Fast Boolean Logic Mapped 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*—As the CMOS technology is gradually scaling down to inherent physical device limits, significant challenges emerge related to scalability, leakage, reliability, etc. Alternative technologies are under research for next-generation VLSI circuits. Memristor is one of the promising candidates due to its scalability, practically zero leakage, non-volatility, etc. This paper proposes a novel design methodology for logic circuits targeting memristor crossbars. This methodology allows the optimization of the design of logic function, and their automatic mapping on the memristor crossbar. More important, this methodology supports the execution of Boolean logic functions within constant number of steps independent of its functionality. To illustrate the potential of the proposed methodology, multi-bit adders and multipliers are explored; their incurred delay, area and energy costs are analyzed. The comparison of our approach with state-of-the-art Boolean logic circuits for memristor crossbar architecture shows significant improvement in both delay (4 to $500\times$) and energy consumption (1.22 to $3.71\times$). The area overhead may decrease (down to 44%) or increase (up to 17%) depending on the circuit's functionality and logic optimization level.**

I. Introduction

As CMOS transistors gradually scale down to the inherent physical device limits, CMOS technology faces major challenges [1–4] such as increased leakage power consumption, saturated performance improvement, reduced reliability, and a more complicated fabrication process. To address these challenges, novel technologies (e.g., memristors [5,6], nanotube [7], graphene [8] and tunnel field-effect transistors [9], etc.) are proposed as the alternative for next-generation VLSI circuits. Among these technologies, memristor is a promising candidate [10,11]. Typically, numerous memristors can be mapped on crossbar architecture where memristors are located in the intersections of horizontal and vertical nanowires. Memristor crossbar is able to provide great scalability, higher integration density, zero leakage power consumption, CMOS fabrication compatibility [10,12–14]. Several potential applications have been proposed such as neuromorphic systems [15,16], non-volatile memories [10,17], novel computing paradigms for data-intensive applications [18], etc. To implement novel computing paradigms, it is crucial to design fundamental components such as Boolean logic functions [19].

Research on memristor-based logic circuits has attracted significant attention both in academy and industry since the first memristor device was fabricated by HP in 2008 [6]. Four types of logic circuits have been proposed: threshold [20,21], majority [21], material implication [22,23], and Boolean [24] logic. Threshold and majority logic circuits are based on threshold and majority logic gates, respectively; both of them are not suitable for crossbar arrays. However, both material implication and Boolean logic have been addressed for the memristor crossbar. In [23], the authors proposed a methodology to implement logic functions using a sequence of material implication operations. However, this methodology suffers from low speed and requires new algorithms to implement arithmetic operations such as addition and multiplication [23,25,26]. In [24], the authors proposed a systematic Boolean logic design methodology; its basic unit is a logic block which implements a Boolean function $f=\overline{M_1+\cdots+M_i+\cdots+M_n}=\overline{M_1}\cdot\cdots\cdot\overline{M_i}\cdot\cdots\cdot\overline{M_n}$, where M_i denotes a minterm and n is the total number of minterms. However, all the minterms are executed sequentially. As a consequence, building a complex logic circuit with numerous minterms will result in a very slow design. In addition, as each logic function requires different numbers of execution steps, the synchronization between different logic functions is a complicated task.

To solve aforementioned issues, this paper proposes a novel design methodology for crossbar-based Boolean logic circuits; it is based on parallel execution of all minterms and therefore significantly increases the overall performance. The contributions of this paper are:

- A methodology to design memristor logic circuits able to execute any Boolean function within constant time.
- Several novel primitive operations (implementing logic gates, copy operations, etc.) supporting the proposed design methodology.
- A novel memristor crossbar architecture supporting the proposed design methodology.
- A novel design technique to further optimize the proposed design methodology by sharing minterms.
- A model to evaluate the delay, area and energy.

The remainder of this paper is organized as follows. Section II briefly describes the background and related work. Section III proposes the design methodology for Boolean logic circuits. Section IV presents a one-bit full adder as a design case study. Section V verifies and evaluates the proposed design methodology. Finally, Section VI concludes the paper.

978-1-4673-7166-7/15/$31.00

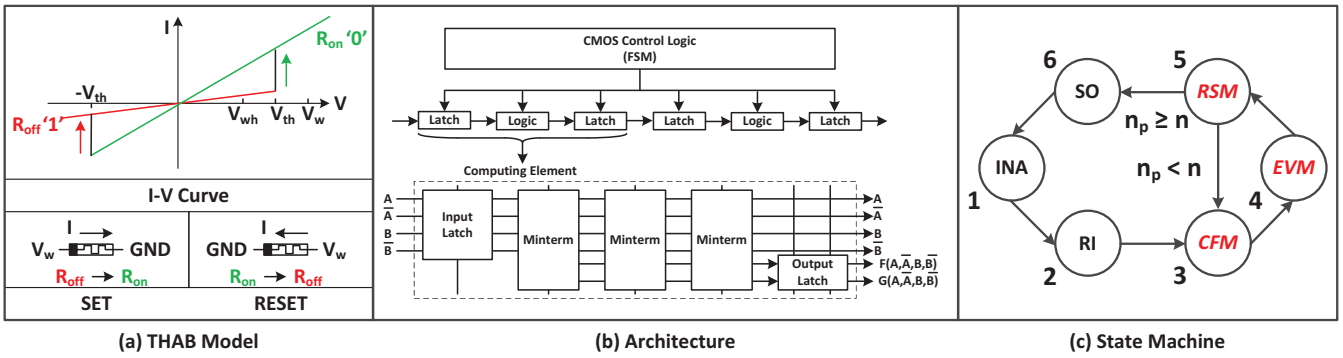

Fig. 1: Snider's Boolean Logic Circuits

II. Background and Related Work

This section describes the memristor model, data representation, control voltages, and the working principle of the previously proposed Boolean logic circuits [24].

