High-Speed and Low-Cost In-Array Memristive Multipliers using SIXOR and TMSL Logics

Roya Rahimi Disfani*, Mojtaba Valinataj* and Nima TaheriNejad†

***Abstract*—Memristive systems have many promising features, making them suitable for both storage and computation. Memristors can perform logical operations and they can be used as the basic structures in digital circuits such as adders and multipliers. In this paper, at first, a new fast and low-cost Full-Adder (FA) is proposed using Single-cycle In-memristor XOR (SIXOR) and Three Memristors Stateful Logic (TMSL) gates that benefits from the advantages of both logics. Then, the proposed FA is used as one of the basic units inside two new array multipliers. The first proposed multiplier is designed in such a way that it has the lowest computational steps (delay) among the existing designs. This design has on average around 70% lower delay compared to the existing designs. The second proposed multiplier, as the low-cost design, requires a very low number of memristors thanks to reusing the existing resources more efficiently, while still having a low delay. This multiplier achieves on average around 36% memristor reduction compared to the state-of-the-art multipliers. Based on the analysis, both proposed array multipliers have notable efficiency advantages compared to the state-of-the-art designs based on different Figures of Merit (FoMs). For example, based on the balanced FoM, in which the number of computational steps and the number of required memristors have equal weight, the first and the second proposed multipliers achieve up to 4.6× and 14.9× improvements, respectively, compared to the existing designs in 64-bit multiplication.**

***Index Terms*—Memristor, in-memory computation, SIXOR, TMSL, multiplier, FA, in-array computing.**

I. Introduction

Theoretical basis of memristors was first presented by Leon Chua in 1971 [1]. However, many years later, Hewlett Packard (HP) realized the first practical memristor in 2008 [2]. Memristor is a newly discovered passive element with varying resistance that can hold its state. This characteristic leads to one of the most important applications of memristors as memory elements, and is beneficial for energy-efficient In-Memory Computation (IMC). Data processing inside memory with less need for data transmission between memory and processing cores is referred to as IMC, which helps to save energy and resources and alleviates the Von-Neumann bottleneck problem [3]–[5]. Moreover, memristive devices can be used in fields like digital and analog computations [6], [7], artificial neural networks [8]–[10], stochastic computing [11], [12], logic circuits [13], [14], neuromorphic circuits [15], [16] and systems [17] and brain-like computing [18], [19].

* The authors are with the Department of Electrical and Computer Engineering, Babol Noshirvani University of Technology, Babol, Iran. (e-mail: {royarahimidisfani, m.valinataj}@nit.ac.ir).

† Author is with Heidelberg University, Heidelberg, Germany and TU Wien, Vienna, Austria. (e-mail: nima.taherinejad@ziti.uni-heidelberg.de).

(Corresponding author: Mojtaba Valinataj).

Stateful logics are one of the best candidates for IMC. In stateful logic, input and output values are represented as the resistance value (state) of a memristor (also called its memristance) [13], [20]. Material Implication (IMPLY) [13], [20], Three Memristors Stateful Logic (TMSL) [14], Memristor-Aided Logic (MAGIC) [21], Fast and Energy-efficient Logic (FELIX) [22] and Single-cycle In-memristor XOR (SIXOR) [23] are the main stateful logics proposed so far. This kind of logic has omitted the need for read and write to perform logical operations or computing. This way, it achieves more reduction in delay and energy consumption [24]. Stateful logics are compatible with crossbar array structure for performing logical operations or computations inside the memory array (in-array computing).

Addition and multiplication are the most important arithmetic operations in many types of processing. The efficiency of a processing core or in general a computing system depends on the performance of these units. Specifically, optimization of the multipliers in terms of area, delay and power consumption can highly improve the whole computing system. Concerning memristor-based multipliers, there exist a few memristive designs that similar to non-memristive multipliers are mainly categorized into serial, array and parallel multipliers. Based on the multiplier type, the cost and delay that are basically on opposite sides can be low or high.

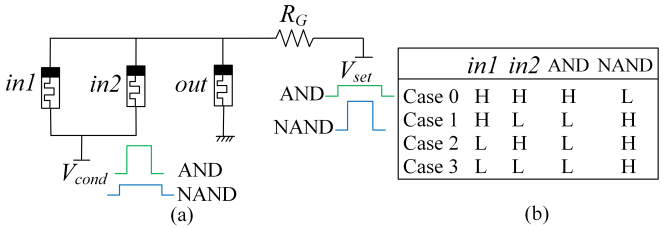
In this paper, we focus on designing more efficient in-array memristive multipliers with respect to area, delay and power consumption utilizing both SIXOR and TMSL logic gates. To do so, a new efficient Full-Adder (FA) cell is proposed based on a combination of SIXOR and TMSL logics. Then, it is used inside an array multiplier, inspired by the Braun multiplier, to obtain two new multipliers that utilize memristors more efficiently thanks to reusing some memristors. This is performed, in some computational steps, by resetting the memristance of the memristors that have the potential of reusing in the next computations. This way, the total number of memristors and as a result, the total area will decrease. The main innovations of this work can be summarized as following:

1) Proposing a more efficient FA structure based on a tuned combination of two different logics, i.e., SIXOR and TMSL

2) Constructing two new fast and low-cost array multipliers utilizing the proposed FA

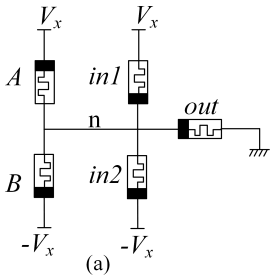
3) Decreasing the number of required memristors by reusing some memristors in the multiplication process while having a higher speed compared to previous designs

The rest of this paper is organized as follows. A review of SIXOR and TMSL memristive logics is presented in Section II. In Section III, previous works on memristive



	in1	in2	AND	NAND
Case 0	H	H	H	L
Case 1	H	L	L	H
Case 2	L	H	L	H
Case 3	L	L	L	H

Fig. 1: TMSL, (a) circuit diagram and (b) truth-table [14].



	in1	in2	out
Case 0	H	H	H
Case 1	H	L	L
Case 2	L	H	L
Case 3	L	L	H

Fig. 2: SIXOR, (a) circuit diagram and (b) truth-table [23].

multipliers are reviewed. The proposed FA and the proposed array multipliers are described in Section IV. Section V presents the simulation results, verifying the operation of the proposed designs. Comparison with the state-of-the-art designs is discussed in Section VI, and the paper is concluded in Section VII.

II. Memristive Logics - Review

The minimum and maximum values of a memristance are represented by R_{on} and R_{off}, respectively, and the memristance can vary between these values. In Stateful logics, the logical values are represented by the resistance of memristors. Hence, R_{on} and R_{off}, also called Low Resistance State (LRS) or L and High Resistance State (HRS) or H, respectively, need to be mapped to logical values [25]. In this work, they are assumed to be '0' for HRS and '1' for LRS. The 'input values' of stateful logics, which are the state of the memristors before the beginning of the logical operation, are written into them either separately (as single memory elements in a crossbar) or are produced as the output state of previous operations. If the output values are needed outside the array, they are read using typical memory read-out operations.

IMPLY is the first stateful logic proposed in 2009 [13] and is widely used in memristive stateful adders. In 2014, MAGIC [21] was proposed to implement NOT, OR, NOR, AND and NAND logical operations, from which only the NOT and NOR structures are compatible with the crossbar array. Since then, other single-cycle memristive logics have been considered which include FELIX [22], TMSL [14] and SIXOR [23] from 2016 to 2021. These logics are also crossbar compatible stateful logics which can be used in the design of various computation units such as adders and multipliers. The logics utilized in this paper are reviewed in the following:

1) TMSL: Placing three memristors and a resistor according to Fig. 1(a), constructs the TMSL logic gate. TMSL was proposed to perform single-cycle and crossbar array compatible AND and NAND logical operations (Fig. 1(b)). The output state is affected by the pulse duration, resistor and amplitude of the applied voltage sources (V_{set} and V_{cond}). Applying a narrow high voltage pulse (V_{cond}) to the input memristors and a wide low voltage pulse (V_{set}) to the output memristor leads to the AND operation. Similarly, applying the voltage pulses with the reverse properties shown in Fig. 1(a) leads to the NAND operation.

In the TMSL logic, the *out* memristor is initialized to H, and H and L are assumed as '1' and '0' logic values, respectively. To perform the AND operation, the output state should be in H only when both of the input memristors are in H (Case 0) as shown in Fig. 1(b). The applied low voltage pulse (V_{set}) is smaller than the threshold voltage of the *out* memristor (V_{th}) and cannot change the output state. In other cases, at least one of the input memristors is in L allowing the high voltage pulse V_{cond} reaches to the *out* memristor and changes the *out* state to L. Performing the NAND operation is similar to AND.

2) SIXOR: In 2021, a new single-cycle and crossbar compatible XOR operator called SIXOR was proposed in [23], shown in Fig. 2(a). As depicted in Fig. 2(a), SIXOR consists of five memristors where *A* and *B* are auxiliary memristors (called C_{aux}) connected to input (*in1* and *in2*) and output (*out*) memristors. The *out* and auxiliary memristors are initialized to H which is assumed to be the '0' logic value. Thus, L represents the '1' logic value in this gate.

Whenever input memristors are at the same state with the same memristance (Case 0 and Case 3 in Fig. 2(b)), the common node voltage (V_n) will become zero which makes the output state remains at H. In Case 1, V_n will be equal to $-V_x$. Hence, *A* sees $2V_x$ across itself which can change the state from H to L. Meanwhile, *in2* changes its state to H due to the current passing through it. So, V_n increases to V_x, approximately, changing the state of *out* to L. In Case 2, *in1* is short-circuit which means V_n=V_x that causes the output state to change from H to L.

III. Memristive Multipliers - Review

Memristive multipliers are a significant part of memristor-based computing systems. Compared to the existing memristive adders, there exist fewer multipliers in the literature which are mostly IMPLY-based designs. Existing memristive multipliers are classified as the following:

1) Shift&Add Multiplier: The IMPLY-based Shift&Add multiplier proposed in [26] is based on the so-called Shift&Add algorithm. In this algorithm, in each stage, the first operand is shifted left but it is added to a sum register only if the corresponding bit in the second operand is '1'. This multiplier operates in serial and because of requiring *N* stages in an $N \times N$-bit multiplier, it includes *N* shifts and maximum *N* add operations. The design proposed in [26] uses optimized IMPLY-based multiplexers, adders and shift registers that lead to a very low-cost design. However, because of the serial nature, it has the highest delay compared to other multipliers. This multiplier requires a low number of memristors equal to $7N + 1$ and a low number of switches equal to $8N - 1$, as well. The output product requires $2N^2 + 21N$ computational steps to be ready in an $N \times N$-bit multiplier.

2) Array Multiplier: The array-type multiplier for performing $N \times N$-bit multiplication consists of rows of FA and

Half-Adder (HA) cells along with N^2 AND gates. IMPLY-based array multiplier presented in [27] uses $(N-1)$ rows of adders and incorporates some optimizations; however, it requires a high number of memristors. Another IMPLY-based array multiplier [28] is based on the Braun multiplier. In the Braun multiplier, the output carries of FA/HA cells in each row are diagonally sent to the next row which makes it faster. However, the array multiplier of [28] still requires a high number of memristors and switches compared to other designs.

