A Mapping Methodology of Boolean Logic Circuits on Memristor Crossbar

Xie, Lei; Du Nguyen, Hoang Anh; Taouil, Mottaqiallah; Hamdioui, Said; Bertels, Koen

DOI
10.1109/TCAD.2017.2695880

Publication date
2018

Document Version
Accepted author manuscript

Published in
IEEE Transactions on Computer - Aided Design of Integrated Circuits and Systems

A Mapping Methodology of Boolean Logic Circuits on Memristor Crossbar

Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Said Hamdioui, Koen Bertels
Laboratory of Computer Engineering,
Delft University of Technology, Delft, the Netherlands
Email: {L.Xie,H.A.DuNguyen,M.Taouil,S.Hamdioui,K.L.M.Bertels}@tudelft.nl

***Abstract*—Alternatives to CMOS logic circuit implementations are under research for future scaled electronics. Memristor crossbar based logic circuit is one of the promising candidates to at least partially replace CMOS technology, which is facing many challenges such as reduced scalability, reliability, and performance gain. Memristor crossbar offers many advantages including scalability, high integration density, non-volatility, etc. The state-of-the-art for memristor crossbar logic circuit design can only implement simple and small circuits. This paper proposes a mapping methodology of large Boolean logic circuits on memristor crossbar. Appropriate place-and-route schemes, to efficiently map the circuits on the crossbar, as well as several optimization schemes are also proposed. To illustrate the potential of the methodology, a multi-bit adder and other nine more complex benchmarks are studied; the delay, area and power consumption induced by both the crossbar and its CMOS control part are evaluated.**

***Index Terms*—Memristor Crossbar, Logic Design, Mapping, Evaluation.**

I. Introduction

As CMOS transistors gradually scale down to the intrinsically physical device limits, CMOS VLSI circuits are facing major challenges such as saturated performance gain, increased leakage power consumption, reduced reliability, and a more complex fabrication process [1–3]. In addition, CMOS-based computers are suffering from memory bottleneck [4], power wall [5], etc. To address these challenges, alternative technologies [6] are under investigation; examples are nanotube [7,8], silicon nanowire FET [9], magnetic/spintronic [10–12], and memristors [13,14]. Among these proposals, memristor crossbar based logic circuit is a promising candidate due to its attractive characteristics in terms of scalability, high integration density, and non-volatility, etc [15,16]. Moreover, based on memristor technology, novel computer architectures for data-intensive applications have been proposed, such as computation-in-memory [17–21], resistive associate processor [22] and Pinatubo [23]; they show a potential of order of magnitude performance improvement as compared to todays' architectures.

To implement such novel computer architectures, logic circuits based on resistive devices, such as memristors, are required; research on this topic is still in infancy stage. As of today, four types of memristor-based logic circuits have been proposed: threshold [24,25], majority [25], material implication [26,27], and Boolean [28,29] logic. Threshold and majority logic circuits use memristor as the weight of inputs and use CMOS current mirror or inverter as the threshold function. Both of them are more suitable for traditional computer architecture as they represent data by using voltage. In contrast, both material implication and Boolean logic seem to be the enabler for the novel computer architectures as they use resistance to represent data, and can be easily integrated with high density memories [30,31]. In [27,32,33], the authors proposed methodologies to implement logic functions using a sequence of material implication operations. However, these methodologies suffer from low speed and require new algorithms to implement arithmetic operations such as addition [27,34,35]. In [28,29], the authors proposed *simple* and *small* Boolean logic designs for memristor crossbar, which partially address the shortcomings of implication logic. Therefore, exploring memristor crossbar logic design for larger circuits is required in order to appropriately assess the potential of such technology.

This paper proposes a mapping methodology of Boolean logic circuits on memristor crossbar to enable the implementation of large logic circuits, and illustrates the methodology for a multi-bit adder. Thereafter, the methodology is applied to nine more complex benchmarks to show its generality. This work is built on our preliminary work published in [29], where the focus was mainly on the implementation of simple Boolean functions. Compared to the preliminary work, the new contributions of this paper are:

- A mapping flow for memristor crossbar enabling large-scale logic circuits.
- Two place-and-route schemes to map large-scale logic circuits on crossbar.
- Design of CMOS peripheral circuits, which act as the control engine of the memristor crossbar.
- Several schemes to optimize the area, delay and power consumption.
- A model to evaluate the performance of the design in terms of area, delay and power consumption, which considers both the crossbar and CMOS parts.

The remainder of this paper is organized as follows. Section II briefly describes the design of resistive Boolean logic. Section III presents the proposed mapping flow, two place-and-route schemes and CMOS circuits to control the crossbar. Section IV discusses several optimization schemes. Section V verifies the methodology using multi-bit adders as a case study, and

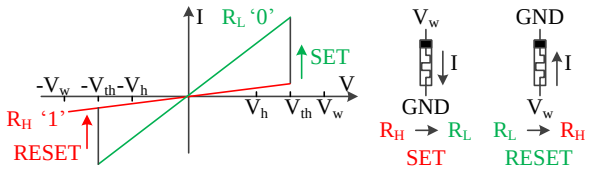

Fig. 1: Ideal Memristor Model.

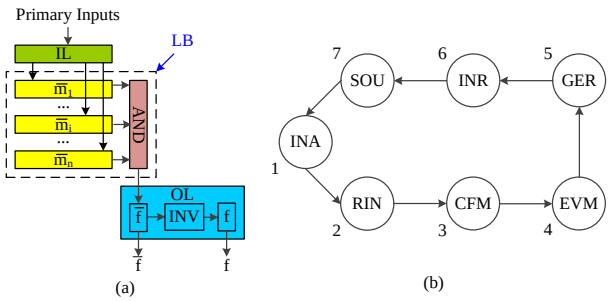

Fig. 2: Working Principle of Boolean Logic: (a) Computing Element, (b) State Machine.

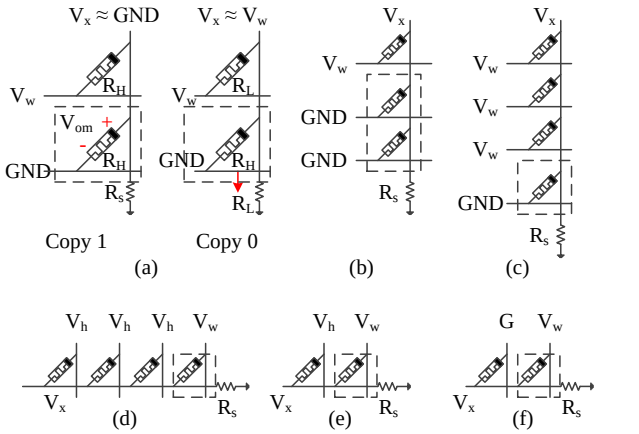

Fig. 3: Implementation of Primitive Operations: (a) Single-Fanout Copy, (b) Multi-Fanout Copy, (c) 3AND1, (d) 3NAND1, (e) INV1, (f) Horizontally Copy.

applies the method to nine more complex benchmarks. Section VI concludes the paper with advantages of the proposal and challenges in the future work.

II. Boolean Logic Design

This section starts first by briefly presenting the memristor model used in this work. Thereafter, it presents the working principle of the resistive Boolean logic we proposed in [29]. Then, the implementation of the primitive (logic) operations (e.g., AND) is given; these are used to build one-bit full adder, which is used in this work later.

A. Memristor Model

The left part of Fig. 1 shows the current-voltage relation of the ideal memristor model used in this work; it has a high (R_H) and low (R_L) resistive states. The memristor switches from one resistive state to another when the absolute value of the voltage across the device is greater than its threshold voltage V_{th}. Otherwise, it stays in its current resistive state. Typically, a memristor requires two different switching threshold voltages to switch from low to high resistance (RESET) and from high to low resistance (SET) [36,37] (see the right part of Fig. 1). For simplicity, we assume that the threshold voltage V_{th} (in absolute value) for both switchings are the same. Here, we use the ideal model as this paper focuses on mapping methodology. Nevertheless, any model can be used such as those in [36,37].

B. Working Principle of Boolean Logic

Our Boolean logic design approach [29] is able to implement any logic function f expressed in the format of sum-of-product (i.e., $f = m_1 + \cdots + m_i + \cdots + m_n = \overline{m_1} \cdot \cdots \cdot \overline{m_i} \cdot \cdots \cdot \overline{m_n}$ where m_i is a minterm of inputs, n the number of minterms); its implementation is referred to as a computing element (CE) as shown in Fig. 2(a). A CE consists of an input latch (IL), an output latch (OL), and a logic block (LB). The LB consists of all the minterms of the Boolean function; each $\overline{m_i}$ is realized using a NAND gate consisting of several memristors depending on the number of its inputs. By ANDing all the $\overline{m_i}$, the $\overline{f}$ can be generated. Finally, $\overline{f}$ is inverted to obtain f. The input and output latches are composed of several memristors depending on the number of inputs and outputs of the Boolean function, respectively.

Memristor-based logic design described above requires a CMOS circuit to control the crossbar part; its behaviour is captured by a state machine as shown in Fig. 2(b). The state machine requires 7 states:

- INA: INitialize All the memristors of a CE to R_H. This state requires RESET operations.
- RIN: Receive INputs. The IL of the CE receives the inputs from primary inputs using CMOS controller to program the resistance of memristors, or from the OL of the previous CE using copy operations. Therefore, this state requires SET, RESET, or copy operations.
- CFM: ConFigure all Minterms. All the minterms are configured simultaneously through copying inputs stored in IL to each minterm in parallel. Hence, this state requires copy operations.
- EVM: EValuate all Minterms. All the $\overline{m_i}$ are evaluated at the same time; each $\overline{m_i}$ is implemented by an NAND operation.
- GER: GEnerate Result. The results of EVM are used as inputs of an AND gate to generate $\overline{f}$, which is the negation of the Boolean function. Therefore, this state needs an AND operation.
- INR: INvert Result. The result of GER is inverted to produce the final result f. Hence, an inversion (INV) is required.
- SOU: Send OUtputs. Finally the result stored in OL is sent to IL of the next CE. Hence, copy operations are employed.