A. Memristor Model

Although memristors have different physical mechanisms [12,13,27], their circuit-level behaviour can be abstracted by two parameters: switching threshold voltage V_{th} (with or without a minimum voltage requirement for resistive switching) [28] and resistive switching behaviour (abrupt or smooth) [29]. Note that these two parameters are independent of each other.

In this paper, we use a simplified memristor model based on a switching threshold voltage and abrupt switching [23,24]. For simplicity, this type of model is referred to as THAB (THreshold-ABrupt) memristor model. The top part of Fig. 1(a) illustrates the current-voltage relation of THAB model. The memristor switches from one resistive state to another when the absolute value of the voltage (either positive or negative) across the device is greater than its threshold voltage V_{th}; see I-V curve of Fig. 1(a). Otherwise, it stays in its current resistive state. Normally, a memristor requires two different switching threshold voltages to switch from low to high resistance (RESET) and from high to low resistance (SET) [28,30]; see the bottom part of Fig. 1(a). The black squares represent the positive terminal of the memristor. For simplicity, we assume that THAB model has the same threshold voltage V_{th} (in absolute value) for both of them.

B. Data Representation and Control Voltages

A memristor has two resistive states: a high (R_{off}) and low (R_{on}) resistance. In this paper, all logic circuits use R_{off} and R_{on} for logic 1 and 0, respectively. It is worth to note that this is different from CMOS logic circuits which use high and low voltages to represent logic 1 and 0, respectively.

To control memristor-based digital circuits, three different voltages are required: V_w, V_{wh}, and *GND*; see Fig. 1(a). V_w is used to program the resistance of a memristor; V_{wh} is used to minimize the impact of sneak path currents by half-select voltage strategy [17]; V_{wh} is then applied to memristors which are not involved in particular operations. V_{wh} is also used to support the implementation of basic logic gate as in the case of NAND [24]. The relationship between V_w, V_{wh}, *GND* and V_{th} is $0<V_{wh}=\frac{V_w}{2}<V_{th}<V_w$. This relationship guarantees $V_w-V_{wh}=2V_{wh}-V_{wh}=V_{wh}<V_{th}$ which is able to prevent undesired switching of non-accessed memristors [17].

C. Snider's Boolean Logic Circuits

G. Snider proposed in [24] a design methodology for memristor crossbar to implement Boolean logic. In this paper we denote this design methodology by "SBLC" (Snider's Boolean Logic Circuits). As Fig. 1(b) shows, SBLC is formed by an alternating cascade of latches and logic blocks; synchronized by a CMOS control logic (realized by a FSM).

Each Boolean function is described as the format of $f=\overline{M_1+\cdots+M_i+\cdots+M_n}=\overline{M_1}\cdot\cdots\cdot\overline{M_i}\cdot\cdots\cdot\overline{M_n}$, and is implemented by a basic *computing element (CE)* as shown in the bottom of Fig. 1(b). 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 based on its number of inputs. The OL of a particular CE is connected to the IL of the next CE by a low-resistance path that consists of nanowires and memristors (which serve as configurable routing switches). Each latch is composed of several memristors depending on the number of inputs/outputs of the Boolean function, and is used temporarily to store data. The results of all minterms are accumulated at OL, which operates as an AND gate.

SBLC requires an appropriate control to execute the steps needed to compute a Boolean function. This is captured by the state machine in Fig. 1(c); n_p represents the number of minterms processed at each step, n the total number of minterms. The state machine consists of six states:

1) INA: INItialize All the memristors of a CE to R_{off}.
2) RI: Receive Inputs. The IL of the CE receives the outputs of the previous CE from its OL. The IL of the CE in the first stage receives the inputs from primary inputs; e.g., A.
3) CFM: ConFigure a Minterm. A particular minterm M_i is configured by copying data from IL. Simultaneously,

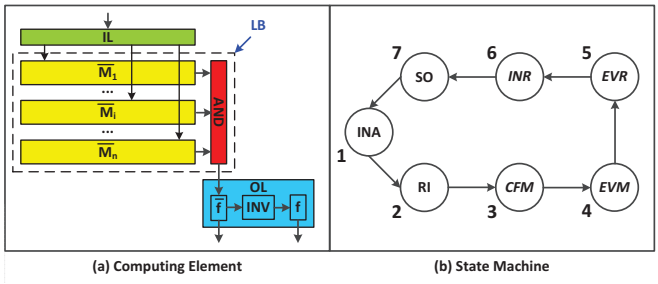

Fig. 2: Fast Boolean Logic Circuits.

the memristor located at the intersection of the column implementing the minterm M_i and row of the output is SET. This forms a low-resistance path between IL, M_i and OL.

4) EVM: EValuate a Minterm. $\overline{M_i}$ is implemented by a NAND gate and its result is accumulated at OL.
5) RSM: ReSet a Minterm. All the memristors of M_i are RESET.
6) SO: Send Outputs. The results of all the outputs stored in OL are sent to IL of the next CE.

Each minterm requires three steps: CFM, EVM and RSM. The CE repeats these three steps until all the minterms are evaluated. In conclusion, a CE with n minterms requires 3n+3 steps to compute the Boolean function. This sequential execution induces slow speed, and requires complex synchronization between multiple LBs.

III. Logic Circuits Design Methodology

This section describes the proposed design methodology, its primitive operations, and how the crossbar architecture should be tuned to make it suitable for the proposed method.

A. Working Procedure and Primitive Operations

The major drawback of SBLC is the sequential execution of minterms; this is due to two reasons. First, the minterms cannot copy data from the input latches simultaneously. Second, all the minterms accumulate their results in the same output latch to produce the final result.