3) Dadda Multiplier: In 1965, L. Dadda proposed a new parallel scheme to speed up the multiplication [29]. This multiplier uses FA/HA cells in some stages but with a higher degree of parallelism compared to array multipliers. The IMPLY-based Dadda multiplier was first presented in [30]. This multiplier is favored for faster computation compared to [26] and [27] but comes at the cost of a large number of memristors and switches.

4) Semi-serial Adder-based Multiplier: The IMPLY-based multiplier proposed in [24] is based on the semi-serial adder proposed in [31]. To perform the $N \times N$-bit multiplication, a $(2N-1)$-bit semi-serial adder is used for $\lceil \frac{N}{2} \rceil$ times. Due to requiring a low number of memristors and switches in the base adder, this multiplier also inherits this property, i.e., area-efficiency. However, its delay (computational steps) is not low and is higher than that of array [27], [28] and Dadda [30] memristive multipliers.

IV. Proposed Designs

Since addition is the basic operation utilized inside the multiplication, improving the memristor-based adders will help to attain more efficient memristive multipliers. Most of the existing memristive FAs are based on the IMPLY logic such as [31]–[37]. The first FA based on a combination of some stateful logics is presented in [23], in which SIXOR, TMSL and FELIX are used to obtain a new and efficient structure. In this section, after stating some design considerations, we introduce a new FA based on a combination of TMSL and SIXOR logics. Then, utilizing this new FA cell and some optimizations, we propose two new array multipliers inspired by the Braun scheme.

A. Design Considerations

As stated in [14], TMSL can perform the AND and NAND logical operations considering certain operational conditions. If '1' and '0' logic values are assumed for H and L, respectively, the AND and NAND operations are obtained according to Fig. 1(b) that we need inside FA/HA cells and to produce the partial products. But this assumption is different from that of SIXOR, which we utilize it in combination with TMSL. In this paper, similar to SIXOR, we assume '0' and '1' logic values for H and L, respectively. To solve this inconsistency, we tune the operational conditions of TMSL in a way that by assuming H and L as '0' and '1', respectively (the same as SIXOR), correct output results will be produced. For this purpose, we use $R_G = 13k\Omega$ for TMSL-AND, $R_G = 3.9k\Omega$ for TMSL-NAND and a pulse duration of 2μs different from those of the TMSL logic in [14], to produce AND and NAND operations with proper logic values for H and L.

TABLE I: Computational Steps of Half-Adder and the Proposed Full-Adder.

Steps	FA operation	HA operation
1	$Half-Carry(HC) = \overline{in1.in2}$	$C_{HA} = in1.in2$
2	$Half-Sum(HS) = in1 \oplus in2$ level correction of HS	$S_{HA} = in1 \oplus in2$
3	$int = \overline{C_{in}.HS}$ Re-initialization of *in1, in2 and* C_{aux}	-
4	$S_{FA} = C_{in} \oplus HS$ $C_{FA} = \overline{HC.int}$	-

In each separate gate, the input memristors are initialized to the desired initial states with the memristance that is exactly mapped to '0' or '1' logic values. However, in a memristive circuit, there may be consecutive gates, or the input memristors might be used in different operations. Thus, maintaining the initial state after the first operation is crucial to prevent errors in the next steps. In practice, the output of a computational step may be used as an input in the next steps and must be in a correct state, even if it does not have an exact or strong '0' (H) or '1' (L) value. The SIXOR and TMSL gates do not show any considerable sensitivity to H state but are somewhat sensitive to the input memristance in the L state. In SIXOR, if the input memristors are not exactly in the L state, an incorrect output state may be produced. This issue is lighter in TMSL gates; however, consecutive changes in the memristance of TMSL gates can still lead to errors in the next steps. To address this problem, the parameters are optimized to produce and maintain a strong H state, while approximating L state to '1' which necessitates level correction in certain computational steps. A level correction is performed by connecting the intended memristors to the highest voltage source of the circuit for a very short time (in the order of nanoseconds). It should be noted that applying a wide high-voltage pulse to a memristor can change its state from H to L. Therefore, a very narrow pulse duration is used, one that does not alter the H state but corrects the L state.

In the proposed designs, level correction is applied to the outputs that will be used as the inputs of the SIXOR gates in the next steps, regardless of whether they are in the H or L state. This results in a small drift for those in the H state. However, this drift does not cause errors in subsequent logic operations. According to our simulations, the SIXOR and tuned TMSL gates interpret the H state with this small drift as '0', producing outputs with correct logic levels (exact '0' for H or very close to '1' for L). Furthermore, the accumulation of state drift is prevented because any potential accumulation is effectively mitigated at the beginning of the next step.

B. Proposed Full-Adder

The computational steps of the proposed FA and also the respective HA are shown in Table I. The output sum (S_{HA}) and carry (C_{HA}) of the HA are obtained by a SIXOR gate and a TMSL-AND gate, respectively, in two steps. For the FA, the HS and the HC are produced in the first two steps. As HS will be used in the next SIXOR gate, a level correction is performed in Step 2 to prevent the wrong result in the next steps. In Table I, re-initialization means adjusting one or

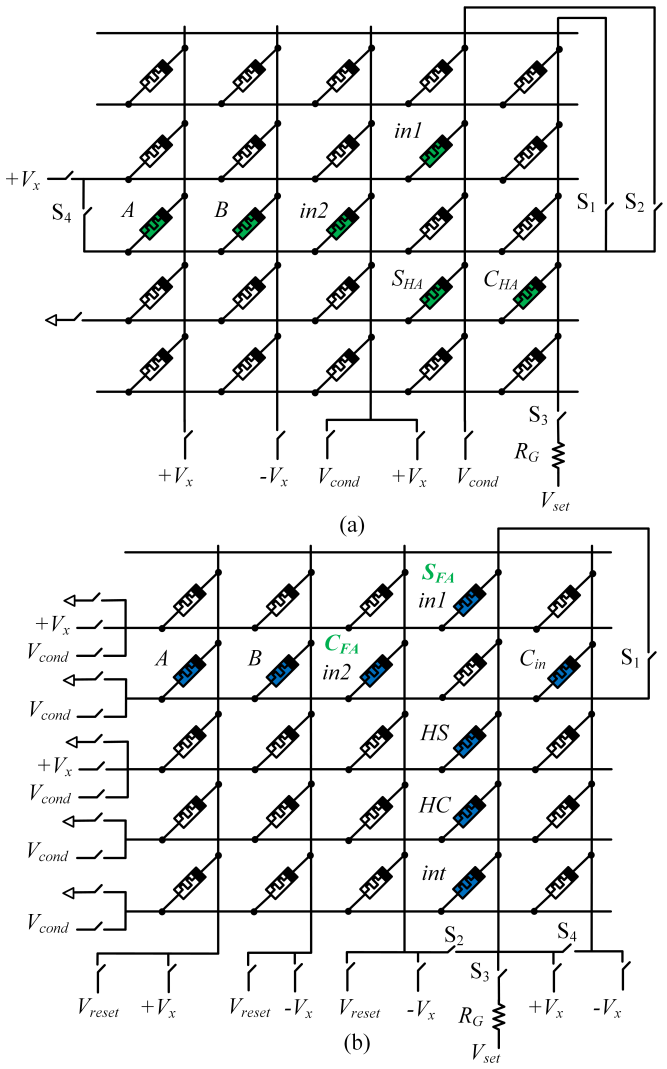


Fig. 3: Schematic of in-array (a) HA and (b) the proposed FA.

more memristors using a reset pulse. To do so, $V_{reset} = 1.5V$ is applied to the positive node of memristors that makes the memristance to be re-initialized to H.

Re-initializing input memristors and auxiliary memristors (*A* and *B* in Fig. 2 as C_{aux}) of SIXOR in Step 3 (Table I) makes them ready to be reused in the next step. This way, the required number of memristors for a single FA is reduced from 12 to 8 making a high area reduction. The output carry (C_{FA}) and Sum (S_{FA}) are finally saved on input memristors.

The in-array structures of the HA and proposed FA are depicted in Fig. 3. HA is implemented by using a SIXOR gate and a TMSL-AND gate as illustrated in Fig. 3(a). A single HA requires six memristors shown in green color as well as four switches (S_1 to S_4) for connecting the desired rows or columns. Similarly, the proposed FA is shown in Fig. 3(b). The eight blue memristors are connected to the desired columns or rows via four switches (S_1 to S_4). These switches are symbolically shown on the crossbar array. In practice, they will be fabricated on the Complementary Metal-Oxide Semiconductor (CMOS) part inside the control and read/write circuit as the periphery of the crossbar. Other switches shown in Fig. 3 are related to the control logic to apply the appropriate voltages.

We will use the proposed FA in the new multipliers.

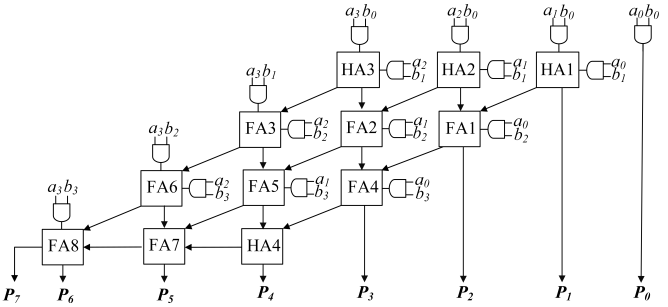


Fig. 4: 4×4-bit Braun-type array multiplier.

However, it can be used for making *N*-bit adders, as well. An *N*-bit simple adder which includes *N* FAs requires $2N + 6$ memristors and produces the output carry and sum in $4N$ computational steps. The first FA of this adder consumes eight memristors. But each of the remaining $N - 1$ FAs only needs two memristors as inputs (equal to $2(N - 1)$ memristors), whereas two auxiliary memristors, HC, HS and *int* memristors of the first FA are reused and C_{in} is also counted in the first FA and the outputs are stored on the input memristors.

