ReRAM-based Ratioed Combinational Circuit Design: a Solution for in-Memory Computing

Carlos Fernandez
Dept. of Electronic Engineering
Universidad Técnica Federico Santa María
Valparaíso, Chile
carlos.fernandezh@sansano.usm.cl

Ioannis Vourkas
Dept. of Electronic Engineering
Universidad Técnica Federico Santa María
Valparaíso, Chile
ioannis.vourkas@usm.cl

***Abstract*— While the von Neumann architecture played a leading role in CMOS-based computing systems for several decades, nowadays in-memory computing is an alternative approach being pursued, with resistive switching devices (memristors) in crossbar arrays considered as the enabling technology. In this context, this paper provides a practical solution for a viable in-memory computing architecture within the reach of today´s technology, through a variability-tolerant ReRAM-based ratioed combinational logic design scheme, inspired on the pseudo-NMOS logic design. The reason we focus on this scheme is because it is simple, crossbar-compatible, completely tolerant to variability, compatible with either filamentary or interfacial switching type devices, and it does not affect the memristor endurance. We highlight all the important characteristics and advantages offered by this scheme, compared to other stateful logic schemes based on memristors, such as IMPLY and MAGIC. We conclude this paper presenting SPICE-based circuit simulation results concerning a 1-bit full adder implementation and show that our proposed ratioed logic design outperforms the rest in terms of speed and area requirements.**

Keywords—memristor; resistive switching; resistive RAM; ReRAM; in-memory computing; NOR logic; SPICE

I. INTRODUCTION

There is no doubt that the von Neumann architecture has played a leading role in CMOS-based computing systems for several decades. Nevertheless, in the era of big data and given the rapid development of information technology, modern computing systems are calling for architectural changes to overcome problems such as the "*von Neumann bottleneck*" and "*memory wall*" [1]. In this context, in-memory computing is an alternative approach currently being pursued, with resistive switching devices (memristors) being among the key enabling device technologies to consider [2], [3].

In such a conceived memristor-based processing unit, the fusion of memory and computing could be achieved using very dense memristive crossbar array as memory module and peripheral circuitry for the control and communication with the memory array [4]. Real "*in-memory computation*" can be performed with the applied voltages not acting as logic signals but instead driving the memristive logic circuits, which are directly implemented in the cross-point cells; logic inputs are the data already stored in the resistive state of the memristors involved in computation. A major requirement is that the topology of logic circuits constructed must be compatible with the crossbar array [5], which appears to be the preferred topology for future resistive memories (ReRAM) that are expected to benefit emerging edge computing applications [6].

There are numerous published works so far about memristive logic [7]. Usually, the proposed circuits constitute hybrid CMOS-memristor implementations that optimize certain performance metric(s) [8]. However, such designs not always aim towards an in-memory computing approach, since the circuits proposed are not crossbar-compatible. On the other hand, several *stateful* memristive logic families have been proposed which use resistance states as the logic input/output. IMPLY logic (with the {IMP, FALSE} set) was probably the first such approach proposed in [9]. MAGIC is another scheme proposed later [10], among many other styles found in the literature. All such design schemes are based on a similar circuit topology for the implementation of logic gates, with memristors connected in series and/or in parallel, and functionally depend on the concept of "*conditional switching*"; i.e. the fact that some memristor (holding the result of computation) conditionally switches its resistive state depending on the state of other memristors (holding the inputs) upon the application of control voltages.

Metrics proposed so far for the evaluation of such logic families focus on latency, energy, and area efficiency, likewise it happens with conventional CMOS circuits. However, the correctness of the logic computations is degraded in the presence of device variability. In fact, according to a recent study [11], IMPLY- (and its extensions) and MAGIC-based logic computations could present reliability issues. Moreover, endurance of memristors is rarely given any importance. This is normal though; despite many researchers have been theorizing over the last decade about memristor-based circuits and applications, probably we are only just now actually learning how to use them properly [12]-[14]. Memristor technology matures day after day, but memristors still have functional imperfections, mostly owing to variability. Thus, either we will make use of them as they are today, or we will simply keep waiting for the perfect memristor to be invented.

