Processing-in-Memory Architecture to Combat the von Neumann Bottleneck*. Singapore: Springer, 2020, pp. 191–213.

[10] L. Amarú, P. Gaillardon, and G. De Micheli, "Majority-based synthesis for nanotechnologies," in *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2016, pp. 499–502.

[11] G. Jaberipur, B. Parhami, and D. Abedi, "Adapting computer arithmetic structures to sustainable supercomputing in low-power, majority-logic nanotechnologies," *IEEE Transactions on Sustainable Computing*, vol. 3, no. 4, pp. 262–273, Oct 2018.

[12] V. Pudi, K. Sridharan, and F. Lombardi, "Majority logic formulations for parallel adder designs at reduced delay and circuit complexity," *IEEE Transactions on Computers*, vol. 66, no. 10, pp. 1824–1830, Oct 2017.

[13] P. Gaillardon, L. Amaru, A. Siemon, E. Linn, R. Waser, A. Chattopadhyay, and G. De Micheli, "The programmable logic-in-memory (plim) computer," in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2016, pp. 427–432.

[14] S. Shirinzadeh, M. Soeken, P. Gaillardon, and R. Drechsler, "Logic synthesis for rram-based in-memory computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 7, pp. 1422–1435, July 2018, doi: 10.1109/TCAD.2017.2750064.

[15] D. Bhattacharjee, A. Easwaran, and A. Chattopadhyay, "Area-constrained technology mapping for in-memory computing using reram devices," in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2017, pp. 69–74.

[16] J. Reuben, D. Fey, and C. Wenger, "A modeling methodology for resistive ram based on stanford-pku model with extended multilevel capability," *IEEE Transactions on Nanotechnology*, vol. 18, pp. 647–656, 2019.

[17] Q. Trinh, S. Ruocco, and M. Alioto, "Time-based sensing for reference-less and robust read in stt-mram memories," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 10, pp. 3338–3348, Oct 2018.

[18] J. Reuben and D. Fey, "A time-based sensing scheme for multi-level cell (mlc) resistive ram," in *2019 IEEE Nordic Circuits and Systems Conference (NORCAS): NORCHIP and International Symposium of System-on-Chip (SoC)*, 2019, pp. 1–6.

[19] L. Amarú, P. E. Gaillardon, and G. D. Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 5, pp. 806–819, May 2016.

[20] A. Levisse, B. Giraud, J. . Noel, M. Moreau, and J. . Portal, "Rram crossbar arrays for storage class memory applications: Throughput and density considerations," in *2018 Conference on Design of Circuits and Integrated Systems (DCIS)*, Nov 2018, pp. 1–6.

[21] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny, and U. C. Weiser, "Memristor-based material implication (imply) logic: Design principles and methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 10, pp. 2054–2066, Oct 2014.

[22] J. Reuben, N. Talati, N. Wald, R. Ben-Hur, A. H. Ali, P.-E. Gaillardon, and S. Kvatinsky, *A Taxonomy and Evaluation Framework for Memristive Logic*. Cham: Springer International Publishing, 2019, pp. 1065–1099.

[23] A. Siemon, S. Menzel, D. Bhattacharjee, R. Waser, A. Chattopadhyay, and E. Linn, "Sklansky tree adder realization in 1s1r resistive switching memory architecture," *The European Physical Journal Special Topics*, vol. 228, no. 10, pp. 2269–2285, 2019.

[24] A. Siemon, R. Drabinski, M. J. Schultis, X. Hu, E. Linn, A. Heittmann, R. Waser, D. Querlioz, S. Menzel, and J. S. Friedman, "Stateful three-input logic with memristive switches," *Scientific Reports*, vol. 9, no. 1, p. 14618, 2019.