Motivated by these two reasons, we propose fast Boolean logic circuit (FBLC) which requires the Boolean function to be expressed in the sum-of-product format; i.e., $f{=}M_1 + \cdots + M_i + \cdots + M_n{=}\overline{\overline{M_1} \cdot \cdots \cdot \overline{M_i} \cdot \cdots \cdot \overline{M_n}}$. The concept of the proposed method is shown in Fig. 2(a). Different from SBLC, FBLC employs a novel primitive operation that allows all the minterms to copy their data simultaneously from the input latch. In addition, all the $\overline{M_i}$ are evaluated (based on NAND operations), and subsequently the results of $\overline{M_i}$ are processed by an AND operation to obtain the final result; this result is stored in OL. As a result, FBLC can process all the minterms in parallel, and therefore a Boolean function can be calculated in a constant number of steps independent of the circuit's functionality as will be explained later in this section. It is worth to note that FBLC uses both primary and complementary inputs as CEs in the next stage typically require both.

TABLE I: Summary of Primitive Operations

State	Operation	Logic Values		Control & Intermediate Signals			
		Input	Output	V_{ci}	V_{co}	V_x	V_{om}
INA	RESET	-	1	-	-	-	V_w ($>V_{th}$)
RI	SFC	1	1	V_w	*GND*	*GND*	0 ($<V_{th}$)
		0	0			V_w	V_w ($>V_{th}$)
CFM	MFC	1	1	V_w	*GND*	*GND*	0 ($<V_{th}$)
		0	0			V_w	V_w ($>V_{th}$)
EVM	SFNAND	A1	0	V_{wh}	V_w	*GND*	V_w ($>V_{th}$)
	MFNAND	AL0	1			V_{wh}	$V_w{-}V_{wh}$ ($<V_{th}$)
EVR	AND	A1	1	V_w	*GND*	*GND*	0 ($<V_{th}$)
		AL0	0			*GND*	V_w ($>V_{th}$)
INR	INV	1	0	V_{wh}	V_w	*GND*	V_w ($>V_{th}$)
		0	1			V_{wh}	$V_w{-}V_{wh}$ ($<V_{th}$)
SO	SFC	1	1	V_w	*GND*	*GND*	0 ($<V_{th}$)
		0	0			V_w	V_w ($>V_{th}$)

FBLC requires 7 steps to compute a Boolean function independent of its functionality. Fig. 2(b) shows the state machine that generates the control signals for FBLC. The required primitive operations of each state are:

- INA: INitialize All the memristors of a CE to R_{off}. This state requires RESET operation.
- RI: REceive Inputs. The IL of the CE receives the OL of the previous CE. The IL of the first stage CE receives the inputs from primary inputs. Therefore, this state requires single-fan-out copy (SFC) operation.
- CFM: ConFigure all Minterms. All the minterms are configured simultaneously by copying inputs stored in IL to each minterm in parallel. Hence, this state requires multi-fan-out copy (MFC) operation.
- EVM: EValuate all Minterms. All the $\overline{M_i}$ are evaluated simultaneously; again, the implementation is based on an NAND operation. Hence, this state requires NAND operations: a basic single-fan-out NAND (SFNAND) used to evaluate the $\overline{M_i}$, and an advanced multi-fan-out NAND (MFNAND) used to further optimize FBLC; the latter will be discussed in section IV.
- EVR: EVAluate Results. The results of EVM are feed to an AND gate to determine $\overline{f}$ of the Boolean function; see Fig. 2(a). Therefore, this state needs an AND operation.
- INR: INvert Results. The result of EVR need to be inverted to achieve the final result f of the Boolean function. Hence, an inversion (INV) operation is required.
- SO: Send Outputs. Finally the result captured in OL is sent to IL of the next CE. Hence, a SFC is needed.

The left part of Table I summarizes the primitive operations required at each state; the right part of the table presents the implementation of primitive operations which will be discussed in the next subsection. The first column shows the states; the second column the required operations. Column 3 and 4 present the logic values of the inputs and outputs, respectively; the logic value of a single-input operation is either 1 or 0. For multi-input operations, we distinguish between A1 (all inputs are 1) and AL0 (at least one input is 0).

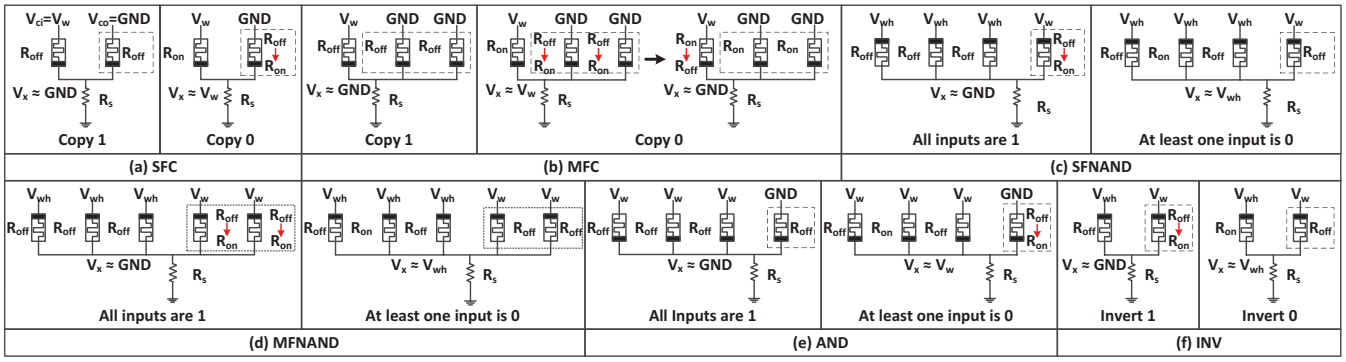

Fig. 3: Realization of Primitive Operations.