The above shows that in order to implement Boolean logic using the described approach, at least five primitive operations are needed: RESET, copy, NAND, AND, and INV; RESET

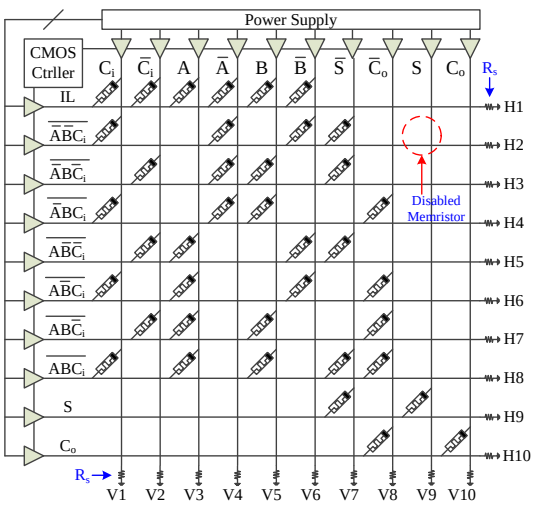

Fig. 4: Implementation of One-Bit Full Adder.

is already described and is shown in Fig. 1; the remaining operations are discussed next.

C. Implementations of Primitive Operations

All primitive operations use R_H and R_L to represent logic 1 and 0, respectively. Fig. 3 shows the implementations of all primitive operations; each implementation consists of one or multiple input and an output memristors. The output memristors are all initialized to R_H prior to any operation (i.e., RESET operation of state INA), and are surrounded by a dashed-line box in the figure. The voltage across the output memristor(s) is denoted by V_{om}, while the voltage of the floating nanowire is denoted by V_x; both of them are used to explain the working principle of primitive operations. In addition, all primitive operations consist of a resistor R_s (see Fig. 3), which satisfies the condition $R_L \ll R_s \ll R_H$; this is required to guarantee that the voltage across the output memristor is close to the desired voltage for proper operations [27]. Fig. 3(a), (c), (d) and (e) show primitive operations with one output memristor or single fanout. Multi fanout operations can be realized by employing multiple output memristors; Fig. 3(b) shows a two-fanout copy operation by employing two output memristors. Note that the positive terminal of each memristor is connected to the vertical nanowire. Fig. 3(f) shows the horizontal copy; it will be used later.

To control primitive operations, three different voltages are required: V_w, V_h, and *GND*; see Fig. 1. V_w is used to program the memristor; V_h is used to minimize the impact of sneak path currents by half-select voltage strategy [38]; V_h is then applied to memristors which are not involved in particular operations within a crossbar. V_h is also used to control NAND and INV as shown in Fig. 3(d) and Fig. 3(e). The relationship between V_w, V_h, *GND* and V_{th} is $0 < V_h = \frac{V_w}{2} < V_{th} < V_w$. This relationship guarantees $V_w - V_h = 2V_h - V_h = V_h < V_{th}$ which prevents undesired switching of non-accessed memristors [34,39].

The copy operation will be used as an example to explain its working principle; the other operations can be understood in a similar way, and more details can be found in [29]. Before performing any operation, the data should be stored in the right locations. Then, for a copy operation, a voltage $V_w > V_{th}$ and *GND* are applied to the input and output memristors, respectively, as shown in Fig. 3(a). In case of copy 1 (R_H), V_x is around 0 as $R_s \ll R_H$. Therefore, $V_{om} = V_x - 0 \approx 0 < V_{th}$. As a result, the output memristor stays at R_H. In case of copy 0 (R_L), $V_x \approx V_w$ as $R_L \ll R_s$. Therefore, $V_{om} = V_x - 0 \approx V_w > V_{th}$. As a result, the output memristor switches to R_L.

TABLE I: Control Voltages for One-Bit Full Adder

States	Control Voltages					
	Row			Column		
	IL	LB	OL	IN	Output	
					OUT	OUTN
	H1	H2-8	H9-10	V1-6	V8,10	V7,9
INA	V_w	V_w	V_w	G	G	G
RIN	G	V_h	V_h	F	V_h	V_h
CFM	V_w	G	V_h	F	V_h	V_h
EVM	V_h	F	V_h	V_h	V_h	V_w
GER	V_h	V_w	G	V_h	V_h	F
INR	V_h	V_h	F	V_h	V_w	V_h
SOU	V_h	V_h	V_w	V_h	V_h	V_h

D. One-Bit Full Adder

The sum (S) and carry (C_o) of a one-bit full adder (FA) are expressed by Eq.1 [40]. Each equation consists of four minterms.

$$\begin{aligned} S &= \overline{\bar{A}\bar{B}C_i} \cdot \overline{\bar{A}B\bar{C}_i} \cdot \overline{A\bar{B}\bar{C}_i} \cdot \overline{ABC_i} \\ C_o &= \overline{\bar{A}BC_i} \cdot \overline{A\bar{B}C_i} \cdot \overline{AB\bar{C}_i} \cdot \overline{ABC_i} \end{aligned} \quad (1)$$

Fig. 4 shows the crossbar implementation of this FA using the principle of Fig. 2 and Fig. 3. For convenience, H# and V# are used to denote a horizontal and vertical nanowire, respectively; a memristor in the crossbar is denoted by M(H#,V#). To implement the FA, two types of memristors are used: active (which can switch between two resistive states) and disabled (which is permanently high resistance) memristor. In the figure, the junctions where disabled memristors are located have no devices as shown in Fig. 4.

The FA is implemented using a CE consisting of an IL, LB and OL. The IL is mapped on the memristors M(H1,V1-V6), since IL consists of primary inputs and their complements. The remaining memristors on H1 are disabled. The LB consisting of seven minterms is mapped on H2-H8, where the minterm $\overline{ABC_i}$ is shared by sum and carry. Each minterm is implemented by placing active memritors at junctions formed by the horizontal nanowire (representing the minterm) and (a) the vertical nanowires associated with the minterm's inputs, or (b) an output for which the minterm is part of. For example, $\overline{\bar{A}\bar{B}C_i}$ on H2 is a minterm of sum. Therefore, memristors M(H2,V1=C_i), M(H2,V4=$\bar{A}$), M(H2,V6=$\bar{B}$), and M(H2,V7=$\overline{S}$) are active; while the remaining memristors on H2 are disabled. The four minterms of S and those of C_o (see Eq. 1) are then ANDed in parallel by column V7=$\overline{S}$ and V8=$\overline{C_o}$, respectively. The OL is realized by H9 and H10. The

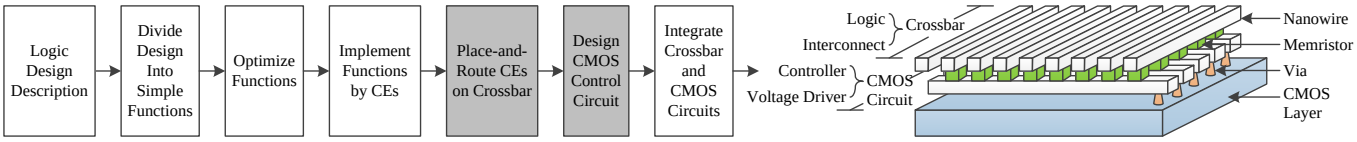

Fig. 5: Mapping Flow and Implementation.

results provided by the two ANDs are then stored at M(H9, V7) and M(H10, V8), which are thereafter inverted and stored at M(H9, V9=S) and M(H10, V10=C_o), respectively. Note the FA implementation requires 10 rows and 10 columns.

To perform the desired primitive operations during each state, appropriate voltages are applied to each horizontal and vertical nanowire of the CE. Table I summarizes the required voltages for the FA; they are straightforwardly derived from the implementations of the primitive operations as shown in Fig. 3. Each row (horizontal nanowire) is associated with the implementation of IL, LB or OL; while the columns are associated with the primary inputs (IN) or outputs (OUT), or their complements (OUTN). For instance, to perform copy operations required by CFM state to configure all minterms in parallel, V_w is applied to row H1, *GND* (G) is applied to rows H2-H8, while columns V1-V6 are left floating (F) (see the row CFM of Table I). It is worth noting that the remaining rows (H9-H10) and columns (V7-V10) are set to V_h in order to minimize the impact of sneak path currents [34,39]. All the circuits mentioned in this paper use this methodology to solve the sneak path problems.

III. Mapping Methodology and Implementation

This section first presents the mapping flow for Boolean logic based on memristor crossbar. Subsequently, it highlights the challenges of place-and-route within crossbar and potential solutions. Finally, it presents the CMOS circuit used to control the memristor crossbar.

A. Mapping Flow

Fig. 5 shows the flow of the mapping methodology. The entire design is first divided into multiple simple Boolean functions (e.g., look-up tables), which can be further optimized by EDA tools such as ESPRESSO [41]. Next, the optimized Boolean functions are implemented using CEs, as presented in Section II. Thereafter, all CEs are placed and routed within the crossbar, and the CMOS circuit (used to control the crossbar) is designed. Finally, the memristor crossbar and CMOS control circuits are integrated together by stacking the crossbar on the CMOS part as shown in Fig. 5. The first three steps are described in our previous work [29]; this section will focus on the place-and-route and CMOS circuit design.

B. Potential Solutions to Place-and-Route

To highlight the challenge of place-and-route, a 4-bit ripple carry adder is used as an example. Fig. 6(a) shows this 4-bit adder which uses four FAs of Fig. 4 as building blocks. A naive solution to place and route these FA blocks is to arrange them adjacently to each other, as the two options

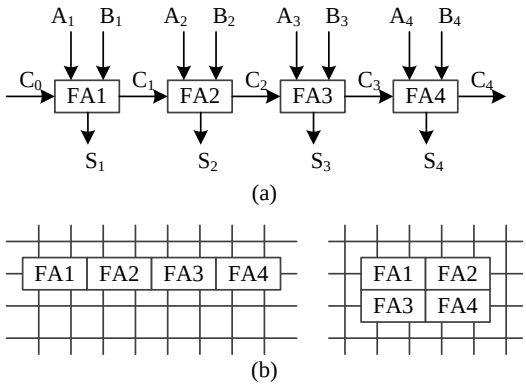

Fig. 6: Place-and-Route Challenges: (a) Block Diagram, (b) Layouts Sharing Nanowires.

