A Memristive Multiplier Using Semi-Serial IMPLY-Based Adder

David Radakovits, Nima TaheriNejad, *Member, IEEE*, Mengye Cai, *Student Member, IEEE*, Théophile Delaroche, and Shahriar Mirabbasi, *Member, IEEE*

***Abstract*— Memristors are among emerging technologies with many promising features, which makes them suitable not only for storage purposes but also for computations. In this work, focusing on in-memory computations, we first present our semi-serial IMPLY-based adder and perform an extensive analysis of its merits. In addition to providing a favorable balance between the number of steps and number of memristors, a key property of the presented adder is its compactness as compared to the state-of-the-art adders. Next, using our semi-serial adder, we propose an IMPLY-based multiplier. We show that the proposed multiplier is more than 5× better than other works based on the figure of merit which gives equal weight to the number of steps (i.e., speed) and required die area. Additionally, we provide a deeper insight into IMPLY-based arithmetic units, their properties, design characteristics, and advantages or disadvantages compared to one another by proposing new figures of merit and performing comprehensive comparative analyses. This facilitates the process of design, or selection, of suitable units for the design engineers and researchers in the field.**

***Index Terms*— Memristors, resistive RAM, memristive circuits, IMPLY, adder, full-adder, semi-serial, multiplier, multiplication circuit.**

I. Introduction

MEMRISTORS as memory elements have been widely researched [1]–[9] and are already used in commercial products [10], [11]. Given the advantages of memristive technologies, especially, in terms of size, speed, and power consumption, one of their emerging applications is their use as in-memory computational elements [3], [12]–[15]. In-Memory Computation (IMC) refers to performing computations inside memory, without performing any read or write operation for the purpose of the computation. This type of operation has many advantages; first, it saves on the energy required for read and write, as well as the transfer of the data to the processing unit. Next, it often enables massive parallelization of operations. Last, and perhaps the most important feature, is that it can alleviate or solve the Von-Neumann bottleneck problem, i.e., the speed and efficiency barrier that cannot be overcome solely with more better performing Central Processing Units (CPUs) and memory, due to the performance gap between the aforementioned units and the data bus that is shared among them [16]. These benefits have been shown in recent works such as [17]–[21].

Manuscript received September 1, 2019; revised November 8, 2019; accepted January 2, 2020. Date of publication January 23, 2020; date of current version May 1, 2020. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC). This article was recommended by Associate Editor B. Gosselin. *(Corresponding author: Nima TaheriNejad.)*

David Radakovits and Nima TaheriNejad are with the Department of Electrical Engineering and Information Technology, Institute of Computer Technology, TU Wien, 1040 Vienna, Austria (e-mail: nima.taherinejad@tuwien.ac.at).

Mengye Cai and Shahriar Mirabbasi are with the Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC V6T 1Z4, Canada.

Théophile Delaroche is with the Bordeaux Graduate School of Electronics, Computer Science, Telecommunications, Mathematics and Mechanics (ENSEIRB), University of Bordeaux INP, 33400 Bordeaux, France.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSI.2020.2965935

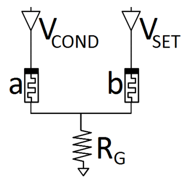


Fig. 1. Circuit-level implementation of IMPLY logic gate using memristors [32].

Adders and multipliers are crucial building blocks of Arithmetic Logic Units (ALUs), since virtually every arithmetic calculation on a computer system involves addition and multiplication. Efficiency of these two units nowadays is even more important given the widespread usage of computation-heavy machine learning applications such as Neural Networks (NNs). Therefore, in this paper, we focus on efficient full-adder design and show that the proposed efficient adder can be directly used in multipliers and offer performance that compares favorably with that of optimized multiplier designs. Among the wide range of memristor-based logic [21]–[30], we have chosen Material Implication (IMPLY) logic [31]–[33] because of its compatibility with IMC and crossbar structure [31], [32]. However, we note that IMPLY is not the only logic with these features, others such as Memristor-Aided Logic (MAGIC) [25] and Fast and energy-efficient Logic in Memory (FELIX) [21] have similar properties too.

In $a \rightarrow b$, the IMPLY operation leads to logic ‘0’ (High Resistance State (HRS) or R_{off}) at the output (which will be stored on b, over-writing its initial input value), only if, a has a logical value of ‘1’ (Low Resistance State (LRS) or R_{on}) and b has a logical value of ‘0’. To perform this operation in a memristive circuit, one connects memristor a and b as shown in Figure 1 and applies two fixed voltages, respectively V_{COND} and V_{SET}, to those memristors. Detailed information on IMPLY operation and how to select the two fixed voltages and resistor R_G can be found in [31]–[33].

The rest of the paper is organized as follows. In Section II, we review the State-of-the-Art (SoA) in IMPLY-based adders. In Section III, we review our proposed semi-serial adder [34]

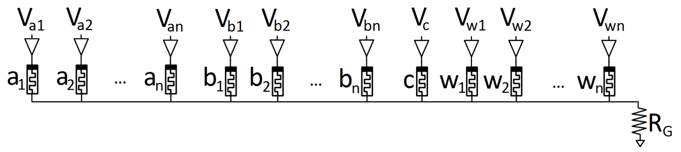

V_{a1} V_{a2} V_{an} V_{b1} V_{b2} V_{bn} V_c V_{w1} V_{w2} V_{wn}
a_1 a_2 ... a_n b_1 b_2 ... b_n c w_1 w_2 ... w_n
R_G

Fig. 2. Serial full-adder topology.

To be able to better evaluate the performance of the adder, we propose several metrics in Section IV, which take into account a range of constraints and considerations that take part in the decision-making process during design time. Based on these metrics, we perform a comprehensive comparative analysis of SoA adders in Section V and position our semi-serial design among them. In Section VI, we propose a multiplier based on our semi-serial adder and compare it with the SoA multipliers. We show that even though, in contrast to other SoA multipliers, we have not performed any optimization to develop the multiplier, due to the outstanding qualities of the semi-serial adder, the performance of the presented multiplier compares favorably with that of the SoA. Finally, we provide concluding remarks in Section VII.

II. Literature Review

IMPLY-based adders are traditionally designed in either serial or parallel fashion. However, these two topologies mark the opposite extremes of the continuum of possible adder designs, where each design represents a compromise between area, i.e., the number of necessary memristors (as well as additional Complementary Metal-Oxide Semiconductor (CMOS) circuitry), and speed, i.e., the number of necessary computational steps. This section provides an overview of such adders that are reported in the literature. A comprehensive comparative analysis is provided in Section V.

A. Serial Adders

In serial designs [33], [35]–[37] all memristors, i.e., input memristors, work memristors, and output memristors, are connected to a common node together with the working resistor, R_G (cf. Figure 2). Fully serial topology therefore consumes minimal space and has the lowest complexity, but at the same time needs the highest number of steps to calculate a sum. Therefore, the main focus for these designs has been reducing the number of steps by devising more efficient algorithms. Currently, $22n$ steps for an n-bit adder presented in [36] is the lowest number of steps reported in the literature for this kind of adders. This work presents another turning point in the respective literature by reducing the number of memristors required from $3n+3$ to $2n+3$. This feat is achieved by reusing the memristors of one of the input variables to store the output result. This clever design takes advantage of the fact that due to intrinsic properties of IMPLY operations, input memristors would lose their initial values in any case. Therefore, overwriting them with the output results does not cause any disadvantage.

B. Parallel Adders

On the other end of the spectrum are parallel topologies [33], [37]. In these topologies every bit of the first summand is located in a separate section together with its associated work memristors $w_{i,j}$, working resistor R_G, and the corresponding bit of the second summand (cf. Figure 3), i.e., each section calculates one bit of the sum. Parallel topologies are the fastest options for IMPLY-based adders, while they require the largest number of memristors. The smallest parallel design [37] in the literature uses $4n + 1$ memristors and calculates the sum in $5n + 16$ steps. This improvement in area is in part due to the reuse of the input memristors. The other notable technique is partitioning of the algorithm, in that the carry memristor is shared by all sections.

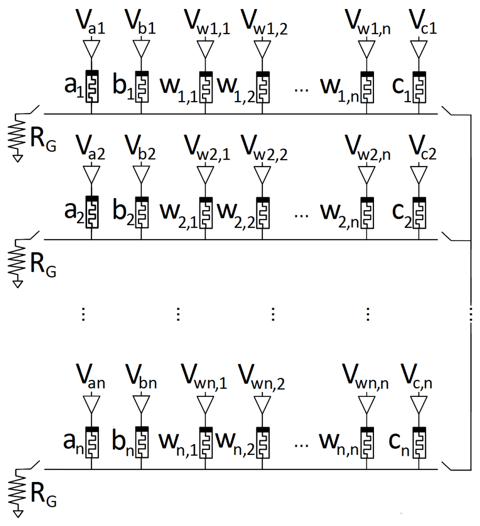

V_{a1} V_{b1} $V_{w1,1}$ $V_{w1,2}$ $V_{w1,n}$ V_{c1}
a_1 b_1 $w_{1,1}$ $w_{1,2}$ ... $w_{1,n}$ c_1
R_G
V_{a2} V_{b2} $V_{w2,1}$ $V_{w2,2}$ $V_{w2,n}$ V_{c2}
a_2 b_2 $w_{2,1}$ $w_{2,2}$ ... $w_{2,n}$ c_2
R_G
V_{an} V_{bn} $V_{wn,1}$ $V_{wn,2}$ $V_{wn,n}$ $V_{c,n}$
a_n b_n $w_{n,1}$ $w_{n,2}$ ... $w_{n,n}$ c_n
R_G

Fig. 3. Parallel full-adder topology.