That said, the focus of this work is placed on *robust memristor-based in-memory computations* and particularly on a ratioed logic scheme, recently introduced in [15], which is inspired on the pseudo-NMOS logic design. The reason we highlight this scheme is because it is simple, crossbar-

This work was supported in part by the Chilean research grants ANID FONDECYT INICIACION 11180706, and ANID-Basal Project FB0008.

compatible, tolerant to variability, compatible with any memristor type (of filamentary or interfacial switching type [16]), and it does not affect the device endurance. The latter is possible because *logic operations are performed by just reading the state of memristors* that act as inputs to the logic functions, thus avoiding the unwanted conditional switching. We report on all the advantages offered by this scheme, compared to IMPLY and MAGIC, and present LTSPICE-based circuit simulation results concerning a 1-bit full adder implementation using a threshold-type bipolar memristor model [17]. Therefore, this work provides *practical solutions for a viable in-memory computing architecture within the reach of today´s technology*, through a variability-tolerant ReRAM-based ratioed combinational logic design scheme.

II. Motivation

Memristors are considered the key enabling technology for computational memories, to be used as on chip accelerators in future computing platforms. Moreover, the polarity-dependent dynamic switching of memristors connected in series has been the basis for several relevant works [18], including IMPLY and MAGIC. Nevertheless, given the currently observed levels of variability, it is not viable to rely logic computations on conditional switching of memristors (or of groups of memristors) connected between them in series or to other resistive elements. Even more critical this situation becomes when chained logic operations are required; an intermediate "regenerating" step is necessary to make sure the device that conditionally switched has the appropriate resistive level, otherwise schemes such as MAGIC or IMPLY result unacceptably error-prone, as shown in [11].

Moreover, one of the major limitations of several stateful logic methods is not only variability of the devices but also the limited endurance of target memristor devices. In this direction, the focus has been lately placed on schemes such as *Scouting Logic* [19] which alleviates the endurance requirement through a modified sense amplifier for the implementation of different logic gates. However, performing logic computations without switching the states of the involved memristors was first seen in the so-called CMOS-*like* memristive logic [20]. The latter was revisited recently in [21], where the authors presented an improved ratioed version of the same concept, very similar to the one highlighted in this work, but still they assumed conditional switching of memristors and an array-incompatible design, eventually ruining the concept of in-memory computing.

Technology is progressing fast and the first steps towards fully memristive ALUs were made recently, with important efforts placed towards the creation of proper synthesis methods for in-memory computing architectures. So, here we briefly describe the ratioed logic design scheme and highlight all the advantages it offers considering the *desired characteristics for a universal memristor-based logic design*, i.e.: (i) area/energy/time-efficiency, (ii) device technology independency, (iii) crossbar array-compatibility, (iv) reliability, (v) switch-less logic operations (thus also non-destructive for the input data), (vi) cascadable logic operations, (vii) high fan-in, (viii) re-configurability (to realize different functions without changing the circuit topology), and (ix) support column/row-wise logic operations in crossbar (to improve the flexibility of information processing). ALL these properties are well supported by the presented logic scheme!

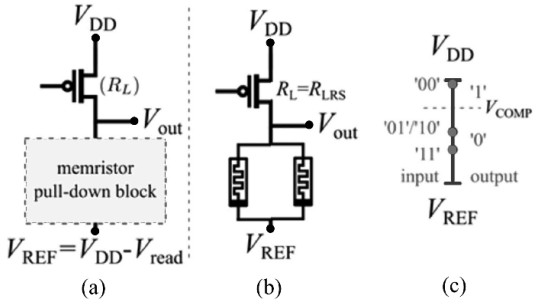

Fig. 1 (a) General schematic for the ratioed logic scheme. (b) A particular example of it for a 2-input (2 memristors) NOR gate. (c) A graph showing the expected output voltage levels for every input logic combination.

