Memristive Computational Memory Using Memristor Overwrite Logic (MOL)

Khaled Alhaj Ali, Mostafa Rizk, Amer Baghdadi, Jean-Philippe Diguet, Jalal Jomaah, Naoya Onizawa, Takahiro Hanyu

Memristive Computational Memory Using Memristor Overwrite Logic (MOL)

Khaled Alhaj Ali, Mostafa Rizk, Amer Baghdadi, Jean-Philippe Diguet,
Jalal Jomaah, Naoya Onizawa, Takahiro Hanyu

***Abstract*—In this paper, we present a novel logic design style, namely memristor overwrite logic (MOL), associated with an original MOL-based computational memory. MOL relies on a fully digital representation of memristor and can operate with different memristive device technologies. Its integration in memristive crossbar arrays and computational memories allows the execution of bit and vector-level primitive logic operations in two computational steps at most. Promising features and performances are demonstrated through the implementation of N-bit full addition using the proposed MOL-based computational memory.**

***Index Terms*—Memristor, Memristor Overwrite Logic (MOL), In-memory computation, Crossbar array, Logic design.**

I. Introduction

MEMRISTOR has been predicted theoretically by Leon Chua [1] in 1971. Chua hypothesized that memristor which is the fourth passive device should exist and hold a relationship between magnetic flux and charge. The first fabrication of a memristor device has been developed by a research team at Hewlett-Packard (HP) Labs [2] in 2008. The device structure is comprised of a stoichiometric (T_iO_2) and an oxygen deficient (T_iO_{2-x}) layer sandwiched between two platinum electrodes. The obtained two-terminal nano-device exhibits a dynamic resistance that can be modulated between two bounds. These bounds correspond to the low and high resistance states, and are referred to as R_{ON} and R_{OFF} respectively. Memristor possesses the ability to retain the last attained resistance value in a non-volatile manner.

Given the nanoscale dimensions of memristors and their unique properties, several innovative applications have emerged. One of the most promising applications is the use of memristors to implement arithmetic blocks, such as full adders [3][4][5]. Compared to pure CMOS implementations, these blocks are of relatively high density and could be packed into small chip area. Other applications involve using memristor-based memories to allow processing within the storage cells. This approach, referred to as in-memory computing, is being explored recently to alleviate time and energy cost of data movement encountered in conventional von Neumann architecture and aggravated by the recent growth in data-centric applications [6]. The concept is different from, yet can be complementary to, that of near-memory computing which dates to the 1990s [7]. Near-memory computing approach aims to place the processing units physically closer to the memory, for example through advanced die stacking technologies or three-dimensional (3D) integration. Despite the reduction in time and distance to memory access, there still exists a physical separation between the memory and the compute units [8]. For in-memory computing, instead of sending large amount of data to the processing cores, part of the tasks are computed in place inside the memory itself [9]. Depending on the application, this can reduce the computational complexity of these tasks and/or the amount of data being accessed, leading to significant performance improvement [6].

In this context, several recent contributions have been proposed to enable computation within memristive memory arrays and can be classified in two categories. The first category involves using the memristor as single-level cell (SLC) [10–16]. The second category includes work that uses the memristor as multi-level cell (MLC) or analog cell [17–19]. MLC-based computing is promising when targeting applications with intensive multiply-accumulate operations, such as convolutional neural networks (CNN) [19]. However, a number of challenges remain in terms of manufacturability and computational accuracy regarding device variability, pattern-dependent current leakage and the area overhead of peripheral circuits [20]. Major semiconductor foundries have not included MLC technology in their development roadmaps in the near future [19]. In contrast, SLC cells have a larger readout margin that makes them tolerant against process variation and resistance drift effects. Based on SLCs, different logic design styles have been introduced together with different realizations on memristive crossbar arrays. The Material Implication (IMPLY) [10] and the Memristor Aided loGIC (MAGIC) [21] have been introduced to enable in-memory logic operations. Although promising results are demonstrated, MAGIC and IMPLY techniques still impose specific technology and design constraints. For instance, in order to attain acceptable performance in these techniques, the ratio R_{OFF}/R_{ON} of the adopted memristive devices should be relatively high. Moreover, authors of [22] have reported that IMPLY does not ensure binary resistance switching of memristors in some cases. More recently, other in-memory computing techniques have emerged as alternatives. Among these, the memristor-based majority (MAJ) [23] has been introduced to overcome the aforementioned limitations. However, other

K. Alhaj Ali and A. Baghdadi are with IMT Atlantique, CNRS Lab-STICC laboratory, Brest, France. M. Rizk is with IMT Atlantique and the School of Engineering, International University of Beirut, Lebanon. J. Diguet is with CNRS Lab-STICC laboratory, Université de Bretagne-Sud, Lorient, France. J. Jomaah and M. Rizk are with Physics Department, Faculty of Sciences, Lebanese University, Beirut, Lebanon. N. Onizawa and T. Hanyu are with the Research Institute of Electrical Communication, Tohoku University, Sendai, Japan. (e-mail: khaled.alhaj-ali@imt-atlantique.fr)

This work is an extension of the following publication: khaled Alhaj Ali *et al.*, Crossbar Memory Architecture Performing Memristor Overwrite Logic, 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), November 2019, Genoa, Italy.

downsides arise at the architectural level. MAJ design style is relatively complex in terms of peripheral circuits as well as excessive in-out data movement which in turn impacts latency.

In this work, we introduce a novel logic design style, namely memristor overwrite logic (MOL), associated with an original MOL-based computational memory. MOL combines the simplicity of MAGIC/IMPLY techniques and the accuracy of MAJ. MOL can operate with different memristive device technologies and allows for significant reduction in the number of required memristors and computational steps.

The rest of the paper is organized as follows. Section II provides a brief survey on existing memristor based logic design styles. Section III presents our proposed MOL approach. Section IV discusses the integration of MOL into the conventional memory configurations. Section V presents our proposed configurable MOL-based computational memory architecture. The design and its configuration methodology are demonstrated by a case study of a N-bit full addition in Section VI. Simulations and performance analysis are illustrated in Section VII. Comparison with available implementations is presented in Section VIII. Finally, Section IX concludes the paper.

Although memristive devices encompass memristors, it is possible to use the term memristor for other memristor devices [24]. In this paper, we use the terms memristor and memristive device interchangeably for simplicity.

II. Memristive devices as computational elements

The versatile nature of memristors allows them to be used as computational elements in addition to their storage role. Implementing Boolean logic with memristors has been widely explored. Several memristor-based logic design styles have been introduced in the literature. Each is adapted for a specific type of applications and surrounded with specific limitations.

A. Logic design styles

The Memristor Ratioed Logic (MRL) has been proposed in [3]. MRL integrates memristors with CMOS transistors to implement combinational functional blocks. These blocks are relatively dense compared to those implemented with pure CMOS transistors. The Memristive Threshold Logic (MTL) has been studied in [25]. The gate uses the configurable conductance of memristors to represent weights during operation. However, these weights are very sensitive to state drift, which can be a critical issue [25]. It is considered simple, but still in preliminary stages of fabrication. IMPLY [10] and MAGIC [21] are intended for in-memory computing. In these design styles, a memristor serves as a memory element as well as a part of a computational gate inside the memory. MAD gate, or Memristors-As-Drivers gate has been presented in [26]. MAD has been introduced to overcome the long delays of the IMPLY operations as well as signal degradation and buffering issues in MRL; however, each MAD gate requires a complex driving circuitry and is thus considered unsuitable for integration inside a memristive memory. MAJ has been proposed in [23]. The authors demonstrated that a single memristor is capable of performing a 3-variable majority function. Using additional inversion function (INV), a Boolean expression is represented using majority-inverter graph (MIG). MIGs are then realized sequentially in conventional memristive crossbar arrays. The complementary resistive switches (CRS) logic has been presented in [27]. CRS logic is capable of realizing two primitive operations denoted as reverse implication (RIMP) and inverse implication (NIMP). This logic design style can be considered as a special case of MAJ (see Section III-B).

In this paper, our target application concerns in-memory computing, so some logic design styles such as MRL, MAD and MTL are excluded.

B. Limitations

MAGIC and IMPLY logic families are widely explored in the literature. Authors of [12][28–31] have presented several approaches where logic functions are broken down into several MAGIC or IMPLY operations. These operations are then performed sequentially inside memristive crossbar arrays. However, these approaches have several design constraints:

- The analysis in [22] reveals that IMPLY cannot achieve the full resistance switching of the output memristor in case both input memristors of the IMPLY gate are in the R_{OFF} resistance state. Hence, the corresponding state of the output memristor is not fully digital.
- Output memristors in IMPLY and MAGIC may be subjected to state drift [10][12].
- The performance of these design styles is highly dependent on the technology of the adopted memristive device (e.g. requirement of memristive devices with high R_{OFF}/R_{ON} ratio) [10][12].
- The corresponding basis functions provided by IMPLY and MAGIC are not diverse enough to allow fast logic mapping with minimum computational cycles.

MAJ-based logic design has been recently explored by several authors [23][15]. MAJ relies on a digital representation of memristors, so the limitations faced in IMPLY and MAGIC can be overcome. However, at the architecture level, other downsides arise:

- In-memory computing architectures based on MAJ, which are available in the literature, require additional load operations, which read data bits outside the memory. This induces the overheads in terms of total critical path, number of cycles and the complexity of the dedicated control unit.
- Architectures based on MAJ involve significant modifications in the peripheral circuitry of the memory. The write operations are performed on bit-lines (BLs) as well as word-lines (WLs) instead of BLs only.

These limitations hold also for CRS logic design approach [27] as it can be considered as a special case of MAJ.

III. Proposed MOL logic

In this section we introduce a new memristor-based logic design style namely Memristor Overwrite Logic (MOL). MOL approach is highly adapted for computing within memristive crossbar arrays and avoids the limitations encountered by pre-existing logic design styles.

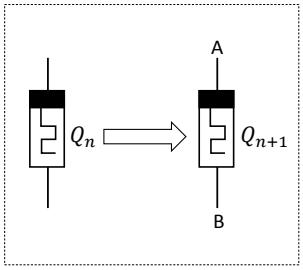
A. Digital representation of memristive devices

The nonvolatile internal resistance state of memristor could be changed according to the magnitude and duration of the applied bias across its terminals [32]. However, a non sufficient magnitude or duration leads to an intermediate resistance state R where $R_{ON} < R < R_{OFF}$. In this case, the state of the memristor can not be considered as binary, which in turn leads to more sophisticated modeling of the internal state of memristive devices in the analog domain. However, in a digital design, we could think about the memristor as a two-state element where its resistance $R \in \{R_{ON}, R_{OFF}\}$ and ignoring any other intermediate states if we succeed to guarantee a sufficient magnitude and duration of the bias across its terminals. Based on this understanding, the internal state of a memristor is defined in the digital domain. Let Q_n be the current internal state of a memristor while Q_{n+1} is the next state after applying a new external bias represented by A and B as shown in Fig. 1(a). Hence, Q_{n+1} will a be function of the logical states at the terminals A and B and the previous internal state Q_n. By considering all the possible combinations of A, B and Q_n as shown in Fig. 1(b), the state equation of a memristive device is expressed as follows:

$$Q_{n+1} = Q_n A + Q_n \overline{B} + A\overline{B} = M_3(A, \overline{B}, Q_n) \qquad (1)$$

where M_3 represents the 3-variable majority function, which is defined in [33]. This expression demonstrates that a majority function is an intrinsic feature of memristive devices [23]. Based

(a)



(b)

Q_n	A	B	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Fig. 1. Memristor: (a) internal state after applying external bias represented by A and B; (b) truth table

on the Boolean expression presented in (1), the equivalent latch circuit of a memristive device is shown in Fig. 2 where Q is the internal state of the memristor. To translate the Boolean value of Q into a resistance between the terminals of the memristor, an analog multiplexer is added. It selects either one of the two resistors, which resistances are R_{ON} or R_{OFF} where $Q = 0$ and $Q = 1$ are mapped to R_{ON} and R_{OFF} respectively. Note that this schematic is valid and useful from the digital perspective, so it cannot be used for simulation in the analog domain.