B. Implementation of Primitive Operations

Fig. 3 shows the implementation of each primitive operation consisting of one or multiple input and output memristors. The output memristors are surrounded by a box with a dashed-line and are all initialized to R_{off} prior to any operation (i.e., RESET operation of state INA). In addition, the implementation of each operation consists of a resistor R_s, which satisfies the condition $R_{on} \ll R_s \ll R_{off}$; this is required to guarantee that the voltage across the output memristors are close to the desired voltage (e.g., V_w, $V_w - V_{wh}$) for appropriate operations [23].

The right part of Table I shows the control and intermediate signals (for each primitive operation) related to the implementation. Column 'V_{ci}' and 'V_{co}' present the control voltages applied to the input and output memristors, respectively. Column 'V_x' presents the voltage of the floating nanowires for given inputs of a particular operations; it is the voltage across R_s and is used to explain the working principle; see Fig. 3. Column 'V_{om}' shows the voltage across the output memristor after all the control voltages are applied. It indicates whether the output memristor remains R_{off} ($V_{om} < V_{th}$) or switch to R_{on} ($V_{om} > V_{th}$).

The primitive operations are implemented as follows:

- State INA: State INA requires RESET operation; its working principle has been explained in Fig. 1(a).
- State RI: State RI requires SFC operation; its working principle is shown in Fig. 3(a). A voltage $V_w > V_{th}$ (see Fig. 1(a)) is applied to the negative terminal of the input memristor while *GND* is applied to the output memristor. In case the input is 1 (R_{off}), $V_x \approx 0$ as $R_s \ll R_{off}$ (see Fig. 3(a)-Copy 1). Therefore, $V_{om} = V_x - 0 \approx 0 < V_{th}$. As a result, the output memristor stays at R_{off}. In case the input is 0 (R_{on}), $V_x \approx V_w$ as $R_{on} \ll R_s$. Therefore, $V_{om} = V_x - 0 > V_{th}$. As a result, the output memristor switches to R_{on} (see Fig. 3(b)-Copy 0).
- State CFM: State CFM requires MFC operation; its working principle is shown in Fig. 3(b) where a MFC operation with fan-out of two is given as an example. V_w is applied to the negative terminal of the input memristor while *GND* is applied to all the output memristors. In case the input is 1, $V_x \approx 0$ as $R_s \ll R_{off}$ (see Fig. 3(b)-Copy 1). Therefore, $V_{om} = V_x - 0 \approx 0 < V_{th}$. As a result, the output memristors stay at R_{off}. In case the input is 0, $V_x \approx V_w$ as $R_{on} \ll R_s$ (see the left part of Fig. 3(b)-Copy 0). Therefore, $V_{om} = V_x - 0 \approx V_w > V_{th}$. As a result, the output memristors switch to R_{on}. However, this operation is *destructive* as shown in the right part of Fig. 3(b)-Copy 0. After output memristors switch to R_{on}, V_x changes from V_w to 0. Therefore, the voltage across the input memristor becomes $V_x - V_w \approx -V_w < -V_{th}$. As a result, the input memristor switches to R_{off}. Nevertheless, this destructive operation has no impact on the correctness of the CE as the data stored in IL is correctly copied to all minterms.
- State EVM: State EVM requires SFNAND operation; its working principle is shown in Fig. 3(c). A voltage $V_{wh} < V_{th}$ (see Fig. 1(a)) is applied to the positive terminal of the input memristors while V_w is applied to the output memristor. In case all the input are 1, $V_x \approx 0$ (see Fig. 3(c)) and $V_{om} = V_w - V_x \approx V_w > V_{th}$. As a result, the output memristor switches to R_{on}. In case at least one input is 0, $V_x \approx V_{wh}$ and $V_{om} = V_w - V_x \approx V_w - V_{wh} < V_{th}$. As a result, the output memristor stays at R_{off}.
- State EVR: State EVR requires AND operation; its working principle is shown in Fig. 3(e). V_w is applied to the negative terminal of the input memristors while *GND* is applied to the output memristor. In case all the inputs are 1, $V_x \approx 0$ and $V_{om} = V_x - 0 \approx 0 < V_{th}$. As a result, the output memristor stays R_{off}. In case at least one input is 0, V_x is around V_w and $V_{om} = V_x - 0 \approx V_w > V_{th}$. As a result, the output memristor switches to R_{on}.
- State INR: State INR requires INV operation; its working principle is shown in Fig. 3(f). V_{wh} is applied to the positive terminal of the input memristor while V_w is applied to the output memristor. In case the input is 1, $V_x \approx 0$ as $R_s \ll R_{off}$ (see Fig. 3(f)-Invert 1). Therefore, $V_{om} = V_w - V_x \approx V_w > V_{th}$. As a result, the output memristor switches to R_{on}. In case the input is 0, $V_x \approx V_{wh}$ as $R_{on} \ll R_s$ (see Fig. 3(f)-Invert 0). Therefore, $V_{om} = V_w - V_x \approx V_w - V_{wh} < V_{th}$. As a result, the output memristor stays at R_{off}.
- State SO: State SO requires the same SFC operation the one used in state RI.

