electronics

MDPI

Article

Robust Circuit and System Design for General-Purpose Computational Resistive Memories

Felipe Pinto and Ioannis Vourkas *

Department of Electronic Engineering, Universidad Tecnica Federico Santa Maria, Valparaiso 2390123, Chile; felipe.pinto.12@sansano.usm.cl
* Correspondence: ioannis.vourkas@usm.cl

Abstract: Resistive switching devices (memristors) constitute a promising device technology that has emerged for the development of future energy-efficient general-purpose computational memories. Research has been done both at device and circuit level for the realization of primitive logic operations with memristors. Likewise, important efforts are placed on the development of logic synthesis algorithms for resistive RAM (ReRAM)-based computing. However, system-level design of computational memories has not been given significant consideration, and developing arithmetic logic unit (ALU) functionality entirely using ReRAM-based word-wise arithmetic operations remains a challenging task. In this context, we present our results in circuit- and system-level design, towards implementing a ReRAM-based general-purpose computational memory with ALU functionality. We built upon the 1T1R crossbar topology and adopted a logic design style in which all computations are equivalent to modified memory read operations for higher reliability, performed either in a word-wise or bit-wise manner, owing to an enhanced peripheral circuitry. Moreover, we present the concept of a segmented ReRAM architecture with functional and topological features that benefit flexibility of data movement and improve latency of multi-level (sequential) in-memory computations. Robust system functionality is validated via LTspice circuit simulations for an *n*-bit word-wise binary adder, showing promising performance features compared to other state-of-the-art implementations.

Keywords: memristor; resistive switching; resistive RAM; ReRAM; in-memory computing; scouting logic; 1T1R crossbar; memristive ALU

Citation: Pinto, F.; Vourkas, I. Robust Circuit and System Design for General-Purpose Computational Resistive Memories. *Electronics* **2021**, *10*, 1074. https://doi.org/10.3390/electronics10091074

Academic Editors: Giovanna Turvani, Mariagrazia Graziano and Fabrizio Riente

Received: 7 March 2021
Accepted: 26 April 2021
Published: 1 May 2021

1. Introduction

Even though, in CMOS-based computing systems, the von Neumann architecture has been dominant for several decades, given the current pressure of exponentially rising amounts of data, the modern computing systems are calling for major architectural changes [1]. In order to overcome the "von Neumann bottleneck" and the performance mismatch between CPU and memory, the development of computational memories constitutes an emerging alternative approach [2]. Along this direction, resistive switching devices (memristors) organized in dense crossbar arrays to form resistive random-access memories (ReRAM) are considered among the key enabling device technologies [3–5].

ReRAM-based in-memory computing can significantly improve the energy efficiency of computing systems [6]. In this context, there have been several works published lately that demonstrate the possibility of natively realizing logic computations on memristors [7,8]. Such approaches use the data already stored in the resistive state of the memristors involved in computation as logic inputs. While important efforts are also being placed towards the development of synthesis algorithms for in-memory computing architectures [9,10], the next revolutionary step will be the development of an arithmetic logic unit (ALU) entirely based on in-memory logic operations with memristors.

Recently, a functional demonstration towards a fully memristive ALU was shown in Reference [11], implementing fundamental arithmetic functions. However, information

processing was built upon sequential stateful logic operations based on *conditional switching* of memristors [12], which implies important latency constraints and reliability issues, as identified in References [13,14], for MAGIC- and IMPLY-based circuits. Likewise, given that majority and inversion operations together form a functionally complete set, some works have suggested using these primitives for in-memory computations. For instance, the PLiM approach [15] constitutes a general purpose in-memory computing platform, which however supports only sequential computations. In the same fashion, the ReVAMP architecture [16] took a step further, using VLIW instructions to exploit parallel majority computations, and thus managed to reduce the total number of required instructions.

In this work, we present design results not only in circuit-level but also in system-level, towards the realization of viable ALU functionality in 1-transistor–1-resistor (1T1R) ReRAM-based computational memories, within the reach of today's technology. Unlike in Reference [11], all computations are equivalent to *modified memory read operations* and take place entirely in the augmented peripheral circuitry of the ReRAM array, which enables either word-wise or bit-wise access. To this end, we exploit the *scouting logic* concept [17] for fast and switch-less computation of primitive logic operations, and also the majority gate [18] to accelerate certain arithmetic operations. We present alternative circuit implementations for the enhanced sensing circuitry that enables both memory and logic operations, whose performance was evaluated in the presence of device-to-device variability in memristors. Moreover, we build upon the concept of a segmented ReRAM architecture, introduced in Reference [19], to obtain further flexibility in data movement during multi-level (sequential) computations and to achieve a reduction in the number of required computing steps. We describe in detail the hardware modules of the proposed computational memory system and discuss a preliminary set of supported instructions, proper for a resistive ALU. Functionality and robustness of the proposed system is validated through LTspice circuit simulations for n-bit word-wise binary addition, based on which we highlight the flexibility and the performance benefits, compared to other published works. The presented results pave the way towards the robust design and implementation of next-generation ALUs in ReRAM-based computational memories.

2. Memory Array Topology and In-Memory Logic Schemes

2.1. Definitions and Assumptions for Memristors

In this work, without loss of generality, we assume threshold-type switching bipolar memristors that store a logic "0" with a high resistive state (R_{OFF} or HRS) and logic "1" with a low resistive state (R_{ON} or LRS) [20], as shown in Figure 1a. According to Figure 1b,c, a SET process (HRS $\rightarrow$ LRS) occurs when the device is forward biased with a voltage of amplitude higher than a V_{SET} threshold, whereas a RESET process (LRS $\rightarrow$ HRS) occurs when it is reverse-biased with a voltage amplitude higher than a $|V_{RESET}|$ threshold. Applied voltages of amplitude lower than such thresholds do not affect the device state and are thus used for memory read operations. Device polarity is defined by the thick black line in bottom electrode (BE) of the memristor symbol in the circuit schematics. All simulation results were based on the model of Yakopcic et al. [21]. It is a threshold-type model of a voltage-controlled memristor belonging to the *hyperbolic sine models* and can capture rich switching dynamics, while it also supports nonlinear HRS and LRS states. The model was tuned with parameter values selected so as to demonstrate switching time in the ns-regime and memristance ratio HRS/LRS = 10^6, being in accordance with experimental results in Reference [22] for amorphous silicon (a-Si)-type memristors.

2.2. One Transistor One Memristor (1T1R) Crossbar Array

Figure 2 shows the design of a $m \times n$ transistor–memristor (1T1R) crossbar, which we assume in this work as memory sub-array. It consists of m wordlines and n bitlines with group-accessed transistors as cross-point selector devices to mitigate sneak path currents, which otherwise severely affect the performance of passive (selector-less) arrays [23,24]. Every wordline, WL_i connects to the gate terminal of all the select transistors in the same

crossbar row, so as to simultaneously select/enable all the n memristive cross-points in a memory word. Each bitline BL_j drives all the m cross-points found in the same column of the array, whose memristors have their bottom electrode (BE) commonly connected to a crossbar output line (OL). Through the sensing circuitry, every OL is selectively connected either to logic components or to ground. Depending on whether a memory (read/write) or a logic operation is performed, the wordline decoder will activate simultaneously between one and three wordlines, while the target bitlines are driven accordingly with the corresponding read/write voltage pulses (or otherwise are left floating).

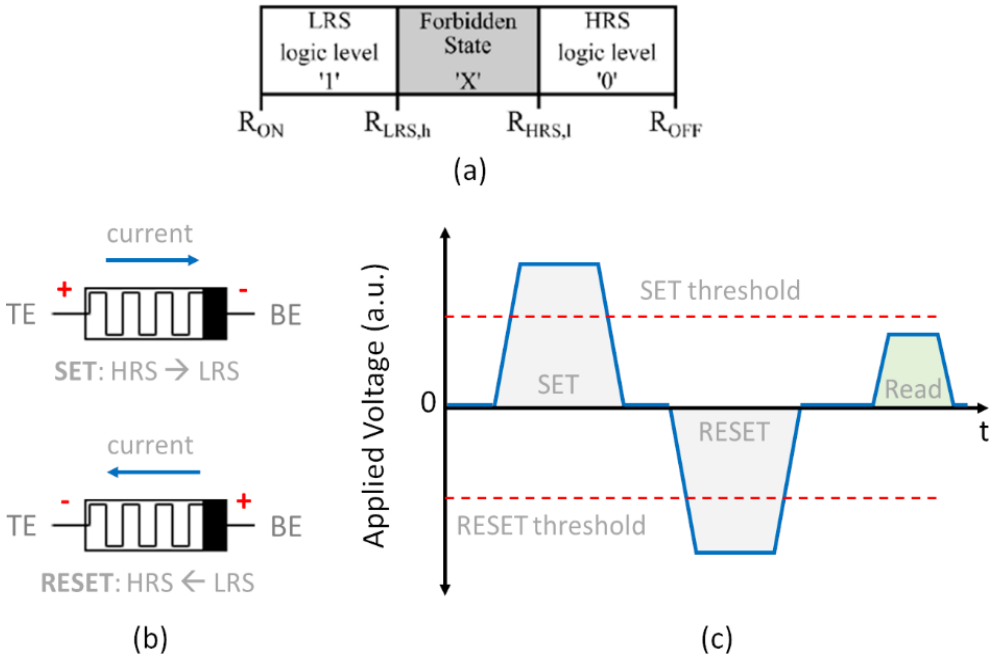


Figure 1. (**a**) Memristance range and HRS/LRS correspondence with logic "0"/"1". Intermediate forbidden state is defined by the highest LRS and the lowest HRS, respectively. (**b**) Memristor switching behavior when forward or reverse-biased. TE/BE stand for Top/Bottom Electrode. (**c**) Cartoon plot showing the required voltage pulses, which are higher than the switching thresholds, to be applied for SET/RESET memory write operations (light gray shade), and pulses of lower amplitude used for memory read operations (light green shade). Dashed horizontal lines denote the SET/RESET thresholds.

2.3. Sensing Circuit Implementations That Enable Memristor-Based Logic Operations

Many logic design schemes in the literature are compatible with the 1T1R crossbar-array memory architecture, shown in Figure 2, using the data stored in the resistance of memristors as inputs to the primitive logic gates. Operation of most such logic circuits is usually based on the voltage divider concept and on proper thresholding to produce correctly the logic output, such as in References [13,25], while operating the memory array in read mode. Generally, a suitable logic style should be tolerant to memristor variability and also independent of particular memristor device technology features.

To this end, here we exploit the *scouting logic* approach [17]. In such scheme, computations take place directly in the enhanced readout circuitry in the form of *modified read operations*, avoiding any conditional switching of the involved memristors. More specifically, a voltage divider is formed between the equivalent parallel resistance of the input memristors (which connect to a common BL_i) and a network of pull-down resistors in the sense amplifier (SA_i). This scheme requires that the SA_i, which is connected to the crossbar OL_i, supports reconfigurable reference voltages. In this direction, Figure 3 shows a voltage-based SA circuit that complies with this requirement. It was originally proposed in Reference [17] to enable memory read operations, as well as 2-input AND/OR/XOR logic

operations. In fact, a read voltage pulse (of amplitude lower than the SET threshold of the memristors) is applied to the target bitlines while activating two wordlines for two-input logic gates. Note that there is no output memristor in this scheme: the logic output is not directly stored in a memristor during the logic operations. Instead, the logic output is the voltage at the output node OL of the aforementioned voltage divider. This constitutes a major departure from stateful logic styles, such as IMPLY or MAGIC, which indeed subject a properly initialized output memristor to a "conditional write" operation [8,13,14]. Moreover, since memory/logic output data are represented in voltage, if required, the output can be stored back to any memory element(s) right afterwards via a reliable memory write operation. Thus, conducting chained operations assumes an intermediate write step to store the output of one stage to a memory location, so that is can later be used as input to subsequent stages. At first glance, such a *read* + *write* operation sequence impacts negatively logic latency. Nevertheless, as shown in the following sections, a properly designed memory module can allow for the simultaneous writing of the memory/logic readout result to the crossbar array (i.e., "a write while reading concept").