B. MOL logic procedure

The state representation of memristor expressed in (1) clarifies its computational capability and simplifies its integration in the digital domain. Six possible cases can be derived from (1) and are listed in (2). Fig. 3 is an illustration of these cases.

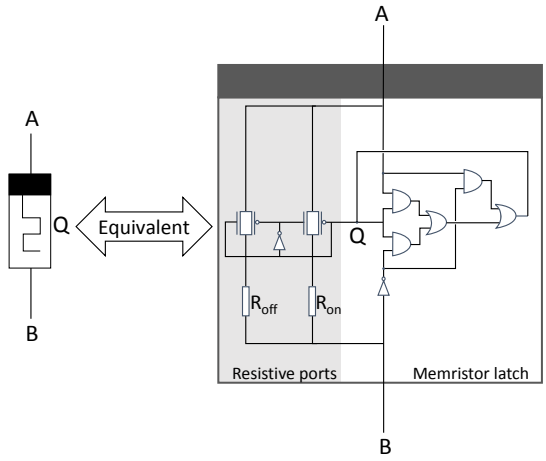


Fig. 2. Equivalent latch circuit of memristor with binary resistive ports

They are split into two groups. The first group includes the cases from 1 to 4, which correspond to MOL. In these 4 cases, a memristor acts as logic accumulator. The previously stored bit Q_n is subjected to OR/AND with the new input $A/\overline{B}$ while the other terminal of the memristor is set to logic "0" or logic "1" depending on the desired function. The obtained output is simultaneously saved in the form of new internal state Q_{n+1}. The remaining cases (i.e. 5 and 6) are achieved by initializing the memristor to a known state (logic "0" or logic "1"). The inputs A and B are sent to the memristor ports simultaneously. The output is saved as the new internal state (Q_{n+1}) of the memristor. In fact, these two cases correspond to CRS logic operations that are explored in the literature [16][23][27].

Although MOL operations are special cases of the 3-variable majority, working with MOL is much simpler. MOL highly resembles the conventional write operation. One end of each memristor is reserved for the input operands, while the other end is employed for selection. In contrast, MAJ employs both terminals of the memristor for the input operands. This makes MOL more adapted to crossbar memory arrays.

$$Q_{n+1} = \begin{cases} Q_n + A\,, & B = 0, \quad case:1 \quad (MOL) \\ Q_n A\,, & B = 1, \quad case:2 \quad (MOL) \\ Q_n + \overline{B}\,, & A = 0, \quad case:3 \quad (MOL) \\ Q_n \overline{B}\,, & A = 1, \quad case:4 \quad (MOL) \\ A\overline{B}\,, & Q_n = 0, \quad case:5 \quad (CRS) \\ A + \overline{B}\,, & Q_n = 1, \quad case:6 \quad (CRS) \end{cases} \qquad (2)$$

The same concept applies to a vector of bits. Fig. 4 illustrates that two consecutive steps are enough for achieving MOL operations on an N-bit vector. In step 1, which is presented in Fig. 4(a), the input vector $I = [I_{N-1}\ I_{N-2}...\ I_1\ I_0]$ is written into the N memristors by mapping logic "0" and logic "1" to the normalized voltage levels $-1V$ and $1V$ respectively while the common horizontal line is set to $0V$. At the end of this step, the resulting state of a given memristor M_k is $Q_k = I_k$. In step 2, the same N memristors are overwritten with the input vector $A = [A_{N-1}\ A_{N-2}...\ A_1\ A_0]$. However, the input voltage level on the common horizontal line is set to $0V$ or $1V$ depending on the desired operation. For the case of MOL-OR (Fig. 4(b)), B

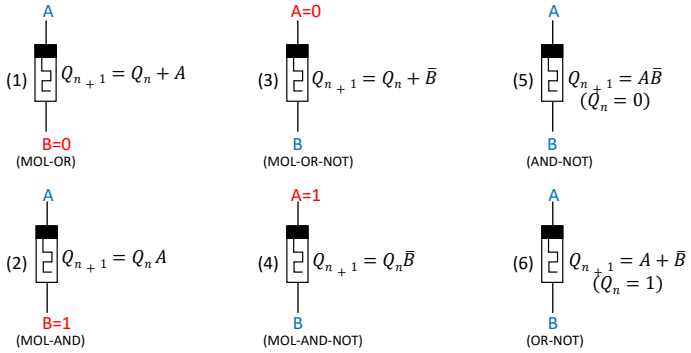


Fig. 3. Six possible logic cases performed by a memristor

is set to $0V$ and the result, which is stored in a given memristor M_k, is $Q'_k = A_k + I_k$. For the case of MOL-AND (Fig. 4(c)), B is set to $1V$ and the result, which is stored in a given memristor M_k, is $Q'_k = A_k I_k$.

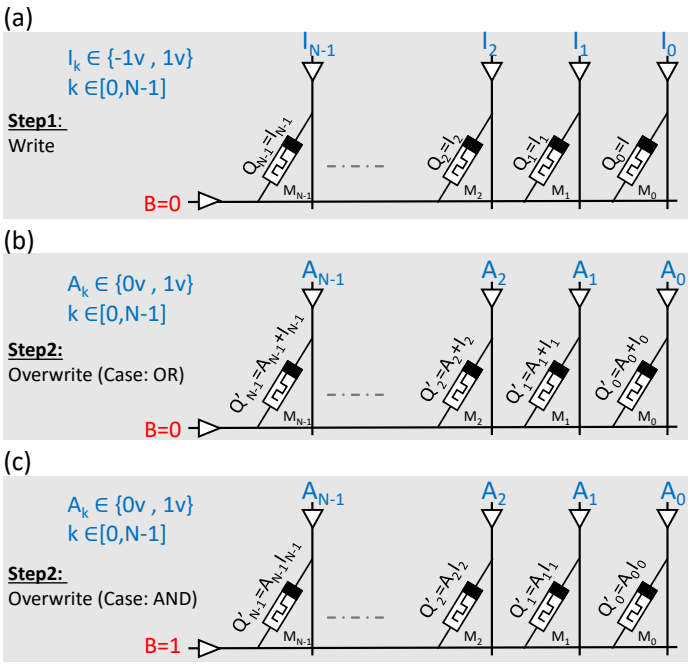


Fig. 4. Performing MOL on a vector of bits; (a) writing N-bits into memritsors; (b) overwrite step for MOL-OR; (c) overwrite step for MOL-AND

C. Performing MOL inside memristive crossbars

The proposed MOL can be performed in memristive crossbar arrays. The input data bits to the crossbar can be either written or combined logically with the currently stored bits inside the crossbar. This can be simply achieved by choosing the appropriate normalized voltage levels for representing the arriving bits (i.e. -1/1 for write and 0/1 for MOL). Fig. 5(a) illustrates that a single or multiple rows of the crossbar could be selected for either MOL-OR or MOL-AND operations with the incoming data bits $I = [I_{N-1}\ I_{N-2}...\ I_1\ I_0]$ being applied on the columns. Similarly, Fig. 5(b) shows that a single or multiple columns of the crossbar could be selected for either MOL-OR-NOT or MOL-AND-NOT operations with the incoming data bits of the vector I applied on the rows.

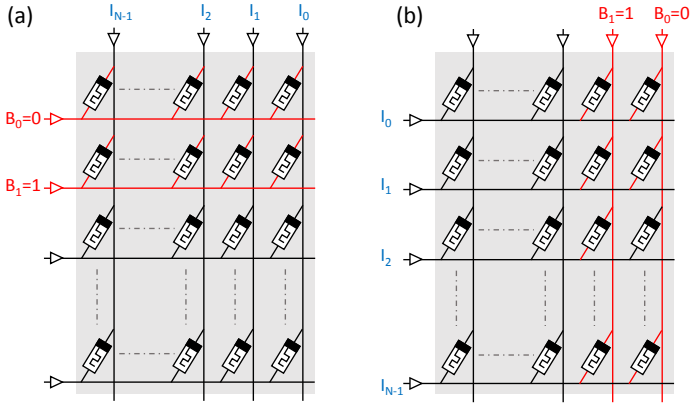


Fig. 5. MOL inside memristive crossbar: (a) MOL-OR or/and MOL-AND; (b) MOL-OR-NOT or/and MOL-AND-NOT

IV. Memory Architecture with MOL Capabilities

Crossbars constitute the core element of emerging memristive memories (e.g. RRAMs and MRAMs). Integrating MOL with crossbar-memory architectures can lead to promising enhancements and provides additional computational capabilities to these memories. However, this imposes updating memory peripheral drivers to cope with MOL operations in addition to its main storage function. Fig. 6(a) presents the proposed memory architecture which is capable of performing MOL. As illustrated in Section III, write and overwrite operations could be performed along the rows as well as the columns of the crossbar. However, in a conventional memory architecture, the flow of the incoming data bits is along bit-lines only while the word-lines are reserved for addressing. Thus, MOL operation, which is similar to a write operation, could be only performed along BLs. In this case, MOL-OR and MOL-AND are the only supported logic operations in the proposed memory architecture. The architecture shown in Fig. 6(a) can be configured in four different modes:

1- Write mode: The input N-bit vector $I = [I_{N-1}\ I_{N-2}...\ I_1\ I_0]$ is first mapped via bit-line driver (BLD) into the normalized voltage levels of $-1V$ and $1V$ corresponding for logic "0" and "1" respectively. Fig. 7(a) presents the schematic of BLD at the transistor level. The respective voltage levels ($-1V$ and $1V$) are then provided to the BLs of the memristive crossbar through the Isolation Block (ISO), which acts in this mode as a connecting switch. Fig. 7(b) illustrates the internal structure of ISO. Simultaneously, the enabled addressing decoder selects a single WL. The selected WL is supplied with a voltage V_{SEL}, which is already shared to the input of each transmission gate corresponding to every WL. The shared voltage V_{SEL} is set to the normalized voltage level of $0V$. The unselected WLs remain floating in the high impedance state (Z).