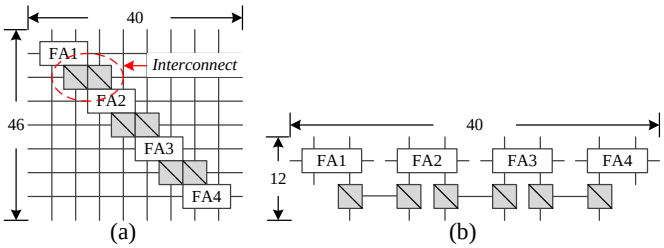

Fig. 7: Place-and-Route Schemes: (a) Diagonal Scheme, (b) Isolated Scheme.

shown in Fig. 6(b). However, typically this cannot be done as the operations of each FA require specific control voltages applied to the horizontal and vertical nanowires (see Table I); sharing these nanowires between different FAs will lead to a conflict of control voltages; and hence impacting each other's operations. Therefore, special attention should be given to place-and-route.

Potential solutions to address the above challenge are:

- Preventing each pair of FAs from sharing the same horizontal or vertical nanowires;
- Breaking the nanowires within the crossbar in order to isolate each FA;
- Stacking FAs on each other rather than having them within the same crossbar layer.

In the rest of this subsection, we will discuss the first two potential solutions in more details. Actually the third potential solution is similar to the second one except that the FAs are stacked.

To prevent each pair of FAs from sharing the same horizontal and vertical nanowires, diagonal place-and-route scheme is proposed. Fig. 7(a) shows the 4-bit adder which is placed and routed using the diagonal scheme. All the FAs are placed

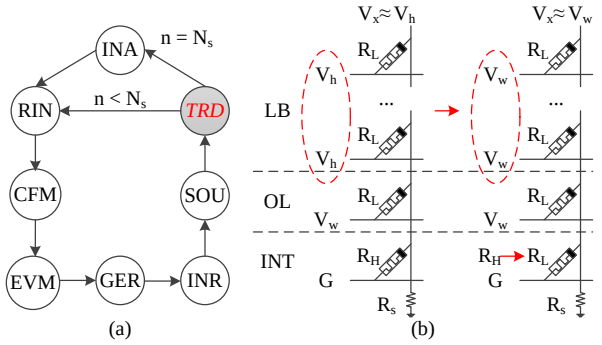

Fig. 8: CMOS Controller: (a) Modified State Machine, (b) State SOU.

TABLE II: Control Voltages for Multi-CE Design

State	Control Voltage						
	Row				Column		
	IL	LB	OL	INT	IN	OUT	OUTN
INA	V_w	V_w	V_w	V_w	G	G	G
RIN	G	V_h	V_h	V_h	F	V_h	V_h
CFM	V_w	G	V_h	V_w	F	V_h	V_h
EVM	V_h	F	V_h	V_h	V_h	V_h	V_w
GER	V_h	V_w	G	V_h	V_h	V_h	F
INR	V_h	V_h	F	V_h	V_h	V_w	V_h
SOU	V_h	V_w	V_w	G	V_h	V_h	V_h
TRD	V_h	V_h	V_h	F	V_w	G	G

in a diagonal pattern, and therefore, no FAs share the same horizontal and vertical nanowires. To route the carry and its negation between FAs (e.g., carry C_1 and it negation $\overline{C_1}$ between FA1 and FA2), two extra rows are reserved for interconnect [42]. As a result, the implementation of the four-bit adder of Fig. 6(a) is mapped on a 46×40 crossbar using the diagonal scheme; see Fig. 7(a).

Fig. 7(b) shows the 4-bit adder which is placed and routed using isolated scheme. By breaking the nanowires within the crossbar, each pair of FAs is isolated. Therefore, all the FA units can be placed adjacently, and the adder requires less crossbar area. Similar to the diagonal scheme, two extra rows are reserved for interconnect to route carry between FAs. The neighbour interconnect segments are isolated with each other, and each FA only connects to the interconnect segment connecting its up- and down-stream FAs. As a result, the implementation of the four-bit adder of Fig. 6(a) is mapped on a 12×40 crossbar using the isolated scheme; see Fig. 7(b). Note that the isolated scheme consumes less crossbar area than the diagonal scheme to place and route the same design; e.g., the crossbar area is reduced from 46×40 to 12×40; a reduction of 74%.

C. CMOS Control Circuits

To control the memristor crossbar, a CMOS circuit is employed; it consists of a controller and voltage drivers. The behaviour of the controller is captured by the state machine in Fig. 8(a); it is generated based on the state machine of Fig. 2(b). As the crossbar consists of multiple CEs, the state machine of Fig. 2(b) is extended with an extra state TRD (transfer data), which is needed to horizontally transfer

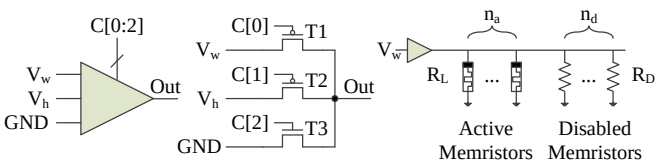

Fig. 9: CMOS Voltage Drivers.

the carry (using the interconnect) between FAs. Note that the execution of the state machine of Fig. 8(a) needs to be repeated N_s times after the initialization, where N_s is the number of stages (i.e., FAs) that the crossbar consists of. For example, N_s for the design of Fig. 6(a) is 4. Consequently, a design with N_s stages requires $7N_s + 1$ execution steps, where 7 is the number of states (initialization excluded).

Table II summarizes the control voltages required for each state of Fig. 8; it is derived from Table I. As already mentioned, additional rows are needed for interconnects; they are included in the table and denoted as 'INT'. Note that the states in the table are extended with state TRD.

In addition, the control voltages of state CFM and SOU in Table II are different from Table I, due to the impact of sneak path currents [43]. Fig. 8(b) shows state SOU as an example. In state SOU, the data in OL is copied to INT; therefore, voltage V_w and *GND* are applied to row OL and INT, respectively. To reduce the impact of sneak path currents, V_h are typically applied to rows LB as half-select voltage [44]. Let assume that OL stores '0' (R_L). As OL stores the results of AND operations of state GER, at least one of the memristors in rows LB is R_L as shown in the left part of Fig. 8(b). After applying control voltages, the voltage of the floating nanowire V_x is around V_h, and the voltage across the output memristor in INT is $V_x - 0 \approx V_h < V_{th}$. As a result, the output memristor stays at R_H, and cannot copy '0'. To solve this issue, V_w is applied to row LB, and hence $V_x \approx V_w > V_{th}$; see the right part of Fig. 8(b). Consequently, the output memristor in INT switches to R_L, and copies 0 from OL.

Next, we will illustrate how to design a voltage driver while taking the restrictions of the crossbar design into account. Fig. 9 shows a possible implementation of voltage drivers, which are parts of the control circuit. A voltage driver is composed of one NMOS and two PMOS pass transistors; the state (i.e. closing or opening) of these transistors are controlled by three-bit signals C[0:2], which are provided by the CMOS controller. To drive a nanowire connecting active memristors as shown in the right part of Fig. 9, the transistors should provide enough current. Therefore, their width-to-length ratio $\frac{W}{L}$ should be carefully determined. Let assume that we have n_a active and n_d disabled memristors. To program a single active memristor, the current supplied by a NMOS transistor should be greater than $I_w = \frac{V_w}{R_L}$ [45]. Therefore, the NMOS transistor typically needs $(\frac{W}{L})_n = 2$ and its area has to be $A_n = 6F^2$ [46,47], where F is the feature size of CMOS technology;

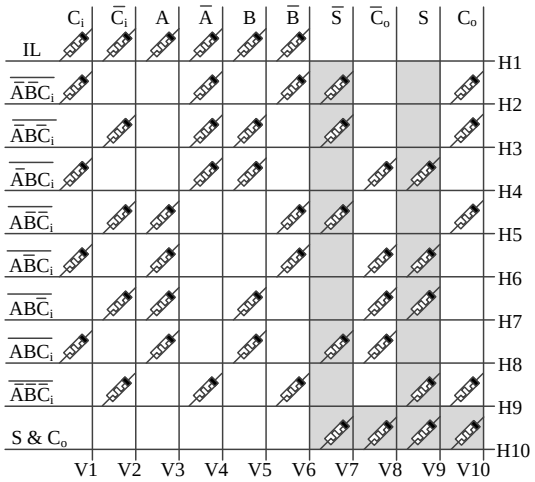

Fig. 10: The FA Calculating All Outputs Simultaneously.

see Eq. 2.

$$\begin{cases} I_{\rm w} = \frac{V_w}{R_L}, \\ (\frac{W}{L})_{\rm n} = 2, \\ A_{\rm n} = 6F^2 \end{cases} \tag{2}$$

To drive n_a active memristors in parallel, $(\frac{W}{L})_{\rm n}$ and $A_{\rm n}$ of the NMOS should be increased n_a times in order to provide the required current $I_{\rm w}$, as shown in Eq. 3.

$$\begin{cases} I_{\rm w} = n_a \frac{V_w}{R_L}, \\ (\frac{W}{L})_{\rm n} = 2n_a, \\ A_{\rm n} = 6n_a F^2 \end{cases} \tag{3}$$

In addition, assume that we have n_d disabled memristors; each of them consumes the current $I_{\rm D}$=$\frac{V_w}{R_D}$, and hence the NMOS transistor should be adjusted as to compensate for the current through n_d disabled memristors, as shown in Eq.4.

$$\begin{cases} I_{\rm w} = n_a \frac{V_w}{R_L} + n_d \frac{V_w}{R_D} = (n_a + \frac{R_L}{R_D} n_d) \frac{V_w}{R_L}, \\ (\frac{W}{L})_{\rm n} = 2(n_a + \frac{R_L}{R_D} n_d), \\ A_{\rm n} = 6(n_a + \frac{R_L}{R_D} n_d) F^2 \end{cases} \tag{4}$$