C. First Proposed Multiplier

For designing an ordinary $N \times N$-bit Braun-type array multiplier, N^2 AND gates, *N* HAs and $N^2 - 2N$ FAs are required. Fig. 4 shows the 4×4-bit Braun multiplier that produces the 8-bit product ($P_7P_6P_5P_4P_3P_2P_1P_0$) based on two 4-bit inputs ($a_3a_2a_1a_0$ and $b_3b_2b_1b_0$). The first proposed multiplier needs $4N^2 - 2N$ memristors. it consumes $2N$ memristors for two *N*-bit input operands and only N^2 memristors for N^2 AND gates that produce the partial products. More precisely, only the output memristor of an AND gate is counted because the input memristors are counted for SIXOR gates. The input memristors of the AND gates are reused as the auxiliary memristors of SIXOR gates, which leads to a reduction in the number of memristors. Since the input memristors of HAs are the same as the output memristors of the previous computational units (AND gates or FAs) and reused auxiliary memristors, each HA just adds two memristors, thus totally $2N$ memristors are required for HAs in the $N \times N$-bit Braun multiplier. In addition, each FA cell consumes only three memristors for storing HS, HC and *int*. This is because the input memristors of a FA are the same as the output memristors of previous computational units and the output bits are saved on the input memristors. As shown in Fig. 4, at least one of the inputs of FAs (except FA7) is the output of AND gates. However, the input memistors are used to save the outputs of FAs. Thus, some bits of the output product (P_0, P_3, P_5, P_6) are saved on the outputs of the AND gates (called a_0b_0, a_0b_2, a_1b_3 and a_2b_3) and the other bits on S_{HA} or C_{HA} memristors. Therefore, each FA requires three memristors and all FAs require $3N^2 - 6N$ memristors, and the total number of required memristors will be $4N^2 - 2N$.

The operation details of the computational steps in the first proposed multiplier are shown in Table II for the 4 × 4-bit multiplication. In general, the preparation of partial products is performed in *N* steps using N^2 AND gates. More precisely, *N* AND gates with different inputs can be grouped (*N* groups altogether). The AND gates of each group can work

TABLE II: COMPUTATIONAL STEPS AND THEIR OPERATIONS IN THE FIRST PROPOSED MULTIPLIER (4×4-BIT).

Step	Executed operation	Equivalent logic
1	**AND** $(a_0, b_0) \rightarrow \boldsymbol{P_0}, (a_1, b_1), (a_2, b_2), (a_3, b_3)$	$\boldsymbol{P_0} = a_0.b_0$, $a_1b_1 = a_1.b_1$, $a_2b_2 = a_2.b_2$, $a_3b_3 = a_3.b_3$
2	**AND** $(a_0, b_1), (a_1, b_2), (a_2, b_3), (a_3, b_0)$	$a_0b_1 = a_0.b_1$, $a_1b_2 = a_1.b_2$, $a_2b_3 = a_2.b_3$, $a_3b_0 = a_3.b_0$
3	**AND** $(a_0, b_2), (a_1, b_3), (a_2, b_0), (a_3, b_1)$	$a_0b_2 = a_0.b_2$, $a_1b_3 = a_1.b_3$, $a_2b_0 = a_2.b_0$, $a_3b_1 = a_3.b_1$
4	**AND** $(a_0, b_3), (a_1, b_0), (a_2, b_1), (a_3, b_2)$	$a_0b_3 = a_0.b_3$, $a_1b_0 = a_1.b_0$, $a_2b_1 = a_2.b_1$, $a_3b_2 = a_3.b_2$
5	$C_{HA_{1-3}} =$ **AND**$(a_0b_1, a_1b_0), (a_2b_0, a_1b_1), (a_3b_0, a_2b_1)$ *a_0, a_1, a_2, b_0, b_1 and $b_2 = 0$*	$C_{HA_1} = a_0b_1.a_1b_0$, $C_{HA_2} = a_2b_0.a_1b_1$, $C_{HA_3} = a_3b_0.a_2b_1$ $a_0 = a_1 = a_2 = b_0 = b_1 = b_2 = 0$
6	$S_{HA_{1-3}} =$ **XOR**$(a_0b_1, a_1b_0) \rightarrow \boldsymbol{P_1}, (a_2b_0, a_1b_1), (a_3b_0, a_2b_1)$ *Level correction of $C_{HA_{1-3}}$, $S_{HA_{2-3}}$, a_0b_2, a_0b_3, a_1b_2, a_1b_3, a_2b_2, a_2b_3, a_3b_1, a_3b_2 and a_3b_3*	$\boldsymbol{P_1} = a_0b_1 \oplus a_1b_0$, $S_{HA_2} = a_2b_0 \oplus a_1b_1$, $S_{HA_3} = a_3b_0 \oplus a_2b_1$
7	$HC_{1-3} =$ **NAND**$(a_0b_2, S_{HA_2}), (a_1b_2, S_{HA_3}), (a_2b_2, a_3b_1)$ *a_0, a_1, a_2, b_0, b_1 and $b_2 = 0$*	$HC_1 = \overline{a_0b_2.S_{HA_2}}$, $HC_2 = \overline{a_1b_2.S_{HA_3}}$, $HC_3 = \overline{a_2b_2.a_3b_1}$ $a_0 = a_1 = a_2 = b_0 = b_1 = b_2 = 0$
8	$HS_{1-3} =$ **XOR**$(a_0b_2, S_{HA_2}), (a_1b_2, S_{HA_3}), (a_2b_2, a_3b_1)$ *Level correction of HS_{1-3}*	$HS_1 = a_0b_2 \oplus S_{HA_2}$, $HS_2 = a_1b_2 \oplus S_{HA_3}$, $HS_3 = a_2b_2 \oplus a_3b_1$
9	$int_{1-3} =$ **NAND**$(C_{HA_1}, HS_1), (C_{HA_2}, HS_2), (C_{HA_3}, HS_3)$ *a_0, a_1, a_2, b_0, b_1 and $b_2 = 0$* *Input memristors of FAs $(a_0b_2, S_{HA_2}, a_1b_2, S_{HA_3}, a_2b_2, a_3b_1) = 0$*	$int_1 = \overline{C_{HA_1}.HS_1}$, $int_2 = \overline{C_{HA_2}.HS_2}$, $int_3 = \overline{C_{HA_3}.HS_3}$ $a_0 = a_1 = a_2 = b_0 = b_1 = b_2 = 0$ $a_0b_2 = S_{HA_2} = a_1b_2 = S_{HA_3} = a_2b_2 = a_3b_1 = 0$
10	$S_{FA_{1-3}} =$ **XOR**$(C_{HA1}, HS_1) \rightarrow \boldsymbol{P_2}, (C_{HA_2}, HS_2), (C_{HA_3}, HS_3)$ $C_{FA_{1-3}} =$ **NAND**$(HC_1, int_1), (HC_2, int_2), (HC_3, int_3)$ *Level correction of $S_{FA_{2-3}}$ and $C_{FA_{1-3}}$*	$\boldsymbol{P_2} = C_{HA_1} \oplus HS_1$, $S_{FA_2} = C_{HA_2} \oplus HS_2$, $S_{FA_3} = C_{HA_3} \oplus HS_3$ $C_{FA_1} = \overline{HC_1.int_1}$, $C_{FA_2} = \overline{HC_2.int_2}$, $C_{FA_3} = \overline{HC_3.int_3}$
11	$HC_{4-6} =$ **NAND**$(a_0b_3, S_{FA_2}), (a_1b_3, S_{FA_3}), (a_2b_3, a_3b_2)$ *a_0, a_1, a_2, b_0, b_1 and $b_2 = 0$*	$HC_4 = \overline{a_0b_3.S_{FA_2}}$, $HC_5 = \overline{a_1b_3.S_{FA_3}}$, $HC_6 = \overline{a_2b_3.a_3b_2}$ $a_0 = a_1 = a_2 = b_0 = b_1 = b_2 = 0$
12	$HS_{4-6} =$ **XOR**$(a_0b_3, S_{FA_2}), (a_1b_3, S_{FA_3}), (a_2b_3, a_3b_2)$ *Level correction of HS_{4-6}*	$HS_4 = a_0b_3 \oplus S_{FA_2}$, $HS_5 = a_1b_3 \oplus S_{FA_3}$, $HS_6 = a_2b_3 \oplus a_3b_2$
13	$int_{4-6} =$ **NAND**$(C_{FA_1}, HS_4), (C_{FA_2}, HS_5), (C_{FA_3}, HS_6)$ *a_0, a_1, a_2, b_0, b_1 and $b_2 = 0$* *Input memristors of FAs $(a_0b_3, S_{FA_2}, a_1b_3, S_{FA_3}, a_2b_3, a_3b_2) = 0$*	$int_4 = \overline{C_{FA_4}.HS_4}$, $int_5 = \overline{C_{FA_5}.HS_5}$, $int_6 = \overline{C_{FA_6}.HS_6}$ $a_0 = a_1 = a_2 = b_0 = b_1 = b_2 = 0$ $a_0b_3 = S_{FA_2} = a_1b_3 = S_{FA_3} = a_2b_3 = a_3b_2 = 0$
14	$S_{FA_{4-6}} =$ **XOR**$(C_{FA_4}, HS_4) \rightarrow \boldsymbol{P_3}, (C_{FA_5}, HS_5), (C_{FA_6}, HS_6)$ $C_{FA_{4-6}} =$ **NAND**$(HC_4, int_4), (HC_5, int_5), (HC_6, int_6)$ *Level correction of $S_{FA_{5-6}}$ and $C_{FA_{4-6}}$*	$\boldsymbol{P_3} = C_{FA_4} \oplus HS_4$, $S_{FA_5} = C_{FA_5} \oplus HS_5$, $S_{FA_6} = C_{FA_6} \oplus HS_6$ $C_{FA_4} = \overline{HC_4.int_4}$, $C_{FA_5} = \overline{HC_5.int_5}$, $C_{FA_6} = \overline{HC_6.int_6}$
15	$C_{HA_4} =$ **AND**(C_{FA_4}, S_{FA_5}) *a_0, a_1, a_2, b_0, b_1 and $b_2 = 0$*	$C_{HA_4} = C_{FA_4}.S_{FA_5}$ $a_0 = a_1 = a_2 = b_0 = b_1 = b_2 = 0$
16	$S_{HA_4} =$ **XOR**$(C_{FA_4}, S_{FA_5}) \rightarrow \boldsymbol{P_4}$ *Level correction of C_{HA_4}*	$\boldsymbol{P_4} = C_{FA_4} \oplus S_{FA_5}$
17	$HC_7 =$ **NAND**(C_{HA_4}, S_{FA_6})	$HC_7 = \overline{C_{HA_4}.S_{FA_6}}$
18	$HS_7 =$ **XOR**(C_{HA_4}, S_{FA_6}) *Level correction of HS_7*	$HS_7 = C_{HA_4} \oplus S_{FA_6}$
19	$int_7 =$ **NAND**(C_{FA_5}, HS_7) *a_1 and $b_1 = 0$* *Input memristors of FAs $(C_{HA_4}, S_{FA_6}) = 0$*	$int_7 = \overline{C_{FA_5}.HS_7}$ $a_1 = b_1 = 0$ $C_{HA_4} = S_{FA_6} = 0$
20	$S_{FA_7} =$ **XOR**$(C_{FA_5}, HS_7) \rightarrow \boldsymbol{P_5}$ $C_{FA_7} =$ **NAND**$(HC_7.int_7)$ *Level correction of C_{FA_7}*	$\boldsymbol{P_5} = C_{FA_5} \oplus HS_7$ $C_{FA_7} = \overline{HC_7.int_7}$
21	$HC_8 =$ **NAND**(a_3b_3, C_{FA_7})	$HC_8 = \overline{a_3b_3.C_{FA_7}}$
22	$HS_8 =$ **XOR**(a_3b_3, C_{FA_7}) *Level correction of HS_8*	$HS_8 = a_3b_3 \oplus C_{FA_7}$
23	$int_8 =$ **NAND**(C_{FA_6}, HS_8) *a_2 and $b_2 = 0$* *Input memristors of FAs $(C_{FA_7}, a_3b_3) = 0$*	$int_8 = \overline{C_{FA_6}.HS_8}$ $a_2 = b_2 = 0$ $C_{FA_7} = a_3b_3 = 0$
24	$S_{FA_8} =$ **XOR**$(C_{FA_6}, HS_8) \rightarrow \boldsymbol{P_6}$ $C_{FA_8} =$ **NAND**$(HC_8, int_8) \rightarrow \boldsymbol{P_7}$	$\boldsymbol{P_6} = C_{FA_6} \oplus HS_8$ $\boldsymbol{P_7} = \overline{HC_8.int_8}$