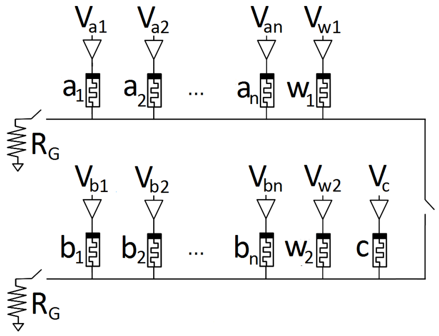
C. Iterative and Semi-Parallel Adders

As mentioned before, in designing memristor-based adders, there is still room for improvements, especially, when targeting applications in which neither speed nor size can be compromised. Therefore, some designs between the two extremes, i.e., fully serial and fully parallel structures, have been proposed. For instance, Rohani *et al.* [38], as shown in Figure 4, use two separate sections to parallelize independent steps within one bit, however, bits are run serially one after the other. This approach makes the semi-parallel adder more compact, since there are only two sections for any n-bit adder, as opposed to n-sections in the parallel designs. However, it is still not as fast as parallel designs, even though it requires less steps compared to serial designs.

Another work with less traditional structure is the ’Iterative’ design [39]. Even though its name implies sequential operations, the work shows more similarities to parallel designs, than serial designs. That is particularly noticeable in the very large number of memristors it needs for its operations which is a property shared by parallel designs. However, in terms of the number of steps, it is closer to serial designs (i.e., it is slower than parallel designs).

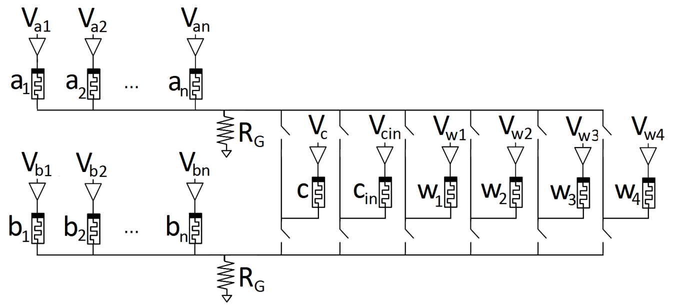
D. Multipliers

While there are several different IMPLY-based adder designs in the literature, the number of multiplier designs



V_{a1} V_{a2} V_{an} V_{w1}
a_1 a_2 ... a_n w_1
R_G
V_{b1} V_{b2} V_{bn} V_{w2} V_c
b_1 b_2 ... b_n w_2 c
R_G

Fig. 4. Multi-bit semi-parallel; Parallelization within one bit and calculating each bit after the other [38].



V_{a1} V_{a2} V_{an}
a_1 a_2 ... a_n
R_G
V_c V_{cin} V_{w1} V_{w2} V_{w3} V_{w4}
c c_{in} w_1 w_2 w_3 w_4
V_{b1} V_{b2} V_{bn}
b_1 b_2 ... b_n
R_G

Fig. 5. Topology of the previously proposed semi-serial full-adder.

is considerably lower. This scarce presence of IMPLY-based multipliers in the literature is remarkable, since multiplication is among the most important functions of ALUs (along with addition). In the following, we have highlighted IMPLY-based multiplier designs:

1) Shift&Add Multiplier: The multiplier in [40] uses the so-called Shift&Add scheme, in which the first factor is repeatedly shifted left and added to a sum register if the corresponding bit in the second factor is ‘1’, otherwise, it is shifted again immediately. Through reuse of registers and optimized IMPLY-based multiplexers and shift registers, the Shift&Add multiplier provides a good compromise between the number of memristors and computational steps. However, these optimizations are achieved through considerable addition of switches and consequently increase the CMOS overhead significantly, which is not fully considered in our comparisons given that actual CMOS overhead numbers which include those control logics are not available.

2) Array Multiplier: The array-type or parallel multiplier in [41] uses $n-1$ full adders for the implementation of an $n \times n$-bit multiplier, causing a high number of memristors. Since the carry bits have to be propagated through all adders, this scheme suffers from a relatively large delay too.

3) Dadda Multiplier: This multiplier in [42] uses the Dadda scheme for multiplication, which was proposed by L. Dadda in 1965 [43]. In this scheme the partial products are added in a distinct tree structure to gain speed but also area savings over the parallel multiplier.

III. Semi-Serial Adder

As mentioned in Section II, most IMPLY-based adder designs are from one of two types: serial or parallel. Whereas serial designs aim for a low number of memristors, thus sacrifice speed, parallel approaches are optimized for speed, but use a large number of work memristors. The goal of our semi-serial adder design is to combine properties of both serial and parallel approaches to achieve a better design with higher Figure of Merit (FoM).

A. Design

Contrary to serial or parallel designs, in our semi-serial adder the input variables a_i and b_i are located in two separate sections while five work memristors (w_1 to w_4 and c) and the carry-in memristor c_{in} are located in a separate section. Our design needs a total number of memristors of $2n+6$; $2n$ for input memristors and output variable (output will be written back on a), 4 work memristors (for any n) and two carry memristors. As can be seen in Figure 5, the input sections have their own work resistor R_G, and each memristor in the separate third section can be connected to either of the two input sections, a and b. The input section a and the carry-in memristor c_{in} are reused to store the resulting sum and carry. Equation (1) and Equation (2) show the formulation of sum and carry in IMPLY logic, which we use to implement the semi-serial adder.

$$S = \left[\overline{(\overline{a} \rightarrow b) \rightarrow \Big((a \rightarrow \overline{b}) \rightarrow c\Big)}\right] \rightarrow \left(\overline{\overline{(a \oplus b)} \rightarrow \overline{c}}\right) \tag{1}$$

$$C_{out} = \left[\overline{\Big((a \rightarrow \overline{b}) \rightarrow c\Big) \rightarrow \overline{(\overline{b} \rightarrow a)}}\right] \tag{2}$$

We refer to this structure as semi-serial, since each bit is calculated in a serial fashion (similar to serial topology), but the placement of work memristors in a separate third section enables parallelism beyond what is possible in the serial topology. The algorithm is spelled out in Table I. In the main body of the algorithm, $\overline{c}$ is used and propagated. This assumption is valid in the middle steps since the algorithm itself produces and propagates $\overline{c}$, however, in the beginning traditionally c_{in} is provided (not $\overline{c}$). Moreover, at the end, typically c_{out} is desired not $\overline{c_{out}}$. Hence, to conform with that, we consider one additional step for the inversion of c_{in} at the beginning and one additional inversion step at the end of the algorithm. These steps, which are run only once at the beginning and at the end of the execution of the algorithm, are highlighted in bold blue in Table I. Therefore, the overall number of steps is $10n+2$. Table II displays the connection status of memristors in the work memristors section (c, c_{in}, w_{1-4}) for every step in the algorithm (shown in Table I). A “U” implies that the memristor is connected to the upper section (a memristors), i.e., the upper switch is closed while the lower switch is open. Analogously, an “L” means that the respective memristor is connected to the lower section (b memristors), i.e., the upper switch is open while the lower switch is closed. A dash (“-”) shows a “don’t care” state, meaning that the memristor could be connected to either or neither of the sections, since the respective memristor is not

TABLE I

EXECUTION STEPS OF ADDITION IN OUR SEMI-SERIAL TOPOLOGY, REQUIRING $10n + 2$ STEPS, $2n + 6$ MEMRISTORS. WORK MEMRISTORS AND c CAN CONNECT TO ANY OF THE TWO SECTIONS AND $c = \overline{c_{in}}$. OPERATIONS IN BOLD BLUE (UNNUMBERED STEPS) ARE PERFORMED ONLY FOR THE VERY FIRST AND VERY LAST BIT OF THE OPERATION

Steps	Operation Executed in:		Equivalent Logic	
	Section 1	Section 2	Section 1	Section 2
1	$\mathbf{c = 0}$ **&** $w_1 = w_2 = 0$	$w_3 = w_4 = 0$	**False(c) &** False(w_1, w_2)	False(w_3, w_4)
-	$\mathbf{c_{in} \to c}$		$\mathbf{c = \overline{c_{in}}}$	
2	$a \to w_1 = w_1'$	$b \to w_3 = w_3'$	$w_1' = \overline{a}$	$w_3' = \overline{b}$
3	$a \to w_3' = w_3''$	$w_1' \to b = b'$	$w_3'' = a \to \overline{b}$	$b' = \overline{a} \to b$
4	$c \to w_2 = w_2'$	$w_3'' \to w_4 = w_4'$	$w_2' = \overline{\overline{c}} = c$	$w_4' = \overline{a \to \overline{b}}$
5	$a = w_1 = 0$	$b' \to w_4' = w_4''$	False(a, w_1)	$w_4'' = (\overline{a} \to b) \to \overline{(a \to \overline{b})} = \overline{a \oplus b}$
6	$w_3'' \to w_2' = w_2''$	$w_4'' \to c = c'$	$w_2'' = (a \to \overline{b}) \to c$	$c' = \overline{a \oplus b} \to \overline{c}$
7	$c' \to a = a'$	$w_2'' \to w_1 = w_1'$	$a' = \overline{\overline{a \oplus b} \to \overline{c}}$	$w_1' = \overline{(a \to \overline{b}) \to c}$
8	$\mathbf{c_{in} = 0}$ **&** $c = w_3 = 0$	$b' \to w_2'' = w_2'''$	$\mathbf{False(c_{in})}$ **&** False(c, w_3)	$w_2''' = (\overline{a} \to b) \to \overline{[(a \to \overline{b}) \to c]}$
9	$w_1' \to w_3 = w_3'$	$b' \to c = c'$	$w_3' = (a \to \overline{b}) \to c$	$c' = \overline{\overline{a} \to b} = \overline{b} \to a$
10	$w_2''' \to a' = a''$	$w_3' \to c' = c''$	$a'' = \big((\overline{a} \to b) \to [(a \to \overline{b}) \to c]\big) \to$	$c'' = [(a \to \overline{b}) \to c] \to \overline{(\overline{b} \to a)} = \overline{C_{out}}$
			$\overline{\overline{a \oplus b} \to \overline{c}} = S(Sum)$	
-	$\mathbf{c \to c_{in}}$		$\mathbf{c_{in} = \overline{c} = C_{out}}$	