As the mobility of PMOS transistor is typically twice lower than that of NMOS [48], the $\frac{W}{L}$ of the PMOS has to be twice larger than that of NMOS. Therefore, the $(\frac{W}{L})_{\rm p}$ and area $A_{\rm p}$ of PMOS are obtained as expressed in Eq.5.

$$\begin{cases} (\frac{W}{L})_{\rm p} = 2(\frac{W}{L})_{\rm n} = 4(n_a + \frac{R_L}{R_D} n_d), \\ A_{\rm p} = 2A_{\rm n} = 12(n_a + \frac{R_L}{R_D} n_d) F^2 \end{cases} \tag{5}$$

Finally, the total area $A_{\rm vd}$ of a single voltage driver which consists of one NMOS and two PMOS pass transistors is given in Eq.6. Typically, $\frac{R_D}{R_L} > 5 \times 10^4$ [49]; hence, the number of active memristors n_a dominates the area of the voltage driver.

$$A_{\rm vd} = A_{\rm n} + 2A_{\rm p} = 30(n_a + \frac{R_L}{R_D} n_d) F^2 \approx 30 n_a F^2 \tag{6}$$

IV. Mapping Optimization Schemes

This section presents three mapping schemes to optimize the delay and/or area of a memristor crossbar logic design. These schemes can be used separately or in a combination.

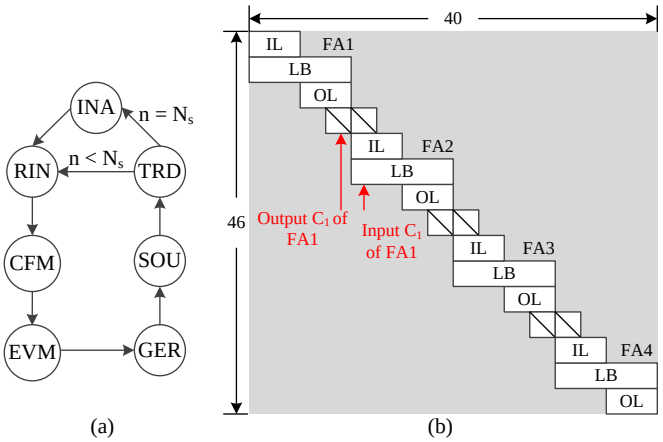

Fig. 11: Diagonally-Mapped Adder: (a) State Machine, (b) Layout.

A. Scheme 1: Calculate All Outputs Simultaneously

The first optimization scheme is calculating both the primary and complementary outputs at GER state. For example, the FA of Fig. 4 is optimized as shown in Fig. 10; its outputs S and $\overline{S}$ are expressed by Eq.7.

$$\begin{aligned} S &= \overline{\bar{A}B\bar{C}_i} \cdot \overline{A\bar{B}\bar{C}_i} \cdot \overline{AB\bar{C}_i} \cdot \overline{\bar{A}\bar{B}C_i} \\ \overline{S} &= \overline{\bar{A}\bar{B}\bar{C}_i} \cdot \overline{\bar{A}BC_i} \cdot \overline{A\bar{B}C_i} \cdot \overline{ABC_i} \end{aligned} \tag{7}$$

All the eight required minterms of output S and $\overline{S}$ are implemented by placing active memristors on related junctions in a similar manner as Fig. 4. Therefore, all the minterms of both S and $\overline{S}$ are calculated at state EVM, and then ANDed to obtain the output S and $\overline{S}$ at state GER. The similar approach can be applied to the output C_o and $\overline{C_o}$; see Fig. 10. In addition, the OL can be implemented by only a single row, as $\overline{S}$ and $\overline{C_o}$ are calculated at state GER, rather than by inverting S and C_o as in Fig. 4(a). As both the primary and complementary outputs are obtained at state GER, state INR of Fig. 8(a) can be removed as shown in Fig. 11(a); hence, the four-bit adder of Fig. 7 (a) reduces the number of execution steps from $7N_s$+1=29 to $6N_s$+1=25. Fig. 11(b) shows the layout of the new four-bit adder implementation, where the FAs of Fig. 4 have been replaced with the FA of Fig. 10. Note that this new implementation requires the same area (i.e., 46×40) as the initial design of Fig. 7(a).

However, this scheme typically requires more area as all the $2^{\rm n}$ minterms of an n-input Boolean function must be mapped on the crossbar, instead of the required minterms only. For instance, the output S only needs four minterms as shown in Eq.7, and they can be implemented by only four rows in the crossbar. As both primary and complementary outputs are required as the inputs of the next FA stage, the output $\overline{S}$ should also be calculated, and hence, all eight minterms should be implemented by eight rows of the crossbar. To alleviate the incurred area overhead, several Boolean functions that have the same minterms can be implemented using the same share of hardware. For instance, the sum and carry of a FA are implemented together as shown in Fig. 10. As a result, it has the same area as the FA of Fig. 4.

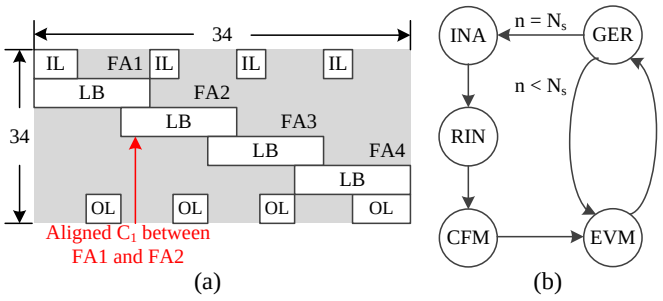

Fig. 12: Align Intermediate Signals: (a) Layout, (b) State Machine.

B. Scheme 2: Align Intermediate Signals

The area and execution steps of the four-bit adder in Fig. 11(b) can be further reduced by aligning the intermediate signals between computing elements. For instance, all the carries between FAs (e.g., FA1 and FA2) of Fig. 11(b) are aligned in the same column as shown in Fig. 12(a). Therefore, the columns initially used for carry transfer are removed. In addition, the carry of a FA can be directly stored in the minterms of the next stage FA, as the carry has been aligned in the same column. Hence, the extra rows initially allocated for interconnect, as well as the parts of IL and OL initially required to store carries, are removed. Finally, the four ILs to store the primary inputs are rearranged at the top of the crossbar, while the four OLs to store final results are rearranged at the bottom of the crossbar, as shown in Fig. 12(a). Fig. 12(b) shows the state machine required by the new implementation of Fig. 12(a). The adder receives all the primary inputs of the four FAs (i.e., C_0, A_i, B_i, $0 \leq i \leq 3$) at state RIN. Then, they are copied to the four FAs to configure their minterms at state CFM. For each stage, the state EVM generates the minterms of the corresponding FA, and state GER ANDs these minterms to generate the sum, carry, and their complements. It is worth noting that the number of execution steps are reduced from $7N_s$+1 (design of Fig. 7(a)) to $2N_s$+3; e.g., for N_s=4, the reduction is 62%. Moreover, the crossbar area is also reduced. For our 4-bit adder case study, the crossbar area is reduced from 46×40 (default design) to 34×34 (Fig. 12(a)); a reduction of 37%.

This optimization scheme is not applicable to all designs; it strongly depends on the place-and-route scheme. For instance, the design using the isolated scheme (e.g., Fig. 7(b)) does not support this scheme; the intermediate signals cannot be aligned in the same column since computing elements are isolated from each other.

C. Scheme 3: Combine Data Transfer and Inversion

Instead of producing intermediate result C_i and its complement $\overline{C_i}$ by each FA, we can rather produce only one of them (e.g., $\overline{C_i}$); the other one (e.g., C_i) will be generated while communicating the intermediate result to the next FA stage. Hence, the required crossbar column to produce C_i can be removed, resulting in less execution steps and area. Applying this scheme to default design of Fig. 7(b) results in the implementation shown in Fig. 13(a). The part where

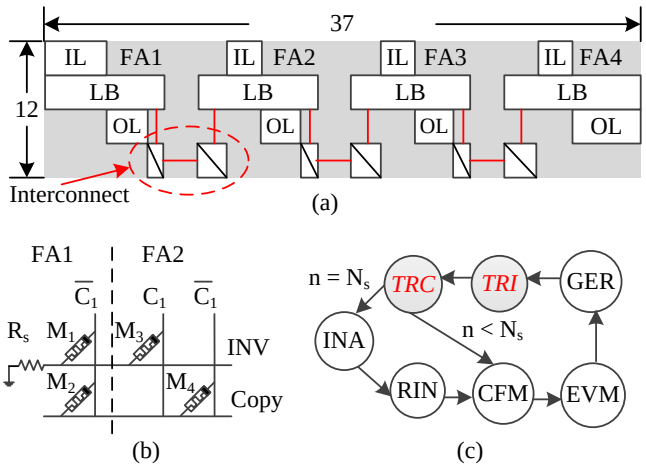

Fig. 13: Use Inversion to Transfer Data: (a) Layout, (b) Interconnect, (c) State Machine.

the combination of data transfer and inversion take place is highlighted, and illustrated in Fig. 13(b). The output $\overline{C_1}$ of FA1 is generated at state GER and directly stored in two memristors M_1 and M_2 of the interconnect. The interconnect feeds FA2 with C_1 and $\overline{C_1}$ via row INV and row Copy; C_1 is provided by inverting $\overline{C_1}$, while $\overline{C_1}$ is provided by just copying M_2 to M_4. In addition, the four ILs to store the primary inputs are rearranged at the top of the crossbar, while the four OLs to store final results are rearranged below the FAs as similar to Fig. 12(a).