simultaneously because of having different input memristors. This reduces the total delay further. So, the outputs of AND gates are computed in N steps (four steps in Table II). As shown in Fig. 4, all the FA/HA cells of each row (except the last row) work in parallel. For the first row, two steps (steps 5 and 6 in Table II) are needed to compute the outputs of HAs including C_{HA1-3} (that means C_{HA1}, C_{HA2} and C_{HA3}) and S_{HA1-3} (that means S_{HA1}, S_{HA2} and S_{HA3}) after the operation of the AND gates. In Table II "*Input memristors of FAs* = 0" means re-initialization. The outputs of the second row (C_{FA1-3} and S_{FA1-3}) are prepared in 4 steps. Similarly, the outputs of the third row (C_{FA4-6} and S_{FA4-6}) are prepared in another 4 steps (end of Step 14). The FA/HA cells of the last row work in serial, which means HA4 in Fig. 4 adds two steps and each one of FA7 and FA8 cells adds 4 more steps to the total computational steps (altogether 24 steps).

Algorithm 1 shows how we can extend the proposed multiplier for $N \times N$-bit multiplication. Furthermore, the flowchart overview of the proposed algorithm is shown in Fig. 5. In general, the total number of computational steps in the first proposed multiplier is $9N - 12$. It is obtained based on the critical path, which consists of N AND gates (N steps), two HAs (4 steps) and $2N-4$ FAs ($4(2N-4)$ steps). As shown in Fig. 3, the in-array structure of each HA or FA circuit needs four switches. In general, in an $N \times N$-bit array multiplier, $4N$ switches are needed for HA operations and $4(N^2 - 2N)$ switches for FAs. Therefore, the total number of switches in the first proposed multiplier is $4N^2 - 4N$.

Algorithm 1 First proposed $N \times N$-bit array multiplier.

Inputs: $a_{N-1}a_{N-2}...a_1a_0, b_{N-1}b_{N-2}...b_1b_0$
Outputs: $P_{2N-1}P_{2N-2}...P_1P_0$
1: $P_0 = a_0$ **AND** b_0,...,$a_{N-1}b_{N-1} = a_{N-1}$ **AND** b_{N-1}
2: $a_0b_1 = a_0$ **AND** $b_1, a_1b_2 = a_1$ **AND** $b_2, ..., a_{N-2}b_{N-1} = a_{N-2}$ **AND** b_{N-1}, $a_{N-1}b_0 = a_{N-1}$ **AND** b_0
.
.
.
N: $a_0b_{N-1} = a_0$ **AND** $b_{N-1}, a_1b_0 = a_1$ **AND** $b_0, ..., a_{N-1}b_{N-2} = a_{N-1}$ **AND** b_{N-2}
N+1: $C_{HA_1} = a_0b_1$ **AND** a_1b_0,...,$C_{HA_{N-1}} = a_{N-2}b_1$ **AND** $a_{N-1}b_0$
N+2: $S_{HA_1} = a_0b_1$ **XOR** $a_1b_0 \rightarrow P_1, ..., S_{HA_{N-1}} = a_{N-2}b_1$ **XOR** $a_{N-1}b_0$
LOOP Process
assuming *in1, in2* and C_{in} as inputs of FAs and $j = 1$
5N-6: **for** $i = 1$ to $N - 2$ **do**
$HC_j, ..., HC_{(j+N-2)} = in1$ **NAND** $in2$
$HS_j, ..., HS_{(j+N-2)} = in1$ **XOR** $in2$
$int_j, ..., int_{(j+N-2)} = C_{in}$ **NAND** HS
$S_{FA_j}, ..., S_{FA_{(j+N-2)}} = C_{in}$ **XOR** $HS \rightarrow P_2$,..., P_{N-1}
$C_{FA_j}, ..., C_{FA_{(j+N-2)}} = HC$ **XOR** int
$j = j + N - 1$
end for
5N-5: $C_{HA_N} = in1$ **AND** $in2$
5N-4: $S_{HA_N} = in1$ **XOR** $in2 \rightarrow P_N$
LOOP Process
9N-12: **for** $i = (N^2 - 3N + 3)$ to $(N^2 - 2N)$ **do**
$HC_i = in1$ **NAND** $in2$
$HS_i = in1$ **XOR** $in2$
$int_i = C_{in}$ **NAND** HS
$S_{FA_i} = C_{in}$ **XOR** $HS \rightarrow P_{N+1}$,..., P_{2N-2}
$C_{FA_i} = HC$ **XOR** $int \rightarrow P_{2N-1}$
end for
return P_{2N-1},..., P_0

Algorithm 2 Second proposed $N \times N$-bit array multiplier.

Inputs: $a_{N-1}a_{N-2}...a_1a_0, b_{N-1}b_{N-2}...b_1b_0$
Outputs: $P_{2N-1}P_{2N-2}...P_1P_0$
1: $P_0 = a_0$ **AND** b_0,...,$a_{N-1}b_{N-1} = a_{N-1}$ **AND** b_{N-1}
2: $a_0b_1 = a_0$ **AND** $b_1, a_1b_2 = a_1$ **AND** $b_2, ..., a_{N-2}b_{N-1} = a_{N-2}$ **AND** b_{N-1}, $a_{N-1}b_0 = a_{N-1}$ **AND** b_0
.
.
.
N: $a_0b_{N-1} = a_0$ **AND** $b_{N-1}, a_1b_0 = a_1$ **AND** $b_0, ..., a_{N-1}b_{N-2} = a_{N-1}$ **AND** b_{N-2}
N+1: $C_{HA_1} = a_0b_1$ **AND** a_1b_0,...,$C_{HA_{N-1}} = a_{N-2}b_1$ **AND** $a_{N-1}b_0$
N+2: $S_{HA_1} = a_0b_1$ **XOR** $a_1b_0 \rightarrow P_1, ..., S_{HA_{N-1}} = a_{N-2}b_1$ **XOR** $a_{N-1}b_0$
LOOP Process
6N-8: **for** $i = 1$ to $N - 2$ **do**
assuming *in1, in2* and C_{in} as inputs of FAs
$HC_i, ..., HC_{(N-1)} = in1$ **NAND** $in2$
$HS_i, ..., HS_{(N-1)} = in1$ **XOR** $in2$
$int_i, ..., int_{(N-1)} = C_{in}$ **NAND** HS
$S_{FA_i}, ..., S_{FA_{(N-1)}} = C_{in}$ **XOR** $HS \rightarrow P_2$,..., P_{N-1}
$C_{FA_i}, ..., C_{FA_{(N-1)}} = HC$ **XOR** int
Re-initialization of *HC*, *HS* and *int*
end for
6N-7: $C_{HA_N} = in1$ **AND** $in2$
6N-6: $S_{HA_N} = in1$ **XOR** $in2 \rightarrow P_N$
LOOP Process
10N-14: **for** $i = (N^2 - 3N + 3)$ to $(N^2 - 2N)$ **do**
$HC_i = in1$ **NAND** $in2$
$HS_i = in1$ **XOR** $in2$
$int_i = C_{in}$ **NAND** HS
$S_{FA_i} = C_{in}$ **XOR** $HS \rightarrow P_{N+1}$,..., P_{2N-2}
$C_{FA_i} = HC$ **XOR** $int \rightarrow P_{2N-1}$
end for
return P_{2N-1},...,P_0

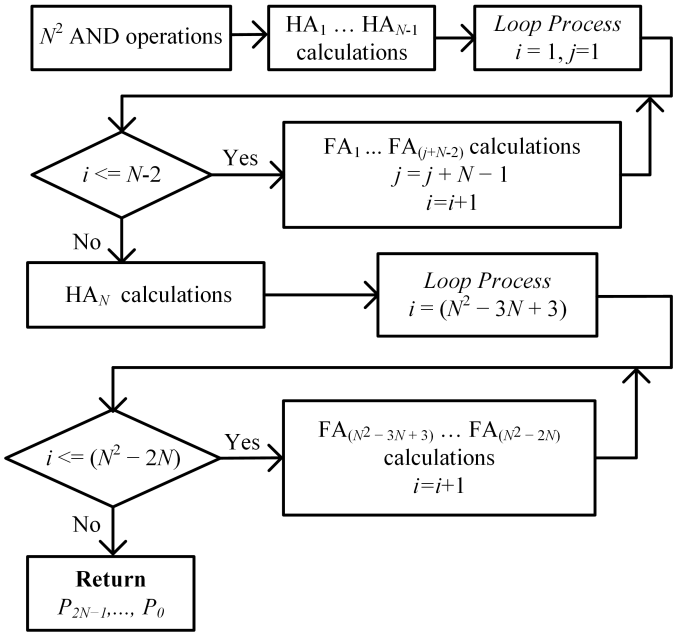


Fig. 5: Flowchart of the first proposed $N \times N$-bit array multiplier.

D. Second Proposed Multiplier

The first proposed multiplier can significantly be improved in terms of area by reducing the number of required memristors. This design as the second proposed Braun-type array multiplier is obtained by reusing some of the FAs instead of using more FAs in different rows of the array multiplier. Based on Fig. 4, two middle rows and in general $N - 2$ rows in $N \times N$-bit array multiplier execute similar operations. Therefore, the FAs of the second row can be reset (re-initialized) using a reset pulse to be able to perform the operation of the third row, as well, after finishing the operation of the second row. Then, these FAs should be reset again to start the operation of the fourth row, and this approach will be continued to the end of the $(N–1)$th row. The last row (Nth row) is different from the middle rows in a HA cell and also in sending the output carries to the left cells. However, the FAs of the second row can be reorganized to perform the operation of the last row, as well, which leads to more area reduction. The memristors of the second row that are reused are the ones that save the state of HS, HC and int of the FAs. These memristors in the first proposed multiplier are unused after the end of a FA operation. This approach acts similar to the pipeline structure in ordinary circuits but it does not require extra registers to save the intermediate results. This leads to more area reduction.

In the second proposed array multiplier, only the memristors of the FAs in the second row are re-initialized and reused as described above. This leads to a significant reduction in the number of memristors from $4N^2 - 2N$ to $N^2 + 7N - 5$ in $N \times N$-bit multiplication. In fact, similar to the first proposed multiplier, $2N$ memristors are required for two N-bit inputs and N^2 memristors are consumed for N^2 AND gates for partial products generation. Each HA needs two additional memristors and thus in total, $2N - 2$ memristors are required. Since each FA needs three memristors and in total, $N - 1$ FAs are used, $3N–3$ memristors are required for the FAs. Therefore, a total sum of $N^2 + 7N - 5$ memristors is required in $N \times N$-bit multiplication.