2- Overwrite mode: In this mode, the function of the memory is switched to perform MOL among its memristive crossbar. As stated above, both MOL-OR and MOL-AND have to be supported. For the case of MOL-OR, the input data bits are mapped to the normalized voltage levels of $0V$ and $1V$ corresponding for logic "0" and logic "1" respectively. The addressing decoder performs its normal selection function for a single WL. ISO is

kept at the connecting state. The level of V_{SEL} is also set to $0V$ as in the case of write mode. The resulting bits of the MOL-OR operation are simultaneously stored in the selected WL. MOL-AND is performed similarly but V_{SEL} is switched to the high voltage level (i.e. $V_{SEL} = 1V$).

3- Read mode: In this mode, a single WL is selected to sense the corresponding states of its allocated memristors individually. BLD is isolated using the ISO block, which acts in this case as an open switch. The selection voltage V_{SEL} is set to $0.5V$ (normalized). The sensing current generated through each memristor have to guarantee a stability of its internal state (no state drift). A sensing amplifier (SA) circuitry, whose architecture is illustrated in Fig. 7(c), is used to measure the voltages across the reference resistors of respective resistances R. R is chosen to be the mid value between R_{ON} and R_{OFF} (i.e. $R = (R_{ON} + R_{OFF})/2$). By considering $R_{ON} < R_{OFF}$, the voltage across a reference resistor, which is in series to the sensed memristor, would be either in the neighborhood of $0V$ or $0.5V$. Depending on the state of the sensed memristor, the three cascaded inverters magnify this difference leading to $-1V$ or $1V$ at the output.

4- Idle mode: In this mode, the memory is not active. The memristive crossbar is totally isolated to preserve its internal state. The IB block is in the isolation mode. Hence, all BLs are in the high impedance state (Z). Moreover, the address decoder is disabled. Thus, none of the WLs is selected, keeping them in the Z state.

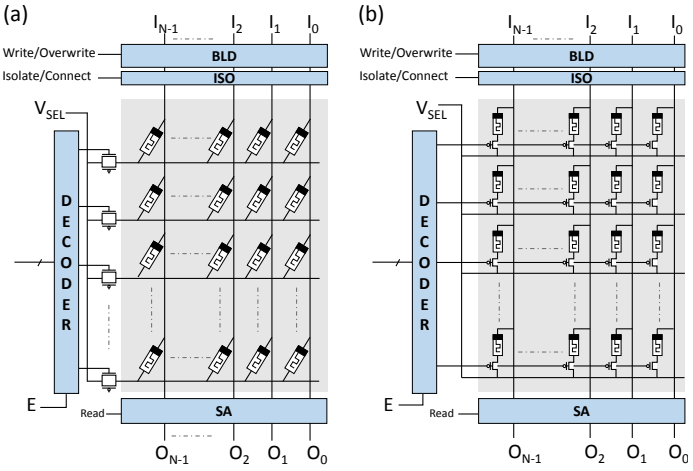


Fig. 6. Configurations of MOL memory architectures: (a) 1M; (b) 1T1M

The architecture presented in Fig. 6(a) adopts the 1-memristor (1M) configuration for the structure of the crossbar. In other words, each cell consists of one memristor which connects the vertical and horizontal nano-wires of the crossbar. However, the 1M crossbar configuration suffers from the sneak paths phenomenon [34]. Sneak paths correspond to current paths through unselected cells in a memristive array. These undesired paths lead in some cases to a drift in the state of unselected memristive cells during write or overwrite operations. Moreover, it gives false estimation about the real logical state of a given selected memristor during reading mode. This phenomenon degrades the overall memory performance. Several efforts have been devoted in the literature to overcome sneak path phenomenon [34] [35] [36]. All proposed methods are limited to a certain crossbar size. Thus, increasing the size of the memristive crossbar beyond a certain limit will eventually lead to the sneak paths. A possible solution to stop these paths is to use a selector in series with each allocated memristive cell. This solution induces overheads in terms of the total utilized area of the memory which in turn loses the ultra high density attained in the 1M case. In [37], a transistor is used as a selector. Thus, each cell

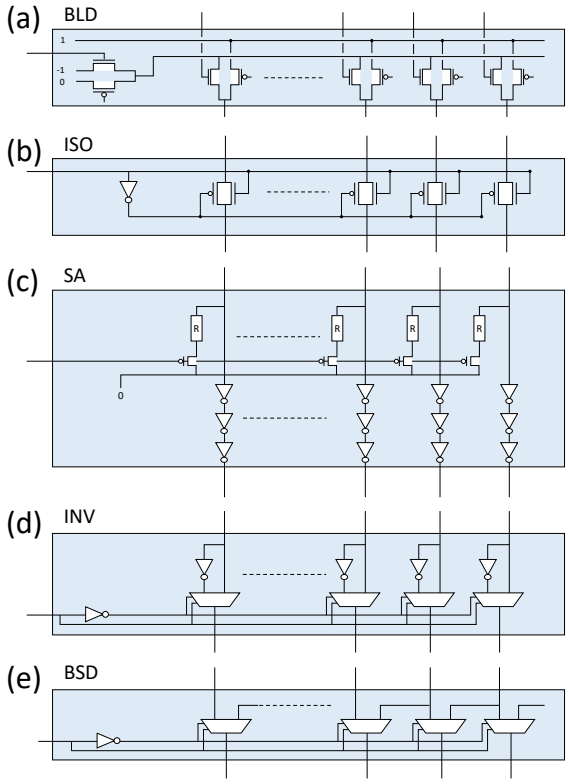


Fig. 7. Drivers architectures for the proposed MOL-memory approach

inside the memory consists of one transistor in series with one memristive device (1T1M). The obtained crossbar architecture for the 1T1M configuration is considered as sneak-path free. Fig. 6(b) presents our proposed 1T1M memory architecture with added MOL capabilities. The WL transmission gates, that have been used in the 1M case are no longer used in the case, of 1T1M memory architecture. Normally, each transmission gate is equivalent to two MOSFETs. Thus, for an $N \times M$ memristive crossbar array, additional $NM - 2N$ MOSFETs are used in the 1T1M architecture compared to that in the 1M case. The obtained 1T1M architecture has the same four control modes previously introduced for the 1M case.

V. MOL-BASED COMPUTATIONAL MEMORY

In this section, a MOL-based computational memory architecture is introduced. The architecture is able to perform MOL operations between two stored word lines. The original architecture, which is formed of two interconnected MOL memory blocks, works in a complementary manner.

A. Architecture

The proposed MOL-memory architectures, which are presented in section IV, act as logic accumulators for the newly arriving bits. In other words, computation in such memory is restricted for logic accumulation. Accordingly, performing general Boolean functions in this memory requires an additional process to load the stored data bits outside the memory. These additional load operations are at odds with the concept of computation inside the memory. To overcome these load operations, we propose the use of two coupled MOL memories (MOL-memory-A and MOL-memory-B), that work in complementary manner. At each time step, one of these memories acts as source of input data-bits of the second memory. The second memory performs MOL with the previously stored bits in its memristive crossbar. Fig. 8 illustrates our proposed computational-memory architecture. The architectures of MOL-memory-A and MOL-memory-B are identical. A controlled inverting driver (INV) is added after the sensing stages of the two memories. The function of this driver is to achieve a complete logic, as the OR and AND logic operations supported by the memories are not universal. So, additional NOT operation is needed to allow the description of any Boolean function. The architecture of INV is illustrated in Fig. 7(d). A 1-bit barrel shift driver (BSD) is added to enable bit-level operations in addition to vector-level operations. The BSD is responsible for ensuring switchable connections between the two memory blocks. It can be reconfigured either to pass the data bits or to shift them on the fly with no need for an additional cycle. The architecture of the BSD is presented in Fig. 7(e). The proposed MOL-memory architecture presented in Fig. 8 is capable of performing numerous operations including logic computation and storage. Table I lists the most important (not all) operations that could be achieved. For each listed operation, a set of appropriate commands are simultaneously sent to the blocks constituting the architecture. A single operation requires one computational step. As an example, the case 19 in Table I corresponds to the arithmetic operation expressed in (3)

$$M_B(n) = M_B(n) \;\; AND \;\; \overline{M_A(m)} \tag{3}$$

where $M_A(m)$ and $M_B(n)$, are the bit-vectors located at the addresses m and n corresponding for MOL-memory-A and MOL-memory-B respectively. For this case, MOL-memory-A is set to the read mode. It reads the bit-vector $M_A(m)$, which undergoes a bitwise inversion through INV block. The 1-bit shifter is disabled. Simultaneously, MOL-memory-B, is set to the overwrite mode to perform MOL-AND with the vector $M_B(n)$. The result of the bitwise logic operation replaces the previous vector $M_B(n)$. The process is performed during one computational step.

B. Performing general arithmetic tasks

Generally, an arithmetic function (e.g. addition, subtraction, compare, etc.) could be expressed in Boolean form. Accordingly, breaking the Boolean form into several MOL operations allows its execution inside the proposed computational memory. Thus, the execution of an arbitrary Boolean function requires several computational steps so MOL operations are executed

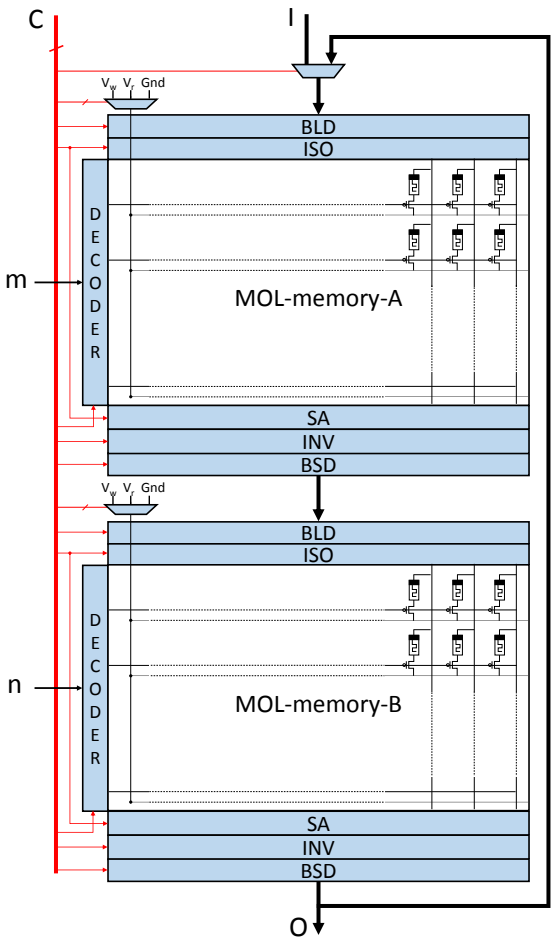


Fig. 8. Computational Memory Architecture

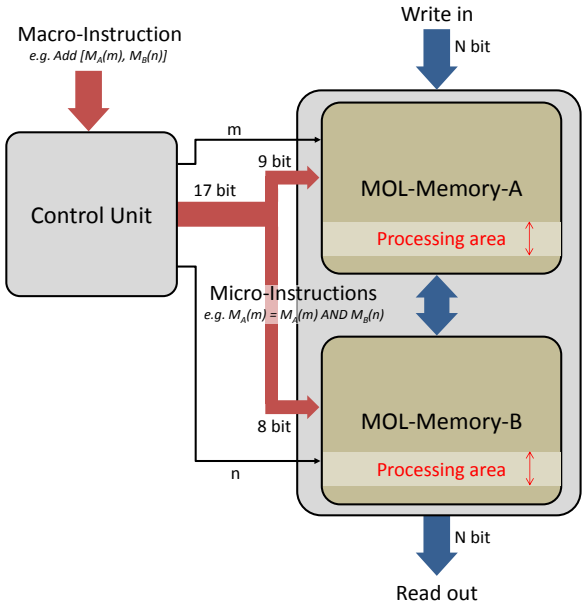


Fig. 9. Architecture diagram of MOL-based computational memory with its dedicated control unit

iteratively to finalize the desired arithmetic task. For this purpose, an external controller, which arranges these iterative operations is needed. Fig. 9 shows the block diagram which