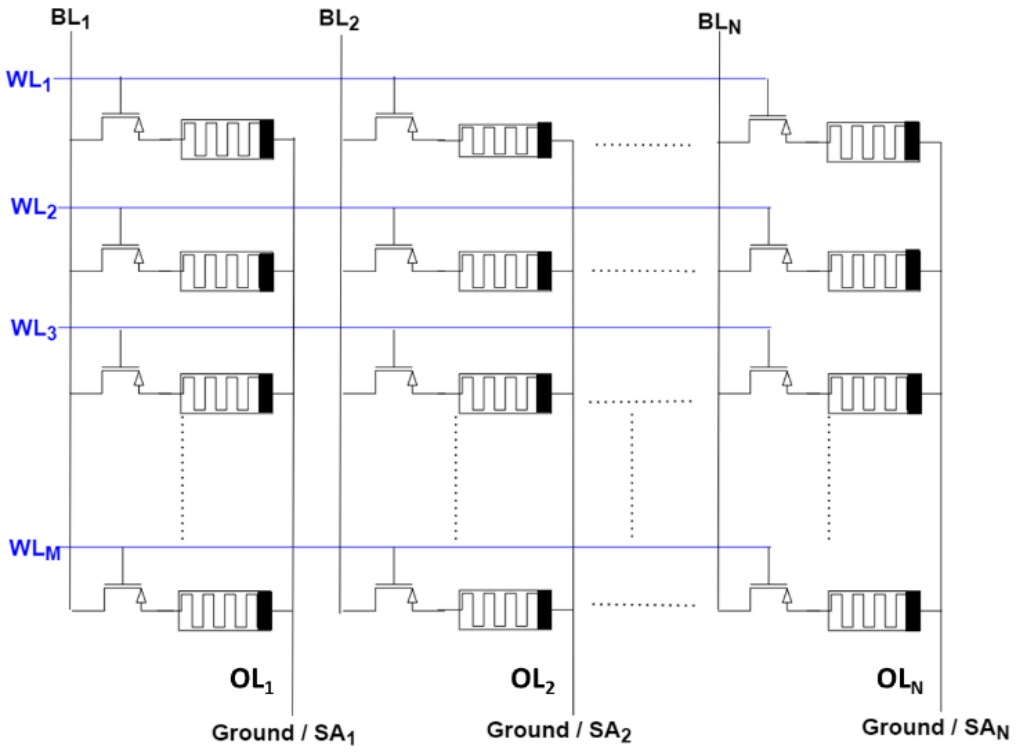


Figure 2. Circuit schematic of a 1T1R crossbar array. WL_i denotes wordlines, whereas BL_i denotes bitlines. The bottom electrodes (BE) of memristors (denoted by the black thick line) in every column output line (OL) connect to a sense amplifier, SA_i, or to ground. (Different color is used for crossing WL and OL lines that are not connected.)

According to Figure 3, in the case of an OR gate, only the transistor S_{r1} is conducting, pulling V_{IN2} to ground, whereas V_{IN1} results from the voltage divider between the two input memristors and resistor R_1. The value of R_1 was selected such that, when at least one of the memristors is in LRS, V_{IN1} will be high enough to be interpreted as logic "1" by the CMOS XOR gate and thus produce a logic "1" output. Note that, for a memory read operation, the SA function is practically equivalent to a logic OR gate but with only one input. Similarly, for an AND gate, there are two pull-down resistors connected in parallel, so that only when both input memristors are in LRS will the V_{IN1} voltage be high enough to cause a logic "1" output. An XOR logic operation is realized by the SA if only the transistor S_{r3} is conductive. In such a case, the series combination of resistors R_1 and R_3 is activated, with V_{IN2} being now equal to the voltage on resistor R_3. When

both memristors are in HRS (LRS), both V_{IN1} and V_{IN2} are low (high) and equivalent to logic "0" (logic "1"). Thus, only when one of the memristors is in LRS does the CMOS XOR gate give a logic "1" output. For AND, OR, and XOR logic operations with more than 2 inputs, the required values for the pull-down resistors might have to be re-calculated. However, by using the exact same SA configuration as for the AND gate while activating a third input memristor (thus a third WL in the crossbar), we figured out that the same circuit can implement a three-input majority (MAJ) logic operation. Certainly, MAJ can be implemented in different ways, e.g., by comparing the current through the crossbar OL with a current threshold, as in Reference [18]. MAJ is worth being considered in such computational memory as it can accelerate certain tasks in arithmetic operations. Therefore, it is important that the considered SA circuit is able to implement MAJ computation as well. For readability reasons, Table A1 in Appendix A presents all possible SA configurations with their equivalent circuits, along with the mathematical expression describing the resulting voltage inputs applied to the CMOS XOR gate.

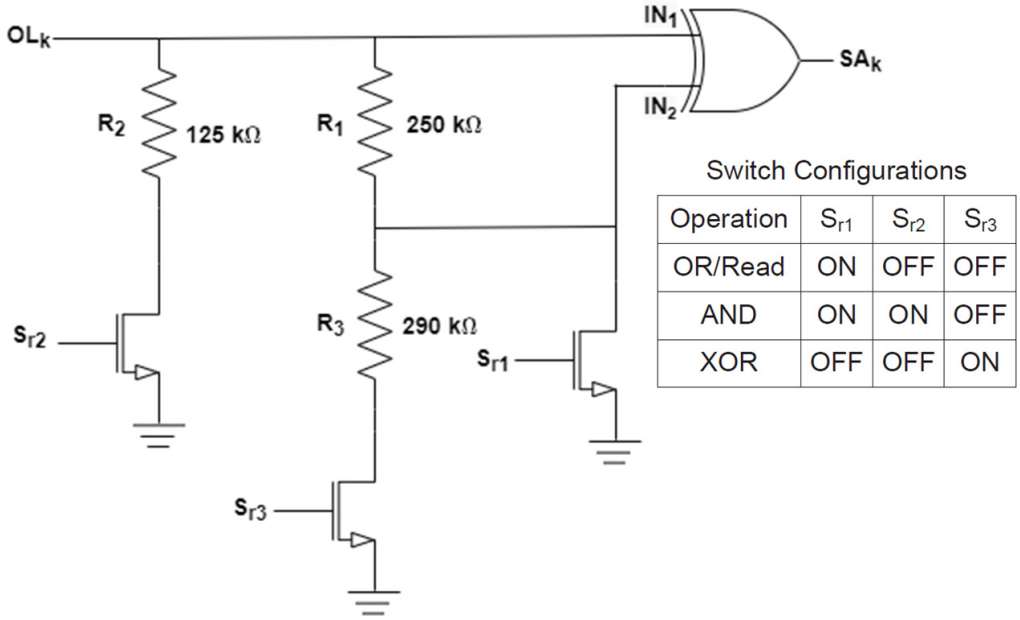


Figure 3. Circuit schematic of voltage-based sense amplifier for scouting logic (adapted from Reference [17]). Inset shows the different possible configurations to activate different voltage reference values at $IN_{1,2}$ terminals. Resistor values were selected based on simulation results, assuming 2-input AND/OR/XOR logic operations, and memristors with HRS/LRS = 125 GΩ/125 KΩ.

All in all, such SA implementation is adequate not only for memory read operations but also because it enables a plurality of primitive logic gates that form the basis for more complex arithmetic operations. However, owing to the underlying voltage divider effect, we figured out that a change in the logic state of any of the input memristors, although it affects the equivalent input memristance R_{OLk} (see Table A1), only leads to slightly modified voltage at the input nodes of the CMOS XOR gate. So, if the inherent variability of HRS and LRS of memristors affects the resulting $V_{IN1,2}$ input voltages to a similar degree, this could potentially lead to erroneous logic computations at the CMOS XOR gate.

In this context, an *enhanced scouting logic* scheme was proposed in Reference [26], but it used a more complex 1T1R array to achieve higher reliability of logic operations. In the same direction, inspired by the crossbar interface circuit proposed by Papandroulidakis et al. in Reference [27], here we designed and evaluated the performance of an alternative and more flexibly parameterizable circuit implementation for the voltage-based SA, which can lead to a more robust behavior against HRS and LRS variability. More specifically, the proposed circuit shown in Figure 4 has a summing amplifier, followed by an inverting amplifier and a set of high-speed voltage comparators in the final stage with configurable

thresholds. We clarify that this represents an alternative system-level SA concept, but yet not a compact and competitive circuit design solution, given the much larger circuit area it occupies.

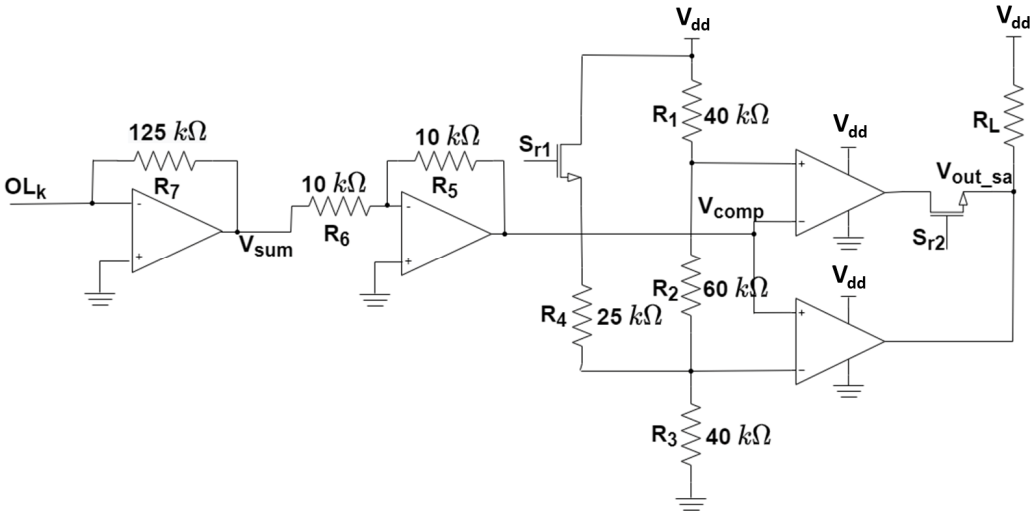


Figure 4. Circuit schematic of an alternative voltage-based sense amplifier for scouting logic. It consists of a summing amplifier, an inverting amplifier, and a configurable comparator stage (far right side of the schematic). Resistor values were selected based on simulation results, assuming 2-input AND/OR/XOR logic operations and memristors with HRS/LRS = 125 GΩ/125 KΩ.

The operation of such a circuit is as follows: through the summing amplifier, we compute a weighted sum of the read voltage, V_{read}, which is commonly applied to all memristors connected to the same crossbar OL. Depending on the SA configuration, different comparisons are enabled based on different threshold voltages, owing to the configurable resistive network ($R_{1–4}$). For example, in case of MAJ, the combination "HRS, HRS" = "00" will produce a very small voltage sum, whereas "HRS, LRS" = "01" (or "10") will give a higher voltage sum, and "LRS, LRS" = "11" will result in the highest voltage sum, which we compare with a voltage threshold in the final stage. For higher reliability, in this case, the resulting voltage threshold should be ideally located in the middle point between the value corresponding to having two memristors in LRS versus having only one memristor in LRS. For both SA circuits, the resistor values were selected based on simulation results to maximize the reliability of the supported logic operations.

For readability reasons, Table A2 in Appendix A presents all possible SA configurations and their equivalent circuits, along with the mathematical expressions describing the resulting voltage inputs applied to the voltage comparator stage. As in Table A1, we again observe here the similarities in the circuit implementation used for Read-OR and for AND-MAJ operations, respectively. For further clarity of circuit performance, we present in Table 1 the $V_{IN1,2}$ input voltages of the CMOS XOR gate of the circuit shown in Figure 3, calculated by using the equations presented in Table A1 for all possible combinations of the input data, expressed in HRS and LRS values. Likewise, Table 1 also presents the resulting input voltage applied to the comparator(s) stage (V_{comp}) and the configurable thresholds ($V_{th,1,2}$) for the circuit shown in Figure 4, calculated by using the equations presented in Table A2. By observing the data, it can be figured out that, indeed, in the alternative SA implementation, a change in the logic state of any input memristor has a much higher impact on the voltage representing the weighted sum in the alternative SA (V_{comp}), compared to the change induced to the output of the voltage divider (V_{IN1}) which is applied to the input nodes of the CMOS XOR gate in the original SA implementation.

Table 1. Calculated voltages at nodes of interest of the SA modules for all possible configurations.

Operation	Input Resistance			Original SA		Alternative SA			Logic Output
	R_1 (KΩ)	R_2 (KΩ)	R_3 (KΩ)	V_{IN1} (V)	V_{IN2} (V)	V_{comp} (V)	V_{th1} (V)	V_{th2} (V)	
Read	125	X	X	0.6	0	0.9	0.571	X	1
Read	125×10^6	X	X	1.8×10^{-6}	0	9×10^{-7}	0.571	X	0
OR	125	125	X	0.72	0	1.8	0.571	X	1
OR	125	125×10^6	X	0.6	0	0.9	0.571	X	1
OR	125×10^6	125×10^6	X	3.6×10^{-6}	0	1.8×10^{-6}	0.571	X	0
AND	125	125	X	0.514	0	1.8	1.333	X	1
AND	125	125×10^6	X	0.36	0	0.9	1.333	X	0
AND	125×10^6	125×10^6	X	1.2×10^{-6}	0	1.8×10^{-6}	1.333	X	0
XOR	125	125	X	0.8	0.433	1.8	0.571	1.429	0
XOR	125	125×10^6	X	0.73	0.37	0.9	0.571	1.429	1
XOR	125×10^6	125×10^6	X	7.8×10^{-6}	4.17×10^{-6}	1.8×10^{-6}	0.571	1.429	0
MAJ	125	125	125	0.6	0	2.7	1.333	X	1
MAJ	125	125	125×10^6	0.514	0	1.8	1.333	X	1
MAJ	125	125×10^6	125×10^6	0.36	0	0.9	1.333	X	0
MAJ	125×10^6	125×10^6	125×10^6	1.8×10^{-6}	0	2.7×10^{-6}	1.333	X	0

Note: X means a value is not required. V_{th1} is equivalent to V_{th} in Table A2 when there is one threshold.