III. ReRAM-based Ratioed Logic Design

For a more proper description, we assume to be using threshold-type switching 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). A SET process (HRS→LRS) occurs when the device is forward biased with a voltage of amplitude higher than a V_{set} threshold, whereas a RESET process (LRS→HRS) occurs when it is reverse-biased with a voltage amplitude higher than a $|V_{reset}|$ threshold. Polarity is defined by the thick black line in the memristor symbol.

The general implementation schematic is shown in Fig. 1(a), which concern a PMOS transistor connected to a pull-down memristor network. The memristors substitute the transistors in the NMOS pull-down block of a conventional pseudo-NMOS ratioed logic gate. A memristor in LRS substitutes a turned-ON transistor, whereas a memristor in HRS substitutes a turned OFF transistor. A particular example of a two-input NOR gate (NOR2) is shown in Fig. 1(b). The state of the memristors is the input to this logic gate. The universal NOR operation is performed by applying a read voltage across the entire network, while making sure that the voltage drop on the memristor pull-down block will not disturb their states (switch-less logic).

The overall circuit works as a voltage divider. The pull-up PMOS transistor is tuned to exhibit a channel resistance $R_L \approx R_{LRS}$ since ratioed circuits depend on proper pull-up/down resistance for correct operation. Contrary to other stateful logic styles, such as IMPLY or MAGIC, the logic output here is not directly stored in a memristor but instead it comes in form of voltage V_{out} described by (1), where R_{eq} is the equivalent resistance of the whole memristor network, and R_L is the equivalent PMOS channel resistance.

$$V_{out} = V_{REF} + (V_{DD} - V_{REF}) \frac{R_{eq}}{R_L + R_{eq}}. \quad (1)$$

As shown in Fig. 1(c), when inputs are "00" (both memristors in R_{HRS}) V_{out} is roughly V_{DD}, which is interpreted as logic '1'. However, if at least one input is '1' (R_{LRS}) then $V_{out} < [(V_{DD} - V_{REF})/2 + V_{REF}]$ falls in the range corresponding to logic '0'. V_{out} is compared to a threshold

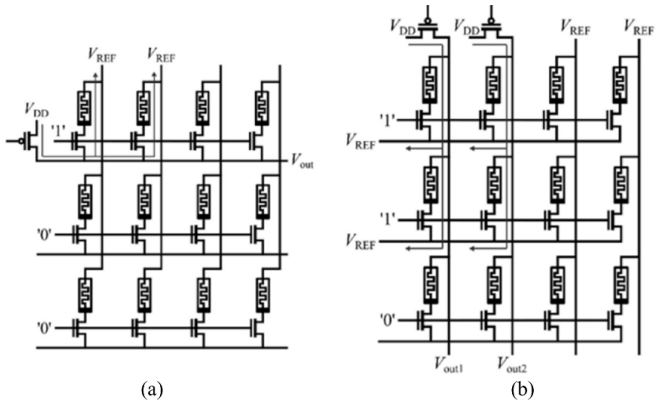

Fig. 2 Ratioed NOR logic gates in a 1T1R crossbar with memristor-transistor cross-point cells (transistors are selector devices). (a) Row-wise NOR2 using the leftmost 2 memristors of the 1st row. (b) Two column-wise NOR2 operations in parallel; one uses the top 2 memristors in the 1st column; the other uses the top 2 memristors of the 2nd column (adapted from [11]).

value V_{COMP} via a comparator in the peripheral circuitry, properly selected so that the V_{out} voltage levels for '0' and '1' are clearly distinguished. Computation is fast as it is equivalent to a simple read-out phase. NOR operations are crossbar-compatible. Fig. 2 shows how they can be executed in a row- or column-wise manner within a 1T1R crossbar array, with the possibility of parallel operations as well.