TABLE I
ENCODING TABLE

		110 101 011 011 1 0				AND Read Write OR						AND Read Write OR			
		Enable Disable	Write Overwrite	Isolate Connect		Pass Invert		Select-out Select-in	Enable Disable	Write Overwrite	Isolate Connect		Pass Invert		Select-out Select-in
Operation	**Binary**	**Micro-Instruction**	$\mathbf{E_A}$	$\mathbf{BLD_A}$	$\mathbf{ISO_A}$	$\mathbf{Mode_A}$	$\mathbf{INV_A}$	$\mathbf{BSD_A}$	$\mathbf{E_B}$	$\mathbf{BLD_B}$	$\mathbf{ISO_B}$	$\mathbf{Mode_B}$	$\mathbf{INV_B}$	$\mathbf{BSD_B}$	**Sel**
0	00000	$M_A(m) = I$	1	1	1	0 1 1	1	1	0	x	0	1 1 1	x	1	0
1	00001	$M_B(n) = I$	0	0	1	0 1 1	0	1	1	1	1	0 1 1	1	1	0
2	00010	$O = M_A(m)$	1	x	0	1 0 1	1	1	0	0	1	1 1 1	1	1	x
3	00011	$O = M_B(m)$	0	x	0	1 1 1	x	1	1	x	0	1 0 1	0	1	x
4	00100	$O = \overline{M_A(m)}$	1	x	0	1 0 1	1	1	0	0	1	1 1 1	0	1	x
5	00101	$O = \overline{M_B(m)}$	0	x	0	1 1 1	x	1	1	x	0	1 0 1	1	1	x
6	00110	$M_A(m) = M_B(n)$	1	1	1	0 1 1	x	1	1	x	0	1 0 1	0	1	1
7	00111	$M_B(n) = \overline{M_A(m)}$	1	x	0	1 0 1	1	1	1	1	1	0 1 1	x	1	x
8	01000	$M_A(m) = \overline{M_B(n)}$	1	1	1	0 1 1	x	1	1	x	0	1 0 1	1	1	1
9	01001	$M_B(n) = \overline{M_A(m)}$	1	x	0	1 0 1	0	1	1	1	1	0 1 1	x	1	x
10	01010	$M_A(m) = M_A(m)\ AND\ I$	1	0	1	1 1 0	1	1	0	x	0	1 1 1	x	1	0
11	01011	$M_B(n) = M_B(n)\ AND\ I$	0	0	1	0 1 1	0	1	1	1	1	1 1 0	1	1	0
12	01100	$M_A(m) = M_B(n)\ OR\ I$	1	0	1	0 1 1	1	1	0	x	0	1 1 1	x	1	0
13	01101	$M_B(n) = M_B(m)\ OR\ I$	0	0	1	0 1 1	0	1	1	1	1	0 1 1	1	1	0
14	01110	$M_A(m) = M_A(m)\ AND\ M_B(n)$	1	0	1	1 1 0	x	1	1	x	0	1 0 1	0	1	1
15	01111	$M_B(n) = M_B(n)\ AND\ M_A(n)$	1	x	0	1 0 1	1	1	1	0	1	1 1 0	x	1	x
16	10000	$M_A(m) = M_A(n)\ OR\ M_B(n)$	1	0	1	0 1 1	x	1	1	x	0	1 0 1	0	1	1
17	10001	$M_B(n) = M_B(m)\ OR\ M_A(n)$	1	x	0	1 0 1	1	1	1	0	1	0 1 1	x	1	x
18	10010	$M_A(m) = M_A(m)\ AND\ \overline{M_B(n)}$	1	0	1	1 1 0	x	1	1	x	0	1 0 1	1	1	1
19	10011	$M_B(n) = M_B(n)\ AND\ \overline{M_A(n)}$	1	x	0	1 0 1	0	1	1	0	1	1 1 0	x	1	x
20	10100	$M_A(m) = M_A(m)\ OR\ \overline{M_B(n)}$	1	0	1	0 1 1	x	1	1	x	0	1 0 1	1	1	1
21	10101	$M_B(n) = M_B(m)\ OR\ \overline{M_A(n)}$	1	x	0	1 0 1	0	1	1	0	1	0 1 1	x	1	x
22	10110	$M_A(m) = M_B(n) << 1$	1	1	1	0 1 1	x	1	1	x	0	1 0 1	1	0	1
23	10111	$M_B(n) = M_A(m) << 1$	1	x	0	1 0 1	1	0	1	1	1	0 1 1	x	1	x
24	11000	$M_A(m) = M_A(m)\ AND\ [M_B(n) << 1]$	1	0	1	1 1 0	x	1	1	x	0	1 0 1	0	0	1
25	11001	$M_B(n) = M_B(n)\ AND\ [M_A(n) << 1]$	1	x	0	1 0 1	1	0	1	0	1	1 1 0	x	1	x
26	11010	$M_A(m) = M_A(n)\ OR\ [M_B(n) << 1]$	1	0	1	0 1 1	x	1	1	x	0	1 0 1	0	0	1
27	11011	$M_B(n) = M_B(m)\ OR\ [M_A(n) << 1]$	1	x	0	1 0 1	1	0	1	0	1	0 1 1	x	1	x
28	11100	$M_A(m) = \overline{M_B(n) << 1}$	1	1	1	0 1 1	x	1	1	x	0	1 0 1	1	0	1
29	11101	$M_B(n) = \overline{M_A(m) << 1}$	1	x	0	1 0 1	0	0	1	1	1	0 1 1	x	1	x

illustrates the general structure of the memory and the controller. When the controller receives an instruction from the processor, it decides the role of the memory whether for storage or computation. Specifically, for the case of computation, the controller breaks the received macro-instruction into several iterative micro-instructions, which can be performed by the proposed memory. In our case, micro-instructions correspond to the set of operations listed in Table I. A processing area should be reserved in each of MOL-memory-A and MOL-memory-B in the proposed computational memory. The area could be dynamically changed according to the need (such as the number of required tasks). Moreover, the location of the processing area could be also changed periodically. The reason for location change is to attain better endurance for the memristive memory cells that are subjected to continuous stress. The design of the controller is beyond the scope of this paper.

VI. MOL BASED IN MEMORY N-BIT FULL ADDITION

In this section, an N-bit full addition is considered as a case study to evaluate the functionality of our proposed computational memory architecture.

A. Proposed iterative N-bit full addition process dedicated for computational MOL-memory

Generally, full adder is the basic digital building block for several computational operations (i.e. addition, subtraction and multiplication). Thus, implementing a full addition process inside the memory is the first step toward in-memory computing. Equations (4) and (5) present the well known expressions of the 1-bit full addition.

$$S = A \oplus B \oplus C_{in} \tag{4}$$

$$C_{out} = AB + BC_{in} + AC_{in} \tag{5}$$

where A and B are the inputs, C_{in} is the input carry value, S is the 1-bit adder output and C_{out} is the output carry. The operator $\oplus$ corresponds to the boolean XOR. Assume that all the inputs are initially stored in the memory. The boolean functions of S and C_{out} are written in the form of sum of products (SoP), so that their expressions could be mapped into the proposed computational memory using sequential MOL operations. The inputs of a given MOL operation should be aligned on the same columns (ie. same bit-lines) in the memory, otherwise, a pre-shifting process is required to align the corresponding inputs. Accordingly, the number of steps required to achieve the computation of S and C_{out} is affected by the relative positions of the input A, B and C_{in} inside the memory. In order to minimize the number of computational steps as well as reserve the minimum possible processing area, a dedicated N-bit addition process is proposed. The process uses a specific sequence of each operation listed in Table I. Consider the two N-bit vectors A^N and B^N. The addition of A^N and B^N leads to the vector sum S^{N+1}. Normally, the additional 1-bit in S^{N+1} is reserved for the expected overflow in the addition process. We propose to follow the procedure illustrated in Algorithm 1 to achieve a vector level addition of A^N and B^N:

- Stage 1: The vector sum S_0 which is of length $N+1$ is initialized by the bitwise XOR of A^N and B^N. Similarly, the vector carry C_0 of length $N+1$ is initialized by the bitwise AND of A^N and B^N. The expressions of S_0 and C_0 are presented in (6) and (7) respectively.

$$S_0 = A \oplus B \tag{6}$$

$$C_0 = AB \tag{7}$$

- Stage 2: Each time, a new vector sum S_{i+1} and vector carry C_{i+1} are created based on their previous values S_i and C_i respectively. Equation (8) and (9) demonstrate the respective

expressions of S_{i+1} and C_{i+1}. This process is repeated $N-1$ times.

$$S_{i+1} = S_i \oplus (C_i << 1) \tag{8}$$

$$C_{i+1} = S_i\,(C_i << 1) \tag{9}$$

The operator "<< 1" stands for the 1-bit shift to the left. At the end of this iterative process, the final obtained vector S_{N-1} corresponds to the sum of A^N and B^N while C_{N-1} will be a zero vector.

Algorithm 1 N-bit addition dedicated for computation inside MOL-memory

```
procedure ADD(A, B)                          ▷ A and B are N-bit vectors
    S_0 ← A ⊕ B
    C_0 ← AB
    for i ← 0 to N − 2 do
        S_{i+1} ← S_i ⊕ (C_i << 1)
        C_{i+1} ← S_i(C_i << 1)
    end for
    return S_{N−1}                           ▷ The sum of A and B
end procedure
```

B. In-memory N-bit full addition procedure

The proposed iterative N-bit addition process can be mapped into the computational MOL-memory using the operations listed in Table I. Fig. 10 shows a space-time representation of the N-bit full addition process, which is realized within MOL-memory-A and MOL-memory-B. For each computational step, the new contents of the memories are listed in a new single column in Fig. 10. Assume the case where the two vectors A^N and B^N, that are subjected to addition, are initially stored inside MOL-memory-A at the addresses m_1 and m_2 respectively. Additional two word-lines have to be reserved inside MOL-memory-B to attain the addition of A^N and B^N. The two stages that are presented in section VI-A are realized as follows:

- Stage 1: Corresponds to the steps between 0 and 5 using the six micro-operations that have the sequence order shown in Fig. 10. At the end of this stage, the bitwise AND of A and B (i.e. $C_0 = AB$) is stored in MOL-memory-A while the XNOR of A and B (i.e. $\overline{S_0} = \overline{A \oplus B}$) is stored in MOL-memory-B.
- Stage 2: In this stage, the steps between 6 and 11 are repeated N-1 times. Their corresponding micro-instructions have the sequence order shown in Fig. 10. Each time the initial vector C_i is shifted to the left by one bit and the resulting vector undergoes bitwise AND with the initial vector S_i. The obtained result is referred as C_{i+1}, which expression is presented in (9). Simultaneously, the shifted version of C_i undergoes bitwise XNOR with the initial vector S_i to obtain the new vector sum S_{i+1}. At the end of this process, the vector $\overline{S_{N-1}}$ is stored in MOL-memory-B. Thus, an additional step is required to make a bitwise inversion of the obtained vector. The resulting vector S_{N-1}, which represents the N-bit addition of the vectors A and B, is stored in MOL-memory-A.