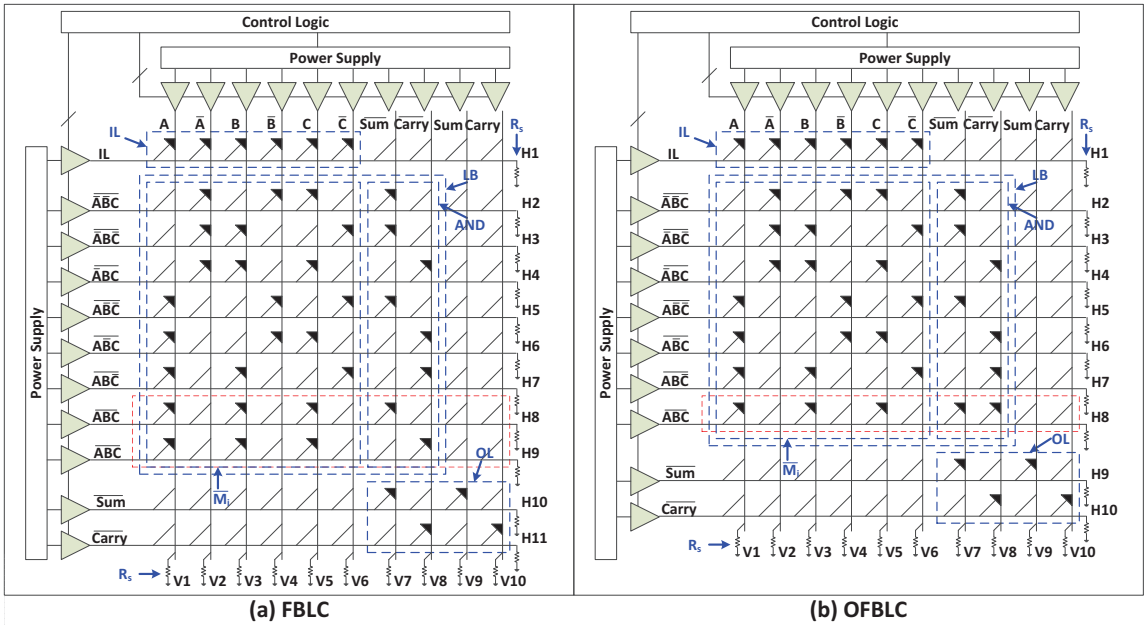

Fig. 4: Implementation of One-Bit Full Adder

C. Computable Memristor Crossbar Architecture

FBLC requires an appropriate memristor crossbar architecture that supports the implementation of all primitive operations. Fig. 4 shows the computable memristor crossbar architecture (CMCA); it is a variant of the normal crossbar architecture [17] with two new features. First, a reference resistor (R_s) is attached to each horizontal and vertical nanowire to implement primitive operations. Second, two types of memristors are required: active memristors which can switch, and disabled memristors which are permanently in the high resistance state [24]. In the figure, the arrow head presents the positive terminal of the active memristor. Disabled memristors are represented by a diagonal line only.

A CMCA consists of a memristor crossbar, CMOS control logic, voltage drivers and a power supply. The memristor crossbar implements a CE including IL, LB and OL. For instance, the crossbar of Fig. 4 implements a 1-bit full adder; row H1 implements IL, row H2 to H9 implement LB (i.e., minterms) and row H10 and H11 implement OL. More details will be explained in the next section. The CMOS circuit consists of voltage drivers, control logic (see Fig. 2(b)), and power supply. Each nanowire is driven by a voltage driver. The voltage drivers deliver different control voltages (i.e., V_w, V_{wh} and *GND*) depending on the operation to be performed. In addition, it can act as a high-impedance state resulting in floating nanowires. The voltage driver can be implemented by three transmission gates [17].

IV. Design Case Study and Optimized FBLC

This section presents a one-bit full adder based on FBLC as a case study. Subsequently, it presents how to further optimize FBLC and a generic mapping process that maps arbitrary Boolean functions to the crossbar.

A. Case Study: One-Bit Full Adder

The sum and carry of a one-bit full adder (FA) can be expressed by Eq. 1. Each expression consists of four minterms. The FBLC implementation of this 1-bit FA can be mapped to CMCA as shown in Fig. 4 (a). For convenience, H# and V# are used to denote a horizontal and vertical nanowire, respectively. We present the *position* of a memristor by P(H#,V#). This 1-bit FA can be implemented using a CE consisting of an IL, LB and OL. The IL is mapped on the memristors located at P(H1,V1-V6) as IL consists of primary inputs and their complements. The remaining memristors on H1 are disabled. The LB consisting of eight minterms is mapped on H2-H9. Each minterm is implemented by placing active memritors at junctions/positions 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, $\bar{A}\bar{B}C$ on H2 is a minterm of Sum. Therefore, the junctions on H2 at P(H2,V2=$\bar{A}$), P(H2,V4=$\bar{B}$), P(H2,V5=C) as well as at P(H2,V7=$\overline{Sum}$) consist of active memristors; while the remaining junctions on H2 contain disabled memristors (non-active). The four minterms of Sum and those of $Carry$ (see Eq. 1) are then ANDed in parallel by column V7=$\overline{Sum}$ and V8=$\overline{Carry}$, respectively. The OL is realized by H10 and H11. The results provided by the two ANDs are then stored at P(H10=$\overline{Sum}$, V7) and P(H11=$\overline{Carry}$, V8), which are thereafter inverted and stored at P(H10, V9=Sum) and P(H11, V10=$Carry$), respectively.

$$\begin{aligned} Sum &= \overline{\bar{A}\bar{B}C} \cdot \overline{\bar{A}B\bar{C}} \cdot \overline{A\bar{B}\bar{C}} \cdot \overline{ABC} \\ Carry &= \overline{\bar{A}BC} \cdot \overline{A\bar{B}C} \cdot \overline{AB\bar{C}} \cdot \overline{ABC} \end{aligned} \quad (1)$$

To control the CE, different voltages are applied to each nanowire during each state. After applying these voltages to