Algorithm 2 shows how to extend the second proposed multiplier for $N \times N$-bit multiplication. It should be noted that the flowchart of the second proposed multiplier is very similar to that of the first one. The main difference is related to the first loop process in which if the answer is yes for the decision box, auxiliary memristors inside FA_1 to FA_{N-1} will be re-initialized. This multiplier requires the computational steps equal to $10N - 14$, which has $N - 2$ computational steps more than that of the first proposed multiplier (less than 11% increase in delay while achieving up to 72% area reduction for $N <= 64$ bits). These extra steps are the number of needed re-initializations for the second row's memristors.

Similar to the first proposed multiplier, the second design needs $4N^2 - 4N$ switches. As the base structure of both proposed multipliers is the same, the same number of switches is required. The reset pulses are applied by the control circuit

TABLE III: PARAMETER SETUP USED IN THE VTEAM MODEL.

Parameter	v_{off}	v_{on}	α_{off}	α_{on}	R_{off}	R_{on}
Value	0.7 V	-10 mV	3	3	1 MΩ	10 kΩ
k_{off}	k_{on}	w_{off}	w_{on}	w_C	a_{off}	a_{on}
1 cm/s	-0.5 nm/s	0 nm	3 nm	100 pm	3 nm	0 nm

TABLE IV: PARAMETER SETUP USED IN THE LOGIC GATES.

Logic \ Parameter	V_{cond} (V)	V_{set} (V)	V_x (V)	V_{reset} (V)	R_G ($k\Omega$)	Pulse duration (μs)
TMSL-AND	1.3	0.6	-	-	13	2
TMSL-NAND	0.6	1.3	-	-	3.9	2
SIXOR	-	-	1.3	-1.5	-	2

which does not add additional switches in the second proposed multiplier.

V. SIMULATION RESULTS

A. Simulation Setup

The proposed FA and multiplier designs were validated through simulations in LTSpice. Voltage-controlled ThrEshold Adaptive Memristor (VTEAM) model [38] is used to model the behavior of memristors in SPICE [39], [31]. Table III shows the parameter values for the VTEAM model. It should be noted that these parameters are obtained by fitting the VTEAM model to our measurements [39] of Knowm "BS-AF-W" discrete memristors [40]. In addition, the considered values for the parameters of the utilized logic gates are shown in Table IV. The parameter setting of SIXOR is based on the referenced values of [23]. TMSL is also validated with different parameter values considering the conditions of the referenced paper [14]. For all experiments, the pulse duration is 2μs with the rise time and fall time that both are set to 5% of the pulse duration. It should be noted that L and H are assumed as the '1' and '0' logic values, respectively. The normalized state greater than 0.5 is assumed as '1', and the value lower than or equal to 0.5 is assumed as '0'.

The built-in measurement tool of LTSpice has been used to evaluate the energy and power consumption of the memristors in the proposed designs. Because the power dissipation of the circuits depends on the states of input memristors, the power consumption is averaged over all input combinations. It should be noted that the memristive technology (or the model) has a direct impact on the reported power and energy consumption. Thus, concerning power and energy, the proposed designs can be compared only to the prior works that have used the same technology or simulated with the same model.

B. Proposed FA

The proposed FA correctly produces the output carry and sum using eight memristors in four steps at 40 μs since for better presentation the 2 μs pulses are applied every 10 μs. Please note that for better visibility in the figures, the circuit starts the operation at a time equal to 10 μs in Figs. 6 to 8. Fig. 6 illustrates the output results of FA for $in1 = 1$, $in2 = 1$ and $C_{in} = 1$, which leads to $C_{FA} = 1$ and $S_{FA} = 1$. As shown in Fig. 6, applying the level correction on HS at Step 2 (between 20 μs and 30 μs) does not make a significant change in its state and it remains in H. Since HS is the input for

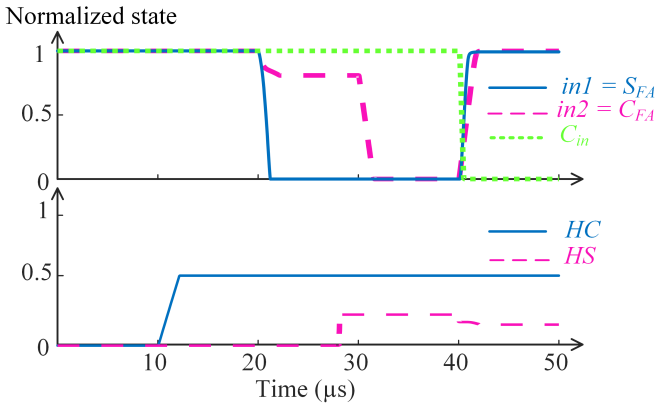


Fig. 6: Simulation of the proposed FA for $in1 = in2 = C_{in} = 1$, leading to $C_{FA} = S_{FA} = 1$.

TMSL-NAND and SIXOR, based on our simulations, a small change above does not affect subsequent operations or their final outputs, and these gates consistently produce outputs with the correct logic levels. Importantly, this change does not lead to cumulative errors, as any potential deviation is neutralized at the start of its subsequent step. For the pulse duration of 2 μs, the proposed FA has an average power consumption of 60.23 μW in eight combinations of inputs (cases or states). The average energy consumption is equal to 0.53 nJ.

C. First proposed multiplier

Fig. 7 shows the simulation of the 4×4-bit multiplication based on the first proposed multiplier for $a_{3-0} = 1011$ and $b_{3-0} = 1001$ as the input operands. The output product is correctly obtained as $P_{7-0} = 01100011$ after 240 μs or in 24 steps. As an example, level corrections applied to the inputs of P_2 according to Table II cause a small change in P_2 during Step 6, as shown in Fig. 7. However, this change does not alter its H state, and P_2 finally reaches its correct value (strong '0') in Step 10. For the pulse duration of 2 μs in each step, the first proposed 4×4-bit multiplier consumes an average power of 111.2 μW and an average energy of 5.87 nJ for all 256 input combinations.

D. Second proposed multiplier

Fig. 8 illustrates the simulation of the 4×4-bit multiplication based on the second proposed multiplier with the input operands of $a_{3-0} = 1110$ and $b_{3-0} = 1100$ to produce $P_{7-0} = 10101000$ after 260 μs in 26 steps. For the pulse duration of 2 μs in each step, the second proposed 4×4-bit multiplier consumes the average power of 107.5 μW and the average energy of 6.15 nJ for all 255 input combinations.

VI. COMPARISON AND DISCUSSION

The basic performance metrics used in this paper for comparison with the previous designs are the number of computational steps (delay), and the number of memristors (area). The number of switches is also used as another metric. It should be noted that the number of switches or drivers does not affect the merits of the memristive circuits as much as delay and area because the Back End Of Line (BEOL) process is usually used to implement memristors in which CMOS switches can

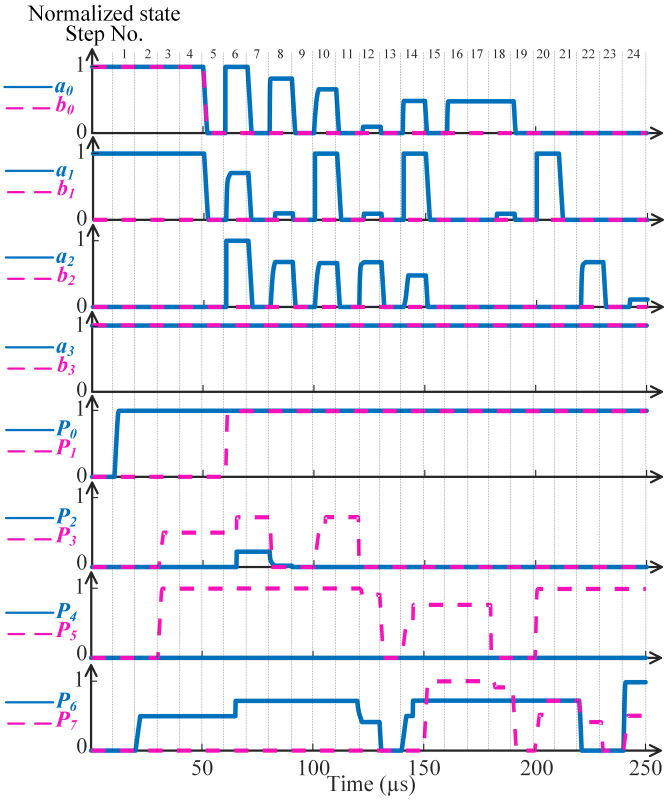


Fig. 7: Simulation of first proposed array multiplier (4×4-bit) with $a_{3-0} = 1011$ and $b_{3-0} = 1001$ leading to $P_{7-0} = 01100011$.

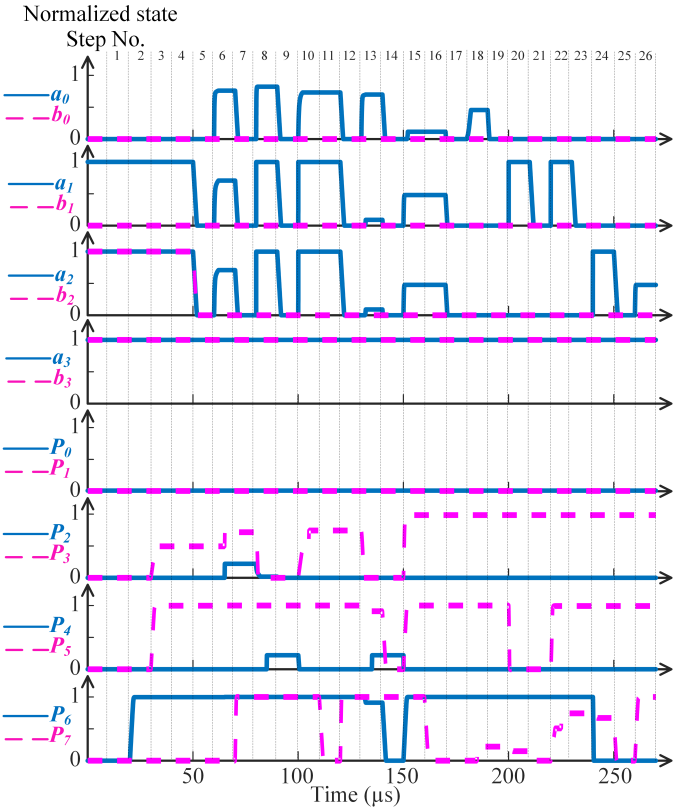


Fig. 8: Simulation of second proposed array multiplier (4×4-bit) with $a_{3-0} = 1110$ and $b_{3-0} = 1100$ leading to $P_{7-0} = 10101000$.

be implemented underneath the memristor crossbar [41]–[44]. Therefore, it only affects the chip area to some extent.