C. Space-time analysis of the N-bit addition process

The total number of computational steps required to complete the N-bit addition is $6N+1$ steps as shown in Fig. 10. The total number of memristors reserved for the execution of the N-bit addition is $4N$ memristors corresponding to four rows of the MOL-memory architecture. These rows include the initial locations of A and B, although the initial bits of the vectors A and B are lost. However, in some cases, the destruction of the input vectors is undesired, especially when these inputs are required for another computational tasks. In order to avoid this case, pre-copy operations of the two input vectors A and B could be performed to reserve safe versions of these vectors. Thus, two additional computational steps are required for this case and the new total number of computational steps becomes $6N + 3$. The considered operation sequence in Fig. 10 corresponds to the case where A and B are both located in MOL-memory-A. However, another two cases should be considered also: (i) If A and B belong to different MOL-memories, one additional pre-copy operation could be performed to drag the input vector contained in MOL-memory-B to MOL-memory-A. (ii) If A and B are both contained in MOL-memory-B, two additional pre-copy operations are needed to drag them to MOL-memory-A. These pre-copy operations are performed to maintain the same operation sequence, which is presented in Fig. 10. Pre-copy operations can be avoided with different sequences (one for each case, with common parts).

VII. Simulation and performance analysis

In this section, we study the performance of the proposed computational memory architecture which is implemented using a realistic model of Magnetic Tunnel Junction (MTJ) device and a CMOS 65nm technology node. The study includes timing analysis, energy consumption and robustness against device variability.

A. Adopted memristive device

Several memristive devices have been explored in the literature. In fact, MOL technique could apply to all types of bipolar memristive devices holding two resistance states R_{ON} and R_{OFF}. Among these devices, memristors such as HfO_x [38] and TiO_2 [32] exhibit promising characteristics with their high switching speed (sub-ns) and their high R_{OFF}/R_{ON} ratio (> 100). However, current memristor technologies suffer from endurance limitations. Although several efforts have been carried out to enhance endurance [38], the allowed number of switchings per memristor is still limited in the range of 10^6 to 10^{12} for the best case. This value is relatively low for targeting intensive computations inside memristive crossbars. The Spin Transfer Torque Magnetic Memory (STT-MRAM) [39], which have been redescribed in terms of memristive systems [40], is considered as one of the most promising nonvolatile memories (NVM). STT-MRAM is eligible for high reliability applications [41] due to its high endurance ($> 10^{15}$) [32]. As illustrated in Fig. 11, an MTJ cell is mainly composed of two ferromagnetic layers sandwiching an ultra-thin tunnel barrier. The resistance of the MTJ cell depends on the relative orientation of magnetization in the free and reference layers. The low resistance state

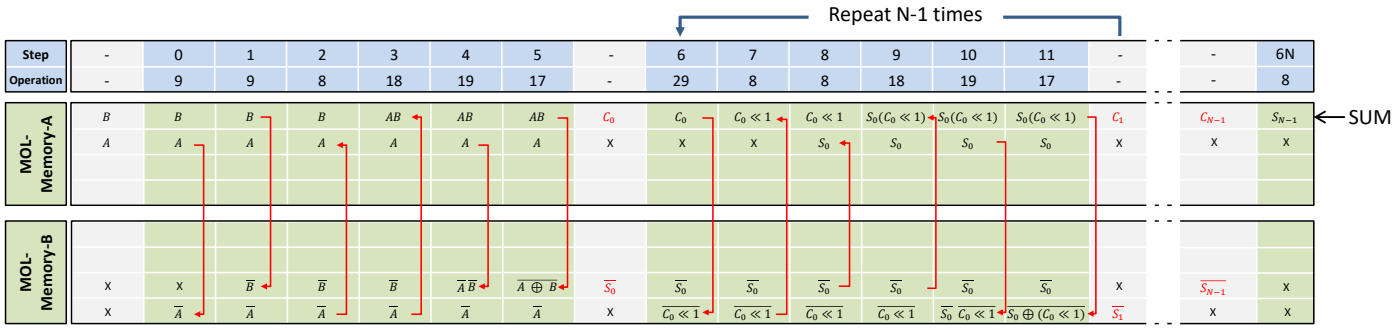


Fig. 10. Operations sequence for an in-memory N-bit addition process using MOL-memory

(logic '0') of the MTJ corresponds to the parallel configuration (P) with resistance R_P, while its high resistance state (logic '1') is reached in the case of anti-parallel configuration (AP) with resistance R_{AP}. The magnitude of the applied current I must exceed a critical value noted as I_{C0} to allow switching. In

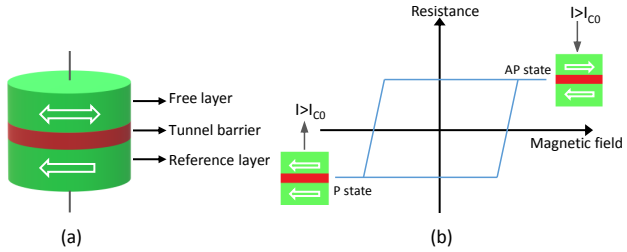


Fig. 11. Typical MTJ: (a) Core structure, (b) Resistance variation

contrast to memristors, MTJs are characterized by a relatively low margin between R_P and R_{AP}. The corresponding margin is commonly evaluated as the Tunnel Magnetoresistance (TMR) ratio, whose expression is presented in (10):

$$TMR = \frac{\Delta R}{R_P} = \frac{R_{AP} - R_P}{R_P} \tag{10}$$

However, such a low margin has no effect on switching an MTJ cell but on the corresponding sensing mechanism of the state of this cell. This requires a more complicated sensing driver to estimate and decide the corresponding state of a given selected row inside the memory. In this work, we have used MTJs with perpendicular magnetic anisotropy (PMA). The adopted PMA MTJ is formed of CoFeB/MgO/CoFeB layers. The physical model describing the static, dynamic and stochastic behaviors of the STT-PMA-MTJ is presented in [42] and [43]. In order to fit with experimental results in the literature, the technology parameters corresponding to the material composition are kept at their default values. Other parameters which depend on the designers' choice are presented in Table II with their corresponding values. It is worth highlighting that the low TMR value of the adopted STT-PMA-MTJ device usually lead to a high complexity of SAs when the memory data needs to be read out. However, in our case, the proposed simple design comprised of three cascaded inverters and a reference resistor was verified to be sufficient when combined with the designed INV and BSD blocks (Fig. 7). In fact, these two CMOS based blocks are leveraged in order to perfectly regenerate the voltage levels corresponding to the amplified output of the cascaded inverters. Fig. 12 shows the switching behavior of an MTJ device when it is fed with a square signal of amplitude $1.2V$. τ_{AP-P} and τ_{P-AP} correspond to the switching delays from AP to P state and the reverse case respectively. In fact, switching delay varies according to the applied voltage level. Fig. 13 illustrates the variation of τ_{AP-P} and τ_{P-AP} with respect to the applied voltage level. The graph indicates that switching delay decreases with the increase of the voltage while switching from P to AP state is faster than the reverse operation (i.e. $\tau_{AP-P} < \tau_{P-AP}$).

The choice of the memristive device type is not constrained by a specified MOL requirement. It can be observed from the mechanism of MOL technique that it involves direct access to the terminals of memristive devices which highly resembles conventional write operation. During the operation of MOL, the potential difference between the terminals of the memristive device always attains a binary level. Accordingly, MOL can be implemented in a wide range of memristive memories without specifying particular device features. In contrast, the structure of pre-existing logic design styles either establishes a series connection of a resistor (e.g. IMPLY) or series connection of the memristive devices (e.g. MAGIC) for normal operation. This undoubtedly prevents direct access to the memristive device terminals and consequently imposes specific device constraints, such as the requirement of sufficient HRS/LRS ratio and/or operated with thresholds type devices only.

TABLE II
ADOPTED VARIABLES AND PARAMETERS FOR PMA MTJ DEVICE

Parameter	Value	Description
t_{ox}	$0.85\,nm$	Thickness of oxide barrier
$TMR(0)$	70%	TMR ratio with 0 stress voltage
$Area$	$\pi \times 20\,nm \times 20\,nm$	MTJ surface
t_{sl}	$1.3\,nm$	Thickness of free layer

B. Performance analysis

Transient simulation has been conducted for the proposed design of the MOL-memory architecture. Based on the adopted STT-PMA-MTJ device and the CMOS $65nm$ process, simulations have been carried out using Cadence Virtuoso toolset. In order to evaluate the performance of the architecture, the N-bit addition process described in Section VI is performed. The size of the crossbar is chosen to be 8×8 for MOL-memory-A as well as MOL-memory-B. The size N is chosen to be 8 bits for

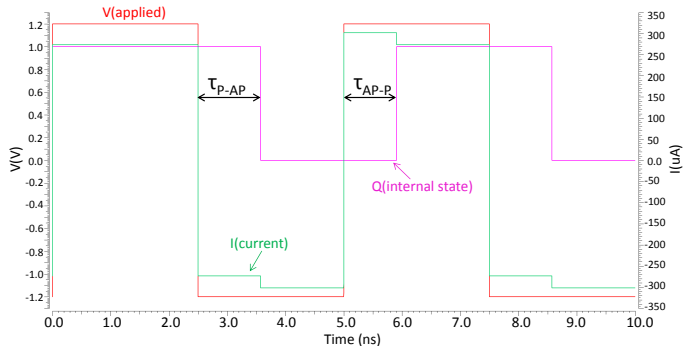


Fig. 12. Switching behavior of MTJ device when fed with square signal

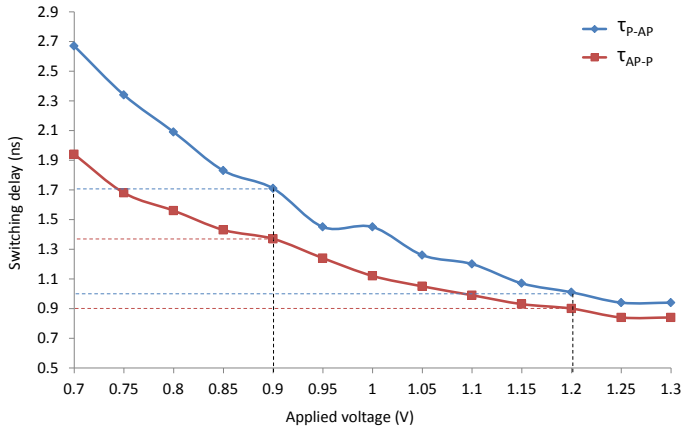


Fig. 13. Switching delay of an MTJ cell as function of applied voltage level

both numbers A and B. The corresponding operating voltage is set to $1.2V$ for logic '1' and $-1.2V$ for logic '0'. Based on the obtained transient results, total latency is evaluated as well as the total energy consumption. As an example, Fig. 14 presents the corresponding internal states of the 4 word-lines that are reserved for the 8-bit addition process, which is performed on the two arbitrary vectors A=[01011011] and B=[00111111]. The control signals of the MOL-memory architecture follow the operation sequence presented in of Fig. 10.