2.4. Performance Comparison in Presence of HRS and LRS Variability

We subjected the two alternative SA implementations, as shown in Figures 3 and 4, to a series of tests in order to evaluate the robustness of logic and memory read operations, while incorporating a certain percentage of variability to the expected HRS and LRS values of the input memristors. More specifically, instead of using fixed HRS/LRS = 125 GΩ/125 KΩ memristance values, the latter represented the mean values of Gaussian distributions. We tested all operations shown in Table 1 for all possible logic input combinations, each time taking 100.000 random samples from the HRS and LRS distributions. Using the equations presented in Tables A1 and A2, while applying V_{read} = 0.85 V and V_{dd} = 2 V, and assuming 0.4 V as threshold voltage for the CMOS XOR gate [17], we calculated the resulting voltages at the nodes of interest of both SA modules. Figure 5 shows the evaluation results, wherein an error corresponds to an erroneous logic output at the SA circuits for a given input combination. We repeated the tests for an increasing SD of HRS and LRS distributions. The results in Figure 5a,b concern 10% and 20%, respectively.

By observing the results in Figure 5, our conclusions for the two alternative SA circuits are as follows:

- Both circuits are robust for memory read and OR logic operations.
- The original scouting SA presents an increasing error percentage up to 20% in MAJ operations when we apply input combinations with only one logic "1" (i.e., "001", "010", and "100"). This is attributed to the fact that the V_{IN1} value for nominal HRS and LRS values (0.36 V in Table 1) is very close to the threshold of the CMOS XOR gate. On the contrary, observed errors in the proposed circuit reach up to 3% for the same input combination when 20% SD is considered.
- The original scouting SA presents an increasing error percentage for the AND operation up to 21% when we apply input combinations with only one logic "1" (i.e., "01" and "10"), whereas the observed error in the proposed alternative circuit generally does not exceed 3% when 20% SD is considered.
- The most error-prone logic operation is XOR, for which the original scouting SA presents errors up to 33% when we apply input combinations with only one logic "1" (i.e., "01" and "10"). On the contrary, the observed errors in the proposed alternative SA topology generally do not exceed 2% when 20% SD is considered.

All in all, it can be figured out that the proposed alternative SA circuit concept is very robust for memristance variability with up to 10% of SD with practically 0% error in all cases, whereas error reached up to 3% when 20% of SD was assumed. Generally, such a small error rate can be addressed by properly engineering the threshold voltages in the comparator stage. However, similar corrections are more difficult to achieve in the original Scouting SA, thus leaving a much smaller space for improvements, given

that its operation is based on the voltage divider concept. It is worth noting that further tests with smaller memristance ratio values, reaching down to HRS/LRS = 10 (not shown in Figure 5), resulted in even worse performance for the original scouting SA for the AND, XOR, and MAJ operations, thus underlying the importance of the wide resistance window of memristors for the design and operation of the Scouting SA circuit. For instance, if the V_{IN1} voltage at the CMOS XOR gate terminal for nominal HRS and LRS is very close to the switching threshold of the CMOS XOR gate, then the slightest perturbance of the input memristance can result in mostly erroneous behavior. Therefore, in the rest of this work, we exploit the proposed alternative SA circuit within a novel computational ReRAM architecture.

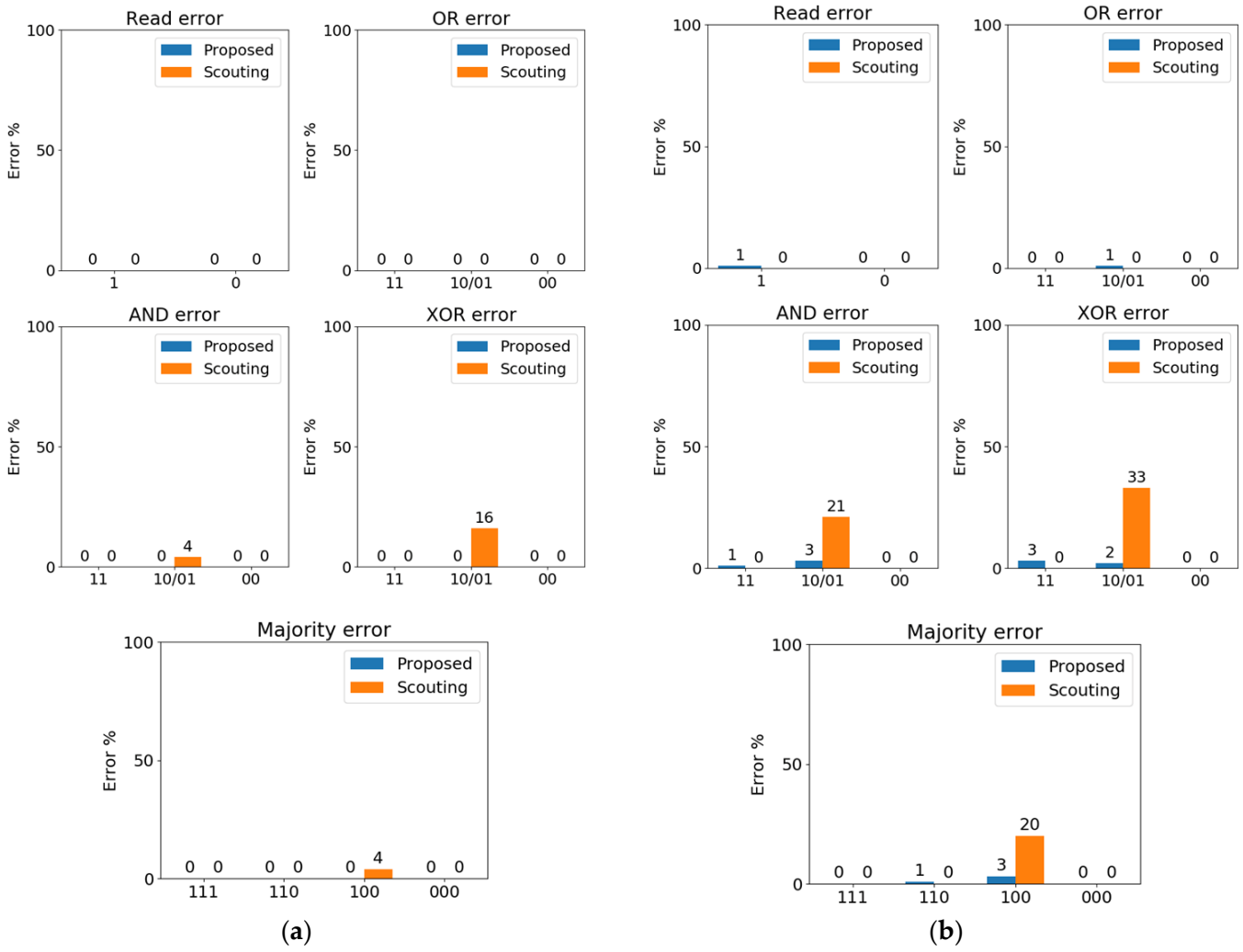


Figure 5. Performance evaluation results for the original "Scouting" SA and the "Proposed" SA circuit, showing the error percentage observed in 100.000 samples for each input combination, for memory and logic operations. Results concern (**a**) 10% and (**b**) 20% SD for the HRS and LRS distributions with mean values HRS/LRS = 125 GΩ/125 KΩ.

3. The "Twin" Computational ReRAM Architecture

3.1. Overall Design Description

Figure 6 presents an overview of the proposed computational ReRAM, whose notion was first introduced in Reference [19]. Its symmetric structure consists of two *"twin" 1T1R crossbar sub-arrays*, each one with dedicated independent row decoders, column drivers, and sense amplifiers. Such a combination of two crossbar sub-arrays was inspired by Reference [27], wherein heterogeneous crossbar banks were used for logic and memory

operations. Assuming that these two sub-arrays have the same dimensions, then each one holds half of the total computational ReRAM storage capacity.

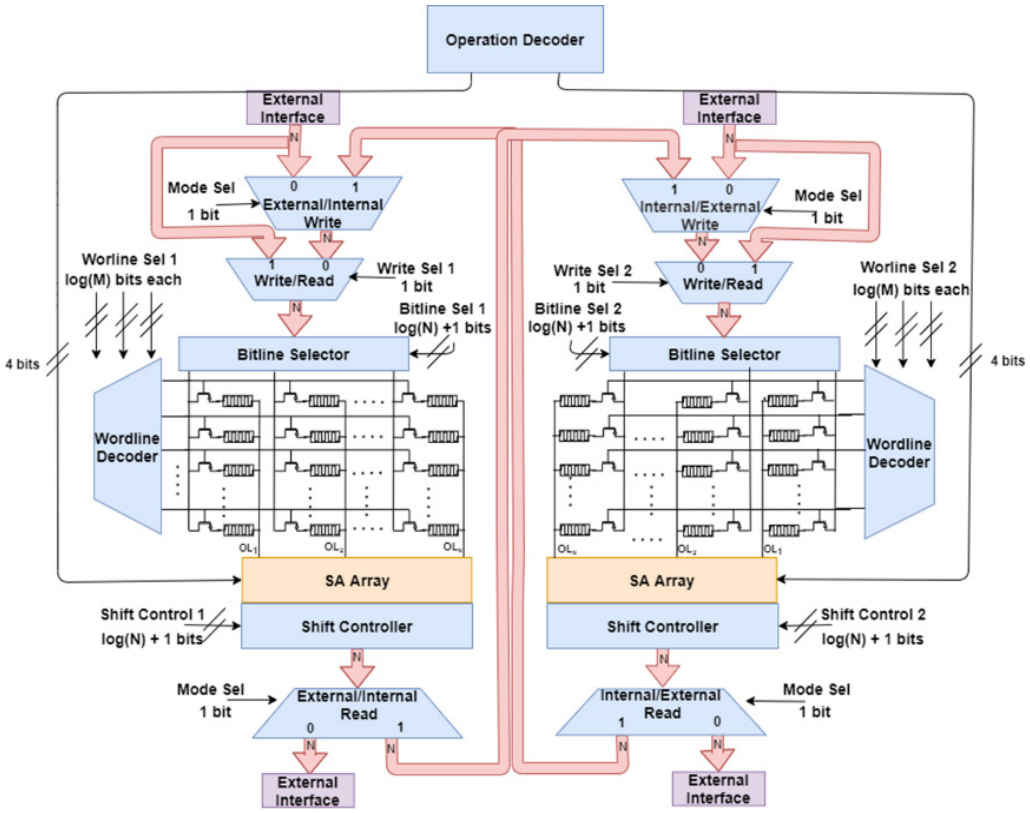


Figure 6. Block level description of the proposed computational memory, consisting of a symmetric segmented structure with two "twin" 1T1R crossbar sub-arrays with dedicated peripheral circuitry and control signals. Adapted from Reference [19].

By observing the peripheral circuitry in Figure 6, at the top of each sub-array, we distinguish the Operation Decoder and the read/write Drivers, along with a Bitline Selector module. At the bottom of the sub-arrays, there is the readout/sensing circuitry (SA Array), along with a Shift Controller, which is used in arithmetic operations and offers flexibility in data storage. The internal/external MUX/DEMUX modules in the write/read interface define whether each sub-array will operate independently (i.e., to write externally applied input data to a sub-array, or to read directly from a sub/array towards the external output) or if the read output of one sub-array is to be simultaneously written to the other. This is defined by the state of the *mode sel bit* in the write drivers. While operating independently, reading/writing from/to each sub-array can be executed simultaneously. On the contrary, when aiming to write the read output data from one sub-array to the other one, a data bus connects the readout stage of each sub-array to the write drivers of the adjacent one. In such a case, the read logic values act as selection signals in the external/internal write driver MUXes of the adjacent sub-array, to select the corresponding SET/RESET voltage to be applied to the top electrode (TE) of the target memristors through the bitlines.

The Bitline Selector in each sub-array allows us to operate either *word-wise* (read or write from/to a complete word) or *bit-wise* (read or write from/to a single bit). When only one bitline is to be accessed, the one indicated by the selection bits is driven and the rest are left floating. Depending on whether we perform a memory or logic operation, the wordline decoders activate up to three wordlines. Note that, for logic operations, the input addresses need to refer to the same sub-array, whereas the output address can point to either sub-array. In the following sections, we show that such segmented design of the ReRAM array benefits the execution of successive in-memory *Scouting logic* computations

by making possible the simultaneous storage of intermediate logic output results to target addresses in the adjacent sub-array, in the same cycle/step. Note that a similar concept was used in Reference [28] to facilitate logic accumulation, required in the memristor overwrite logic (MOL) style.