Fig. 13(c) shows the state machine required by this new implementation; it makes use of two additional states; transfer using inversion (TRI), and transfer using copy (TRC). The adder receives all the primary inputs of the four FAs at state RIN. For each stage, state CFM configures the minterms of the corresponding FA; state EVM generates the results of these minterms; state GER provides logic AND of these minterms to obtain the complementary carry (e.g., $\overline{C_1}$); state TRI transfers C_i to the next stage using inversion; state TRC feeds the next FA stage with $\overline{C_i}$. Note that state TRI and TRC cannot be combined, because inversion and copy require different control voltages to the column related to output carry (e.g., $\overline{C_1}$ of FA1); see also Fig. 3(e) and (f). It is worth noting that the number of execution steps is reduced from $7N_s$+1=29 (design of Fig. 7(b)) to $5N_s$+2=22 steps; a reduction of 25%. Meanwhile, the area is also reduced from 12×40 to 12×37; a reduction of 7.5%.

V. Evaluation

To validate the proposed approach, we select four designs as a case study, and perform two experiments on all of them. These four designs are four-bit ripple carry adders with different place-and-route and optimization schemes; they are:

- ID design: **I**nitial design based on **D**iagonal place-and-route scheme as shown in Fig. 7(a).
- II design: **I**nitial design based on **I**solated place-and-route scheme as shown in Fig. 7(b).
- OD design: An **O**ptimized version of **ID** design which is shown in Fig. 12(a); it incorporates the optimization

TABLE III: Description of Benchmarks

Circuit	Function	No. Input	No. Output	No. LUT4
alu4	ALU	14	8	1522
apex2	Logic	39	3	1878
apex4	Logic	9	19	1262
des	Data Encryption	256	245	1591
ex5p	Logic	8	63	1064
misex3	Logic	14	14	1397
pdc	Logic	16	40	4575
seq	Logic	41	35	1750
spla	Logic	16	46	3690

scheme 1 (i.e., calculate all outputs simultaneously) and 2 (i.e., align intermediate signals). Note that it is not possible to incorporate all the three discussed optimization schemes in the ID design. E.g., as the use of optimized scheme 2 removes the interconnects between FAs, scheme 3 (which makes use of the interconnect) cannot be used. Based on the nature of the three schemes, incorporating scheme 1 and 2 is the best in order to optimize delay.

- OI design: An **O**ptimized version of I**I** design which is shown in Fig. 13(a); it incorporates optimization scheme 1 and 3. Again, incorporating all the three schemes is not possible. Hence, scheme 1 and 3 are the best to use with II design in order to optimize the delay.

The three experiments performed are the following.

- Verification of the mapping methodology; in this experiment, exhaustive SPICE simulations of all the designs are performed in order to check the correct functionality of the generated designs with all the possible input combinations.
- Evaluation and comparison of the four designs; in this experiment, all the four-selected designs are evaluated and compared in terms of area, delay and power consumption.
- To further illustrate the generality of our mapping methodology, it is applied to nine benchmarks selected from MCNC20 benchmark suite [50]. Table III summarizes the functions of the benchmarks and their input number, output number and number of required 4-input LUTs. These benchmarks spread in a wide range of input number, output number and circuit size (quantified by the number of LUTs). Area and delay are used as performance metrics to compare the initial and optimized designs.

Next, we first briefly review the simulation platform, and the parameters and models used to evaluate performance metrics. Thereafter, we provide the results.

A. Simulation Setup and Performance Metrics

The simulation platform consists of a memristor crossbar, a CMOS controller, and voltage drivers. The memristor model, controller, and voltage drivers are described by Verilog-A, and the crossbar array by SPICE netlist. The behavioural function of each of four selected designs is first verified using HSPICE simulator. Thereafter, each initial design and its optimized version are evaluated and compared with each other in terms of area, delay and power consumption. Note that memristor models are used for SPICE simulations, while the controller and drivers are described by Verilog-A so that the entire design (consisting of both the crossbar and CMOS part) can be simulated and verified using HSPICE simulator. To estimate the performance of CMOS controller, we will use a Verilog version.

TABLE IV: Simulation Parameters

Parameter	Description	Value
Technology		
Memristor (TaO_x) [49,51,52]		
F (nm)	Feature size	90
T (nm)	Thickness of TaO_x	8
V_{th} (V)	Threshold voltage	1.5
R_L ($k\Omega$)	Low resistance	200
$\frac{R_H}{R_L}$	–	7k
$\frac{R_D}{R_H}$	–	50
A_m (μm^2)	Area of a memristor	0.0324
T_{sw}(ns)	Switching time (max of SET and RESET)	1.71
E_m	Endurance of a memristor (max switching number)	10^{12}
Nanowire (Copper) [53]		
κ	Dielectric constant of interlayer spacing	3.9
ρ ($\mu\Omega$cm)	Resistivity of Copper	8
T_{nw}(nm)	Thickness of the nanowire ($= F$)	90
W_{nw}(nm)	Width of the nanowire ($= F$)	90
C_{nw} (fF/μm)	Capcitance in unit length	0.26
R_{nw} (Ω/μm)	Resistance in unit length	9.88
CMOS		
Use UMC 90nm Library @ 500MHz)		
Design [29]		
N_R	No. of rows in crossbar	–
N_C	No. of columns in crossbar	–
N_{step}	No. of execution steps	–
$n_{a,xbar}$	No. of all active memristors in the whole crossbar	–
V_w (V)	Program voltage	2.1
V_h (V)	Half-select voltage	1.05
$\frac{R_s}{R_L}$	–	10

To evaluate the benchmarks in Table III, a Matlab script is developed to read, map and estimate the metrics under consideration both for initial and optimized design versions; the optimized version uses diagonal mapping scheme and the optimization scheme 2. The inputs of the Matlab script are files in Berkeley logic interchange format (BLIF) of each benchmark [50]; BLIF consists of the minterms of each 4-input LUT which can be directly mapped to crossbar using our methodology.

Table IV summarizes the used simulation parameters; they are classified into technology and design parameters. The technology parameters are taken from [49,51–53], and provide realistic values for memristor as well as nanowires used to build the crossbar. For CMOS controller and voltage drivers, the UMC 90nm library is used. On the other hand, the design related parameters consist of those specifying the design itself (e.g., N_{step}, which is different for different designs), and those which specifies the requirements for the correctness of the design operations (e.g., values of control voltages of the crossbar) taken from [29,54]. Four parameters (i.e., N_R, N_C, N_{Step}, and $n_{a,xbar}$) are design dependent; hence for different benchmarks, they will have different values.

Three metrics are used to evaluate the performance: area, delay, and power consumption, while considering the

crossbar, CMOS voltage drivers and CMOS controller. Next, we will show how these evaluation metrics are determined. An adder is used as an example to illustrate the evaluation model and other benchmarks use the similar model.

The area of a single adder (A_{adder}) is expressed in Eq.8.

$$A_{\text{adder}} = \max\{A_{\text{xbar}}, A_{\text{cmos}}\} \tag{8}$$

where A_{adder} is defined as the maximum of the two values A_{xbar} and A_{cmos}, where A_{xbar} gives the crossbar area and A_{cmos} the area of entire CMOS part; note that we select only the max of the two values as the crossbar is stacked on the top of CMOS part.

A_{xbar} is a product of (N_{R}+1) crossbar rows by (N_{C}+1) columns as expressed in Eq.9.

$$\begin{cases} A_{\text{xbar}} &= (N_{\text{R}}+1)\cdot(N_{\text{C}}+1)\cdot A_{\text{m}} \\ A_{\text{m}} &= 4F^2 \end{cases} \tag{9}$$

where '+1' is needed as the implementation requires the use of a resistive element R_s to perform the appropriate logic operations (see e.g., Fig. 3(a)). A_{m} gives the memristor area.

A_{cmos} consists of the area of all voltage drivers ($A_{\text{vd,all}}$) and that of the control state machine (A_{ctrl}) as expressed in Eq.10.

$$\begin{cases} A_{\text{cmos}} &= A_{\text{vd,all}} + A_{\text{ctrl}} \\ A_{\text{vd,all}} &= A_{\text{vd,row}} + A_{\text{vd,col}} \\ &= \sum_{i=1}^{N_{\text{R}}} \left(30 n_{\text{a,i}} F^2\right) + \sum_{j=1}^{N_{\text{C}}} \left(30 n_{\text{a,j}} F^2\right) \\ &= 30F^2\big(\sum_{i=1}^{N_{\text{R}}} n_{\text{a,i}} + \sum_{j=1}^{N_{\text{C}}} n_{\text{a,j}}\big) \\ &= 30F^2(n_{\text{a,xbar}} + n_{\text{a,xbar}}) \\ &= 60 n_{\text{a,xbar}} F^2 \end{cases} \tag{10}$$

The value of $A_{\text{vd,all}}$ is derived from Eq.6, which expresses the area of a single voltage driver used to drive a row or column with n_a active memristors. $A_{\text{vd,all}}$ is the sum of all the voltage drivers used to drive both rows and columns as shown in Eq.10, where $n_{\text{a,xbar}}$ is the total number of all the active memristors in the crossbar. The value of A_{ctrl} is provided by Cadence RTL compiler.

The delay of a single adder (D_{adder}) is expressed in Eq.11.

$$\begin{cases} D_{\text{adder}} &= N_{\text{step}} \cdot D_{\text{step}} \\ D_{\text{step}} &= D_{\text{xbar}} + D_{\text{ctrl}} \end{cases} \tag{11}$$

where D_{adder} is the product of the execution step number N_{step} and the delay of a single step D_{step}. D_{step} is the sum of the crossbar delay D_{xbar} and that of the CMOS controller D_{ctrl}.

The value of D_{ctrl} is provided by Cadence RTL compiler. On the other hand, D_{xbar} consists of the memristor switching time

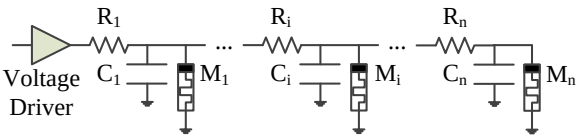

Fig. 14: Elmore RC Delay Model.