As far as re-configurability is concerned, for instance, OR gates are possible just by including an inverter stage after the comparator in the periphery, without any modification to the circuit structure. With the OR/NOR gates directly available, a particular case for one input (1 memristor) is the COPY/NOT gate. This allows to copy the content (original or inverted) of any cell to another in a single step, unlike MAGIC or IMPLY whose corresponding COPY operation lasts longer. It is worth noting that the logic functions can be also reconfigured at the hardware level by merely changing the applied control voltages (we leave this out of scope for readability reasons).

The result of computation, whenever required, is stored back to a memory element right after the read-phase through an additional programming step. This way we guarantee that the logic output is always converted to the corresponding resistive level in a reliable manner, without any conditional switching taking place, which will most likely suffer from variability effects. Note also that, even though not shown here, the reader should bear in mind that in any practical ReRAM architecture, the programming SET/RESET stage will most likely incorporate a read phase first, so that over stressing of devices is avoided. For instance, you will not apply a RESET pulse to a device already in HRS, nor a SET pulse to a device already in LRS; otherwise stuck-at-ON/OFF faults might appear [11]. Therefore, the result of any logic computation will be always properly written to a memristor, and no previous initialization of the output memristor is required.

We show in Fig. 3 a schematic with a simplified peripheral circuit enabling NOR/NOT/COPY logic operations. The memory (logic) operation is performed when the "write/op" signal is '1(0)', so that the bottom electrode of the memristors is connected to V_{DD} (for logic) or ground (for memory operations). The logic output V_{out} is momentarily stored in a

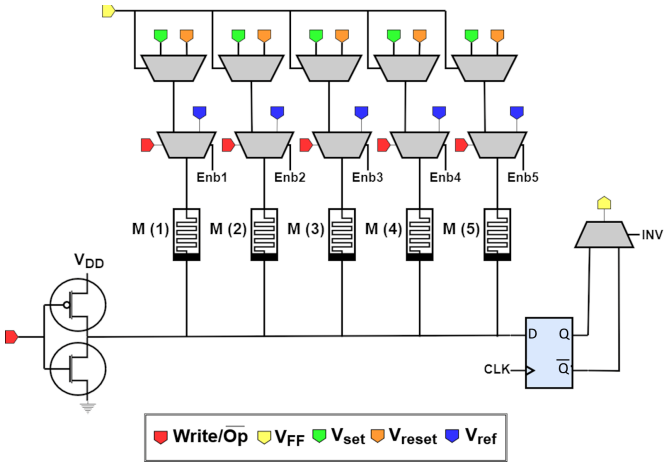

Fig. 3 Circuit schematic of a crossbar row along with the peripheral circuits required for memory and ratioed logic NOR*n* (*n*-inputs) operations. Logic output V_{out} is taken at the horizontal row-line driven to the input of a FF.

flip-flop (FF), thus both the original signal and its complement are readily available. After the read (logic) operation, with "write/op" = '1' we connect the common row-line to ground so that we can perform a memory operation and write the logic output (original or inverted) to a specific memristor by means of the enable signals (Enb_X). The INV input allows obtaining the inverted output; the selected FF output V_{FF} is used to select between V_{set}/V_{reset} programming voltages, so that we can write a logic '1'/'0', which also enables the NOT and COPY (NOT + set INV = '1') operations. The FF and the comparator certainly add some area and power overhead, but the rest of the peripheral circuitry is a typical crossbar array driver.

It is worth mentioning that, since output is expressed in voltage instead of memristance, conducting chained operations in this scheme is possible owing to the intermediate write step which allows storing the logic output of one stage to a memristor, so that is can later act as input for the next stage. Therefore, in principle the delay of every gate in chained logic operations is two steps (read + write). However, consider that read phase is much faster than write phase, so practically it is reasonable to say that the delay of a logic operation is the same as it results in other logic schemes, such as MAGIC or IMPLY, where conditional switching of memristors is required (equivalent to a single write phase). Last, we highlight the fact that any type of memristors could be used in such a ratioed scheme; this is not the case for MAGIC or IMPLY for example, where threshold-type switching is required, along with a particular relation for the SET/RESET threshold voltages that must hold for the operations to be carried out correctly (read further in [11]).