3.2. Hardware Modules for Bit/Word-Wise Memory and Logic Operations

In this section, we shed light on each one of the modules composing the system shown in Figure 6. At the readout stage, there is an array of n identical sensing modules (SA Array). Figure 7 shows a block level description of one such module, which connects to a crossbar output line (OL_k). Its functionality is configured via four control bits. Two of them are used as configuration bits of the SA circuit, as shown in Figure 4, for memory (Read) or logic operations. Another bit (Output Op Sel) is used in the top DEMUX to define whether the crossbar OL_k will connect to the SA circuit, or to ground, which is necessary for memory write (SET/RESET) operations. One last bit is used as selection line in the output MUX shown at the bottom, to make readily available the *inversion* of the memory/logic result. The basic logic functions supported by the system (AND, OR, NOT, XOR, and MAJ) and their complements form a basis for more complex arithmetic operations.

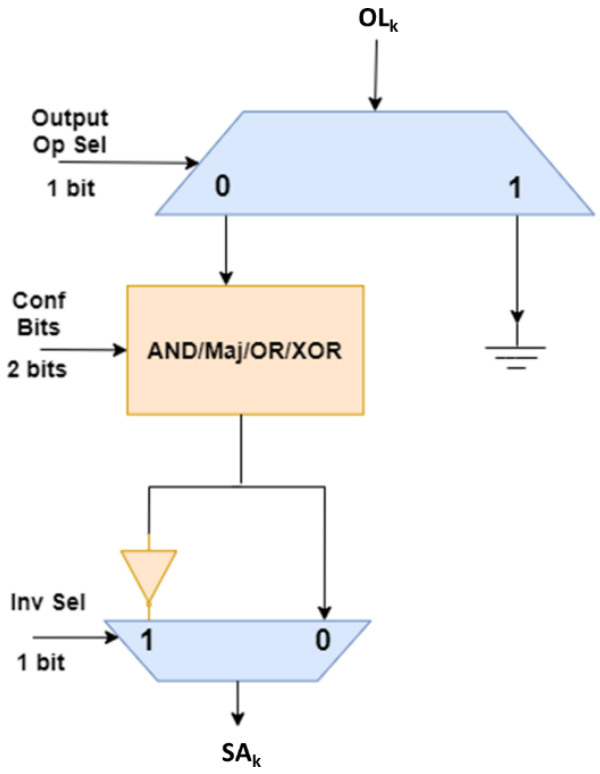


Figure 7. Compact block-level description of the configurable voltage-based sensing module, which connects to a crossbar output line OL_k (BE of memristors), enabling either memory or logic operations. The block entitled "AND/MAJ/OR/XOR" corresponds to the SA circuit in Figure 4.

The output of the SA Array is selectively connected to the external interface, or to the write drivers of the adjacent sub-array, via the Shift Controller. The latter consists of a MUX-based implementation, as shown in Figure 8, that applies a left/right logical shift to the SA output according to the 1+log$n$ control bits. For example, when the desired number of displacements is two and a left shift is indicated, all the internal MUXes at the top half of Figure 8 connect their third input line to output, such that the input data "In_N, In_{N-1}, ... In_2, In_1" are reorganized as "In_{N-2}, In_{N-3}, ... 0, 0". When the number of displacements to apply is 0, the output is equivalent to the input. These logic values are used at the write

driver MUXes as selection signals to allow the corresponding SET/RESET write voltages to be applied to the memristors that will store the memory/logic results.

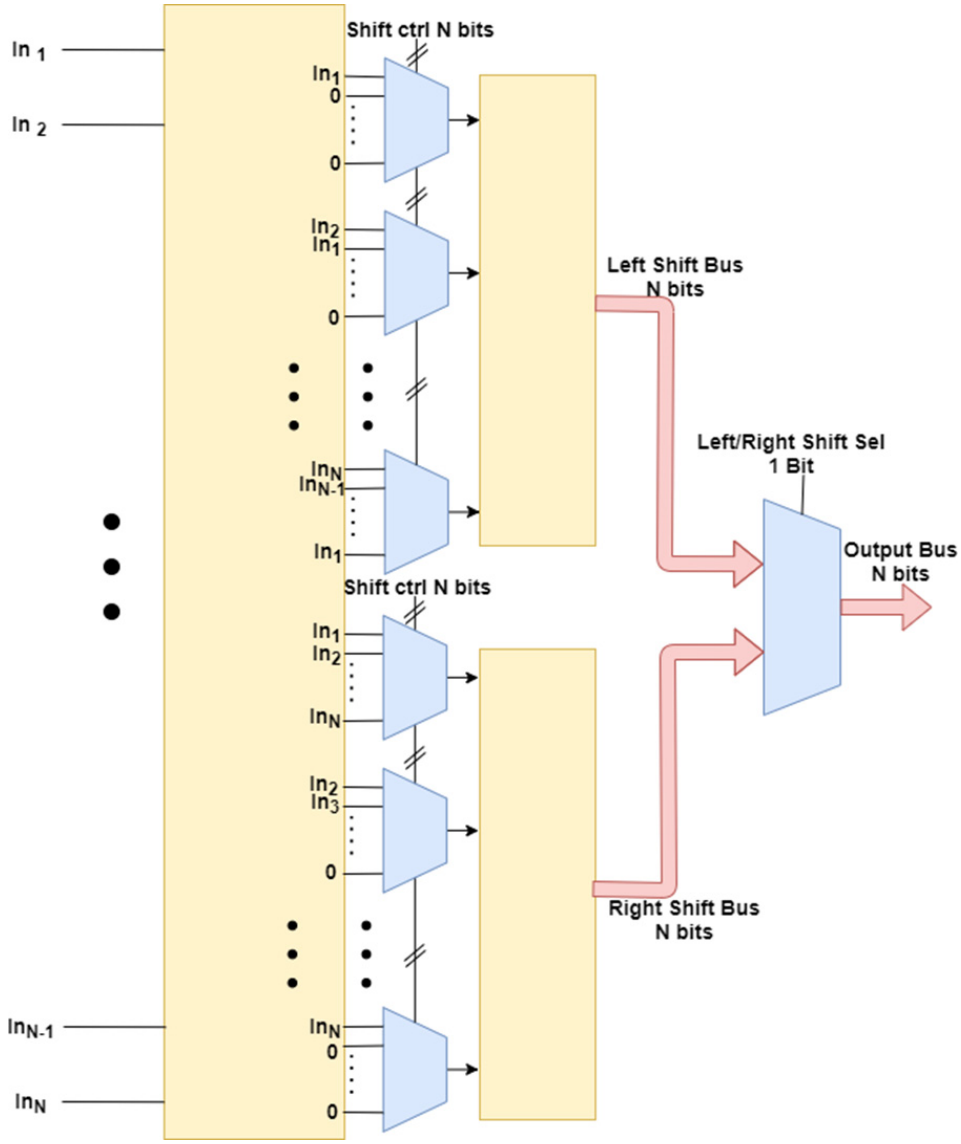


Figure 8. Compact block-level description of the Shift Controller module, which applies a number of logical left/right displacements to the N input lines $In_1 \ldots In_N$ equal to the number given by the Shift Ctrl bits. The yellow blocks hide routing of interconnection lines between the modules (not shown for clarity).

It is worth noting that the favorable performance characteristics of the *twin-array* are achieved owing to the enhanced peripheral circuitry. Figures 6–8 present system-level designs rather than compact transistor-level circuit designs; thus, the area overhead compared to the driving circuit required only for memory operations is difficult to estimate. Enhanced peripheral circuitry could lead to associated *heat dissipation* problems. In this context, in order to minimize the area overhead in the SA Array, one solution would be to combine two crossbar OL per SA module, as suggested in Reference [18]. Thus, the number of SA modules in each sub-array would be half the number of OL. However, such solution will impact latency, as any read/write operation on a memory word would require two steps to be completed, since only half of the OL will be connected each time to ground.

3.3. Supported Set of Instructions/Operations

Table 2 summarizes all the basic operations supported by the proposed computational ReRAM system, which altogether form a basis for more complex operations. The states of the *mode selection bit* and the four *control bits* of the SA module (see Figure 7) are shown, where we use X to denote either logic "1" or "0" value. The b_3 bit is the selection line of SA DEMUX; b_2b_1 are the configuration bits for the internal SA module, whereas b_0 defines the inversion of the memory/logic output in the output MUX. Memory read/write operations require the mode selection bit to be "0" to take place in a target sub-array independently. On the contrary, the copy operation requires the mode selection bit to be "1" since the target sub-array is different from the source sub-array. However, in all logic operations the mode selection bit can take any value, since the results can be either driven directly to the external output or written to the adjacent sub-array. Finally, the bit-shift/selection operations do not take place in the SA but instead in their respective modules, thus there is no SA configuration code shown in their case. According to the list of operations in Table 2, we defined a generic form for the corresponding ReRAM instructions, shown in Figure 9. More specifically, Figure 9a shows the code describing the address of a target word in the two sub-arrays; the most significant 1+log$m$ bits define the target sub-array and the selected wordline in the Wordline Decoder, whereas the last 1+log$n$ bits are the control bits of the Bitline Selector, allowing to activate the entire word or only a specific bitline. The address code field is present in all forms of the instructions, as shown in Figure 9b–d. They consist of an *opcode* represented by the four SA control bits shown in Table 2, followed by the *mode selection bit* (Mode Code), the output/input address fields and the inputs of the Shift Controller.

Figure 9b corresponds to a memory read or to a logic operation, whose output is sent towards the external output. Therefore, the destination/output address field is not used. Depending on the type of operation, more than one input address can be used. In fact, the input address field holds up to three different addresses, since the supported logic operations accept up to three inputs. The last 1+log$n$ bits indicate the shift direction and the number of displacements to be applied to the read data. Likewise, Figure 9c corresponds to a memory write operation of externally applied input data. Part of the input address field is here used to hold the data to be stored, whereas the Shift Controller bits are not used. The output address field holds the destination address. Figure 9d corresponds to internal memory read/logic operations, where the output data are stored to the adjacent sub-array, as indicated by the *mode selection bit*. Note that, as mentioned before, for logic operations, the input addresses need to refer to the same sub-array, whereas the output address can point to either sub-array. Storing the result to the adjacent sub-array can be done in the same cycle, which is very beneficial when intermediate results for chained logic operations need to be stored and used afterwards, as shown in the following sections. However, if the logic result has to be stored to the same sub-array, with the current system design by default, the data will first be written to the adjacent sub-array and then be copied to the destination word in the next cycle. With a modified version of the driving circuitry (not shown here), the read results could be locally stored in the periphery instead, as in Reference [25], to avoid unnecessarily double writing to memristors.

Table 2. Summary of operations supported by the proposed computational memory system. X means a *don't care* value.

Operation	Description	SA Ctrl Bits $b_3b_2b_1b_0$	Mode Sel Bit
Copy	*Copy data to **adjacent** crossbar*	0000	**1**
Inv	*Inversion of the value in the SA output*	XXX**1**	X
OR	*2-input logic OR for two words in the same sub-array*	0000	X
AND	*2-input logic AND for two words in the same sub-array*	00**1**0	X
XOR	*2-input logic XOR for two words in the same sub-array*	0**1**00	X
MAJ	*3-input MAJORITY for three words in the same sub-array*	00**1**0	X
Write	*Write **external** input data to a memory word*	**1**XXX	**0**
Read	*Read data stored in a memory word, to the **external** output*	0000	**0**
Bit shift	*Apply left/right logical shift to the SA output through the Shift Controller*	*N/A*	X
Bit selection	*Activate one target bitline through the Bitline Selector*	*N/A*	X