TABLE II

CONNECTION STATUS OF MEMRISTORS IN THE WORK SECTION FOR EVERY STEP OF THE ALGORITHM AS SHOWN IN TABLE I. 'U' MEANS UPPER SWITCH IS CLOSED, 'L' MEANS LOWER SWITCHED IS CLOSED, AND DASH ('-') REPRSENTS A "DON'T CARE" STATE MEANING THAT EITHER OR BOTH SWITCHES COULD BE OPEN OR CLOSED (THE MEMRISTOR IS IN INACTIVE AND COULD BE CONNECTED TO EITHER OF THE SECTIONS OR TO NONE)

Step	c	c_{in}	w_1	w_2	w_3	w_4
1	L	-	L	L	U	U
-	L	L	-	-	-	-
2	-	-	U	-	L	-
3	-	-	L	-	U	-
4	L	-	-	-	U	U
5	-	-	U	-	-	L
6	L	-	-	U	U	L
7	U	-	L	L	-	-
8	U	U	-	L	U	-
9	L	-	U	-	U	-
10	L	-	-	U	L	-
-	U	U	-	-	-	-

TABLE III

PARAMETERS FOR 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_{on}	k_{off}	w_{off}	w_{on}	w_C	a_{off}	a_{on}
-0.5 nm/s	1 cm/s	0 nm	3 nm	107 pm	3 nm	0 nm

used in that step (i.e., it is disconnected from supply voltage and is inactive).

B. Simulations

1) Setup: The design was tested through simulations in LTSpice. To model the behavior of memristors, we used the Voltage-controlled ThrEshold Adaptive Memristor (VTEAM) model [44] implemented in SPICE [34], [45]. The parameters used for the setup of the model can be found in Table III. Table IV shows the parameters used for the IMPLY logic.

2) Results: We could verify that our design can correctly perform an n-bit summation using $2n + 6$ memristors and the number of necessary steps is $10n + 2$. Simulating all input combinations resulted in expected behaviors in all cases.

TABLE IV

PARAMETERS FOR IMPLY LOGIC

Parameter	V_{SET}	V_{COND}	V_{RESET}	R_G	t_{pulse}
Value	1V	$900mV$	5V	40kΩ	30μs

Our simulations yielded a power consumption of 9.87nJ per bit. This figure does not include the inversion of the carry for the first and last bit. These inversions cause an additional overhead of 1.33nJ, which results in a total power consumption of $(9.87n + 1.33)$nJ. Power consumption and energy numbers were obtained using simulations in LTSpice and contain the average[1] consumption of memristors only. We note that the energy consumption of memristors can substantially differ based on the model (and technology) used. However, our numbers can provide a good base for comparison in the future, since the models we have used are available to the public and others can use them to simulate their own adders and obtain power consumption numbers that are comparable with ours. In Section III-C.2, we report the power and energy consumption of the CMOS control logic and the overall consumption.

Figure 6 shows the SPICE simulation of a 1-bit full addition using our semi-serial design. Figure 7 shows the simulation of a complete 4-bit addition. The input values are $a_{4-1} = 1011$, $b_{4-1} = 0100$, $c_{in} = 0 = \overline{c}$ and the results are correctly calculated as $a_{4-1} = 1111$, $c_{in} = 0 = \overline{c}$ and are available in memristors a_i and c_{in} at 1.26ms, which equals 42 steps.

C. Discussions

1) Crossbar Compatibility: Our semi-serial design requires external switches for the work memristor section

[1]Since the power consumption of a single IMPLY-gate depends on the logical state of the inputs, we averaged the energy consumption over the four possible input combinations and use that figure as the generic power consumption of the gate.

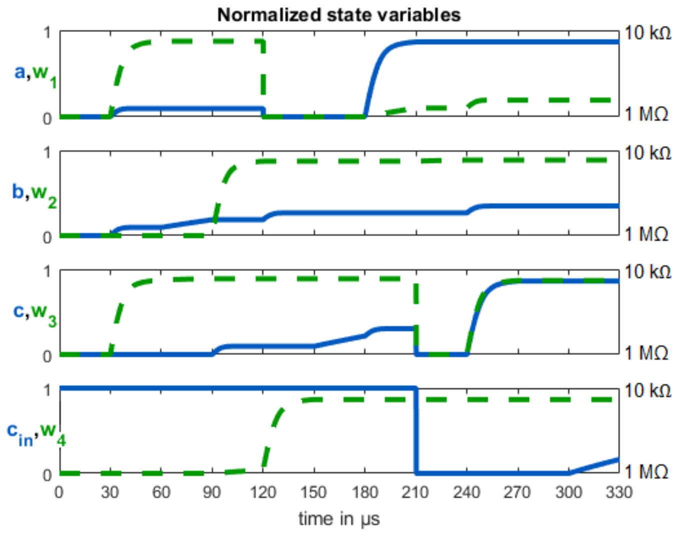


Fig. 6. Simulation of 1-bit full adder with $a = 0$, $b = 0$, $c_{in} = 1$. One IMPLY step takes 30μs.

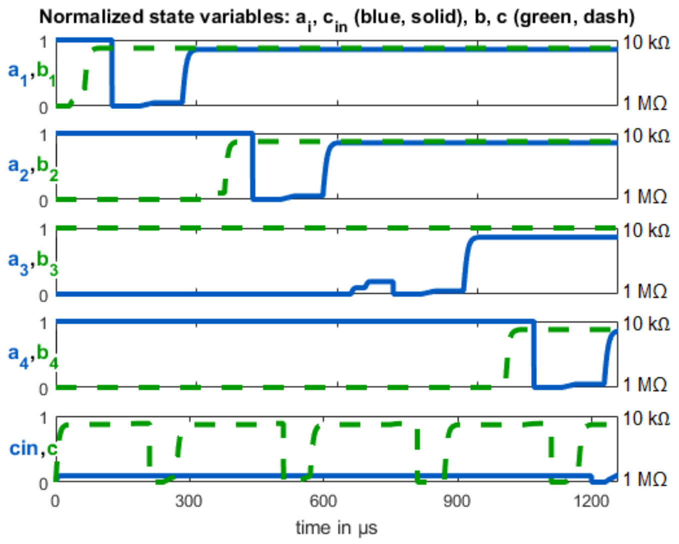


Fig. 7. Simulation of 4-bit full adder with $a_{4-1} = 1011$, $b_{4-1} = 0100$, $c = 0 = \overline{c_{in}}$. Results are available in a_i and c_{in} at 1.26ms.

(c, c_{in}, w_{1-4}) to connect them to different sections. However, such memristors are usually implemented in a Back End Of Line (BEOL) process[2] which leads to CMOS switches being located underneath the crossbar [50]–[53]. This can lead to —virtually— no die area overhead. The area overhead of these externally provided switches can be estimated using our proposed FoMs in Equation (6) & Equation (7) (cf. Section IV) and is considered in the comparison in Section V. We note that prior research [54]–[57] has shown that 1T1R structure is preferable over a 1R structure, because of the robustness of logic operation and reduced parasitic effects, such as the sneak path problem. Using a 1T1R structure, the additional switches could be implemented easier and would lead to relatively less CMOS overhead.

2) CMOS Control Logic: Here, to the best of our knowledge, for the first time in the literature of IMPLY-based adders, we have synthesized the control logic of the adder and report its properties. To this end, we wrote the Hardware Description Language (HDL) code for a state-machine corresponding to the algorithm given in Table I. This state-machine is capable of controlling an n-bit semi-serial adder for any given value of n, with no or minimal changes in the hardware, as it takes the number of bits of the adder, n, as an input variable. We have synthesized it in a 65-nm CMOS technology and the design led to $378\mu m^2$ ($21\mu m \times 18\mu m$) of die area and a power consumption of $271nW$. Adding the energy consumption of the control logic to the energy consumed in the memristors, as reported in Section III-B.2, we obtain an overall energy consumption of $(9.95n + 1.33)$ nano Joules, where n denotes the number of bits.