1) Timing analysis: The first two steps correspond to the initialization of vectors A and B inside MOL-memory-A. The corresponding sum S=[10011010] is evaluated after $6N + 1$ computational steps which is equal to 49 for $N = 8$. In fact, the max delay is noticed to be $\tau_{Max} = 1.7\,ns$ which is greater than the max switching delay of MTJ devices operating at $1.2V$. This is due to the voltage drop noticed along CMOS drivers. The actual voltage supplied to MTJ devices is $0.9V$ (could be interpreted from Fig. 13). This significant voltage drop (25%) is due to the adoption of low values of R_P and R_{AP}. Moreover, the width W of MOSFETs has a direct effect on the voltage drop percentage. This voltage drop could be mitigated by increasing W, but this induces overheads on the total area of CMOS drivers.

Therefore, the duration (T) of each computational step must be greater than τ_{Max}. The variability in τ_{Max} due to the stochastic switching behavior of MTJs should also be considered. Thus, an additional guard interval (τ_g) is introduced to guarantee the switching of the MTJs. The resulting step duration for the proposed MOL-memory architecture is $T = \tau_{Max}+\tau_g = 1.7+\tau_g$. We set τ_g at $100\,ps$ which corresponds to 6% of τ_{Max}, so the duration T is equal to $1.8\,ns$. The minimum time required for finalizing the addition operation (neglecting the 2 initialization steps) is evaluated as $49 \times 1.8\,ns = 88.2\,ns$.

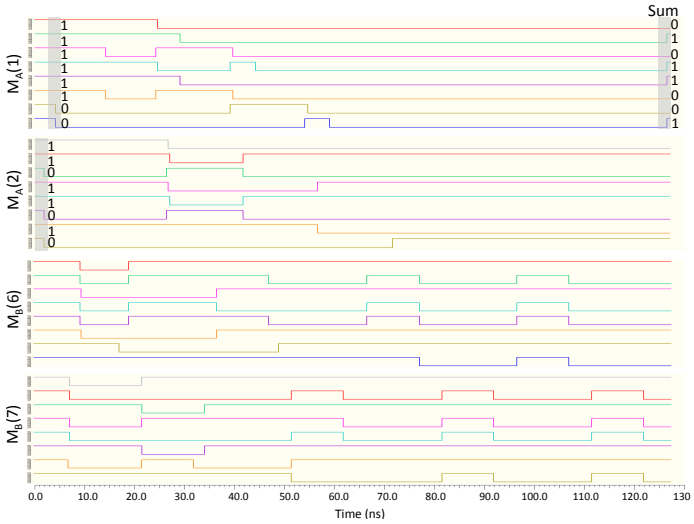


Fig. 14. Transient simulation for the in-memory 8-bit addition process

2) Robustness against resistance variability: Due to the limit of the manufacturing technology, the actual thickness of oxide layer and free layer of MTJ devices cannot be fixed at a constant value. They typically vary in a small range, but can lead to a relatively important variation in the values of LRS and HRS of MTJ. Therefore, we have examined the effect of MTJ resistance variability on the performance of our proposed MOL-based computational memory architecture. Simulations are conducted by performing the 8-bit addition. The adopted MTJ parameters TMR, t_{sl} and t_{ox} are kept as presented in Table II while subjecting them to a random process. The parameters are chosen to follow either uniform or Gaussian distribution. In Gaussian distribution, no error has been detected even when reaching a variation percentage of 21% for TMR, t_{sl} and t_{ox}. As for uniform distribution, the tolerated variation reaches 7%. This demonstrates the robustness of the proposed design against the resistance variability of MTJ devices.

3) Energy estimation: Energy consumption differs according to the operation: read, write or performing computation. In this section we will focus on the energy consumed by the memristive crossbar of the MOL-memory architecture neglecting the consumed energy by the peripheral drivers.

(i) Write-energy: Consider a single MTJ device located inside MOL-memory architecture. The energy consumed when a single bit is written into this MTJ device mainly depends on its previous resistance state (R_P or R_{AP}) and its final one.

Hence, the 4 cases for write-energy are considered in (11).

$$\begin{aligned} E_{w_{0/0}} &= \frac{{V_w}^2}{R_{AP}^{'}}T \\ E_{w_{0/1}} &= \frac{{V_w}^2}{R_{AP}^{'}}\tau_{_{AP-P}} + \frac{{V_w}^2}{R_{P}^{'}}(T-\tau_{_{AP-P}}) \\ E_{w_{1/0}} &= \frac{{V_w}^2}{R_{P}^{'}}\tau_{_{P-AP}} + \frac{{V_w}^2}{R_{AP}^{'}}(T-\tau_{_{P-AP}}) \\ E_{w_{1/1}} &= \frac{{V_w}^2}{R_{P}^{'}}T \end{aligned} \tag{11}$$

where $E_{w_{i/j}}$ corresponds to the write-energy needed to put the MTJ device in state $i \in \{0,1\}$, after it was in the previous state $j \in \{0,1\}$; V_w and T are the write voltage and write duration respectively; $R_{AP}^{'}$ and $R_{P}^{'}$ represent the resistance states of 1T1M cell. $R_{AP}^{'} = R_{AP} + R_{MOS}$ and $R_{P}^{'} = R_{P} + R_{MOS}$. Generally, the values of i and j are not deterministic, but the 4 cases presented in (11) are considered as equiprobable, since there is no pre-knowledge about the data bits inside the memory as well as the bits that would be written. Thus, the average write-energy is estimated as the average sum of the 4 write-energy cases as presented in (12).

$$E_w = \frac{1}{4}\sum_{i,j} E_{w_{i/j}} = \frac{{V_w}^2}{R_{AP}^{'}}\left(\frac{T}{2} - \frac{\Delta\tau}{4}\right) + \frac{{V_w}^2}{R_{P}^{'}}\left(\frac{T}{2} + \frac{\Delta\tau}{4}\right) \tag{12}$$

where $\Delta\tau = \tau_{_{P-AP}} - \tau_{_{AP-P}}$. Assuming that the term $\frac{\Delta\tau}{4}$ is almost negligible compared to $\frac{T}{2}$, the overall expression in (12) is simplified in (13).

$$E_w \approx \frac{{V_w}^2}{2R_w}T \quad \text{with} \quad R_w = \frac{R_{P}^{'}R_{AP}^{'}}{R_{P}^{'} + R_{AP}^{'}} \tag{13}$$

R_w represents the equivalent resistance of two MTJs having opposite states and connected in parallel.

(ii) Read-energy: Reading a single MTJ device requires a sensing voltage V_r and a reference resistor R_{Ref} connected in series with a MOSFET. The total resistance of this 1T1R cell is $R_{Ref}^{'}$. The corresponding state of the sensed MTJ device is assumed to be stable. The two possible cases for read-energy are presented in (14).

$$\begin{aligned} E_{r_0} &= \frac{{V_r}^2}{R_{AP}^{'} + R_{Ref}^{'}}T \\ E_{r_1} &= \frac{{V_r}^2}{R_{P}^{'} + R_{Ref}^{'}}T \end{aligned} \tag{14}$$

where E_{r_0} and E_{r_1} represents the required energy consumption for sensing AP and P states respectively during a period T. The corresponding average read-energy is expressed in (15).

$$E_r = \frac{{V_r}^2}{2R_r}T \quad \text{with} \quad R_r = \frac{(R_{P}^{'} + R_{Ref}^{'})(R_{AP}^{'} + R_{Ref}^{'})}{(R_{P}^{'} + R_{Ref}^{'}) + (R_{AP}^{'} + R_{Ref}^{'})} \tag{15}$$

(iii) Computation-energy: Computational operations that are performed inside MOL-memory architecture are classified into MOL or copy operations. Table III summarizes the energy consumed by each type of operation. Using the specifications of the adopted MTJ which are listed in Table IV, the average energy consumed by a MOL operation could be expressed as $E_{MOL} = E_w/2 + E_r = 0.196\,pj$ whereas that consumed by a copy operation is calculated as $E_{COPY} = E_w + E_r = 0.333\,pj$.

Normally, computation inside MOL-memory architecture is performed on N bits simultaneously. For the N-bit addition process which is performed within $6N+1$ cycles, $3N$ cycles corresponds to MOL operations while $3N+1$ cycles corresponds to copy operations. Thus, the overall consumed energy (E_T) could be expressed as in (16).

$$E_T = (3N)(N\,E_{MOL}) + (3N+1)(N\,E_{COPY}) \tag{16}$$

By substituting the corresponding values of E_{MOL} and E_{COPY} presented in Table III, the expression of the total energy becomes $E_T = 1.587N^2 + 0.333N$. Specifically, for the 8-bit addition process, E_T is equal to $104.2\,pJ$. The value of the energy consumption extracted by simulation is $124.43\,pJ$.

TABLE III
ENERGY CONSUMED BY A COMPUTATIONAL OPERATION

In1	In2	MOL-AND	MOL-OR	Copy
0	0	$E_{w_{0/0}} + E_{r_0}$	E_{r_0}	$E_{w_{0/0}} + E_{r_0}$
0	1	E_{r_0}	$E_{w_{1/0}} + E_{r_0}$	$E_{w_{1/0}} + E_{r_0}$
1	0	$E_{w_{0/1}} + E_{r_1}$	E_{r_1}	$E_{w_{0/1}} + E_{r_1}$
1	1	E_{r_1}	$E_{w_{1/1}} + E_{r_1}$	$E_{w_{1/1}} + E_{r_1}$

TABLE IV
SPECIFICATIONS

Specification	Value
R_{AP}	$6\,K$
R_P	$3.97\,K$
R_{MOS}	$0.5\,K$
R_{Ref}	$4.8\,K$
V_w	$0.588\,V$
V_r	$0.9\,V$
$\tau_{_{AP-P}}$	$1.4\,ns$
$\tau_{_{P-AP}}$	$1.7\,ns$
T	$1.8\,ns$

VIII. COMPARISON

In this section, the proposed MOL-memory architecture has been compared with recently published relevant designs (listed in Table V) targeting in-memory computing. The comparison has been carried out based on the performance of N-bit addition in terms of latency, energy consumption and utilized area. Note that the considered area incorporates only the memristors involved in the computation regardless of the size of the crossbar.