It should be noted that the VTEAM model we have used was fitted to Knowm discrete devices which is for single memristors, not an Integrated Circuit (IC). Substantial parasitics of discrete devices increase their delay and energy consumption, considerably compared to integrated devices. Consequently, the absolute numbers of delay or energy consumption cannot be directly compared to IC designs. Technological parameters affect the delay and power consumption of the proposed designs as well as other designs. Hence, for a fair comparison of memristive circuits, the same memristive technology should be used to simulate (and implement) the circuits that need to be compared, and the circuits with different memristive technologies cannot be directly compared.

A. *Figures of Merit*

In addition to the main metrics (the number of computational steps, memristors and switches), we consider four figures of merits (Figures of Merit (FoMs)) as defined in [24] to better compare different designs.

Balanced FoM (FoM_B) is shown in Eq. (1) assuming an equal weight for the number of memristors (N_M) and the number of computational steps (N_S):

$$FoM_B = \frac{1}{N_M \times N_S} \tag{1}$$

When the area is a more important factor compared to the speed, memristor-centered FoM (FoM_M) can be used as

$$FoM_M = \frac{1}{N_M^2 \times N_S} \tag{2}$$

in which N_M has a higher effect.

Similarly, speed is a more important factor in the speed-centered FoM (FoM_S):

$$FoM_S = \frac{1}{N_M \times N_S^2} \tag{3}$$

FoM_C is computed using Eq. (4) to evaluate the impact of required CMOS switches, where N_C denotes the number of switches. To prevent the infinite value for the FoM_C when no CMOS switch is used, 1 is added to N_C.

$$FoM_C = \frac{1}{N_M \times N_S \times (1 + N_C)} \tag{4}$$

Based on the defined FoMs, always a larger FoM represents more merit. Eq. (5) can be used to obtain the improvement percentage of the proposed designs compared to other designs:

$$Imp.(\%) = \frac{F_{better} - F_{worse}}{F_{worse}} \times 100 \tag{5}$$

In Eq. (5), F_{better} and F_{worse} are the FoMs of better and worse designs, respectively. This equation can also be used to obtain the improvements over other designs with respect to the basic performance metrics (i.e., the number of memristors, steps and switches). However, for the basic metrics, the sign of the obtained *Imp.* should be reversed because a lower number of memristors, steps and switches represent a more efficient design.

B. *FA Comparison*

A list of stateful FAs is represented in Table V. The proposed FA along with the SIXOR-based FA of [23] has the

TABLE V: SUMMARY OF STATEFUL FULL ADDERS.

Designs	# of mem.	# of steps	FoM$_B$	FoM$_M$	FoM$_S$
Iterative [37]	8	18	7m	0.9m	0.4m
IMPLY serial [34]	5	22	9.1m	1.8m	0.4m
IMPLY parallel [35]	5	21	9.5m	1.9m	0.4m
FELIX [22]	9	6	18.5m	2.1m	3.1m
Semi-serial [31]	8	12	10.4m	1.3m	0.9m
Semi-parallel [36]	5	17	11.7m	2.3m	0.7m
ORNOR [47]	8	17	7.3m	0.9m	0.4m
Cascading logic [48]	7	13	11m	1.5m	0.8m
SIXOR-based [23]	9	4	27.7m	3.1m	6.9m
MIMO parallel [45]	5	10	20m	**4m**	2m
IMPLY-based CSA [49]	16	30	2.1m	0.1m	0.1m
Cascading logic [46]	10	9	11.1m	1.1m	1.2m
Proposed	8	4	**31.2m**	3.9m	**7.8m**

TABLE VI: SUMMARY OF STATEFUL *N*-BIT ADDERS.

Designs	# of mem.	# of steps	$N = 32$		
			FoM$_B$	FoM$_M$	FoM$_S$
Iterative [37]	$8N$	$21N-3$	5.8μ	0.02μ	0.001μ
IMPLY serial [34]	$2N+3$	$22N$	21.2μ	0.3μ	0.03μ
IMPLY serial [35]	$2N+3$	$23N$	20.3μ	0.3μ	0.03μ
IMPLY parallel [35]	$4N+1$	$5N+16$	44μ	0.3μ	0.2μ
Semi-serial [31]	$2N+6$	$10N+2$	44.4μ	0.6μ	0.1μ
Semi-parallel [36]	$2N+3$	$17N$	27.4μ	0.4μ	0.05μ
ORNOR [47]	$6N+6$	$2N+15$	63.9μ	0.3μ	0.8μ
SIXOR-based [23]	$6N+3$	$2N+2$	77.7μ	0.4μ	1.2μ
MIMO parallel [45]	$5N$	$N+9$	$\mathbf{152.4\mu}$	0.9μ	$\mathbf{3.7\mu}$
IMPLY-based CSA [49]	$19\frac{N}{2}+6$	$3N+27$	26.2μ	0.1μ	0.2μ
Proposed	$2N+6$	$4N$	111.6μ	$\mathbf{1.6\mu}$	0.9μ

lowest delay among all FAs. These FAs are on average, 4.2 times faster than the others. The proposed FA, compared to the SIXOR-based FA of [23] (as the fastest existing design), requires a lower number of memristors. Moreover, all previous FAs that have a lower number of memristors compared to the proposed FA, incur much more delay. Altogether, the proposed FA is the best design among all existing FAs with respect to FoM$_B$ and FoM$_S$, and lags only in terms of FoM$_M$ compared to [45] as shown in Table V. In this table, the best FoMs are shown as bold numbers. In practice, a single FA is rarely used. Therefore, the number of steps and the number of memristors for the *N*-bit simple adders are shown in Table VI along with three main FoMs for $N = 32$ as an example. For larger *N*s, the proposed adder needs a lower number of memristors compared to the fastest existing design in [23] and requires only 3 memristors more than the compactest designs. Altogether, the *N*-bit adder constructed by the proposed FA is among the best adders with respect to the main FoMs. It should be noted that the designs from [45] and [46] are based on different memristor models compared to that of our proposed designs.

C. Multipliers Comparison

Tables VII and VIII show the main performance metrics of two proposed stateful array multipliers in comparison with previous stateful multipliers for 8×8-bit, and 64×64-bit multipliers (*N* equal to 8 and 64), respectively. These tables also depict the improvements of two proposed multipliers in percent compared to previous designs. *Imp.*$_{(1)}$ and *Imp.*$_{(2)}$ stand for the improvement of the first and the second proposed multipliers, respectively. It should be noted that a positive result shows an improvement over an older design. However, a negative result means that the older design has a better performance metric compared to the newer design.

According to Tables VII and VIII, the optimization of memristor reuse in the second proposed multiplier leads to 52.1% and 72.1% reduction in the number of memristors compared to the first proposed multiplier for 8×8-bit and 64×64-bit multiplications, respectively. As a result, the second proposed multiplier has the lowest number of memristors compared to all existing array and parallel multipliers except that of [53]. The area improvement of the first and the second proposed multipliers over the previous array and parallel multipliers is up to 42% and 72.2%, respectively, for 8×8-bit multipliers. This means that up to 1.7× and 3.6× memristor reductions are obtained respectively, for the first and the second proposed multipliers compared to previous designs in the literature. For 64×64-bit multipliers (Table VIII), the area improvement of the first and the second proposed multiplier over previous designs are up to 42.3% and 83.9%, respectively, which means reduction in the number of memristors up to 1.7× and 6.2× for the first and the second proposed multipliers compared to previous designs.

Based on Tables VII and VIII, the delay, i.e., the number of computational steps, is highly improved in both proposed designs. The first proposed multiplier is the fastest design and the second proposed multiplier is the second fastest design among all comparable multipliers in the literature. For 8×8-bit multipliers (Table VII), the first proposed design achieves from 1.5× to 24.5× higher speed compared to previous designs. These improvements are 1.4× to 22.3× for the second proposed multiplier. Also, for 64×64-bit multipliers (Table VIII), the first proposed design is faster from 1.3× to 192.4× compared to previous designs. Similarly, for the second proposed multiplier, the improvements are between 1.2× to 173.4×. Thus, compared to the state-of-the-art multipliers such as MultPIM [53] that could reduce the number of memristors, the proposed designs are the fastest and therefore can be the top choice for applications where the speed is more important than the cost.

The effect of multiplier size *N* on the cost and delay (number of memristors, steps and switches) of different multipliers is shown in Fig. 9 for *N* from 4 to 64 bits. Based on Fig. 9 (a), the second proposed multiplier is always better than all array and parallel multipliers with respect to the number of memristors except that of [53]. In addition, it will have an increasing improvement over the state-of-the-art designs when the multiplier size increases. The first proposed multiplier requires a lower number of memristors in all multiplier sizes only compared to Array [27]. However, based on the zoomed diagram, it also consumes a lower number of memristors compared to Array [28] for $N < 30$. With respect to other designs, the main point is that Array [28] does not operate well for $N < 9$ since it needs the highest number of memristors compared to all existing designs.

According to Fig. 9 (b), two proposed multipliers have the highest speed in comparison with the previous multipliers in all multiplier sizes. The improvement of the first and the second proposed multipliers over the best of previous designs with respect to delay, i.e., Array [28], approaches to 25% and 17%, respectively, for larger multipliers sizes even beyond $N = 64$. Concerning other designs, it is worth mentioning that the curve for the multiplier of [24] crosses the curve for Shift&Add [26], several times for $N < 20$ and the curve of

TABLE VII: COMPARISON OF STATEFUL MULTIPLIERS FOR $N = 8$.

Designs	Number of memristors				Number of steps				Number of switches			
	Total	$N=8$	Imp.$_{(1)}$ (%)	Imp.$_{(2)}$ (%)	Total	$N=8$	Imp.$_{(1)}$ (%)	Imp.$_{(2)}$ (%)	Total	$N=8$	Imp.$_{(1)}$ (%)	Imp.$_{(2)}$ (%)
Shift&Add [26]	$7N+1$	57	-76.2	-50.4	$2N^2+21N$	296	79.7	77.7	$8N-1$	63	-71.9	-71.9
Array [27]	$7N^2-8N+9$	393	38.9	70.7	$24N-35$	157	61.8	58	$8N^2-8N+9$	457	51	51
Dadda [30]*	NA	385	37.7	70.1	NA	106	43.4	37.7	NA	482	53.5	53.5
Semi-serial [24]	$2N^2+N+2$	138	-42.5	16.7	$\lceil\log_2 N\rceil(10N+2)$ $+4N+2$	280	78.6	76.4	$12\lceil\frac{N}{2}\rceil+\lfloor\frac{N-1}{2}\rfloor$	51	-77.2	-77.2
Array [28]	$3N^2+28N-2$	414	42	72.2	$12N-6$	90	33.3	26.7	N^2	64	-71.4	-71.4
Serial [50]	N^2+2	66	-72.5	-42.6	$27N^2-32N$	1472	95.9	95.5	-	-	-	-
MAT-based [51]	$2N^2+3N$	152	-36.6	24.3	N^2+8N-8	120	50	45	$4N$	32	-85	-85
Semi-parallel [52]	$\lceil\frac{N}{2}\rceil(4N+3)$	140	-41.6	17.9	$\lceil\log_2 N\rceil(\frac{39}{2})N+5N+2$	510	88.2	87	$8\lceil\frac{N}{2}\rceil+\lfloor\frac{N-1}{2}\rfloor+2N^2$ $-N+3(\lceil\log_2 N\rceil-1)$	161	-28.1	-28.1
MultPIM [53]	$14N-7$	105	-56.2	-8.7	$N\log_2 N+14N+3$	139	56.8	52.5	-	-	-	-
1st proposed	$4N^2-2N$	240	-	52.1	$9N-12$	60	-	-9.1	$4N^2-4N$	224	-	0
2nd proposed	N^2+7N-5	115	-52.1	-	$10N-14$	66	9.1	-	$4N^2-4N$	224	0	-

- *Imp.*$_{(1)}$ and *Imp.*$_{(2)}$ stand for the improvement of the first and the second proposed multipliers, respectively.
* The reference source does not provide equations.