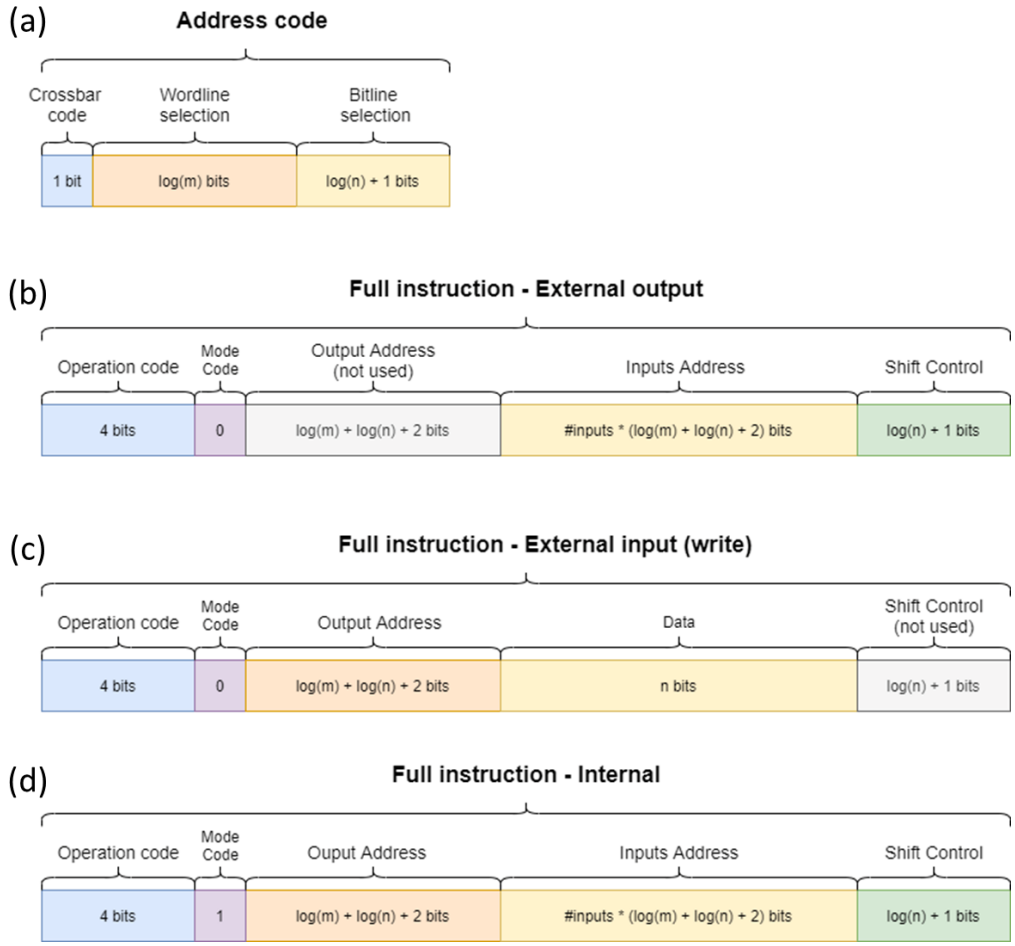


Figure 9. Description of the fields composing the system instructions. (**a**) The form of the address code present in all instruction types. The full instruction form for memory read/logic operations towards the external interface is shown in (**b**), and for memory write operations of externally applied input, it is shown in (**c**). The full instruction form for memory read/logic operations when the output is written to the adjacent sub-array is shown in (**d**). Unused fields are shown in a light gray color.

4. Examples of Memory and Logic Operations

4.1. System-Level Configuration Example

For readability reasons, Figure 10 shows graphically the required system configuration in order to perform an XOR operation on all bitlines of two active words of the left sub-array and store the result to an active word in the right sub-array, in the same cycle. The embedded text description in all blocks that represent the different modules of the system in Figure 10 describes their actual operation. For clarity, all the inactive modules are shown in a light gray color. Specifically, a read voltage is applied through the external interface to all bitlines of the left sub-array. The two words holding the input data are activated by the Wordline Decoder. In the SA Array, the four configuration bits activate the XOR operation in the internal module of every SA Array element (see the inset), without inversion of the logic output. Moreover, no shift is applied to the read data, which are connected through the output DEMUX to the internal data bus, and are thus connected to the selection lines of the MUXes in the write drivers of the right sub-array. There, an internal write operation is performed on all bitlines. The output/destination address is applied to the Wordline Decoder. The internal modules in the entire SA Array connect all OL lines to the ground, as required for the write operations to take place, so that the logic output can be simultaneously written to the target word.

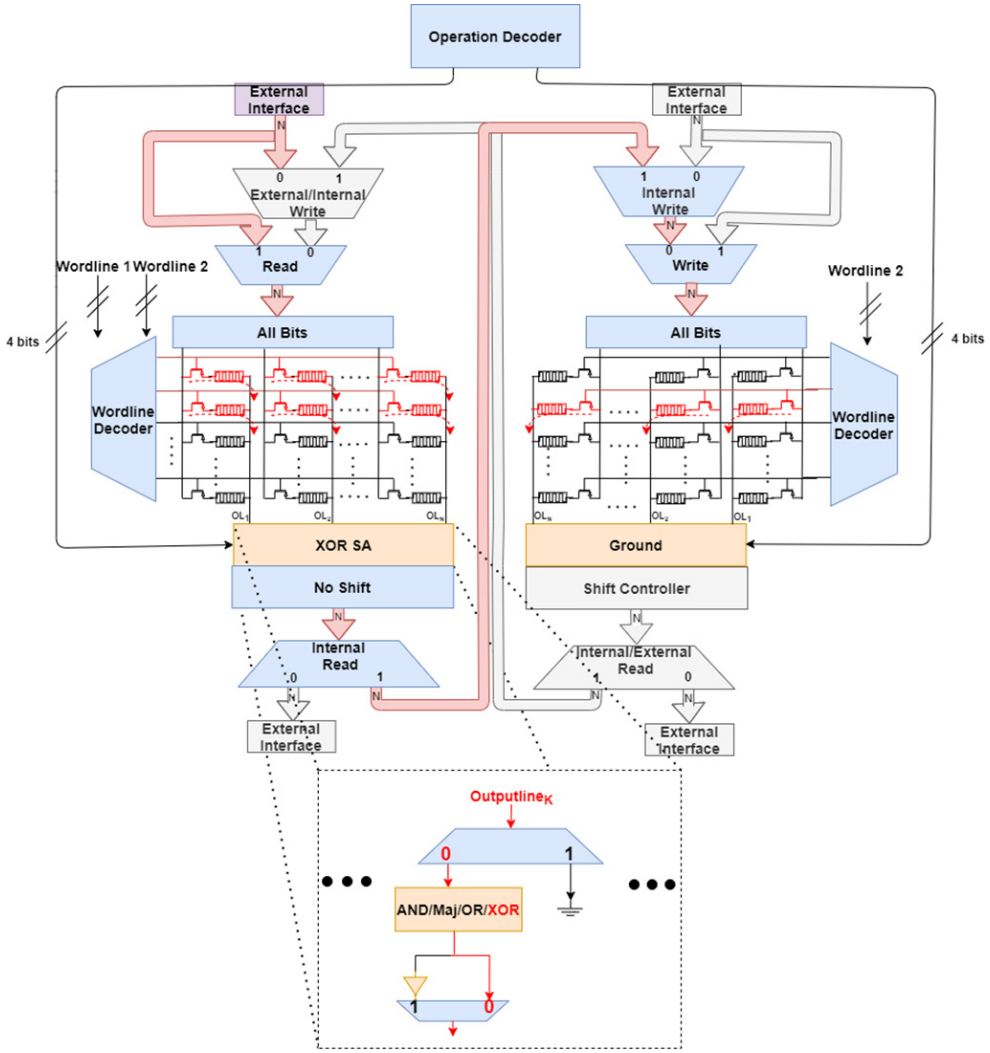


Figure 10. Configuration of the system's modules to perform word-wise XOR operation with simultaneous storage of results to the adjacent sub-array. Text in blocks reveals the actual action performed in all modules. Inactive modules are shown in a light gray shade. The active wordlines in the two sub-arrays and the valid SA configuration (shown in the inset) are highlighted in red color.

Such behavior is validated in the following sections through LTspice circuit simulation results. For the select cross-point transistors in each sub-array, we used 1 µm CMOS technology models. In most of the peripheral blocks, we preferred a behavioral description of circuit components to minimize simulation overhead and emphasize the functional characteristics of the proposed computational ReRAM, rather than the impact of MOS parasitics, which we mostly assumed negligible in the peripheral circuitry. Notwithstanding the above, in the internal SA circuit shown in Figure 4, found in the SA Array, we used macro-models of LM319 high-speed voltage comparators, with V_{dd} = 2 V. For all memristors, we used the hyperbolic sine-type model of a bipolar threshold-based switching memristor proposed by Yakopcic et al. [21]. Such a model has been correlated against several published device characterization data with very good precision, closely approximating performance of physics-based models [29]. Parameters were set in accordance with experimental data for amorphous silicon (a-Si)-type memristors [22] that are suitable for digital applications, as follows: $a1 = 1.6 \times 10^{-4}$, $a2 = 1.6 \times 10^{-4}$, $b = 0.05$, $Vp = 1.088$, $Vn = 1.088$, $Ap = 81{,}600{,}000$, $An = 81{,}600{,}000$, $xp = 0.985$, $xn = 0.985$, $alphap = 0.1$, $alphan = 0.1$, and $xo = 0.01$. The read/write pulses applied to the bitlines were 150 ns wide, and the amplitude was 1.7 V for SET, −1.5 V for RESET, and 0.9 V for READ operations. The corre-

sponding memristance boundary values (for a given V_{read} voltage) were R_{ON} = 125 KΩ and R_{OFF} = 125 GΩ, whereas the SET/RESET switching time was 10 ns.

4.2. Simulation Results for Individual Memory and Logic Operations

Here we present circuit simulation results concerning the execution of all the individual operations supported by the designed computational memory system. Figure 11 shows the simulation results for a sequence of memory and logic operations taking place in three memristors that are connected to the first bitline of crossbar N° 1. We validate functionality, showing the voltage applied to the bitline, the evolution of the logic state of three vertically aligned memristors, and the output of the corresponding SA connected to OL_1. The latter will either reflect the result of a memory read/logic operation or will be 0 V when a write operation takes place and the OL is connected to ground. Note also that, during every cycle, the MUX/DEMUXes of the crossbar sub-array are enabled 30 ns after the read/write voltages have been correctly set up in the bitline drivers, to make sure the SA output is correctly updated and all MUX/DEMUXes have valid selection signals.

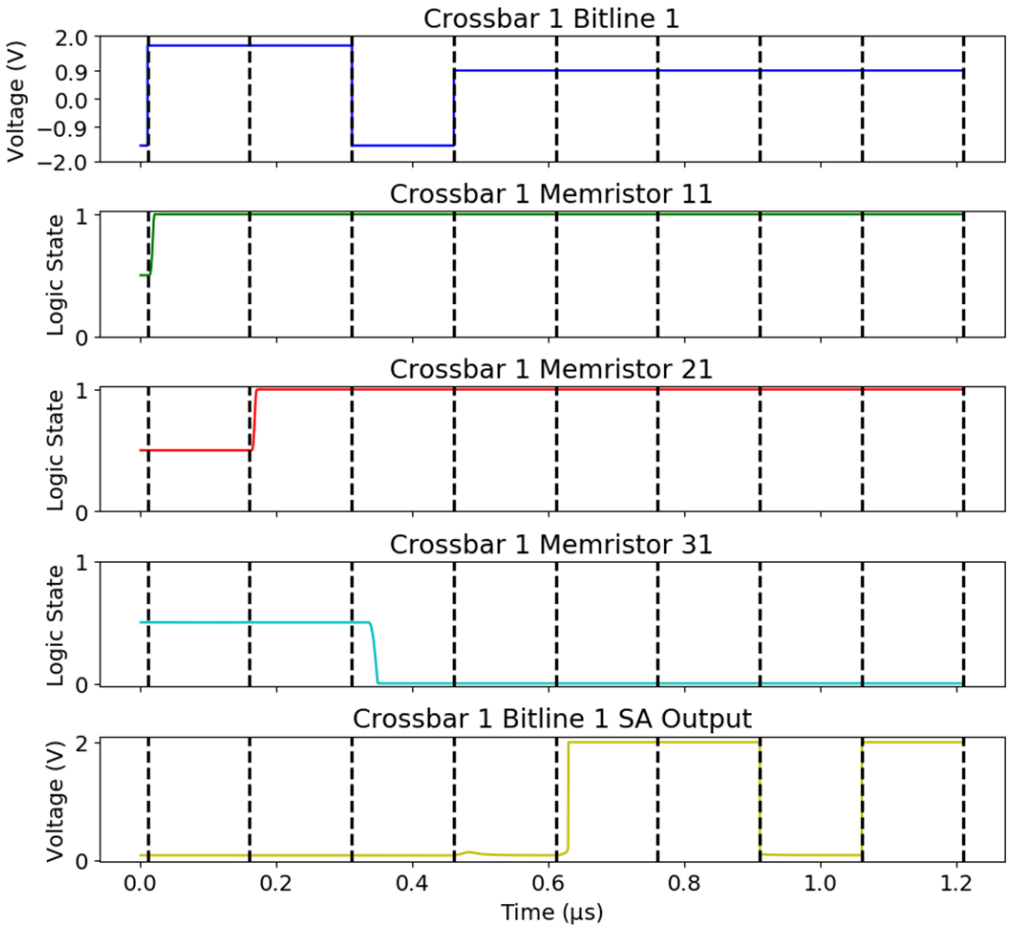


Figure 11. Circuit simulation results for operations performed in a single bitline of the computational ReRAM. From top to bottom, we observe the voltage applied to bitline BL_1, the evolution of the logic state (expressing conductivity in the model) of the memristors in words 1–3 connected to BL_1 (notation "memristor XY" in the legend means wordline X and bitline Y), and the output voltage of the SA connected to OL_1. Logic "1" corresponds to 2 V in the SA output voltage. Vertical dashed lines designate different 150 ns–wide cycles of operation.