T_{sw} and the RC delay due to signal propagation through the nanowires D_{nw}; expressed in Eq.12.

$$\begin{cases} D_{\text{xbar}} &= T_{\text{sw}} + D_{\text{nw}} \\ D_{\text{nw}} &= \sum_{i=1}^{n} \left[C_i(\sum_{j=i}^{n} R_j)\right] \\ &= (n^2 + 4n - 21/8) R_{\text{nw}} \cdot C_{\text{nw}} \cdot F^2 \text{ [55]} \\ n &= \max\{N_{\text{R}}, N_{\text{C}}\} \\ &\text{where} \\ R_1 &= \frac{3}{2}F \cdot R_{\text{nw}} \\ R_i &= 2F \cdot R_{\text{nw}}, \quad 1 < i \leq n \\ C_1 &= \frac{3}{2}F \cdot \frac{C_{\text{nw}}}{2} \\ C_i &= 2F \cdot C_{\text{nw}}, \quad 1 < i < n \\ C_n &= 2F \cdot C_{\text{nw}} + \frac{3}{2}F \cdot C_{\text{nw}} \end{cases} \tag{12}$$

where D_{nw} equals to the time to propagate the signal from the voltage driver to the nth memristor; it is given by Elmore model as shown in Eq.11 [53,55]. Fig. 14 shows the equivalent circuit to model the RC delay in a row or column of a crossbar. As the resistance value of the memristor device in its ON as well as in its OFF state is order of magnitudes higher than the nanowire resistance, the impact of memristor resistance in the modelled delay by Eq.11 is neglected [55]. Note that R_{nw} and C_{nw} denote the resistance and capacitance of nanowire in unit length (see Table IV).

The power consumed by a single adder P_{adder} is expressed by Eq.13, and is the sum of the power consumed by crossbar (P_{xbar}), by the CMOS voltage drivers ($P_{\text{vd,all}}$) and by the controller (P_{ctrl}) for all steps N_{step} to be executed.

$$\begin{cases} P_{\text{adder}} &= \sum_{n=1}^{N_{\text{step}}} (P_{\text{xbar}} + P_{\text{vd,all}} + P_{\text{ctrl}})_n \\ P_{\text{xbar}} &= \sum_{\text{all devices}} P_{\text{R}} \\ &= \sum_{\text{all devices}} \frac{V_{\text{R}}^2}{R} \end{cases} \tag{13}$$

For each execution step, P_{xbar} is the total power consumed by all the devices within the crossbar, which consists of the active and disabled memristors, and the resistors R_s. P_{R}=$\frac{V_{\text{R}}^2}{R}$ is the power consumed by each device, where V_{R} is the voltage across the device and R is its resistance, and they are both obtained using SPICE simulations. $P_{\text{vd,all}}$ is the power consumed by all the voltage drivers; we assume that the voltage drivers are almost ideal voltage sources and their power consumption is very small as compared with that consumed by the crossbar (which constitutes the load of the voltage drivers). The value of P_{ctrl} is provided by Cadence RTL compiler. To evaluate the power of each design, we first estimate the power for each input combination (2^8 in total), and thereafter calculate the average power consumption.

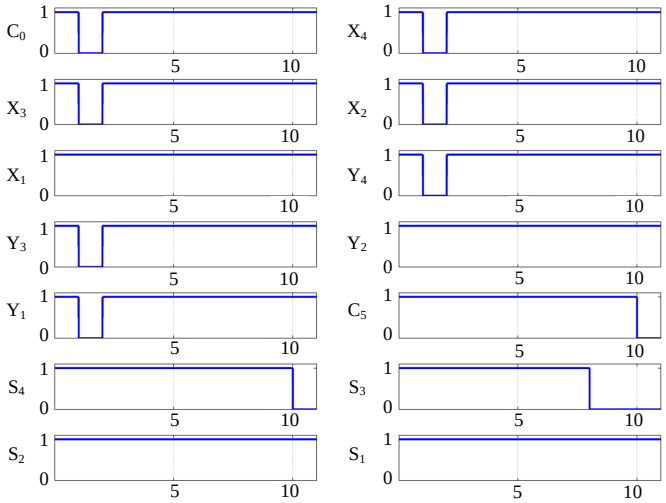

Fig. 15: Waveform of OD Adder SPICE Simulation.

B. Simulation Results

In the first experiment, all the four selected designs are exhaustively verified using SPICE simulations for all the possible input combinations. The simulation results have validated the correctness of the approach. For instance, Fig. 15 shows the waveform of a SPICE simulation of the OD design when input X='0001', Y='0010' and C_0=0. After eleven execution steps, the final results are C_5=0 and S='0011'. It is worth to note that the input data of the input latch are destructed after they are copied to the computing elements [29]; this has no impact on the correctness of the circuit as the inputs have been stored in minterms of the computing elements to calculate the final results.

In the second experiment, the performance of all the four selected designs is estimated in terms of area, delay and power consumption; the results are discussed next.

Area: Fig. 16(a) shows the area of the selected designs. Among all the designs, OI design uses the smallest crossbar as its FAs are adjacently placed. OD design requires the smallest CMOS part as its state machine is the simplest, and consumes the least overall area. The optimized designs require (up to 55%) less area than initial designs as their controllers have less states and less output control signals. Note that for these small designs, the CMOS area dominates the overall area. To further explore the scalability of the designs and their impact on area, we estimate the area of n-bit adders based on OD design (n=2^k, 2≤k≤7). Fig. 16(b) shows the area ratio of the crossbar over the CMOS part (including both the controller and voltage drivers). Clearly for larger adders (in this case for k>6), the crossbar area surpasses that of CMOS part.

Delay: Fig. 17(a) shows the required number of execution steps for each of the four designs, while Fig. 17(b) shows the corresponding delay for a single execution step D_{step} for each design and its breakdown; see also Eq.9. Obviously, the optimized designs have lower execution time and OD design performs by far the best. As each design is based on the same memristor crossbar technology, each design has the same

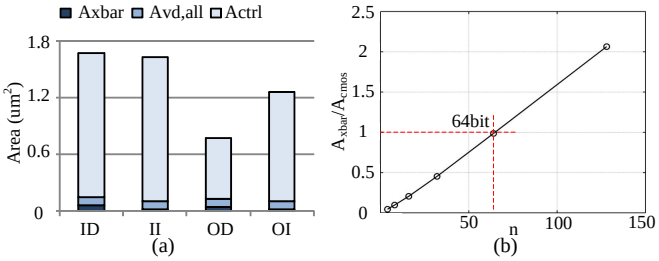

Fig. 16: Area of Selected Designs: (a) Area, (b) Scalability Exploration.

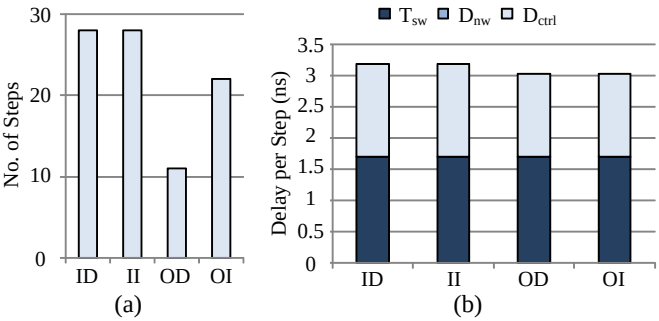

Fig. 17: Delay of Selected Designs: (a) Number of Execution Steps, (b) Delay per Execution Step.

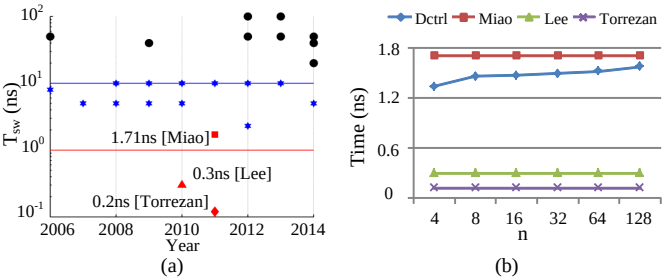

Fig. 18: Delay Scalability Exploration: (a) Tsw for Memristor Technology, (b) Dctrl and Tsw for n-Bit OD Designs.

memristor switching time T_{sw}=1.71 ns. The interconnect delay D_{nw} is negligible for all designs as compared with other delays. Moreover, CMOS controller delay D_{ctrl} for optimized designs is about 11% less than that of the initial designs.

The memristor technology is not mature yet and its switching time is still being improved. Fig. 18(a) shows the switching time of some reported resistive devices over the years [44,49,56–58], while Fig. 18(b) selects three reported T_{sw} and compares them with the controller delay for n-bit adders based on OD design (n=2^k, 2≤k≤7). Clearly, the CMOS delay may become the major critical component with respect to the performance of crossbar based logic designs. Nevertheless, the potential of the crossbar is enabling the massive parallelism, (where the same CMOS circuit may control different parallel designs within crossbar) and reducing the overall delay-operation of the whole design; hence this increases the overall throughput.

Power: Fig. 19 (a) shows the power consumption P_{adder} of a single adder and its breakdown for R_{L}=200kΩ [49], where

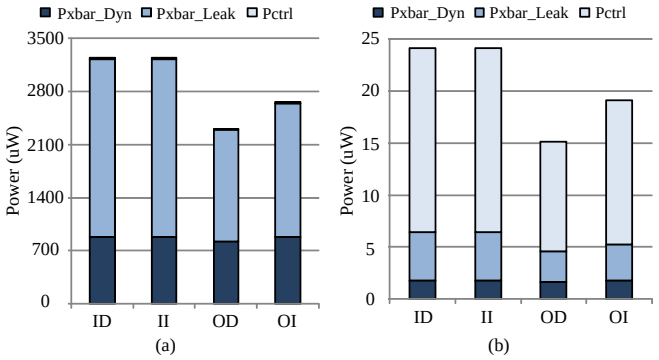

Fig. 19: Power Consumption of Selected Designs Using Different R_{L}: (a) R_{L}=200kΩ, (b) R_{L}=100MΩ.