IV. Metrics

In the literature, arithmetic units such as adders or multipliers are usually compared regarding their size, i.e., the number of necessary memristors, and their speed, i.e., the number of necessary computational steps till the result is ready. The comparison and judgment of performance on these two figures, stand-alone and separately, barely provides a good and fair insight to their merit. Firstly, because not each factor has always the same weight. Moreover, other factors need to be taken into account, e.g., the number of switches and drivers (complexity of the system). In addition, these factors cannot be assumed to have an equally large impact on the performance and merit of a design. This is mainly because of two reasons:

i) Memristors are usually implemented in a BEOL process, which means that CMOS switches can be implemented underneath the memristor crossbar [50]–[53] and thus only affect the size of the chip to an extent which is not necessarily in proportion to the overall CMOS area.
ii) Since memristors have a very small area footprint, the speed of a memristive system can be assumed to have a larger impact on whether it can out-compete traditional CMOS systems, than its size.

Hence, here we propose several FoMs which consider these factors as well as different design constraints.

3) Balanced: We start with an equally balanced FoM, Eq. (3), which is proportional to both the number of memristors (namely n_M) and number of steps (shown by n_S).

$$FoM_B = \frac{1}{n_M \cdot n_S} \tag{3}$$

All proposed FoMs are formulated such that larger figures represent larger merit.

4) Speed-Centered: To evaluate designs for cases where the speed of a design is more important than its size (due to aforementioned reasons and/or design requirements), the FoM in Equation (4) can be used.

$$FoM_S = \frac{1}{n_M \cdot n_S^2} \tag{4}$$

[2]In a BEOL process, the CMOS switches and circuits are produced in a traditional CMOS technology, and once that process is finished the die is transferred to a different facility with a memristive technology which produces the memristors on top of the CMOS-finished die. There are already companies which offer such a service commercially, e.g., see MOSIS C5 CMOS by Neurobit [46], CMOS BEOL Memristor service by KNOWM [47], or CAE-Leti services [48] through Circuits Multi Project (CMP) organization [49].

5) Memristor-Centered: If there is a larger emphasis on the number of memristors in a design, the FoM in Equation (5) can be used.

$$FoM_M = \frac{1}{n_M^2 \cdot n_S} \tag{5}$$

6) CMOS Overhead: To include the impact of necessary additional CMOS switches, the FoM in Equation (6) can be used, where n_C denotes the number of CMOS switches. The constant 1 is added to n_C to ensure the FoM will not be infinite in cases where no CMOS switches are used.

$$FoM_C = \frac{1}{n_M \cdot n_S \cdot (1 + n_C)} \tag{6}$$

7) Area-Centered: The area of the overall memristive system (including both memritors and classical CMOS circuitry) is often assessed by the number of memristors and the number of CMOS switches. However, we need to consider that

i) memristors are a lot smaller than CMOS switches, and
ii) memristors are usually implemented in a BEOL process. Therefore, the CMOS circuitry is usually located underneath the memristor crossbar [50]–[53].

We therefore propose the FoM in Equation (7), which considers that a CMOS switch is by a factor of c larger than a memristor and may be located underneath the crossbar.

$$FoM_A = \frac{1}{n_S \cdot \max(n_M,\, c \cdot n_C)} \tag{7}$$

In Section V, we provide a comprehensive comparative analysis using the FoMs proposed here. We note that not all these FoMs are equally important. For example, between the Area-centered (FoM_A), CMOS Overhead (FoM_C), and Balanced (FoM_B), the Area-centered metric is the most important metric since it provides the most realistic figure regarding the actual area used by the system and consequently its die area and cost. Moreover, they all assume an equal weight on the importance of area and speed, which may not be always the case. We clarify this matter by a few examples.

For instance, if the IMC is intended to enhance an off-chip (large) storage memory, cost of the memory and consequently compactness of the design is more important. In such a case, speed might be a secondary concern since the competing solution (transferring data to the processor, processing it, and storing it back in the memory again) is significantly slower and hence any gain in speed is appreciated. On the other hand, if the IMC is intended to enhance an on-chip (cache) memory, both area (cost) and speed might be important, depending on the type of the processor. In lower-end processors, cost needs to be kept low, but speed is important too since by slower designs it may be faster to process the data in the processor as opposed to in memory. However, for higher-end high-performance processors usually cost is less of a concern and speed has a much higher priority and importance. For such a design Speed-centered (FoM_S) is probably a more representative FoM.

Lastly, we would like to remark that the FoMs we have provided here are not conclusive and there are many other ways to assess the merit of a design. In particular, when a certain design and its constraints are being considered, a designer can and should try to extract a FoM which appropriately represents the goals of their design. We believe the FoMs that we have provided here are good representatives for generic cases, such as the examples above, and hope that they can inspire designers and provide engineers a guideline to come up with their own case-specific FoMs.

V. Comparative Analysis

In this section, we present a comprehensive comparative analysis of IMPLY-based adder designs in the literature regarding their performance using FoMs presented in the previous sections and measured by the number of memristors, number of computational steps and number of switches necessary for the design. We note, that we do not compare the designs regarding power consumption, since, as stated in [34] too, power consumption is rarely reported and given the large variation among memristors, a fair comparison requires simulation with the same model and in similar conditions (such as simulation setup). In our comparisons, numbers reported for improvement in each performance figure, P, have been calculated as

$$\text{Imp.}(\%) = \frac{P_{worse} - P_{better}}{P_{worse}} \times 100, \tag{8}$$

where P is the place holder for the parameters compared (e.g., number of memristors, steps, or switches), and P_{worse} is the number associated with the worse design and P_{better} is the better number. E.g., when comparing the number of necessary memristors in two designs, P_{worse} would be the number of memristors for the design which needs more memristors and P_{better} would be the number of memristors for the design with smaller number of required memristors. Given the performance figures considered in this paper, the design with smaller numbers (less memristors, less switches or smaller number of steps) is always better, hence filling the P_{better}.

Serial designs mainly split into two groups regarding their number of memristors. Here, designs that use $2n + 3$ memristors [36], [37] stand against those that use $3n + 3$ memristors [33], [35], which is a remarkable improvement of asymptotically[3] 33%. This improvement mainly comes from reuse of input and work memristors. The fastest serial design [36] needs $22n$ steps. All serial designs share this characteristic that no switches are necessary in them. Compared to parallel designs, serial designs use half of the memristors, a feature that only our semi-serial [34] and semi-parallel [38] design achieve.

Parallel designs [33], [37] need $5n + 16$ or $5n + 18$ steps, which is approximately a quarter of the number of steps needed by serial designs. However, they need $2\times - 4.5\times$ more memristors. The smallest parallel design [37] saves 55% over the parallel design in [33] through a pipeline-type partitioning of the algorithm. The major drawback of all parallel designs is the high demand of switches which is proportional to the bit width of the adder. This is reflected by a comparably low merit as measured by the FoMs in Equation (6) and

[3]For $n \to \infty$.

TABLE V

SUMMARY OF COMPARISONS BETWEEN IMPLY-BASED ADDERS. NUMBER OF SWITCHES ONLY APPLIES TO ADDITIONAL SWITCHES

Designs	Number of Memristors			Number of Steps			Number of Switches		
	Total	$n = 32$	Imp.	Total	$n = 32$	Imp.	Total	$n = 32$	Imp.
Serial [33]	$3n + 3$	99	29%	$29n$	928	65%	0	0	-100%
Serial [35]	$3n + 3$	99	29%	$23n$	736	56%	0	0	-100%
Serial [36]	$2n + 3$	67	-4%	$22n$	704	54%	0	0	-100%
Serial [37]	$2n + 3$	67	-4%	$23n$	736	56%	0	0	-100%
Parallel [33]	$9n$	288	76%	$5n + 18$	178	-45%	$2n$	64	81%
Parallel [37]	$4n + 1$	129	46%	$5n + 16$	176	-45%	n	32	63%
Iterative [39]	$8n$	256	73%	$21n - 3$	669	52%	0	0	-100%
Semi-Parallel [38]	$2n + 3$	67	-4%	$17n$	544	41%	3	3	-75%
Semi-Serial [34]	$2n + 6$	70	-	$10n + 2$	322	-	12	12	-

TABLE VI

COMPARISON OF IMPLY-BASED ADDERS REGARDING THEIR FIGURES OF MERIT. ALL FIGURES ARE CALCULATED BASED ON $n = 32$. BOLD NUMBERS DENOTE THE BEST DESIGN ACCORDING TO EACH FIGURE OF MERIT

Designs	FoM_B		FoM_S		FoM_M		FoM_C		FoM_A [*]	
	#	Imp.	#	Imp.	#	Imp.	#	Imp.	#	Imp.
Serial [33]	10.9μ	308%	11.7n	1075%	109.9n	476%	10.9μ	-219%	10.9μ	197%
Serial [35]	13.7μ	223%	18.6n	639%	138.6n	357%	13.7μ	-302%	13.7μ	136%
Serial [36]	21.2μ	109%	30.1n	358%	316.4n	100%	**21.2μ**	-521%	21.2μ	53%
Serial [37]	20.3μ	119%	27.6n	400%	302.7n	109%	20.3μ	-494%	20.3μ	60%
Parallel [33]	19.5μ	127%	109.6n	26%	67.7n	836%	0.3μ	1037%	11.0μ	195%
Parallel [37]	44.0μ	1%	**250.3n**	-82%	341.4n	86%	1.3μ	156%	22.2μ	46%
Iterative [39]	5.8μ	660%	8.7n	1479%	22.8n	2679%	5.8μ	-71%	5.8μ	454%
Semi-Parallel [38]	27.4μ	62%	50.4n	173%	409.5n	55%	6.9μ	-101%	27.4μ	18%
Semi-Serial [34]	**44.4μ**	-	137.8n	-	**633.8n**	-	3.4μ	-	**32.3μ**	-

[*] $c = 8$ for all calculated FoM_A, cf. Section IV.