TABLE II: Control Voltages for One-Bit Full Adder

State/Operation	Horizontal Nanowire	Vertical Nanowire
INA/RESET	H1-H11:V_w	V1-V10:*GND*
RI/SFC	PSOL:V_w; H1:*GND*; H2-H11:V_{wh}	V1-V6:F; V7-V10:V_{wh}
CFM/MFC	H1:V_w; H2-H9:*GND*; H10-H11:V_{wh}	V1-V6:F; V7-V10:V_{wh}
EVM/SFNAND	H1:V_{wh}; H2-H9:F; H10-H11:V_{wh}	V1-V6:V_{wh}; V7-V8:V_w; V9-V10:V_{wh}
EVR/AND	H1:V_{wh}; H2-H9:V_w; H10-H11:*GND*	V1-V6:V_{wh}; V7-V8:F; V9-V10:V_{wh}
INR/INV	H1-H9:V_{wh}; H10-H11:F	V1-V8:V_{wh}; V9-V10:V_w
SO/SFC	H1-H9:V_{wh}; H10-H11:V_w; NSIL:*GND*	V1-V6:V_{wh}; V7-V10:F

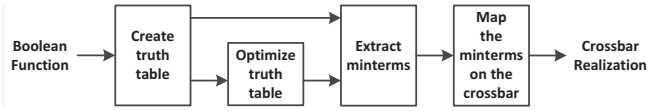

Fig. 5: Mapping Process for an Arbitrary Boolean Function

nanowires, the related memristors perform the specific primitive operations which together execute a Boolean function. The applied voltages to each horizontal and vertical nanowire of the 1-bit FA during each state are summarized in Table II; they are derived from Table I. PSOL and NSIL represent the OL of the previous stage and the IL of the next stage, respectively. For instance, to perform MFC operation (see Fig. 3(b)) required by CFM state in order to configure all minterms in parallel, V_w (V_{ci} in the row CFM of Table I) is applied to horizontal nanowire H1 while *GND* (V_{co} in the row CFM of Table I) is applied to horizontal nanowires H2-H9; vertical nanowires V1-V6 are floating (F) (V_x in the row CFM of Table I). V_{wh} is applied to all the other horizontal and vertical nanowires in order to minimize the impact of sneak path currents [31].

B. Optimized Fast Boolean Logic Circuits

We observed that the nanowires H8 and H9 in Fig. 4 (a) compute the same minterm $\overline{ABC}$ but for the different outputs Sum and $Carry$, respectively. Sharing this minterm would reduce both area and energy consumption. Based on this, we introduce a minterm-sharing strategy. Outputs with a common minterm will be connected to a shared nanowire and employ a MFNAND instead of SFNAND. The MFNAND operation works similarly as SFNAND except that the results are obtained in multiple output memristors as shown in Fig. 3(d). Fig. 4(b) shows the result after applying the minterm-sharing strategy to the 1-bit FA of Fig. 4(a); $\overline{ABC}$ is now presented only by H8. FBLC with the minterm-sharing is referred to as optimized FBLC (OFBLC).

C. Generic Mapping Process

The implementation of the 1-bit FA can be generalized and any Boolean function can be mapped on CMCA by using its truth table as shown in Fig. 5. First, an initial truth table is created based on the Boolean function. This truth table can be further optimized using existing logic optimization tools (e.g., ESPRESSO [32]). Next, minterms are extracted from the (optimized) truth table. Finally, the minterms are mapped on the crossbar, similarly as shown for the 1-bit FA.

V. Results and Evaluation

This section first verifies the 1-bit FA using SPICE simulation; thereafter, it describes the evaluation metrics and comparison between SBLC, FBLC and OFBLC.

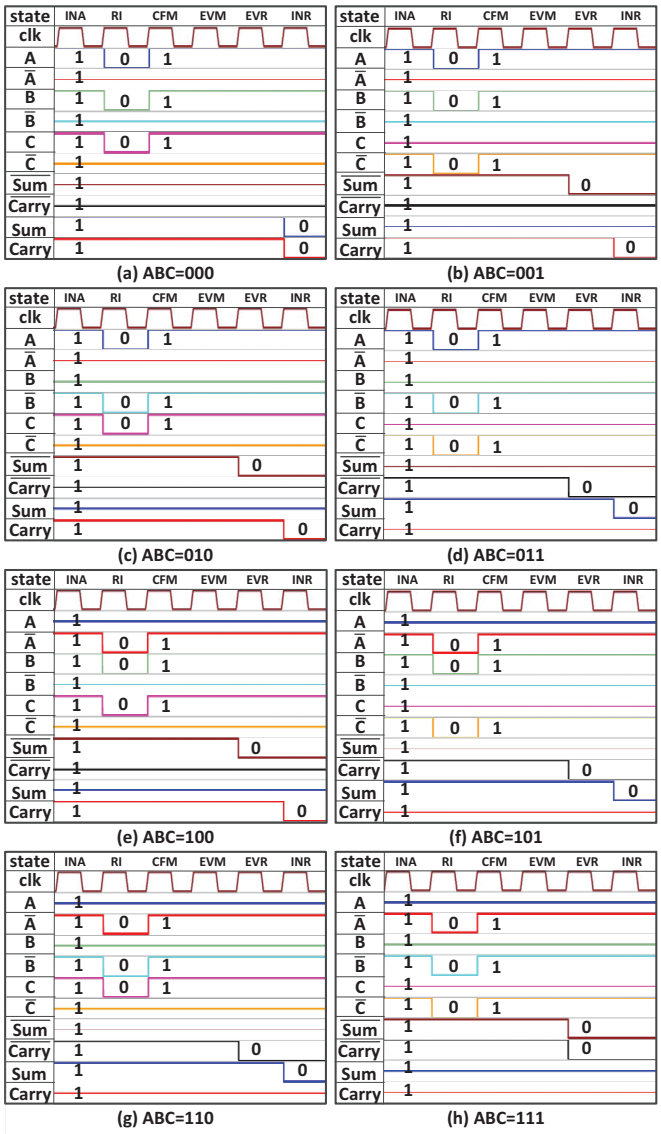

Fig. 6: OFBLC Simulation Result of a One-Bit Full Adder.