TABLE VIII: COMPARISONS OF STATEFUL MULTIPLIERS FOR $N = 64$.

Designs	Number of memristors				Number of steps				Number of switches			
	Total	$N=64$	Imp.$_{(1)}$ (%)	Imp.$_{(2)}$ (%)	Total	$N=64$	Imp.$_{(1)}$ (%)	Imp.$_{(2)}$ (%)	Total	$N=64$	Imp.$_{(1)}$ (%)	Imp.$_{(2)}$ (%)
Shift&Add [26]	$7N+1$	449	-97.2	-90.1	$2N^2+21N$	9536	94.1	93.4	$8N-1$	511	-96.8	-96.8
Array [27]	$7N^2-8N+9$	28169	42.3	83.9	$24N-35$	1501	62.4	58.3	$8N^2-8N+9$	32265	50	50
Dadda [30]*	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
Semi-serial [24]	$2N^2+N+2$	8258	-49.2	45	$\lceil\log_2 N\rceil(10N+2)$ $+4N+2$	4110	86.3	84.8	$12\lceil\frac{N}{2}\rceil+\lfloor\frac{N-1}{2}\rfloor$	415	-97.4	-97.4
Array [28]	$3N^2+28N-2$	14078	-13.4	67.7	$12N-6$	762	26	17.8	N^2	4096	-74.6	-74.6
Serial [50]	N^2+2	4098	-74.8	-9.7	$27N^2-32N$	108544	99.5	99.4	-	-	-	-
MAT-based [51]	$2N^2+3N$	8384	-48.4	45.9	N^2+8N-8	4600	87.7	86.4	$4N$	256	-98.4	-98.4
Semi-parallel [52]	$\lceil\frac{N}{2}\rceil(4N+3)$	8288	-49	45.2	$\lceil\log_2 N\rceil(\frac{39}{2})N+5N+2$	7810	92.7	92	$8\lceil\frac{N}{2}\rceil+\lfloor\frac{N-1}{2}\rfloor+2N^2$ $-N+3(\lceil\log_2 N\rceil-1)$	8430	-47.7	-47.7
MultPIM [53]	$14N-7$	889	-94.5	-80.4	$N\log_2 N+14N+3$	1283	56	51.2	-	-	-	-
1st proposed	$4N^2-2N$	16256	-	72.1	$9N-12$	564	-	-9.9	$4N^2-4N$	16128	-	-
2nd proposed	N^2+7N-5	4539	-72.1	-	$10N-14$	626	9.9	-	$4N^2-4N$	16128	-	-

- *Imp.*$_{(1)}$ and *Imp.*$_{(2)}$ stand for the improvement of the first and the second proposed multipliers, respectively.
* The reference source provides neither equations nor numbers for $N = 64$.

MAT-based [51] for $N = 57$. However, it highly outperforms over Shift&Add [26] for larger multiplier sizes. Also, semi-parallel [52] outperforms Shift&Add [26] with respect to speed only after $N = 50$.

As mentioned before, the in-array structures of both proposed multipliers need the same number of additional switches. Therefore, based on Tables VII and VIII, the proposed multipliers require around $2\times$ lower switches compared to Array [27] and Dadda [30] multipliers for 8×8-bit and 64×64-bit multipliers. However, the proposed multipliers require more switches compared to some of the previous multipliers. As shown in Fig. 9 (c), the curves showing the number of switches for the multipliers of [24], [26] and [51] increase very closely while they are lower than that of the proposed multipliers. Array [27] is the worst among all existing designs according to its drastic growth when the multiplier size increases. On the other hand, MAT-based [51] is the best design for $N <= 64$.

To achieve a deeper view with respect to the situation of the proposed multipliers compared to different multipliers, the introduced FoMs are computed for 8×8-bit as shown in Table IX. The highest values of FoMs are boldfaced to show the best design. This table also depicts the improvements of two proposed multipliers in percent compared to previous designs and each other.

Considering $N = 8$ in Table IX, the second proposed multiplier achieves the best FoM$_B$, FoM$_M$ and FoM$_S$, i.e., three out of four FoMs among all designs. Moreover, the first proposed multiplier has a better FoM$_B$ and FoM$_S$ compared to all previous designs. The amounts of improvements in these FoMs are also high. For example, the second proposed

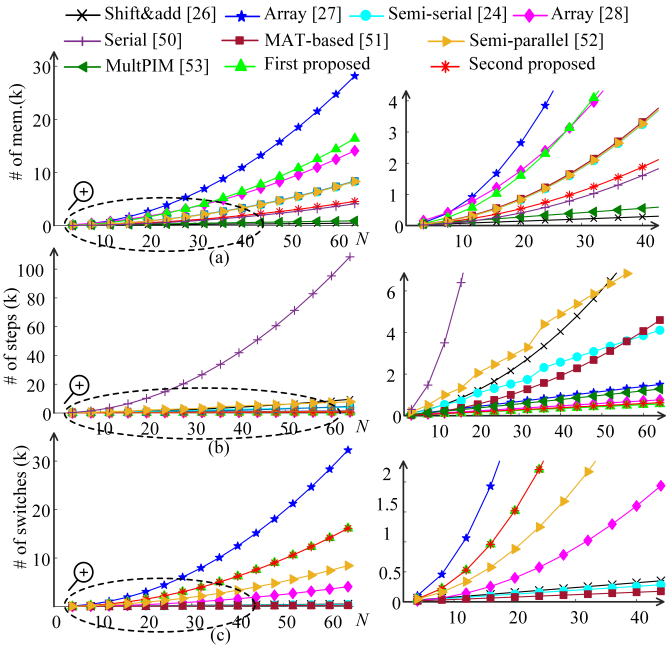


Fig. 9: Effect of multiplier size N on the number of (a) memristors (b) steps, and (c) switches for $N \times N$-bit multipliers based on $N = 4$ to 64 bits.

multiplier achieves an improvement between $1.1\times$ (compared to Shift&Add [26]) and $27.8\times$ (compared to Array [27]) concerning the memristor-centered FoM (FoM$_M$). Similarly, its improvement is between $4\times$ and $285.2\times$ compared to previous designs with respect to the speed-centered FoM (FoM$_S$). With respect to FoM$_C$, the second proposed multiplier is better compared to all array and parallel multipliers. It is only not

TABLE IX: COMPARISON OF STATEFUL MULTIPLIERS BASED ON THE FIGURES OF MERITS FOR $N = 8$. BOLD NUMBERS DENOTE THE BEST DESIGN ACCORDING TO EACH FIGURE OF MERIT.

Designs	FoM_B			FoM_M			FoM_S			FoM_C		
	#	Imp.$_{(1)}$ (%)	Imp.$_{(2)}$ (%)	#	Imp.$_{(1)}$ (%)	Imp.$_{(2)}$ (%)	#	Imp.$_{(1)}$ (%)	Imp.$_{(2)}$ (%)	#	Imp.$_{(1)}$ (%)	Imp.$_{(2)}$ (%)
Shift&Add [26]	59.3μ	17	122.1	1039.8n	-259.4	10.2	200.2n	478.1	897.1	926.1n	-200.1	-58.1
Array [27]	16.2μ	328.4	713	41.2n	602.2	2680.8	103.2n	1021.5	1834.3	35.4n	771.7	1554.2
Dadda [30]	24.5μ	183.3	437.5	63.6n	354.9	1701.4	231.2n	400.6	763.4	50.7n	508.7	1055
Semi-serial [24]	25.9μ	167.9	408.5	187.5n	54.3	511	92.4n	1152.6	2060.4	497.7n	-61.3	17.7
Array [28]	26.8μ	158.9	391.4	64.8n	346.4	1668.1	298.2n	288.1	569.4	412.9n	-33.8	41.8
Serial [50]	10.3μ	573.8	1178.6	155.9n	85.6	634.4	7n	16434.3	28417.1	-	-	-
MAT-based [51]	54.8μ	26.6	140.3	360.7n	-24.7	217.6	456.8n	153.4	337	**1661.3n**	-438.3	-183.7
Semi-parallel [52]	14μ	395.7	840.7	100n	189.3	1045.7	27.5n	4108.7	7158.9	86.4n	257.2	577.8
MultPIM [53]	68.5μ	1.3	92.3	652.5n	-116.2	83.2	492.9n	134.8	305	-	-	-
1st proposed	69.4μ	-	89.8	289.3n	-	296	1157.4n	-	72.5	308.6n	-	89.8
2nd proposed	**131.7μ**	-89.8	-	**1145.7n**	-296	-	**1996.2n**	-72.5	-	585.6n	-89.8	-

- *Imp.*$_{(1)}$ and *Imp.*$_{(2)}$ stand for the improvement of the first and the second proposed multipliers, respectively.

better than Shift&Add [26] and MAT-based [51] multiplier. The first proposed multiplier is better than Array [27], Dadda [30] and semi-parallel [52] multipliers.

D. Discussions

The obtained values of different FoMs along with the basic metrics can help the designers to choose the required multiplier based on the design goals and constraints. The proposed array multipliers in this paper are better in most of the basic performance metrics and most of the investigated FoMs compared to state-of-the-art designs. Thus, they can be chosen as a beneficial structure especially the second proposed multiplier.