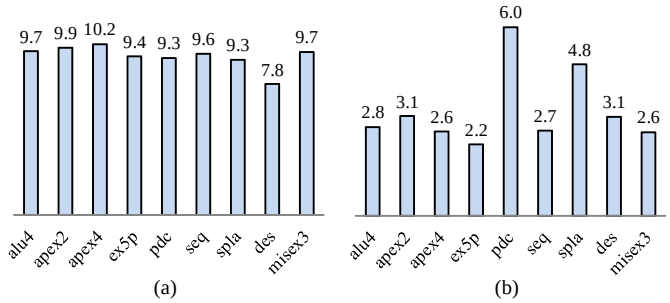

Fig. 20: Improvement Ratios for Different Benchmarks: (a) Area, (b) Delay.

R_{L} is the memristor resistance in its ON state; see also Eq.13. The power consumption of the crossbar P_{xbar} consists of the dynamic part ($P_{\mathrm{xbar,dyn}}$) as a result of resistive switching, and the leakage part ($P_{\mathrm{xbar,leak}}$) induced by sneak path currents. The difference between the dynamic power consumption of the four designs is marginal. However, the crossbar leakage is at least twice higher than the crossbar dynamic power. This highlights one of the major challenges the crossbar designs is facing, namely sneak path currents [52]; and shows the need of providing solutions. Note that the optimized designs consume about 30% less power than the initial designs, as they require less crossbar area. Fig. 19 also shows that the power consumption of the CMOS part P_{ctrl} is negligible.

One possible solution to reduce the power consumption in the crossbar (especially the part caused by sneak path currents) is to increase the value of R_{L}. For example, Fig. 19(b) shows the power consumption P_{adder} and its breakdown when R_{L} is increased to 100MΩ [59]. As a result, both the dynamic and leakage power consumption of the crossbar are significantly reduced, and the power consumed by CMOS controller becomes dominant now. Note that the optimized designs consume approximately 20% less power than the initial designs, as they have simpler CMOS controllers. Another possible solution to reduce the leakage power is reducing the duration of control voltages.

The results show systematically that the OD is the best design with respect to the three considered metrics.

In the third experiment, the performance of the nine benchmarks in Table III is estimated in terms of area and delay. Fig. 20 (a) and (b) shows the improvement ratios realized by optimized designs. As compared to the initial designs, the area of optimized designs is 7.8X to 10.2X smaller, and the delay is 2.2X to 6.0X shorter (due to a reduced number of execution steps).

Overall, the methodology can be used to not only map logic design on the crossbar, but also to evaluate its performance while considering optimization schemes. The results also show that the optimization techniques can significantly improve the performance of designs in terms of area and delay when the logic circuits become larger and more complex.

VI. Conclusion

This paper has proposed a mapping methodology of Boolean logic circuits on memristor crossbar as well as several optimization schemes. The performance of mapped logic circuits can be evaluated including both memristor and CMOS parts.

The proposed methodology provides the following advantages.

- **Generality**: The proposed methodology is potential to map arbitrary logic circuits as long as they are based on Boolean logic, such as adders, ALUs, data encryption (see also Table III).
- **Scalability**: The proposed methodology is scalable to map logic circuits as large as possible, but it should also consider the technology restrictions (e.g., sneak path current issues [43,44]) for appropriate functionality.
- **Automation**: The proposed methodology provides a potential to automate the mapping of large-scale logic circuits on memristor crossbar, which can be incorporated with existing logic synthesis tools (e.g., ODIN II [60] and ABC [61]).
- **Evaluation**: The proposed methodology provides performance evaluation for both the crossbar and CMOS part in terms of area, delay and power consumption, which can be used to compare between different designs based on resistive Boolean logic.
- **Modularity**: The proposed methodology uses a modular approach; this facilitates the improvement of the approach if need. E.g., in Fig. 5, the block 'implement functions by CEs' can be updated without the need to touch any other blocks in the flow.

In order to improve the logic based on memristor crossbar and related tools for mapping, more efforts should be paid to address the following challenges in the future work.

- **Support Other Logic Types**: As our method is modular, it is possible to be tuned to support other logic styles such as logic circuits proposed in [26,62–64].
- **Innovative Logic Design Styles**: As memristor logic circuit is still in infancy stage, innovative logic circuits based on resistive switching should be invented to maximize the potential of memristor crossbar. For instance, memristor crossbar may be suitable for analog circuits as a single memristor can represent a multi-level value, instead of a binary value [38].

- **Impact of Unreliable Technology**: Memristor technology is still under development, and therefore logic circuits based on crossbar are facing reliability challenges due to limited device endurance, device-to-device variation, cycle-to-cycle variation [44,65,66].
- **Sneak Path Issues**: The crossbar-based logic may fail due to the sneak path currents [34,35], which are the unexpected currents within the crossbar [43,44]. Even though several approaches have been proposed such as adding selector devices (e.g., CMOS transistor) [43,66], using complementary resistive switches [35], applying half-select voltages [34,38], how to efficiently apply these techniques to large-scale circuit is still under research. In addition, based on our simulations, when the ON/OFF current ratio increases, the impact of sneak path currents reduces.
- **Implementation Consideration**: Some effects should be considered in the future implementation. First, parasitic nanowire resistance causes the IR-drop. It can be solved by slightly increasing the control voltage V_w and V_h (e.g., 10%). In addition, increasing the low resistance is also helpful to reduce the impact of parasitic nanowire resistance. Second, the needed resistance ratio $\frac{R_H}{R_L}$ is typically 1000 to 10000. Third, the area of voltage driver is related to the placement of memristors within the crossbar. This may impact the scalability of the design. A possible solution is balancing the voltage drivers in the CMOS layer. Fourth, to isolate the crossbars, the nanowires should be broken and isolated materials (e.g., SiO_2) should be inserted. Some other possible solutions are reported in [67].
- **Complexity of CMOS Controller**: As the logic circuits based on memristor crossbar is scaling up, the complexity of CMOS controller is also increasing in order to compensate the driving force and support more execution states of the FSM. Therefore, it is crucial to design efficient CMOS control circuity in terms of area and delay being able to drive appropriate number of logic blocks in the crossbar. Some approaches may be helpful, such as sharing the CMOS controller between different logic circuits, pipelining the computing elements to simplify the controller, etc. In addition, a simpler controller is likely to have a shorter delay. Therefore, it is possible to reduce the power consumption due to a shorter duration of applied control voltages.

Overall, the proposed mapping methodology sets a step towards the implementation of large-scale resistive computing architectures, and provides an opportunity to examine the potential of memristor crossbar architectures.

REFERENCES

[1] B. Hoefflinger, *Chips 2020: a guide to the future of nanoelectronics*. Springer Science & Business Media, 2012.

[2] S. Borkar, "Design perspectives on 22nm cmos and beyond," in *Proceedings of the 46th Annual Design Automation Conference (DAC)*. ACM, 2009, pp. 93–94.

[3] S. Hamdioui *et al.*, "Reliability challenges of real-time systems in forthcoming technology nodes," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. EDA Consortium, 2013, pp. 129–134.

[4] J. L. Hennessy *et al.*, *Computer architecture: a quantitative approach*. Elsevier, 2011.

[5] F. J. Pollack, "New microarchitecture challenges in the coming generations of cmos process technologies (keynote address)," in *Proceedings of the 32nd annual ACM/IEEE international symposium on Microarchitecture (MICRO)*. IEEE, 1999, p. 2.

[6] D. E. Nikonov *et al.*, "Overview of beyond-cmos devices and a uniform methodology for their benchmarking," *Proceedings of the IEEE*, vol. 101, pp. 2498–2533, 2013.

[7] A. Bachtold *et al.*, "Logic circuits with carbon nanotube transistors," *Science*, vol. 294, pp. 1317–1320, 2001.

[8] L. Ding *et al.*, "Cmos-based carbon nanotube pass-transistor logic integrated circuits," *Nature communications*, vol. 3, p. 677, 2012.

[9] L. Amarú *et al.*, "New logic synthesis as nanotechnology enabler," *Proceedings of the IEEE*, vol. 103, pp. 2168–2195, 2015.

[10] B. Behin-Aein *et al.*, "Proposal for an all-spin logic device with built-in memory," *Nature nanotechnology*, vol. 5, pp. 266–270, 2010.

[11] B. Behin-Aein *et al.*, "All-spin logic," in *Device Research Conference (DRC), 2010*. IEEE, 2010, pp. 41–42.

[12] M. Fuhrer *et al.*, "Spintronics: A path to spin logic," *Nature physics*, vol. 1, pp. 85–86, 2005.

[13] L. O. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, pp. 507–519, 1971.

[14] D. B. Strukov *et al.*, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, 2008.

[15] ITRS, "Beyond cmos white paper," 2014.

[16] W. Zhao *et al.*, "Nanodevice-based novel computing paradigms and the neuromorphic approach," in *Circuits and Systems (ISCAS), IEEE International Symposium on*. IEEE, 2012, pp. 2509–2512.

[17] S. Hamdioui *et al.*, "Memristor based computation-in-memory architecture for data-intensive applications," in *Proceedings of the Design, Automation & Test in Europe (DATE)*. EDA Consortium, 2015, pp. 1718–1725.

[18] S. Hamdioui *et al.*, "Memristor for computing: Myth or reality?" in *Proceedings of the Design, Automation & Test in Europe (DATE)*. EDA Consortium, 2017, pp. 1729–1725.

[19] H. A. Du Nguyen *et al.*, "Computation-in-memory based parallel adder," in *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH 15)*. IEEE, 2015, pp. 57–62.

[20] A. Haron *et al.*, "Parallel matrix multiplication on memristor-based computation-in-memory architecture," in *High Performance Computing & Simulation (HPCS), 2016 International Conference on*. IEEE, 2016, pp. 759–766.

[21] J. Yu *et al.*, "Skeleton-based design and simulation flow for computation-in-memory architectures," in *Nanoscale Architectures (NANOARCH), 2016 IEEE/ACM International Symposium on*. IEEE, 2016, pp. 165–170.

[22] L. Yavits *et al.*, "Resistive associative processor," *IEEE Computer Architecture Letters*, vol. 14, pp. 148–151, 2015.