IV. –SPICE Circuit Simulation Results

In this section we present LTSPICE-based circuit simulation results for a 1-bit Full Adder (FA) design, as a case study of a combinational circuit implemented in chained NOR logic. Fig. 4 shows the logic design of a decomposed FA in a multi-level circuit comprising two Half Adders (HA). We simulate the FA function using the circuit shown in Fig. 3 in LTSPICE, corresponding to a single crossbar row. For the peripheral blocks, behavioral description was used to

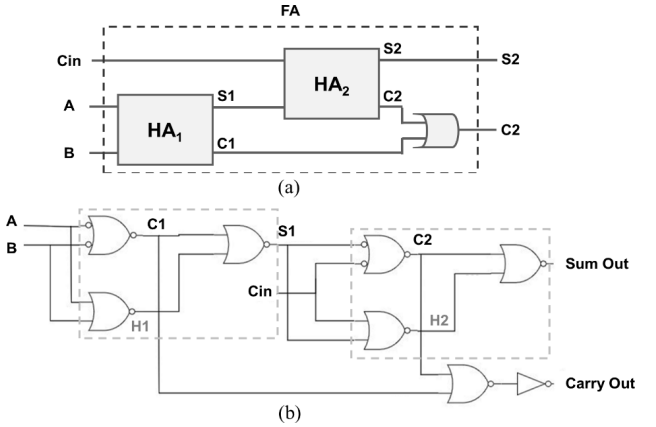

Fig. 4 (a) Block diagram of a 1-bit Full Adder comprising 2 Half Adders. (b) Circuit schematic corresponding to (a) implemented in NOR logic. S_xC_x are the Sum and Carry outputs of HA_x, whereas H_x is just an internal signal. Blocks with dashed lines show the gates corresponding to every HA stage.

TABLE I. 1-BIT FULL ADDER COMPUTING STEPS FOR RATIOED LOGIC

Step	Operation	M1	M2	M3	M4	M5
0	*INIT*	$\boldsymbol{A}$	$\boldsymbol{B}$			$\boldsymbol{Cin}$
1	*(M1)NOR(M2) →M3*	A	B	$H1$		Cin
2	*NOT(M1) →M1*	$\bar{A}$	B	$H1$		Cin
3	$NOT(M2) \rightarrow M2$	$\bar{A}$	$\bar{B}$	$H1$		Cin
4	$(M1)NOR(M2) \rightarrow M4$			$H1$	$C1$	Cin
5	$(M3)NOR(M4) \rightarrow M1$	$S1$			$C1$	Cin
6	$COPY(M5) \rightarrow M2$	$S1$	Cin		$C1$	
7	$COPY(M4) \rightarrow M5$	$S1$	Cin			$C1$
8	*(M1)NOR(M2) →M3*	$S1$	Cin	$H2$		$C1$
9	*NOT(M1) →M1*	$\overline{S1}$	Cin	$H2$		$C1$
10	$NOT(M2) \rightarrow M2$	$\overline{S1}$	$\overline{Cin}$	$H2$		$C1$
11	$(M1)NOR(M2) \rightarrow M4$			$H2$	$C2$	$C1$
12	$(M3)NOR(M4) \rightarrow M1$	*Sum Out*			$C2$	$C1$
13	*NOT((M4)NOR(M5)) →M2*	***Sum Out***	***Carry Out***			

minimize simulation overhead and emphasize on the logic function of the circuit, rather than on the impact of MOS parasitics, which we assumed negligible. In this FA example, memristors 1 and 2 have the input data, whereas the memristor 5 has the *Carry-in* (*Cin*) bit. Sum and Carry outputs of the FA will be stored in memristors 1 and 2, respectively. This decomposed FA version was purposely selected to highlight reusability of cross-points and repeatability of execution in chained logic computations, which can further simplify synthesis of complex logic functions. Another reason why we chose to store the result in the same memristors that hold the input data, was in order for computation to be carried out with the minimum number of required memristors; 2 more are necessary if the initial input data should be kept in memory.