Equation (7). This shows, that judging the performance of IMPLY-based systems solely on low numbers of memristors or computational steps falls short in providing a comprehensive image, if the additional CMOS circuitry is not considered. For instance, this is shown by the fact, that the improvement of our semi-serial adder over the adder in [37] accounts to only 1% according to the balanced FoM in Equation (3), whereas considering the necessary additional switches in the parallel design results in an improvement of 46%, as measured by the Area-centered FoM in Equation (7).

Among designs, which cannot be assigned to either serial or parallel group, our semi-serial design [34] uses three more memristors ($2n + 6$ instead of $2n + 3$) than the semi-parallel design [38], while using significantly less computational steps (41% asymptotically). Please note, that the difference in number of memristors is a constant number of 3, which becomes less important with growing n. Compared to conventional parallel and serial designs, the common advantageous feature of these two designs is that they require as many memristors as serial designs (which is significantly less than parallel designs), whereas they are faster than existing serial designs. In particular, the semi-serial design presented here, which is faster than the semi-parallel design, requires only $10n + 2$ steps. This is, asymptotically, twice as many as the number of steps a parallel design needs, however, this is balanced by requiring two times less memristors compared to the parallel design in [37] and four times less memristors compared to the parallel design in [33]. This leads to our design achieving the best FoM_B, FoM_S, and FoM_A among all designs.

Table V provides an overview of the IMPLY-based adders in the literature, where their respective number of necessary memristors, number of steps and number of necessary switches are given. Table VI provides an overview of all proposed FoMs (as defined in Section IV) for all SoA adders we analyzed in this work. As can be seen from the table, our design outperforms all other designs as measured by 3 out of 5 FoMs.

VI. SEMI-SERIAL ADDER-BASED MULTIPLIER

Here, we present our fourth contribution of the paper by proposing a new multiplier design, which is based on our semi-serial adder. In order to form an $n \times n$-multiplier, a $(2n - 1)$-bit semi-serial adder circuit is replicated $\lceil \frac{n}{2} \rceil$ times for $a \times b$, where a and b are binary numbers (inputs) of size n. In the beginning, the i-th adder holds a_{2i} and a_{2i+1} in its work memristors $w_{i,0}$ and $w_{i,2}$, respectively. Each adder holds, in both summand registers[4], (i.e., sections a and b in the semi-serial adder design, cf. Section III) b shifted to the left (one bit), where b is the second input of the product $a \times b$. First, the respective set of partial products is calculated in every adder, i.e., in the i-th adder the partial products $a_{2i} b_j$ and $a_{2i+1} b_j$ are calculated simultaneously, where j is a natural number belonging to $[0, n - 1]$. Therefor, the operation $a_k \times b_j = a_k \wedge b_j$ is performed, which represents the calculation of one partial product. This expression is reformulated to

[4]Since an $n \times n$-bit multiplier is constructed of adders with a summand register width of $2n - 1$ only, in the last adder the Most-Significant Bit (MSB) of the stored factor has to be placed in the Least-Significant Bit (LSB) of the summand register.

TABLE VII

EXECUTION STEPS OF PARTIAL PRODUCT CALCULATION IN THE PROPOSED TOPOLOGY FOR THE EXAMPLE OF A 2 × 2-BIT MULTIPLIER. UPPERCASE LETTERS DENOTE THE BITS OF THE OPERANDS OF THE MULTIPLICATION $A \times B$, LOWERCASE LETTERS DENOTE MEMRISTORS. THE OPERATIONS IN BOLD BLUE ARE PERFORMED ONLY ONCE IN EACH ADDER, AT THE END OF PARTIAL PRODUCT CALCULATION. UNNUMBERED LINES ARE EXECUTED SIMULTANEOUSLY WITH THE PREVIOUS LINE

Steps	Operation Executed in:		Equivalent Logic	
	Section 1	Section 2	Section 1	Section 2
1	$w_1' = w_0 \to w_1$	$w_3' = w_2 \to w_3$	$w_1' = A_0 \to 0 = \overline{A_0}$	$w_3' = A_1 \to 0 = \overline{A_1}$
2	$w_1'' = a_1 \to w_1'$	$w_3'' = b_2 \to= w_3'$	$w_1'' = B_0 \to \overline{A_0}$	$w_3'' = B_0 \to \overline{A_1}$
3	$a_0' = w_1'' \to a_0$	$b_1' = w_3'' \to= b_1$	$a_0' = (B_0 \to \overline{A_0}) \to 0 = B_0 A_0$	$b_1' = (B_0 \to \overline{A_1}) \to 0 = B_0 A_1$
4	$w_1 = 0$	$w_3 = 0$	FALSE(w_1)	FALSE(w_3)
5	$w_1' = w_0 \to w_1$	$w_3' = w_2 \to w_3$	$w_1' = A_0 \to 0 = \overline{A_0}$	$w_3' = A_1 \to 0 = \overline{A_1}$
	$a_1 = 0$	$b_2 = 0$	FALSE(a_1)	FALSE(b_2)
6	$w_1'' = a_2 \to w_1'$	$w_3'' = b_0 \to w_3'$	$w_1'' = B_1 \to \overline{A_0}$	$w_3'' = B_1 \to \overline{A_1}$
7	$a_1' = w_1'' \to a_1$	$b_2' = w_3'' \to b_2$	$a_1' = (B_1 \to \overline{A_0}) \to 0 = B_1 A_0$	$b_1' = (B_1 \to \overline{A_1}) \to 0 = B_1 A_1$
8	$\mathbf{a_2 = 0}$	$\mathbf{b_0 = 0}$	**FALSE($\mathbf{a_2}$)**	**FALSE($\mathbf{b_0}$)**

Equation (9) and implemented step-wise in every adder.

$$a_k b_j = (a_k \to (b_j \to 0)) \to 0 \tag{9}$$

This logic operation requires three IMPLY steps (and two FALSE steps in the general case). After calculating the partial products, the adders calculate the overall product by calculating the sum of all partial products steps by step. This means, that first the sets of partial products inside every adder are summed up (every adder runs a semi-serial algorithm), then two by two adders are connected to calculate the intermediate sums until all sets of partial products are summed up. This overall sum represents the output or the calculated product of $a \times b$. The result of multiplication and its carry out will be located in $c_{in}\&a$ memristors of the very first adder of the system (c_{in} as the most significant bit or carry-out, and a as the rest of the output value). A 2 × 2-bit example of the algorithm that is used to calculate the partial products can be found in Table VII.

Once partial products are calculated, the adders continue in their usual semi-serial adding algorithm. Figure 8 shows an example of a 4×4-bit multiplier, consisting of two semi-serial adders and one additional switch to connect the adders during summation phase. Note that apart from the additional switch, no changes to the semi-serial adder circuit are necessary. In other words, the adder is only duplicated $\lceil \frac{n}{2} \rceil$ times for an $n \times n$-bit multiplier. The number of necessary additional switches therein is $\lfloor \frac{n-1}{2} \rfloor$.

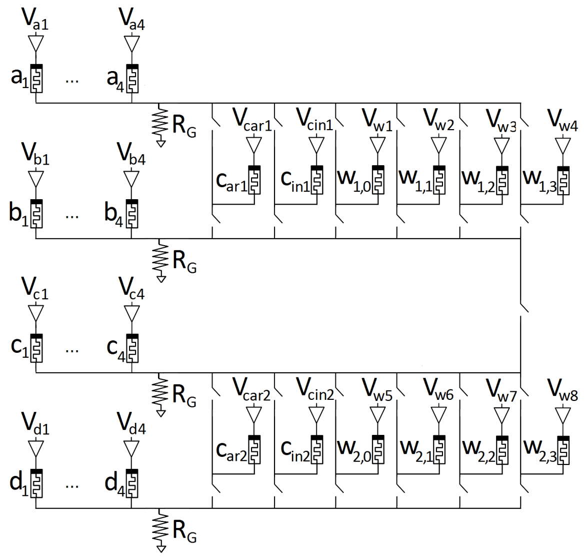


Fig. 8. 4×4-bit multiplier consisting of two semi-serial adders.

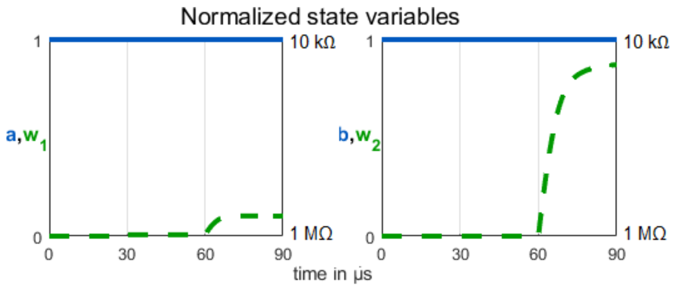


Fig. 9. IMPLY realization of $w_2 = a \wedge b = 1 \wedge 1 = 1$, shown in Equation (9). One IMPLY step takes 30μs.