A. One-Bit Full Adder Verification

The one-bit full adder based on OFBLC shown in Fig. 4(b) is verified by SPICE simulations. The memristor model, control logic and voltage drivers are described by Verilog-A modules, while the memristor crossbar circuit by a SPICE netlist. The technology parameters and control voltages are listed in Table III. The logic values 1 and 0 are represented by R_{off} and R_{on}, respectively. As only a single CE is simulated, the inputs are directly programmed in state RI (instead of copying data from a previous CE) and state SO is ignored.

All the 8 possible input combinations of the 1-bit FA have been

TABLE III: Simulation Parameters

Parameter	Value	Parameter	Value
R_{on}	100Ω	V_w	1.4V
R_{off}	200kΩ	V_{wh}	0.7V
V_{th}	1V	R_s	1kΩ

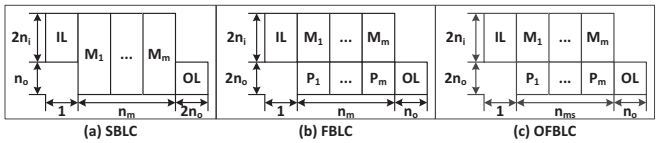

Fig. 7: CE Size for SBLC and (O)FBLC.

verified as shown in Fig. 6. The figure shows for each state the resistance values of the inputs (i.e., A, $\overline{A}$, B, $\overline{B}$, C and $\overline{C}$; see Fig. 4(b) at P(H1,V1-V6)) and outputs (i.e., $\overline{Sum}$, $\overline{Carry}$, Sum and $Carry$; see Fig. 4(b) at P(H9,V7), P(H10,V8), P(H9,V9) and P(H10,V10), respectively). The inputs of the 1-bit FA are programmed during state RI while the outputs are valid at state INR. For instance, Fig. 6(a) shows the case the inputs of the adder are all zeor (ABC=000). Its corresponding outputs ($\overline{Sum}$=1,$\overline{Carry}$=1,Sum=0, $Carry$=0) are visible at state INR.

B. Evaluation Metrics

The evaluation metrics delay, area and energy consumption are used to compare the performance between SBLC and (O)FBLC.

The delay is defined as the number of steps required to execute a CE; each step, executed by a state, is assumed to complete in one cycle. For a fair comparison, we add an extra INR state to SBLC to compensate for the complementary outputs. The delay of a CE that consists of n_m minterms requires $3n_m$+4 cycles for SBLC while 7 cycles for (O)FBLC as summarized in Table IV.

Fig. 7 shows the layout dimensions of a generic CE for SBLC, FBLC and OFBLC. In the figure, n_i presents the number of inputs, n_o the number of outputs, n_m the total number of minterms without considering minterm-sharing, and n_{ms} ($n_{ms} \leq n_m$) the total number of minterms with minterm-sharing. The calculated area is shown in Table IV and includes both active and disabled memristors.

The energy consumption of a CE for a specific input combination is dominated by the resistive switching activity (from high-to-low and low-to-high), as memristors have near-zero leakage power consumption [10]. We assume that both high-to-low and low-to-high resistive switching consume the same amount of energy E_{sw}. In addition, we assume that each input combination occurs with the same probability. The average energy consumption of all the input combinations is used as energy metric.

C. Comparison

To compare (O)FBLC with SBLC, we designed CEs that implement 2- and 4-bit full adders and multipliers. Based on the generic mapping process of Fig. 5, we developed automated scripts that map the adders and multipliers on the crossbar and subsequently evaluate their delay, area, and energy consumption. Both initial and optimized (using ESPRESSO [32]) truth tables are used for comparison. The results are shown in Table V. SBLC with the initial truth table as input is utilized as baseline for comparison. The results of (O)FBLC are nomalized to this baseline which is denoted by the improvement factor (IF).

TABLE IV: Delay and Area Calculation of a CE

Methodology	Delay	Area
SBLC	$3n_m$+4	$(2n_i+n_o)(1+n_m+2n_o)$
FBLC	7	$(2n_i+2n_o)(1+n_m+n_o)$
OFBLC	7	$(2n_i+2n_o)(1+n_{ms}+n_o)$

From the table we conclude the following:

- For SBLC designs, the delay increases significantly when the number of minterms increases. In contrast, the delay of (O)FBLC designs is constant (7 cycles) for all the four cases and up to 500 times faster than those of SBLC designs.
- The area of OFBLC designs is typically smaller than FBLC designs (up to 66% area improvement), while FBLC designs perform area-wise always worse than SBLC (up to 31% area increase). However, the area of OFBLC designs can be larger or smaller than those of SBLC designs depending on whether ESPRESSO is used or not. In case ESPRESSO is not used, we observe that the area of OFBLC designs is smaller than those of SBLC designs (up to 56% area decrease) due to the minterm sharing. In case ESPRESSO is used, we observe that the area of OFBLC designs is typically larger than those of SBLC designs (up to 17% area increase).
- OFBLC designs consume in all cases the lowest energy as they have the least number of minterms, and therefore the least number of switching memristors. In contrast, SBLC designs perform worst; their energy consumption is 1.36 to 3.71 times higher than those of OFBLC designs.
- In case ESPRESSO is not used, OFBLC designs are able to reduce the number of minterms significantly with respect to SBLC and FBLC designs due to minterm sharing. Therefore, OFBLC designs perform best in all these three aspects. However, the benefit of minterm sharing reduces when the logic optimization (i.e., ESPRESSO) is used.

The results of the experiments clearly show that although (O)FBLC designs have typically a slight penalty in terms of area with respect to SBLC designs, but it has a significant improvement both in delay and energy consumption.