[23] S. Li *et al.*, "Pinatubo: a processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 173.

[24] L. Gao *et al.*, "Programmable cmos/memristor threshold logic," *IEEE Transactions on Nanotechnology*, vol. 12, pp. 115–119, 2013.

[25] G. S. Rose *et al.*, "Leveraging memristive systems in the construction of digital logic circuits," *Proceedings of the IEEE*, vol. 100, pp. 2033–2049, 2012.

[26] J. Borghetti *et al.*, "Memristive switches enable stateful logic operations via material implication," *Nature*, vol. 464, pp. 873–876, 2010.

[27] E. Lehtonen *et al.*, "Memristive stateful logic," in *Memristor Networks*. Springer, 2014, pp. 603–623.

[28] G. Snider, "Computing with hysteretic resistor crossbars," *Applied Physics A*, vol. 80, pp. 1165–1172, 2005.

[29] L. Xie *et al.*, "Fast boolean logic mapped on memristor crossbar," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, 2015, pp. 335–342.

[30] Intel xpoint memory. [Online]. Available: http://www.intel.com/content/www/us/en/architecture-and-technology/non-volatile-memory.html

[31] Crossbar 3d rram. [Online]. Available: http://www.crossbar-inc.com/

[32] A. Raghuvanshi *et al.*, "Logic synthesis and a generalized notation for memristor-realized material implication gates," in *Proceedings of the 2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2014, pp. 470–477.

[33] F. S. Marranghello *et al.*, "Sop based logic synthesis for memristive imply stateful logic," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, 2015, pp. 228–235.
[34] S. Kvatinsky *et al.*, "Memristor-based material implication (imply) logic: design principles and methodologies," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, pp. 2054–2066, 2014.
[35] A. Siemon *et al.*, "A complementary resistive switch-based crossbar array adder," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 5, pp. 64–74, 2015.
[36] S. Kvatinsky *et al.*, "Team: Threshold adaptive memristor model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, pp. 211–221, 2013.
[37] X. Guan *et al.*, "A spice compact model of metal oxide resistive switching memory with variations," *IEEE electron device letters*, vol. 33, pp. 1405–1407, 2012.
[38] K.-H. Kim *et al.*, "A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications," *Nano letters*, vol. 12, pp. 389–395, 2011.
[39] X. Zhu *et al.*, "Performing stateful logic on memristor memory," *Circuits and Systems II: Express Briefs, IEEE Transactions on*, vol. 60, pp. 682–686, 2013.
[40] B. Parhami, *Computer arithmetic*. Oxford university press, 1999, vol. 20, no. 00.
[41] P. C. McGeer *et al.*, "Espresso-signature: A new exact minimizer for logic functions," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, pp. 432–440, 1993.
[42] L. Xie *et al.*, "Interconnect networks for memristor crossbar," in *Nanoscale Architectures (NANOARCH), 2015 IEEE/ACM International Symposium on*. IEEE, 2015, pp. 124–129.
[43] S. Hamdioui *et al.*, "Memristor based memories: Technology, design and test," in *Design & Technology of Integrated Systems In Nanoscale Era (DTIS), 9th IEEE International Conference On*. IEEE, 2014, pp. 1–7.
[44] J. J. Yang *et al.*, "Memristive devices for computing," *Nature nanotechnology*, vol. 8, pp. 13–24, 2013.
[45] X. Tang *et al.*, "A high-performance low-power near-vt rram-based fpga," in *Field-Programmable Technology (FPT), 2014 International Conference on*. IEEE, 2014, pp. 207–214.
[46] P.-E. Gaillardon *et al.*, "Design and architectural assessment of 3-d resistive memory technologies in fpgas," *IEEE Transactions on Nanotechnology*, vol. 12, pp. 40–50, 2013.
[47] M. Mao *et al.*, "Optimizing latency, energy, and reliability of 1t1r reram through appropriate voltage settings," in *Computer Design (ICCD), 2015 33rd IEEE International Conference on*. IEEE, 2015, pp. 359–366.
[48] J. M. Rabaey *et al.*, *Digital integrated circuits*. Prentice hall Englewood Cliffs, 2002, vol. 2.
[49] F. Miao *et al.*, "Anatomy of a nanoscale conduction channel reveals the mechanism of a high-performance memristor," *Advanced materials*, vol. 23, pp. 5633–5640, 2011.
[50] "Fpga place-and-route challenge." [Online]. Available: http://www.eecg.toronto.edu/ vaughn/challenge/challenge.html
[51] M. Zangeneh *et al.*, "Performance and energy models for memristor-based 1t1r rram cell," in *Proceedings of the great lakes symposium on VLSI*. ACM, 2012, pp. 9–14.
[52] J. J. Yang *et al.*, "The mechanism of electroforming of metal oxide memristive switches," *Nanotechnology*, vol. 20, p. 215201, 2009.
[53] D. B. Strukov *et al.*, "Cmol fpga: a reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices," *Nanotechnology*, vol. 16, p. 888, 2005.
[54] L. Xie *et al.*, "Boolean logic gate exploration for memristor crossbar," in *Design and Technology of Integrated Systems in Nanoscale Era (DTIS), 2016 International Conference on*. IEEE, 2016, pp. 1–6.
[55] W. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *Journal of applied physics*, vol. 19, pp. 55–63, 1948.
[56] M.-J. Lee *et al.*, "A fast, high-endurance and scalable non-volatile memory device made from asymmetric ta2o5- x/tao2- x bilayer structures," *Nature materials*, vol. 10, pp. 625–630, 2011.
[57] A. C. Torrezan *et al.*, "Sub-nanosecond switching of a tantalum oxide memristor," *Nanotechnology*, vol. 22, p. 485203, 2011.
[58] Crossbar 3d rram. [Online]. Available: https://nano.stanford.edu/stanford-memory-trends
[59] C. Schindler *et al.*, "Electrode kinetics of cu-sio2-based resistive switching cells: Overcoming the voltage-time dilemma of electrochemical metallization memories," *Applied physics letters*, vol. 94, p. 2109, 2009.
[60] P. Jamieson *et al.*, "Odin ii-an open-source verilog hdl synthesis tool for cad research," in *Field-Programmable Custom Computing Machines (FCCM), 2010 18th IEEE Annual International Symposium on*. IEEE, 2010, pp. 149–156.
[61] Abc: A system for sequential synthesis and verification. [Online]. Available: https://people.eecs.berkeley.edu/ alanmi/abc/abc.htm
[62] E. Linn *et al.*, "Beyond von neumann? logic operations in passive crossbar arrays alongside memory operations," *Nanotechnology*, vol. 23, p. 305205, 2012.
[63] S. Kvatinsky *et al.*, "Mrlmemristor ratioed logic," in *2012 13th International Workshop on Cellular Nanoscale Networks and their Applications*. IEEE, 2012, pp. 1–6.
[64] S. Kvatinsky *et al.*, "Magicmemristor-aided logic," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 61, pp. 895–899, 2014.
[65] H.-S. P. Wong *et al.*, "Metal–oxide rram," *Proceedings of the IEEE*, vol. 100, pp. 1951–1970, 2012.
[66] R. Waser *et al.*, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Advanced Materials*, vol. 21, pp. 2632–2663, 2009.
[67] I. Vourkas *et al.*, "Alternative architectures toward reliable memristive crossbar memories," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, pp. 206–217, 2016.

Lei Xie (S'15) received his Bachelors and Masters degree in Microelectronics from Xian Jiaotong University, Xian, China. Currently, he is pursuing a PhD at the Computer Engineering Lab in the Delft University of Technology, Delft, the Netherlands. His research interest is memristor-based logic circuit design.

Hoang Anh Du Nguyen (S'15) received the M.Sc. degrees in computer engineering from the Delft University of Technology, Delft, The Netherlands. She is currently a pursuing the Ph.D. degree with the Computer Engineering Laboratory, Delft University of Technology. Her current research interests include Computation-in-Memory (CIM) architecture for Big-Data, memristor based systems and synthesis automation.

Mottaqiallah Taouil (S'10 – M'15) received the M.Sc. and Ph.D. degrees (both with Hons.) in computer engineering from the Delft University of Technology, Delft, The Netherlands. He is currently a Post-Doctoral Researcher with the Dependable Nano-Computing Group, Delft University of Technology. His current research interests include reconfigurable computing, embedded systems, very large scale integration design and test, built-in-self-test, and 3-D stacked integrated circuits, architectures, design for testability, yield analysis, and memory test structures.

Said Hamdioui (M'99 – SM'11) is currently a Chair Professor on Dependable and Emerging Computer Technologies at the Computer Engineering Laboratory of the Delft University of Technology (TUDelft), the Netherlands. Prior to joining TUDelft, Hamdioui worked for Intel Corporation (Californina, USA), Philips Semiconductors R&D (Crolles, France) and for Philips/ NXP Semiconductors (Nijmegen, The Netherlands). His research focuses on two domains: Dependable CMOS nano-computing (including Reliability, Testability, Hardware Security) and emerging technologies and computing paradigms (including 3D stacked ICs, memristors for logic and storage, in-memory-computing). He owns one patent and has published one book and co-authored over 170 conference and journal papers. He delivered dozens of keynote speeches, distinguished lectures, and invited presentations and tutorial at major international forums/conferences/schools and at leading semiconductor companies. Hamdioui is a Senior member of the IEEE, Associate Editor of IEEE Transactions on VLSI Systems (TVLSI), and he serves on the editorial board of IEEE Design & Test, and of the Journal of Electronic Testing: Theory and Applications (JETTA). He is also member of AENEAS/ENIAC Scientific Committee Council (AENEAS =Association for European NanoElectronics Activities).

Koen Bertels is Professor and Head of the Computer Engineering Laboratory at Delft University of Technology. His research focuses on heterogeneous multicore computing, investigating topics ranging from compiler technology, runtime support and architecture. He recently started working on quantum computing as a principal investigator in the Qutech research center. He served as general and program chair for various conferences such as FPL, RAW, ARC. He co-authored more than 30 journal papers and more than 150 conference papers.