Table I enumerates the sequential steps followed to realize the FA of Fig.4(b) using 5 memristors (M(1) through M(5) in Fig. 3), describing the operation executed in every step and the corresponding result/signal stored in the state of the memristors. For instance: NOT(M1)→ M4 reads "the logic state of M1 is inverted and is stored in M4". Cells left blank mean that the state of that memristor is not important for the computation. Steps 1 through 5 correspond to the first HA, whereas steps 8 through 12 correspond to the second HA. Step 13 corresponds to the stage where the *Carry out* is computed and stored in M2. Note that the "read + write" sequence for every logic computation corresponds to one row/step in Table I. Following the steps in Table I, Fig. 5 shows the corresponding simulation results for the FA implemented using the circuit shown in Fig. 3. More specifically, we considered inputs A = '0', B = '1', and Cin = '1'. The evolution of the resistive state of all memristors is shown in Fig. 5(b). Fig. 5(a) shows the evolution of the control signals "write/op and Enb_x" for the execution of the computing steps. At t = 0s, we assume all memristors are in LRS (logic '1'), so we first program only M1 to HRS (for logic '0'). Next, follow the operations corresponding to the first HA. We set Enb_1 and Enb_2 to logic '1' while write/op is '0' to perform a logic NOR2 on M1 and M2, and then we set both write/op and Enb_3 to '1' to write the logic output ('0' in this case) to M3 by applying a V_{reset} pulse, as described in step 1 of Table I. In the same way, next we perform a NOT to M1 and write the result back to M1. Once the first HA operations are complete, an intermediate phase copies *Cin* to M2 and the *carry* of the first HA (*C1*) to M5. This way, the second HA follows the exact same steps on the same memristors like in the first HA; the synthesis of the logic operations can be quite simplified this way. However, these two copy operations could be avoided without loss of generality to complete execution in just 11 steps instead of 13. The last operation stores the *Carry out* bit to M2, whereas M1 holds the *Sum out* bit. For all memristors we used the threshold-type model of bipolar memristors of [17]. Based on the switching time of memristors, the Enb_x pulse-width was selected equal to 150ns, with 150ns pulse-separation selected arbitrarily for readability of the results.

We evaluated the same circuit implementation following the MAGIC and IMPLY schemes, which both required a minimum of 6 memristors. IMPLY is a well-studied scheme and similar results for a FA implementation were found in [9], thus were not repeated here. We confirmed that the FA implementation requires 29 steps! On the other hand, MAGIC requires the same number of steps with the ratioed approach if we assume that all memristors that act as output to hold intermediate results are already properly initialized in HRS; otherwise, intermediate RESET steps are required, which increase the total steps in 22. This comparison assumed always an implementation using the minimum number of memristors possible. If this is not a limitation, then MAGIC and IMPLY could perform better; IMPLY operations IMP and FALSE are highly parallelizable [9]. But destructive and/or error-prone operation owing to variability impact in conditional switching, is definitely a weakness of both schemes. Last, we wish to comment on the fan-in possibility of the ratioed logic scheme. According to [11], the NOR gate could support up to tens of inputs with still a reliable read-out operation, something not possible in the other two schemes. Everything considered, the discussed ratioed logic scheme in practice outperforms IMPLY and MAGIC in all considered metrics, with power efficiency being the most difficult to evaluate but expected similar to that of MAGIC computations.