VI. Conclusion

Boolean logic circuits based on memristor crossbar are a promising candidate for the next-generation VLSI circuits. This paper proposed fast Boolean logic circuits (FBLC) for memristor crossbars able to execute any Boolean function within constant time. Comparing with state-of-the-art memristor-based Boolean logic circuits, FBLC has significant improvement in terms of delay and energy consumption. Moreover, FBLC provides a systematic design methodology starting from Boolean functions all the way down to memristor crossbar implementations. This process is compatible with existing logic optimization tools (e.g., ESPRESSO), and

TABLE V: Evaluation of Logic Circuits

Full Adder							Multiplier						
	Initial			ESPRESSO				Initial			ESPRESSO		
Metric	SBLC	FBLC	OFBLC	SBLC	FBLC	OFBLC	Metric	SBLC	FBLC	OFBLC	SBLC	FBLC	OFBLC
Size (bits)	2						Size (bits)	2					
n_m	48	48	31	23	23	23	n_m	14	14	9	8	8	7
n_m IF	1	1	1.55	2.09	2.09	2.09	n_m IF	1	1	1.56	1.75	1.75	2
Delay	148	7	7	73	7	7	Delay	46	7	7	28	7	7
Delay IF	1	21.14	21.14	2.03	21.14	21.14	Delay IF	1	6.57	6.57	1.64	6.57	6.57
Area	715	832	560	390	432	432	Area	276	304	224	204	208	192
Delay IF	1	0.86	1.28	1.83	1.66	1.66	Delay IF	1	0.91	1.23	1.35	1.33	1.44
Energy	352	259	174	142	100.50	100.50	Energy	100	73.75	53.75	58	43.88	39.88
Energy IF	1	1.36	2.02	2.48	3.50	3.50	Energy IF	1	1.36	1.86	1.72	2.28	2.51
Size (bits)	4						Size (bits)	4					
n_m	1280	1280	511	135	135	135	n_m	678	678	225	156	156	128
n_m IF	1	1	2.50	9.48	9.48	9.48	n_m IF	1	1	3.01	4.35	4.35	5.30
Delay	3844	7	7	409	7	7	Delay	2038	7	7	472	7	7
Delay IF	1	549.14	549.14	9.40	549.14	549.14	Delay IF	1	291.14	291.14	4.32	291.14	291.14
Area	29693	36008	14476	3358	3948	3948	Area	16680	21984	7488	4152	5280	4384
Area IF	1	0.82	2.05	8.84	7.52	7.52	Area IF	1	0.76	2.23	4.02	3.16	3.80
Energy	14108	11553	4632	982	722	722	Energy	6812	5461.30	1837.30	1271	966.37	774.37
Energy IF	1	1.22	3.05	14.37	19.54	19.54	Energy IF	1	1.25	3.71	5.36	7.05	8.80

therefore, feasible to be integrated into existing design flows. Based on the above features, FBLC can be used as one of the candidates to design basic building blocks in memristor-based VLSI circuits.

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] M. Di Ventra *et al.*, "Memcomputing: a computing paradigm to store and process information on the same physical platform," *arXiv preprint arXiv:1211.4487*, 2012.

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

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

[7] G. Agnus *et al.*, "Two-terminal carbon nanotube programmable devices for adaptive architectures," *Advanced Materials*, vol. 22, pp. 702–706, 2010.

[8] F. Schwierz, "Graphene transistors," *Nature nanotechnology*, vol. 5, pp. 487–496, 2010.

[9] K. Boucart *et al.*, "Double-gate tunnel fet with high-κ gate dielectric," *IEEE Transactions on Electron Devices*, vol. 54, pp. 1725–1733, 2007.

[10] ITRS ERD report. [Online]. Available: http://www.itrs.net/Links/2010ITRS/Home2010.htm

[11] 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.

[12] R. Waser *et al.*, "Redox-based resistive switching memories–nanoionic mechanisms, prospects, and challenges," *Advanced Materials*, vol. 21, pp. 2632–2663, 2009.

[13] H.-S. P. Wong *et al.*, "Metal–oxide rram," *Proceedings of the IEEE*, vol. 100, pp. 1951–1970, 2012.

[14] J. J. Yang *et al.*, "Memristive devices for computing," *Nature nanotechnology*, vol. 8, pp. 13–24, 2013.

[15] H. Kim *et al.*, "Memristor-based multilevel memory," in *Cellular nanoscale networks and their applications (CNNA), 12th international workshop on*. IEEE, 2010, pp. 1–6.

[16] J. R. Burger *et al.*, "Variation-tolerant computing with memristive reservoirs," in *Nanoscale Architectures (NANOARCH), IEEE/ACM International Symposium on*. IEEE, 2013, pp. 1–6.

[17] 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.

[18] 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.

[19] P. Mazumder *et al.*, "Memristors: Devices, models, and applications," *Proceedings of the IEEE*, vol. 100, pp. 1911–1919, 2012.

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

[21] 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.

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

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

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

[25] 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.

[26] 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.

[27] L. Chua, "Resistance switching memories are memristors," *Applied Physics A*, vol. 102, pp. 765–783, 2011.

[28] S. Kvatinsky *et al.*, "Team: Threshold adaptive memristor model," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 60, pp. 211–221, 2013.

[29] W. Lu *et al.*, "Two-terminal resistive switches (memristors) for memory and logic applications," in *Design Automation Conference (ASP-DAC), 16th Asia and South Pacific*. IEEE, 2011, pp. 217–223.

[30] H. Elgabra *et al.*, "Mathematical modeling of a memristor device," in *Innovations in Information Technology (IIT), International Conference on*. IEEE, 2012, pp. 156–161.

[31] 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.

[32] 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.