1) MOL vs IMPLY and MAGIC:

- Except for the parallel approach in [10], our proposed design, which uses only $6N+1$ steps to perform addition, outperforms all IMPLY and MAGIC based designs listed in Table V in terms of number of computational steps. In fact, [10] uses the parallel approach which is intended to increase the level of parallelism in computation. However this approach requires significant modifications in the crossbar structure by adding connections between its rows. This leads to an increased area compared to the conventional crossbar structure.
- The step delay in our proposed design is $1.8\,ns$. Although the designs presented in [12], [30] and [13] adopt memristive

TABLE V
COMPARISON OF DIFFERENT LOGIC FAMILIES FOR N-BIT ADDITION IN TERMS OF AREA, LATENCY AND ENERGY CONSUMPTION

Reference	Method	# Steps	Step delay	Latency (ns)	Area (# memristive cells)	Energy (pJ)
(This work)	MOL	$6N+1$	$1.8ns$	$10.8N+1.8$	$4N$	$1.587N^2+0.333N$
[10]	IMPLY Serial	$29N$	-	-	2	$\sim 9.5N$
[10]	IMPLY Parallel	$5N+18$	-	-	$6N-1$	$\sim 9.5N$
[11]	IMPLY	$89N$	-	-	4	-
[12]	MAGIC Area optimized	$15N$	$1.3ns$	$19.5N$	5	$\sim 3.365N$
[12]	MAGIC Latency optimized	$12N+1$	$1.3ns$	$15.6N+1.3$	$11N-1$	$\sim 3.365N$
[12]	MAGIC Transpose I	$15N+1$	$1.3ns$	$19.5N+1.3$	$22N-3$	$\sim 6.53N$
[12]	MAGIC Transpose II	$10N+3$	$1.3ns$	$13N+3.9$	$13N-3$	$\sim 4.72N$
[13]	MAGIC	$12N+1$	$1.12ns$	$13.44N+1.12$	$14N+1$	$0.684N$
[30]	MAGIC (Naive mapping)	$12N$	$1.43ns$	$17.6N$	$15N$	$0.684N$
[30]	MAGIC (Compact mapping)	$16N$	$1.43ns$	$22.8N$	$24N$	$0.894N$
[14]	MAGIC	$20N+15$	$1.89ns$	$37.8N+28.35$	12	$0.3N$
[15]	MAJ (Naive)	$\sim 22N$	-	-	$\sim 4N$	-
[15]	MAJ (MIG rewriting)	$\sim 16N$	-	-	$\sim 3N$	-
[15]	MAJ (Rewriting and compilation)	$\sim 15N$	-	-	$\sim 2N$	-
[16]	CRS (PC-Adder)	$2N+4$	-	-	$2N+1$	-
[16]	CRS (TC-Adder)	$4N+5$	-	-	$N+2$	-

devices that provide better step delay ($1.12\,ns$ to $1.43\,ns$), the total latency in our proposed design is still the minimum ($10.8N+1.8\,ns$). The best case achieved with the competitor designs is recorded in [12] with $13N+3.9\,ns$ (i.e. $\sim$20% more latency).

- In the proposed design, $4N$ memristors participate in the execution of the N-bit addition. This number ranges from $11N-1$ to $24N$ for the majority of the designs based on MAGIC, so our proposed design exhibits $\times 1.75$ to $\times 5$ area reduction. On the other hand, the IMPLY based serial approach [10], MAGIC based area optimized design [12] and the design presented in [11] use a fixed number of memristors to perform addition operation. In other words, the required number of memristors is independent of the size N of the addition operation. This area optimization comes at the cost of high number of computational steps ($\times 2.5$ to $\times 18.8$).
- The average energy consumed in pJ for the memristive crossbar in our design is $1.5867N^2+0.333N$. This quadratic expression indicates a significant energy consumption in the order of $\times N$ as compared to the linear energy expressions for the other designs listed in the table. The reason for this energy gap is that for each step the same bitwise operation is performed on the whole word-line (size N). However, the other approaches from the literature perform 1 bit operation in each step. Although our methodology induces overheads on the total energy consumption, working on the vector level rather than bit level greatly simplifies the corresponding control unit and reduces its complexity.

2) MOL vs MAJ and CRS:

- Logic representation using MIGs has experimentally shown promising results in logic optimization [44]. Memristive devices can efficiently execute the intrinsic resistive MAJ operation. The authors of [23][15] present a programmable in-memory computing system namely Programmable Logic-in-Memory (PLiM). The instruction set for the PLiM architecture is based on the MAJ operation. As investigated in [15], the number of required memristors for the addition is $\sim 2N$, which is equal to 50% of that in our approach. However, the execution of an N-bit addition inside PLiM requires $15N$ cycles for the best case, which is $\times 2.5$ the number of cycles required in our proposed design. This high number of computational steps is related to the repeated read out operations of intermediate results, which impacts in addition the step delay and energy consumption (not evaluated in [15]).
- The number of computational steps achieved in [16], which uses the CRS approach, is less than that of our proposed computational memory. However, other parameters such as the step delay which is not investigated by the authors is expected to be greater. This is due to the fact that the presented architecture, based on two separated memory blocks, uses an intermediate control unit which reads data bits from one memory block and redistribute them along BLs and WLs of the other memory block. This process increases significantly the overall critical path and consequently the step delay. The number of memristive cells required in [16] is also less than that in our proposed design. However, it is clear that based on this approach, the reserved area corresponds to a fixed location inside the memory, as the input bits cannot be shared to all WLs especially for large memory sizes. This affects the endurance of memristive cells participating in the computation which are subjected to continuous stress.

As explained in Section V-B, the proposed memristive computational memory is able to perform any general arithmetic function by breaking it into a netlist of iterative MOL operations. As MOL is based on the primitive AND/OR operations, the ABC tool [45], which has been employed for existing logic design styles [30][31], could be also leveraged in order to realize the synthesis task. This will be considered in our future work.

IX. CONCLUSION

In this paper, the MOL design style is introduced together with an original architecture for MOL-based computational memory. This novel logic design style is inspired from a digital representation of memristors. Unlike existing approaches, MOL can operate with different memristor technologies, regardless of LRS-HRS margin and with linear as well as threshold-type memristive devices. Furthermore, the proposed original computational memory architecture, with appropriate drivers and control sequences, allows the execution of numerous logic operations, at bit or vector-level, in one or two computational steps at most. In order to illustrate the benefits of the proposed

approach and to evaluate its performances, the implementation of an N-bit full addition using the proposed MOL-based computational memory has been detailed. The design is simulated in Cadence Virtuoso environment using CMOS 65nm process and realistic model parameters for STT-PMA-MTJ device. Results comparison with existing recent approaches demonstrates significant reductions in terms of latency and area.

REFERENCES

[1] L. Chua, "Memristor-the missing circuit element," *IEEE Trans. on circuit theory*, vol. 18, no. 5, pp. 507–519, 1971.

[2] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, p. 80, 2008.

[3] S. Kvatinsky, N. Wald, G. Satat, A. Kolodny, U. C. Weiser, and E. G. Friedman, "MRL—memristor ratioed logic," in *proc. of the Int. Workshop on Cellular Nanoscale Networks and their Applications*, 2012.

[4] J. Chowdhury, K. Das, and K. Rout, "Implementation of 24T memristor based adder architecture with improved performance," *Int. Journal of Electrical, Electronics and Data Communication*, vol. 3, no. 6, 2015.

[5] K. A. Ali, M. Rizk, A. Baghdadi, J.-P. Diguet, and J. Jomaah, "MRL crossbar-based full adder design," in *proc. of the IEEE Int. Conf. on Electronics, Circuits and Systems (ICECS)*, 2019.

[6] A. Sebastian, M. Le Gallo, R. Khaddam-Aljameh, and E. Eleftheriou, "Memory devices and applications for in-memory computing," *Nature Nanotechnology*, pp. 1–16, 2020.

[7] D. Patterson *et al.*, "A case for intelligent RAM," *IEEE Micro*, vol. 17, no. 2, pp. 34–44, 1997.

[8] S. Khoram, Y. Zha, J. Zhang, and J. Li, "Challenges and opportunities: from near-memory computing to in-memory computing," in *Proc. of the ACM on International Symposium on Physical Design*, 2017, pp. 43–46.

[9] S. Hamdioui *et al.*, "Applications of Computation-In-Memory Architectures based on Memristive Devices," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 486–491.

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

[11] E. Lehtonen and M. Laiho, "Stateful implication logic with memristors," in *proc. of the IEEE Int. Symp. on Nanoscale Architectures*, 2009.

[12] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using Memristor-Aided logic (MAGIC)," *IEEE Trans. on Nanotechnology*, vol. 15, no. 4, pp. 635–650, 2016.

[13] P. Thangkhiew, R. Gharpinde, D. N. Yadav, K. Datta, and I. Sengupta, "Efficient implementation of adder circuits in memristive crossbar array," in *proc. of the IEEE Region 10 Conf. (TENCON)*, 2017.

[14] P. L. Thangkhiew, R. Gharpinde, P. V. Chowdhary, K. Datta, and I. Sengupta, "Area efficient implementation of ripple carry adder using memristor crossbar arrays," in *proc. of the Int. Design & Test Symp. (IDT)*, 2016.

[15] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, and R. Drechsler, *Logic Synthesis for Majority Based In-Memory Computing, Chapter in Advances in memristors, memristive devices and systems*. Springer, 2017.

[16] A. Siemon, S. Menzel, R. Waser, and E. Linn, "A complementary resistive switch-based crossbar array adder," *IEEE journal on emerging and selected topics in circuits and systems*, vol. 5, no. 1, pp. 64–74, 2015.

[17] C.-X. Xue *et al.*, "A 1Mb multibit ReRAM computing-in-memory macro with 14.6 ns parallel MAC computing time for CNN based AI edge processors," in *proc. of the IEEE Int. Solid-State Circuits Conference-(ISSCC)*, 2019.

[18] W.-H. Chen *et al.*, "A 16Mb dual-mode ReRAM macro with sub-14ns computing-in-memory and memory functions enabled by self-write termination scheme," in *proc. of the IEEE Int. Electron Devices Meeting (IEDM)*, 2017.

[19] C.-X. Xue *et al.*, "A 22nm 2Mb ReRAM compute-in-memory macro with 121-28TOPS/W for multibit MAC computing for tiny AI edge devices," in *proc. of the IEEE Int. Solid-State Circuits Conference-(ISSCC)*, 2020.

[20] W.-H. Chen *et al.*, "CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors," *Nature Electronics*, vol. 2, no. 9, pp. 420–428, 2019.

[21] S. Kvatinsky *et al.*, "MAGIC—Memristor-Aided Logic," *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 61, no. 11, pp. 895–899, 2014.

[22] X. Fang and Y. Tang, "Circuit analysis of the memristive stateful implication gate," *Electronics Letters*, vol. 49, no. 20, pp. 1282–1283, 2013.

[23] P.-E. Gaillardon *et al.*, "The programmable logic-in-memory (PLiM) computer," in *proc. of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2016.

[24] D. Biolek, Z. Biolek, and V. Biolkova, "Pinched hysteretic loops of ideal memristors, memcapacitors and meminductors must be 'self-crossing'," *Electronics letters*, vol. 47, no. 25, pp. 1385–1387, 2011.