V. CONCLUSIONS

This work presented a practical solution for the robust implementation of in-memory computing, based on a ratioed

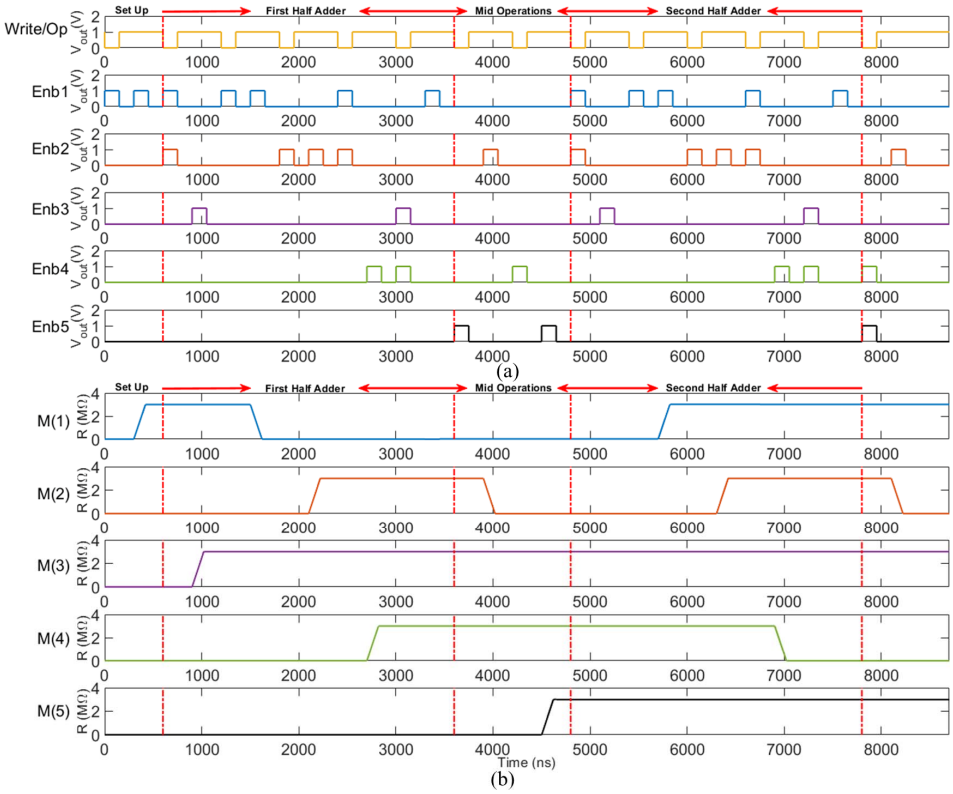

Fig. 5 Simulation results for the execution steps of a 1-bit Full Adder as described in Table I. (a) Control signals applied for the execution of the processing steps, to the enable signals and the write/op input. (b) Time evolution of the memristance of the 5 memristors considered, as shown in the circuit schematic of Fig. 3. Memristor model parameters [17] were as follows: beta = 5e13, Ron = 5kOhm, Roff = 3e6 Ohm, and threshold voltage for SET and RESET Vt = 0.5V.

logic design compatible with crossbar-based ReRAM. To the best of our knowledge, the presented scheme is essentially the only one which complies with ALL the required properties for a universal ReRAM-based logic family (see Section II).

REFERENCES

[1] M. A. Nugent et al., "Thermodynamic-RAM technology stack," *Int. J. Parallel, Emergent Distrib. Syst*., vol. 33, no. 4, pp. 430–444, 2018

[2] H. Li, T. F. Wu, S. Mitra, and H.-S. P. Wong, "Resistive RAM-centric computing: Design and modeling methodology," *IEEE Trans. Circuits Syst. I, Regular Papers*, vol. 64, no. 9, pp. 2263–2273, Sep. 2017

[3] C. Li, et al., "In-Memory Computing with Memristor Arrays," *in Proc. IEEE Int, Memory Workshop* (IMW), Kyoto, Japan, 13-16 May, 2018