The three memristors are purposely initialized in an intermediate state. In the first cycle, a positive write voltage is applied to store logic "1" to memristor 11. Likewise, in the second cycle, we store logic "1" to memristor 12, whereas, in the third cycle, we apply a

negative write voltage to store logic "0" to memristor 31. The SA output voltage is kept in 0 V during the first three cycles. A read voltage is applied in the fourth cycle to memristor 31; thus, a logic "0" is observed in the SA output. From the fifth cycle onward, the logic operations take place. First, a logic OR between memristors 11 and 12, resulting in a logic "1" SA output. In the sixth cycle, a logic AND over the same input data keeps the SA output unaffected. In the seventh cycle, a logic XOR is performed, resulting in a logic "0" output. Finally, an MAJ is performed over the three input memristors, resulting in a logic "1" output, as expected.

4.3. Simulation Results for n-Bit Binary Addition

We designed and simulated in LTspice the complete computational memory system of Figure 6. Thus, here we present circuit simulation results concerning the execution of a binary addition of two memory words. In each bitline, the addition of A_i and B_i bits (along with carry-in C_i bit) takes place according to equation $S_i = A_i \oplus B_i \oplus C_i$ for Sum and $C_i = A_{i-1}B_{i-1} + B_{i-1}C_{i-1} + A_{i-1}C_{i-1}$ for Carry, respectively. This way, through such a case study of arithmetic computing, we highlight all the major benefits offered by the proposed "twin" computational ReRAM.

More specifically, the circuit simulated has two 1T1R crossbar sub-arrays, each of size 4×3, and all the write drivers and circuit modules described in previous sections. We assume that all cross-point memristors in the system initially have an arbitrarily selected resistance close to their R_{OFF} (HRS) value. The simulation starts with an initialization phase which lasts three cycles, in which we update the memory content to be used as input to the logic operation, and we also RESET the devices in two auxiliary words which will hold intermediate data during computations. During the next four cycles, the logic operations in both sub-arrays are carried out. The result of the binary addition is stored in a memory word in the last cycle (cycle N° 8). Figure 12 shows the simulation results where every different 150 ns cycle is designated by vertical dashed lines. More specifically, Figure 12a shows the voltages applied to the bitlines of each sub-array, whereas Figure 12b shows the evolution of the logic state of the memristors in the words involved in the computation. Finally, Figure 12c shows the output voltage of the SA Array of the two crossbars. For readability reasons, in Table 3, we describe graphically all the simulated computational steps required to perform the binary addition of numbers "011" and "010". In Table 3, we use the notation "word[N° crossbar][N° row]" to refer to a complete word, whereas for operations on a single bit of a specific word, we use "bit[N° crossbar][N° row][N° column]". Left/right sub-array is mentioned as crossbar N° 1/N° 2. We included a series of schematics as a guide to the eye for all the operations, highlighting the active word-/bit-lines in red color, while showing the logic inputs applied to the bitlines during the write operations, and the SA Array configuration at the bottom of every sub-array.

During initialization, in the first two cycles, we write "011" to word 1 of crossbar 1 and "010" to word 2 of crossbar 1, which are the words to be used as inputs (A_i and B_i) for the binary addition. Next, in cycle N° 3, we write simultaneously "000" both to word 3 of crossbar 1 and to word 2 of crossbar 2, which will hold intermediate results of Carry bits. The first logic operation takes place in the fourth cycle, being a logic XOR(A, B) with the contents of words 1 and 2 of crossbar 1. The result is written to word 1 of crossbar 2 in the same cycle. This can be verified by observing Figure 12c; the final output of the SA modules of crossbar 1 shows "100", which is the same with the final state of the memristors observed in word 1 of crossbar 2 in Figure 12b. At the same time, we observe that the SA array of crossbar 2 connects all bitlines to ground for the write operation to take place correctly. Next, we sequentially compute the resulting Carry bit from each bitline using the majority operation according to equation $C_{out} = MAJ(A, B, C_{in}) = AB + BC_{in} + AC_{in}$. During Carry bit computations, given that the produced C_i in every stage i acts as input for stage i+1, we exploit the Shift Controller to apply a left shift to the SA output of crossbar 1, and the Bitline Selector of crossbar 2 to activate only one target bitline, where the computed C_i should be stored.

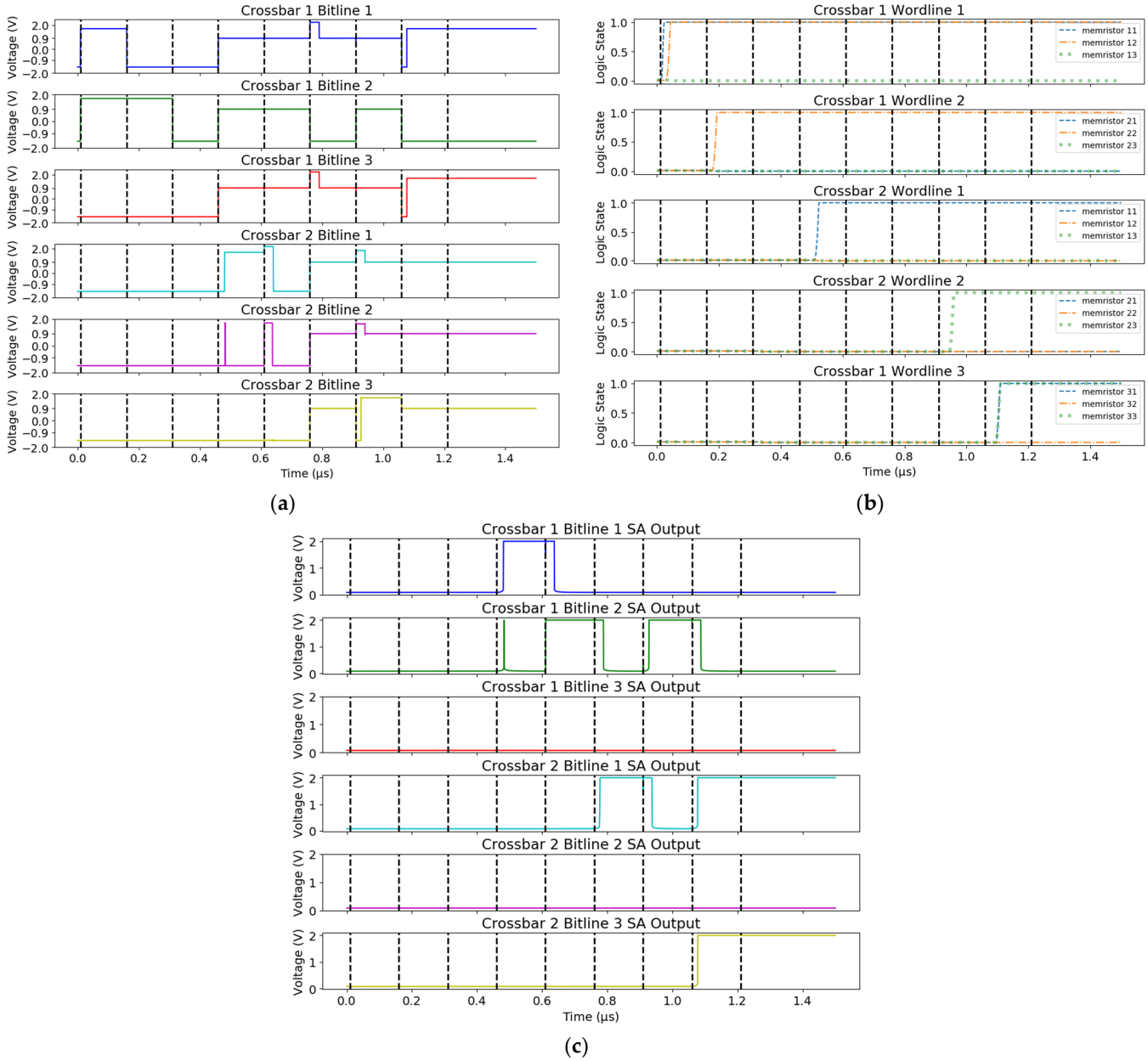
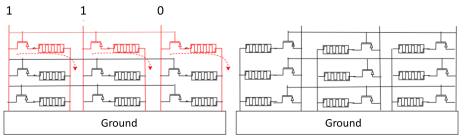
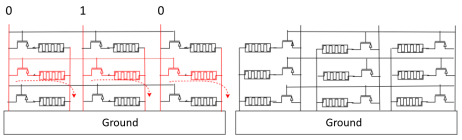
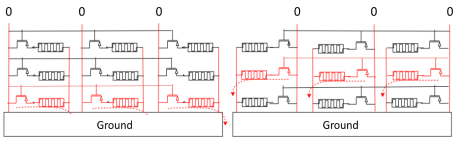
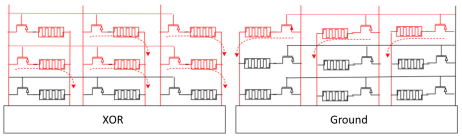
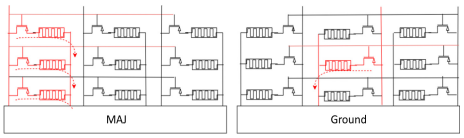
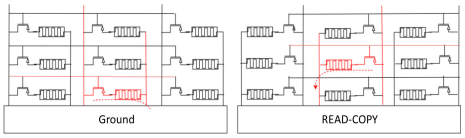
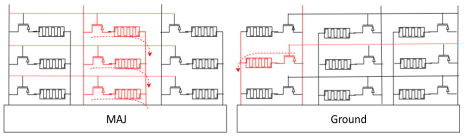
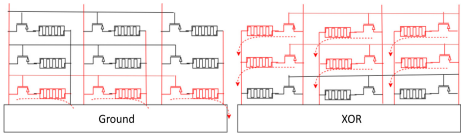


Figure 12. Circuit simulation results for a binary addition performed inside the computational ReRAM. (**a**) Shows the voltages applied to three bitlines of interest in both sub-arrays. (**b**) Shows the evolution of the logic state of the memristors in all the words involved in the computation. Notation "memristor XY" in the legend means the device in wordline X and bitline Y. (**c**) Shows the output voltage of the SA Array modules of the two sub-arrays. Logic "1" corresponds to 2 V in the SA output voltage. Vertical dashed lines designate different 150 ns–wide cycles of operation.

By observing Figure 12b,c, it can be figured out that, in the fifth cycle, the result of the MAJ operation at bitline 1 of crossbar 1, which is logic "0", is written to the memristor in word 2 and bitline 2 of crossbar 2. Given that C_{in} is zero for the LSB stage, crossbar 2 has already a logic "0" in bitline 1, owing to the RESET write operation performed in cycle N° 3. Next, the recently produced C_i is copied to the memristor in word 3 and bitline 2 of crossbar 1, so that, in cycle N° 7, we can compute again the MAJ operation. However, this time, MAJ is performed at bitline 2 of crossbar 1 to produce the last Carry bit, which we simultaneously store in the memristor in word 2 and bitline 3 of crossbar 2. In Figure 12b, we can confirm that a logic "1" is stored in crossbar 2 as a final Carry bit to the memristor in word 2 and bitline 3, as expected. Finally, in the eighth cycle, we compute the result of Sum

with another XOR operation performed in crossbar 2 with the contents of word 1, which holds the result of the previous XOR operation, and word 2, which holds the computed Carry bits. The result is stored simultaneously to word 3 of crossbar 1, which eventually holds "101" (equal to "011" + "010"), as we can confirm by checking Figure 12b.

Table 3. Summary of simulated computational steps required to perform the binary addition.

Cycle	Operation	Output/Destination	Input	Schematic Guide
1	Write	word 11	011	
2	Write	word 12	010	
3	Write	word 22	000	
	Write	word 13	000	
4	XOR	word 21	word 11 word 12	
5	MAJ	bit 222	bit 111 bit 121 bit 131	
6	Copy	bit 132	bit 222	
7	MAJ	bit 223	bit 112 bit 122 bit 132	
8	XOR	word 13	word 21 word 22	

For each one of the eight total simulated cycles described in Table 3, we present in Table 4 the corresponding form of the system instructions that are executed. We separated in different columns all the different fields explained in Figure 9. The different colors in the values given for input/output addresses designate the different fields of the address code in Figure 9a: red for the crossbar N° bit, orange for the Wordline Selection bits, blue for the Bitline Selection bits, and purple for the bit which defines bit-/word-wise operation. For

instance, in the eighth cycle, we perform an XOR (opcode = "0100") in crossbar 2 (crossbar N° bit = 1) with the contents of word 1 and word 2 (wordline selection bits = "01"/"10") on all bitlines (bitline selection bits = "00"), as reflected in the input addresses. The result is stored simultaneously (mode bit = 1) without any logical shift applied, to crossbar 1 (crossbar N° bit = 0) in word 3 (wordline selection bits = "11") in all bitlines, as reflected in the output address information. In the same way, we read all the presented instructions. Note that, in the first four cycles, the input address field holds the logic input to be stored in memory.

Table 4. Form of instructions executed to perform the binary addition. Colors in the values given for input/output addresses designate the different fields of the address code in Figure 9a. X means a *don't care* value.