[25] J. Rajendran, H. Manem, R. Karri, and G. S. Rose, "An energy-efficient memristive threshold logic circuit," *IEEE Trans. on Computers*, vol. 61, no. 4, pp. 474–487, 2012.

[26] L. Guckert and E. E. Swartzlander, "MAD gates—memristor logic design using driver circuitry," *IEEE Trans. Circuits Syst. II: Exp. Briefs*, vol. 64, no. 2, pp. 171–175, 2016.

[27] E. Linn, R. Rosezin, S. Tappertzhofen, U. Böttger, and R. Waser, "Beyond von Neumann—logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, no. 30, p. 305205, 2012.

[28] K. C. Rahman, D. Hammerstrom, Y. Li, H. Castagnaro, and M. A. Perkowski, "Methodology and design of a massively parallel memristive stateful IMPLY logic-based reconfigurable architecture," *IEEE Trans. on Nanotechnology*, vol. 15, no. 4, pp. 675–686, 2016.

[29] R. B. Hur and S. Kvatinsky, "Memristive memory processing unit (MPU) controller for in-memory processing," in *proc. of the Int. Conf. on the Science of Electrical Engineering (ICSEE)*, 2016.

[30] R. Gharpinde, P. L. Thangkhiew, K. Datta, and I. Sengupta, "A scalable in-memory logic synthesis approach using memristor crossbar," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 2, pp. 355–366, 2018.

[31] P. L. Thangkhiew, R. Gharpinde, and K. Datta, "Efficient mapping of Boolean functions to memristor crossbar using MAGIC NOR gates," *IEEE Trans. Circuits Syst. I: Reg. Papers*, no. 99, pp. 1–11, 2018.

[32] J. J. Yang, D. B. Strukov, and D. R. Stewart, "Memristive devices for computing," *Nature Nanotechnology*, vol. 8, no. 1, p. 13, 2013.

[33] L. Amaru, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A new paradigm for logic optimization," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 35, no. 5, pp. 806–819, 2015.

[34] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, and K. N. Salama, "Memristor-based memory: The sneak paths problem and solutions," *Microelectronics Journal*, vol. 44, no. 2, pp. 176–183, 2013.

[35] Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "Sneak-path constraints in memristor crossbar arrays," in *proc. of the IEEE Int. Symp. on Information Theory (ISIT)*, 2013.

[36] S. Shin, K. Kim, and S.-M. Kang, "Analysis of passive memristive devices array: Data-dependent statistical model and self-adaptable sense resistance for RRAMs," *Proc. of the IEEE*, vol. 100, no. 6, 2012.

[37] Y. Huai *et al.*, "Spin-transfer torque MRAM (STT-MRAM): Challenges and prospects," *AAPPS bulletin*, vol. 18, no. 6, pp. 33–40, 2008.

[38] H. Lee *et al.*, "Evidence and solution of over-reset problem for HfO_x based resistive memory with sub-ns switching speed and high endurance," in *proc. of the Int. Electron Devices Meeting*, 2010.

[39] S. Ikeda *et al.*, "A perpendicular-anisotropy CoFeB–MgO magnetic tunnel junction," *Nature materials*, vol. 9, no. 9, p. 721, 2010.

[40] X. Wang, Y. Chen, H. Xi, H. Li, and D. Dimitrov, "Spintronic memristor through spin-torque-induced magnetization motion," *IEEE electron device letters*, vol. 30, no. 3, pp. 294–297, 2009.

[41] T. Hanyu *et al.*, "Standby-power-free integrated circuits using MTJ-based VLSI computing," *Proc. of the IEEE*, vol. 104, no. 10, 2016.

[42] Y. Wang, Y. Zhang, E. Deng, J.-O. Klein, L. A. Naviner, and W. Zhao, "Compact model of magnetic tunnel junction with stochastic spin transfer torque switching for reliability analyses," *Microelectronics Reliability*, vol. 54, no. 9-10, pp. 1774–1778, 2014.

[43] Y. Wang *et al.*, "Compact thermal modeling of spin transfer torque magnetic tunnel junction," *Microelectronics Reliability*, vol. 55, no. 9-10, pp. 1649–1653, 2015.

[44] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "Majority-inverter graph: A novel data-structure and algorithms for efficient logic optimization," in *proc. of the IEEE Design Automation Conference (DAC)*, 2014.

[45] Berkeley Logic Synthesis and Verification Group. (2005). ABC — a system for sequential synthesis and verification. [Online]. Available: https://people.eecs.berkeley.edu/ alanmi/abc/

Khaled Alhaj Ali received his master degree in Signal, Telecommunications, Image and speech (STIP), in 2016, from the Lebanese University, Beirut, Lebanon. He is pursuing a PhD program at the Electronics department of IMT Atlantique, Brest, France. His current research is focused on the design of circuits and architectures using emerging memristor technologies.

Mostafa Rizk received his Maitrise-es degree in Electronics, M.Sc in Biomedical Physics, and M.Sc in Signal, Telecom, Image, and Speech from the Lebanese University in 2007, 2008 and 2010 respectively. Furthermore, he received his Ph.D. degree in Sciences and Technologies of Information and Communication from Telecom Bretagne, France in 2014 and a Ph.D degree in Electronics and Communication from the Lebanese University in 2015. Dr. Rizk was a post-doctoral researcher at UBS University (France) and Lab-STICC laboratory CNRS, Lorient, France. Currently, Dr. Rizk is an assistant professor at International University of Beirut, Lebanon and associate researcher at IMT-Atlantique, France. His general research interests include both algorithm development for digital baseband components and corresponding hardware/software implementations and digital circuit design; network-on-chip design and new MPSoC architectures based on emerging non-volatile memory technologies. His recent research activities target mainly the information and communication technologies (ICT) in systems of drones.

Amer Baghdadi is a Professor at IMT Atlantique/Lab-STICC laboratory. He received his Engineering degree in 1998, Master of Science degree in the same year and PhD degree in 2002, all from Grenoble INP (Institut National Polytechnique), France. Furthermore, he received the accreditation to supervise research (HDR) in Sciences and Technologies of Information and Communication in 2012 from the University of Southern Brittany (UBS), France. Prof. Baghdadi general technical area concerns both algorithm development for digital baseband components and corresponding hardware/software implementations and digital circuit design. His research activities target mainly digital communication applications, in addition to other application domains, and more particularly the design of flexible digital physical layer for future wireless communication standards. Prof. Baghdadi is IEEE Senior Member. He serves on the technical program committee for several international conferences. He co-authored more than 100 papers on scientific journals and proceedings of international conferences.

Jean-Philippe Diguet is a CNRS director of research at Lab-STICC, Lorient/Brest, France. He received the Ph.D. degree from Rennes University (France) in 1996. In 1997, he has been a visitor researcher at IMEC (Belgium). He has been an associate professor at UBS University (France) until 2002. In 2003, he co-funded the dixip company in the domain of wireless embedded systems. Since 2004 he is a CNRS researcher at Lab-STICC, where he has been heading the MOCS team until 2016. He has been a visitor researcher at the University of Queensland, Australia in 2010 and an invited Prof. at Tohoku University, Japan in Nov. 2014 and May 2019, and at Univ. of São Paulo, Brazil, in Nov. 2016. His current work focuses on various aspects of embedded system design: Designs and Tools for NoC-based MPSoC architectures including memory-based computing, Self-adaptivity for uncertain environments as autonomous vehicles and Design of dedicated hardware accelerators.

Jalal Jomaah was born in Lebanon, in 1967. He graduated from the Institut National Polytechnique de Grenoble (Grenoble INP), France, in 1992. He received the MS and Ph.D. degrees in electronics from the same University, in 1992 and 1995, respectively. In 2002, he obtained the Habilitation diploma from the INPG authorizing him to supervise PhD. Dissertations. He joined the Laboratoire de Physique des Composants à Semiconducteurs (LPCS), INP Grenoble, in 1992, where he has been involved in research on the characterization, modeling, and simulation of Silicon-On-Insulator MOS transistors, RF characterization, Microwave SC devices and Electromagnetic modeling. He became Maître de Conférences (Associate Professor) at Grenoble INP in 1996 where he continued his research activities at IMEP (Institute of Microelectronics, Electromagnetism and Photonics, Grenoble INP/CNRS/UJF). He is now Professor at the Lebanese University where he created a Master of Science in Microwave Engineering and a research laboratorie (LEMT). His main research activities were and are in the field MOS/SOI device physics, fluctuations and low and high frequencies noise, radio-frequencies applications, EM modeling, Microwave detection. He has supervised more than 20 Ph.D. and was involved in several national and international research projects on the low and high frequencies noise, reliability, modeling and characterization of SOI devices for RF applications and low-temperature physics. Dr Jomaah has co-authored over 50 publications in international scientific journals, 150 communications at international conferences (16 invited papers and review articles). He has held visiting professorships at Mc Master University of Canada, Osaka University of Japan, Seoul University of South Korea, Athens University of Greece, Ain Shams University of Egypt, and United Arab Emirates University. He obtained in 2017 from CNRS Lebanon the “Research Excellence Award” and in 2018 the “Career in Science Excellence Award” from Lebanese Association for the Advancement of Science (LAAS).

Naoya Onizawa (M’09) received the B.E., M.E., and D.E. degrees in electrical and communication engineering from Tohoku University, Japan, in 2004, 2006, and 2009, respectively. He was a PostDoctoral Fellow with the University of Waterloo, Canada, in 2011, and McGill University, Canada, from 2011 to 2013. In 2015, he was a Visiting Associate Professor with the University of Southern Brittany, France. He is currently an Assistant Professor with the Research Institute of Electrical Communication, Tohoku University. He is also a Researcher with JST PRESTO, Japan. His main interests and activities are in the energy-efficient VLSI design based on the asynchronous circuits and probabilistic computation and their applications, such as brain-like computers. Dr. Onizawa received the Best Paper Award at the 2010 IEEE ISVLSI, the Best Paper Finalist at the 2014 IEEE ASYNC, and the Kenneth C. Smith Early Career Award for Microelectronics Research at the 2016 IEEE ISMVL.

Takahiro Hanyu (S’87–M’89–SM’12) received the B.E., M.E., and D.E. degrees in electronic engineering from Tohoku University, Sendai, Japan, in 1984, 1986, and 1989, respectively. He is currently a Professor with the Research Institute of Electrical Communication, Tohoku University. His general research interests include nonvolatile logic circuits and their applications to ultra-low-power and/or highly dependable VLSI processors, and post-binary computing and its application to brain-inspired VLSI systems. Dr. Hanyu was a recipient of the Sakai Memorial Award from the Information Processing Society of Japan in 2000, the Judge’s Special Award at the 9th LSI Design of the Year from the Semiconductor Industry News of Japan in 2002, the Special Feature Award at the University LSI Design Contest from ASP-DAC in 2007, the APEX Paper Award of Japan Society of Applied Physics in 2009, the Excellent Paper Award of IEICE, Japan, in 2010, Ichimura Academic Award in 2010, the Best Paper Award of IEEE ISVLSI 2010, the Paper Award of SSDM 2012, the Best Paper Finalist of IEEE ASYNC 2014, and the Commendation for Science and Technology by MEXT, Japan in 2015.