[4] J. Reuben *et al*., "Memristive logic: A framework for evaluation and comparison," *in Proc. 27th Int. Symp. Power Timing Model., Optim. Simul*., Thessaloniki, Greece, Sep. 2017, pp. 1–8

[5] H.-S. P. Wong, *et al*., "Metal-Oxide RRAM," *IEEE Proc*., vol. 100, no. 6, pp. 1951-1970, 2012

[6] S. Hamdioui, *et al*., "Applications of Computation-In-Memory Architectures based on Memristive Devices," *in Proc. Design, Autom. & Test in Europe Conf. & Exhibition* (DATE), Florence, Italy, 2019

[7] I.Vourkas and G. Ch. Sirakoulis, "Emerging memristor-based logic circuit design approaches: A review," *IEEE Circuits Syst. Mag.*, vol. 16, no. 3, pp. 15–30, Third Quarter 2016

[8] Y. Zhou, *et al*., "A hybrid memristor-CMOS XOR gate for nonvolatile logic computation," *Phys. Status Solidi A*, vol. 213, No. 4, pp. 1050–1054, 2016

[9] S. Kvatinsky, *et al*., "Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies," *IEEE Trans. on VLSI Syst*., Vol. 22, No. 10, pp. 2054 - 2066, 2014

[10] S. Kvatinsky, *et al*., "MAGIC—Memristor aided LoGIC," *IEEE Trans.Circuits Syst. II, Exp. Briefs*, vol. 61, no. 11, pp. 895–899, 2014

[11] M. Escudero, I. Vourkas, A. Rubio, and F. Moll, "Memristive Logic in Crossbar Memory Arrays: Variability-Aware Design for Higher Reliability", *IEEE Trans. Nanotechnol*., vol. 18, pp. 635-646, 2019

[12] Knowm Inc., [Online]. Available: https://knowm.org

[13] M. Lanza, *et al*., "Recommended Methods to Study Resistive Switching Devices," *Adv. Electronic Materials*, Vol. 5, No. 1, pp. 1800143, 2019

[14] N. TaheriNejad, *et al*. "From Behavioral Design of Memristive Circuits and Systems to Physical Implementations," *IEEE Circuits Syst. Mag*, vol. 19, no. 4, pp. 6 - 18, 2019

[15] M. Escudero, I. Vourkas, A. Rubio, and F. Moll, "Variability-tolerant memristor-based ratioed logic in crossbar array," *in Proc. Int. Symp. Nanoscale Architectures*, Athens, Greece, Jul. 2018, pp. 13–18

[16] G. Sassine, *et al*., "Interfacial versus filamentary resistive switching in TiO2 and HfO2 devices," *J. Vac. Sci. Technol*. vol. B34 (012202), 2016

[17] Y. Pershin and M. Di Ventra, "SPICE model of memristive devices with threshold," *Radioengineering*, vol. 22, no. 2, pp. 485–489, 2013

[18] I. Vourkas, *et al*., "On the Generalization of Composite Memristive Network Structures for Computational Analog/Digital Circuits and Systems" *Microelectronics J*., vol. 45, no. 11, pp. 1380-1391, Nov. 2014

[19] L. Xie *et al*., "Scouting logic: A novel memristor-based logic design for resistive computing," *in Proc. IEEE Comput. Soc. Annu. Symp. VLSI*, Bochum, Germany, Jul. 2017, pp. 176–181

[20] I. Vourkas and G. Ch. Sirakoulis, "A novel design and modeling paradigm for memristor-based crossbar circuits," *IEEE Trans. Nanotechnol*., vol. 11, no. 6, pp. 1151–1159, Nov. 2012

[21] X. Cui, *et al*., "The Synthesis Method of Logic Circuits based on the NMOS-like RRAM Gates," *IEEE Access* (2020), *in press*