Cycle	Opcode	Mode	Output	Input(s)	Shift
1	1XXX	0	001000	011	XXX
2	1XXX	0	010000	010	XXX
3	1XXX	0	011000	000	XXX
	1XXX	0	110000	000	XXX
4	0100	1	101000	001000 010000	000
5	0010	1	110101	001011 010011 011011	001
6	0000	1	011101	110101	000
7	0010	1	110111	001101 010101 011101	001
8	0100	1	011000	101000 110000	000

4.4. Performance Comparison Results

In Table 5, we summarize the comparison results w.r.t. *circuit area* and *latency* required for the implementation of two-input OR/NOR/XOR/AND/NAND and three-input MAJ logic operations that are supported by the proposed computational ReRAM, using state-of-the-art crossbar-compatible in-memory computing approaches of the literature. Specifically, circuit area is expressed in *N° of cross-points* involved in computation, and latency is expressed in total *N° of required cycles/steps*. Data for IMPLY logic were based on Reference [30] for MAJ and on Reference [31] for OR/NOR/XOR/AND/NAND operations. IMPLY, MAGIC, and NAND are "stateful" logic styles; thus, the required memristors always include a device to store the logic output. MAGIC style [32] assumes NOR and NOT crossbar-compatible operations, and memristor reutilization was taken into consideration to minimize the required area for multi-level computations. The NAND style [33] assumes single-step AND/NAND operations. Finally, the MAJ+NOT [18] is a "nonstateful" logic style; thus, intermediate write steps are considered to store in memory the intermediate results of multi-level logic operations. By observing Table 5, we conclude that the proposed computational memory system allows for the execution of all such in-memory computations in a single step and in a "nonstateful" manner, requiring only as many memristors as the gate inputs.

Likewise, in Table 6, we summarize the comparison results w.r.t. the resources required for the addition of two n-bit binary numbers. The proposed approach, owing to the enhanced sensing scheme, as well as the word-wise operation and the "write while reading" capability allowed by the "twin" ReRAM concept, required only $2n$+2 steps/cycles and involved $3n$ total cross-points in the operation. The $2n + 2$ steps correspond to two word-

wise XOR operations and n computations of the carry bit for every bitline, each of which needs two operations (a MAJ and a COPY). The $3n$ total cross-points are those involved in computation and do not include the devices in the words that hold the input data, which are assumed already available in memory in the same sub-array; the same concept is used in the rest of area computations for all logic schemes mentioned in Table 6. We compare the proposed approach to others that are based on "stateful" logic schemes which use two-input primitive logic operations, such as MAGIC [32], IMPLY [12,32,34], and NAND [33]. One recent modification of IMPLY logic is ORNOR [35], which enables three-input logic operations, thus improving the overall computing latency compared to original IMPLY-based operations. We included in this comparison the work by Reuben et al. in Reference [36], which presented an accelerated MAJ-based execution of binary addition, compared to previous work, such as Reference [18]. By observing Table 6, we conclude that, the smaller the area, the larger the number of total steps needed, especially for the IMPLY-based schemes, which require many sequential operations while occupying just two cross-points. All in all, the proposed computing approach requires almost 50% less cross-points, compared to schemes that perform parallel computations, while taking the least possible cycles. The only exception is the MAJ+NOT style [36], which clearly outperforms the rest in computing steps. However, this comes at the cost of larger area, since it requires more than 12× the number of memristors used by the proposed approach. Along with such performance improvements over the rest of the evaluated schemes, it is worth highlighting the higher tolerance to device variability of the *Scouting Logic*, compared to the "stateful" schemes that assume "conditional" switching of memristors. This notwithstanding, the proposed computational ReRAM assumes by default a higher area complexity in the sensing modules, which could be improved by a compact circuit design and by applying architectural solutions, such as by multiplexing the sensing modules.

Table 5. Resources required for individual logic gates using different in-memory logic approaches.

	IMPLY [30,31]		**MAGIC [32]**		**NAND [33]**		**MAJ+NOT [18]**		**Proposed**	
	Steps	**Area**	**Steps**	**Area**	**Steps**	**Area**	**Steps**	**Area**	**Steps**	**Area**
OR	2	3	2	4	2	6	3	3	**1**	**2**
AND	3	3	2	5	1	3	1	3	**1**	**2**
NOR	3	3	1	3	3	6	3	3	**1**	**2**
NAND	2	3	3	5	1	3	1	3	**1**	**2**
XOR	4	5	3	6	3	6	5	6	**1**	**2**
MAJ	26	5	6	6	7	7	1	3	**1**	**3**

Note: all logic gates concern 2 inputs, except for MAJ, which has 3 inputs.

Table 6. Resources required for n-bit binary addition using different in-memory logic approaches.

Description	**Latency (Steps)**	**Circuit Area (N° of Memristors)**	**Reference**
MAGIC (NOR, area optimized)	$15n$	5	Talati et al. [32]
MAGIC (lNOR, latency optimized)	$12n + 1$	$11n - 1$	Talati et al. [32]
IMPLY (parallel)	$5n + 18$	$6n - 1$	Kvatinsky et al. [31]
IMPLY (serial)	$22n$	2	Rohani et al. [34]
IMPLY (semi-serial)	$17n$	2	Rohani et al. [12]
NAND	$10n$	9	Huang et al. [33]
ORNOR	$2n + 15$	$6n + 6$	Siemon et al. [35]
MAJ + NOT	$4\log_2 n + 6$	$6(6n + 16)$	Reuben et al. [36]
Enhanced Scouting (XOR and MAJ)	$\mathbf{2n + 2}$	$\mathbf{3n}$	**This work**

Note: n stands for the number of bits in each one of the added binary numbers.

5. Conclusions

Towards the efficient design and implementation of next-generation ALUs in ReRAM-based computational memories, this work highlighted some promising system design concepts to consider, introducing a segmented 1T1R array which uses an augmented peripheral circuitry to improve logic latency of non-stateful logic schemes, where computations are performed via modified memory read operations. Alternative designs for the sensing circuitry were proposed that were proved to be robust in the presence of device-to-device variability in memristors. We identified the set of all supported primitive operations/instructions of the proposed computational memory system and addressed system-level design issues towards the design of a ReRAM-based general-purpose computational memory with ALU functionality. Circuit simulation results validated functionality of the designed system, which demonstrated important performance improvements over other state-of-the-art in-memory computing approaches both for elementary logic operations and for *n*-bit binary addition.

Author Contributions: Conceptualization, F.P. and I.V.; methodology, F.P. and I.V.; software, F.P.; validation, F.P.; formal analysis, F.P.; investigation, F.P. and I.V.; resources, F.P. and I.V.; data curation, F.P.; writing—original draft preparation, I.V.; writing—review and editing, F.P. and I.V.; visualization, F.P. and I.V.; supervision, I.V.; project administration, I.V.; funding acquisition, I.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by ANID FONDECYT INICIACION, grant number 11180706, and by ANID-Basal Project, grant number FB0008.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Appendix A

Table A1 presents all possible configurations of the SA shown in Figure 3 with their equivalent circuits.

Table A2 presents all possible configurations of the SA shown in Figure 4 and their equivalent circuits, along with the mathematical expressions describing the resulting voltage inputs applied to the voltage comparator stage.

Table A1. Equivalent circuits for all possible configurations of the sensing module.

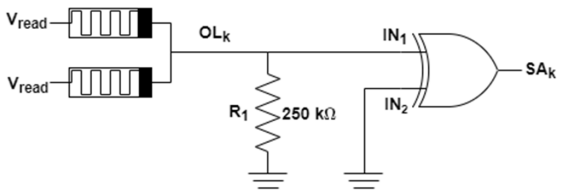
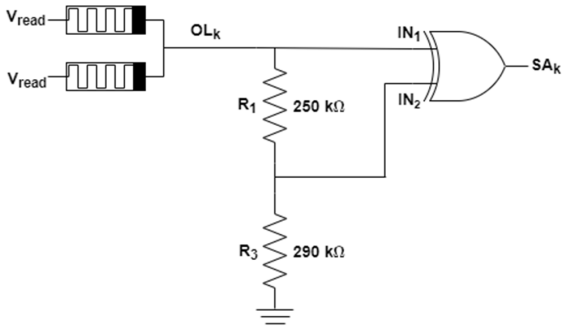
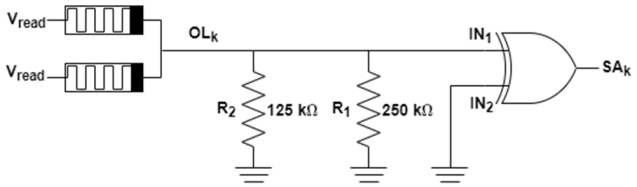
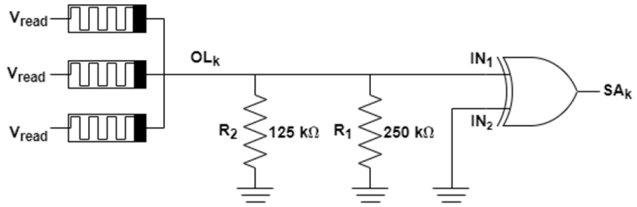
SA Operation	Equivalent Circuit
Read	 $V_{IN_1} = V_{read}\left(\frac{R_1}{R_{OL_k} + R_1}\right)$
OR	 $V_{IN_1} = V_{read}\left(\frac{R_1}{R_{OL_k} + R_1}\right)$

Table A1. *Cont.*

SA Operation	Equivalent Circuit
XOR	 $V_{IN_1} = V_{read}\left(\frac{R_1 + R_3}{R_{OL_k} + R_1 + R_3}\right)$ $V_{IN_2} = V_{IN_1}\left(\frac{R_3}{R_3 + R_1}\right)$
AND	 $V_{IN_1} = V_{read}\left(\frac{R_1 // R_2}{R_{OL_k} + R_1 // R_2}\right)$
MAJ	 $V_{IN_1} = V_{read}\left(\frac{R_1 // R_2}{R_{OL_k} + R_1 // R_2}\right)$

Note: R_{OLk} in equations stands for the equivalent resistance of the parallel input memristors.

Table A2. Equivalent circuits for all possible configurations of the alternative sensing module.

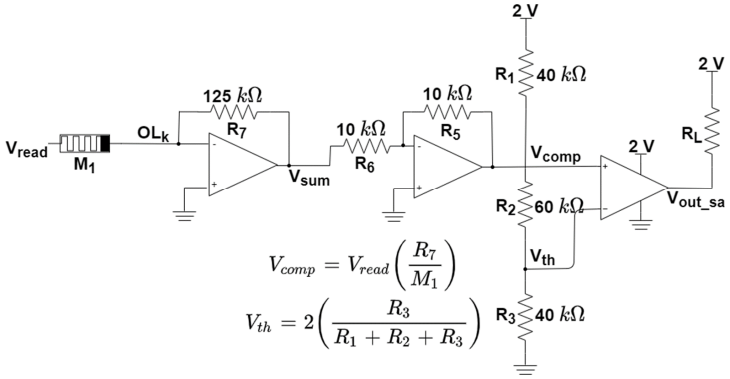
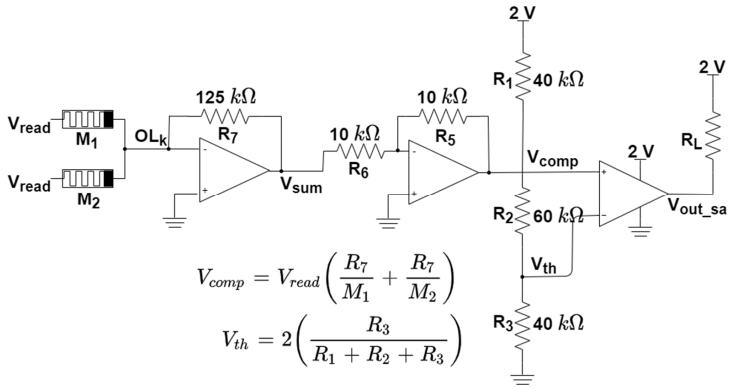
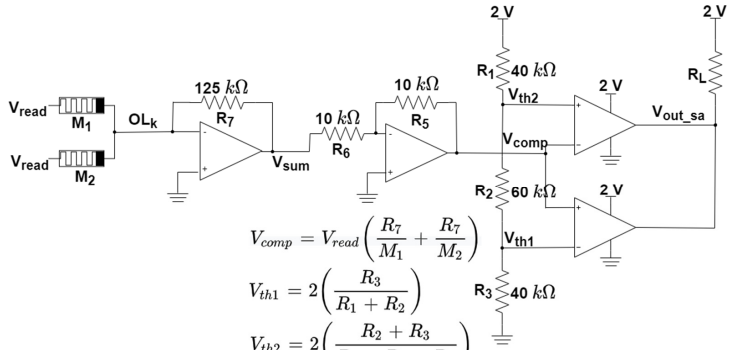
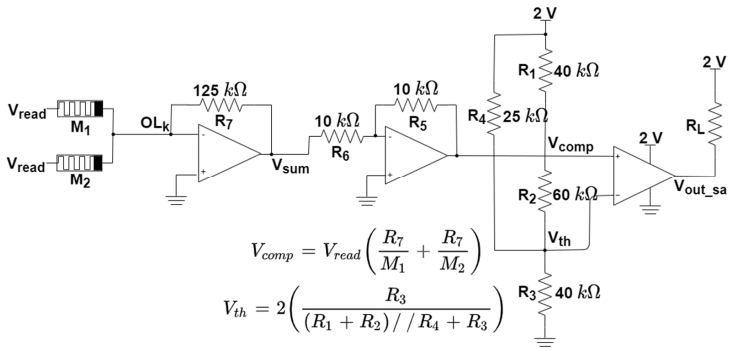
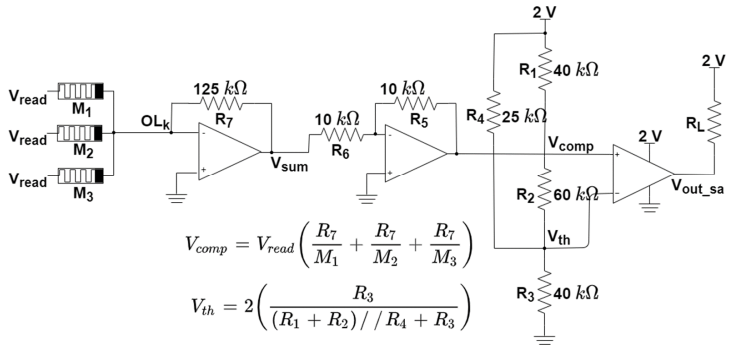
SA Operation	Equivalent Circuit
Read	 $V_{comp} = V_{read}\left(\frac{R_7}{M_1}\right)$ $V_{th} = 2\left(\frac{R_3}{R_1 + R_2 + R_3}\right)$