A. Simulations

The simulation of our multiplier design was carried out on two levels of abstraction. The behavioral correctness of the multiplier algorithm was examined using a Matlab simulation, assuming ideal memristor behavior. Since the multiplier algorithm breaks down into the calculation of partial products and the subsequent addition of those products, we verified the correct calculation of partial products on circuit level by simulating it in SPICE using the VTEAM model, using the same setup mentioned in Section III-B. The subsequent addition is carried out using our semi-serial adder algorithm, which was verified in Section III. Figure 9 shows the LTSpice simulation of the calculation of one partial product $a \wedge b$, where $a = 1$ and $b = 1$. Here the work memristors w_1 and w_2 are assumed to be already initialized to HRS (logic '0'). The result of $a \wedge b$ is stored in w_2. Figure 10 shows the correct calculation of $w_2 = a \wedge b$ for all input combinations of a and b. An IMPLY step takes $30\mu s$ in both simulations. As in Section III-B, a SPICE implementation of the VTEAM model was used for the simulation. The same parameters where used as described above.

Figure 11 shows the simulation of a 2 × 2-bit multiplier in our proposed topology. This simulation was done in Matlab

TABLE VIII
COMPARISON OF IMPLY-BASED MULTIPLIERS FOR $n = 32$

Designs	Number of Memristors			Number of Steps			Number of Switches		
	Total	$n = 32$	Imp.	Total	$n = 32$	Imp.	Total	$n = 32$	Imp.
Shift&Add [40]	$7n + 1$	225	-89%	$2n^2 + 21n$	2720	36%	$8n - 1$	255	19%
Array [41]	$7n^2 - 8n + 9$	6921	70%	$24n - 35$	733	-58%	$8n^2 - 8n + 9$	7945	97%
Dadda [42]	NA*	NA*	NA*	NA*	NA*	NA*	NA*	NA*	NA*
Proposed	$2n^2 + n + 2$	2082	-	$\lceil log_2\ n \rceil(10n + 2) + 4n + 2$	1740	-	$12\lceil \frac{n}{2} \rceil + \lfloor \frac{n-1}{2} \rfloor$	207	-

* Source provides numbers only for an 8-bit Dadda multiplier, see Table IX for a comparison based on $n = 8$.

TABLE IX
COMPARISON OF IMPLY-BASED MULTIPLIERS FOR $n = 8$

Designs	Number of Memristors			Number of Steps			Number of Switches		
	Total	$n = 8$	Imp.	Total	$n = 8$	Imp.	Total	$n = 8$	Imp.
Shift&Add [40]	$7n + 1$	57	-57%	$2n^2 + 21n$	296	5%	$8n - 1$	63	19%
Array [41]	$7n^2 - 8n + 9$	393	65%	$24n - 35$	157	-44%	$8n^2 - 8n + 9$	457	89%
Dadda [42]	NA*	385	64%	NA*	106	-62%	NA*	482	89%
Proposed	$2n^2 + n + 2$	138	-	$\lceil log_2\ n \rceil(10n + 2) + 4n + 2$	280	-	$12\lceil \frac{n}{2} \rceil + \lfloor \frac{n-1}{2} \rfloor$	51	-

* Source does not provide expressions for the number of memristors, steps and switches.

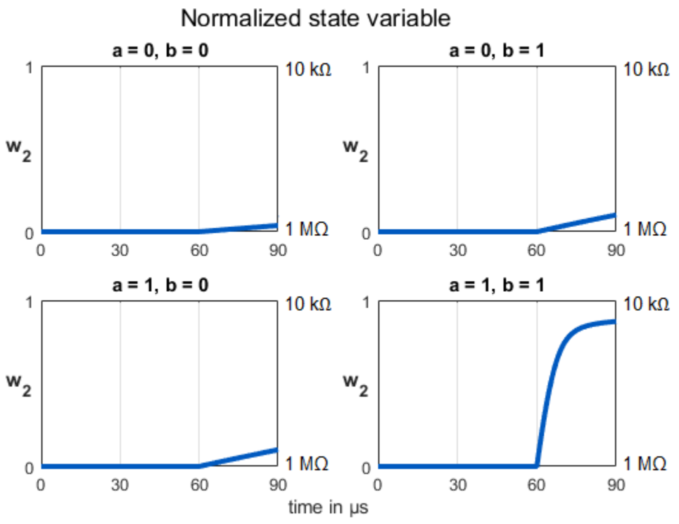


Fig. 10. IMPLY realizations of $w_2 = a \wedge b$, see Equation (9), for all input combinations. One IMPLY step takes 30μs.

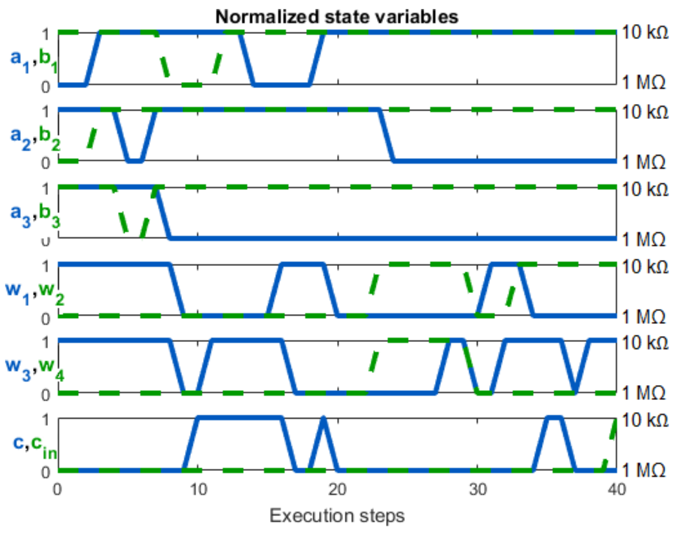


Fig. 11. Simulation of a 2×2-bit multiplier with $a = 11$, $b = 11$ and $c_{in} = 0$. The calculated product is located in $c_{in}\&a_{3-1} = 1001$ after 40 execution steps.

on a functional level, i.e., the memristors are assumed to have ideal behavior. As can be seen in Figure 11, the result of the multiplication $a \times b$ is available after 40 steps, where inputs are $a = 11$ and $b = 11$. The correct result can be found in $c_{in}\&a_{3-1} = 1001$.

B. Comparison and Discussion

Table VIII provides a summary of the features and performance figures (number of memristors, computational steps and switches) of our proposed IMPLY-based multiplier design and others in the literature for a 32-bit multiplier. Since [42] does not provide any expressions for the number of necessary memristors, steps and switches, we could not calculate them for $n = 32$. Hence, we used the the respective numbers which they provide for an 8-bit multiplier and created a second table, namely Table IX, which provides performance figures for all designs based on $n = 8$. Table X and Table XI provide an overview of the FoMs for each of the multiplier designs studied here. In Table X the FoMs are calculated based on $n = 32$, whereas in Table XI the FoMs are calculated based on $n = 8$, since, as mentioned before, the source for the Dadda-multiplier [42] reports performance and feature numbers only for an 8-bit multiplier. The best design according to each FoM is boldfaced in both tables to facilitate its identification. The improvements were calculated using Equation (8). As can be seen in Table X, our multiplier design manages to improve over the array- and Dadda-type multipliers up to a $50\times$, thanks to significant savings in the number of memristors and switches. A 32-bit implementation of Shift&Add multiplier [40] outperforms all other multiplier designs as measured by 4 of 5 FoMs, whereas its 8-bit implementation outperforms others in 3 out 5 (due to Dadda winning the best FoM_S in the 8-bit implementation). This is caused by a few factors;

TABLE X

COMPARISON OF IMPLY-BASED MULTIPLIERS REGARDING THEIR FIGURES OF MERIT. ALL FIGURES ARE CALCULATED BASED ON $n = 32$. BOLD NUMBERS DENOTE THE BEST DESIGN ACCORDING TO EACH FIGURE OF MERIT

Designs	FoM_B		FoM_S		FoM_M		FoM_C		$FoM_A{}^*$	
	#	Imp.	#	Imp.	#	Imp.	#	Imp.	#	Imp.
Shift&Add [40]	**1643.0n**	-492%	**600.7p**	-279%	**7262.2p**	-5377%	**6382.8p**	-381%	180.2n	532%
Array [41]	197.1n	40%	268.9p	-70%	28.5p	366%	24.8p	5250%	21.5n	1186%
Dadda [42]+	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
Proposed	276.0n	-	158.6p	-	132.6p	-	1327.1p	-	**276.0n**	-

* $c = 8$ for all calculated FoM_A, cf. Section IV.

+ Source provides numbers only for an 8-bit Dadda multiplier, hence, in this case the FoMs for a 32-bit implementation cannot be calculated. See Table XI for a comparison of FoMs based on $n = 8$.