We note that memristive technology is still in a maturing phase and so are the respective models. Consequently, there is a gap between the simulation and implementation results due to the effect of non-idealities, especially those not reflected in the models [54]. However, taping out design is at the moment very challenging and out of reach for most (including for us at the moment), especially for non-technical reasons. In the literature, we barely see any tape-out or measurement results, especially for any circuit that needs more than a handful of memristors [54]. Nevertheless, we have taken certain steps to close this gap and ensure a higher reliability and practical relevance of our design. First and foremost, we have used a model that has been experimentally fitted to real devices and thus includes some of the effects of non-idealities. Moreover, as device variation is one of the main challenges of memristive circuits, we have investigated the effect of memristance variation on the proposed designs. To do so, the proposed multipliers are simulated when the memristance changes from 5% to 20% which means that we analyzed cases where $8k\Omega < R_{on} < 12k\Omega$ and $800k\Omega < R_{off} < 1200k\Omega$. The simulations showed that this variation does not affect the correct operation of the designs. Concerning the power and energy consumption, it should be noted that the corresponding results are not reported for the previous multipliers. Moreover, the proposed multipliers can be compared to the circuits that are implemented using the same memristor model in the same technology. The memristor model and technology of the proposed multiplier in [24] are the same as ours but in [24] only the energy consumption of the semi-serial adder used in that multiplier was reported, and the results for the multiplier were not reported. However, we have estimated the energy consumption of the semi-serial adder-based multiplier in [24] based on the reported values for the utilized semi-serial adder. As a $(2N-1)$-bit semi-serial adder is replicated $\lceil \frac{N}{2} \rceil$ times for $N \times N$-bit multiplication, a 7-bit semi-serial adder is replicated two times for a 4×4-bit multiplier. Based on [24], the semi-serial adder consumes $(9.87N + 1.33)$ nJ which leads to 70.42 nJ for each adder. Hence, approximately 140.84 nJ is consumed by the 4×4-bit multiplier which is $24\times$ more than that of the first proposed multiplier and $22.9\times$ more than that of the second proposed multiplier. Therefore, the energy consumption of the proposed multipliers is considerably reduced compared to [24].

VII. CONCLUSIONS

In this paper, we proposed an efficient in-array FA which is based on a combination of SIXOR and TMSL stateful logics. This FA requires one fewer memristor compared to the best existing FA while having the same speed. The proposed FA was used as a basic building-block in the proposed array multipliers. More efficient reusing of the memristors in the proposed array multipliers than the previous designs as well as using the proposed low-cost and high-speed FA helped us to reach two new low-cost and high-speed multipliers suitable for in-array computing. The first and the second proposed multipliers achieved on average 29% and 21%, respectively, higher speed compared to the fastest existing design in [28]. Based on the introduced figures of merit, both proposed multipliers especially the second one achieved a higher performance compared to previous designs. The second proposed multiplier is the best design in three out of four figures of merit. For 8×8-bit multiplication, this multiplier achieved up to $12.8\times$, $27.8\times$ and $285.2\times$ improvement, in the balanced, memristor-centered and speed-centered figures of merit, respectively, compared to previous designs. Thus, the proposed usage of different single-cycle stateful memristive logic gates together (instead of using a single logic such as IMPLY) can lead to a faster in-array computation with a lower cost in a more complex unit such as a multiplier.

REFERENCES

[1] L. O. Chua. Memristor—the missing circuit element. *IEEE Transactions on Circuit Theory*, CT-18(5):507–519, September 1971.

[2] D. B. Strukov *et al.* The missing memristor found. *Nature*, 453:80–83, May 2008.

[3] M. A. Zidan *et al.* The future of electronics based on memristive systems. *Nature electronics*, 1:22–29, 2018.

[4] N. Taherinejad *et al.* Memristors' potential for multi-bit storage and pattern learning. In *2015 IEEE European Modelling Symposium (EMS)*, pp. 450–455, Oct 2015.

[5] N. Taherinejad *et al.* Fully digital write-in scheme for multi-bit memristive storage. In *2016 13th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE)*, pp. 1–6, Sept 2016.

[6] G. Papandroulidakis *et al.* Crossbar-based memristive logic-in-memory architecture. *IEEE Transactions on Nanotechnology*, 16(3):491–501, May 2017.

[7] M. F. Tolba *et al.* Memristor fpga ip core implementation for analog and digital applications. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 66(8):1381–1385, 2019.

[8] M. Huang *et al.* Global-gate controlled one-transistor one-digital-memristor structure for low-bit neural network. *IEEE Electron Device Letters*, 42(1):106–109, 2021.

[9] Y. Yu *et al.* Design of multilayer cellular neural network based on memristor crossbar and its application to edge detection. *Journal of Systems Engineering and Electronics*, 34(3):641–649, 2023.

[10] A. Horváth *et al.* Deep memristive cellular neural networks for image classification and segmentation. *IEEE Transactions on Nanotechnology*, pp. 1–8, 2024.

[11] M. R. Alam *et al.* Exact stochastic computing multiplication in memristive memory. *IEEE Design Test*, pp. 1–8, 2021.

[12] M. R. Alam *et al.* Sorting in memristive memory. *ACM Journal on Emerging Technologies in Computing Systems*, pp. 1–22, 2022.

[13] E. Lehtonen and M. Laiho. Stateful implication logic with memristors. In *2009 IEEE/ACM International Symposium on Nanoscale Architectures*, 2009.

[14] P. Huang *et al.* Reconfigurable nonvolatile logic operations in resistance switching crossbar array for large-scale circuits. *Advanced Materials*, 28(44):9758–9764, 2016.

[15] Q. Xu *et al.* Locally active memristor-based neuromorphic circuit: Firing pattern and hardware experiment. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 70(8):3130–3141, 2023.

[16] Q. Xu *et al.* Firing pattern in a memristive hodgkin–huxley circuit: Numerical simulation and analog circuit validation. *Chaos, Solitons Fractals*, 172:113627, 2023.

[17] C. Wang *et al.* Complementary digital and analog resistive switching based on alo_x monolayer memristors for mixed-precision neuromorphic computing. *IEEE Transactions on Electron Devices*, 70(8):4488–4492, 2023.

[18] M. Guo *et al.* Pruning and quantization algorithm with applications in memristor-based convolutional neural network. *Cognitive Neurodynamics*, 01 2023.

[19] Q. Xu *et al.* Extreme multistability and phase synchronization in a heterogeneous bi-neuron rulkov network with memristive electromagnetic induction. *Cognitive Neurodynamics*, 17(3):755–766, 2023.

[20] J. Borghetti *et al.* 'Memristive' switches enable 'stateful' logic operations via material implication. *Nature*, 464:873–876, April 2010.

[21] S. Kvatinsky et al. MAGIC; memristor-aided logic. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 61(11):895–899, Nov 2014.

[22] S. Gupta *et al.* FELIX: Fast and energy-efficient logic in memory. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, November 2018.

[23] N. TaheriNejad. SIXOR: single-cycle in-memristor XOR. *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, 29(5):925–935, 2021.

[24] D. Radakovits *et al.* A memristive multiplier using semi-serial imply-based adder. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(5):1495–1506, 2020.

[25] R. R. Disfani *et al.* Operational conditions analysis for memristive stateful logics-a study on imply and tmsl. In *2022 20th IEEE Interregional NEWCAS Conference (NEWCAS)*, pp. 480–484. IEEE, 2022.

[26] L. Guckert and E. E. Swartzlander. Optimized memristor-based multipliers. *CSI2017*, 64(2):373–385, February 2017.

[27] L. E. Guckert. *Memristor-based arithmetic units*. PhD thesis, University of Texas at Austin, 2016.

[28] N. Revanna *et al.* The future of computing—arithmetic circuits implemented with memristors. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pp. 745–749. IEEE, 2017.

[29] L. Dadda. Some schemes for parallel multipliers. *Alta frequenza*, 34:349–356, 1965.

[30] L. Guckert and E. E. Swartzlander. Dadda multiplier designs using memristors. In *ICICDT2017*, pp. 1–4. IEEE, May 2017.

[31] N. TaheriNejad *et al.* A semi-serial topology for compact and fast IMPLY-based memristive full adders. In *2019 IEEE New Circuits and Systems symposium (NewCAS)*, pp. 1–5, 2019.

[32] S. Kvatinsky *et al.* Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(10):2054–2066, October 2014.

[33] M. Teimoory et al. Optimized implementation of memristor-based full adder by material implication logic. In *ICECS2014*, pp. 562–565, 2014.

[34] S. G. Rohani and N. TaheriNejad. An improved algorithm for IMPLY logic based memristive full-adder. In *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1–4, April 2017.

[35] A. Karimi and A. Rezai. Novel design for a memristor-based full adder using a new imply logic approach. *Journal of Computational Electronics*, 17(3):1303–1314, Sep 2018.

[36] S. Ganjeheizadeh Rohani *et al.* A semiparallel full-adder in imply logic. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 28(1):297–301, Jan 2020.

[37] K. C. Rahman et al. Memristor based 8-bit iterative full adder with space-time notation and sneak-path protection. In *MWSCAS2017*, pp. 695–698. IEEE, 2017.

[38] S. Kvatinsky *et al.* VTEAM: A General Model for Voltage-Controlled Memristors. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 62(8):786–790, August 2015.

[39] D. Radakovits and N. TaheriNejad. Implementation and characterization of a memristive memory system. In *2019 IEEE 32nd Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1–5, May 2019.

[40] Knowm. Knowm inc, https://knowm.org., 2017.

[41] S. Bhat *et al.* Skynet: Memristor-based 3D IC for artificial neural networks. In *IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pp. 109–114, July 2017.

[42] H. Manem *et al.* An extendable multi-purpose 3D neuromorphic fabric using nanoscale memristors. In *IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA)*, pp. 1–8, May 2015.

[43] C. Li *et al.* Three-dimensional crossbar arrays of self-rectifying Si/SiO2/Si memristors. *Nature communications*, 8:15666, 2017.

[44] M. Hu *et al.* Memristor-based analog computation and neural network classification with a dot product engine. *Advanced Materials*, 30(9):1705914, 2018.

[45] M. Jiang *et al.* An efficient memristive alternating crossbar array and the design of full adder. *Nonlinear Dynamics*, 111(21):20331–20345, 2023.

[46] B. Li *et al.* Highly efficient reconfigurable stateful logic operations based on cui memristor-only arrays prepared with a solution-based process. *IEEE Journal of the Electron Devices Society*, 11:269–273, 2023.

[47] A. Siemon *et al.* Stateful three-input logic with memristive switches. *Scientific Reports*, 9, 12 2019.

[48] K. M. Kim and R. S. Williams. A family of stateful memristor gates for complete cascading logic. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(11):4348–4355, 2019.

[49] N. Kaushik and S. Bodapati. Imply-based high-speed conditional carry and carry select adders for in-memory computing. *IEEE Transactions on Nanotechnology*, 22:280–290, 2023.

[50] B. Bagheralmoosavi *et al.* Power-area efficient serial imply-based 4: 2 compressor applied in data-intensive applications. *arXiv preprint arXiv:2407.09980*, 2024.

[51] J. Sun *et al.* Efficient data transfer and multi-bit multiplier design in processing in memory. *Micromachines*, 15(6):770, 2024.

[52] W. Liang *et al.* An imply-based memristive multiplier for computing-in-memory systems with weight-stationary cnn acceleration. In *2022 IEEE International Conference on Integrated Circuits, Technologies and Applications (ICTA)*, pp. 112–113. IEEE, 2022.

[53] O. Leitersdorf *et al.* Multpim: Fast stateful multiplication for processing-in-memory. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(3):1647–1651, 2021.

[54] N. TaheriNejad and D. Radakovits. From behavioral design of memristive circuits and systems to physical implementations. *IEEE Circuit and Systems (CAS) Magazine*, 19(4):6–18, Fourthquarter 2019.