Table A2. *Cont.*

SA Operation	Equivalent Circuit
OR	 $V_{comp} = V_{read}\left(\frac{R_7}{M_1} + \frac{R_7}{M_2}\right)$ $V_{th} = 2\left(\frac{R_3}{R_1 + R_2 + R_3}\right)$
XOR	 $V_{comp} = V_{read}\left(\frac{R_7}{M_1} + \frac{R_7}{M_2}\right)$ $V_{th1} = 2\left(\frac{R_3}{R_1 + R_2}\right)$ $V_{th2} = 2\left(\frac{R_2 + R_3}{R_1 + R_2 + R_3}\right)$
AND	 $V_{comp} = V_{read}\left(\frac{R_7}{M_1} + \frac{R_7}{M_2}\right)$ $V_{th} = 2\left(\frac{R_3}{(R_1 + R_2)//R_4 + R_3}\right)$
MAJ	 $V_{comp} = V_{read}\left(\frac{R_7}{M_1} + \frac{R_7}{M_2} + \frac{R_7}{M_3}\right)$ $V_{th} = 2\left(\frac{R_3}{(R_1 + R_2)//R_4 + R_3}\right)$

Note: $M_{1–3}$ stands for the memristance of any input memristor.

References

1. Aly, M.M.S.; Wu, T.F.; Bartolo, A.; Malviya, Y.H.; Hwang, W.; Hills, G.; Markov, I.; Wootters, M.; Shulaker, M.M.; Wong, H.-S.P.; et al. The N3XT Approach to Energy-Efficient Abundant-Data Computing. *Proc. IEEE* **2018**, *107*, 19–48. [CrossRef]
2. Sebastian, A.; Tuma, T.; Papandreou, N.; Le Gallo, M.; Kull, L.; Parnell, T.P.; Eleftheriou, E. Temporal correlation detection using computational phase-change memory. *Nat. Commun.* **2017**, *8*, 1115. [CrossRef] [PubMed]
3. Crossbar Inc. Available online: https://www.crossbar-inc.com (accessed on 30 April 2021).
4. Liu, C.; Wu, T. ReRAM-based Circuit and System Design for Future Storage and Computing. In Proceedings of the 2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), Chengdu, China, 26–30 October 2018.
5. Im, I.H.; Kim, S.J.; Jang, H.W. Memristive Devices for New Computing Paradigms. *Adv. Intell. Syst.* **2020**, *2*. [CrossRef]
6. Govoreanu, B.; Kar, G.; Chen, Y.-Y.; Paraschiv, V.; Kubicek, S.; Fantini, A.; Radu, I.; Goux, L.; Clima, S.; Degraeve, R.; et al. 10×10 nm^2 Hf/HfOx crossbar resistive RAM with excellent performance, reliability and low-energy operation. In Proceedings of the 2011 IEEE International Electron Devices Meeting, Washington, DC, USA, 5–7 December 2011; pp. 31.6.1–31.6.4.
7. Li, C.; Belkin, D.; Li, Y.; Yan, P.; Hu, M.; Ge, N.; Jiang, H.; Montgomery, E.; Lin, P.; Wang, Z.; et al. In-Memory Computing with Memristor Arrays. In Proceedings of the 2018 IEEE International Memory Workshop (IMW), Kyoto, Japan, 13–16 May 2018; pp. 1–4.
8. Vourkas, I.; Sirakoulis, G.C. Emerging Memristor-Based Logic Circuit Design Approaches: A Review. *IEEE Circuits Syst. Mag.* **2016**, *16*, 15–30. [CrossRef]
9. Yadav, D.N.; Thangkhiew, P.L.; Datta, K. Look-ahead mapping of Boolean functions in memristive crossbar array. *Integr. VLSI J.* **2019**, *64*, 152–162. [CrossRef]
10. Ye, W.; Cui, X.; Ma, Y.; Wei, F. An Improved Synthesis Method of Logic Circuits based on the NMOS-like RRAM Gates. In Proceedings of the 2020 IEEE 15th International Conference on Solid-State & Integrated Circuit Technology (ICSICT), Kunming, China, 3–6 November 2020; pp. 1–3. [CrossRef]
11. Cheng, L.; Li, Y.; Yin, K.; Hu, S.; Su, Y.; Jin, M.; Wang, Z.; Chang, T.; Miao, X. Functional Demonstration of a Memristive Arithmetic Logic Unit (MemALU) for In-Memory Computing. *Adv. Funct. Mater.* **2019**, *29*, 1905660. [CrossRef]
12. Rohani, S.G.; Taherinejad, N.; Radakovits, D. A Semiparallel Full-Adder in IMPLY Logic. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2020**, *28*, 297–301. [CrossRef]
13. Escudero, M.; Vourkas, I.; Rubio, A.; Moll, F. Memristive Logic in Crossbar Memory Arrays: Variability-Aware Design for Higher Reliability. *IEEE Trans. Nanotechnol.* **2019**, *18*, 635–646. [CrossRef]
14. Zhu, X.; Long, H.; Li, Z.; Diao, J.; Liu, H.; Li, N.; Xu, H. Implication of unsafe writing on the MAGIC NOR gate. *Microelectron. J.* **2020**, *103*, 104866. [CrossRef]
15. Gaillardon, P.E.; Amarú, L.; Siemon, A.; Linn, E.; Waser, R.; Chattopadhyay, A.; Micheli, G.D. The programmable logic-in-memory (plim) computer. In Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 14–18 March 2016; pp. 427–432.
16. Bhattacharjee, D.; Devadoss, R.; Chattopadhyay, A. ReVAMP: ReRAM based VLIW architecture for in-memory computing. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, 27–31 March 2017; pp. 27–31.
17. Xie, L.; Du Nguyen, H.; Yu, J.; Kaichouhi, A.; Taouil, M.; AlFailakawi, M.; Hamdioui, S. Scouting Logic: A Novel Memristor-Based Logic Design for Resistive Computing. In Proceedings of the 2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Bochum, Germany, 3–5 July 2017; pp. 176–181.
18. Reuben, J. Binary Addition in Resistance Switching Memory Array by Sensing Majority. *Micromachines* **2020**, *11*, 496. [CrossRef] [PubMed]
19. Pinto, F.; Vourkas, I. Design Considerations for the Development of Computational Resistive Memories. In Proceedings of the 2021 IEEE Latin American Symp. Circuits and Systems (LASCAS), Arequipa, Perú, 21–25 February 2021. in press.
20. Vourkas, I.; Sirakoulis, G.C. Memristor-Based Nanoelectronic Computing Circuits and Architectures. In *Emergence, Complexity and Computation*; Springer International Publishing: Cham, Germany, 2016; Volume 19.
21. Yakopcic, C.; Taha, T.M.; Subramanyam, G.; Pino, R.E. Generalized Memristive Device SPICE Model and its Application in Circuit Design. *IEEE Trans. Comput. Des. Integr. Circuits Syst.* **2013**, *32*, 1201–1214. [CrossRef]
22. Lu, W.; Kim, K.-H.; Chang, T.; Gaba, S. Two-terminal resistive switches (memristors) for memory and logic applications. In Proceedings of the 16th IEEE Asia and South Pacific Design Automation Conference (ASP-DAC 2011), Yokohama, Japan, 25–28 January 2011; pp. 217–223.
23. Flocke, A.; Noll, T.G. Fundamental analysis of resistive nano-crossbars for the use in hybrid Nano/CMOS-memory. In Proceedings of the2006 Proceedings of the 32nd IEEE European Solid-State Circuits Conference, Montreux, Switzerland, 19–21 September 2006; pp. 328–331.
24. Seok, J.Y.; Song, S.J.; Yoon, J.H.; Yoon, K.J.; Park, T.H.; Kwon, D.E.; Lim, H.; Kim, G.H.; Jeong, D.S.; Hwang, C.S. A review of three-dimensional resistive switching crossbar array memories from the integration and materials property points of view. *Adv. Funct. Mater.* **2014**, *24*, 5316–5339. [CrossRef]
25. Fernandez, C.; Vourkas, I. ReRAM-based Ratioed Combinational Circuit Design: A Solution for in-Memory Computing. In Proceedings of the 2020 International Conference on Modern Circuits and Systems Technologies (MOCAST), Bremen, Germany, 7–9 September 2020.

26. Yu, J.; Du Nguyen, H.A.; Abu Lebdeh, M.; Taouil, M.; Hamdioui, S. Enhanced Scouting Logic: A Robust Memristive Logic Design Scheme. In Proceedings of the 2019 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH), Qingdao, China, 17–19 July 2019; pp. 1–6.
27. Papandroulidakis, G.; Vourkas, I.; Abusleme, A.; Sirakoulis, G.C.; Rubio, A. Crossbar-Based Memristive Logic-in-Memory Architecture. *IEEE Trans. Nanotechnol.* **2017**, *16*, 491–501. [CrossRef]
28. Ali, K.A.; Rizk, M.; Baghdadi, A.; Diguet, J.-P.; Jomaah, J.; Onizawa, N.; Hanyu, T. Memristive Computational Memory Using Memristor Overwrite Logic (MOL). *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2020**, *28*, 2370–2382. [CrossRef]
29. Kalkur, T.S.; Pawlikiewicz, M. Verilog-A modeling of filamentary-based complementary resistance switching devices. In Proceedings of the 2018 IEEE 18th International Conference on Nanotechnology (IEEE-NANO), Cork, Ireland, 23–26 July 2018; pp. 1–4.
30. Lehtonen, E.; Laiho, M. Stateful implication logic with memristors. In Proceedings of the 2009 IEEE/ACM International Symposium on Nanoscale Architectures, San Francisco, CA, USA, 30–31 July 2009; pp. 33–36.
31. Kvatinsky, S.; Satat, G.; Wald, N.; Friedman, E.G.; Kolodny, A.; Weiser, U.C. Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2014**, *22*, 2054–2066. [CrossRef]
32. Talati, N.; Gupta, S.; Mane, P.; Kvatinsky, S. Logic Design Within Memristive Memories Using Memristor-Aided loGIC (MAGIC). *IEEE Trans. Nanotechnol.* **2016**, *15*, 635–650. [CrossRef]
33. Huang, P.; Kang, J.; Zhao, Y.; Chen, S.; Han, R.; Zhou, Z.; Chen, Z.; Ma, W.; Lifeng, L.; Liu, L.; et al. Reconfigurable Nonvolatile Logic Operations in Resistance Switching Crossbar Array for Large-Scale Circuits. *Adv. Mater.* **2016**, *28*, 9758–9764. [CrossRef] [PubMed]
34. Rohani, S.G.; Taherinejad, N. An improved algorithm for IMPLY logic based memristive Full-adder. In Proceedings of the 2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE), Windsor, ON, Canada, 30 April–3 May 2017; pp. 1–4.
35. Siemon, A.; Drabinski, R.; Schultis, M.J.; Hu, X.; Linn, E.; Heittmann, A.; Waser, R.; Querlioz, D.; Menzel, S.; Friedman, J.S. Stateful Three-Input Logic with Memristive Switches. *Sci. Rep.* **2019**, *9*, 1–13. [CrossRef] [PubMed]
36. Reuben, J.; Pechmann, S. Accelerated Addition in Resistive RAM Array Using Parallel-Friendly Majority Gates. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2021**, in press. [CrossRef]