TABLE XI

COMPARISON OF IMPLY-BASED MULTIPLIERS REGARDING THEIR FIGURES OF MERIT. ALL FIGURES ARE CALCULATED BASED ON $n = 8$. BOLD NUMBERS DENOTE THE BEST DESIGN ACCORDING TO EACH FIGURE OF MERIT

Designs	FoM_B		FoM_S		FoM_M		FoM_C		$FoM_A{}^*$	
	#	Imp.	#	Imp.	#	Imp.	#	Imp.	#	Imp.
Shift&Add [40]	**59.3μ**	-129%	200.2n	-117%	**1039.8n**	-454%	**926.1n**	-86%	6.7μ	31%
Array [41]	16.2μ	60%	103.2n	-12%	41.2n	355%	35.4n	1306%	1.7μ	402%
Dadda [42]	24.5μ	6%	**231.2n**	-150%	63.6n	195%	50.7n	881%	2.4μ	258%
Proposed	25.9μ	-	92.4n	-	187.5n	-	497.7n	-	**8.8μ**	-

* $c = 8$ for all calculated FoM_A, cf. Section IV.

i) the underlying efficiency of Shift&Add multipliers,
ii) the high degree of optimization that is inherent to the building blocks of the design, such as multiplexers and shift registers, and
iii) extensive usage of external CMOS circuitry such as multiplexers and shift registers.

In particular, the last factor makes the comparison with our proposed multiplier difficult, since our design relies significantly less on external CMOS circuitry. Moreover, we have not, and cannot, consider the CMOS circuits which are necessary to implement the state machine controlling the Shift&Add, Array, and Dadda multiplier. This could further change the balance to our advantage, in particular, regarding FoM_A. We note that our proposed multiplier, even when the aforementioned factors are not considered, outperforms all other designs in both 8-bit and 32-bit implementations in terms of FoM_A. In particular, compared to the Shift&Add multiplier [40], which is the best design according to the majority of FoMs, our design is better in terms of FoM_A by a large margin of 532%. In other words, our proposed design is more than 5× more area-efficient than the Shift&Add multiplier [40].

We contend that the Area-centered FoM from Equation (7) (cf. Section IV) provides the most realistic estimation of merit when it comes to die area, because not only it takes the CMOS circuitry into account, but also it considers the fact that supplementary CMOS circuitry is lying underneath the memristor crossbar thanks to implementation of memristors in BEOL, which is the most common practice [50]–[53]. Besides that, our multiplier is designed by simply duplicating our semi-serial adder without adding any additional building blocks, which helps to keep complexity of CMOS circuitry low.

We note that even though our design is 36% faster than Shift&Add multiplier [40], and uses 19% less switches, but needing 89% more memristors leads to Shift&Add multiplier [40] having a better FoM in 4 out of 5 FoMs. Additional improvements in latency can be achieved by pipelining consecutive multiplications, as suggested in [58]. For instance, the latency of the Shift&Add multiplier can be reduced to $24n$ [58], which is a significant improvement over the $2n^2+21n$. However, pipelining is outside the scope of this work and a future plan. Even though it is plausible to assume that pipeling improves the performance of our multiplier too, since we have not pipelined our design, we cannot compare the latency of pipelined designs. Therefore, here we report the numbers for designs without pipelining them.

Another example on the lack of respresentativeness of one parameter in the merit of a design is comparison of our design with Array Multiplier [41]. Our design is 58% slower than Array Multiplier [41], however, needing 70% less memristors leads to our design outperforming that design in 4 out of 5 FoMs. These two examples (comparison with Shift&Add and Array multipliers) show us that being better or worse in one aspect does not present a holistic view on the merit of a design. Therefore, a designer needs to choose a FoM, which represents the design constraints the best and assess any design with appropriate FoM and decide on design decisions which help the design in meeting the requirements.

VII. CONCLUSIONS

In this paper, we extended the evaluation of our semi-serial IMPLY-based memristive full-adder design, which outperforms other designs in the literature, as measured by 3 out of 5 Figures of Merit (FoMs). To enhance the comparability of memristive systems regarding their performance, we proposed four new FoMs. A designer can choose the best suiting FoM with respect to the design goal (smallest or fastest possible design or a balanced design). Moreover, inspired by the proposed FoMs they can come up with a new FoM which better represents the specifics of their design constraints and trade-offs. We also reviewed IMPLY-based multiplier designs in the literature and proposed a new multiplier design that is based on our semi-serial full-adder. This multiplier design manages to perform well compared to the designs in the literature, especially if the area overhead caused by additional necessary CMOS switches is considered. To further improve comparability and quality of performance, calculations of power consumption (based on memristor models that are calibrated with experimental data and verified with practical implementations) and die area (as provided in this paper, by simulation and/or implementation of the necessary CMOS circuitry for switches and control logic) should be carried out for future works.

ACKNOWLEDGMENT

Access to CAD tools and technology was facilitated by CMC Microsystems.

REFERENCES

[1] D. Niu, Y. Chen, and Y. Xie, "Low-power dual-element memristor based memory design," in *Proc. 16th ACM/IEEE Int. Symp. Low Power Electron. Design (ISLPED)*, New York, NY, USA, Aug. 2010, pp. 25–30.

[2] Y. Ho, G. M. Huang, and P. Li, "Dynamical properties and design analysis for nonvolatile memristor memories," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 58, no. 4, pp. 724–736, Apr. 2011.

[3] B. Mohammad, D. Homouz, and H. Elgabra, "Robust hybrid memristor-CMOS memory: Modeling and design," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 21, no. 11, pp. 2069–2079, Nov. 2013.

[4] V. S. Baghel and S. Akashe, "Low power memristor based 7T SRAM using MTCMOS technique," in *Proc. 5th Int. Conf. Adv. Comput. Commun. Technol.*, Feb. 2015, pp. 222–226.

[5] H. Kim, M. P. Sah, C. Yang, and L. O. Chua, "Memristor-based multilevel memory," in *Proc. 12th Int. Workshop Cellular Nanosc. Netw. Appl. (CNNA)*, Feb. 2010, pp. 1–6.

[6] M. Zangeneh and A. Joshi, "Design and optimization of nonvolatile multibit 1T1R resistive RAM," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 8, pp. 1815–1828, Aug. 2014.

[7] N. TaheriNejad, P. D. S. Manoj, and A. Jantsch, "Memristors' potential for multi-bit storage and pattern learning," in *Proc. IEEE Eur. Modelling Symp. (EMS)*, Oct. 2015, pp. 450–455.

[8] N. TaheriNejad, M. P. D. Sai, M. Rathmair, and A. Jantsch, "Fully digital write-in scheme for multi-bit memristive storage," in *Proc. 13th Int. Conf. Elect. Eng., Comput. Sci. Autom. Control (CCE)*, Sep. 2016, pp. 1–6.

[9] D. Radakovits and N. TaheriNejad, "Implementation and characterization of a memristive memory system," in *Proc. IEEE Can. Conf. Elect. Comput. Eng. (CCECE)*, May 2019, pp. 1–4.

[10] G. Hilson. (2015). *IMEC, Panasonic Push Progress on ReRAM, EETimes*. [Online]. Available: https://www.eetimes.com/document.asp?doc_id=1327307

[11] ReRAM. (2017). *Crossbar*. [Online]. Available: https://www.crossbar-inc.com/en/

[12] B. Govoreanu *et al.*, "10×10 nm^2 Hf/HfO$_x$ crossbar resistive RAM with excellent performance, reliability and low-energy operation," in *Proc. Int. Electron Devices Meeting*, Dec. 2011, pp. 31.6.1–31.6.4.

[13] A. Sinha, M. S. Kulkarni, and C. Teuscher, "Evolving nanoscale associative memories with memristors," in *Proc. 11th IEEE Int. Conf. Nanotechnol.*, Aug. 2011, pp. 860–864.

[14] R. S. Williams. (2010). *Finding the Missing Memristor*. Accessed: Nov. 6, 2017 [Online]. Available: https://www.youtube.com/watch?v=bKGhvKyjgLY

[15] S. Hamdioui *et al.*, "Memristor for computing: Myth or reality?" in *Proc. Conf. Design, Autom. Test Eur. (DATE)*. Leuven, Belgium: European Design and Automation Association, 2017, pp. 722–731.

[16] M. A. Zidan, J. P. Strachan, and W. D. Lu, "The future of electronics based on memristive systems," *Nature Electron.*, vol. 1, no. 1, pp. 22–29, Jan. 2018.

[17] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, "Pinatubo: A processing-in-memory architecture for bulk bitwise operations in emerging non-volatile memories," in *Proc. ACM 53rd Annu. Design Autom. Conf. (DAC)*, New York, NY, USA, 2016, pp. 173-1–173-6, doi: 10.1145/2897937.2898064.

[18] R. B. Hur and S. Kvatinsky, "Memory processing unit for in-memory processing," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit. (NANOARCH)*, Jul. 2016, pp. 171–172.

[19] P. Gaillardon *et al.*, "The programmable logic-in-memory (PLiM) computer," in *Proc. Design, Autom. Test Eur. Conf. Exhibit. (DATE)*, Mar. 2016, pp. 427–432.

[20] G. Papandroulidakis, I. Vourkas, A. Abusleme, G. C. Sirakoulis, and A. Rubio, "Crossbar-based memristive logic-in-memory architecture," *IEEE Trans. Nanotechnol.*, vol. 16, no. 3, pp. 491–501, May 2017.

[21] S. Gupta, M. Imani, and T. Rosing, "FELIX: Fast and energy-efficient logic in memory," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2018, pp. 1–7.

[22] L. Gao, F. Alibart, and D. B. Strukov, "Programmable CMOS/memristor threshold logic," *IEEE Trans. Nanotechnol.*, vol. 12, no. 2, pp. 115–119, Mar. 2013.

[23] E. Linn, R. Rosezin, C. Kügeler, and R. Waser, "Complementary resistive switches for passive nanocrossbar memories," *Nature Mater.*, vol. 9, no. 5, pp. 403–406, May 2010.

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

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

[26] I. Vourkas and G. C. Sirakoulis, *Memristor-Based Nanoelectronic Computing Circuits and Architectures*, vol. 19. Cham, Switzerland: Springer, 2016.

[27] A. Siemon, S. Menzel, R. Waser, and E. Linn, "A complementary resistive switch-based crossbar array adder," *IEEE J. Emerg. Sel. Topics Circuits Syst.*, vol. 5, no. 1, pp. 64–74, Mar. 2015.

[28] N. Talati, S. Gupta, P. Mane, and S. Kvatinsky, "Logic design within memristive memories using memristor-aided logic (MAGIC)," *IEEE Trans. Nanotechnol.*, vol. 15, no. 4, pp. 635–650, Jul. 2016.

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

[30] G. Snider, "Computing with hysteretic resistor crossbars," *Appl. Phys. A, Solids Surf.*, vol. 80, no. 6, pp. 1165–1172, Mar. 2005.

[31] E. Lehtonen and M. Laiho, "Stateful implication logic with memristors," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit.*, Jul. 2009, pp. 33–36.

[32] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, and R. S. Williams, "'Memristive' switches enable 'stateful' logic operations via material implication," *Nature*, vol. 464, no. 7290, p. 873, 2010.

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

[34] N. TaheriNejad, T. Delaroche, D. Radakovits, and S. Mirabbasi, "A semi-serial topology for compact and fast IMPLY-based memristive full adders," in *Proc. IEEE New Circuits Syst. Symp. (NewCAS)*, Jun. 2019, pp. 1–5.

[35] M. Teimoory, A. Amirsoleimani, J. Shamsi, A. Ahmadi, S. Alirezaee, and M. Ahmadi, "Optimized implementation of memristor-based full adder by material implication logic," in *Proc. 21st IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Dec. 2014, pp. 562–565.

[36] S. G. Rohani and N. TaheriNejad, "An improved algorithm for IMPLY logic based memristive full-adder," in *Proc. IEEE 30th Can. Conf. Elect. Comput. Eng. (CCECE)*, Apr. 2017, pp. 1–4.

[37] A. Karimi and A. Rezai, "Novel design for a memristor-based full adder using a new IMPLY logic approach," *J. Comput. Electron.*, vol. 17, no. 3, pp. 1303–1314, Sep. 2018.

[38] S. G. Rohani, N. Taherinejad, and D. Radakovits, "A semiparallel full-adder in IMPLY logic," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 1, pp. 297–301, Jan. 2020.

[39] K. C. Rahman, M. R. Khan, and M. A. Perkowski, "Memristor based 8-bit iterative full adder with space-time notation and sneak-path protection," in *Proc. IEEE 60th Int. Midwest Symp. Circuits Syst. (MWSCAS)*, Aug. 2017, pp. 695–698.

[40] L. Guckert and E. E. Swartzlander, "Optimized memristor-based multipliers," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 2, pp. 373–385, Feb. 2017.

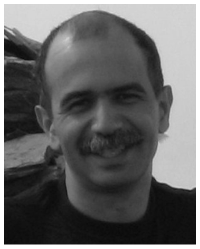
[41] L. E. Guckert *et al.*, "Memristor-based arithmetic units," Ph.D. dissertation, Univ. Texas Austin, Austin, TX, USA, 2016. [Online]. Available: http://hdl.handle.net/2152/46491
[42] L. Guckert and E. E. Swartzlander, "Dadda multiplier designs using memristors," in *Proc. IEEE Int. Conf. IC Design Technol. (ICICDT)*, May 2017, pp. 1–4.
[43] L. Dadda, "Some schemes for parallel multipliers," *Alta Frequenza*, vol. 34, pp. 349–356, 1965.
[44] S. Kvatinsky, M. Ramadan, E. G. Friedman, and A. Kolodny, "VTEAM: A general model for voltage-controlled memristors," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 62, no. 8, pp. 786–790, Aug. 2015.
[45] M. Jungwirth, D. Radakovits, and N. TaheriNejad. (Mar. 2018). *SPICE Implementation of VTEAM Model*. [Online]. Available: https://www.ict.tuwien.ac.at/staff/taherinejad/projects/memristor/files/VTEAM.sub
[46] Neuro-Bit. (2017). *Bio Inspired Technologies LLC*. [Online]. Available: http://www.bioinspired.net/home.html
[47] Knowm. (2019). *Knowm Inc*. [Online]. Available: https://knowm.org
[48] P. Clarke. (Dec. 2018). *Leti 300 mm Fab Extended for PCM, ReRAM, Quantum Computing*. [Online]. Available: https://www.eenewsanalog.com/news/leti-300mm-fab-extended-pcm-reram-quantum-computing
[49] P. Clarke. (Aug. 2018). *Leti, CMP Offer OxRAM Multiproject Wafer Service*. [Online]. Available: https://www.eenewsanalog.com/news/leti-cmp-offer-oxram-multiproject-wafer-service
[50] S. Bhat, S. Kulkami, J. Shi, M. Li, and C. A. Moritz, "Skynet: Memristor-based 3D IC for artificial neural networks," in *Proc. IEEE/ACM Int. Symp. Nanosc. Archit. (NANOARCH)*, Jul. 2017, pp. 109–114.
[51] H. Manem, K. Beckmann, M. Xu, R. Carroll, R. Geer, and N. C. Cady, "An extendable multi-purpose 3D neuromorphic fabric using nanoscale memristors," in *Proc. IEEE Symp. Comput. Intell. Secur. Defense Appl. (CISDA)*, May 2015, pp. 1–8.
[52] C. Li *et al.*, "Three-dimensional crossbar arrays of self-rectifying Si/SiO$_2$/Si memristors," *Nature Commun.*, vol. 8, p. 15666, Nov. 2017.
[53] M. Hu *et al.*, "Memristor-based analog computation and neural network classification with a dot product engine," *Adv. Mater.*, vol. 30, no. 9, Mar. 2018, Art. no. 1705914.
[54] H. Wan *et al.*, "*In situ* observation of compliance-current overshoot and its effect on resistive switching," *IEEE Electron Device Lett.*, vol. 31, no. 3, pp. 246–248, Mar. 2010.
[55] C. Li *et al.*, "Analogue signal and image processing with large memristor crossbars," *Nature Electron.*, vol. 1, no. 1, pp. 52–59, Jan. 2018.
[56] C. Li *et al.*, "In-memory computing with memristor arrays," in *Proc. IEEE Int. Memory Workshop (IMW)*, May 2018, pp. 1–4.
[57] N. TaheriNejad and D. Radakovits, "From behavioral design of memristive circuits and systems to physical implementations," *IEEE Circuits Syst. Mag.*, vol. 19, no. 4, pp. 6–18, 2019.
[58] L. Guckert and E. E. Swartzlander, Jr., *System Design With Memristor Technologies*. Edison, NJ, USA: IET, 2018.

David Radakovits received the B.Sc. degree in electrical engineering and information technology from TU Wien, Vienna, Austria, where he is currently a Graduate Student. His B.Sc. thesis was on Binary Storage on Memristors. He continues to work on memristive circuits and systems for his M.Sc. degree in embedded systems. His main research interests are memristive circuits and systems, embedded systems, systems-on-chip, and hardware security. He has published five articles at IEEE journals and conferences on different aspects of memristive circuits and systems.

Nima TaheriNejad (Member, IEEE) received the Ph.D. degree in electrical and computer engineering from The University of British Columbia, Vancouver, Canada, in 2015. He is currently a Universität Assistant with the TU Wien (formerly known as the Vienna University of Technology), Vienna, Austria, where his areas of work include self-awareness in resource-constrained cyber-physical systems, embedded systems, systems on chip, memristor-based circuit and systems, health-care, and robotics. He has published two books and more than 40 peer-reviewed articles. He has also served as a reviewer, an editor, an organizer, and the chair for various journals, conferences, and workshops. In the field of memristive circuits and systems, his focus has been on physical implementations, reliability, memory, logic (particularly IMPLY), and in-memory calculations. He has received several awards and scholarships from universities and conferences he has attended.

Mengye Cai (Student Member, IEEE) received the Ph.D. degree in electrical and computer engineering from The University of British Columbia, Vancouver, Canada, in 2019. From 2013 to 2015, he was a Research Assistant with the Implantable Systems Laboratory, Western University, London, Canada. He is currently a Post-Doctoral Fellow with the Systems-on-Chip (SOC) Laboratory, The University of British Columbia. His research interests include analog and mixed-signal integrated circuits design, implantable or wearable biomedical devices/sensors, RF monitoring and diagnosing, low-power radio system, in-memory computations, CMOS image sensor, and wireless power transmission/management.

Théophile Delaroche, photograph and biography not available at the time of publication.



Shahriar Mirabbasi (Member, IEEE) received the B.Sc. degree in electrical engineering from the Sharif University of Technology, Tehran, Iran, in 1990, and the M.A.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Toronto, Toronto, ON, Canada, in 1997 and 2002, respectively. Since August 2002, he has been with the Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC, Canada, where he is currently a Professor. His current research interests include analog, mixed-signal, RF, and mm-wave integrated circuit and system design, with an emphasis on communication, sensor interface, and biomedical applications.